Solutions and Examples for Apache Administrators

第2版
Apache2.0、2.2対応

Apache クックブック

Webサーバ管理者のためのレシピ集

Ken Coar
Rich Bowen
著

笹井 崇司 訳

O'REILLY®
オライリー・ジャパン

Apacheクックブック
第2版
Webサーバ管理者のためのレシピ集

Ken Coar
Rich Bowen 著
笹井 崇司 訳

O'REILLY®
オライリー・ジャパン

SECOND EDITION

Apache Cookbook

Ken Coar, Rich Bowen

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

まえがき

Apache Webサーバは非常にすばらしいソフトウェアだ。Apache Software Foundationが配布している基本パッケージは、非常に完成度が高く、とても強力である。しかも、ソフトウェアが肥大化しないよう、多大な労力が払われている。特にすばらしいのは、拡張性を考慮して設計されている点だ。もし期待していた機能が「箱から取り出したまま」のApacheパッケージになくても、たいていの場合は、その機能を実現するよう拡張できる。Apache Software Foundationが配布しているパッケージには、たくさんの拡張(「モジュール」と呼ぶ)が含まれている。読者の要求を満たしてくれるモジュールがなくても、Apache Webサーバには何百万人ものユーザがいるので、すでに同じような作業をした人がいるはずだ。そして、要求を満たすようにサーバを変更したり改良したりするレシピを作り出してくれているだろう。

本書は、このようなレシピを集めたものである。Usenetニュースグループ、Apache FAQ、Apache関連のメーリングリスト、"How To"の質問が入ったメール、IRCチャットチャンネルに投稿された質問や課題、ボランティアからの投稿などが情報源になっている。

本書に書かれているものは、すべて実体験に基づいており、筆者自身か筆者に助けを求めてきた人たちが遭遇した問題だ。ソースコードの基本的なコンパイル方法から、SSLによる暗号化が必要なURLの取り扱いといった複雑な問題まで、幅広い話題を取り上げた。

本書には、100を超える課題とその解決策を収めてある。題材は、主にユーザが遭遇する頻度に基づいて選んでおり、「本書の構成」に示したようなテーマごとに、大まかに分類してある。

本書で取り上げたレシピは、主にサーバ全体を管理しているWebマスターにとって役立つものだが、.htaccessファイルを使ってWebディレクトリの動作をカスタマイズしたいユーザにも役立つはずだ。

Apacheクックブックは、実践的な参考書になるよう執筆した。レシピや章をひとつひとつ読んだだけでは、全体を理解できないような理論的な書籍ではない(1つのRFCを読んだだけでは、Roy Fielding †の数多くのRFC全体を理解できないように)。本書では、特定の課題に的を絞ってその解決策を紹介した。課題は、目次や索引で探せるようにしてある。

† Clueというボードゲームの無名の達人であり、無名のHTTP開発者(訳者追記:訳注:Roy Fieldingは、HTTPの主要な開発者であり、Apache Software Foundation共同創設者でもある。)。

本書の内容

本書の題材の多くは、Q&Aの議論やコンサルティングから引用したものであり、できるだけ完全を期すよう心がけた。もちろん、現時点でまだ満足のいく答えが得られていない(少なくとも筆者の知識の及ぶ限りでは)課題も「レシピ」に入っている。これは読者をからかったり、困らせたり、失望させるためではない。こうしたレシピを入れたのは、完全を期すためだ。筆者たちはこうした課題を無視してしまうのではなく、ちゃんと検討したのだということを知ってほしいためである。

永久に解決できない課題というものは、ほとんど存在しない。不完全なレシピに解決策が見つかったら、本書のWebサイトにすぐに掲載し、本書の改訂時には反映させるつもりだ。もし本書で取り上げているが説明していない課題や、本書で全く取り上げていない課題について、何か解決策を見つけた人がいたら、ぜひ本書の調査チームに知らせてほしい。その解決策を Web サイトで紹介し、次の改訂版に載せるようにしたい。

もしかすると、解決策を提供してくれるのは、あなたかもしれない。

対象プラットフォーム

本書のレシピは、2つの主要なプラットフォームである Unix 系システム(Linux や FreeBSD、Solaris など)とWindowsを対象にしている。特定のプラットフォームに依存しないものも多く、その場合にはありがたくOSやハードウェアに関する説明を省略させてもらった。筆者の個人的な好みや経験から、Windowsプラットフォームよりも Unix系システムの方が完成度が高くなっている。Windows特有のレシピに対する投稿や助言、訂正があれば、ありがたく将来の改訂版に反映するつもりだ。

他の書籍

現在、Apache Webサーバとその操作方法について解説した書籍はたくさんある。代表的なものをいくつか挙げておく。

- 『Apache ハンドブック第3版』オライリー・ジャパン発行(原書『Apache: The Definitive Guide, Third Edition, by Ben and Peter Laurie』O'Reilly Media 発行)
- 『Pro Apache, Third Edition』Peter Wainwright 著、Apress 発行、和書未刊
- 『Apache Administrator's Handbook』Rich Bowen 他著、Macmillan 発行、和書未刊

また、次の Web ページは Apache 関連書籍の発行を追跡しているので、ときどきチェックしておくとよいだろう。

- http://httpd.apache.org/info/apache_books.html

他の情報源

書籍からだけではなく、オンラインからもたくさんの情報を入手することができる。Apache Webサーバの

利用と管理を専門に扱う、いろいろなWebサイトやメーリングリスト、Usenetニュースグループがある。こうしたWebサイトは無数にあるが、ここでは活発に役に立つ情報を提供しているところをいくつか挙げておく。

- Usenet ニュースグループの comp.infosystems.www.servers.unix と comp.infosystems.www.servers.ms-windows
  これらのニュースグループは、特にApacheを専門に扱っているわけではないが、Apacheに関連する投稿が多く、熟練のApacheユーザが集まっている。もしニュースにアクセスすることができない場合や、Usenet へアクセスする方法がわからない場合には、http://groups.google.com/ を調べてみるとよい。

- Web サイト Apache Today http://apachetoday.com/
  Internet.com によって運営されており、定期的に Web サーバや Web サーバを最大限に活用する方法に関する記事を掲載している。

- メーリングリスト users@httpd.apache.org
  ここには、Apacheソフトウェアにさまざまな経験を持った人たちがいる。Apache開発者もここで見つけることができる。投稿するには、メーリングリストに参加登録する必要がある。参加登録するには、http://httpd.apache.org/userslist.html を訪れてみよう。

- IRC チャンネル #apache irc.freenode.net ネットワーク
  他にもたくさんのIRCネットワーク上に、#apache IRCチャンネルがある。しかし、筆者に会う機会が一番あるのは freenode ネットワークだろう。

Apache Webサーバには「公式な」サポートメディアというものは存在しないということを指摘しておく必要があるだろう。Apacheは主にボランティアが開発したフリーなソフトウェアであり、実際のところ「公式な」サポート経路というものは存在しない。しかし、ここに挙げた非公式のサポートフォーラムは、いろいろな質問にうまく答えてくれるかもしれない。

本書の構成

本書は、13 の章と 2 つの付録から構成されている。

1章 インストール
標準のApacheソフトウェアの基本的なインストール方法を説明する。Unix系システムにおいてソースコードからインストールする方法、Windows において Apache 開発者によって作られた MSI (Microsoft ソフトウェアインストーラ)パッケージからインストールする方法などを紹介する。

2章 一般的なモジュールの追加
最もよく使われるサードパーティ製モジュールのインストール方法を詳しく説明する。ここでは、一

般的なモジュールのインストール方法についても紹介する。この手順は、それほど複雑なインストール手順を必要としないモジュールにも適用できるだろう。

3章 ログの記録

Webサイトへの訪問をログに記録する方法や、Apacheのエラーログの仕組みに関するレシピを紹介する。

4章 バーチャルホスト

1つのApacheサーバで複数のWebサイトを運用する方法、そのための設定ファイルの書き方について説明する。

5章 エイリアスとリダイレクトとリライト

URLの処理方法、ファイル参照の制御方法、URLを別のURLへ書き換える方法、他のWebサイトを指す方法について説明する。

6章 セキュリティ

インターネット上の不正な侵入や情報漏洩からどうやってApacheサーバを保護するか、基本的な課題を取り上げる。

7章 SSL

SSL対応のブラウザで安全なトランザクションが行えるように、Apache Webサーバを設定する。これは、送金や医療記録のような機密情報を扱う場合には、不可欠になる。

8章 動的コンテンツ

実行時にスクリプトを使ってサーバを強化し、ユーザごとにスクリプトを処理する方法を説明する。

9章 エラー処理

Webサーバのエラーメッセージを、自分の好みにカスタマイズする方法を説明する。

10章 プロキシ

Apacheサーバがユーザと Webページの間のプロキシ(代理)として動作するように設定し、できるだけ透過にシームレスに処理する方法を説明する。

11章 パフォーマンス

パフォーマンスのボトルネックを解消する方法、Apacheサーバ全体の機能を改善する方法など、いろいろなレシピを紹介する。

12章 ディレクトリのリスト表示

ディレクトリのリストをWebページとして表示するモジュールについて、そのカスタマイズ方法を説明する。

13章 その他のトピック

他の章にうまく当てはまらなかった各種トピックを紹介する。

付録 A Apache の正規表現
Apache ディレクティブにおいて、正規表現を使ってパターンマッチングする方法を説明する。

付録 B トラブルシューティング
メッセージの見方や、よくある設定の問題など、基本的なトラブル解決テクニックを紹介する。

本書の表記法

本書は、ある特定の表記法に従って書かれている。表記法に慣れてしまえば、コメント、入力すべきコマンド、与えるべき値などを、簡単に区別できるようになるだろう。場合によっては、本文中の用語やコード例で書体が変わっているところがある。書体(等幅やボールドなど)の違いが何を意味しているかは、後ほど説明する。

プログラムの表記

本書で説明したコード例の多くは、実際のアプリケーションのコードではなく、スクリプトから抜粋したものである。コマンドラインプロンプト(例えば、Unix系システムのxtermや、WindowsのDOSコマンドプロンプト)からコマンドを入力した場合には、次のように書いてある。

```
% find /usr/local -name apachectl -print
# /usr/local/apache/bin/apachectl graceful
C:>cd "\Program Files\Apache Group\Apache\bin"
C:\Program Files\Apache Group\Apache\bin>apache -k stop
```

Unix系システムでは、コマンドプロンプトが#で始まる場合には、スーパーユーザ(ユーザ名がrootのユーザ)としてログインする必要があることを示している。%で始まる場合には、どんなユーザでもそのコマンドを実行できることを示している。

書体の表記

本書では次のような表記法を使っている。

太字(Bold)
グラフィカルインターフェイス上のメニュー項目に使用する。

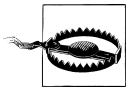
等幅(Constant Width)
関数名、コマンドオプション、コンピュータ出力、環境変数名、文字列リテラル、コード例に使用する。

等幅ボールド(**Constant Width Bold**)
コンピュータ上のダイアログや例において、ユーザが入力するものを示すときに使用する。

等幅イタリック(*Constant Width Italic*)

置き換え可能なパラメータ、ファイルシステム上のパス、変数名に使用する。

このアイコンはヒントやアドバイス、一般的な注意事項を表す。



このアイコンは警告や勧告を表す。

本文の表記

本書は、ある特定の話題について(例えば、Perl プログラミング言語など)ではなく、一般的な話題を扱っているため、詳細については他の情報源を紹介して参照できるようにしてある。最もよく参照しているのは次の 2 つだ。

Unix 系システムのオンラインマニュアル("manpage")

manpage を参照してほしいときには、例えば、「詳しくは、kill(1)の manpage を参照すること」と書いてある。括弧中の番号はマニュアルのセクション番号である。次のコマンドを実行することで、アクセスすることができる。

```
% man  1 kill
```

Apache Web サーバのドキュメント

このドキュメントを参照してほしいときには、例えば、「詳しくはmod_authのドキュメントを参照すること」と書いてある。この場合には、次のような Web ページを指している。

```
http://httpd.apache.org/docs/mod/mod_auth.html
```

実際のモジュールではなく、特定のApacheディレクティブを参照している場合もある。このような場合には、次の Web ページでディレクティブ名を探すと、適切な Web ページを見つけることができる。

```
http://httpd.apache.org/docs/mod/directives.html
```

このページには、標準のApacheパッケージで利用可能なすべてのディレクティブの一覧が載っている。非標準のディレクティブやサードパーティ製モジュールに特有のディレクティブであれば、そのモジュールと同じ場所にドキュメントがあるはずだ。先に挙げたリンクは、バージョン1.3用のドキュメントを指している。バージョン 2.0 用のドキュメントにアクセスするには、URL 中の "docs/" を "docs-2.0/" に書き換えればよい。

サンプルコードの使用

本書の目的は、読者の仕事の手助けをすることである。一般に、本書に掲載しているコードは各自のプログラムやドキュメントに使用して構わない。コードの大部分を転載する場合を除き、私たちに許可を求める必要はない。例えば、本書のコードブロックをいくつか使用するプログラムを作成するために、許可を求める必要はない。なお、O'Reilly から出版されている書籍のサンプルコードを CD-ROM として販売したり配布したりする場合には、そのための許可が必要である。本書や本書のサンプルコードを引用して問題に答える場合、許可を求める必要はない。ただし、本書サンプルコードのかなりの部分を製品マニュアルに転載するような場合には、そのための許可が必要である。

作者の帰属を明記する必要はないが、そうしてもらえるとうれしい。帰属を明記する際には、Ken Coar、Rich Bowen 共著『Apache クックブック』（Copyright 2008 O'Reilly Media,Inc.）のように、タイトル、著者、出版社、ISBN などを盛り込んでほしい。サンプルコードの使用について、正規の使用の枠を超える、またはここで許可している範囲を超えると感じる場合は、permissions@oreilly.com までご連絡いただきたい。

謝辞

最初は、すべてのレシピを筆者たちだけで作ろうと考えていたが、物理的に不可能だということがわかった。

本書を執筆中、たくさんの人が助けてくれた。課題を提示し、解決策を提供し、校正、レビュー、編集、「精神的な支援」をしてくれたみんなには、心から感謝している。Nat Torkington（本書の編集者であり、その忍耐でヘラクレスのような偉業を成し遂げてくれた）、Sharcoとirc.freenode.netの#apacheにいる人たち、Mad Toftum、Morbus Iff（Kevin Hemenway という偽名で FBI に知られている）と、Andy Holman には、特に感謝したい。

Ken Coar

本書を、私の大事な妻、Cathy Coarに捧げる。彼女は、20年以上の間、ヘラクレスのような偉大な愛と応援で私を支えてくれた。

神のような忍耐と理解を持った O'Reilly のスタッフには、心から感謝する。WriterBase 作者支援グループと Cabal メーリングリストで、たくさんの慰めと助言を与えてくれた人たちにも感謝する。テクニカルレビューアたちがとても優れたフィードバックをしてくれたおかげで、本書はよいものになった。

Apache Web サーバドキュメントに関わっている人たちと、Apache ソフトウェアを開発している人たちには感謝状を贈りたい。Apache Webサーバドキュメントがなければ、本書にこれだけたくさんの情報を盛り込むことは、到底不可能だっただろう。Apache 開発者がいなければ、本書は存在すらしなかっただろう。

Apacheソフトウェアのユーザは、メーリングリストやIRCチャンネル、メールを通して、やりがいのある課題を頻繁に投げかけてくれた。そのひらめきに感謝する。みんな無意識のうちに、本書のレシピを提供してくれたことになる。

しかし、最も私が感謝を捧げたいのは妻、Cathyだ。彼女の忍耐と応援、そして、建設的な批評がなければ、決して本書の執筆は成し遂げられなかっただろう。

Rich Bowen

本書を、毎日たくさんの質問に答えている#apacheのエキスパートたちと、そのエキスパートに質問する、将来エキスパートになっていくであろうビギナーたちに捧げる。

本書を世に出すのに関わった大勢の人たちに深く感謝する。Tatiana、我慢強く手伝ってくれてありがとう。

最後に、私のすばらしい家族に感謝したい。Sarah、君はいつも本屋で私の本を探してくれる。Isaiah、その無限のエネルギーと堅い抱擁に感謝する。最愛のものたちへ、私が無意味な美しさを見つけるのを手伝ってくれて感謝している。

目　次

コラム目次

1章 インストール

このクックブックを活用するには、とにかくApache Webサーバソフトウェアをインストールしておく必要がある。したがって、インストールを題材にしたレシピから始めるのが一番よいだろう。

Apache Webサーバをインストールするには、いろいろな方法がある。Apacheのようなオープンソースソフトウェアの特徴の1つは、誰でもインストールキットが作れることだ。Debianや FreeBSD、Red Hat、Mandrake、Hewlett-Packardといったベンダは、ファイルの配置やデフォルトの設定をカスタマイズして、他のソフトウェアとうまく合わせている。こうしたカスタマイズの結果、残念ながら、パッケージ済みのインストールキットは、お互いに全く違ったものになってしまっている。そのため、誰かの手助けがほしいときには、現在使っているインストールキットに詳しい人を探さなければならない。

各種パッケージ済みキットを使ってインストールすることもできるが、自分でソースコードからビルドしてインストールすることもできる。ソースから自分でインストールするのには、利点もあれば欠点もある。ソースからインストールすると、何がどこにインストールされるか、正確に把握することができるが、インストール先に選んだ場所がバイナリの追加パッケージが期待している場所と違って問題になることもある。

Webサーバを設定するのは今回限りで、2度とやらないというのであれば、システムベンダが提供しているパッケージを利用するのがよいだろう。しかし、ソースにパッチを当てたり、モジュールを追加、削除したり、サーバをあれこれいじりたいのであれば、ソースコードから自分でビルドするのが望ましい(本書の筆者はいじくりまわすのが習慣になっていて、「いつも」ソースコードからビルドしている)。

本章では、よく使われるパッケージ済みキットのインストール方法と、ソースコードから自分でサーバをビルドする方法について説明する。

なお本章では、サーバに静的にモジュールを組み込むのではなく、DSO(Dynamic Shared Object:動的共有オブジェクト)を使うことを想定している。DSOアプローチは非常にお勧めだ。これを使うと、サーバ全体を再ビルドしなくても、個々のモジュールを簡単にアップデートすることができる。さらに、サーバからモジュールを追加、削除したければ、設定ファイルを編集するだけでよい。

Unix系システムにおけるDSOは、通常、.soという拡張子になっている。Windowsでは、.dllという拡張子になる。

レシピ1.1 Red Hat Linuxのパッケージからインストールする

課題

Red Hat Linuxのサーバがあるので、Red Hatが提供、保守しているパッケージを使って、Apache Webサーバをインストール、または、アップグレードしたい。

解決

Red Hatのユーザ登録サービスであるRHN（Red Hat Network）のメンバになっていれば、Red Hatのup2dateツールを使って、Apacheパッケージを保守することができる。

```
# up2date -ui apache apache-devel apache-manual
```

最近のバージョンを使っているなら、次のように実行すればよい。

```
# up2date -ui httpd httpd-devel httpd-manual
```

RHNのメンバでなくても、Red Hatのサーバ（ftp://ftp.redhat.comやftp://updates.redhat.com）からパッケージをダウンロードし、次のコマンドでインストールすることができる。

```
# rpm -Uvh apache
```

解説

rpmコマンドの-Uvhオプションは、それぞれ次のような意味がある。

- U：システムにインストール済みのパッケージをアップグレードする。まだインストールされていなければインストールする。
- v：途中経過を表示して、インストールが順調に進んでいることを示す。
- h：シャープ記号（#）を使って、インストールの進捗状況を示す。

Red Hatが自社のプラットフォーム向けに保守しているパッケージを利用すれば、簡単で比較的に標準的な方法でインストールできるという利点がある。しかし、Red HatがRPMパッケージとしてまとめたバージョンにしかアップデートできない、という欠点もある。そのため、最新の安定したバージョンが使えるようになるのが、数週間から数ヶ月ほど遅れてしまうことがよくある。

またプラットフォームによる互換性の問題が生じることもある。例えば、Red Hat Linuxが提供するApacheのバージョンが1.3から2.0に変わると、新しいバージョンのOSでは2.0パッケージしか利用できなくなる可能性がある。同様に、Red Hat Linuxの古いバージョンを使っていると、新しいApacheパッケージがうまくインストールできないおそれがある。

apache-develパッケージも一緒にインストールしておくとよいだろう。このパッケージはかなり小さく、

ディスクの使用容量にそれほど影響しないが、サードパーティ製モジュールを正しくインストールするのに必要なファイルや機能を含んでいる†。

参照

- ftp://ftp.redhat.com/ にある Red Hat の全プラットフォーム向けリリースアーカイブ
- ftp://updates.redhat.com/ にある Red Hat の追加アップデート（正誤表）アーカイブ

レシピ 1.2　Debian のパッケージからインストールする

課題

Debian、または Debian をベースにした Ubuntu などのディストリビューションが動いているコンピュータがあり、そこに Apache をインストールしたい。

解決

apt-get を使って、apache2 パッケージをインストールすればよい。

```
# apt-get install apache2-mpm-prefork
```

解説

どんなパッケージシステムを使っている Linux ディストリビューションでも、最もよいのは、ディストリビューション自体が配布しているパッケージを使い続けることだ。アップデートは簡単だし、すでにインストールしてある他のパッケージとの互換性も高い。Debian の場合には、apt-get を使えばよい。

apache2-devパッケージも一緒にインストールしておくとよいだろう。apache2-devパッケージには、apxsのようなサードパーティ製モジュールをインストールするのに役立つユーティリティが含まれており、きっと役に立つはずだ。

Debianは設定ファイルの配置が独特で、他のディストリビューションとは違っている。モジュールとサイト（バーチャルホスト）はともに、サブディレクトリに置かれており、Apache の Debian バージョンに付属するユーティリティを使って、自由に有効、無効にすることができる。例えば、あるモジュールを有効にするには、a2enmodコマンドを使えばよい。このコマンドは、指定したモジュールをロードするよう、サーバの設定ファイルを適切に変更してくれる。例えば、rewrite モジュールを有効にするには、次のように実行する。

```
# a2enmod rewrite
```

Debian がどこにファイルやディレクトリを配置するかを知りたいなら、http://wiki.apache.org/httpd/DistrosDefaultLayout を調べるとよい。

参照

- http://wiki.apache.org/httpd/DistrosDefaultLayout

† 古いバージョンの Red Hat Linux はもはやサポートされていないので注意すること。

- man a2enmod
- man apt-get

レシピ 1.3 Windows に Apache をインストールする

課題

Windows プラットフォームに、Apache Web サーバをインストールしたい。

Windowsシステムにおいて、すでにApacheがインストール済みであれば、新しいバージョンをインストールする前に、必ずアンインストールしておく必要がある。これを忘れると、予期せぬ状況になるおそれがある。詳しくはレシピ 1.7 を参照。

解決

Windows は GUI 指向の環境であり、Windows 用の Apache インストールも GUI を備えている。

最も簡単なインストール方法は、Apache Webサイトhttp://httpd.apache.org/downloadからMSI(Microsoftソフトウェアインストーラ)パッケージをダウンロードして、実行することだ。以下では、この方法で実際にインストールしたときの画面例を説明する。

インストール手順の各ステップは独立しており、ファイルが実際にインストールされるまでは、これまでに選択した内容を変更することができる。最初の画面(図1-1)は、これから何をしようとしているのか、どのバージョンのパッケージをインストールしようとしているのかを確認しているだけだ。

2番目の画面(図1-2)は、Apacheのライセンスを表示している。その基本的な考えを要約すると次のようになる。「このソフトウェアを使って何をしても構わないが、許可なくApacheのマーク(羽根やApacheという名前などの商標)を使ってならない。また、Apache ソフトウェアに基づいてビルドしたものには、適切な帰属を付けなければならない。(これはパッケージを外部に配布する場合にのみ適用され、内部のネットワークで使うだけであれば必要ない。)」このライセンス条項に同意しないと、この画面より先に進めない。

図 1-1 Apache の MSI インストールの最初の画面

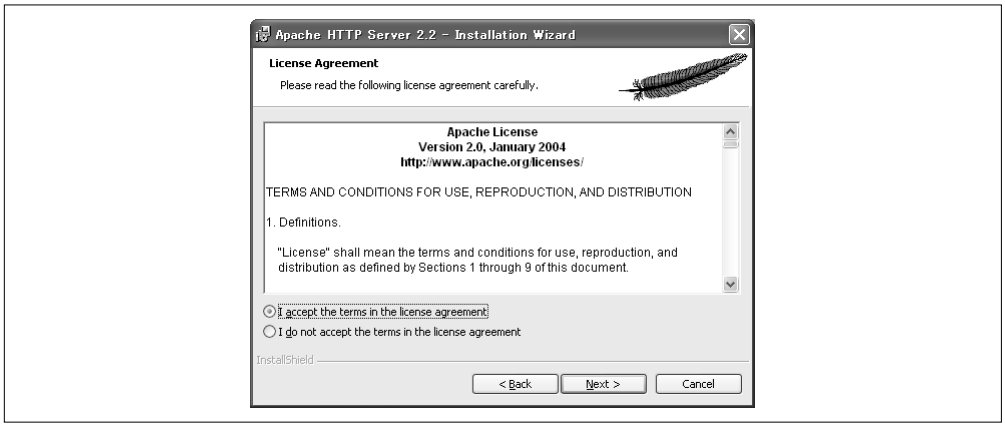


図 1-2　ライセンスへの同意

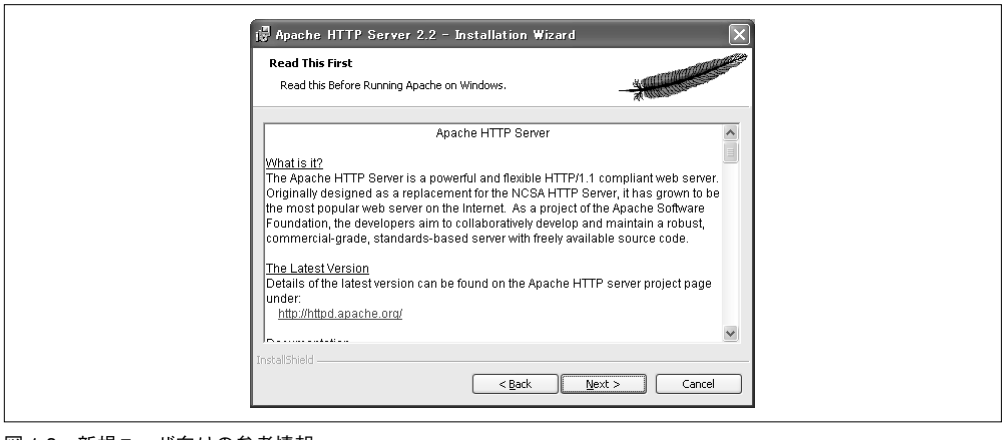


図 1-3　新規ユーザ向けの参考情報

図1-3は、Apacheソフトウェアの新規ユーザが読んでおくべき情報を示している。しっかり読んでおこう。

Apacheを初めてインストールする場合には、サーバの初期設定に関する情報を尋ねられる（図1-4）。すでにApacheがインストール済みの場合には、この画面で何を入力しても、既存の設定が上書きされることはない。

図1-4のサーバ名（Server Name）のフィールドには、ネットワークドメイン（Network Domain）のフィールドと同じ値を入力している。これは、Webサイトの"www"というプレフィックスを省略したドメイン名を使うことが増えてきているためだ（例えば、http://www.oreilly.com/ではなく、http://oreilly.com/を使う）。ここで指定したサーバ名は単なる助言のようなものだ。インストール作業中に初期値として設定されるが、設定ファイルを編集すれば、後から変更することができる。これはIPアドレスに解決可能なサーバ名である必要がある。

次の画面（図 1-5）では、パッケージのどの部分をインストールしたいか尋ねられる。上級ユーザ以外は、Typical（標準）を選択すればよいだろう。Custom（カスタム）オプションを選択すると、Apacheドキュメントをインストールするかどうかなど、インストールする項目を選択することができる。

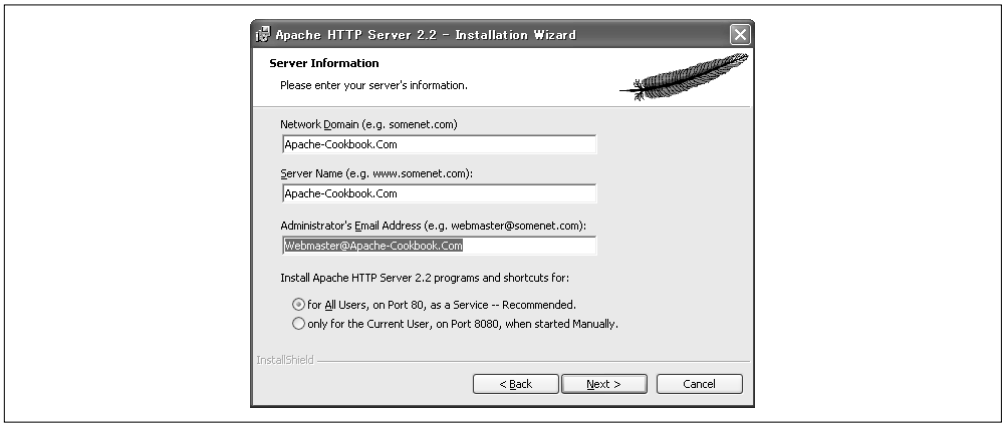


図 1-4　サーバの初期設定

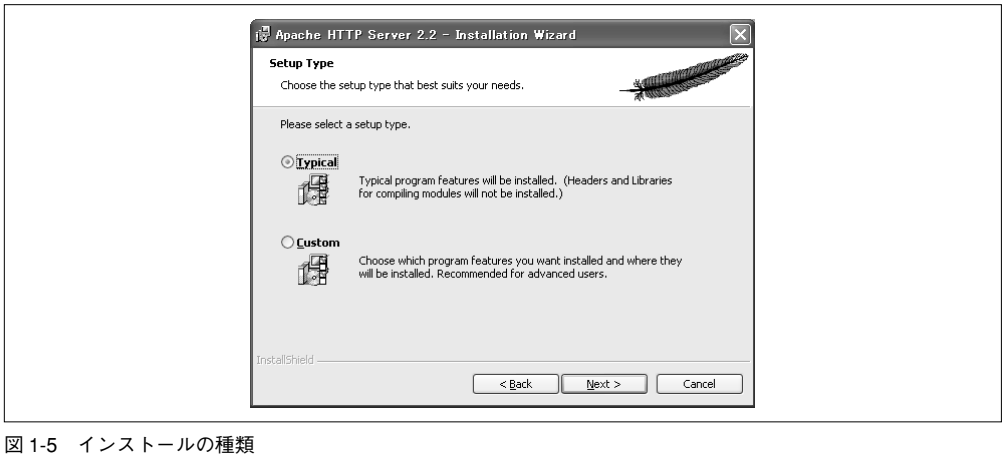


図 1-5　インストールの種類

図1-6では、ソフトウェアのインストール場所が尋ねられる。この画面では、デフォルトのインストール場所が表示されている。ここで指定した場所が ServerRoot になる。

すべての質問に答えると、図1-7のような画面が現れる。この画面が、前の画面に戻って設定内容を変更する最後の機会になる。ここでInstall(**インストール**)ボタンをクリックすると、指定した場所にパッケージがインストールされる。

図1-8と図1-9が、WindowsのMSIインストールの最後の画面であり、インストールの進捗状況を示している。処理が終了すれば、Apache のインストールは完了だ。(図 1-4 で "for All Users, on Port80, as a Service" (すべてのユーザが利用でき、ポート番号 80 でサービスとして起動する)オプションを選択していれば、Apache が起動する。)

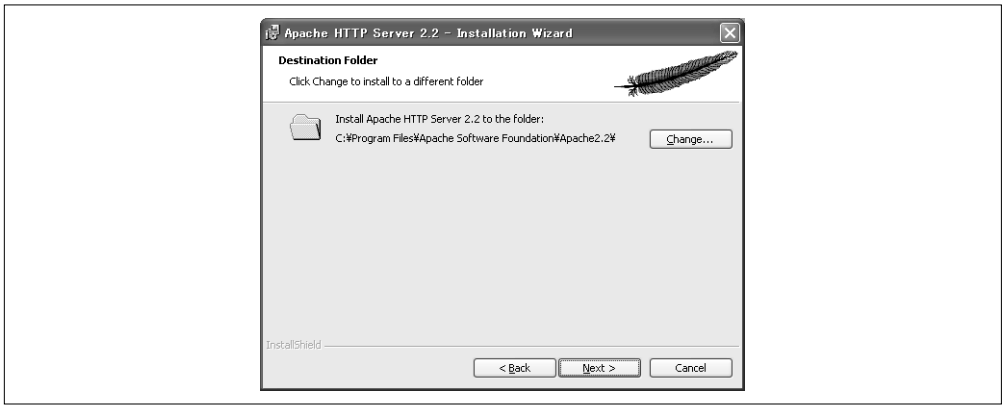


図 1-6　ServerRoot ディレクトリ

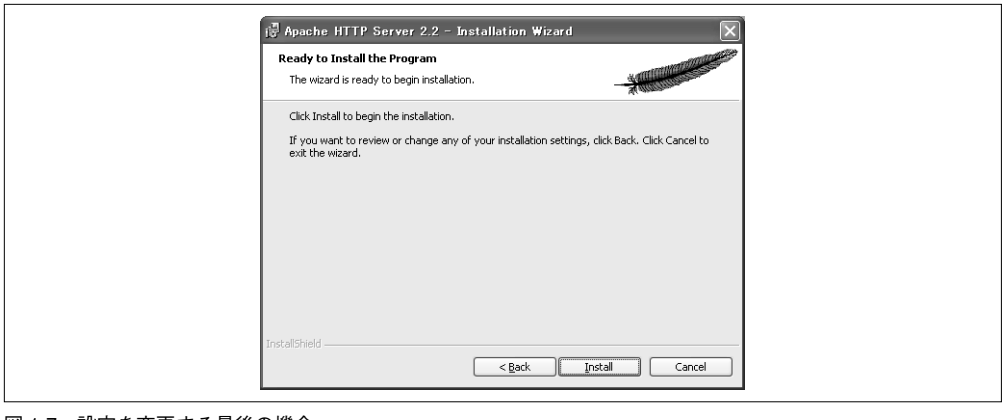


図 1-7　設定を変更する最後の機会

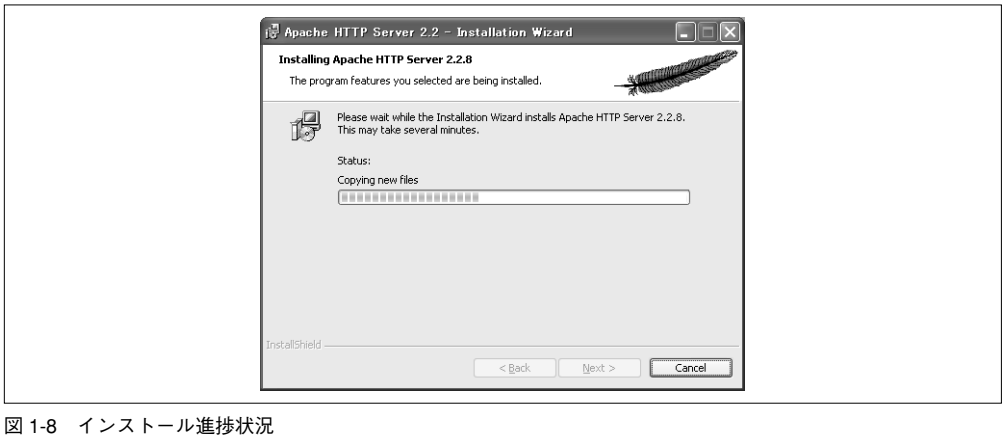


図 1-8　インストール進捗状況

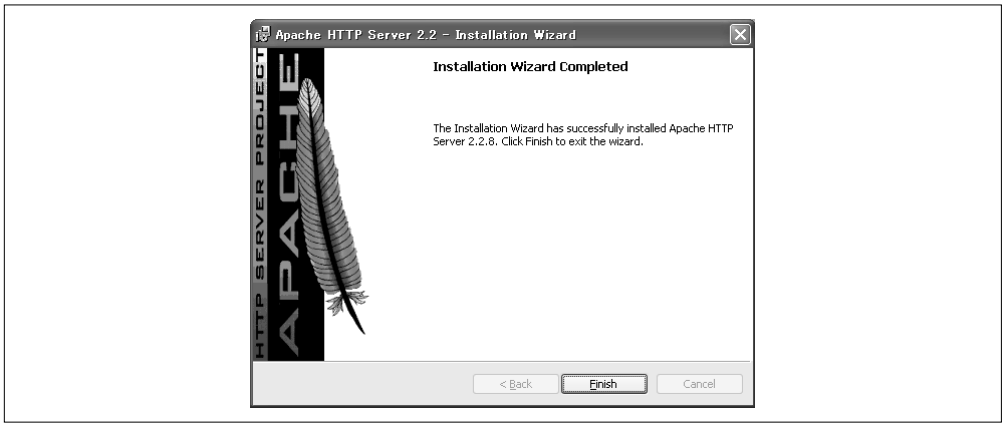


図 1-9　MSI インストールの完了

解説

Apache サーバを Windows でうまく動かし、さらに、他の Windows アプリケーションと同じように管理できるようにするためには、多大な労力が払われている。そのおかげで、Windowsユーザにはお馴染みの方法（InstallShield や MSI）で、基本的なインストールができるようになっている。

これまでApacheを動かしたことがなく、初めてインストールする場合には、デフォルトのままがよいだろう。こうしておくと、ファイルが所定の場所にインストールされるので、誰かの手助けが必要なときにわかりやすい。

Apache サーバをサービスとして起動するよう選択した場合（図 1-4 を参照）には、起動条件を変更することができる。どのユーザとして実行すべきか、自動的に起動すべきかなど、Windowsの他のサービスと同様に設定することができる。図 1-10 は変更方法の一例であり、スタートメニューの［**マイコンピュータ**］を右ク

図 1-10　Apache サービスの変更

リックして、ポップアップメニューから［**管理**］を選ぶと、このウィンドウが現れる。

参照

- http://www.apache.org/licenses/ にある Apache ライセンス
- レシピ 1.7

レシピ 1.4　Apache のソースコードをダウンロードする

課題

Apache Web サーバをソースコードから自分で直接ビルドしたいが（レシピ 1.5 を参照）、どこからソースコードを入手すればよいかわからない。

解決

ソースコードを入手するには、いろいろな方法がある。Subversion を使うと、ほぼリアルタイムで最新バージョンにアクセスすることができる。リリースされた tarball（tar 形式のアーカイブ）をダウンロードしてもよいし、使っているディストリビューションが用意したソースパッケージをインストールしてもよい。パッケージ済みの tarball からインストールする場合には、まず、http://httpd.apache.org/download.cgi から tarball をダウンロードし、次のコマンドを実行する。

```
% tar xzvf httpd-2.2.8.tar.gz
```

使っている tar のバージョンが、zip 圧縮されたアーカイブを処理する z オプションをサポートしていない場合には、代わりに次のコマンドを実行すればよい。

```
% gunzip -c < httpd-2.2.8.tar.gz | tar xvf -
```

最新の Apache2.2 ソースリポジトリからインストールする場合には、次のコマンドを実行する。ただし、完全に動作する保障はないことに注意しよう。

```
% svn checkout http://svn.apache.org/repos/asf/httpd/httpd/branches/2.2.x/ httpd-2.2
```

開発者が付けたタグ名がわかっているなら、最新のコードの代わりに、特定のリリースバージョンを取得することもできる。例えば、次のようにすると、最新バージョンではなく、より安定していると思われる 2.2.8 リリースのソースコードを取得することができる。

```
% svn checkout http://svn.apache.org/repos/asf/httpd/httpd/tags/2.2.8/ httpd-2.2.8
```

ソースツリーで使われているタグ名を知るには、http://svn.apache.org/viewvc/httpd/httpd/tags/にアクセスしてみるか、次のコマンドを実行すればよい。

```
% svn ls http://svn.apache.org/repos/asf/httpd/httpd/tags/
```

開発者は、いろいろな目的でタグを使っている。リリース用のファイルのバージョンには、必ず*n.m.e*という形式のタグが付いている。特定のリリースバージョンを取得したいときには、このタグを使うとよい。

解説

どんな方法でソースをインストールしたとしても、ソースツリーはいつでも設定、ビルドできる状態になっている。ソースが準備できれば、すぐにパッケージのビルドに進むことができる（レシピ 1.5 を参照）。

Subversionを使ってソースをインストールした場合には、ソースを常に最新にしておくことができる。ソースディレクトリの最上位で、次のコマンドを実行すればよい。

```
% svn update
```

このコマンドは、最後にダウンロード、更新したときから、開発者が変更または追加したファイルを、更新または取得する。ソースコードを最新バージョンに更新すると、開発者が現在作業中のファイルを取得することができる。更新作業は、一部のファイルを取得するだけで完了するだろう。信頼性を求めるのであれば、リリースされているバージョンで我慢しよう。その方が広くテストされており、より信頼性が高い。

参照

- レシピ 1.5

レシピ 1.5　ソースコードから Apache をビルドする

課題

パッケージ済みのインストールキットからインストールするのではなく、ソースコードから直接Apache Web サーバをビルドしたい。

解決

ここでは、すでにApacheのソースツリーが手元にあるものとする。tarballからでも、Subversionからでも、配布パッケージからでも、どのようにインストールしたソースでも構わない。ソースツリーの最上位ディレクトリで次のコマンドを実行すると、ほとんどの標準モジュールをDSOとして使うサーバをビルドすることができる。

```
% ./buildconf
% ./configure --prefix=/usr/local/apache \
> --with-layout=Apache --enable-modules=most --enable-mods-shared=all \
> --with-mpm=prefork
% make
# make install
```

各種オプションとその意味について、もっと詳しく知りたいなら、次のコマンドを実行すればよい。

```
% ./configure --help
```

解説

ソースコードからサーバをビルドするのは、複雑で時間もかかる。しかし、ソースコードを変更したければ、自分でビルドするしかない。自分でビルドすれば、共有オブジェクトライブラリを使ったり、モジュールで利用可能なデータベースルーチンを使ったりするなど、もっと自分自身でサーバをコントロールすることができる。独自のApacheモジュールを開発する場合にも、必ずソースコードからビルドする必要がある。

モジュールを静的にサーバに組み込みたければ、--enable-mods-shared=*list* を --enable-mods=*list* に置き換えればよい。

configureスクリプトには、いろいろなオプションがある。これまでconfigureを使ってApacheをビルドしたことがなければ、デフォルトのオプションを変更するときに、オンラインのチュートリアル（例えば、http://httpd.apache.org/docs-2.2/install.html、http://httpd.apache.org/docs-2.0/install.html）を調べてみよう。デフォルトのオプションでもちゃんと動くサーバをビルドすることはできるが、ファイルシステムの場所やモジュールの選択が好みとは違ったり、不要なモジュールが含まれてしまったり、必要なモジュールがなかったりするかもしれない（例については 2 章を参照）。

参照

- レシピ 1.4
- http://httpd.apache.org/docs-2.2/install.html
- http://httpd.apache.org/docs-2.0/install.html

レシピ 1.6　Apache を起動、停止、再起動する

課題

適切なツールを使って、必要なときにサーバの起動、停止ができるようにしたい。

解決

Unix系システムでは、apachectlスクリプトを使えばよい。Windowsでは、スタートメニューにあるApacheフォルダのオプションを使えばよい。

解説

Apacheの基本パッケージには、簡単にサーバをコントロールするためのツールが含まれている。Unix系システムの場合、通常はapachectlという名前のスクリプトがあるはずだ。ただし、パッケージ済みのディストリビューションの場合、別のツールに置き換えられていたり、名前が変わっていることがあるので注意しよう。このツールは、一度に 1 つのアクションを実行する。どのアクションを実行するかは、コマンドラインの引数で指定する。以下に、主要なオプションを挙げる。

apachectl start
: サーバがまだ動いていない場合には、サーバを起動する。すでに動いている場合には、何もせずに、警告メッセージを表示する。

apachectl graceful
: このオプションを指定すると、サーバは設定ファイルを再ロードして、行儀良く再起動する。現在接続中のコネクションはそのまま継続することができる。サーバがまだ動いていない場合には、サーバを起動する。

apachectl restart
: gracefulオプションと同様に、このオプションを指定すると、サーバは設定ファイルを再ロードして再起動する。ただし、現在接続中のコネクションは即座に切断する。サーバがまだ動いていない場合には、サーバを起動する。

apachectl stop
: このオプションを指定すると、サーバを即座にシャットダウンする。現在接続中のコネクションは即座に切断する。

WindowsでApacheのMSIインストールを実行した場合には、サーバをコントロールするためのメニュー項目(図1-11)が追加されているはずだ。

ここでは、基本的なサーバのコントロール操作(起動、停止、再起動)について、Unix系システムとWindows、それぞれの解決策を説明した。起動、停止の目的は言うまでもないだろう。サーバ全体の設定ファイル(例え

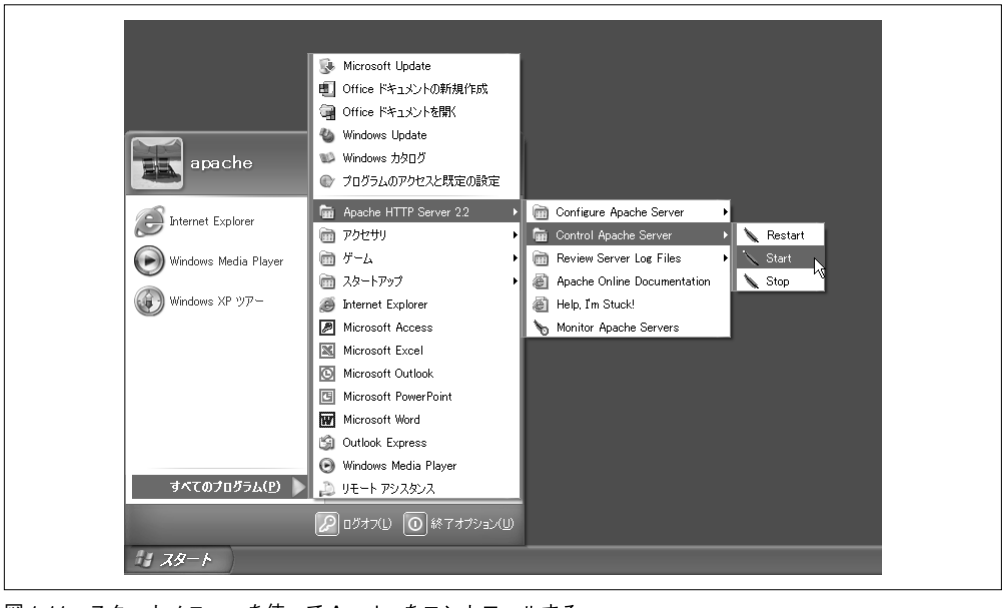


図1-11 スタートメニューを使ってApacheをコントロールする

ば、httpd.conf)を変更したときには、その変更を反映させるために必ずサーバを再起動する必要がある。

参照

- レシピ 1.1
- レシピ 1.3

レシピ 1.7　Apache をアンインストールする

課題

システムにインストールした Apache ソフトウェアを削除したい。

解決

Red Hat Linux で、RPM ツールを使って Apache をインストールした場合、次のコマンドを実行することによって、Apache を削除することができる。

```
# rpm -ev apache
```

他のパッケージシステムにも同様の仕組みがあるはずだ。こういった削除の仕組みがない場合、すべてのファイルをきれいに消してしまうには、非常に面倒な手作業が必要になる。

Windowsの場合、MSIでインストールした他のソフトウェアと同じように、Apacheを削除することができる(図 1-12 を参照)。

解説

残念ながら、Unix系システムにインストールされたApacheを削除する一般的な方法はない。例えば、Red HatのRPMのようなパッケージシステムでは、何をインストールしたか覚えているため、この解決策にあるように、すべての構成要素を削除することができる。しかし、ソースからビルドしてインストールした場合(レシピ1.5を参照)、どこにファイルが置かれているかを覚えておくのは、ビルドしてインストールした人の責任だ。Windowsの場合も、ソースからインストールしたのであれば、同じことが言える。コントロールパネルの［**プログラムの追加と削除**］が使えるのは、MSI や InstallShield パッケージのおかげだ。

Unix系システムでは、サーバをビルドしたディレクトリにアクセスできるなら、そこにあるconfig.niceファイルの `--prefix` オプションを探してみよう。少なくとも出発点にはなるはずだ。Apache 2.0 をインストールした場合、ディスク上に作られるディレクトリは次のようになっている。

```
bin
build
cgi-bin
conf
error
```

図 1-12　Apache ソフトウェアをアンインストールする

htdocs
icons
include
lib
logs
man
manual
modules

参照

- レシピ 1.5

レシピ 1.8　どのバージョンの Apache を使えばよいか

課題

自分にとって、どのバージョンの Apache を使えばよいか知りたい。

解決

全員が満足するたった1つの正解というものはない。Apache HTTP サーバの開発チームは、非常によい仕事をしており、リリースしたソフトウェアはすべて非常に品質がよく、安定しており、安全な製品になっている。またリリースするたびに、以前のリリースで見つかった問題が修正されている。したがって、常に最新のバージョンを使うべきだ、というのが筆者らの意見だ。

つまり、本書の執筆時点では、2.2.xブランチの最新リリース（現在、2.2.8）になる。2.4がリリースされたときには、2.4 にアップグレードするのがよいだろう。

解説

この質問は思ったほど簡単ではない。唯一の解を示したいところだが、古いバージョンのソフトウェアを使い続けるのに非常に説得力のある理由があるときもある。しかし、こうした理由も、数年前と比べると、かなり根拠が薄れてきている。

バージョン1.3を使い続けている人が最もよく主張する理由は、「mod_somethingを使っているが、このモジュールはまだバージョン 2.2 では使えない」というものだ。Apache 2 が登場して間もないときには、これはもっともな理由であった。Apache 2では使えないモジュールがまだたくさんある、ということは誰もが納得する常識だった。

しかし、多くの主要な Linux ディストリビューションが Apache 2 をデフォルトの Web サーバに採用し始めると、Apache 2で使えるモジュールもだんだん増えてきた。また、同じ機能を実装した別のモジュールを作る人も現れてきて、この説得力のあった理由もだんだん真実ではなくなってきた。

本書執筆時点で、まだ「Apache 2 では利用不可能」という状態にあるのは、ほんのわずかな商用モジュールだけのようだ。もはや、かつてほど説得力のある理由ではなくなっているのである。

もう1つよく言われる理由は、「Apache 1.3でたくさんのバーチャルホストと複雑な設定のサイトを大規模に運用しており、これを Apache 2へ移行するには膨大な作業が必要だ」というものだ。これも大変説得力のある理由だ。しかし、Apache 1.3は保守のみという状態であり、2.xブランチで開発されている新機能は決して使えないということを認識しておく必要がある。また、おそらくもっと重要なのは、Apacheのフリーオンラインサポートをしてくれている人の多くは、すでにApache 2を使っており、Apache 1.3のことをだんだん忘れていくということだ。したがって、安定したインストールがあり、技術的な困難もなく、新しい機能に後れをとっても構わないと言うのであれば、1.3 を使い続けるのは妥当な解になるだろう。

しかし、これから新しいWebサーバをインストールするのであれば、最新バージョンを使わない理由はないはずだ。1.3の時代に得た経験は役に立つし、2.2の新機能を使うことができる。従来からある機能にも、よりよい実装を手に入れることができる。

参照

- http://httpd.apache.org/docs/2.2/new_features_2_2.html
- http://httpd.apache.org/docs/2.2/new_features_2_0.html
- http://www.oreilly.com/catalog/9780596529277/

レシピ 1.9　config.nice を使ってアップグレードする

課題

ソースコードから Apache Web サーバをビルドしたが、全く同じ設定オプションを使ってアップグレードしたい。

解決

新しいバージョンのソースを以前と別のツリーに展開し、以前のバージョンのビルド時に作られたconfig.niceスクリプトを実行すればよい。

このテクニックは、主に同じメジャーバージョン間、例えば、2.0.17から2.0.59や、2.2.0から2.2.4へアップグレードすることを想定している。新しいメジャーバージョン(例えば、2.0.17から2.2.4)にアップグレードするときに古い設定オプションを使ってしまうと、うまく動かないおそれがある。

例えば、ずっと以前にバージョン2.0.17をビルドしてインストールしてあり、これを2.0.59にアップグレードする場合には、次のようにすればよい。

```
# cd /usr/local/build
# tar xvf /tmp/httpd-2.0.59.tat.gz
# cd httpd-2.0.59
# ../httpd-2.0.17/config.nice
# make
```

解説

コンパイルオプションやインストールオプションを指定してconfigureスクリプトを実行すると、config.niceという名前のファイルが作られる。config.niceスクリプトは、以前指定したオプションを付けてconfigureを実行する。したがって、最後に動かしたときに指定したオプションを自分ですべて覚えておいたり、書き残しておいたりする必要はない。

config.niceに追加オプションを指定すると、そのオプションを追加してconfigureを実行することができる。configureを実行した結果、そのオプションを追加した、新しいconfig.niceがまた作られることになる。

参照

- レシピ1.5

レシピ1.10 システム起動時にApacheを起動する

課題

システム起動時に、Apache Webサーバを自動的に起動したい。

解決

Windowsでは、Apacheをサービスとしてインストールすると、他のサービスと同様に、自動的に起動するよう設定することができる。管理ツールのサービスを開いて、変更するだけでよい。Unix系システムでは、プラットフォームによって設定方法は違ってくる。Red Hatベースのシステムでは、次のようにすればよい。

```
# cp path/to/apachectl /etc/rc.d/init.d/httpd
# vi /etc/rc.d/init.d/httpd   # '# chkconfig 3 92 10' を追加
# chkconfig --add httpd
# chkconfig --levels 35 httpd on
```

これにより、実行レベル3と5の通常シーケンスの一部として、Apacheを起動(または、終了)することができる。

解説

この解決策は、Red Hatベースのプラットフォーム(例えば、Fedora CoreやRHEL)に特有のものだ。他のプラットフォームやディストリビューションの場合には、代わりに/etc/rc.localを編集したり、apachectlスクリプトを/etc/rc3.dなどにコピーしたりする必要があるだろう。具体的には、オペレーティングシステムのドキュメントを調べてほしい。

参照

- レシピ 1.5
- レシピ 1.6

レシピ 1.11　便利な設定オプション

課題

ソースコードからApacheをビルドするときに使うconfigureスクリプトには、たくさんのオプションがあるが、どのオプションが本当に重要なのかよくわからない。

解決

使いたいと思うかもしれない、最も重要で役に立つオプションをいくつか紹介しておく。

`--prefix`
: ファイルを置くディレクトリツリーの最上位を指定する。Apache 2.xの場合、デフォルトは通常、`--prefix=`*`/usr/local/apache2`*になっているが、配置を変更したければ、このオプションで変更することができる(次の`--enable-layout`オプションも参照)。

`--enable-layout`
: このオプションを使うと、あらかじめ定められたファイルシステム構成、すなわち、`make install`でファイルが置かれる場所を選択することができる。どのレイアウトがどこにファイルをインストールするかは、ソースツリーの最上位にあるconfig.layoutを調べるとわかる。現在定義されているレイアウトを以下に示す。

Apache
beos
BSDI
Darwin
Debian
FreeBSD
GNU
Mac OS X Server
OpenBSD
opt
RedHat
Solaris
SuSE

空白を含むレイアウト名を使うには、クォーテーションマーク('や") で囲まなければならない。

```
% ./configure --enable-layout="Mac OS X Server"
```

--enable-mods-shared
このオプションを使うと、どのモジュールを静的リンクではなくDSOとしてビルドするか、コントロールすることができる。*most* という便利なショートカットがある。

--enable-ssl
安全性の高いサーバを動かしたいのであれば、このオプションを付ける必要がある。SSLモジュールはデフォルトでは無効になっている。

--enable-suexec
suexec機能を組み込んでビルドしたいときには、このオプションを指定する。他のサーバビルド部分にも依存しているため、後からsuexecをビルドするのではなく、メインのサーバビルド設定時に指定しておく必要がある。

--with-apr, --with-apr-util
複数バージョンのApache Portable Runtimeライブラリとユーティリティがインストールされている場合(例えば、Subversionにあるソースから Apacheをビルドしているシステムの場合はこうなっているかもしれない)、Apacheサーバが互換性のあるバージョンのAPRで確実にビルドされるよう、これらのオプションで指定しなければならないことがある。

--with-included-apr
このオプションは、互換性のあるバンドルされたバージョンのAPRが使われるように指定する便利で素早い方法だ。しかし、このオプションはApache 2.2以降のバージョンでしか利用することができない。

--with-mpm
　マルチプロセッシングモデルまたはMPMは、サーバがリクエストをどのように処理するか、スレッドと子プロセスの関係を定義したものだ。configureスクリプトは通常、プラットフォームに適したものを選択してくれるが、自分で上書きしたいときもある。例えば、PHPスクリプトモジュールを利用するつもりなら、問題を避けるためにも prefork MPM を使う必要があるだろう。

--with-port
　このオプションは、rootでないユーザでサーバをビルドしたが、システムデーモンとして動かしたいときに役に立つ。configure スクリプトは、root であるかないかによって、デフォルトのポート番号を選んでいる。このオプションを付けると、この動作を上書きすることができる。最もよく使われるのは次のオプションだ。

```
--with-port=80
```

解説

次のようにスクリプトを実行すると、すべてのconfigureオプションに関する最低限のドキュメントが表示される。

```
% ./configure --help
```

しかし、もっと詳しく理解するには、Apacheのドキュメントを調べるか、ソースコードを調べる必要があるだろう。

参照

- http://httpd.apache.org/docs/2.2/install.html
- http://httpd.apache.org/docs/2.0/install.html

レシピ 1.12　Apache のファイルを見つける

課題

ソースコードまたはインストールキットを使って Apache Web サーバをインストールしたが、ファイルがどこに置かれたのか知りたい(これは後でアンインストールするときに役立つ)。

解決

ソースコードからソフトウェアをインストールした場合には、ソースディレクトリの最上位にあるconfig.layout ファイルを調べればよい。configure スクリプトに与えられた --enable-layout オプションと一致する <Layout> ブロックを探す。(もし何も指定されていなければ、Apache レイアウトが使われているはずだ。)

RPM パッケージからインストールした場合には、-ql オプションを使うと、ファイルがインストールされた場所を調べることができる。

```
% rpm -ql httpd
```

Ubuntu などのディストリビューションが用意したキットからインストールした場合には、ディストリビューションのドキュメントを調べて、どこにファイルが格納されているかを見つければよい。

解説

誰でもインストールキットが作れるのは、オープンなソフトウェアの利点であり欠点でもある。誰もが他人とは違ったオプションを選ぶことができる。Apache ソースパッケージには「共通」レイアウトのリストがあり、多くのインストールキットはこのうちのいずれかを使っている。

参照

- http://httpd.apache.org/docs/2.2/programs/configure.html
- http://httpd.apache.org/docs/2.0/programs/configure.html

2章
一般的なモジュールの追加

Apache Webサーバには、基本ディストリビューションには含まれていないが、非常に人気のあるモジュールがたくさんある。こうしたモジュールが別に配布されているのは、ライセンスやサポートのためだ。Apache開発者の判断により、Apache Software Foundation から配布されていないモジュールもある。また、別のプロジェクトにとって不可欠なものもある。例えば、Apache 1.3用のmod_sslは、Apacheとは別に開発、保守されている。これは米国の輸出規制法のため(このパッケージが最初に開発されたときには、今よりもっと厳しい制限があった)だけでなく、コアのソフトウェアにも変更が必要になるのでApache開発者が統合しないことを決めたという事情もある。

本章では、サードパーティ製モジュールのうち特に人気があるモジュールをインストールするためのレシピを紹介する。Unix 系システムと Windows のインストールを別々のレシピで説明している場合もある。

最もまとまったサードパーティ製モジュールの一覧は、http://modules.apache.org にある Apache Module Registryだろう。本章で紹介するような、とても人気のあるモジュールや複雑なモジュールには専用のサイトが存在しているものもある。

たくさんのサードパーティ製モジュールが利用可能だが、モジュール開発者の多くは1つのモジュールにしか関心がない。そのためモジュールの数だけインストール方法があるということになりかねない。本章の最初のレシピでは、たいていのApache 1.3のモジュールに適用することができるインストール方法を説明する。しかし、この方法とは別のもっときめ細かな手順があるかもしれないので、個々のパッケージの取り扱い説明書をチェックする必要がある。

モジュールの多くは、Apacheソフトウェアパッケージの作成や配布をしている団体から(例えば、Mandrake社やRed Hat社からRPMの形で)入手することができる。しかし、こうしたビルド済みのモジュールパッケージの場合、パッケージ作成者が特定のインストール環境を想定していることがある。したがって、サーバをソースからビルドしてファイルの配置場所をカスタマイズして使っている場合、パッケージ化されたモジュールのインストールに失敗してしまうことがよくある。

本章で紹介するモジュールはすべて、Unix 系システム上の Apache 1.3 に対応している。Windows および Apache2.0 の対応状況については、表 2-1 に示した。

表 2-1 モジュール対応状況

モジュール名	Windows 対応	Apache 2.0 対応
mod_dav	はい	含まれており、インストールは不要
mod_perl	はい	はい
mod_php	はい	はい
mod_ssl	はい	含まれており、インストールは不要

レシピ2.1 一般的なサードパーティ製モジュールをインストールする

課題

本章で紹介されていないサードパーティ製モジュールをダウンロードしたが、これをインストールしたい。

解決

モジュールのソースファイルを展開したディレクトリに移動して、次のコマンドを実行する。

```
% /path/to/apache/bin/apxs -cia module.c
```

解説

そのサードパーティ製モジュールが.cファイル1つだけなら、この解決策にあるようにすれば、ビルドしてインストールできるだろう。モジュールが複数のソースファイルで構成されている場合には、そのモジュール固有のインストール手順があるはずだ。

-ciaオプションはコンパイルして(compile)、インストールして(install)、有効にする(activate)ことを意味している。インストールとは、Apacheがモジュールを検索する場所に.soファイルを置くことを意味している。有効にするとは、モジュールが使えるようにhttpd.confファイルに設定を追加することを意味している。

参照

- apxs の manpage。通常、ServerRoot/man/man8/apxs.8 にある。

レシピ2.2 Unix系システムにmod_davをインストールする

課題

サーバにWebDAV†機能を追加して、利用可能にしたい。WebDAVを使うと、FTPを使わなくても、信頼性、安全性の高い方法で、リモートユーザがファイルの追加、削除、更新などのドキュメント操作を行うことができる。

† WebDAV（Web Distributed Authoring and Versioning）とは、HTTP 1.1を拡張して、分散環境でWebサーバ上のファイルの管理を実現するためのプロトコルだ。RFC 2518で定義され、現時点では、RFC 4918が最新バージョンである。DAVと呼ばれることもある。

解決

Apache 2.0以降を使っているのであれば、mod_davは自動的に利用可能になっている。ただし、コンパイル時に--enable-davを付けておく必要がある。Apache 1.3を使っているのであれば、http://webdav.org/mod_dav/からmod_davのソースパッケージをダウンロードして、展開し、次のコマンドを実行する。

```
% cd mod_dav-1.0.3-1.3.6
% ./configure --with-apxs=/usr/local/apache/bin/apxs
% make
# make install
```

そして、サーバを再起動する。レシピ6.18を読んでおくこと。

解説

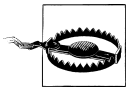
mod_davソースパッケージは、カプセル化された、よくできたモジュールだ。既存のサーバにも簡単にビルドして組み込むことができる。うまくインストールできたか確かめるには、WebDAVで管理する場所をサーバ上に作り、WebDAV対応のツールでその場所にアクセスできることを確認すればよい。ツールにはcadaverがお勧めだ。cadaverはオープンソースのコマンドラインベースのWebDAVツールである。（cadaverツールのURLはこのレシピの最後にある）。

サーバでWebDAV操作ができるようにするには、少なくともhttpd.confファイルに2つのディレクティブを追加する必要がある。最初のディレクティブは、ロックデータベースの場所を示している。ロックデータベースは、WebDAV操作が衝突するのを防ぐために使用され、サーバが書き込み可能なディレクトリでなければならない。例えば、次のようにすればよい。

```
# cd /usr/local/apache
# mkdir var
# chgrp nobody var
# chmod g+w var
```

そして、httpd.confファイルの他のコンテナの外部に、次の行を追加する。

```
<IfModule mod_dav.c>
    DAVLockDB var/DAVlock
</IfModule>
```



DAVLockDBの場所は、NFSでマウントされたファイルシステム上であってはならない。NFSは、mod_davに必要とされるロック機能をサポートしていないためだ。NFSファイルシステム上にロックデータベースを置いてしまうと、予期せぬ結果になるだろう。

次に、WebDAV機能をテストするために、一時的なディレクトリを作成する。

```
# cd /usr/local/apache
# mkdir htdocs/dav-test
# chgrp nobody htdocs/dav-test
# chmod g+w htdocs/dav-test
```

このディレクトリでWebDAV操作ができるようにするため、httpd.confファイルに次のセクションを追加する。

```
<Directory "/usr/local/apache/htdocs/dav-test">
    DAV On
</Directory>
```

そして、サーバを再起動する。ローカルURI /dav-testは、WebDAV操作を受け付けるようになっているはずだ。cadaverツールを使ってテストするには、次のコマンドを実行すればよい。以下のような出力が得られるはずだ。

```
% cd /tmp
% echo "Plain text" > dav-test.txt
% cadaver
dav:!> open http://localhost/dav-test
Looking up hostname... Connecting to server... connected.
dav:/dav-test/> put dav-test.txt
Uploading dav-test.txt to '/dav-test/dav-test.txt': (reconnecting...done)
Progress: [= == == == == == == == == == == == == == ==>] 100.0% of 11
    bytes succeeded.
dav:/dav-test/> propset dav-test.txt MyProp 1023
Setting property on 'dav-test.txt': (reconnecting...done) succeeded.
dav:/dav-test/> propget dav-test.txt MyProp
Fetching properties for 'dav-test.txt':
Value of MyProp is: 1023
dav:/dav-test/> propdel dav-test.txt MyProp
Deleting property on 'dav-test.txt': succeeded.
dav:/dav-test/> close
Connection to 'localhost' closed.
dav:!> exit
% rm dav-test.txt
```

ここで、propertyとは、WebDAVリソースの属性のことだ。リソースサイズのようにシステムが管理しているものもあれば、ユーザが自由に追加、変更、削除できるものもある。

mod_davが正しく動くことを確認したら、htdocs/dav-testディレクトリを削除し、httpd.confファイルから対応する<Directory>ブロックを削除しておこう。レシピ 6.18 にあるガイドラインに従うこと。

参照

- レシピ 6.18
- http://webdav.org/mod_dav
- http://webdav.org/cadaver

レシピ 2.3　Windows に mod_dav をインストールする

課題

mod_dav を使って、既存の Apache 1.3 サーバで、WebDAV 機能が使えるようにしたい。

解決

Apache 2.0 には mod_dav が標準モジュールとして組み込まれているので、自分で mod_dav をダウンロードしてビルドする必要はない。

Apache 1.3の場合は、自分でインストールする必要がある。http://webdav.org/mod_dav/win32からWindows用の mod_dav パッケージをダウンロードして、展開する。インストール済みの Apache の ServerRoot ディレクトリに、xmlparse.dllとxmltok.dllがあることを確認しておく。もしなければ、Apacheディレクトリを調べて、この 2 つのファイルを ServerRoot にコピーしておく必要がある。mod_dav には Expat パッケージが必要であり、Expat パッケージは Apache Web サーバのバージョン 1.3.9 以降に含まれている。Expat はこれらの DLL ファイルをロードし、mod_dav がこれらの DLL ファイルを利用する。

mod_dav の DLL ファイルを Apache のモジュール用ディレクトリに置く。

```
C:\>cd mod_dav-1.0.3-dev
C:\mod_dav-1.0.3-dev>copy mod_dav.dll C:\Apache\modules
C:\mod_dav-1.0.3-dev>cd \Apache
```

次に、httpd.conf ファイルに次の行を追加する。

```
LoadModule dav_module modules/mod_dav.dll
```

httpd.confファイルの中で、ClearModuleListディレクティブが使われており、その後でモジュールを追加し直している場合には、mod_davをAddModuleで追加する必要があるだろう。代わりに、ClearModuleListディレクティブの後に、LoadModule で mod_dav を入れてもよい†。

解説

mod_davソースパッケージは、カプセル化された、よくできたモジュールだ。既存のサーバにも、簡単にビルドして組み込むことができる。うまくインストールできたか確かめるには、WebDAVで管理する場所をサーバ上に作り、WebDAV対応のツールでその場所にアクセスできることを確認すればよい。WebDAVにアクセスできる Windows Explorer（Windows 2000 以降）で閲覧するか、cadaver などの WebDAV ツールが使え

† Apache 2.0 以降で、ClearModuleList ディレクティブと AddModule ディレクティブは削除された。

る別のシステムからアクセスして確認してもよいだろう。

サーバでWebDAV操作ができるようにするには、少なくともServerRoot/conf/httpd.confファイルに2つのディレクティブを追加する必要がある。最初のディレクティブは、ロックデータベースの場所を示している。ロックデータベースは、WebDAV操作が衝突するのを防ぐために使用され、サーバが書き込み可能なディレクトリでなければならない。例えば、次のようにすればよい。

```
C:\Apache-1.3>mkdir var
```

WebDAV を利用可能にするため、httpd.conf ファイルに次の行を追加する。

```
<IfModule mod_dav.c>
    DAVLockDB "C:/Apache-1.3/var/dav-lock"
</IfModule>
```

次に、mod_dav がうまく動いているかテストするために、一時的なディレクトリを作成する。

```
C:\Apache-1.3>mkdir htdocs\dav-test
```

このテスト用のディレクトリで WebDAV 操作ができるようにするため、<IfModule> コンテナを変更する。

```
<IfModule mod_dav.c>
    DAVLockDB "C:/Apache-1.3/var/dav-lock"
    <Directory "C:/Apache-1.3/htdocs/dav-test">
        DAV On
    </Directory>
</IfModule>
```

そして、サーバを再起動して、WebDAVクライアントで/dav-testにアクセスしてみる。もし別のシステムからcadaverを使ってアクセスしているなら、詳しい説明はレシピ2.2を参照すること。Windows Explorerを使って mod_dav をテストしたいのであれば、以降の説明を読むとよい。

Windows Explorer を使って mod_dav をテストする

htdocs\dav-test ディレクトリで WebDAV 操作ができるようにしたら、サーバを再起動して、Windows Explorerを起動する。次の手順に従って、WebDAVでディレクトリにアクセスしてみよう。Windows Explorerは、サーバと同じローカルシステム上で実行してもよいし、サーバにアクセス可能な別のWindowsシステム上で実行してもよい。

1. 「**マイ ネットワーク**」をクリックする。
2. Windows Explorer の右の枠に、「**ネットワーク プレースの追加**」という項目があるはずだ。これをダブルクリックする。

3. 場所を尋ねられたら、次のURLを入力する。

   ```
   http://127.0.0.1/dav-test/
   ```

   Windows Explorerを別のシステム上で実行しているのであれば、127.0.0.1の部分をmod_davをインストールしたサーバ名に置き換えればよい。

4. 「**次へ**」をクリックした後、この場所に好きな名前を付けるか、もしくは、デフォルトのままにしておく。

5. ダイアログの操作が完了すると、Windows Explorerはステップ4で指定した名前の付いた新しいウィンドウを開くはずだ。このディレクトリは空なので、このウィンドウも空になっているはずだ。

6. Windows Explorerのメインウィンドウで、ファイルがある任意のディレクトリ（どんなディレクトリでもよい）に移動する。

7. Windows Explorerのメインウィンドウからファイルを選んで、ステップ5で開いたウィンドウにコントロールキーを押しながらドラッグする。

8. Windowsは簡単なコピー経過のダイアログウィンドウを表示する。転送が完了すると、そのファイルが目的のウィンドウに現れるはずだ。

おめでとう！　WebDAVを使って、Webサーバにファイルをアップロードすることができた。テストが完了したら、htdocs\dav-testディレクトリと設定ファイルの`<Directory "C:/Apache-1.3/htdocs/dav-test">`ブロックを忘れずに削除しておくこと。さもないと、他の誰かがサーバにファイルをアップロードしてしまうおそれがある。

参照

- レシピ6.18
- http://webdav.org/mod_dav/

レシピ2.4　Unix系システムにmod_perlをインストールする

課題

Perlスクリプトのパフォーマンスを向上し、Webサーバに簡単に組み込めるようにするため、mod_perlスクリプトモジュールをインストールしたい。

解決

Apache 1.3を使っているのであれば、http://perl.apache.org/からmod_perlのソースパッケージをダウンロードして、展開する。次に、以下のコマンドを実行する。

```
% perl Makefile.PL \
>     USE_APXS=1 \
>     WITH_APXS=/usr/local/apache/bin/apxs \
>     EVERYTHING=1 \
>     PERL_USELARGEFILES=0
% make
% make install
```

そして、サーバを再起動する。

Apache 2.0 以降を使っている場合も、同様の処理をすればよい。mod_perl 2.0 のソースパッケージをダウンロードして、展開する。それから、次のコマンドを実行する。

```
% perl Makefile.PL MP_APXS=/usr/local/apache2/bin/apxs
```

解説

mod_perlソースパッケージはかなり複雑なモジュールであり、サーバに組み込む方法がいくつかある。このレシピで解説しているのは、最も速く、影響も少ない方法だ。もしやりたいことと合わなければ、パッケージ展開後のディレクトリにある、各種 README.* ファイルを調べるとよい。基本となる言語が C ではなく Perl であるため、インストール手順は他のモジュールとはかなり異なっている。

サーバをうまく再起動することができれば、mod_perlが利用可能になり、サーバの一部として設定されるはずだ。サーバがうまく動いているかどうか調べるには、httpd.confファイルを少し変更して、スクリプトを追加すればよい。mod_perl 操作の簡単なテスト手順を以下に示す。

1. mod_perl スクリプトを置くディレクトリを作成する。

   ```
   # cd ServerRoot
   # mkdir lib lib/perl lib/perl/Apache
   ```

2. mod_perl の起動方法を指示するために、サーバの conf/ ディレクトリに startup.pl という名前のファイルを作成する。

   ```
   #!/usr/bin/perl
   BEGIN {
       use Apache ( );
       use lib Apache->server_root_relative('lib/perl');
   }
   use Apache::Registry ( );
   use Apache::Constants ( );
   use CGI qw(-compile :all);
   use CGI::Carp ( );
   1;
   ```

3. 次に、テストに使う lib/perl/Apache/HelloWorld.pm ファイルを作成する。

```
package Apache::HelloWorld;
use strict;
use Apache::Constants qw(:common);
sub handler {
    my $r = shift;
    $r->content_type('text/plain; charset=ISO-8859-1');
    $r->send_http_header;
    $r->print("Hello, world!Love, mod_perl.\n");
    return OK;
}
1;
```

4. 続いて、サーバの設定ファイルを編集して、mod_perl が必要な場所に置けるようディレクティブを追加し、テストスクリプトを呼び出すよう指示する。httpd.confファイルに、次の行を追加すればよい。

```
<IfModule mod_perl.c>
    PerlRequire conf/startup.pl
    <Location /mod_perl/howdy>
        SetHandler perl-script
        PerlHandler Apache::HelloWorld
    </Location>
</IfModule>
```

5. さあ、サーバを再起動して、http://localhost/mod_perl/howdyにアクセスし、このスクリプトをリクエストしてみよう。正しく設定できていれば、単に "Hello, world!　Love, mod_perl." という言葉を含むページが返ってくるはずだ。

参照

- http://perl.apache.org
- 『Apache拡張ガイド(上)サーバサイドプログラミング』、『Apache拡張ガイド(下)APIリファレンス』(オライリー・ジャパン発行、原書『Writing Apache Modules with Perl and C』、Doug MacEachern、Lincoln Stein 著、O'Reilly Media 発行)
- 『mod_perl Developer's Cookbook』、Geoffrey Young、Paul Lindner、Randy Kobes 著、Sams 発行

レシピ2.5　Unix系システムにmod_phpをインストールする

課題

既存の Apache Web サーバに、mod_php スクリプトモジュールを追加したい。

解決

Webサイト http://php.net/ から mod_phpのソースパッケージをダウンロードし(ダウンロードのリンクをたどればよい)、展開する。次に、以下を実行する。

```
% cd php-5.2.3
% ./configure \
>     --with-apxs2=/usr/local/apache/bin/apxs
% make
# make install
```

サーバを再起動する。

解説

正しくインストールできたか確認するために、サーバの DocumentRoot に info.php という名前のファイルを作る。このファイルには次の 1 行を書いておく。

```
<?php phpinfo(); ?>
```

httpd.conf ファイルに次の行を追加しよう。

```
<IfModule mod_php5.c>
    AddHandler application/x-httpd-php .php
</IfModule>
```

サーバを再起動して、ブラウザでinfo.phpドキュメントが取得できるかどうか試してみよう。現在アクティブになっているPHPオプションについて、詳しい説明が表示されるはずだ。もし表示されていれば、インストールは成功している。テストが終わったら、info.php ファイルを削除しておこう。PHP では、たくさんの追加オプションや拡張が利用できる。ここで紹介したレシピは、最も基本的なインストールについてのみ解説した。

参照

- レシピ 8.16
- レシピ 8.17
- http://php.net/

レシピ 2.6 Windows に mod_php をインストールする

課題

Windows 上の既存の Apache Web サーバに、mod_php スクリプトモジュールを追加したい。

解決

このレシピでは、実行すべきコマンドを明示するのではなく、手順について大まかに解説しておく†。

1. http://php.net/ から、API 拡張を含む PHP の Windows バイナリ向け .zip ファイルをダウンロードする（.exe ファイルではないことに注意する）。
2. 今後もコンテンツを置いておけるディレクトリ（例えば、C:\PHP4）に.zipファイルを展開する。WinZip を使っているなら、[Use folder names]チェックボックスを選択しておくこと。こうすると、.zipファイル中のディレクトリ構造が保持される。
3. PHP4\SAPI\php4apache.dll ファイルを、Apache の ServerRoot の下にある \modules\ ディレクトリにコピーする。
4. コマンドプロンプトウィンドウで、.zipファイルを展開したPHP4ディレクトリに移動し、以下のコマンドを実行する。

```
...\PHP4>copy php.ini-dist %SYSTEMROOT%\php.ini
...\PHP4>copy php4ts.dll %SYSTEMROOT%
```

（Windows 95 もしくは Windows 98 にインストールする場合には、%SYSTEMROOT% ではなく %WINDOWS% を使うこと）

5. %SYSTEMROOT%\php.ini ファイルを編集する。extensions_dir で始まる行を探して PHP4\extensions ディレクトリを指すように変更する。例えば、.zipファイルをC:\PHP4に展開した場合には、この行は次のようになる。

```
extensions_dir = C:\PHP4\extensions
```

6. Apache の ServerRoot にある conf\httpd.conf ファイルを編集する。他の LoadModule 行の近くに次の行を追加する。

```
LoadModule php4_module modules/php4apache.dll
```

そして、.php ファイルに適用できる範囲のどこかに、次の行を追加する。

```
<IfModule mod_php4.c>
    AddType application/x-httpd-php .php
</IfModule>
```

7. サーバを再起動する。これで、PHP モジュールが利用可能になるはずだ。

† 訳注：このレシピは、PHP4 をインストールする場合について説明している。PHP5 の場合には、MSI パッケージを使ってインストールすることができる。詳細は http://php.net/ を参照してほしい。

解説

WindowsにPHPモジュールをインストールするにはたくさんの細々とした手作業が必要だ。正しくインストールできたか確認するために、サーバのDocumentRootにinfo.phpという名前のファイルを作る。このファイルには次の1行を書いておく。

```
<?php phpinfo(); ?>
```

サーバを再起動して、ブラウザでinfo.phpドキュメントが取得できるかどうか試してみよう。現在有効になっているPHPオプションに関する詳しい説明が表示されるはずだ。PHPでは、たくさんの追加オプションや拡張が利用できる。ここで取り上げたレシピは最も基本的なインストールについてのみ解説した。もっと詳しく知りたければ、PHP4ディレクトリのinstall.txtファイルやWebサイトにあるドキュメントを参照すること。

参照

- http://php.net/

レシピ 2.7 mod_ssl をインストールする

課題

mod_ssl セキュア HTTP モジュールを使って、Apache サーバに SSL サポートを追加したい。

解決

Windows

WindowsへのSSLのインストールについては、レシピ7.2で解説しているので、ここでは簡単に述べておく。Microsoft Windows プラットフォームでソースコードをビルドした経験があまりないなら、ApacheFriends.org から XAMPP を取得するのがよいだろう。

Apache 2.0

mod_sslはApache 2.0に含まれているが、ソースからビルドする際に、自動的にコンパイル、インストールが行われるわけではない。./configureを実行するときに、--enable-sslオプションを追加する必要がある。また、設定ファイルで、LoadModuleディレクティブを使って、mod_sslを利用可能にする必要がある。

Apache 1.3

Unix系システムにmod_sslをインストールするには、Webサイトhttp://www.modssl.org/からtarballパッケージをダウンロードし、展開する。次に、以下を実行する。

```
% cd mod_ssl-2.8.14-1.3.27
% ./configure \
>    --with-apache=../apache_1.3.27 \
>    --with-ssl=SYSTEM \
>    --prefix=/usr/local/apache
% cd ../apache_1.3.27
% make
% make certificate
```

解説

mod_sslパッケージは、ベースとなるApacheのソースコードに変更を加える必要があり、インストールしたいmod_sslパッケージのバージョンと現在動かしているApacheディストリビューションのバージョンがマッチしていなければならない。Apacheのインストールにソースが含まれていない場合(例えば、バイナリのRPMやその他のベンダのディストリビューションをインストールした場合など)には、mod_sslを追加することはできないだろう。

mod_sslを使うには、Apacheのソースだけではなく、PerlとOpenSSLライブラリもインストールしておく必要がある。ビルド時の設定パラメータに`--with-ssl`オプションを付けて、OpenSSLがインストールされている場所を指定する。ベンダのディストリビューションのディレクトリにあるなら、特別なキーワード`SYSTEM`を指定すると、自動的に設定してくれるので、自分で探す必要はない。

他のApacheモジュールと違って、mod_sslを追加するときには、Apacheのソースディレクトリにある`configure`スクリプトではなく、mod_sslディレクトリにある`./configure`スクリプトを実行することに注意しよう。mod_sslの`configure`スクリプトは、Apacheのディレクトリに移動して、`configure`スクリプトを直接実行するようになっている。

このレシピはほんの基本にすぎない。mod_sslにはたくさんの追加コンポーネントや機能があり、設定時に指定することができる。詳しくは、mod_sslのソースディレクトリにあるREADMEファイルやINSTALLファイル、mod_sslのWebサイトhttp://www.modssl.org/を参照すること。

参照

- レシピ7.3
- http://www.modssl.org/

レシピ2.8　modules.apache.orgでモジュールを探す

課題

ある機能を持ったApacheモジュール、または、名前がわかっているApacheモジュールを探したい。Apache Module Registryというものがあると聞いたことがある。

解決

http://modules.apache.org/にアクセスして、ほしい機能に関連するキーワードやモジュール名の一部を

使って検索すればよい。

解説

Apache Module Registry とは、簡単にモジュールのありかがわかるよう、モジュールの作者がボランティアで登録している非公式のサイトだ。

すべてのサードパーティ製モジュールがこのサイトに登録されているわけではない。モジュールはたいてい、SourceForgeや作者自身のホームページに置かれている。http://modules.apache.org を探しても見つからなければ、SourceForge(http://sourceforge.net)やFreshMeat(http://freshmeat.net)を探してみるか、Googleなどの検索エンジンを使ってWeb検索してみよう。

参照

- http://sourceforge.net
- http://freshmeat.net

レシピ 2.9 mod_security をインストールする

課題

簡単で強力なフィルタリングの仕組みを利用するために、mod_securityモジュールをインストールしたい。

解決

1. http://modsecurity.org/download から mod_security とコアルールをダウンロードする。

ダウンロードしたら、ファイルが改ざんされていないか、PGPシグニチャを確認すべきだ。詳しくは、mod_securityのWebサイトを参照すること。

2. そのキット(コアルールではなく)を作業ディレクトリに展開する。

```
% cd /usr/local/build
% tar xzf /tmp/modsecurity-apache_2.1.1.tar.gz
```

3. 展開したディレクトリに移動して、付属の Makefile を使ってパッケージをビルドする。make のコマンドラインには、以下のように ServerRoot の値を指定しておく。

```
% cd /usr/local/build/modsecurity-apache_2.1.1/apache2
% make top_dir=/usr/local/apache2
# make top_dir=/usr/local/apache2 install
```

他のサードパーティ製モジュールと違って、mod_securityは単にApacheのapxsツールを呼び出すのではなく、このモジュールが提供する仕組みを使ってビルドしなければならない。

4. コアルールを ServerRoot のサブディレクトリに展開する。

```
# cd /usr/local/apache2/conf
# mkdir mod_security
# cd mod_security
# tar xzf /tmp/modsecurity-core-rules_2.1-1.4.tar.gz
```

5. httpd.conf ファイルを編集して、適切な場所に次の行を追加する。

```
LoadModule security_module modules/mod_security2.so
Include conf/mod_security/*.conf
```

6. サーバを再起動する。

解説

mod_securityパッケージに付属しているMakefileは、モジュールをビルドして、正しい場所にインストールするだけだ。この機能を有効にするには、自分で設定する必要がある。最近のバージョンには、blogスパムやよくあるアタックに対処するためのコアルールが含まれている。ルールは別のtarballとしても利用可能で、ソフトウェア本体に組み込まれているものよりも頻繁に更新されている。

mod_securityの現在のバージョンは、Apache Webサーバのバージョン2だけをサポートしている。1.3バージョンをサポートする古いバージョンもあるが、今後保守される見込みはない。

参照

- mod_security の Web サイト http://modsecurity.org

レシピ 2.10　なぜモジュールがうまく動かないのか

課題

サードパーティ製モジュールをインストールしようとしたが、Apache Web サーバがモジュールを認識してくれない。

解決

そのパッケージが Apache のどのバージョンに対応しているのかをはっきりさせるために、モジュールのソースコードやドキュメントを調べよう。それでもわからなければ、作者に尋ねてみよう。

解説

Apache Webサーバに大きな変更があり、APIが変更されて互換性がなくなることがある。できるだけこう

した事態が起こらないよう多大な労力が払われているが、どうしても避けられない場合もある。

Webサーバが、互換性のないモジュールをロードしてクラッシュしてしまうのを避けるために、サーバは「マジックナンバ」を組み込んでいる。この番号はビルド時に記録され、APIのバージョンと関連付けられている。サーバがDSOモジュールをロードしようとすると、モジュールのマジックナンバとサーバのマジックナンバを比べて、互換性がなければモジュールをロードしない仕組みになっている。

開発チームは、メジャーバージョン番号をマジックナンバにして互換性を維持しようとしているが、完全ではない。Apache 1.3用にビルドしたモジュールは、その後にビルドしたバージョン1.3のサーバでもたいてい動くはずだが、2.0のサーバでは間違いなく動かない。逆に、2.0用のモジュールは、1.3のサーバでは動かないだろう。

参照

- Apache Module Registry―― http://modules.apache.org

3章 ログの記録

Apacheは処理するすべてのリクエスト情報を記録する機能を備えており、通常は記録するようになっている。ログの記録方法をコントロールして後でログから役に立つ情報を抽出するのは、初めて訪れた土地で情報を集めるのと同じくらい、とても重要なことだ。

Apacheはログファイルに2種類のデータを記録している。1つは、リクエストそのものに関する情報であり、もう1つは、リクエスト処理中に発生した異常状態(例えば、ファイルのパーミッションについてなど)に関するメッセージだ。Webマスターは、エラーに関するログについてはあまりコントロールできないが、リクエスト処理に関するログ(動作ログ)についてはそのフォーマットや情報量をかなりコントロールできる。サーバはリクエストの動作状況を、複数のフォーマットで複数のログファイルに記録することができるが、エラーメッセージのコピーを記録しているにすぎない。

動作ログについて知っておくべきことは、リクエストの処理が完了した「後で」、ログエントリをフォーマットして書き込むという点だ。リクエストの開始時刻から終了時刻までの間隔が長くなると、影響が出てくるおそれがある、ということを意味している。

例えば、ログファイルをローテーションしている場合、非常に大きなファイルをダウンロードすると、そのリクエストのログエントリが、リクエストを開始した時点のログファイルではなく、リクエストが完了した時点の新しいログファイルに書き込まれてしまうことがある。これとは対照的に、エラーメッセージはエラーが発生した時点でエラーログに書き込まれる。

Webサーバは、動作している限りログファイルに情報を記録し続ける。したがって、人気のあるサイトではログファイルが巨大になってしまう。それほど人気のないサイトであっても、嫌になるくらい大きくなることがある。ファイルサイズが大きくなるのを防ぐために、たいていのサイトでは、定期的にログファイルをローテーションしている。ログファイルをローテーションするとは、簡単に言うと、現在のファイルに書き込むのをやめて新しいファイルに書き込むということだ。Apacheは記録が失われないよう動作しているので、定めた時刻表通りにうまくローテーションさせるには、少々作業が必要になる。本章では、この作業を確実に成功させるためのレシピをいくつか紹介している(レシピ3.8とレシピ3.9を参照)。

ログ宣言のCustomLogディレクティブとErrorLogディレクティブは、<VirtualHost>コンテナの内部に置いてもよいし、外部(メインサーバやグローバルサーバ、グローバルスコープと呼ぶこともある)に置いてもよいし、両方に置いてもよい。ログエントリはこのどちらかにだけ記録される。<VirtualHost>コンテナ内にログ用ディレクティブを置いた場合、このコンテナで処理するリクエストやエラーに関するメッセージは、この

ログ用ディレクティブで指定したログファイルに書き込まれ、グローバルに宣言されたログファイルには書き込まれない。逆に、<VirtualHost>コンテナ内にログ用ディレクティブを置いていない場合には、グローバルなディレクティブの設定に基づいてログエントリが記録される。

ログ用ディレクティブがどのスコープに適用されたとしても、そのスコープ内にあるすべてのCustomLogディレクティブは個別に扱われる。つまり、グローバルスコープに1つ、<VirtualHost>コンテナの中に2つのCustomLogディレクティブがある場合、両方のディレクティブが使われる。同様に、一方のCustomLogディレクティブでenv=オプションを使っている場合も、同じスコープにあるもう1つのCustomLogディレクティブがどのリクエストをログに記録するかには影響を与えない。

動作ログはWebが世の中に出現したときから存在しており、初期のユーザがどの情報をログとして記録するかを決めるのには、それほど時間がかからなかった。この結果、CLF（Common Log Format）と呼ばれるcommonログフォーマットができた。Apacheの表記法で書くと、次のようなフォーマットになる。

```
"%h %l %u %t \"%r\" %>s %b"
```

これは、次に述べる項目をログに記録することを意味している。クライアントのホスト名もしくはIPアドレス(%h)、クライアントのユーザ名(RFC 1413で定義されているもの、ApacheでIdentifyCheck Onディレクティブを使って監視するよう設定している場合)(%l)、クライアントを認証するのに使うユーザ名(サーバが弱いアクセス制御を課している場合)(%u)、リクエストの受信時刻(%t)、実際のHTTPリクエスト行(%r)、サーバがリクエストを処理した後の最終的なステータス(%s)、サーバのレスポンスとして送信したコンテンツのバイト数(%b)。

HTTPプロトコルが進化するにつれて、CLFでは不十分だということがわかり、機能強化したcombinedログフォーマットが作られた。

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

このフォーマットでは、Referer(綴りに注意。仕様でも間違った綴りが使われている)とUser-agentという2つの項目が追加された。Refererとは、リクエストしたドキュメントにリンクしていたページのURLを示している。User-agentとは、リクエストを発行したブラウザやクライアントソフトウェアの名前とバージョンを示している。

この2つのフォーマットはどちらも広く利用されており、ログ解析ツールの多くは、ログエントリがこのどちらかのフォーマットで書かれていることを想定している。

Apache Webサーバ標準の動作ログモジュールを使うと、ユーザ独自のフォーマットを作ることができる。このモジュールはいろいろな設定ができるようになっており、mod_log_configという名前が付いている(!)。Apache 2.0には、mod_logioというモジュールも追加された。これは、mod_log_configを強化して、ネットワーク上で実際に送受信されたバイト数をログに記録する機能を備えたものだ。これでも要求に合わなければ、非常に多くのサードパーティ製モジュールがあるので、http://modules.apache.org/を探してみるとよい。

commonログフォーマットおよびcombinedログフォーマットにおけるステータスコードについては、意味がちょっとわかりにくいので、説明しておく必要があるだろう。ステータスコードはHTTPプロトコル仕様ドキュメント(現時点では、RFC 2616であり、ftp://ftp.isi.edu/in-notes/rfc2616.txtから入手可能)で定義されて

いる。表3-1は、本書の執筆時点でHTTP仕様に定義されているステータスコードを簡単に説明したものだ。別の仕様(例えば、WebDAV)でも追加のステータスコードを定義しているが、ここでは取り上げない。もっと高度で、たくさんあるためだ。

表 3-1　HTTP ステータスコード

コード	定義	意味
情報 1xx		
100	Continue	リクエストを継続可能
101	Switching protocols	プロトコルの切り替え
成功 2xx		
200	OK	成功
201	Created	リソース作成に成功
202	Accepted	リクエストを受理
203	Nonauthoritative information	信頼できない情報
204	No content	コンテンツがない
205	Reset content	コンテンツのリセット
206	Partial content	部分コンテンツ
リダイレクト 3xx		
300	Multiple choices	複数の選択肢
301	Moved permanently	恒久的に移動
302	Found	一時的に他の場所に移動
303	See other	他の場所を参照
304	Not modified	更新されていない
305	Use proxy	プロキシを使用
306	(Unused)	(未使用)
307	Temporary redirect	一時的なリダイレクト
クライアントエラー 4xx		
400	Bad request	不正なリクエスト
401	Unauthorized	認証が必要
402	Payment required	支払いが必要
403	Forbidden	アクセス禁止
404	Not found	リソースが見つからない
405	Method not allowed	許可されていないメソッド
406	Not acceptable	受理できない
407	Proxy authentication required	プロキシの認証が必要
408	Request timeout	リクエストがタイムアウト
409	Conflict	矛盾が発生
410	Gone	存在しない
411	Length required	Content-Length が必要
412	Precondition failed	前提条件で失敗
413	Request entity too large	リクエストエンティティが大きすぎる
414	Request-URI too long	リクエスト URI が長すぎる
415	Unsupported media type	サポートしていないメディアタイプ
416	Requested range not satisfiable	リクエストしたレンジが範囲外
417	Expectation failed	Expect された処理に失敗

表 3-1 HTTP ステータスコード(続き)

コード	定義	意味
サーバエラー 5xx		
500	Internal server error	サーバ内部のエラー
501	Not implemented	未実装
502	Bad gateway	不正なゲートウェイ
503	Service unavailable	サービスが利用不能
504	Gateway timeout	ゲートウェイのタイムアウト
505	HTTP version not supported	サポートしていない HTTP バージョン

表3-1に示した一行の要約は、あまりに簡潔で混乱してしまうかもしれないが、少なくともサーバが何が起こったと考えているか、ヒントを与えてくれる。最初の数字は、ステータスコードをクラスやカテゴリに分類するのに使われている。例えば、5で始まるステータスコードはすべて、リクエストの処理中に何か問題が発生し、サーバはその原因がクライアント側ではなくサーバ側にあると考えている、ということを示している。

ステータスコードについて詳しく知りたければ、HTTPプロトコルに関するドキュメントやRFC自体を読む必要がある。

レシピ 3.1 ログエントリにもっと詳しい情報を記録する

課題

アクセスログのエントリに、もう少し詳しい情報を追加したい。

解決

common ログフォーマットではなく、combined ログフォーマットを使えばよい。

```
CustomLog logs/access_log combined
```

解説

Apacheはデフォルトでは、commonログフォーマットでログを記録する。しかし、LogFormatディレクティブとして combined ログフォーマットが定義されており、これを利用することもできる。

combinedログフォーマットは、commonログフォーマットに含まれていない2つの追加情報を記録することができる。1つはReferer(クライアントがどこからリンクをたどってきたか)であり、もう1つがUser-agent(ユーザがどんなブラウザを使っているか)である。

主要なログファイル解析ソフトウェアはすべて、commonログフォーマットだけではなくcombinedログフォーマットも扱うことができる。また、追加されたフィールドを使った統計情報を提供してくれるソフトウェアも多い。combined ログフォーマットを使うことで失うものはなく、追加の情報を得ることができる。

参照

- http://httpd.apache.org/docs/2.0/mod/mod_log_config.html
- http://httpd.apache.org/docs/2.2/mod/mod_log_config.html

レシピ 3.2　もっと詳しいエラーを取得する

課題

問題をデバッグするために、エラーログにもっと詳しい情報がほしい。

解決

httpd.confファイルのLogLevel行を変更(もしくは、追加)する。後述するパラメータを指定することができる。

例えば、次のようにすればよい。

```
LogLevel Debug
```

解説

エラーログのレベルはいくつかの階層に分かれており、キーワードで指定することができる。LogLevelのデフォルト値はwarnになっている。以下に、重要度の高いものから順に、指定可能な値を示した。

emerg
: 緊急事態。Web サーバが利用できない状態

alert
: すぐに対処が必要

crit
: 致命的な状態

error
: エラー状態

warn
: 警告

notice
: 正常だが、重要な情報

info
: 追加情報

debug
　デバッグ用のメッセージ

emerg を指定すると最小限の情報しか記録されず、debug を指定すると最大限の情報が記録されることになる。debugレベルにすると、調査したい問題とは無関係の情報までたくさん記録されてしまうので、問題が解決したら以前の設定に戻すのがよいだろう。

ログの記録レベルは階層的な性質を備えている。つまり、あるレベルを指定すると、それより重大度の高いレベルの情報がすべて記録される。唯一の例外は、noticeレベルのメッセージは、LogLevelディレクティブの設定とは無関係に常に記録されるというところだ。

重大度のレベルは、かなり大雑把に定義されており、その運用もさらに大雑把だ。言い換えると、あるエラー状態をログに記録するとき、その重大度を何にするかは、コードを書いた開発者の判断で決まる。つまり、人によって判断が変わってくるということになる。

以下に、さまざまな重大度のメッセージ例をいくつか紹介する。

```
[Thu Apr 18 01:37:40 2002] [alert] [client 64.152.75.26] /home/smith/public_html/
    test/.htaccess: Invalid command 'Test', perhaps mis-spelled or defined by a
    module not included in the server configuration
[Thu Apr 25 22:21:58 2002] [error] PHP Fatal error:  Call to undefined function:
    decode_url(  ) in /usr/apache/htdocs/foo.php on line 8
[Mon Apr 15 09:31:37 2002] [warn] pid file /usr/apache/logs/httpd.pid overwritten --
    Unclean shutdown of previous Apache run?[Mon Apr 15 09:31:38 2002] [info] Server built: Apr 12
2002 09:14:06
[Mon Apr 15 09:31:38 2002] [notice] Accept mutex: sysvsem (Default: sysvsem)
```

これらは、運用中のWebサーバでよく見かける、ごく普通のメッセージだ。ログレベルを debug に設定すると、次のような意味不明のメッセージを見かけることがあるだろう。

```
[Thu Mar 28 10:29:50 2002] [debug] proxy_cache.c(992): No CacheRoot, so no caching.
    Declining.
[Thu Mar 28 10:29:50 2002] [debug] proxy_http.c(540): Content-Type: text/html
```

実際に、このようなメッセージが記録される。これらのデバッグメッセージは、Apache開発者が、proxyモジュールの実行状況を理解するために入れたものだ。

参照

本書の執筆中にも、Apacheのエラーメッセージの辞書を作ろうという試みが進んでおり、エラーがどんな意味で、どのように対処すればよいかをまとめようとしている。しかし、現時点では具体的なものはまだ出来上がっていない。完成すると、以下の Apache サーバ開発者のサイトにアナウンスされるはずだ。

http://httpd.apache.org/dev

また、本書の Web サイトでも紹介するつもりだ。

```
http://apache-cookbook.com
```

LogLevelディレクティブについて詳しく知りたければ、ApacheのWebサイトにあるドキュメントを参照すること。

```
http://httpd.apache.org/docs/2.2/mod/core.html#loglevel
```

レシピ 3.3　POST の内容をログに記録する

課題

Web フォームへの入力など、POST メソッドで投稿されたデータを記録したい。

解決

mod_dumpioがインストールされていて、それが有効になっているかどうか確認する。設定ファイルに、次のように追加すればよい。

```
# DumpIOLogLevel notice - 2.2.4 以降の場合
LogLevel debug
DumpIOInput On
```

または、mod_security を使う場合には、次のようにすればよい。

```
SecAuditLogType Concurrent
SecAuditLogStorageDir /var/www/audit_log/data/
SecAuditLog /var/www/audit_log/index
SecAuditLogParts ABCFHZ
```

解説

mod_dumpio は Apache 2.0（2.0.53 以降）で新しく追加されたモジュールだ（つまり、Apache 1.3 では利用できない）。これを使うと、HTTP トランザクションの入出力をすべてログに記録することができる。ここで紹介した例では、DumpIOInput ディレクティブを使って入力のログ記録のみを有効にした。

Apache 2.0 と 2.2（2.2.4 未満）では、LogLevel に debug を設定しておく必要がある。Apache 2.2.4 以降では、新しい DumpIOLogLevel というディレクティブが導入され、エントリを記録する LogLevel を設定することができる。例えば、DumpIOLogLevel を notice に設定しておくと、LogLevel を notice 以上に設定したときに記録されることになる。

POST データのためのログエントリは、次のようになる。

```
[Sun Feb 11 16:49:27 2007] [debug] mod_dumpio.c(51): mod_dumpio:

dumpio_in (data-HEAP): 11 bytes
[Sun Feb 11 16:49:27 2007] [debug] mod_dumpio.c(67): mod_dumpio:

dumpio_in (data-HEAP): foo=example
```

このログエントリでは、フォーム foo の値が example に設定されていることを示している。

mod_dumpio の出力は、非常にわかりにくい。1つのリクエストのログエントリは通常、30行から50行にもなる。ここに挙げたログエントリは、POST を記録したログのほんの一部にすぎない。

mod_securityを使っても、リクエストデータのログを記録することができる。このレシピに紹介したようにmod_securityを設定すると、利用可能なすべてのリクエストヘッダとリクエストボディをログファイルに記録することができる。

参照

- http://modsecurity.org
- http://httpd.apache.org/docs/2.2/mod/mod_dumpio.html

レシピ3.4 プロキシ経由のクライアントのIPアドレスをログに記録する

課題

クライアントがプロキシ経由でページをリクエストしたときにも、実際のクライアントのIPアドレスをログに記録したい。

解決

解決策はない。

解説

残念ながら、HTTPプロトコル自体が、これをできないようにしている。クライアント側から見ると、プロキシは完全に透過になっている。コンテンツを実際に格納しているサーバ側から見ると、プロキシは、リクエストの身元を隠してほとんどわからなくしてしまう。

取り得る最善の策は、リクエストの送信元のIPアドレスをログに記録することだ。ブラウザから直接やってきた場合には、これはクライアントのIPアドレスになる。1つ以上のプロキシ経由でやってきた場合には、実際にサーバと通信しているプロキシサーバのアドレスになるだろう。

combinedとcommonのいずれのログフォーマットにも書式制御文字%hが含まれている。この制御文字は、(リモート)クライアントの身元を表している。しかし、HostnameLookupsディレクティブの設定によっては、IPアドレスではなくホスト名になることがある。常にクライアントのIPアドレスをログファイルに記録したければ、代わりに書式制御文字 %a を使えばよい。

参照

- HTTP プロトコル仕様（RFC 2616）ftp://ftp.isi.edu/in-notes/rfc2616.txt

レシピ 3.5　クライアントの MAC アドレスをログに記録する

課題

サーバにアクセスしているクライアントのMAC（ハードウェア）アドレスを記録したい。

解決

たいていのネットワーク環境では、MAC アドレスを確実にログに記録することはできないし、Apache では全く記録できない。

解説

MACアドレスはLAN（ローカルエリアネットワーク）でしか意味がなく、広域のネットワーク通信においては利用できない。ネットワークパケットがルータを通過するとき、例えば、LANから外に出るときには、ルータは通常、MAC アドレスのフィールドをルータ自身のハードウェアアドレスに書き換えてしまう。

参照

- TCP/IP プロトコル仕様（http://www.rfc-editor.org/rfcsearch.html にアクセスして “TCP” で検索すればよい）

レシピ 3.6　Cookie をログに記録する

課題

クライアントがサーバに送ったすべてのCookieと、サーバがクライアントのデータベースに設定するよう依頼したすべてのCookieを記録したい。これはCookieを使ったWebアプリケーションをデバッグするのに役立つ。

解決

クライアントから受信した Cookie をログに記録するには、次のように指定する。

```
CustomLog logs/cookies_in.log "%{UNIQUE_ID}e %{Cookie}i"
CustomLog logs/cookies2_in.log "%{UNIQUE_ID}e %{Cookie2}i"
```

サーバが設定してクライアントに送った Cookie の値をログに記録するには、次のように指定する。

```
CustomLog logs/cookies_out.log "%{UNIQUE_ID}e %{Set-Cookie}o"
CustomLog logs/cookies2_out.log "%{UNIQUE_ID}e %{Set-Cookie2}o"
```

2.0.56より以前のバージョンでは、複数のCookieが含まれる(可能性がある)場合には、デバッグ用途で書式制御文字%{Set-Cookie}oを使うことは推奨されていない。これは、最初のCookieしかログファイルに記録されないためだ。例については、次の解説を参照すること。

解説

Cookieフィールドは、非常に長くて複雑になりがちで、先に紹介した設定では、Cookieのログが別々のファイルにまたがって記録されてしまうことがある。環境変数UNIQUE_IDを使うと、Cookieのログエントリを、クライアントからのリクエストのアクセスログに関連付けることができる(サーバでmod_unique_idが有効になっており、動作ログのフォーマットに書式制御文字%{UNIQUE_ID}eとして環境変数が含まれているものとする)。

本書の執筆時点では、ヘッダフィールドとしてCookieとSet-Cookieが最もよく使われている。Cookie2とそれに対応するSet-Cookie2ヘッダフィールドは、元の仕様の欠点を修正するために新しく作られたが、まだ広く普及してない。

Cookieヘッダフィールドの構文は、時が経つにつれ変化してきたため、このようにログを記録するように指示しても、Cookieを完全に記録できるとは限らない。

覚えておいてほしいのは、これらのログ用ディレクティブは、すべてのCookieを記録してしまうということだ。特に関心のあるCookieだけを記録してくれるわけではない。例えば、次に示したのはクライアントからのリクエストに対するログエントリであり、2つのCookieを含んでいる。1つはRFC2109-1という名前で、もう1つはFC2109-2という名前になっている。

```
PNCSUsCoF2UAACI3CZs RFC2109-1="This is an old-style cookie, with space characters
    embedded"; RFC2109-2=This_is_a_normal_old-style_cookie
```

ログエントリは1つだが、2つのCookieに関する情報が含まれている。

次に示したのはCookieを設定する場合の例であり、サーバがレスポンスヘッダに入れて送信したSet-Cookieヘッダフィールドを示している。

```
Set-Cookie: RFC2109-1="This is an old-style cookie, with space characters embedded";
    Version=1; Path=/; Max-Age=60; Comment="RFC2109 demonstration cookie"
Set-Cookie: RFC2109-2=This_is_a_normal_old-style_cookie; Version=1; Path=/; Max-
    Age=60; Comment="RFC2109 demonstration cookie"
```

そして、このレスポンスに対応するログエントリは、次のようになる(ログファイルには1行で書き込まれているが、ページに収めるために行を折り返してある)。

```
eCF1vsCoF2UAAHB1DMIAAAAA RFC2109-1=\"This is an old-style cookie, with space
    characters embedded\"; Version=1; Path=/; Max-Age=60; Comment=\"RFC2109
    demonstration cookie\", RFC2109-2=This_is_a_normal_old-style_cookie;
    Version=1; Path=/; Max-Age=60; Comment=\"RFC2109 demonstration cookie\"
```

Apache httpdの2.0.56より以前のバージョンは、複数のCookieを正しくログに記録できなかった。ログには 1 つの Cookie しか記録されないだろう。

参照

- RFC 2109『HTTP State Management Mechanism』(IETF による Cookie と Set-Cookie ヘッダフィールドの定義)ftp://ftp.isi.edu/in-notes/rfc2109.txt
- RFC 2965『HTTP State Management Mechanism』(IETF による Cookie2 と Set-Cookie2 ヘッダフィールドの定義) ftp://ftp.isi.edu/in-notes/rfc2965.txt
- Netscape 社による Cookie の原案 http://wp.netscape.com/newsref/std/cookie_spec.html

レシピ3.7　ローカルページからの画像のリクエストをログに記録しない

課題

自分のサイトにある画像へのリクエストをログに記録したいが、自分自身のページからリクエストした場合にはログに記録したくない。ログファイルのサイズを小さくしたい場合や、芸術作品を不正に利用しているサイトを見つけたい場合に役立つ。

解決

サイト外部からのリクエストだけをログに記録するには、SetEnvIfNoCase を使う。

```
<FilesMatch \.(jpg|gif|png)$>
    SetEnvIfNoCase Referer "^http://www.example.com/" local_referrer=1
</FilesMatch>
CustomLog logs/access_log combined env=!local_referrer
```

解説

Webサーバにあるドキュメントは、同じサーバに置いてある画像を参照していることが多い。しかし、ログを解析するときに本当に関心があるのは、ページ自体への参照である。ローカルページにアクセスされたときに発生する画像へのリクエストをログに記録しないようにするには、どうしたらよいだろうか？

SetEnvIfNoCase では、画像にリンクしているページが www.example.com というサイトであり、リクエストが GIF、PNG、もしくは、JPEG 画像に対するものであれば、環境変数 local_referrer を設定している。

SetEnvIfNoCaseはSetEnvIfと似ているが、変数を比較するときに大文字と小文字を区別しない点が違っている。

CustomLog ディレクティブでは、この環境変数 local_referrer が設定されていないリクエストをすべてログに記録する。つまり、自分のページからリンクされた画像へのリクエストを除いて、すべてのリクエストがログに記録されることになる。

このレシピがうまく機能するのは、参照しているページ情報(Referer)をクライアントが報告してくれる場

合だけだ。参照しているページのURLに関心がない人もいるし、この情報を含めるかどうかをユーザが選択できるようにしているクライアントもある。また、インターネット上には、代理として働いてこの情報を隠してしまう「匿名」サイトもある。

参照

- レシピ 6.5

レシピ 3.8 特定の時間にログファイルをローテーションする

課題

サーバのシャットダウンと再起動をせずに、Apache のログファイルを特定の時間に自動的にローテーションしたい。

解決

CustomLog ディレクティブと rotatelogs プログラムを使って、次のようにすればよい。

```
CustomLog "| /path/to/rotatelogs /path/to/logs/access_log.%Y-%m-%d 86400" combined
```

解説

rotatelogsスクリプトは、パイプ経由のログと呼ばれるApacheの機能を使っている。パイプ経由のログというのは、ログ出力をファイルではなく別のプログラムに送る機能のことだ。rotatelogs スクリプトを Web サーバと実際のディスク上のログファイルの間に入れると、サーバを再起動せずに新しいファイルを作ることができる。このスクリプトは、決まった時間に新しいファイルを作り、そのファイルにログを書き始めてくれる。

rotatelogsスクリプトの最初の引数は、ログを記録するファイルのベース名を示している。その名前に1つ以上の%文字が含まれていると、strftime(3)のフォーマット文字列として扱われる。さもなければ、ローテーション時刻(1970年1月1日からの秒数)を10桁の数字で表した文字列がベース名の後に付く。例えば、ベース名がfooのときのログファイル名はfoo.1020297600のようになる。これに対し、ベース名がfoo.%Y-%m-%dのときのログファイル名は、foo.2000-04-29 のようになる。

2番目の引数は、ローテーション間隔(秒)を示している。システム時刻がこの値の倍数になるたびに、ログファイルはローテーションされる。例えば、1日24時間は86,400秒なので、ローテーション間隔を86400にしておくと、システム時刻で毎晩午前0時に新しいログファイルが作られる。システム時刻は1970年1月1日の午前0時を起点としており、この24時間の倍数が毎晩午前0時になるためだ。

ローテーション間隔は経過した実際のクロック秒数であり、夏時間のために時刻が変わってもローテーション間隔には影響を与えないということに注意しよう。

参照

- rotatelogs の manpage。次のコマンドで調べることができる。

```
% man -M ServerRoot/man rotatelogs.8
```

ServerRootは、httpd.confのServerRootディレクティブで実際に設定している値に置き換えること。オンラインドキュメントはhttp://httpd.apache.org/docs/2.2/programs/rotatelogs.htmlで見ることができる。

レシピ 3.9　毎月始めにログファイルをローテーションする

課題

毎月始めに、前月のログファイルをクローズして、新しいログファイルをオープンしたい。

解決

Apacheディストリビューションにはこれを実現するスクリプトは含まれていないが、このような役に立つ機能を提供してくれるフリーのプログラムがある。それはcronologというプログラムで、http://cronolog.orgから入手することができる。

cronolog を入手して、インストールする。そして、設定ファイルに次の行を追加すればよい。

```
CustomLog "|/usr/bin/cronolog /path/to/logs/access%Y%m.log" combined
```

解説

cronologには長い歴史があり、標準のrotatelogsユーティリティにはないが、みんながほしいと思っている機能をたくさん提供している。rotatelogsも年々改良されているが、cronologにはログファイルがすぐに大きくなるサイトにとって有用な機能がたくさんある。

その 1 つが、日や週、月、年で自動的にログファイルをローテーションする機能だ。ログファイルのファイル名のフォーマットは、CustomLog ディレクティブの指定に基づいている。

この例では、新しい月の初めにログファイルをローテーションする。ログファイル名には、年と月を表す変数(%Y と %m)が含まれている。

参照

- http://httpd.apache.org/docs/logs.html#piped
- http://httpd.apache.org/docs/2.2/programs/rotatelogs.html
- http://cronolog.org

レシピ 3.10 IP アドレスではなくホスト名をログに記録する

課題

動作ログには、IP アドレスではなくホスト名を記録したい。

解決

Apacheディレクティブで実行時の名前検索を解決にすると、Webサーバがリクエストを処理するときに、ホスト名を解決することができる。

```
HostnameLookups On
```

または、Apacheの通常処理ではIPアドレスを使い、パイプ経由のログ処理でエントリ記録の一部としてホスト名を解決するようにしてもよい。

```
HostnameLookups Off
CustomLog "| /path/to/logresolve -c >> /path/to/logs/access_log.resolved" combined
```

または、ApacheにはIPアドレスをログに記録させ、後からログファイルを解析してホスト名を解決してもよい。httpd.conf ファイルに次の行を追加する。

```
CustomLog /path/to/logs/access_log.raw combined
```

その後、ログファイルを次のコマンドで解析する。

```
% /path/to/logresolve -c < access_log.raw > access_log.resolved
```

解説

Apacheの動作ログの記録メカニズムでは、クライアントのIPアドレスかホスト名のどちらか(もしくは両方)を記録することができる。サーバがホスト名を直接ログに記録するには、時間をかけて、IPアドレスをホスト名に変換するためのDNS検索を実行する必要がある。これはサーバのパフォーマンスに深刻な影響を及ぼすおそれがある。IPアドレスをホスト名に変換するにはネームサービスに問い合わせる必要があり、この応答を待っている間、サーバの子プロセスやスレッドはクライアントのリクエストを処理できなくなるためだ。1つの代替案としては、サーバにはクライアントのIPアドレスだけを記録させ、ログファイルの後処理と解析の段階で、アドレスから名前に変換するのがよいだろう。少なくとも、Webサーバが名前解決のオーバヘッドの影響を直接受けないよう、別プロセスに処理を任せるべきだ。

これは理論的にはすばらしい案だ。しかし、実際にはいくつかの落とし穴がある。まず、Apacheに入っているlogresolveアプリケーション(通常、ServerRootのbinサブディレクトリにインストールされる)はログエントリの先頭に現れたIPアドレスしか名前解決できない。そのため、標準と異なるログファイルフォーマットを使うと、ほとんど対応できない。次に、IPアドレスを記録してから名前解決するまでにある程度時間が空いてしまうと、DNSが変わってしまって間違った結果になるおそれもある。これは、ISPによって動的に

割り当てられたIPアドレスを使っている場合には、特に問題になる。とはいえ、このように動的に割り当てられたIPアドレスのホスト名はもはや参考にならないことが多い。

パイプ経由でlogresolveに直接ログ記録を送っている場合には、さらに欠点がある。少なくともApache 1.3.24では、logresolveは出力バッファをすぐにフラッシュしないので、ログ処理やシステムがクラッシュするとデータが失われてしまうおそれがあるのだ。

しかし実際には、ログ解析ソフトウェアはすべて、名前の解決機能を備えている。ログ解析の前にログファイルのIPアドレスを名前解決するよりも、ログ解析ソフトの名前解決機能を使うのが最も妥当だろう。

参照

- logresolveのmanpage。次のコマンドで調べることができる。

```
% man -M ServerRoot/man/logresolve.8
```

- http://httpd.apache.org/docs/2.2/programs/logresolve.html

レシピ3.11　バーチャルホストごとに別のログで管理する

課題

バーチャルホストごとに動作ログを分けたい。しかし、複数のCustomLogディレクティブを使って、複数のログファイルをすべてオープンするということはしたくない。

解決

Apacheに付属するsplit-logfileプログラムを使えばよい。ローテーションされた後でログファイルを分割するには、次のコマンドを実行する(ServerRootは正しいパスで置き換えること)。

```
# cd ServerRoot
# mv logs/access_log logs/access_log.old
# bin/apachectl graceful
[古いログファイルが完全にクローズされるまで待機]
# cd logs
# ../bin/split-logfile < access_log.old
```

ログに記録するたびに適切なファイルに分けて記録するには、httpd.confファイルに次の行を追加する。

```
CustomLog "| /path/to/split-logfile /path/to/logs" combined
```

解説

split-logfileをうまく動かすためには、"%v"で始まるログフォーマットを使わなければならない(vの後に空白を入れることに注意)。こうすることにより、それぞれのログエントリの最初にバーチャルホスト名が入る。split-logfileはこのバーチャルホスト名を使って、そのエントリをどのファイルに記録すべきか判断す

る。このバーチャルホスト名は、書き込む直前に記録から削除される。

アクセスログファイルを分割するには、2つの方法がある。1つは、書き込み、クローズ、ローテーションされた後に分割する方法で、もう1つは、エントリが実際に記録されるときに分割する方法だ。クローズ済みのログファイルを分割するには、split-logfileスクリプトにログファイルを与えるだけでよい。エントリが実際に書き込まれるたびに別のファイルに分割して記録するには、ログメッセージが直接そのスクリプトにパイプされるよう、設定ファイルを変更すればよい。

それぞれの方法には利点もあれば欠点もある。ローテーションを利用する方法では、2倍のディスク容量(分割前のログと分割後のログのため)が必要になり、また、ログファイルが完全にクローズされたことを確認しなければならない。(残念ながら、実際にサーバをシャットダウンしたり、無作法に再起動したりする以外に、これを実現する簡単で確実な方法はない。通信速度が遅い場合には、行儀よく再起動した後かなり長い時間、古いログファイルがオープンされたままになることが多い)。エントリを記録するたびに分割する方法では、ログ処理が停止してしまうと大きな影響を受ける。Apacheは、自動的にログ処理を再起動するが、記録待ちのログメッセージが溜まってしまい、サーバが止まったようにみえることがある。

参照

- レシピ 3.10

レシピ 3.12 プロキシ経由のリクエストをログに記録する

課題

プロキシ経由のリクエストを、サーバに直接届いたリクエストとは別のログファイルに記録したい。

解決

SetEnv ディレクティブを使って、プロキシサーバ経由で届いたリクエストに目印を付ける。次のように設定すると、条件付きのログ記録を実現することができる。

```
<Directory proxy:*>
    SetEnv is_proxied 1
</Directory>
CustomLog logs/proxy_log combined env=is_proxied
```

また、バージョン 2.x では、<Proxy> ブロックを使うことができる。

```
<Proxy *>
    SetEnv is_proxied 1
</Proxy>
CustomLog logs/proxy_log combined env=is_proxied
```

解説

Apache 1.3には<Directory>ディレクティブに特別な構文があり、これを使うとプロキシモジュールを通して解釈されるリクエストにのみ、このディレクティブを適用することができる。"*"は、どんなドキュメントにも一致するワイルドカードのように見えるが、それは間違いで、実際にはワイルドカードではない。指定できるのは、proxy:http://example.com/foo.htmlのような完全なパスとマッチさせるか、すべてにマッチする"*" を使うかのどちらかである。proxy:http://example.com/*.html と指定することはできない。

プロキシ対象のパスごとに別々のディレクティブを適用したければ、他のモジュールを利用する必要がある。サーバが直接処理するリクエストではなく、サーバを通過するリクエストを扱う(つまり、サーバは配信サーバではなくプロキシになる)ので、プロキシ経由の特定のドキュメントに<Files>や<FilesMatch>コンテナを使うことはできない。また、<Location>や<LocationMatch>コンテナも使うことができない。これらは<Directory>コンテナの中に置くことができないためである。しかし、mod_rewrite の機能を使うと、リクエストされたドキュメントのパスに基づいて判断することができる。例えば、次のようにすると、プロキシ経由の画像へのリクエストを別のログファイルに記録することができる。

```
<Directory proxy:*>
    RewriteEngine On
    RewriteRule "\.(gif|png|jpg)$" "-" [ENV=proxied_image:1]
    RewriteCond "%{ENV:proxied_image}" "!1"
    RewriteRule "^" "-" [ENV=proxied_other:1]
</Directory>
CustomLog logs/proxy_image_log combined env=proxied_image
CustomLog logs/proxy_other_log combined env=proxied_other
```

<Directory proxy:*>コンテナ内のディレクティブは、サーバを通過したリクエストにのみ適用される。最初のRewriteRuleディレクティブは、リクエストされたドキュメントが.gif、.png、.jpgで終わるときに環境変数proxied_imageを設定する。RewriteCondディレクティブはこの環境変数をチェックし、設定されていなければ、次の RewriteRule で別の環境変数 proxied_other を設定する。2つの CustomLog ディレクティブは、設定されている環境変数に基づいて、それぞれのタイプのリクエストを別々のログファイルに記録する。

参照

- mod_rewrite と mod_log_config のドキュメント

レシピ3.13　バーチャルホストのエラーを別々のログファイルに記録する

課題

アクセスログとは違って、Apacheは1箇所にしかエラーを記録しない。特定のバーチャルホストに関するエラーを、グローバルなエラーログだけでなく、そのバーチャルホスト用のエラーログにも記録したい。

解決

少なくとも 2 つの方法がある。

1. パイプ経由のログを使って、エントリをカスタムスクリプトに送る。そのスクリプトがエラーメッセージをコピーして適切なファイルに書き込む。
2. パイプ経由のログを使って、ログエントリをコピーする。

```
ErrorLog "| tee logfile1 | tee logfile2 > logfile3"
```

解説

動作ログと違って、Apacheはエラーメッセージを1箇所にしか記録しない。エラーが特定のバーチャルホストに関するものであり、このバーチャルホストの<VirtualHost>コンテナにErrorLogディレクティブが指定されている場合、エラーはこのファイルにだけ記録される。つまり、グローバルなエラーログには記録されない。<VirtualHost>にErrorLogディレクティブが指定されていない場合、エラーはグローバルなエラーログにだけ記録される。(グローバルなエラーログとは、<VirtualHost>コンテナの外部に宣言した、最後に出現するErrorLogディレクティブに指定したファイルのことを指している。)

現在のところ、うまく動かす唯一の方法は、別のプロセスでコピーすることだ(つまり、パイプ経由のログを使って、エラーが発生するたびにエラーメッセージをそのプロセスに送る)。ここで紹介した2つの解決策のうち最初の解決策は、自分でカスタムスクリプトを作る必要があるが最も融通がきく。エントリをコピーするだけなら 2 番目の解決策の方が簡単だが、プラットフォーム上にteeプログラムが必要になる(通常、Windowsには存在しない)。teeプログラムがレコードを受け取った後にバッファをフラッシュしなかった場合、メッセージの記録に遅延が発生するおそれがある。そのため、パイプが壊れたりシステムがクラッシュしたりすると、そのメッセージが記録されずに失われてしまうことがある。

別のアプローチとしては、エラーログをsyslogに送り、syslogサーバが複数の場所にログエントリを記録するようにしてもよい。

参照

- http://httpd.apache.org/docs/logs.html#piped

レシピ 3.14 サーバの IP アドレスをログに記録する

課題

リクエストに応答したサーバのIPアドレスをログに記録したい。これは、複数のアドレスを持ったバーチャルホストがある場合に必要になる。

解決

LogFormat ディレクティブまたは CustomLog ディレクティブで、書式制御文字 %A を使う。

```
CustomLog logs/served-by.log "%A"
```

解説

書式制御文字%Aは、動作ログの該当する場所に、ローカルIPアドレス(つまり、サーバのアドレス)を入れることを表している。これはサーバが複数のIPアドレスを扱っているときに役に立つ。例えば、設定ファイルが次のようになっているとする。

```
Listen 10.0.0.42
Listen 192.168.19.243
Listen 223.41.0.80
<VirtualHost 192.168.19.243>
    ServerName Private.Example.Com
</VirtualHost>
<VirtualHost 10.0.0.42 223.41.0.80>
    ServerName Foo.Example.Com
    ServerAlias Bar.Example.Com
</VirtualHost>
```

この設定は、Foo.Example.Comにアクセスするのに、内部ユーザには10.0.0.42を使わせて、外部ユーザには公開している別のアドレス(223.41.0.80)を使わせたいという場合(例えば、ネットワークカード上で、外部からのトラフィックと内部からのトラフィックを分離したいときなど)に意味がある。2番目のバーチャルホストは、ServerNameは1つだが、2つのアドレス宛てのリクエストを受信することができる。ログフォーマットに書式制御文字%Aを使うと、このサイトへのアクセスがどちらのネットワークインターフェイスから届いたのか記録できるので、それぞれのアクセス回数を調べるのに役に立つ。

参照

- mod_log_config のドキュメント

レシピ 3.15　参照しているページをログに記録する

課題

自分のページを参照しているページのURLをログに記録したい。また、どうやってこのサイトにたどり着けいたのか、知りたいこともある。

解決

動作ログフォーマットに、次の書式制御文字を追加する。

```
%{Referer}i
```

解説

リクエストヘッダにRefererというフィールドが入っている場合がある。Refererとは、現在のリクエストにリンクしているページのURLを指している。例えば、ファイルa.htmlが次のようなリンクを含んでいるとする。

```
<a href="b.html">another page</a>
```

このリンクをたどると、b.htmlのリクエストヘッダには、a.htmlのURLを値としたRefererフィールドが含まれることになる。

Refererフィールドはリクエストヘッダに必須とされているわけではなく、その値は信頼できるものではない。どこからやって来たのか教えないソフトウェアや匿名ツールを好むユーザもいるが、こうしたユーザは非常に少ないので、たいていのWebサイトでは無視してしまってもよいだろう。

参照

- レシピ3.17
- レシピ6.5

レシピ3.16 ブラウザのソフトウェア名をログに記録する

課題

サイトにアクセスしてきた訪問者が、どんなソフトウェアを使っているのか知りたい。例えば、たくさんのユーザが使っているブラウザがわかれば、そのブラウザに合わせて見栄えを最適化することができる。

解決

動作ログフォーマットに、次の書式制御文字を追加する。

```
%{User-Agent}i
```

解説

たいていのリクエストヘッダには、User-agentと呼ばれるフィールドが含まれている。これは、リクエストするのに使われたクライアントソフトウェアの名前とバージョンを表している。例えば、User-agentフィールドは次のようになるだろう。

```
User-Agent: Mozilla/4.77 [en] (X11; U; Linux 2.4.4-4GB i686)
```

これは、クライアントがNetscape Navigator 4.77であって、Linuxシステム上で動いており、GUIシステムにはX-windowsを使用していることを表している。

User-agentフィールドはリクエストヘッダに必須とされているわけではなく、その値は信頼できるものでは

ない。何を使っているのかを教えないソフトウェアや匿名ツールを好むユーザもいる。User-agentを偽ることにより、特定のブラウザだけを対象としたサイトにアクセスできるようにしているソフトウェアもある。ユーザにはおかしな習慣があり、どのブラウザを好んでいるかはWebマスターには関係ないことだと考えている。そのため、できるだけブラウザはわからないものとして、サイトを設計するのがよいだろう。User-agentフィールドの値に基づいて何か判断しようとしているなら、それが偽造されていないと信じるしかない。いずれにせよ、偽造されていると知る方法もない。

参照

- レシピ 3.17

レシピ3.17　リクエストヘッダの任意のフィールドをログに記録する

課題

クライアントがリクエストヘッダとして送った任意のフィールドの値を記録したい。例えば、訪問者の需要に合わせて、コンテンツの種類を調整するのに役立つ。

解決

動作ログフォーマットの記述に、%{...}iログフォーマット変数を使う。例えば、Hostヘッダをログに記録するには、次のようにすればよい。

```
%{Host}i
```

解説

Webブラウザが送るHTTPリクエストは、かなり複雑だ。クライアントがブラウザではなく特別なアプリケーションの場合、サーバにとって意味のある追加のメタデータが入っていることがある。例えば、有用なリクエストヘッダフィールドの1つに、Acceptフィールドがある。Acceptフィールドは、クライアントがどんな種類のコンテンツを受信することができ、どれを受信したいのかをサーバに伝える。これを記録するには、次のような CustomLog を指定すればよい。

```
CustomLog logs/accept_log "\"%{Accept}i\""
```

実際に記録されるログエントリは、次のようになるだろう。

```
PNb6VsCoF2UAAH1dAUo "text/html, image/png, image/jpeg, image/gif,
    image/x-xbitmap, */*"
```

このリクエストを作ったクライアントは、HTMLと何種類かの画像を処理する用意はあるが、いざというときにはサーバが提供したものは何でも受け取る（ワイルドカード*/*エントリによって示されている）ということを伝えている。

参照

- レシピ 3.15
- レシピ 3.17

レシピ3.18 レスポンスヘッダの任意のフィールドをログに記録する

課題

サーバがレスポンスヘッダに入れた任意のフィールドの値を記録したい。これは、スクリプトやアプリケーションをデバッグするのに役立つ。

解決

動作ログフォーマットの記述に、%{...}oログフォーマット変数を使う。例えば、Last-Modifiedヘッダをログに記録するには、次のようにすればよい。

```
%{Last-Modified}o
```

解説

サーバの設定にもよるが、Apacheがリクエストに応答するときに送るHTTPレスポンスは、かなり複雑になることがある。高度なスクリプトやアプリケーションサーバは、サーバのレスポンスにカスタムフィールドを追加することがある。どんな値が設定されているかを調べることは、アプリケーションの問題を追跡するのに非常に役に立つ。

サーバが受信するフィールドではなく、サーバが送信するフィールドを記録するという点を除いて、このレシピはレシピ3.17とよく似ている。詳しくはレシピ3.17を参照すること。唯一の違いは、ログの書式制御文字の構文だ。レスポンスフィールドでは、制御文字oを使ってログに記録したが、リクエストフィールドでは制御文字 i を使ってログに記録する。

参照

- レシピ 3.17

レシピ 3.19 MySQL データベースに動作ログを記録する

課題

サーバへのアクセスをただのテキストファイルに記録するのではなく、データベースに直接記録して、もっと簡単に解析できるようにしたい。

解決

http://www.outoforder.cc/projects/apache/からmod_log_sqlの最新バージョンを入手し、モジュールの手順通りにインストールする(レシピ 2.1 を参照)。そして、次のコマンドを実行する。

```
# mysqladmin create apache_log
# mysql apache_log < access_log.sql
# mysql apache_log
mysql> grant insert,create on apache_log.* to webserver@localhost identified by 'wwwpw';
```

次に、httpd.conf ファイルに次の行を追加する。

```
<IfModule mod_log_sql.c>
    LogSQLLoginInfo mysql://webserver:wwwpw@dbmachine.example.com/apache_log
    LogSQLCreateTables on
</IfModule>
```

そして、VirtualHost コンテナに、ログ用ディレクティブを追加する。

```
LogSQLTransferLogTable access_log
```

解説

実際に、これらのコマンドを実行するときには、*webserver*や*wwwpw*の値を推測しにくいユーザ名とパスワードで置き換える必要がある。参照で紹介したWebサイトにあるドキュメントを調べよう。このモジュールの2.0リリースでは、設定の構文が変わってしまっているので、ここに紹介した例がインストールしたモジュールのバージョンで動くかどうか確認すること。

参照

- http://www.outoforder.cc/projects/apache/mod_log_sql

レシピ 3.20　syslog にログを記録する

課題

ログエントリを syslog に送って記録したい。

解決

エラーログを syslog に記録するには、次のように、Apache に syslog に記録するよう指示するだけでよい。

```
ErrorLog syslog:user
```

syslogのレポートクラスには、user以外にもいくつかある。環境によっては、例えば、local1の方が適していることもある。

アクセスログを syslog に記録するには、少し作業が必要だ。設定ファイルに次の行を追加する。

```
CustomLog |/usr/local/apache/bin/apache_syslog combined
```

apache_syslog は次のようなプログラムになる。

```
#!/usr/bin/perl
use Sys::Syslog qw( :DEFAULT setlogsock );

setlogsock('unix');
openlog('apache', 'cons', 'pid', 'user');

while ($log = <STDIN>) {
    syslog('notice', $log);
}

closelog;
```

解説

syslogにログを記録するのには、説得力のある理由がいくつかある。1番目の理由は、たくさんのサーバのログを集中管理できることだ。2番目の理由は、syslogを監視して、ある特定のイベントが発生したときに適切な通知を送ってくれるようなツールがたくさんあることだ。Apacheでこうしたツールを利用すると、さまざまな環境で役立つだろう。また、サーバに障害が発生したり、何か突発的な故障が発生したときには、物理的に他のマシンのログに記録するようにすると、サーバに何が起こったかわかり非常に役に立つ。

Apache は、syslog へのエラーログの記録をデフォルトでサポートしている。syslog を使うと非常に役に立つログになる。syslogは、単なる情報としてのメッセージではなく、エラー状態を追跡するのに使われることが多いためだ。

ErrorLog ディレクティブの構文では、引数として syslog を指定することができ、また、特定の syslog ファシリティを指定することができる。この例では、syslog ファシリティとして user を指定している。/etc/syslog.conf ファイルには、特定のログをどのファシリティに送るか、ファイルやリモートの syslog サーバなどを指定することができる。

Apacheは、syslogへのアクセスログの記録をデフォルトではサポートしていない。そのため、パイプ経由のログディレクティブを使う必要がある。これを実現するためには、Sys::Syslog モジュールを使って、簡単な Perl プログラムを作ればよい。Sys::Syslog モジュールは Perl の標準モジュールであり、Perl をインストールしていれば使えるはずだ。パイプ経由のログのハンドラはサーバ起動時に開始し、サーバ動作中はSTDINに対する入力を受け取るだけなので、Perl を使ってもパフォーマンスが低下することはない。

複数のWebサーバを運用していて、それらのログを1つのログファイルにまとめて記録したければ、すべてのサーバのログが中央の syslog サーバの syslog ファシリティを指すようにすればよい。このとき、ログエントリが順番通りにならないことがあるのに注意しよう。実際には問題にはならないが、最初は奇妙に見えるかもしれない。この影響を小さくするには、NTP(Network Time Protocol)を使って、Webサーバ間のクロックの同期をとるとよい。

ネットワーク経由の syslog サーバの設定についてもっと詳しく知りたければ、syslog のマニュアルを調べよう。

最後に、使っているOSによっては、loggerユーティリティを使って同じことを実現できるかもしれない。

その場合、次のように設定すればよい。

```
AccessLog "|/usr/bin/logger" combined
```

参照

- syslogd と syslog.conf の manpage

レシピ 3.21　ユーザディレクトリごとにログを記録する

課題

各ユーザディレクトリのWebサイト（つまり、http://server/~usernameでアクセスできるサイト）ごとに、ログファイルを持てるようにしたい。

解決

httpd.conf ファイルに、次のようなディレクティブを追加する。

```
CustomLog "|/usr/local/apache/bin/userdir_log" combined
```

そして、/usr/local/apache/bin/userdir_log ファイルには、次のコードを書いておく。

```
#!/usr/bin/perl

my $L = '/usr/local/apache/logs'; # ログディレクトリ

my %is_open = (); # File handle cache
$|=1;
open(F, ">>$L/access_log"); # デフォルトのエラーログ

while (my $log = <STDIN>) {
    if ($log =~ m!\s/~(.*?)/!) {
        my $u = $1;
        unless ($is_open{$u}) {
            my $fh;
            open $fh, '>>' . $L . '/'. $u;
            $is_open{$u} = $fh;
        }
        select ($is_open{$u});
        $|=1;
        print $log;
    }
    else {
        select F;
```

```
            $|=1;
            print F $log;
        }
    }

    close F;
    foreach my $h (keys %is_open) {
        close $h;
    }
```

解説

ユーザディレクトリにあるWebサイトへのリクエストは、通常はメインサーバのログに記録されて、ユーザのサイトごとの区別はされていない。そのため、ユーザが自分のWebサイトに関するログメッセージを見つけるのは、非常に難しい。

このレシピを使うと、ユーザディレクトリ宛てのリクエストは、ユーザごとのログファイルに振り分けられる。ユーザディレクトリのWebサイト以外に対するリクエストは、メインのログファイルに送られる。すべてのログメッセージが個人のログファイルだけではなく、メインのログファイルにも記録されるようにしたければ、ログハンドラを修正すれば可能だ。

必要なディスクアクセスを小さくするために、アクセスするたびにファイルをオープン、クローズするのではなく、ファイルハンドラをキャッシュしている。そのため、非常に多くのファイルハンドラが常にオープンされることになる。ユーザのWebサイトを非常にたくさん運営しているサイトでは、システムのリソースが足りなくなってしまうことがある。

Perlはデフォルトで出力をバッファリングするため、出力をバッファリングしないようスクリプトで明示的に指示する必要がある。こうしておくと、ログエントリを即座にログファイルに記録することができる。そのためには、autoflush変数"$|"に真の値を設定すればよい。この例では、最近選択したファイルハンドラに対して、出力をバッファリングしないよう設定している。こうした対策をしないと、出力がバッファリングされてしまい、ログファイルに何も書き込まれていないように見えてしまう。

別のアプローチとして、mod_rewriteを使って環境変数を設定し、LogFormatディレクティブにその変数を追加してもよい。

```
RewriteRule ^/~([^/]+)/ - [E=userdir:$1]
LogFormat "%{userdir}e %h %l %u %t \"%r\" %>s %b" common
```

こうしておくと、split-logfileスクリプトを使って、ユーザごとにログファイルを分割することができる。

参照

- http://httpd.apache.org/docs/2.0/mod/mod_log_config.html
- http://httpd.apache.org/docs/2.2/mod/mod_log_config.html
- http://httpd.apache.org/docs/2.2/programs/other.html

4章
バーチャルホスト

本名とニックネームという複数の名前で知られている人がいるように、複数のWebサイトを扱うWebサーバがある。Apacheの設定ファイルでは、代替サーバもメインサーバも、すべてバーチャルホストと呼び(vhostと書くこともある)、<VirtualHost>ディレクティブを使って識別する。Apacheは、アクセスするのに使われた名前に合わせて適切なレスポンスを返す。これは「Jonesさん」と呼ばれたときと「やあ、Debbie」と呼ばれたときで、異なる応対をすることに似ている。Apacheを適切に設定すると、1つのシステムで複数のWebサイトをサポートすることができる。システムを正しく設定するには、割り当てられているIPアドレスなどの情報を知っておく必要がある。

Apacheは、2種類のバーチャルホストをサポートしている。1つは、アドレスベースまたはIPベースのバーチャルホストであり、システムへのアクセスを電話番号のような数値のネットワークアドレスに関連付けている。Bruce Wayne(バットマンの正体)は、自宅の居間にある電話に出るときに「はい、バットマンです」とは言わないし、バットマンの秘密基地にある電話に出るときに「はい、Bruce Wayneです」とは言わない。しかし、電話に出ているのは全く同じ人だ。これは、同じWebサーバが、異なるアドレス宛てのリクエストを受信するのによく似ている。電話をかけた人が番号を間違って「やあ、Steve!」と言ったとしても、電話はまだ同じように応答する。Bruce Wayneが応対しているということを、バットマンの電話で告白するようなことはない。

もう1つは、ネームベースのバーチャルホストであり、呼ばれた名前に基づいてサーバが応答する。ここでも電話の例を考えてみよう。Dave、Joyce、Amaterasu、Georgの4人がルームメイトとしてアパートを共有しているとき、このうちの1人と話したければ、同じ電話番号にかける必要がある。このように、1つの電話番号を複数の人で共有するのと同じように、1つのIPアドレスを複数のWebサイトで共有することができる。ただし、Apacheのバーチャルホストで共有するすべてのIPアドレスは、NameVirtualHostディレクティブを使って宣言しておく必要がある。

Apacheの最も簡単な設定は、バーチャルホストを使わないことだ。この場合、設定ファイルにあるすべてのディレクティブが、サーバ全体の操作に対して適用される。<VirtualHost>コンテナの外部にあるディレクティブで定義された環境のことを「デフォルトサーバ」や「メインサーバ」、「グローバルサーバ」と呼ぶことがあるが、公式な名前はない。これはバーチャルホストを設定に追加するときの重要な要素となる。

ところで、<VirtualHost>コンテナを追加すると何が起こるのだろうか?コンテナの外部にあるディレクティブはどのように解釈されて、バーチャルホストにどんな影響を与えるのだろうか?

簡単な答えはないが、基本的には個々の設定ディレクティブによって影響は異なる。バーチャルホストに受け継がれるものもあるし、デフォルト値にリセットされるものもある。これまで全く定義されていなかったように動作するものもある。これを確認するには、それぞれのディレクティブに関するドキュメントを調べる必要がある。

バーチャルホストには主に2つの方式がある。IPベースのバーチャルホストは、バーチャルホストごとに固有のIPアドレスを割り当てる。ネームベースのバーチャルホストは、同じIPアドレス上で1つ以上のバーチャルホストが動かすが、それぞれ異なる名前を持っている。本章では、それぞれの方式の設定方法と、同じサーバ上で両方の方式を組み合わせる方法について解説する。また、バーチャルホストを使ったときによく起こる問題とその修正方法についても説明する。

問題が発生したり、悩ましいエラーメッセージが出力されたりするのを避けるため、IPアドレスとポート番号の両方が指定できるようになっているディレクティブには、IPアドレスだけでなくポート番号も指定しておくことを、強く推奨する。例えば、次のように設定するのではなく、

```
NameVirtualHost *
```

次のように、ポート番号も指定する。

```
NameVirtualHost *:80
```

通常のWeb操作にはポート80を使い、SSLリクエストにはポート443を使うことが多い。

レシピ 4.1 ネームベースのバーチャルホストを設定する

課題

IPアドレスは1つしかないが、このシステムで1つ以上のWebサイトをサポートしたい。

解決

<VirtualHost>セクションとNameVirtualHost *:80ディレクティブを使って、次のように設定する。

```
ServerName 127.0.0.1
NameVirtualHost *:80

<VirtualHost *:80>
    ServerName TheSmiths.name
    DocumentRoot "C:/Apache/Sites/TheSmiths"
</VirtualHost>

<VirtualHost *:80>
    ServerName JohnSmith.name
    DocumentRoot "C:/Apache/Sites/JustJohnSmith"
</VirtualHost>
```

解説

IPアドレスを入手するのがますます難しくなってきているため、1つのApacheサーバで複数のWebサイトを運営するには、ネームベースのバーチャルホストを使うのが最も一般的な方法だ。このレシピは、バーチャルホストを利用したいユーザにとって、きっと役に立つだろう。

このレシピにある"*:80"は、指定したホストがすべてのアドレスで動作することを意味している。1つのIPアドレスしか持っていないマシンでは、そのアドレスで動作するのはもちろん、ループバックやローカルホスト(localhost)アドレスでも動作する。つまり、サーバシステムの前に座って、ローカルにWebサイトを見ることができる。

<VirtualHost>コンテナディレクティブの引数は、NameVirtualHostディレクティブの引数と一致している必要がある。ここにホスト名を書いてしまうと、サーバ起動時にバーチャルホストが無視されてしまい、バーチャルホスト宛てのリクエストはどこか他のところへいってしまうおそれがある。ネームサーバがダウンしていたり、Apacheサーバが起動したときに応答がなかったりすると、Apacheは<VirtualHost>セクションとNameVirtualHostディレクティブを対応付けることができなくなる。

設定ファイルに定義したバーチャルホスト以外へのリクエストは、最初に定義したバーチャルホストに送られる。この例の場合、バーチャルホストの1つとして明示的に定義されていないホスト名宛てに送られたリクエストは、TheSmiths.nameというバーチャルホストに送られることになる。

"httpd -S"を実行すると、特に役に立つ情報を得ることができる。Apacheがバーチャルホストの設定をどのように理解しているのか、自分の理解と一致しているかどうか確認するとよい。"httpd -S"はバーチャルホストの設定について、どのホストがネームベースで、どのホストがIPベースで、デフォルトが何であるか、といった情報を返す。

ServerAliasディレクティブを使うと、特定のバーチャルホストに複数の名前を定義することができる。次のように設定すればよい。

```
ServerName TheSmiths.name
ServerAlias www.TheSmiths.name Smith.Family.name
```

理解しておくべき重要なことは、設定ファイルのメインボディで定義されているサーバ(メインサーバやデフォルトサーバ)がアクセス不能になったときには、バーチャルホストが代わりを引き受けるということだ。そのホストに対して、明示的にバーチャルホストのセクションを作成しておく必要がある。そのバーチャルホストをデフォルトにしたければ、設定ファイルの中でそのホストを最初に定義すればよい。

Apacheの設定にネームベースのバーチャルホストを追加したからといって、魔法のようにDNSサーバにエントリを追加してくれるわけではない。最初に、DNSサーバにレコードを追加しなければならない。そうして初めて、その名前をIPアドレスに変換することができる。ユーザがブラウザのロケーションバーにサーバ名を入力すると、コンピュータはまずDNSサーバと通信してその名前を検索し、IPアドレスに変換しようとする。DNSにレコードがなければ、ブラウザはサーバを見つけることができない。

DNSサーバの設定について詳しく知りたければ、動かしているDNSソフトウェアのドキュメントを調べよう。自分でDNSサーバを動かしていなければ、利用しているISPに問い合わせよう。

参照

- http://httpd.apache.org/docs/2.2/vhosts/

レシピ4.2 デフォルトのネームベースのバーチャルホストを設定する

課題

名前を指定したリクエストであっても、IPアドレスを使用したリクエストあっても、マッチしなかったすべてのリクエストをデフォルトのホストに振り向けたい。できれば「ホストが見つかりません」というエラーメッセージを返したい。

解決

他の<VirtualHost>セクションよりも前に、次のセクションを追加すればよい。

```
<VirtualHost *:80>
    ServerName default
    DocumentRoot /www/htdocs
    ErrorDocument 404 /site_list.html
</VirtualHost>
```

解説

このレシピは、ネームベースのバーチャルホストを使っている環境で利用することができる。ここでは<VirtualHost *:80>という記述を使ったもう1つのバーチャルホストを定義しており、この前に対応するNameVirtualHost *:80という記述があることを想定している。ここでは、わかりやすくするためにdefaultという名前を使ったが、好きな名前を付けることができる。

あまり役に立たない404のエラーメッセージを出して、ユーザを立ち往生させるのではなく、ErrorDocument 404にサーバで利用可能なサイトのリストを設定しておくと、ユーザにとって役に立つコンテンツを見せることができる。また、利用可能なサイトのリストをDirectoryIndexにも設定しておくと、サイトのトップページに直接やってきたユーザも有用な情報を手に入れることができる。

すべての有効なホスト名をServerNameやServerAliasに明示的に定義しておくのもよいだろう。誰もデフォルトのサイトにたどり着けなくなる。それでも、誰かがIPアドレスで直接そのサイトにアクセスしたり、適切なバーチャルホストを作る前にホスト名をそのアドレスに追加した場合には、デフォルトのサイトに振り向けて役に立つ情報を示すことができる。

参照

- レシピ4.4

レシピ 4.3　アドレスベースのバーチャルホストを設定する

課題

システムには複数の IP アドレスが割り当てられており、IP アドレスごとに Web サイトを動かしたい。

解決

利用したい IP アドレスごとに、バーチャルホストのセクションを作ればよい。

```
ServerName 127.0.0.1

<VirtualHost 10.0.0.1>
    ServerName Example.Com
    DocumentRoot "C:/Apache/Sites/Example.Com"
</VirtualHost>

<VirtualHost 10.0.0.2>
    ServerName JohnSmith.Example.Com
    DocumentRoot "C:/Apache/Sites/JustJohnSmith"
</VirtualHost>
```

解説

ここで定義したバーチャルホストは、ホスト名が何であっても、指定のIPアドレス宛てのリクエストをすべて受け取る。ここに定義されていないIPアドレス宛てのリクエストは、設定ファイルのメインボディに定義されたバーチャルホストに送られる。

ServerNameに指定した名前は、必要に応じてバーチャルホストの基本名として使われるが、リクエストを正しいホストに対応付ける過程には使われない。どのバーチャルホストがリクエストを処理するかを判断するのに使われるのは、IP アドレス(Host ヘッダフィールドではない)だけである。

参照

- http://httpd.apache.org/docs/2.2/vhosts/

レシピ4.4　デフォルトのアドレスベースのバーチャルホストを設定する

課題

どのアドレスベースのバーチャルホストにも対応付けられないリクエストを、すべて受け取るようなバーチャルホストを作りたい。

解決

default キーワードを使って、デフォルトのホストを指定すればよい。

```
<VirtualHost _default_>
    DocumentRoot /www/htdocs
</VirtualHost>
```

解説

default キーワードを使うと、バーチャルホストが設定されていない「アドレス:ポート」宛てのリクエストをすべて受け取るようなバーチャルホストを作ることができる。

_default_ディレクティブは、次のように特定のポート番号を指定することができるので、指定しておくべきである。

```
<VirtualHost _default_:443>
```

このように設定すると、バーチャルホストとして明示的に設定されていないすべてのアドレスのポート443宛てのリクエストを、このバーチャルホストが受け取る。SSLバーチャルホストは通常、この_default_の構文を使って設定する。そのため、デフォルトのSSL設定ファイルでは、SSLを有効にするのに必要なディレクティブと一緒にこの構文を見かけることが多い。

_default_は、ネームベースのバーチャルホストの場合には、期待するようには動かない。バーチャルホストに指定されていない名前にはマッチせず、バーチャルホストが設定されていない「アドレス:ポート」にのみマッチする。デフォルトのネームベースのバーチャルホストを作りたいなら、レシピ4.2を参照すること。

参照

- レシピ 4.2

レシピ4.5 アドレスベースとネームベースのバーチャルホストを混在させる

課題

システムに複数のIPアドレスが割り当てられており、それぞれのアドレスで1つ以上のWebサイトを動かしたい。

解決

IP アドレスごとに NameVirtualHost ディレクティブを指定し、それぞれを IP アドレスが 1 つしかない場合と同じように設定すればよい。

```
ServerName 127.0.0.1
NameVirtualHost 10.0.0.1:80
NameVirtualHost 10.0.0.2:80

<VirtualHost 10.0.0.1:80>
    ServerName TheSmiths.name
    DocumentRoot "C:/Apache/Sites/TheSmiths"
</VirtualHost>
```

```
<VirtualHost 10.0.0.1:80>
    ServerName JohnSmith.name
    DocumentRoot "C:/Apache/Sites/JustJohnSmith"
</VirtualHost>

<VirtualHost 10.0.0.2:80>
    ServerName Example.Com
    DocumentRoot "C:/Apache/Sites/Example.Com"
</VirtualHost>

<VirtualHost 10.0.0.2:80>
    ServerName DoriFerguson.Example.Com
    DocumentRoot "C:/Apache/Sites/JustDoriFerguson"
</VirtualHost>
```

解説

<VirtualHost>の引数に、ワイルドカード"*"引数ではなくサーバのアドレスを指定すると、バーチャルホストはそのIPアドレスだけを待ち受けるようになる。ただし、<VirtualHost>の引数は、そのバーチャルホストに該当する NameVirtualHost の引数と一致していなければならないことに注意しよう。

参照

- http://httpd.apache.org/docs/2.2/vhosts/

レシピ4.6　mod_vhost_aliasを使ってバーチャルホストをまとめる

課題

全く同じ設定のバーチャルホストがたくさんあるので、まとめて扱いたい。

解決

mod_vhost_alias モジュールが提供している VirtualDocumentRoot ディレクティブと VirtualScriptAlias ディレクティブを使って、次のように設定すればよい。

```
VirtualDocumentRoot /www/vhosts/%-1/%-2.1/%-2/htdocs
VirtualScriptAlias  /www/vhosts/%-1/%-2.1/%-2/cgi-bin
```

解説

このレシピでは、mod_vhost_aliasモジュールが提供するディレクティブを使う。このモジュールはデフォルトでは無効になっているため、自分でApacheをビルドした場合にはインストールされていないかもしれない。

このディレクティブを使うと、リクエストをその宛先ホスト名のパーツで構成したディレクトリに対応付

けることができる。それぞれの変数はホスト名のパーツを表しており、ホスト名ごとに異なるディレクトリに対応付けることができる。

この例では、*www.example.com* にあるコンテンツへのリクエストは、/www/vhosts/com/e/example/htdocs ディレクトリまたは/www/vhosts/com/e/example/cgi-bin ディレクトリ(CGI リクエストの場合)に振り向けられる。利用可能なすべての変数を表 4-1 に示す。

表 4-1 mod_vhost_alias 変数

変数	意味
%%	% 文字を挿入する
%p	バーチャルホストのポート番号を挿入する
%M.N	名前(のパーツ)を挿入する

M と *N* は正および負の整数を指定する。表 4-2 にその変数の値の意味を示す。

表 4-2 変数の値の意味

変数	意味
0	名前全体
1	名前の最初のパーツ
-1	名前の最後のパーツ
2	名前の 2 番目のパーツ
-2	名前の最後から 2 番目のパーツ
2+	名前の 2 番目以降のパーツすべて
-2+	名前の最後から 2 番目以前のパーツすべて

引数の最初の部分(*%M.N* の *M*)にある値は、ホスト名の該当パーツを表している。引数の2番目の部分(*N*)は、ホスト名の該当パーツ内の特定の 1 文字を表している。例えば、ホスト名が *www.example.com* の場合、変数の意味は表 4-3 に示すようになる。

表 4-3 ホスト名が www.example.com の場合の値

値	意味
%0	www.example.com
%1	www
%2	example
%3	com
%-1	com
%-2	example
%-3	www
%-2.1	e
%-2.2	x
%-2.3+	ample

ドメイン名の先頭文字やトップレベルドメイン、あるいは単にホスト名で分けるなど、バーチャルホストの数に応じて適切なディレクトリ構造を作るとよいだろう。

mod_vhost_aliasモジュールでは、DOCUMENT_ROOT環境変数を設定していないことに注意しよう。この値に依存しているアプリケーションは、このようなバーチャルホスト環境ではうまく動かないことがある。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_vhost_alias.html
- http://httpd.apache.org/docs/2.2/vhosts/

レシピ4.7　書き換えルールを使ってバーチャルホストをまとめる

課題

たくさんのバーチャルホストをサポートすることを目的としたmod_vhost_aliasのようなモジュールはあるが、用途が限られており、すべてのバーチャルホストを全く同じ設定にする必要がある。たくさんのバーチャルホストをサポートしたいが、それぞれ異なる設定にしたいため、mod_vhost_aliasモジュールを使うことができない。

解決

mod_rewrite モジュールのディレクティブを使って、ホスト名に基づいたディレクトリに対応付ける。

```
RewriteEngine on
RewriteCond   "%{HTTP_HOST}"     "^(www\.)?([^.]+)\.com"
RewriteRule   "^(.*)$"           "/home/%2$1"
```

解説

mod_vhost_alias モジュールは便利で、それぞれのバーチャルホストの設定がホスト名とドキュメントツリーを除いてすべて同じ場合には、最良の選択だ。ただし mod_vhost_alias を使うと、他の URL マッピングモジュール、例えば、mod_userdir、mod_rewrite、 mod_aliasのようなモジュールが使えなくなる。これは大きな制約になる。mod_rewrite を使うと、効率は落ちるが、柔軟性高く実現することができる。

例えば、mod_vhost_alias を使うと、すべてのホストで mod_vhost_alias を使わなければならない。これに対して、mod_rewriteを使うと、一部のホストには書き換えルールを使い、それ以外には従来通りのバーチャルホストの設定方法を使うことができる。

この解決策で示したディレクティブでは、*www.something.com*（あるいは、*www*がなくてもよい）宛てのリクエストをディレクトリ /home/something に対応付けている。

参照

- レシピ 5.17
- http://httpd.apache.org/docs/2.2/vhosts/

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 4.8 バーチャルホストごとにログを記録する

課題

バーチャルホストごとに別々のログファイルを使いたい。

解決

バーチャルホストの宣言ごとに、ErrorLog ディレクティブと CustomLog ディレクティブを指定すればよい。

```
<VirtualHost *:80>
    ServerName   waldo.example.com
    DocumentRoot /home/waldo/www/htdocs

    ErrorLog     /home/waldo/www/logs/error_log
    CustomLog    /home/waldo/www/logs/access_log combined
</VirtualHost>
```

解説

設定ファイルの本体部分や<VirtualHost>セクションには、いろいろなログ用ディレクティブを置くことができる。バーチャルホストの定義内にディレクティブを置くと、そのバーチャルホストのログエントリは、メインサーバの設定で定義されたログファイルではなくバーチャルホストの定義で指定したログファイルに書き込まれる。

バーチャルホストごとにログファイルを作ると、サーバで利用可能なファイルとネットワーク接続の総数の上限に到達するおそれがある。例えば、バーチャルホストが 100 個あると、そのエラーログと動作ログで、200個のファイルがオープンされることになる。サーバの上限が256個だとすると、56 個までのリクエストしか平行して処理できなくなる。
ここで挙げた数は単なる例であり、実際にオープンできるファイルの最大数はプラットフォームによって異なっている。通常はもっと大きな値になっているだろう。実際の上限を知りたいなら、プラットフォームのドキュメントを調べよう。
したがって、バーチャルホストのログはすべて同じファイルに記録して、解析、調査するときに、後から分割することを推奨する。

ここで紹介したレシピでは、メインのログディレクトリではなく、特定のユーザのホームディレクトリにログファイルを置いている。こうすると、ユーザはログファイルにアクセスしやすくなるが、そのディレクトリに適切なパーミッションを設定しておかないと問題になることがある。ファイルのパーミッションについては、6 章の解説を参照すること。

参照

- 3 章
- 6 章
- レシピ 4.9

レシピ 4.9　バーチャルホストごとにログファイルを分割する

課題

たくさんのバーチャルホストがあるが、すべてのバーチャルホストのログを 1 つのログファイルに記録して、後からそのログファイルをバーチャルホストごとに分割したい。

解決

設定ファイルに、次のように設定する。

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost
CustomLog logs/vhost_log vhost
```

ログファイルがローテーションした後、次のコマンドを実行する。

```
split-logfile < logs/vhost_log
```

解説

このレシピにある LogFormat ディレクティブでは、common ログファイルフォーマットとよく似たログファイルを作成するが、アクセスされたバーチャルホストの名前が追加されている。split-logfile ユーティリティは、このログファイルをバーチャルホストごとに分割する。

参照

- レシピ 3.11

レシピ 4.10　ポートベースのバーチャルホストを設定する

課題

ポートが異なる HTTP 接続ごとに、異なるコンテンツを公開したい。

解決

<VirtualHost> ディレクティブで、ポート番号を明示的に指定すればよい。

```
Listen 8080

<VirtualHost 10.0.1.2:8080>
    DocumentRoot /www/vhosts/port8080
</VirtualHost>

Listen 9090

<VirtualHost 10.0.1.2:9090>
    DocumentRoot /www/vhosts/port9090
<VirtualHost>
```

解説

ポートベースのバーチャルホストは、本章で紹介した他のテクニックと比べると、あまり一般的ではない。しかし、これが役に立つ場合もある。1つのIPアドレスしかない場合や、DNSにホスト名を追加することができない場合、ISPがポート80への着信トラフィックを遮断している場合などには、別のポートでバーチャルホストを動かせると役に立つ。

また、開発環境においても、開発者や設定ごとに異なるポートを使って、別々のhttpdプロセスを動かせると便利だ。

Webサイトを訪問するユーザは、URLにポート番号を指定する必要がある。例えば、この例の2番目のバーチャルホストからコンテンツを取得するには、次の URL にアクセスすればよい。

```
http://server.example.com:9090
```

参照

- http://httpd.apache.org/docs/2.2/vhosts/

レシピ 4.11 複数のアドレス上で同じコンテンツを表示する

課題

2つのアドレス上で同じコンテンツを表示したい。

解決

<VirtualHost> ディレクティブに、2つのアドレスを指定すればよい。

```
NameVirtualHost 192.168.1.1:80
NameVirtualHost 172.20.30.40:80

<VirtualHost 192.168.1.1:80 172.20.30.40:80>
    DocumentRoot /www/vhosts/server
    ServerName server.example.com
```

```
    ServerAlias server
</VirtualHost>
```

解説

この設定は、ネットワーク内部で使うアドレスとネットワーク外部からのみアクセス可能なアドレスを持ったマシンを使う場合に、とても役に立つ。アドレスが2つしかなければ、レシピ4.1で紹介した"*"の記法を使えばよいだろう。しかし、もっと多くのアドレスがある場合には、このレシピにあるように、どのアドレスにどのコンテンツを表示したいのか、アドレスを明示的に指定すればよい。

参照

- http://httpd.apache.org/docs/2.2/vhosts/

レシピ4.12　データベースを使ってバーチャルホストを定義する

課題

バーチャルホストを定義するのに、毎回設定ファイルを編集するのではなく、データベースを使って定義したい。

解決

http://www.outoforder.cc/projects/apache/mod_vhost_dbi から mod_vhost_dbi を入手し、このモジュールを使って、バーチャルホストをデータベースから読み込むように設定する。

```
PoolDbiDriver         Server1  mysql
PoolDbiHost           Server1  192.168.1.50
PoolDbiUsername       Server1  datauser
PoolDbiPassword       Server1  password
PoolDbiDBName         Server1  vhosts
PoolDbiConnMin        Server1  1
PoolDbiConnSoftMax    Server1  1
PoolDbiConnHardMax    Server1  5
PoolDbiConnTTL        Server1  30

<VirtualHost *:80>
  VhostDbiEnabled On
  VhostDbiConnName Server1
  VhostDbiQuery "SELECT ServerName, DocumentRoot, Username " \
        FROM vhost_info WHERE ServerName = &{RequestHostname}"
</VirtualHost>
```

解説

mod_vhost_dbiはサードパーティ製モジュールであり、バーチャルホストの設定をデータベースに置くこと

で、設定ファイルを修正したり、Apacheを再起動したりすることなく、バーチャルホストを更新することができる。

このモジュールに関する完全なドキュメントとコードは、http://www.outoforder.cc/projects/apache/mod_vhost_dbiから入手することができる。ここで紹介した設定を使えば、バーチャルホストを動かすことができるはずだ。vhost_infoテーブルには、バーチャルホストごとにレコードを作成する必要がある。このレコードは、ServerName、DocumentRoot、Usernameを含んでいなければならない。ServerNameとDocumentRootはその名前通りの意味であり、Usernameはsuexecが CGI 処理を実行するユーザ ID を指す。

残念ながら、mod_vhost_dbiでできることは限られており、提供しているのはこの3つの設定ディレクティブだけだ。

5章
エイリアスとリダイレクトとリライト

Apacheはリクエストを受信すると、DocumentRootディレクトリにあるファイルをクライアントに提供しようとする。しかし、別の場所にあるリソースを提供したいときもある。例えば、Webサイトにドキュメント一式を置きたいとき、ドキュメントを新しい場所に移動しなくても今ある場所に置いたまま提供できると便利だろう。

本章では、このような話題を3つの大きなカテゴリに分けて紹介する。「エイリアス」は、URLを特定のディレクトリに対応付けることを意味している。「リダイレクト」は、URLを別のURLに対応付けることを意味している。そして、「リライト」は、mod_rewriteモジュールを使って、URLを書き換えてしまうことを意味している。

また、関連するレシピとして、ファイルシステム上の想定外の場所にあるリソースにURLを対応付ける方法も紹介する。

ここで取り上げた話題は、リンク切れを避けたり、定期的に大幅な変更(ファイルやディレクトリをあちこちに移動したり、さらには別のサーバに移動するなど)をすることがあるサイトを運営しているWebマスターにとって、特に興味深いものだろう。Apache Webサーバのリダイレクトとリライトの機能を使うと、このような見苦しい舞台裏の混乱をサイトの訪問者に見せなくて済む。

レシピ 5.1　URL をディレクトリに対応付ける

課題

DocumentRoot以外のディレクトリにあるコンテンツをクライアントに提供したい。例えば、すでに特定のディレクトリにドキュメントがあり、そのドキュメントをWebサイトに置きたいが、ApacheのDocumentRootディレクトリには移動したくないことがある。

解決

httpd.confに、次のようにAliasディレクティブを追加すればよい。

```
Alias "/desired-URL-prefix" "/path/to/other/directory"
```

解説

この例では、/desired-URL-prefixで始まるURLを、/path/to/other/directoryディレクトリにあるファイルに対応付けている。例えば、クライアントが次のURLに対してリクエストすると、

```
http://example.com/desired-URL-prefix/something.html
```

/path/to/other/directory/something.htmlというファイルがクライアントに送られる。

Unix系システムでは、単にメインのドキュメントディレクトリからターゲットのディレクトリに対してシンボリックリンクを張り、Options +FollowSymLinksディレクティブ†を有効にしても、同じ効果を得ることができる。しかし、明示的にAliasを使うと、もっとディレクトリの把握がしやすくなる。ディレクトリにシンボリックリンクを張ると、コンテンツの場所を把握するのが難しくなる。さらに、シンボリックリンクがあちこちにあると、ファイルシステム上の公開するつもりのない部分まで公開してしまうおそれもある。

対応付けしたディレクトリにアクセスできるようにするためには、さらに設定ディレクティブを追加する必要があるかもしれない。アクセスできない場合には、「サーバの設定によって拒否されました」というエラーメッセージがerror_logファイルに記録される。これはかなりよくあることだ。Apacheのドキュメント(http://httpd.apache.org/docs/2.2/misc/security_tips.html#protectserverfiles)では、デフォルトでDocumentRootディレクトリ以外へのすべてのアクセスを拒否するように設定することを推奨している。したがって、次のような設定ブロックを追加して、対象となるディレクトリの設定を上書きする必要がある。

```
<Directory "/path/to/other/directory">
    Order allow,deny
    Allow from all
</Directory>
```

これにより、指定したディレクトリにアクセスすることができる。

Aliasはスラッシュ(/)を非常に厳密に扱うことに注意すること。例えば、次のAliasディレクティブを考えてみる。

```
Alias "/puppies/" "/www/docs/puppies/"
```

このディレクティブは、/puppies/で始まるURLをエイリアスするが、/puppiesというURL(末尾にスラッシュ(/)がない)はエイリアスしない。これは「末尾のスラッシュ問題」を引き起こすおそれがある。ユーザがhttp://example.com/puppiesというURLにアクセスすると404のエラーが返ってきて、末尾にスラッシュを付けたhttp://example.com/puppies/というURLにアクセスすると望みのディレクトリにあるコンテンツを受け取ることができる、という現象だ。この問題を回避するためには、Aliasの引数には末尾のスラッシュを付けなければよい。

最後に、Aliasの最初の引数の末尾にスラッシュを付けるなら、2番目の引数にも付けてあるか確認しよう。次のような例を考えてみる。

† http://httpd.apache.org/docs/2.2/mod/core.html#optionsにあるOptionsディレクティブに関するドキュメントを参照すること。

```
Alias "/icons/" "/usr/local/apache/icons"
```

ユーザが http://example.com/icons/test.gif をリクエストすると、Apache は期待する /usr/local/apache/icons/test.gifではなく、/usr/local/apache/iconstest.gifというファイルを提供しようとする。この問題は、「スラッシュパリティの維持」と呼ばれ、次のようなしゃれた言い回しがある。「エイリアスにスラッシュを付けたら、ディレクトリにも付けろ。エイリアスにスラッシュを付けないなら、ディレクトリにも付けるな。」

参照

- http://httpd.apache.org/docs/2.2/mod/mod_alias.html
- http://httpd.apache.org/docs/2.2/mod/core.html#options

レシピ 5.2　既存のコンテンツを新しい URL でアクセスする

課題

既存のディレクトリを違う名前でアクセスできるようにしたい。

解決

httpd.conf で Alias ディレクティブを使って、次のように設定する。

```
Alias "/newurl" "/www/htdocs/oldurl"
```

解説

Alias は、URL を DocumentRoot ディレクトリツリーの外部にあるディレクトリに対応付けるために使うのが一般的だが、必ずしもそうする必要はない。あるコンテンツをいろいろな名前でアクセスできるようにしたいことも多い。よくあるのは、ディレクトリ名を変更しても、以前のURLでアクセスできるようにしたい場合だ。あるいは、いろいろな人が同じコンテンツを別の名前で参照したい場合もある。

Alias はローカルのURI（http://example.com/foo/bar.txtのうち/foo/bar.txtの部分）にだけ影響を与えるものであり、ホスト名部分（http://example.com/の部分）には影響を与えないということに注意しよう。URLのホスト名部分を変更したいのであれば、Redirect ディレクティブや RewriteRule ディレクティブを使えばよい。

参照

- レシピ 5.1
- http://httpd.apache.org/docs/2.2/mod/mod_alias.html
- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.3　ユーザに独自の URL を与える

課題

システムのユーザごとに独自の Web スペースを与えたい。

解決

ユーザのホームディレクトリの下にそのユーザのWebスペースを作りたい場合には、httpd.confファイルに次の行を追加すればよい。

```
UserDir public_html
```

ある1つの場所に全ユーザのWebディレクトリを置きたい場合には、次の行を追加する。

```
UserDir "/www/users/*/htdocs"
```

URLにチルダ(~)を入れずに、ユーザがホームディレクトリにアクセスできるようにしたい場合には、mod_rewriteを使って対応付けることができる。

```
RewriteEngine On
RewriteCond "/home/$1/public_html" -d [NC]
RewriteRule "^/([^/]+)/(.*)" "/home/$1/public_html/$2"
```

最後に、mod_perlがインストールされていれば、次のように、もっと高度なこともできる(これも同様に、httpd.confファイルに追加する)。

```
<Perl>
# この特権を与えたくないユーザたち
my %forbid = map { $_ => 1 } qw(root postgres bob);
opendir H, '/home/';
my @dir = readdir(H);
closedir H;
foreach my $u (@dir) {
    next if $u =~ m/^\./;
    next if $forbid{$u};
    if (-e "/home/$u/public_html") {
        push @Alias, "/$u/", "/home/$u/public_html/";
    }
}
</Perl>
```

解説

最初の解決策は、ここに挙げたレシピのうち、最も簡単でよく使われるものだ。UserDirディレクティブを設定すると、システムのユーザであれば誰でも、自分のホームディレクトリにpublic_htmlという名前のディレクトリを作って、そこにWebコンテンツを置けるようになる。このWebスペースは、チルダ(~)の後にユーザ名が続くURLでアクセスすることができる。例えば、bacchusという名前のユーザのWebスペースは、次のURLでアクセスすればよい。

```
http://www.example.com/~bacchus/
```

標準で配布されているソースコードからApacheをインストールした場合には、デフォルトの設定ファイルにこの設定例がすでに含まれているはずだ。設定ファイルには、/home/*/public_htmlディレクトリを参照している<Directory>セクションも含まれており、オプションやパーミッションがいくつか有効になっている。誰でもユーザのWebサイトにアクセスできるようにするには、このセクションのコメントを外す必要がある。このセクションは次のようになっている。

```
<Directory "/home/*/public_html">
    AllowOverride FileInfo AuthConfig Limit
    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
    <Limit GET POST OPTIONS PROPFIND>
        Order allow,deny
        Allow from all
    </Limit>
    <LimitExcept GET POST OPTIONS PROPFIND>
        Order deny,allow
        Deny from all
    </LimitExcept>
</Directory>
```

ただし、これらのディレクティブが何をしているのかよく確認してから、設定ファイルのこのセクションのコメントを外すこと。

2番目の解決策では、最初の解決策と違って、UserDirに与える引数が絶対パスになっている。絶対パスを使うと、ユーザのホームディレクトリからの相対パスではなく、実際のファイルシステム上のパスとして解釈される。パス中の"*"はユーザ名で置き換えられる。例えば、http://example.com/~smith/は/www/users/smith/htdocsと解釈される。このディレクトリ構造もまた、前の例と同じようにオプションやパーミッションを設定する必要がある。

3番目の解決策は少し高度で、URLにチルダ(~)を使わずにUserDir機能を提供する。

RewriteCondディレクティブを使って、まずユーザのホームディレクトリが存在するかどうかチェックし、存在していれば、リクエストをそのディレクトリに書き換える。最初にこのチェックを行うことにより、他のURLはこれまで通り正しく処理したまま、正当なユーザ名で始まるURLだけをユーザのホームディレクトリに書き換えることができる。

この書き換えルールは、mod_rewriteのあまり知られていない事実をうまく利用している。それは、RewriteRuleが常に最初に評価され、もしマッチすればその後にRewriteCondが評価される、というものだ。これにより、RewriteCondで使われている$1の値は、次の行にあるRewriteRuleでセットされているにもかかわらず、RewriteCond中で使うことができる。

4番目の、最後の解決策にはmod_perlモジュールが必要であり、/homeディレクトリの階層にあるすべての最上位ディレクトリ(通常はユーザのホームディレクトリ)に対してエイリアスの対応付けを行う。ユーザ名の前にチルダが付かないという点において、この解決策は最初に挙げた1番目と2番目の解決策とは違っ

ている。ユーザsmithのWebページは、http://example.com/~smith/ではなくhttp://example.com/smith/になる。しかし、ファイルシステム上の場所は/home/smith/public_htmlのままだ。

いずれの解決策においても、リクエストされたディレクトリとそこに至るまでの上位のパスに含まれるディレクトリは、Apacheユーザ(通常、nobodyやwww、もしくはhttpd)にとって読み出し可能になっている必要があり、また、Apacheユーザに対して実行ビットが設定されている必要がある。こうして初めて、Apacheサーバはそのディレクトリからコンテンツを読み出すことができる。例えば、ユーザbobの場合、ディレクトリ/、/home、/home/bob、/home/bob/public_html(または他の解決策でこれらに相当するディレクトリパス)のすべてに実行アクセスが必要であり、さらに最後のディレクトリには読み出しアクセスも必要になる。

Unix系システムでは、次のコマンドでパーミッションを設定することができる。

```
% chmod o+x / /home /home/bob
% chmod o+rx /home/bob/public_html
```

ディレクトリ内にあるファイルは読み出し可能にしておくだけでよい。

```
% find /home/bob/public_html -type f | xargs chmod 644
```

このコマンドはサブディレクトリ以下を再帰的にチェックし、ディレクトリ以外のすべてのファイルのパーミッションを変更する。

最初の解決策を使った場合、通常は他のユーザ全員に自分のディレクトリを読み出し可能にしてしまうため、ファイルのパーミッションを心配するユーザが多い。この問題をユーザに警告して、個人的なファイルが世界中の人々に読まれてしまわないように注意すべきである。

1番目の解決策に比べて、2番目の解決策を使ったときの利点は、ユーザのホームディレクトリ以外の場所にファイルを格納できることだ。この場合、ユーザはホームディレクトリにあるファイルのパーミッションを適切に設定したままでよい。誰かが自分の個人的なファイルに自由にアクセスするかもしれない、という心配をせずに、ファイルをホームディレクトリに格納することができる。

最後の解決策はこれまでの解決策と全く違っており、mod_perlモジュールをインストールしておく必要がある。mod_perlが提供する<Perl>設定ディレクティブを使って、これまでに説明した一連のディレクティブを設定ファイルに挿入する。設定ファイルにPerlコードを書いておくと、サーバ起動時に動的に設定を追加することができる。

サーバ起動時、このPerlコードは、public_htmlディレクトリを持っているユーザを見つけるために/home/ディレクトリをチェックする。そして、見つかったpublic_htmlディレクトリに対してAliasを作成する。これまで見てきた最初の2つの解決策と比べて、この解決策を使ったときの利点は、やっかいなチルダ文字がもはやURLに含まれないという点である。このチルダをプロにふさわしくないと思っている人も多い。こうすることで、今やユーザbacchusはhttp://www.example.com/bacchus/というURLで個人のWebスペースにアクセスできるようになる。

コードの先頭の%forbidリストは、何らかの理由でこの特殊なエイリアスを設定すべきではないユーザを並べたものだ。この機能を使うとセキュリティ上のリスクを引き起こすかもしれないユーザ(例えば、root)や、このような特権を与えるほど信用していないユーザを排除することができる。

これまでに見てきた例と同様に、この場合も、<Directory>セクションで、/home/*/public_htmlディレクトリを読み出し可能に設定しておく必要がある。

もちろん、ユーザのホームディレクトリではなく別の場所からコンテンツを提供したい場合には、その場所を指すようにエイリアスを作成するようにコードを変更すればよい。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_userdir.html

レシピ5.4　1つのディレクティブで複数のURLをエイリアスする

課題

複数のURLを同じディレクトリに対応付けたいが、Aliasディレクティブをいくつも書きたくない。

解決

httpd.confでAliasMatchディレクティブを使うと、正規表現でマッチングすることができる。

```
AliasMatch "^/pupp(y|ies)" "/www/docs/small_dogs"
AliasMatch "^/P-([[:alnum:]])([^/]*)" "/usr/local/projects/$1/$1$2"
```

解説

AliasMatchディレクティブには正規表現を使うことができ、URLを任意のパターンでマッチさせ、そのパターンを望みのURLに対応付けることができる。少し自由度の高いAliasだと考えればよい。

最初のAliasMatchは、/puppyあるいは/puppiesで始まるURLを、ディレクトリ/www/docs/small_dogsに対応付ける。2番目のAliasMatchは、リクエストされたプロジェクトのURLを、プロジェクト名の頭文字で分類したプロジェクトディレクトリにうまく対応付けるように作られている。例えば、プロジェクトExampleのURIが/P-Example/だとすると、このURIはディレクトリ/usr/local/projects/E/Example/に対応付けられることになる。

Apacheの正規表現については、付録Aで詳しく解説している。

参照

- 付録A
- 『詳説正規表現　第3版』（オライリー・ジャパン発行、原書『Mastering Regular Expressions』Jeffrey Friedl著、O'Reilly&Associates発行）

レシピ5.5 複数のURLを同じCGIディレクトリに対応付ける

課題

たくさんのURLを同じCGIディレクトリに対応付けたいが、ScriptAliasディレクティブをいくつも書きたくない。

解決

httpd.confでScriptAliasMatchディレクティブを使うと、正規表現でマッチングすることができる。

```
ScriptAliasMatch "^/[sS]cripts?|cgi(-bin)?/" "/www/cgi-bin/"
```

解説

これは前のレシピよりも複雑なので、付録Aを読む必要があるかもしれない。このレシピにおいて、ScriptAliasMatchディレクティブは、/script/、/scripts/、/Script/、/Scripts/、/cgi/、/cgi-bin/で始まるリクエストを、ディレクトリ/www/cgi-bin/に対応付けている。このディレクトリにあるファイルはすべてCGIプログラムとして扱われる。

こうしたディレクティブは、ディレクトリ構造の混乱を解消するために使われている。Webサイトを最初からうまく設計していれば、このようなものは不要だろう。しかし、Webサイトの再設計や再配置を初めて行うときには、このような曲芸が必要になることもある。

参照

- レシピ5.4
- 付録A

レシピ5.6 ユーザごとにCGIディレクトリを作る

課題

メインサーバのCGIディレクトリにアクセスするのではなく、ユーザごとに独自のcgi-binディレクトリが持てるようにしたい。

解決

httpd.confに以下を追加すればよい。

```
<Directory "/home/*/public_html/cgi-bin/">
    Options ExecCGI
    SetHandler cgi-script
</Directory>
ScriptAliasMatch "/~([^/]+)/cgi-bin/(.*)" "/home/$1/public_html/cgi-bin/$2"
```

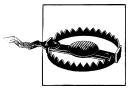
解説

これを実現するのに、ScriptAliasディレクティブは使えない。ユーザごとにScriptAliasの最初の引数が異なってしまうためだ。<Directory> コンテナと ScriptAliasMatch ディレクティブは、機能的には等価である。

このレシピを使うと、ユーザは自分のWebスペースにCGIスクリプトを置くことができるようになる。次の文字列で始まる URL にアクセスすると、そのファイルは CGI スクリプトとして扱われる。

```
http://www.example.com/~username/cgi-bin/
```

suexecが有効になっていれば、このターゲットディレクトリで実行されるCGIスクリプトは、URL中に含まれるユーザの権限で動作する。例えば、http://www.example.com/~rbowen/cgi-bin/example.cgiというURLでアクセスされた CGI プログラムは、ユーザ *rbowen* の権限で動作することになる。



ユーザが独自のスクリプトを設定することができ、誰のチェックもなく、自動的に実行するようにしてあると、問題を引き起こすおそれがある。悪意を持ったユーザ(そう思いたくはないが)がいたり、セキュリティホールのあるスクリプトがあるかもしれない。

参照

- レシピ 8.1

レシピ 5.7　別の場所にリダイレクトする

課題

特定の URL 宛てのリクエストを、別のサーバにリダイレクトしたい。

解決

httpd.conf で Redirect ディレクティブを使って、2 番目の引数に絶対 URL を指定すればよい。

```
Redirect "/example" "http://www2.example.com/new/location"
```

解説

Alias が URL をローカルファイルシステム上のリソースに対応付けるのに対し、Redirect は通常、URL を別のサーバ上のURLに対応付ける。2番目の引数は完全なURLであり、クライアント(ブラウザ)に送り返される。クライアントはこの新しい URL 宛てに次のリクエストを送る。

Redirectディレクティブはパス情報を保持しているということを知っておく必要がある。したがって、このレシピを使うと、http://original.example.com/example/something.html 宛てのリクエストは、http://other.example.com/new/location/something.html にリダイレクトされることになる。

リダイレクトには味付けが何種類かある。Redirectディレクティブと最初のURL引数との間に、適切なキーワードを挿入すると、リダイレクトの種類を指定することができる。いずれのリダイレクトも、クライアントがリクエストしたドキュメントが現在どこにあるのかを教えるものだ。リダイレクトの種類は、将来どこ

でドキュメントを探したらよいかをクライアントに教えている。何もキーワードが指定されていないときには、デフォルトの temp が使われる。

temp
: temporary リダイレクトは、そのドキュメントは現在はリクエストされた場所にはないが、いつかまたその場所に戻ることが期待できるときに利用する。クライアントは元のリクエストで使ったURLを覚えておき、将来、同じドキュメントをリクエストするときに利用する。

permanent
: permanent リダイレクトは、クライアントがリクエストしたドキュメントがその場所にはなく、もうその場所を探すべきではないということをクライアントに示している。言い換えると、クライアントはリダイレクトレスポンスに示された新しい場所を覚えておき、このリソースに対する今後のリクエストは、すべてその新しい場所宛てにすべきであるということだ。

gone
: このキーワードは、ドキュメントがもうその場所には存在せず、もはやリクエストすべきではないということを意味している。これは、404 Not Found エラーレスポンスとは違っている。gone リダイレクトは、ドキュメントはもうそこにはないが、以前はそこあったということを認めている。

seeother
: seeotherリダイレクトは、元のドキュメントはもうそこにはなく、別の場所の別のドキュメントに置き換わっている、ということをクライアントに伝える。例えば、クライアントが次の URL をリクエストしたとする。

    ```
    http://example.com/chapter2.html
    ```

  しかし、サーバは seeother リダイレクトを使って、次のように応答したとする。

    ```
    http://bookname.com/edition-2/chapter2.html
    ```

  これは、目的とするコンテンツが別の URL から取得できること、そして、そのコンテンツを取得するには 2 番目のリクエストをすべきであることを示している。

これは、303 See otherステータスコードの動作を本当によく理解しているときにだけ使うのがよいだろう。seeotherリダイレクトが適切かどうかわからなければ、代わりにtemporaryリダイレクトを使えばよい。

何もキーワードを指定しなかったときには、デフォルトでtemporaryリダイレクトになる。以下に、いろいろなディレクティブ形式の例を示す。ErrorDocument を使ってサーバのレスポンスをカスタマイズしたいときのため、HTTP ステータスコードも一緒に示した。

```
#
# 次の 3 つは同じ意味であり、ステータスコード 302 を返す。
#
Redirect      /foo.html http://example.com/under-construction/foo.html
Redirect temp /foo.html http://example.com/under-construction/foo.html
RedirectTemp  /foo.html http://example.com/under-construction/foo.html
#
# 次の 2 つは同じ意味であり、ステータスコード 301 を返す。
#
Redirect permanent /foo.html http://example.com/relocated/foo.html
RedirectPermanent  /foo.html http://example.com/relocated/foo.html
#
# 以前のURLはなくなってしまったが、そのコンテンツは指定した新しいドキュメントに置き換わった、ということをクライアントに教える。
# ステータスコード 303 を返す。
#
Redirect seeother /foo.html http://example.com/relocated/bar.html
#
# そのドキュメントは意図的に削除され、もうそこには戻ってこないことをクライアントに教える。ステータスコード 410 を返す。
# 2 番目の絶対 URL の引数がないことに注意すること。
#
Redirect gone     /foo.html
```

参照

- http://httpd.apache.org/docs/2.2/mod/mod_alias.html

レシピ 5.8　複数の URL を同じ宛先にリダイレクトする

課題

複数の URL を同じ宛先にリダイレクトしたい。例えば、/fish と /Fishing 宛てのリクエストを http://fish.example.com/ にリダイレクトしたい。

解決

httpd.conf で RedirectMatch ディレクティブを使って、正規表現でマッチングさせる。

```
RedirectMatch "^/[fF]ish(ing)?(/.*)?" "http://fish.example.com/$2"
```

解説

このレシピでは、fish や fishing で始まる URL 宛てのリクエストを、fish.example.com という別のサーバの URL へリダイレクトする。"f" は大文字でも小文字でも構わない。Redirect と同様に、パス情報があれば保持される。例えば、http://original.server/Fishing/tackle.html 宛てのリクエストは、相対リンクはそのまま維持されて、http://fish.example.com/tackle.html にリダイレクトする。

これまでに紹介した例と同様に、RedirectMatchも、任意のテキストをパターンマッチングするのに正規表現を使うことができる。

参照

- 付録A

レシピ5.9 URLで大文字と小文字を区別しないようにする

課題

リクエストするURLに、大文字と小文字のどちらを使っても構わないようにしたい。

解決

URLで大文字と小文字を区別しないようにするには、mod_spelingモジュールを使うとよい。

```
CheckSpelling On
```

解説

mod_spelingモジュールはApacheの標準ディストリビューションの一部であるが、デフォルトでは無効になっている。そのため、使うときには明示的に有効にする必要がある。

mod_spelingは、URLで大文字と小文字を区別しないようにするだけではなく、その名前が示しているように、簡単なスペルチェック機能も提供している。特に、「Not Found(ページが見つからない)」というエラーになる場合、mod_spelingはユーザが意図したファイルを何とか見つけようとする。似たようなスペル、文字の入れ替え、似ているようにみえる数字と文字(Oと0や、1と1など)の入れ替えなどをして、ファイルを探そうとする。

mod_spelingをインストールすると、特定のスコープ(ディレクトリやバーチャルホスト、サーバ全体など)でCheckSpellingディレクティブをOnに設定すれば、そのスコープでスペルチェック機能が有効になる。

なお、mod_spelingというモジュール名のスペルはこれで正しい(英語としては、spellingが正しいが)。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_speling.html

レシピ5.10 シンボリックリンクを使わずにPHPソースをハイライトする

課題

シンボリックリンクを設定せずに、PHPスクリプトのソースをシンタックスハイライトして表示したい。

解決

httpd.confや.htaccessファイルに次の行を追加する。

```
RewriteRule "^(.+\.php)s$" "$1" [T=application/x-httpd-php-source]
```

Apache バージョン 2.2 以降であれば次のように設定する。

```
RewriteRule "^(.+\.php)s$" "$1" [H=application/x-httpd-php-source]
```

別の解決

httpd.conf ファイルに次の行を追加する。

```
RewriteRule "^(.*\.php)s$" "/cgi-bin/show.php?file=$1" [PT,L]
```

そして、次のような show.php という名前のファイルを作成し、サーバの /cgi-bin/ ディレクトリに置く。

```
<?php
/*
 * シンボリックリンクやコピーをせずに、PHP スクリプトのソースをハイライト表示する
 */
if ((!isset($_GET))
    || (!isset($_GET['file']))
    || (!($file = $_GET['file']))) {
    /*
     * 必要な引数がなければ、ここから抜ける
     */
    return status('400 Bad Request',
        "Data insufficient or invalid.\r\n");
}

$file = preg_replace('/\.phps$/', '.php', $file);
if (!preg_match('/\.php$/', $file)) {
    return status('403 Forbidden',
        "Invalid document.\r\n");
}
$docroot = $_SERVER['DOCUMENT_ROOT'];
if ((!preg_match(";^$docroot;", $file))
    || (!preg_match(";^/home/[^/]+/public_html;", $file))) {
    return status('403 Forbidden',
        "Invalid document requested.\r\n");
}
Header('Content-type: text/html; charset=iso-8859-1');
print highlight_file($file);
return;

function status($msg, $text) {
    Header("Status: $msg");
```

```
    Header('Content-type: text/plain; charset=iso-8859-1');
    Header('Content-length: ' . strlen($text));
    print $text;
}
?>
```

解説

PHPインタプリタには、PHPソースコードのシンタックスを色付けして表示する組み込み関数がある。通常、設定ファイルに次の行を追加すると、.phps ファイルに対して呼び出すことができる。

```
AddHandler application/x-httpd-php-source .phps
```

しかし、この機能を使うためには、PHPファイルをコピーするかシンボリックリンクを作成して、.phpファイルの拡張子を .phps に置き換えなければならない。これはちょっと現実的ではないし、不便だ。

このレシピを使うとこの作業が不要になる。拡張子が.phpsのファイル宛てのリクエストを、拡張子が.phpの同名のファイルに書き換えて、php-source ハンドラをこのリクエストに関連付けている。

RewriteRuleディレクティブの[H]フラグはバージョン2.2で新しく追加されたものだ。以前のバージョンでは、代わりに、[T]フラグを使わなければならない。これらは同じ機能を提供している。

別の解決にあるスクリプトでは、スクリプトのソースをハイライトして表示するのに、PHPの組み込み関数を使っている。最初の解決策が1行の変更で済むのと比べると複雑だが、サーバを再起動せずに、スクリプトを変更するだけでよい。$docrootに対してpreg_match関数を実行して、リクエストされたファイルがサーバのDocumentRoot以下にあるかどうか確認している。次のpreg_match関数は、ファイルがユーザのpublic_htmlディレクトリにあっても許可するのに使われている。

参照

- レシピ 2.5

レシピ 5.11 リクエストされた URL のテキストを書き換える

課題

リクエストされた URL に含まれる *string1* をすべて *string2* に変更したい。

解決

```
RewriteRule "(.*)string1(.*)" "$1string2$2" [N,PT]
```

解説

[N]フラグを付けると、Apache は書き換えルールを再実行する。このルールは、RewriteCond が失敗するまで繰り返し実行される。つまり、URL中に置き換えたい文字列が含まれている限り、再実行が繰り返されることになる。文字列がすべて置き換えられると、RewriteCond が失敗して、ルールの実行も停止する。[PT]フ

ラグを付けると、mod_rewrite は、書き換えられた URL を Apache に渡し、その後の処理は書き換えられた URL に対して実行される。

無限ループにならないように注意しよう。例えば、*string1* が *string2* の一部になっていると無限ループになってしまう。

参照

- 付録 A

レシピ 5.12　パス情報を CGI 引数に書き換える

課題

URL の一部として引数を渡したいが、その URL の要素を CGI の QUERY_STRING 引数に書き換えたい。

解決

これは単なる例であり、環境や必要性に合わせて、RewriteRule の行を適切に変更する必要がある。

```
RewriteEngine on
RewriteRule "^/book/([^/]*)/([^/]*)" "/cgi-bin/book.cgi?subject=$1&author=$2" [PT]
```

解説

これは、異なる処理方法をする2つの古くなったシステム（例えば、クライアントアプリケーションやベンダのスクリプト）を 1 つに繋げるようなときに、有効な方法だ。

例えば、この解決策の RewriteRule は、次の URL

```
http://www.example.com/book/apache/bowen
```

を、次のように書き換える。

```
http://www.example.com/cgi-bin/book.cgi?subject=apache&author=bowen
```

RewriteRule ディレクティブに付けられた[PT]フラグは、Apache に URL の変更後も処理を継続するように指示している。このフラグを付けないと、サーバは書き換えられたURLを直接ファイル名として処理しようとしてしまい、CGI スクリプトとして処理されなくなる。[PT]フラグを付けると、複数の RewriteRule ディレクティブを使って、URL をさらに書き換えることもできる。

URL に元からクエリ文字列が含まれている場合には、[PT]を[QSA,PT]に変更しなければならない。QSA とは「クエリ文字列の追加（"Query String Add"）」を意味しており、書き換えで作ったクエリ文字列を元の URL のクエリ文字列に追加することができる。QSAを付けないと、元のクエリ文字列は新しいものに置き換えられてしまう。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.13 他のサイトからのアクセスを拒否する

課題

自分のサイトの画像(あるいは、ドキュメントなど)が他のサイトのページで使われないようにして、自分のサイトから参照されたときだけアクセスできるようにしたい。

解決

httpd.conf に次の設定を追加する。

```
RewriteEngine On
RewriteCond "%{HTTP_REFERER}" !=""
RewriteCond "%{HTTP_REFERER}" "!^http://mysite.com/.*$" [NC]
RewriteRule "\.(jpg|gif|png)$ - [F]
```

解説

このレシピは、一連のRewriteCondディレクティブを使って、画像が自分のサイトにあるドキュメントの中からリクエストされているのか、別のサーバにあるページに組み込まれているのかを判断している。もし別のサーバにあるページに組み込まれていれば、他のサイトが自分の画像を盗もうとしていると考えられるので、阻止したい。

最初のルールは、Referer(参照元)が設定されているかどうかをチェックする。Referer情報を送らないクライアントもあるし、Referer情報を送らないように設定できるブラウザもある。Referer情報を送らないすべてのクライアントからのリクエストを拒否してしまうと、たくさんの正当なリクエストまで拒否してしまうことになる。そのため、ここではこのケースを許容している。

次に、参照元が自分のサイト以外の外部サイトからかどうかをチェックする。外部のサイトからであれば、ルールの適用を続ける。内部のサイトからであれば、書き換え処理を中断する。

最後に、そのリクエストが画像ファイルに対するものかどうかをチェックする。画像以外のファイル、例えば、HTML ファイルに対してであれば、外部からのリンクを許可する。

この最後のルールに到達すると、ようやく他のWebサイトのページから画像ファイルがリクエストされているということがわかる。RewriteRule がこのリクエストにマッチすると、クライアントに Forbidden(リクエスト拒否)を返す。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html
- レシピ 5.14

レシピ5.14　Refererのないリクエストを説明ページにリダイレクトする

課題

レシピ5.13で説明したように、Referer（参照元）URLのないリクエストがサーバにやってきたら、なぜリクエストを受け付けられないか説明するページにリダイレクトしたい。

解決

設定ファイルに次の行を追加する。

```
RewriteEngine On
RewriteCond "%{HTTP_REFERER}" "^$"
RewriteRule "(.*)" "/cgi-bin/need-referer" [PT,E=ORIG:$1]
```

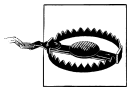
これらのディレクティブは、適切なスコープに置くよう注意すること。さもないと、そのサイトのURLをブラウザに入力しても、誰もアクセスできなくなる。

解説

この解決策で解決しようとしている問題は、どこでURLを得たかという情報（Refererリクエストヘッダフィールドに渡される）を付けずにサーバにやってきたリクエストだ。リンクをたどってきたのではなく、誰かがページに直接アクセスしたことを示している場合も多いし、画像のinformaltableの場合にはユーザが"Save As"を実行した結果かもしれない。自分のサイトのリンクをたどってきたリクエストだけアクセスできるようにしたければ、RewriteCondディレクティブを次のように変更すればよい。

```
RewriteCond "%{HTTP_REFERER}" "!http://%{SERVER_NAME}/" [NC]
```

RewriteRuleは、リクエストをneed-refererという名前のCGIスクリプトに渡し、アクセスされたURIを環境変数ORIGに設定して、スクリプトがURIを参照できるようにしている。このとき、URIのローカル部分だけが含まれる（ホスト名は含まれない）ことに注意すること。



Refererリクエストヘッダフィールドがないことは、必ずしも不正なアクセスを意味するわけではない。どうやってそこにたどり着けいたかは知ったことではなく、たどってきた経路を記録されたくない人もいる。しかし、Apacheにそのヘッダフィールドが省略された理由を教えるすべはない。need-refererスクリプトを使うと、その状況を説明することができる。

参照

- レシピ5.13

レシピ 5.15 クエリ文字列に基づいて書き換える

課題

ある URI をクエリ文字列の値に基づいて、別の URI に書き換えたい。

解決

httpd.conf に以下を追加する。

```
RewriteCond "%{QUERY_STRING}"   "^user=([^=]*)"
RewriteRule "/people"           "http://%1.users.example.com/" [R]
```

解説

mod_rewrite のマッチングと書き換え処理では、クエリ文字列を URI の一部と見なさいため、別々に扱う必要がある。この例では、リクエストを次のように変換する。

```
http://example.com/people?user=jones
http://jones.users.example.com/
```

[R]フラグを付けると、mod_rewrite は、RewriteRule ディレクティブが作った URL にリダイレクトするよう指示する。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_alias.html

レシピ 5.16 サーバのすべてあるいは一部のリクエストを SSL にリダイレクトする

課題

SSL で保護されていない Web スペースの一部を、SSL で保護された安全な領域へリダイレクトしたい。

解決

次の RewriteRule を使うと、ポート番号 80 に対するすべてのリクエストを SSL で保護された領域へリダイレクトすることができる。

```
RewriteCond "%{SERVER_PORT}"       "^80$"
RewriteRule "^(.*)$"               "https://%{SERVER_NAME}$1" [R,L]
```

特定の URL だけを、SSL で保護されたバージョンにリダイレクトするには、次のように設定すればよい。

```
RewriteRule "^/normal/secure(/.*)" "https://%{HTTP_HOST}$1" [R,L]
```

次のように設定すると、環境変数HTTPSが設定されているかどうかをチェックすることができる。

```
RewriteCond "%{HTTPS}" "!=on"
RewriteRule "^(/secure/.*)" "https://%{HTTP_HOST}$1" [R,L]
```

または、単に、httpd.confファイルのhttpセクションで、次のようにRedirectディレクティブを使えばURLをHTTPSとして扱うことができる。

```
Redirect "/" "https://secure.example.com/"
```

httpsではなく、httpのスコープ内だけで設定するよう注意すること。さもないと、すべてのhttpsリクエストがループしてしまうことになる。

解説

最初の解決策は、ポート80（通常、暗号化されていないHTTPポート）にやってきたすべてのリクエストを、同じサーバの同じ場所に、SSLを通してアクセスするようリダイレクトする。ここでは、SERVER_NAMEを使っていることに注意してほしい。サイト全体のリダイレクトなので、サーバの公式な名前を使うのが最も簡潔だ。

2番目の解決策にあるディレクティブは、http://myhost/normal/secureにあるサーバのWebスペース全体を、https://myhost/ をルートとするSSLで保護されたWebスペースにリダイレクトする。ここでは、SERVER_NAMEではなくHTTP_HOSTを使っており、これはサーバ名ではなく、訪問者のブラウザ上に表示されているロケーションとスキームを意味している。

HTTP_HOSTを使った場合、ポート番号が含まれていると、URLがおかしくなってしまうおそれがあることに注意しよう。HTTP_HOSTにポート番号が含まれる可能性があるなら、ルールの中で、これを補正する必要がある。

SSLとSSLでない場所に対するパスが違っていることに注意しよう。セキュリティの違いを除いて、パスを同じにしたければ、3番目の解決策で示したようにディレクティブを使うとよい。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ5.17　ディレクトリをホスト名に変換する

課題

1つのホスト名にある複数のパス名を、別々のホスト名に変換したい。

解決

httpd.conf で RewriteRule を使って、次のように設定すればよい。

```
RewriteRule "^/(patha|pathb|pathc)(/.*)" "http://$1.example.com$2" [R]
RewriteRule "^/([^./]*)(/.*)" "http://$1.example.com$2" [R]
RewriteRule "^/~([^./]*)(/.*)" "http://$1.example.com$2" [R]
```

解説

最初のレシピは、http://example.com/*pathseg*/some/file.html からのリクエストのうち、*pathseg* が patha、pathb、pathc のリクエストだけを別のホスト、例えば、http://*pathseg*.example.com/some/file.html にリダイレクトする。

2番目のレシピも同様のことをするが、最上位レベルのすべてのパスセグメントをリダイレクトする。

3番目のレシピはその中間で、すべての「ユーザ」へのリクエストを、そのユーザと同じ名前の個別のホストにリダイレクトする。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ5.18 すべてのリクエストを1つのホストにリダイレクトする

課題

システムにやってくるすべてのリクエストを特定のホストにリダイレクトしたい。

解決

httpd.conf に以下を追加する。

```
RewriteCond "%{HTTP_HOST}"   "!^www.example.com"        [NC,OR]
RewriteCond "%{SERVER_NAME}" "!^www.example.com"        [NC]
RewriteRule "(.*)"           "http://www.example.com$1" [R]
```

解説

この解決策では、このディレクティブで指定した範囲内のサーバで処理するすべてのリクエスト（www.example.com ホスト宛てを除く）を、www.example.com にリダイレクトする。

2つの RewriteCond ディレクティブは、リダイレクト方法に関係なく、www.example.com 以外のホスト宛てのすべてのリクエストを捕捉するのに使われる。NC（No Case）フラグを付けることで、正規表現において大文字と小文字を区別しないようにしている。つまり、文字が大文字であっても小文字であってもマッチする。

OR フラグは論理和を意味しており、結びついた2つの条件のうちどちらか一方が真であれば、条件が満たされてルールを適用する。

最後に、R フラグを指定することでリダイレクトするよう指示し、ブラウザはここで生成された URL に対

して次のリクエストを送ることになる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.19　ドキュメント名を引数に変換する

課題

ドキュメントに対するリクエストをCGIスクリプトやその他ハンドラにリダイレクトして、その引数としてドキュメント名を与えたい。

解決

httpd.conf で RewriteRule を使って、次のように設定する。

```
RewriteRule "^/dir/([^./]*)\.html" "/dir/script.cgi?doc=$1" [PT]
```

解説

この解決策は、指定した場所にあるHTMLドキュメント宛てのすべてのリクエストを、ハンドラスクリプトへのリクエストに変換する。このとき、スクリプトは、QUERY_STRING環境変数の引数としてドキュメント名を受け取ることができる。

この後に続く URL の書き換えや操作を正しく実行するために、PT フラグを付けておく必要がある。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.20　パスとクエリ文字列間で要素を書き換える

課題

URL の一部をクエリ文字列に、あるいは逆に、クエリ文字列を URL の一部に書き換えたい。

解決

http://example.com/path/to/5 を http://example.com/path/to?id=5 に書き換えるには、次のように設定すればよい。

```
RewriteRule "^(/path/to)/(\d+)" "$1?id=$2" [PT]
```

逆に、http://example.com/path/to?id=5 を http://example.com/path/to/5 に書き換えるには、次のように設定すればよい。

```
RewriteCond "%{QUERY_STRING}" "\bid=(\d+)\b"
RewriteRule "(/path/to)" "$1/%2" [PT,QSA]
```

.htaccessファイルに書く場合、RewriteRuleディレクティブに与える引数はこれとは違う構造になる。この解決策では、サーバ全体の設定ファイルに書く場合の構文を示している。

解説

URLの一部をクエリ文字列に変えたり、クエリ文字列をURLの一部に変えたりするなど、サイトのURL構造を変更しなければならないことがよくある。例えば、ブログのソフトウェアパッケージを別のものに変えたときなど、基盤となるソフトウェアを変えたときに必要になることが多い。新しいソフトウェアでは、URLを以前と違う形式にする必要があるが、以前の形式のURLを無効にはしたくない。以前のURLは、ユーザにブックマークされていたり、Webで広まっているかもしれないためだ。

mod_rewriteを使うと、この問題を完璧に解決してくれる。実現できるのはもちろん、実演してみせるのも非常に簡単だ。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.21 ホスト名をディレクトリに書き換える

課題

http://bogus.example.com/ 宛てのリクエストを、http://example.com/bogus/ に書き換えたい。

解決

httpd.confでRewriteRuleを使って、次のように設定すればよい。

```
RewriteCond "%{HTTP_HOST}" "^([^.]+)\.example\.com" [NC]
RewriteRule "(.*)" "http://example.com/%1$1" [R]"
```

リダイレクトをせずに、これを透過的に実現することもできる。

```
RewriteCond "%{HTTP_HOST}" "^([^.]+)\.example\.com$" [NC]
RewriteRule "(.*)" "/%1$1" [PT]
```

解説

このテクニックは、ワイルドカードのホスト名をサポートするときに必要になることがある。ワイルドカードを使うと、1つのシステムにもかかわらず、URLがサブディレクトリではなく別のホストを指しているような錯覚を与えることができる。もちろん、リダイレクト([R])フラグを使うと、錯覚が無効になってしまうだろう。書き換えたURLがエンドユーザに見えてしまうためだ。ユーザに対して完全に透過的に見せたいので

あれば、2番目の解決策を使うとよい。クライアントからは決して見えないようにサーバ内部で書き換えて、同じ効果を得ることができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 5.22　URL セグメントをクエリ文字列に書き換える

課題

次のリクエストを、

http://example.com/foo/bar.html

次の URL 宛てに書き換えたい。

http://example.com/cgi-bin/remap?page=foo/bar.html

解決

設定ファイルに次の行を追加する。

```
RewriteRule "/(.*)" "/cgi-bin/remap?page=$1" [QSA,PT]
```

解説

この解決策では、ユーザからは完全に透過的に書き換えを実行する。変換はすべてサーバの内部で行われ、ユーザのクライアントに新しい URL をロードさせることはない。

QSA オプションを付けると、元のリクエストにあったクエリ文字列の情報を保持したまま、RewriteRule によって追加されたクエリ文字列をマージすることができる。PT オプションは、処理が完全に終了したのではなく、リクエストの処理を続行するよう、サーバに指示する。これは、新しいURLにおけるCGIスクリプトの呼び出しを正しく処理するのに必要だ。

参照

- レシピ 5.21

レシピ 5.23　AliasMatch、ScriptAliasMatch、RedirectMatch を使う

課題

Aliasディレクティブや Redirectディレクティブの基本機能を使いたいが、1つずつディレクティブを使うのではなく、関連するケースをまとめて処理したい。

解決

グルーピング構文を使って、URI をファイルシステムのパスに変換することができる。

```
AliasMatch "/users/(.*)/" "/home/webonly/$1/"
```

次のように設定すると、クライアントのブラウザに、移転先の新しい URL を覚えさせることができる。

```
RedirectMatch permanent "/projects/([^/]+)(/.*)" "http://$1.projectdomain.com$2"
```

解説

AliasMatch、ScriptAliasMatch、RedirectMatchディレクティブは、他のApacheの設定と同様の正規表現の構文を使うことができる。条件を作ったり環境変数を参照したりできないという点で、mod_rewrite ディレクティブほど強力ではないが、簡単な構文だからこそ、サーバのパフォーマンスに与える影響も小さくて済む。

参照

- レシピ 5.4
- レシピ 5.5

6章 セキュリティ

本章で取り上げるセキュリティとは、人に見せたいものを見せ、見せたくなものを見せないようにすることだ。さらに、Web以外の手段によるアクセスを制限するために、サーバにどんな対策をしなければならないか、という課題もある。本章では、悪意を持ったアクセスやWebサイトの改ざんからサーバを守るためにとるべき予防策を解説する。

最もよく聞かれる質問は、どうやってドキュメントを保護してアクセスを制限するのか、ということだ。残念ながら、このテーマは複雑で、Webアーキテクチャの性質からもこの質問の回答は込み入った話になり、手頃な回答がないことも多い。

セキュリティに関する標準的な用語と方法論では、アクセス制御に適用するプロセスを2つのステップに分割している。Web の場合には、サーバが次の 2 つの質問をすることに相当する。

- あなたは本当に本人ですか？
- あなたはここにいることを許可されていますか？

これらのステップは、それぞれ、「認証(Authentication)」、「許可(Authorization)」と呼ばれている。現実世界での例を挙げると、航空会社の係員が写真付きの身分証明書をチェックし(認証)、飛行機に搭乗する前にチケットをチェックする(許可)ということだ。

認証は、「弱い認証」と「強い認証」の2つに分類することができる。弱い認証は、エンドユーザが提示した証明書の正しさに基づいている(証明書は本当の持ち主から盗まれたものかもしれないので、「弱い」と名付けられている)。これに対して、強い認証は、エンドユーザがほとんど、あるいは全くコントロールできず、リクエストごとに変えられないような特性(例えば、システムの IP アドレスなど)に基づいている。

認証と許可のチェックは明らかに別の作業であるが、Apache Webサーバのモジュール環境では、その適用範囲が少し曖昧になっている。たくさんのセキュリティモジュールがあるが、その大きな違いは、証明書の格納方法(ファイルやデータベース、LDAPディレクトリなど)にある。それにもかかわらず、証明書を格納場所から取り出し、クライアントが提示した証明書を検証し、認証されたユーザにリソースへのアクセス許可があるかチェックする、といったコードもすべて各モジュールが提供しなければならない。つまり、モジュール間で多くの機能が重複していることになる。動作やコマンドもかなり似ているのだが、コードが共有されていないため、期待するほどは似ていないことも多い。Apache 2.0 以降では、こうした機能の重複はいくら

か解消されてきている(本書の執筆時点ではまだ開発中である)。

Webサーバのコンテンツにアクセスするときにパスワードが必要になるという課題に加えて、サーバを攻撃から守るというもっと大きな課題もある。他のソフトウェアと同様に、Apacheは、その歴史上何度となく、攻撃者がホストサーバのコントロールを不正に乗っ取ることが可能な状態になっていたことがある。例えば、サイトの管理者がアクセスを許可したつもりのないファイルに攻撃者がアクセスして変更できることもあったし、攻撃者がターゲットサーバ上でコマンドを実行できることもあった。したがって、こうした攻撃を受けないよう、どんな対策をとる必要があるのか理解しておくことが重要だ。

最も重要な対策は、新しいリリースの通知を受け取り、CHANGESファイルをよく読んで、新しいバージョンで自分が影響を受けるかもしれないセキュリティホールが修正されているかどうか確認することだ。Apacheサーバの最新バージョンを使うことは、セキュリティの脆弱性と戦う上で、一般的に実施しておくことが望ましい対策だ。

本章のレシピでは、必要度の高いパスワード手順の実装方法とともに、外部の攻撃からサーバを守るために必要なツールを紹介する。

認証と許可

機密文書にアクセスできるかどうかをチェックするとき、実際には２つの異なる作業が必要になる。１つは、ユーザが誰であるかを確認することで、もう１つは、そのユーザにそのドキュメントを閲覧する許可があるかどうかを確認することである。

最初の部分はユーザが誰であるかを確認することで、認証と呼ばれる。Webサーバはユーザが誰であるかを知らないので、ユーザ名とそれに対応するパスワードのような身元を証明する情報を提示する必要がある。サーバは提示された情報(証明書と呼ぶ)をデータベースにある情報と比較し、一致すれば処理を続行する。データベースに存在しなかったり、情報が一致しなかったりする場合には、サーバはエラー状態を返してそのユーザを拒否する。

サーバがユーザ本人だと確認すると、そのドキュメントにアクセスすることが許可されているユーザのリストを調べて、このユーザがリストに載っているかどうか確認する。これは許可と呼ばれる。ユーザがリストに載っていれば、サーバは通常通りアクセス処理を続行する。リストに載っていなければ、サーバはエラー状態を返してそのアクセスを拒否する。

返すエラーについては、この2つの異なる作業で違いはない。認証に失敗したときも、許可されていないときも、同じ401コード(許可されていない)が返される。これは、証明書が正しかったのかどうか攻撃者にばれることを防ぐためである。

レシピ 6.1　システムのアカウント情報を Web 認証に利用する

課題

Unix 系システム上のすべてのユーザが、すでに割り当てられているユーザ名とパスワードを使って、Web 上で認証できるようにしたい。

解決

mod_auth モジュールを使って領域を設定し、AuthUserFile に /etc/passwd を指定すればよい。

```
<Directory "/home">
    AuthType Basic
    AuthName HomeDir
    AuthUserFile /etc/passwd
    Require valid-user
    Satisfy All
</Directory>
```

解説

まず、SSLを使ってサイトを保護していない限り、システムのアカウント情報を使ってWeb認証をするのはとてもまずい考えだ、ということを強調しておきたい。その理由の 1 つは、ユーザの証明書を偶然手に入れた侵入者が、Web上で保護されたファイルにアクセスできるだけでなく、実際にシステムにログインして深刻な被害を与えることができるためだ。もう 1 つの理由は、Web 上のログインは、たいていの OS が備えているほどのセキュリティ制御ができるわけではないためだ。Web上では、侵入者がユーザ名のパスワードをしつこく何度も変えて攻撃しても、システムは何も防御対策をとらない。mod_auth モジュールが Apache のエラーログにメッセージを記録するだけだ。しかし、たいていのOSでは、paranoidモードに入り、少なくとも何度かログインに失敗した後、しばらくの間はログインができないようになっている。

リスクを許容できると考えていたり、自分の状況には当てはまらないために、まだシステムのアカウント情報を使って Web 認証を行いたいと言うのであれば、この解決策にあるようなディレクティブを httpd.conf に追加すればよい。mod_auth が使う証明書レコードのフィールドの構文と順序は、/etc/passwd の 1 行の標準レイアウトと同じだ(これは偶然ではない)。mod_authは単純なテキストファイル形式を使っており、各行はユーザ名とパスワードで始まり、オプションとして追加フィールドを含めることができる。各フィールドはコロンで区切られている。例えば、次のような行になる。

```
smith:$apr1$GLWeF/..$8hOXRFUpHhBJHpOUdNFe51
```

mod_authは、パスワードの後の追加フィールドをすべて無視するので、/etc/passwdファイルをそのまま使うことができる。この例にあるパスワードは、暗号化されていることに注意しよう。

htpasswdユーティリティを使うと、mod_authの証明書ファイルを管理することができるが、このhtpasswdを/etc/passwdファイルに使ってはいけない。/etc/passwdには通常のアカウント管理ツールを使う必要

がある。

このテクニックはシャドウパスワードを使っているときには使えないことに注意しよう。/etc/passwdのパスワードフィールドには、パスワード情報が入らないためである。代わりに、パスワードは/etc/shadowファイルに格納されており、rootユーザしか読み出せなくなっている。Apacheは非特権ユーザとして動作しているため、このファイルを読み出すことはできない。最近のUnix系OSはたいてい、デフォルトのユーザ認証に /etc/shadow を使っている。

参照

- コラム「認証と許可」
- コラム「HTTP とブラウザと証明書」
- コラム「弱い認証と強い認証」
- htpasswd の manpage
- passwd(5)の manpage

レシピ 6.2　1 度だけ使えるパスワードを設定する

課題

訪問者がサイトに 1 度だけログインできるような証明書を提供したい。

解決

Apache の標準機能で使える解決策はない。

解説

コラム「HTTP とブラウザと証明書」で説明するように、サイトに「ログイン」しているように見えるのは錯覚だ。望むような、1度限りという効果を出すには、サーバは次のようなステップをすべて満たす必要がある。

1. ユーザが最初に正当な証明書を提示した事実を記録しておく。
2. 何らかの方法で、その事実をユーザの「セッション」と関連付ける。
3. セッション情報が最初に成功したときと違っていれば、以後、その証明書は 2 度と受け入れない。

最後のステップはそう簡単な作業ではなく、Apacheの標準ディストリビューションに備わっている機能ではない。問題をさらに複雑にしているのは、いったん証明書の確認が成功すると、タイムアウトを開始する必要があることだ。こうすることで、ユーザがいったん認証に成功した後、そのブラウザのセッションを何日間か開いたままにしても、アクセスが維持されないようにする。

この要求を満たすには、独自の解決策が必要になる。残念ながら、筆者の知る限り、このような機能を提供するオープンな公開モジュールは存在していない。しかし、モジュールのレジストリを探してみると、サー

ドパーティ製の実装が見つかるかもしれない。

参照

- レシピ 6.3
- http://modules.apache.org

HTTP とブラウザと証明書

Webの動作について、よく誤解されていることがある。あるページをブラウザで表示している間、そのサイトに接続していると思ってしまうのは自然なことだ。しかし、実際にはそうではない。ブラウザがサーバからページを取得すると、サーバもブラウザもコネクションを切断し、お互いを忘れてしまう。リンクをたどったり、同じサーバの別のページをリクエストしたりすると、全く新しいやり取りが始まる。

よく考えると、これは至極当然のことだ。ユーザがランチに出かけたり、帰宅したりしている間も、ブラウザがサーバに接続し続けるのは意味がない。

各トランザクションが互いに独立していて無関係であるとき、このトランザクションは「ステートレス」であると言われる。これは、HTTP のアクセス制御がどのように機能するかに影響を与えている。

パスワードで保護されたページを訪れても、Webサーバはそのユーザが以前アクセスしたことがあるかどうかを覚えていない。クライアント(ブラウザ)とサーバがお互いに話をしているHTTPレベルで考えると、クライアントは毎回、自分が誰であるかを証明しなければならない。自分の情報を覚えているのはクライアントなのである。

あるセッションで、保護された領域に初めてアクセスすると、実際にはクライアントとサーバで次のようなやり取りが行われている。

1. クライアントがページを要求する。
2. サーバは次のように応答する。「このリソースにアクセスする許可が与えられていない(`401 Unauthorized`ステータス)。このリソースは認証領域 *XYZ* にある。」(この情報は、`WWW-Authenticate` レスポンスヘッダフィールドにより伝えられる。詳しくは RFC 2616 を参照すること)
3. クライアントが対話型のブラウザでなければ、この時点でおそらくステップ7に進むことになる。対話型のブラウザであれば、ユーザにユーザ名とパスワードの入力を要求し、サーバが伝えた認証領域の名前を表示する。
4. クライアントはユーザから証明書を受け取ると、そのドキュメントに対してリクエストを再発行する。ただし、今度は証明書を付けたリクエストを送る。
5. サーバは受け取った証明書を調べる。証明書が有効なものであれば、アクセスを許可して、そのドキュメントを返す。証明書が無効なものであれば、ステップ 2 と同じレスポンスを返す。
6. クライアントが再び許可されていないというレスポンスを受け取ると、それに関するメッセージを表示し、ユーザに再度、ユーザ名とパスワードを入力するかどうかを尋ねる。ユーザがはいと答えると、クライアントはステップ 3 の処理に戻る。

7. ユーザがユーザ名とパスワードを再入力しないことを選択すると、クライアントはあきらめて、サーバからの 401 Unauthorized レスポンスを受け入れる。

クライアントはサーバとの認証に成功すると、その証明書とURL、それに関連する認証領域を覚えておく。そして今後、同じドキュメントやその下位にあるURL(例えば、/foo/bar/index.htmlは/foo/index.htmlの下位にある)をリクエストするときには、同じ証明書を自動的に送る。この場合、ステップ4から処理が始まるので、クライアントとサーバ間のチャレンジ／レスポンスのやり取りは依然として発生しているが、ユーザの目からは見えなくなっている。このため、ユーザがサイトに「ログイン」していると誤解してしまうのだ。

以上、HTTP の弱い認証がどのように機能しているかを説明した。多くの対話型の Web ブラウザには、クライアントを終了すると、証明書を忘れてしまうという共通の特徴がある。このため、ブラウザの新しいセッションでは、保護されたドキュメントにアクセスするたびに再認証が必要になる。

レシピ 6.3 パスワードを期限切れにする

課題

ユーザのユーザ名とパスワードを、ある特定の時刻あるいは一定期間経過後に期限切れにしたい。

解決

Apache の標準機能で使える解決策はないが、サードパーティ製の解決策がいくつかある。

解説

コラム「HTTP とブラウザと証明書」を参照してほしい。Apache でこの機能を実現するには、単なる有効なユーザ名とパスワード以上のものを格納しなければならない。証明書の有効期限のような情報も管理する必要があるだろう。Apacheの標準ディストリビューションには、このような機能を提供してくれるモジュールはない。

この問題に対して、サードパーティ製の解決策がいくつかある。例えば、Apache::Htpasswd::Perishableや、mod_perl ハンドラの Apache::AuthExpire などだ。

この問題には2つの少し違った見方があり、解決策の選び方に影響を与えるだろう。1つは、ユーザを認証してから一定時間経過した後、または、ある一定期間利用しなかった後にログアウトさせ、もう一度ログインを強制するというものだ。もう 1 つは、ある一定期間経過後に、特定のユーザ名／パスワードの組を完全に期限切れにして、2度と使えないようにするというものだ。後者は、1度限りの使い捨てパスワードの代わりに使われることがあるが、HTTP で実装するのは実用的ではない。

Apache::Htpasswd::Perishable は 2 番目の解釈を部分的に実装しており、パスワードファイルに有効期限情報を追加する。これはApache::Htpasswdを継承しており、 2つのメソッド、expireとexpandを追加している。expireはパスワードの有効期限を設定し、expandは有効期限を延長する。例えば、次のコードは、パスワードファイルを開いて特定のユーザエントリに有効期限を設定する。

```
use Apache::Htpasswd::Perishable;

my $pass = Apache::Htpasswd::Perishable->new("/usr/local/apache/passwords/user.pass")
    or die "Could not open password file.";
$pass->expire('waldo', 5); # 有効期限を5日後に設定する
```

このメカニズムを有効なものにするには、期限切れになったパスワードを定期的にパスワードファイルから削除する必要がある。これは、次のようなcronスクリプトを毎日走らせることで実現できる。このスクリプトは、有効期限の切れたユーザ情報を削除する。

```
#!/usr/bin/perl
use Apache::Htpasswd::Perishable;

my $password_file = '/usr/local/apache/passwords/user.pass';

open(F,$password_file) or die "Could not open password file.";
my @users;
while (my $user = <F>) {
    $user =~ s/^([^:])+:.*$/$1/;
    push @users, $user;
}
close F;

my $pass = Apache::Htpasswd::Perishable->new($password_file) or die
    "Could not open password file.";
foreach my $user (@users) {
    $pass->htDelete($user) unless $pass->expire($user) > 0;
}
```

これに対して、Apache::AuthExpireは「ログインセッション」のタイムアウトを実装している。一定期間利用しなければ、ユーザは再認証が必要になる。こうすると、ユーザが「ログイン中」のまま、しばらくコンピュータから離れても、ユーザを保護することができる。HTTPはステートレスであり、実際にはログイン中という概念はない。しかし、同じアドレスから繰り返し接続があることを監視することによって、ステートを擬似的に実現することができる。Apache::AuthExpireによる期限切れの機能を使うには、まずCPANからモジュールをダウンロードして、インストールする。

```
# perl -MCPAN -e shell
cpan> install Apache::AuthExpire
```

そして、このモジュールをApacheサーバが認証ハンドラとして使用するために、次のように設定する。

```
PerlAuthenHandler Apache::AuthExpire
PerlSetVar DefaultLimit 7200
```

ここで説明した例では、コネクションのアイドル状態が7200秒(すなわち2時間)経過すると、タイムアウトするようにしている。

参照

- レシピ 6.2
- http://modules.apache.org
- http://search.cpan.org/author/JJHORNER/Apache-AuthExpire/AuthExpire.pm
- http://search.cpan.org/author/ALLENDAY/Apache-Htpasswd-Perishable/Perishable.pm

レシピ 6.4 アップロードサイズを制限する

課題

ユーザがドキュメントをアップロードできるWebホスティングサービスが増えるにつれ、アップロードのサイズがあまりに巨大になってきた。少し創造力を働かせて、サーバのセキュリティ機能を使うことでアップロードサイズを制限したい。

解決

アップロードサイズを10,000バイトに制限したいとする。最も簡単なのは、LimitRequestBodyディレクティブを適切なスコープに追加することだ。

```
<Directory "/usr/local/apache2/uploads">
    LimitRequestBody 10000
</Directory>
```

ユーザがあまりに大きなリクエストを送ろうとすると、エラーメッセージを受け取ることになる。しかし、Apacheのエラーメッセージはデフォルトのままなので、ErrorDocument 413ディレクティブや、もっと複雑な(もっと自由度がある)次のような裏の手を使って、うまくメッセージを作るとよい。

```
SetEnvIf Content-Length "^[1-9][0-9]{4,}" upload_too_large=1
<Location /upload>
    Order Deny,Allow
    Deny from env=upload_too_large
    ErrorDocument 403 /cgi-bin/remap-403-to-413
</Location>
```

以下のような/cgi-bin/remap-403-to-413スクリプトを使うと、レスポンスをうまく作成することができる。

```
#!/usr/local/bin/perl
#
# 403 エラーを 413 に変換する Perl スクリプト
```

```
# アップロードサイズが大きすぎるために拒否されたときに利用する。
#
if ($ENV{'upload_too_large'}) {
    #
    # 大きすぎる!
    #
    print <<EOHT
Status: 413 Request Entity Too Large
Content-type: text/plain; charset=iso-8859-1
Content-length: 84

Sorry, but your upload file exceeds the limits
set forth in our terms and conditions.
EOHT
}
else {
    #
    # 本来の「拒否」エラー
    #
    my $uri = $ENV{'REDIRECT_REQUEST_URI'};
    my $clength = 165 + length($uri);
    print <<EOHT
Status: 403 Forbidden
Content-type: text/html; charset=iso-8859-1
Content-length: $clength

<html>
 <head>
  <title>Forbidden</title>
 </head>
 <body>
  <h1>Forbidden</h1>
  <p>
  You don't have permission to access $uri
  on this server.
  </p>
 </body>
</html>
EOHT
}
exit(0);
```

解説

このスクリプトは、リクエストが403 Forbiddenエラーになったときに(Denyディレクティブが機能して)呼び出される。このスクリプトは、実際に禁止状態にあるのかアップロードするファイルが大きすぎるのかをチェックし、それに合わせて適切なエラーページを表示する。

どちらの場合も、Status CGI レスポンスヘッダフィールドを返していることに注意しよう。これは正しいステータスをクライアントに伝えるために必要になる。こうしないと、スクリプトの呼び出しに成功することで、ステータスコードが 200 OK になるため、不適切なステータスを返してしまう。200 という間違ったステータスコードを返してしまうと、ブラウザは、ファイルがうまくアップロードできたとユーザに報告してしまう。これは、エラーページのメッセージと矛盾しているので、ユーザを混乱させてしまうことになる。

実際には、「送信データが大きすぎる」というエラーに対応するステータスコード(413)が存在しているので、ここでは、Deny による 403(Forbidden)ステータスを 413 に対応付けている。

Content-lengthフィールドは、フォーム入力のPOSTリクエストに含まれるデータ量を示すのにも使われている。そのため、あまりに小さい値を設定すると、フォーム入力でエラーが発生してしまうことがあるので、注意しなければならない。

参照

- 9章

レシピ 6.5 画像がサイトの外部から使われるのを禁止する

課題

他のサイトが自分のシステム上の画像をリンクし、帯域幅を盗んで、まるでその画像がそのサイトにあるように見えることがある。自分のサーバにあるドキュメントからのみ、画像にアクセスできるようにしたい。

解決

画像が置いてあるディレクトリの.htaccessファイルか、httpd.confファイルの適切な<Directory>コンテナに、次のように追加する。*myserver.com* は自分のドメイン名に置き換えること。

```
<FilesMatch "\.(jpg|jpeg|gif|png)$">
    SetEnvIfNoCase Referer "^http://([^/]*\.)?myserver.com/" local_referrer=1
    Order Allow,Deny
    Allow from env=local_referrer
</FilesMatch>
```

実は、次のレシピを使うと、さらにもう一歩進んで、サイトの外部から画像にアクセスしたユーザに別の画像を返すことができる。

```
SetEnvIfNoCase Referer "^http://([^/]*\.)?myserver.com/" local_referrer=1
RewriteCond "%{ENV:local_referrer}" "!=1"
RewriteRule ".*" "/Stolen-100x100.png" [L]
```

解説

最初の解決策は、リクエストが自分のドキュメント以外からのリンクからであれば、画像ファイルに対するすべてのリクエスト拒否して、403 Forbiddenステータスを返す。別のWebサイトから画像にリンクすると、そのReferer（参照元）が許可されたサーバ名と一致しないので、画像ではなくエラーが返されることになる。

このテクニックは、Refererリクエストヘッダフィールドが付いていないリクエストで問題になるので注意すること。例えば、匿名サービスを通してサイトを訪問した人や、この情報を送らないようにブラウザを設定している人には、エラーが返されてしまう。

2番目の解決策は最初の解決策と似ているが、リクエストを拒否するのではなく、リクエストされた画像を別の画像に置き換えるところが違っている。この解決策を使うと、警告メッセージや訪問者が画像を盗もうとしているのを阻止するのに役立つような絵をStolen-100x100.pngとして用意しておき、送り返すことができる。

このテクニックを使うと、問題が素早く解決するかもしれない。盗みを働いているサイトを訪問すると、「この画像は盗まれたものだ！」ということがすぐわかるためだ。サイトの持ち主は通常、そのままにしておこうとは思わないはずだ。単に403（Forbidden）エラーを返してしまうと、参照しているページには壊れた画像のアイコンが出てくるだけだ。最近は、誰もがそのアイコンを見慣れていて、そこには何もないのだと思ってしまうだけだろう。

参照

- レシピ 6.21

レシピ 6.6　弱い認証と強い認証の両方を要求する

課題

あるリソースに対して、弱い認証と強い認証の両方を要求したい。例えば、特定の場所からのみサイトにアクセスでき、さらにユーザにパスワードの提示を要求したい。

解決

Satisfy ディレクティブを使うと、両方の認証を要求することができる。

```
<Directory /www/htdocs/sensitive>
    # すべてのアクセス制限を適用
    Satisfy All

    # パスワードを要求
    AuthType Basic
```

```
    AuthName Sensitive
    AuthUserFile /www/passwords/users
    AuthGroupFile /www/passwords/groups
    Require group salesmen

    # 特定のネットワークからのアクセスを要求
    Order deny,allow
    Deny from all
    Allow from 192.168.1
</Directory>
```

解説

この例では、ユーザがリソースにアクセスするためには、salesmenグループの一員であることを提示するためのログインと、192.168.1 ネットワークにあるマシンからアクセスしていることの両方が必要になる。

Satisfy Allディレクティブは、指定したスコープに対して、すべてのアクセス制御手段を適用することを要求する。ユーザが、IPアドレスの一致しないマシンからこのリソースにアクセスしようとすると、ブラウザにはすぐにForbiddenエラーメッセージが表示される。ログファイルには、次のようなエラーメッセージが記録されるだろう。

```
[Sun May 25 15:31:53 2003] [error] [client 208.32.53.7] client denied by server
    configuration: /usr/local/apache/htdocs/index.html
```

ユーザが必要とされるIPアドレスを使っていても、パスワード入力のためのダイアログボックスが現れて、有効なユーザ名とパスワードを提示しなければならない。

少し先の話になるが、Apache 2.4リリースでは、このレシピの構文は少し変わる予定だ。Allow、Deny、Satisfyディレクティブは長い間、Apache 管理者を混乱させているため、この構文は修正されることになっている。

新しい構文は、次のようなものになる。これまでの構文をまだ使いたい人は、mod_access_compatモジュールを使えばよい。Satisfy All コマンドは次のようになる。

```
<SatisfyAll>
    Require group salesmen
    Require ip 192.168.1
</SatisfyAll>
```

SatisfyOne ディレクティブも参照すること。これは、Satisfy any 構文を置き換えるものだ。

参照

- レシピ 6.9

弱い認証と強い認証

HTTP に対する基本的な Apache のセキュリティモデルは、弱い認証と強い認証という概念に基づいている。弱い認証のメカニズムは、ユーザが自ら提示した情報に頼っている。一方、強い認証のメカニズムは、ユーザに尋ねることなく入手した証明書を使う。例えば、ユーザ名とパスワードは弱い証明書であり、ユーザのクライアントシステムの IP アドレスは強い証明書と見なされる。

この2種類の違いは、Apacheが認証失敗をどのように扱うかにある。弱い証明書が無効であった場合、サーバは`401 Unauthorized`ステータスで応答し、ユーザが再度アクセスを試みることを許している。これに対して、強い証明書が要求されるときに認証に失敗すると、`403 Forbidden` ステータスを返し、再試行の機会がないことを示す。

さらに、強い証明書と弱い証明書を組み合わせて要求することもできる。これは、Satisfy ディレクティブを使ってコントロールすることができる。要求できるのは次の 5 つである。

- なし、認証は必要ない
- 強い証明書だけが必要
- 弱い証明書だけが必要
- 強い証明書と弱い証明書の両方を受け入れ、いずれかが有効であればアクセスを許可
- 強い証明書と弱い証明書の両方が必要

レシピ 6.7　.htpasswd ファイルを管理する

課題

HTTP の Basic 認証で使うパスワードファイルを作りたい。

解決

`htpasswd` ユーティリティを使うと、パスワードファイルを作ることができる。表 6-1 にその使い方を説明する。

表 6-1　htpasswd を使ったパスワードファイルの管理

コマンド	アクション
`% htpasswd -c user.pass waldo`	user.passという名前のパスワードファイルを作成し、ユーザwaldoのエントリを新しく登録する。パスワードを入力するよう指示される。
`% htpasswd user.pass ralph`	パスワードファイルuser.passにユーザralphのエントリを追加する。パスワードを入力するよう指示される。
`% htpasswd -b user.pass ralph mydogspot`	パスワードファイルuser.passにユーザralphを追加し、そのパスワードを mydogspot に設定する。

あるいは、Perlモジュール Apache::Htpasswd を使って、プログラムでパスワードファイルを管理してもよい。

```
use Apache::Htpasswd;
$pass = new Apache::Htpasswd("/usr/local/apache/passwords/user.pass") or
    die "Couldn't open password file.";

# エントリを追加
$pass->htpasswd("waldo", "emerson");

# エントリを削除
$pass->htDelete("waldo");
```

解説

htpasswd ユーティリティは Apache に付属しているプログラムであり、bin サブディレクトリにインストールされている。

Apacheのサードパーティ製ディストリビューションでは、htpasswdプログラムがパスの通ったディレクトリにコピーされているかもしれない。しかし通常、パスには入っていないので、htpasswd を実行するには、プログラムをパスの通ったディレクトリに置くか、/usr/local/apache/bin/htpasswd のように完全パスで指定する必要がある。

表 6-1 の最初の行では、新しいパスワードファイルを指定した場所に作成する。この例では、user.pass という名前の新しいパスワードファイルを作成し、ユーザwaldoのためのユーザ名とパスワードを追加している。これを実行すると、希望するパスワードを入力するように指示される。入力すると、さらに確認のために、もう一度パスワードを入力するよう指示される。

-cフラグは新しいパスワードファイルを作成することを指示する。同名のファイルがすでに存在していても新しいファイルが作成されるため、このフラグを付けるのは初回だけにするよう注意しよう。以後、このフラグを使うと、既存のパスワードファイルは消されて、新しい(ほとんど空の)ファイルに置き換えられてしまう。

表6-1の2行目では、既存のパスワードファイルに対してパスワードを追加している。すでに説明したように、ユーザは希望するパスワードを入力するよう指示され、入力すると、さらに確認のために、もう一度パスワードを入力するように指示される。ほとんどのプラットフォームでは、パスワードファイルにはデフォルトでcryptアルゴリズムが使われる。Windows、Netware、TPFといったプラットフォームでは、デフォルトで MD5 アルゴリズムが使われる。

crypt を使うプラットフォームでは、パスワードファイルの各行は次のようなものになる。

```
waldo:/z32oW/ruTI8U
```

ユーザ名とコロンの後に続く部分が、暗号化されたパスワードだ。各行にユーザ名とパスワードの組が並

んでいる。

htpasswdユーティリティにはいろいろなオプションがある。例えば、-mフラグを指定すると、パスワードの暗号化にMD5を利用することができる。-bフラグを指定すると、パスワード入力が指示される代わりに、コマンドラインでパスワードを指定することができる。-nフラグを指定すると、パスワードファイルを書き換えずに、結果を標準出力に表示することができる。

htpasswdユーティリティを使ってパスワードを対話形式で入力するのではなく、スクリプト中でパスワードを生成したいときには、-bフラグが特に便利だ。表6-1の3行目で、この構文を説明している。

Apache 2.0.46以降のバージョンでは、-Dフラグを付けると、パスワードファイルから既存のユーザを削除することができる。

```
% htpasswd -D user.pass waldo
```

これ以前のバージョンでは、別の方法でパスワードファイルからユーザを削除しなければならない。例えば、次のように、grepを使って1行削除してもよいし、単にテキストエディタでファイルを編集してもよい。

```
% egrep -v '^waldo:' user.pass >!user.pass
```

Apache::Htpasswdは、Kevin Meltzerによって作成され、CPAN（http://cpan.org）から入手することができる。このモジュールは、Apacheパスワードファイルに対するPerlインターフェイスを提供している。このモジュールを使うと、このレシピで説明したようなPerlコードを数行書くだけで、CGIプログラムなどからパスワードファイルを編集することができる。

このレシピで紹介したメソッド以外にも、パスワードファイル内の特定のパスワードをチェックしたり、パスワードファイルに含まれるユーザのリストを取得したり、特定のユーザの暗号化されたパスワードを読み出したりするメソッドがある。このすばらしいモジュールの機能について詳しく知りたければ、このモジュールのドキュメントを参照してほしい。

最後に、パスワードファイルについて注意しておく。パスワードファイルはWebからアクセスできない場所（つまり、ドキュメントディレクトリの外部）に格納することを強く推奨する。パスワードファイルをドキュメントディレクトリに置いてしまうと、誰かがそのファイルをダウンロードして、それに対してブルートフォース攻撃をかけて、最終的にパスワードを解読してしまう危険があるためだ。

参照

- レシピ6.8
- http://search.cpan.org/author/KMELTZ/Apache-Htpasswd/Htpasswd.pm

レシピ6.8　Digest認証のパスワードファイルを作成する

課題

Digest認証で使うためのパスワードファイルを作りたい。

解決

次の一連のコマンドを使って、Digest認証によって保護したい領域のための証明書ファイルを作成することができる。

```
% htdigest -c "By invitation only" rbowen
% htdigest "By invitation only" krietz
```

解説

Digest認証は、mod_auth_digestモジュールで実装されており、ユーザ名、パスワード、認証領域のMD5ハッシュを使って、クライアントの証明書をチェックする。Apacheにはhtdigestユーティリティが付属しており、これを使ってパスワードファイルを作ることができる。

このコマンドの構文はhtpasswdユーティリティと非常に似ているが、パスワードファイルを使用する認証領域を指定しなければならないところが違っている。作成されるファイルはユーザごとに1行のエントリで構成されており、次のようなものになる。

```
rbowen:By invitation only:23bc21f78273f49650d4b8c2e26141a6
```

htpasswdで作るパスワードファイルのエントリがどこでも使えるのに対して、htdigestで作るパスワードファイルのエントリは、特定の認証領域でしか使えないことに注意しよう。暗号化されたハッシュには、領域情報も含まれているためだ。

htpasswdと同様に、-cフラグを指定すると、新しいファイルを作り、既存のファイルを上書きしてしまうおそれがある。htdigestを実行すると、パスワードを入力するよう指示され、入力すると、さらに確認のために、もう一度パスワードを入力するよう指示される。

htdigestには、htpasswdにあるような追加オプションはない。

参照

- レシピ6.7

レシピ6.9 サブディレクトリのセキュリティを弱める

課題

次のように、サイトの一部の領域に厳しいセキュリティを適用したいときがある。

```
<Directory /usr/local/apache/htdocs/BoD>
    Satisfy All
    AuthUserFile /usr/local/apache/access/bod.htpasswd
    Require valid-user
</Directory>
```

Apacheのスコープ規則により、このセキュリティは、対象のディレクトリと下位のサブディレクトリに

あるすべてのドキュメントに対して適用されてしまう。しかし、BoD/minutes のような特定のサブディレクトリは、制限なしでアクセスできるようにしたい場合がある。

解決

Satisfy ディレクティブを使えばよい。サブディレクトリにある .htaccess ファイルか、適切な <Directory> コンテナに、次の設定を追加する。

```
Satisfy Any
Order Deny,Allow
Allow from all
```

解説

こうすると、Apacheは弱い認証(ユーザの証明書を使う)か強い認証(IPアドレスを使う)のどちらか1つの認証メカニズムの要件を満たせば、アクセスを許可する。この場合、訪問者の発信元に関係なく、強い認証メカニズムが常に満たされることになる。

対象とするディレクトリ以下のすべてのサブディレクトリにも、新しいデフォルトのセキュリティ条件を設定してしまうということに注意しよう。つまり、1つのサブディレクトリの鍵を開けるだけではなく、それ以下のすべてのサブディレクトリの鍵も開けてしまうことになる。

参照

- レシピ 6.6
- レシピ 6.10

レシピ 6.10　選択的に制限を解除する

課題

ドキュメントにアクセスするのに、ユーザ名とパスワードを要求するよう制限をかけたいが、一般に公開したいドキュメントもいくつかある。例えば、index.html は一般に公開したいが、そのディレクトリにある他のファイルにアクセスするには、パスワード認証を要求したい。

解決

Satisfy Any ディレクティブを .htaccess や httpd.conf ファイルの適切な場所に追加すればよい。

```
<Files index.html>
    Order Deny,Allow
    Allow from all
    Satisfy Any
</Files>
```

これは.htaccessファイルに置いてもよいし、影響範囲を制限するために、<Directory>コンテナの中に置い

てもよい。

```
<Directory "/usr/local/apache/htdocs">
    Satisfy All
    Order allow,deny
    Deny from all
    <Files *.html>
        Order deny,allow
        Allow from all
        Satisfy Any
    </Files>
</Directory>
```

解説

他のファイルやディレクトリ全体にどんな制限をかけたとしても、この解決策における<Files>コンテナは、誰もが制限なく index.html ファイルにアクセスできるようにする。Satisfy All を指定すると、Apache はいずれかの制限をかけるのではなく、その場所にかけられたすべての制限を適用する。この場合、Allow from all という制限がかけられており、すべてのクライアントからのアクセスを許可する。

この方法は、シェルのグローバル文字を使って、簡単に任意のファイル名のパターンに適用することができる。ファイル名に正規表現が使えるよう拡張するには、<Files>ディレクティブの代わりに<FilesMatch>ディレクティブを使うとよい。

参照

- レシピ 6.9
- レシピ 6.6
- http://httpd.apache.org/docs/mod/mod_access.html
- http://httpd.apache.org/docs/2.0/mod/mod_access.html

レシピ 6.11 ファイル所有権を使ってアクセスを許可する

課題

システムのファイル所有権に基づいたユーザ認証を行いたい。つまり、ファイルを所有しているユーザ名と認証相手のユーザ名が一致する場合に、アクセスを許可したい。

解決

Require file-owner ディレクティブを使えばよい。

```
<Directory /home/*/public_html/private>
    AuthType Basic
    AuthName "MyOwnFiles"
```

```
    AuthUserFile /some/master/authdb
    Require file-owner
</Directory>
```

解説

このレシピの目的は、ディレクトリ/home/jones/public_html/privateにアクセスするには、ユーザ名jonesとして認証されなければならないということだ。

この例では、ユーザはシステムのパスワードファイルに対して認証されるのではなく、AuthUserFileに指定したファイルに対して認証される。Apacheは単に、認証で使われた名前が対象とするファイルやディレクトリの所有者の名前と一致することを要求する。これはmod_authの機能であり、他の認証モジュールでは利用できないことに注意しよう。

これは、Apache 1.3.22で追加された機能だ。Apache 2.2では、mod_authz_ownerモジュールがこの機能を提供している。

参照

- Require file-owner キーワード―― http://httpd.apache.org/docs/mod/mod_auth.html#require

レシピ6.12　ユーザの証明書をMySQLデータベースに格納する

課題

ユーザ名とパスワード情報を MySQL データベースに格納して、ユーザ認証に使いたい。

解決

Apache 1.3 では、mod_auth_mysql を使えばよい。

```
Auth_MySQL_Info db_host.example.com db_user my_password
Auth_MySQL_General_DB auth_database_name

<Directory /www/htdocs/private>
    AuthName "Protected directory"
    AuthType Basic
    require valid-user
</Directory>
```

Apache 2.2 以降では、mod_authn_dbi を使えばよい。

```
AuthnDbiDriver Config1 mysql
AuthnDbiHost Config1 db.example.com
AuthnDbiUsername Config1 db_username
```

```
AuthnDbiPassword Config1 db_password
AuthnDbiName Config1 auth_database_name
AuthnDbiTable Config1 auth_database_table
AuthnDbiUsernameField Config1 user_field
AuthnDbiPasswordField Config1 password_field
AuthnDbiIsActiveField Config1 is_active_field

AuthnDbiConnMin Config1 3
AuthnDbiConnSoftMax Config1 12
AuthnDbiConnHardMax Config1 20
AuthnDbiConnTTL Config1 600

<Directory  "/www/htdocs/private">
  AuthType Digest
  AuthName  "Protected directory"
  AuthBasicProvider dbi
  AuthnDbiServerConfig Config1
  Require valid-user
</Directory>
```

解説

mod_auth_mysqlという名前のモジュールはたくさん存在する。この例で使ったmod_auth_mysqlモジュールは、http://www.diegonet.com/support/mod_auth_mysql.shtml から入手することができる。作成しなければならないデータベースフィールドや、モジュールが提供している追加オプションに関する詳しい説明は、このモジュールの Web サイトにあるドキュメントを調べてほしい。

Apache 2.2 以降を使っている場合には、新しい認証フレームワークを利用したmod_authn_dbiモジュールを使いたいかもしれない。このモジュールは、http://mod-auth.sourceforge.net/から入手することができる。Apache 2.2 の新しい認証 API を使うと、以前のバージョンではできなかったことができるようになる。例えば、AuthTypeディレクティブのBasicをDigestに変更するだけで、1つのモジュール（例えば、mod_authn_dbi）を、Basic認証にもDigest認証にも使うことができる。（先の例では、AuthBasicProviderはAuthDigestProviderになる）。

mod_authn_dbiは、libdbiを利用している。libdbiは、汎用のデータベースアクセスライブラリであり、お気に入りのデータベースサーバを使って認証サービスを行うことができる。libdbiドライバは、普及しているほとんどすべてのデータベースサーバを利用することができる。mod_authn_dbi についてもっと詳しく知るには、このモジュールの Web サイトにあるドキュメントを調べてほしい。

参照

- http://www.diegonet.com/support/mod_auth_mysql.shtml
- http://mod-auth.sourceforge.net/

レシピ 6.13　認証されたユーザ名にアクセスする

課題

認証されたユーザの名前が知りたい。

解決

mod_phpのようなスクリプティングモジュールには、サーバが設定した値にアクセスする標準インターフェイスを提供しているものがある。例えば、PHPスクリプトの中から、認証に使われたユーザ名を取得するには、$_SERVER スーパーグローバル配列のフィールドにアクセスすればよい。

```
$auth_user = $_SERVER['REMOTE_USER'];
```

Perl あるいは mod_perl スクリプトでは、次のようにすればよい。

```
my $username = $ENV{REMOTE_USER};
```

SSI（Server-Side Include）ディレクティブでは、次のようにすればよい。

```
こんにちは、<!--#echo var="REMOTE_USER" --> さん。来てくれてありがとう。
```

スクリプティングモジュールによっては、特別な手段でこのような情報にアクセスできるかもしれない。こうした手段がなかったり、非スクリプトのアプリケーションであっても、環境変数REMOTE_USERを調べるという手段が使える。

解説

ユーザが認証されると、認証に使われた名前が環境変数 REMOTE_USER に設定される。CGI プログラムや SSI ディレクティブ、PHPファイル、その他いろいろな方法でこの変数にアクセスすることができる。この値は、access_log ファイルにも記録される。

認証モジュールがこの変数に値を設定するのは慣習であり、聞くところによると値を設定しないサードパーティ製の認証モジュールもあるようなので注意しよう。もしこの値が設定されなくても、別の手段でこの情報にアクセスできるはずだ。

参照

- レシピ 6.14

レシピ 6.14　認証に使われたパスワードを取得する

課題

ユーザが認証に使ったパスワードを取得したい。

解決

標準のApacheモジュールでは、この値を取得することはできない。しかし、独自の認証メソッドを書けば、Apache の API を使って値を取得することができる。Apache 1.3 の API では、ap_get_basic_auth_pw 関数を調べればよい。Apache 2.0 の API では、get_basic_auth 関数を調べればよい。

mod_perl を使って認証ハンドラを書く場合には、get_username 関数を使ってユーザ名とパスワードを取り出すことができる。

```
my ($username, $password) = get_username($r);
```

適切なフラグを付けてパッケージをリビルドすれば、サーバで実行されるCGIスクリプトからもこの情報を入手できるようになるだろう。Apache 1.3 と 2.0 では、次のように設定すればよい。

```
% CFLAGS="$CFLAGS -DSECURITY_HOLE_PASS_AHTORIZATION"
```

解説

セキュリティ上の理由から、ユーザ名は環境変数から入手できても、認証に使われたパスワードは容易に入手できないようになっている。この背景にあるのは、悪意のある人がパスワードを盗むのが簡単になると、悪用されてしまうためだ。したがって、パスワードを入手するのはほとんど不可能になるように設計されている。

これを変更する唯一の方法は、特別なコンパイルフラグを付けて(通常行うべきではない)ソースコードからサーバをビルドし直すことだ。別の方法としては、独自の認証モジュールを書けば、パスワードにアクセスすることができる。自分のコードではパスワードを検証しなければならないので、これは当然のことだろう。

参照

- レシピ 6.13

レシピ 6.15 パスワードのブルートフォース攻撃を防ぐ

課題

パスワードクラッカーから攻撃された場合など、認証に繰り返し失敗したときには、そのユーザ名を無効にしたい。

解決

Apacheにある標準の認証モジュールでは、これを実現することはできない。よくある方法は、ログファイルを注意深く監視することだ。あるいは、Apache::BruteWatchのようなものを使って、ユーザが攻撃されると通知されるようにしてもよい。

```
PerlLogHandler Apache::BruteWatch
PerlSetVar BruteDatabase     DBI:mysql:brutelog
PerlSetVar BruteDataUser     username
PerlSetVar BruteDataPassword password

PerlSetVar BruteMaxTries     5
PerlSetVar BruteMaxTime      120
PerlSetVar BruteNotify       rbowen@example.com
```

解説

HTTPがステートレスであるという性質と、技術的には、ユーザは「ログイン」しているわけではないという事実（コラム「HTTP とブラウザと証明書」を参照）のため、認証の試行の間には何の関連もない。mod_security のような Apache 監査ツールの多くは、1つのリクエストを基本として動いており、1つ1つのリクエストを比較して、複数のリクエストから分析結果を構築することはできない。このため、特定のユーザ名で繰り返しログインを試みることができてしまい、それをツールで自動的に簡単に検出することはできない。

最善の策は、注意深くログファイルを監視するか、あるいは、ログファイルを監視する何らかのプロセスを走らせることだ。

その 1 つの方法が Apache::BruteWatch であり、ログファイルを監視して、特定のアカウントがブルートフォース攻撃の標的になると通知してくれる。この例のように設定すると、2分間に5回、そのアカウントに対する認証が失敗すると、その状況がサーバ管理者に通知される。これにより、サーバ管理者は、攻撃者のアドレスからサイトへのアクセスを遮断するなどの適切な対策をとることができる。

Apache::BruteWatchは、CPAN（CPAN.org）から入手することができる。これを動かすには、mod_perlも必要だ。執筆時点で、リアルタイムでこのような処理をするモジュールは、これしか見つからなかった。

参照

- コラム「HTTP とブラウザと証明書」

レシピ 6.16　Digest 認証か Basic 認証を使う

課題

Basic 認証か Digest 認証を使いたいが、その違いを理解したい。

解決

Basic 認証を使ったアクセス制御をするには、AuthType Basic と htpasswd ツールを使う。

Digest 認証を使ったアクセス制御をするには、AuthType Digest と htdigest ツールを使う。

解説

Basic 認証は、その名前の通り原始的であり、安全とは言えない。Basic 認証では、ユーザの証明書を可逆

アルゴリズム(基本的にBase64エンコーディング)で符号化し、リクエストヘッダの一部として平文のまま送信する。その送信内容を傍受すれば、誰でも(何でも)簡単に証明書の符号化を解読して、悪用することができてしまう。このため、Basic認証の使用は、それほど機密性が高くないドキュメントを保護したい場合や、他に代替手段がない場合に限るべきである。

これに対して、Digest認証はもっと安全な方法であり、証明書の傍受、なりすまし、リプレイアタックに対して、より安全に防御できる。重要なのは、ユーザ名とパスワードを平文でネットワーク上に流さないということだ。

Basic認証を使うための領域を用意するには、単にユーザ名とパスワードの組を格納して、その場所をサーバに教えればよい。パスワードは、暗号化されることもあれば、暗号化されないこともある。同じ証明書をサーバ上のすべての領域で使うこともできるし、全く別のサーバにコピーして、別のサーバで使うことさえできる。証明書はいろいろなデータベースに格納することもできる。Basic認証の証明書をテキストファイルや、GDBMファイル、MySQLデータベース、LDAPディレクトリなどに格納するいろいろなモジュールがある。

Digest認証のセットアップは少々ややこしい。その1つは、証明書を他の領域に持っていっても使うことができないという点だ。証明書を作るときには、その証明書を適用する領域を指定する必要がある。もう1つは、Apacheパッケージが現在対応している証明書の格納手段は、テキストファイルだけだという点だ。Digest認証の証明書をLDAPディレクトリやOracleデータベースに格納したければ、それができるサードパーティ製モジュールを探すか、自分でモジュールを書くしかない。

この複雑なセットアップ手順に加えて、Digest認証は現在のところ、まだ市場に浸透していないという悩みがある。つまり、たとえApacheがDigest認証をサポートしていても、すべてのブラウザやその他のWebクライアントがサポートしているわけではない。したがって、ユーザが他に利用できるものがないため、結局、Basic認証を使うしかないかもしれない。しかし、時間が経つにつれ、このようなケースも少なくなってきており、Digest認証をサポートしていないWebクライアントはほんのわずかになっている。

参照

- レシピ 6.18

レシピ 6.17　URL に埋め込まれた証明書にアクセスする

課題

ユーザが証明書を埋め込んだURL(例えば、http://user:password@host/)でサイトにアクセスするとき、URLから証明書を取り出して、検証などに使いたい。

解決

解決策はない。よく誤解されるが、これは問題ではない。

解説

URLにユーザ名とパスワードを埋め込むと、パスワード入力を指示されることなく、パスワードで保護さ

れたサイトに直接アクセスすることができるリンクを作ることができる。しかし、よく誤解されているのだが、このときもユーザ名とパスワードは実際には通常と同じ方法で(つまり、WWW-Authenticate ヘッダを使って)サーバに送られており、URLの一部として送られているわけではない。ブラウザはURLを分解し、適切なリクエストヘッダフィールドに変換して、サーバに送っている。

参照

- コラム「HTTP とブラウザと証明書」

レシピ 6.18　WebDAV を保護する

課題

WebDAVを使って、ユーザがWebドキュメントをアップロードしたり、その他の管理ができるようにしたいが、サーバのセキュリティ上の危険を増やしたくはない。

解決

WebDAV を利用するのに、認証が必要になるよう設定すればよい。

```
<Directory "/www/htdocs/dav-test">
   Order Allow,Deny
   Deny from all
   AuthDigestFile "/www/acl/.htpasswd-dav-test"
   # Apache2.2 では AuthDigestFile "/www/acl/.htpasswd-dav- test" の代わりに
   #  AuthDigestProvider file
   #  AuthUserFile "/www/acl/.htpasswd-dav-test"
   # とする
   AuthDigestDomain "/dav-test/"
   AuthName "DAV access"
   Require valid-user
   Satisfy Any
</Directory>
```

解説

WebDAV 操作は、サーバのリソースを変更することができる。mod_dav はサーバの一部として動くため、WebDAVが有効になっている場所は、サーバのUserディレクティブに指定したユーザによって書き込み可能な状態にしておく必要がある。これは、Apache Web サーバの一部として動いている CGI スクリプトや、その他のモジュールからも、書き込み可能になっていることを意味している。リモートからの変更操作へのアクセス制御を行うには、WebDAVを有効にする場所へのアクセス制御を有効にしなければならない。ユーザレベルの認証のような、弱い認証制御手段を使うのであれば、この解決策で示したように、Basic認証ではなく Digest 認証を使うべきである。

<Directory> コンテナにある内容を、dav-test/.htaccess ファイルに書いても構わない。認証データベース

(AuthDigestFile ディレクティブで指定する)は、サーバの URI スペースにあるわけではなく、ブラウザや WebDAV ツールを使っても取得することはできないことに注意しよう。

認証データベースと.htaccessファイルは、サーバユーザによって変更可能な状態にしてはいけない。また、WebDAV ユーザが変更できるようにしてもいけない。

参照

- レシピ 6.16

レシピ6.19 Webサーバユーザがファイルを書き込めないようにして WebDAV を有効にする

課題

WebDAV を動かしたいが、Apache サーバユーザがドキュメントファイルを書き込めるようにはしたくない。

解決

2 つの Web サービスを別々のユーザで実行する。例えば、ユーザ dav、グループ dav で、DAV を有効にしたサーバを動かす。これに対して、もう1つのサーバは、コンテンツを提供する役割を持ち、ユーザnobody、グループ nobody で動かす。ユーザ dav、あるいは、グループ dav は、Web コンテンツに書き込みできるように設定する。

1つのWebサーバだけが、特定のポート／IPアドレスの組み合わせを使うことができるのを忘れないようにしよう。このため、WebDAVを有効にしたサーバは、WebDAVを無効にしてあるサーバとは別のアドレスやポート番号(あるいは両方とも別のもの)を使う必要がある。

解説

DAVに関する大きなセキュリティ上の関心は、DAVでコンテンツを更新するためには、Webサーバユーザがコンテンツを変更できるようにしなければならないということだ。これは、CGI プログラムや SSI ディレクティブ、あるいは、Webサーバで動作している他のプログラムからも、コンテンツが編集できてしまうことを意味している。Apache のセキュリティガイドラインでは Web サーバユーザにとってファイルが書き込み可能な状態にしないよう警告しているが、DAV を使うにはそうせざるを得ない。

2つのApacheサーバを動かすことによって、この制限をうまく回避することができる。DAVを有効にした Web サーバは別のポートで動いており、User と Group ディレクティブを使って、次のように別のユーザとグループを設定している。

```
User dav
Group dav
```

davは、対象となるWebコンテンツの所有者だ。もう1つのWebサーバは、ユーザにコンテンツを提供する役割を持ち、どのドキュメントにも書き込み権限がないユーザで動かす。

DAVを有効にしたWebサーバは、しっかり認証を実施すべきであり、そのサイトの編集権限のあるユーザだけがその領域にアクセスできるようにする必要がある。このサーバは、インストールするモジュールや実行する子プロセス(あるいはスレッド)の数を制限して、非常に軽くなるように設定するのがよいだろう。

最後に、Apache 2.0で導入されたperchild MPMは、異なるユーザIDで異なるバーチャルホストを実行するという考えをサポートしているということを述べておく。これを使うと、1つのバーチャルホストだけでDAVを有効にすれば、このレシピを実現できてしまうだろう。しかし、本書執筆時点では、perchild MPMはまだうまく機能していない。

参照

- http://httpd.apache.org/docs-2.0/mod/mod_dav.html
- http://httpd.apache.org/docs-2.0/mod/perchild.html

レシピ6.20　特定のURLに対するプロキシ経由のアクセスを制限する

課題

プロキシサーバを使っているユーザに、特定のURLや特定のパターンのURL(MP3やストリーミングビデオファイルなど)をアクセスさせたくない。

解決

次のキーワードを指定して、アクセスを遮断することができる。

```
ProxyBlock .rm .ra .mp3
```

特定のバックエンドURLを指定して、アクセスを遮断することもできる。

```
<Directory proxy:http://other-host.org/path>
    Order Allow,Deny
    Deny from all
    Satisfy All
</Directory>
```

あるいは、正規表現によるパターンマッチングを使って、アクセスを遮断することもできる。

```
<Directory proxy:*>
    RewriteEngine On
    #
    # Real形式の動画と音楽ファイルへのプロキシ経由のアクセスを禁止
    #
```

```
    RewriteRule "\.(rm|ra)$" "-" [F,NC]
    #
    # このサイトを経由した .mil サイトへのアクセスを許可しない
    #
    RewriteRule "^[a-z]+://[-.a-z0-9]*\.mil($|/)" "-" [F,NC]
</Directory>
```

解説

いずれの解決策でも、クライアントがブロックされたURLにアクセスしようとすると、サーバから403 Forbiddenステータスを受け取ることになる。

最初の解決策では、proxyモジュールに組み込まれているProxyBlockディレクティブの機能を使っている。これは簡単で、効果的な方法だ。アクセスを捕捉すると、今後、同じURLに対するアクセスをより少ない労力でブロックすることができる。しかし、利用可能なパターンマッチングはごく限定されており、混乱を招くおそれがある。例えば、次のように指定したとする。

```
ProxyBlock .mil
```

こうすると、サーバは、http://www.navy.mil/ だけでなくhttp://example.com/spec.mil/list.html のアクセスも拒否してしまう。これはおそらく意図したものと違うだろう。

2番目の解決策では、取得しようとしているURL（あるいは、ProxyPassディレクティブの場合にはゲートウェイ）に基づいて制限をかけることができる。

3番目の解決策では、ブロックしたいものを複雑なパターンを使って組み立てることができる。自由度が高く強力だが、効率はやや悪くなる。他の解決策では不十分だとわかったときにだけ、この方法を使うのがよいだろう。

<DirectoryMatch>コンテナも同様の働きをし、複雑なパターンを使うことができる。

RewriteRuleディレクティブには2つのフラグを指定している。最初のフラグF（Forbidden）は、URLがパターンにマッチしたときに、403 Forbiddenエラーを返すようサーバに指示している。2番目のフラグNC（No Case）は、パターンマッチで大文字と小文字を区別しないように指示している。

mod_rewriteを使った解決策の欠点は、使えるパターンがあまりに限定されていることだ。最初のRewriteRuleパターンは、クライアントがパス情報やクエリ文字列を指定したり、コンテンツ配信元のサーバがこの種のファイルに対して異なるサッフィクス名を使ったりすると、無効になってしまうおそれがある。少し頭を使えばこのような状況も回避できるだろうが、あまりに多くの可能性を1つの正規表現パターンに押し込めようとしないよう注意してほしい。一般的には、1つのRewriteRuleディレクティブよりも、複数のRewriteRuleディレクティブにした方がよい。何でもできる1つの複雑なRewriteRuleにしてしまうと、誰も内容を理解できないし、問題を引き起こしやすい。

参照

- mod_proxy のドキュメント
  http://httpd.apache.org/docs/mod/mod_proxy.html
  http://httpd.apache.org/docs/2.2/mod/mod_proxy.htm

- mod_rewrite のドキュメント
  http://httpd.apache.org/docs/mod/mod_rewrite.html
  http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

レシピ 6.21　ラッパーを使ってファイルを保護する

課題

標準の Web 認証（メンバ限定の領域など）ではない別の方法で、ファイルへのアクセスを制限したい。

解決

httpd.conf において、スクリプトだけからアクセスできるようにしたいコンテンツを含む <Directory> コンテナに、以下を追加する。

```
RewriteEngine On
RewriteRule "\.(dll|zip|exe)$" protect.php [NC]
RewriteCond %{REMOTE_ADDR} "!^my.servers.ip"
RewriteRule "\.cgi$" protect.php [NC]
```

以下は protect.php の例であり、リクエストされたドキュメントのローカル URI を表示する。

```
<?php
/*
 * 実際にリクエストされたドキュメントのURLは、$_SERVER['REQUEST_URI']に入る。
 * このURLを元に、すべきことを適切に決定すればよい。
 */
Header('Content-type: text/plain');
$body = sprintf("Document requested was: %s\n", $_SERVER['REQUEST_URI']);
Header('Content-length: ' . strlen($body));
print $body;
?>
```

解説

このレシピが必要になる状況では、Webプロトコルに組み込まれた標準のメカニズムではなく、Cookieを使った認証と許可が使われてきた。サイトのドキュメントに対するリクエストは、Cookie使ってチェックする。もしCookieがなかったり、期限切れであったり、その有効性に疑念を持つような問題があれば、ログインページにリダイレクトする。

この方法はかなり普及していて、わかりやすい方法だ。さらに必要なことは、Cookieに基づいてファイルへのアクセスを制限し、URLのみのリクエストをブロックすることだ。

この目的を達成するためには、ラッパー(この解決策ではprotect.phpに相当する)を作ればよい。保護しているドキュメントがリクエストされたときに、このラッパーが呼び出される。protect.phpスクリプトでは、Cookieを検証した後、環境変数からファイル名を取り出して、拡張子からコンテンツの種類を判断し、ファイルをオープンしてコンテンツを送信するようにすればよい。

解決策の例を見てみよう。これらmod_rewriteディレクティブが適用されるドキュメントのうち、拡張子が.dll、.zip、.exe、.cgiのいずれかで終わるドキュメントがリクエストされて、かつ、そのリクエストがWebサーバ以外のシステムからやってきた場合には、代わりにprotect.phpスクリプトを呼び出している。この解決策のprotect.phpスクリプトは、単にリクエストされたドキュメントのローカルURIを表示するだけだが、アクセス制御などの機能を追加するのは簡単だ。

アクセス制御がこのラッパーの主な目的であり、アクセスが許可されると、ラッパーはリクエストされたドキュメントをクライアントに送信する。この場合、ラッパーはリクエストされたドキュメントのファイルシステム上のパスを取り出し、PHPのfpassthru()ルーチンを使ってそのドキュメントをオープンして、クライアントに送信すればよい。あるいは、URLをファイルのようにオープンできるPHPの機能、fopen("http://*ドキュメントのURL*")関数を使って、ドキュメントにアクセスしてもよい(ドキュメントがスクリプトであるときなど、サーバ処理を必要とする場合には、後者の方法を使う必要がある。)

動的なドキュメントの場合、ラッパーを再び起動してしまい、ループに陥ってしまう。これを回避するには、リクエストしたホストがサーバ以外の場合にのみ、動的ドキュメントにラッパーを適用するようにすればよい。リクエストしたのがWebサーバ自身の場合には、ラッパーをすでに実行しているので、再実行する必要はない。サーバは通常通りドキュメントを処理して、コンテンツをラッパーに送る。ラッパーはまだ元のリクエストを処理しており、きちんとコンテンツをクライアントに送る。この処理はRewriteCondディレクティブで行われており、リクエストがサーバ自身から送られたものでなければ、スクリプトへのリクエストをラッパーに送る。

この方法は、CGIリクエストごとに少なくとも2つのリクエストが同時に呼び出されてしまい、それほど洗練されていないし、パフォーマンスもあまりよくない。しかし、とにかく問題を解決することはできる。

参照

- 5章

レシピ6.22 悪意のあるスクリプトからサーバのファイルを保護する

課題

Webサーバにあるファイルをしっかり保護しておかないと、Webサーバで実行しているスクリプトがファイルにアクセスして、編集したり、破壊したりするおそれがある。このような事態が起こらないようにしたい。

解決

すべてのファイルについて、nobody ユーザや nobody グループから書き込み可能でないことを確認する。そして、機密性の高いファイルについては、nobody ユーザや nobody グループから読み出し可能でないことを確認する。

```
# find / -user nobody
# find / -group nobody
```

解説

User と Group ディレクティブを使うと、どのユーザおよびグループの権限で Web サーバを動作させるかを指定することができる。これらの値は、どちらも nobody という値が設定されていることが多いが、設定によって違っていることもある。たいていの場合、全く新しいユーザとグループを作るのが望ましい。こうしておくと、気付かないうちに余計な権限をユーザに与えてしまうことがなくなる。

すべてがここで指定したユーザやグループの権限で動くため、サーバがアクセス可能なファイルやディレクトリは、サーバで実行するスクリプトからもアクセス可能になる。ファイルのパーミッションの設定によっては、あるバーチャルホストで実行しているスクリプトが、意図的または偶然に、別のバーチャルホストに含まれるファイルを変更したり、削除してしまう可能性もある。

理想的には、スクリプトのデータファイルとして利用するといった明確な目的がない限り、サーバユーザはサーバにあるファイルを所有すべきでないし、書き込み可能であってもいけない。また明確な目的がある場合でも、本物のデータベースを使うことを推奨する。こうすると、ファイル自体をサーバユーザが変更することはできなくなる。また、単にファイルをサーバから書き込み可能にしたいだけなら、そのファイルを/cgi-bin/ のような、Web アクセスが可能な場所に置いてはいけない。

参照

- レシピ 8.13
- レシピ 6.23

レシピ 6.23　ファイルに適切なパーミッションを設定する

課題

最高レベルのセキュリティが実現できるように、ファイルのパーミッションを設定したい。

解決

ServerRoot にある bin ディレクトリは、所有者をユーザ root、グループ root にして、パーミッションを 755（rwxr-xr-x）にすべきである。このディレクトリに含まれるファイルも同様に、所有者を root.root（ユーザ root、グループ root）にして、パーミッションを 755 にすべきだ。

htdocsやcgi-bin、iconsといったドキュメントディレクトリは、Webサイトの開発モデルに最も適したパーミッションを設定しなければならない。しかし、どんな事情があっても、これらのディレクトリやそこに含

まれるファイルは、Web サーバユーザから書き込み可能になっていてはいけない。

ここで示した解決策は、Unix系システム特有のものだ。しかし、他のOSを使っている場合にも、実装方法は異なるが、ここに述べた原則に従うべきである。

conf ディレクトリとその中に含まれるすべてのファイルは、root にだけ読み書き可能にすべきだ。

include と libexec ディレクトリは誰でも読み出し可能だが、誰からも書き込みできないようにしなければならない。

logs ディレクトリは、所有者を root として、root が書き込み可能にすべきである。望むなら、他のユーザもこのディレクトリにあるファイルを読むことができるようにしてもよい。ユーザがログファイルにアクセスできるのは、特にトラブルシューティング時に役に立つ。

man ディレクトリは、すべてのユーザが読み出し可能でなければならない。

最後に、proxyディレクトリは、所有者をサーバユーザとして、サーバユーザに書き込み可能にしておかなければならない。

たいていのUnix系ファイルシステムでは、ディレクトリの中にあるファイルを見えるようにするために、ディレクトリにx(実行可能)ビットを設定しなければならない。

解説

知っておいてほしいのは、Apacheサーバにおいて、ファイルのパーミッションを正しく設定する方法を10人に尋ねたら、10通りの答えが返ってくるということだ。ここで説明した推奨設定は、できるだけ疑り深く考えたものである。世界観やユーザをどれくらい信頼しているかに基づいて、これらの推奨を自由に緩和してもらっても構わない。しかし、ファイルのパーミッションをこれ以上厳しく制限してしまうと、Apacheサーバは動かなくなるだろう。もちろん例外はあり、もっと疑り深く設定したい場合については、後で取り上げることにする。

パーミッションを設定するときに最も重要なのは、Apacheサーバユーザ(Apacheをどのユーザ権限で実行するか)だ。Apacheプロセスをどのユーザおよびグループの権限で実行するかは、httpd.confファイルのUserディレクティブとGroupディレクティブで設定することができる。このユーザは、ほとんどすべてのファイルに読み出しアクセスできる必要があるが、どこにも書き込みアクセスできるようにしてはならない。

binディレクトリの推奨パーミッションは、誰でもその中にあるプログラムを実行できるよう設定することだ。ユーザが htpasswd や htdigest ユーティリティを使ってパスワードファイルを作成したり、suexec ユーティリティを使って CGI プログラムを走らせたり、httpd -v で Apache のバージョンをチェックしたりするなど、このディレクトリにあるプログラムを実行するためにこの設定が必要になる。これらのアクセスを許可しても、既知のセキュリティ上の危険はないはずだ。通常の状態では、非特権ユーザがWebサーバ自体を停止したり、起動したりすることはできない。これらのファイルやディレクトリは、決してroot以外のユーザに書き込み可能にしてはいけない。さもないと、信用できないファイルが置かれて、それをroot権限で実行してしまうおそれがある。

心配性のサーバ管理者は、binディレクトリとそこにあるコンテンツをrootにだけ、読み出し、実行可能にしたいと思うかもしれない。しかし、そうしたときの実際の利点は、他のユーザがユーティリティやhttpdサーバを実行することができなくなるだけだ。ユーティリティには、htpasswdやhtdigestのように、Webマスターだけでなく、コンテンツ提供者(つまりユーザ)が実行することを意図しているものもある。

confディレクトリは、サーバ設定ファイルを含んでおり、好きなだけ厳しくパーミッションを設定すればよい。サーバ設定ファイルを読んでも、サーバ上に余計な権限を獲得できるわけではないが、サーバに危害を加えようとしている人にとっては情報が多いと役に立つ。したがって、このディレクトリはrootにだけ読み出し可能にしたくなるだろう。しかし、そこまで心配することはないと考えている人が多い。

ドキュメントディレクトリは、サーバごとに推奨設定が変わってくるので、パーミッションを推奨するのが特に難しい。コンテンツ提供者が1人だけのサーバでは、ドキュメントディレクトリは所有者をそのユーザにして、Apacheユーザが読み出し可能にしておけばよい。コンテンツ提供者が複数いるサーバでは、ファイルを変更できるユーザのグループを作って、ファイルの所有者をこのグループにして、Apacheユーザが読み出し可能にしておけばよい。iconsディレクトリには、このルールを適用する必要はない。iconsディレクトリにあるコンテンツはめったに変更されないので、どんなユーザも書き込み可能である必要はない。

includeとlibexecディレクトリは、Apache実行ランタイムに必要なファイルを含んでおり、rootだけが起動するので、rootにだけ読み出し可能であればよい。しかし、includeディレクトリはCのヘッダファイルを含んでおり、ユーザがこれらのヘッダファイルを使うアプリケーションをビルドするのに必要なので、アクセスできるようにしておくと便利だ。

logsディレクトリは、どんな事情があろうと、root以外のユーザに書き込み可能であってはいけない。logsディレクトリが他のユーザに書き込み可能になっていると、起動時にApacheプロセスを支配し、サーバのroot権限を獲得できてしまうおそれがある。他のユーザがこのlogsディレクトリのファイルを読み出せるようにするかどうかは読者次第だが、必ずしも必要なわけではない。しかし、たいていのサーバでは、ユーザがログファイルにアクセスできると、とても便利だ。サーバ管理者に問い合わせることなく、自分で問題解決することができる。

manディレクトリには、Apacheに付属するさまざまなユーティリティのmanpageが入っている。これは、すべてのユーザに読み出し可能にすべきである。しかし、manpageをシステムのmanパスに移動しておくのがお勧めだ。あるいは、Apacheをインストールするときに、mandir引数にシステムのmanディレクトリの場所を指定して、そこにmanpageをインストールするようにしてもよい。

最後に、proxyディレクトリは、所有者をサーバユーザにして、サーバユーザが書き込み可能にしておかなければならない。これは、サーバユーザには何も書き込み可能にしてはいけない、という鉄則に対する唯一の例外だ。proxyディレクトリには、mod_proxyが生成、管理しているファイルが入っており、非特権ユーザで動いているApacheプロセスが書き込み可能になっている必要がある。mod_proxyを使ってproxyサーバを動かしていないのであれば、このディレクトリを完全に削除しても構わない。

参照

- 『入門 Unix オペレーティングシステム 第5版』(オライリー・ジャパン発行、原書『Learning the Unix Operating System, Fifth Edition』、Jerry Peek、Grace Todino、John Strang 著、O'Reilly Media 発行)

- http://www.onlamp.com/pub/a/bsd/2000/09/06/FreeBSD_Basics.html

レシピ 6.24 最小限のモジュールを動かす

課題

不要なモジュールをすべて削除して、潜在的なセキュリティホールが発覚するリスクを軽減したい。本当に必要なのはどのモジュールだろうか？

解決

Apache 1.3 では、3 つのモジュールだけで必要最小限のサーバを動かすことができる（実際には、全くモジュールがなくても何とか動かすことはできるが、推奨できない）。

```
% ./configure --disable-module=all --enable-module=dir \
> --enable-module=mime --enable-module=log_config \
```

Apache 2.x では、少し面倒で、不要なモジュールを 1 つ 1 つ無効にしなければならない。

```
% ./configure --disable-access \
> --disable-auth --disable-charset-lite \
> --disable-include --disable-log-config --disable-env --disable-setenvif \
> --disable-mime --disable-status --disable-autoindex --disable-asis \
> --disable-cgid --disable-cgi --disable-negotiation --disable-dir \
> --disable-imap --disable-actions --disable-alias --disable-userdir
```

Apache 2.x も 1.3 と同様に、mod_dir、mod_mime、mod_log_config を有効にしておくとよいだろう。それには、これらのモジュールをこのパラメータのリストに入れないようにすればよい。

解説

セキュリティに関してよく推奨されている考えは、不要なものはすべて排除しなさいというものだ。不要なもの、使わないものは、セキュリティに関する告知を見落としたり、安全なように設定するのを忘れてしまいがちだ。何が必要で何が不要なのかという問題は、なかなか答えが見つからないものだ。

Apacheのパッケージディストリビューションでは、すべてを有効にしていることが多い。そのため、実際には不要なモジュールも動かしてしまっており、ユーザはそれに気付いていないことさえある。

このレシピでは、Apache で実行するモジュールを最小限に減らし、できるだけ最小の Apache サーバを作ろうとしている。つまり、このうちどれかのモジュールを外すと、Apacheは起動さえしなくなり、もちろん Web サイトとして機能しなくなる。

Apache 1.3

Apache 1.3では、この質問の答えはとても簡単だ。3つのモジュールにまで減らすことができる。どうして

もやりたくて、どうなるかよくわかっているのなら、実際にはモジュールをすべて削除してしまっても構わない。

mod_dir モジュールは、/ 宛てのリクエストを、/index.html または DirectoryIndex で指定したディレクトリのデフォルトドキュメント宛てのリクエストに変換する。このモジュールがないと、ユーザがブラウザにホスト名だけを入力しても、すぐに404エラーが返ってきて、デフォルトドキュメントが取得できなくなる。URLには必ずホスト名とファイル名を指定するよう、ユーザに要求しても構わないのであれば、このモジュールを削除することもできる。しかし、Web サイトは非常に使いにくくなってしまうだろう。

mod_mime は、特定のファイルの MIME タイプが何であるかを判断し、Apache が適切な MIME ヘッダを送れるようにする。これにより、ブラウザはそのファイルをどのように表現したらよいかを知ることができる。mod_mime がないと、Web サーバはすべてのファイルを DefaultType ディレクティブで設定した MIME タイプだとして処理してしまう。これが実際のファイルタイプと一致していればよいが、そうでなければ、ブラウザはそのドキュメントを正しく表現できなくなってしまう。Web サイトが1種類のファイルだけで構成されているのであれば、このモジュールを削除してしまっても構わない。

最後に、mod_log_configは、技術的には全く必要がないものだが、使うことを強く推奨する。動作ログを記録せずにWebサーバを動かすと、サイトがどのように利用されているか知ることができず、サーバの健全性に害を及ぼすおそれがある。しかし、ApacheのErrorLog機能を無効にすることはできないということに注意しよう。したがって、本当にWebサイトのアクセス情報を気にしないのであれば、mod_log_configを削除してしまっても構わない。それでもまだ、エラーログ情報は手に入れることができる。

Apacheをこのような最小限の状態で動かすには、デフォルトで配布されている設定ファイルを変更する必要がある。特に、Order、Allow、Denyディレクティブは、mod_accessが提供しているものなので削除する必要がある。また、mod_log_configを削除したときには、LogFormatとCustomLog ディレクティブも削除する必要がある。設定ファイルにあるセクションの多くは、<IfModule> セクションで保護されており、必要なモジュールがなくても問題ないだろう。

Apache 2.x

Apache 2.x では、新しい設定ユーティリティが使われており、そのコマンドラインの構文は複雑になっている。特に、すべてのモジュールを削除してくれるような1つのコマンドラインオプションというものはなく、--disable オプションで各モジュールを指定しなければならない。

Apache 2.xで必要となる最小限のモジュールは、Apache 1.3の場合と同じだ。先に説明したように、mod_dir と mod_mime、mod_log_config を使うことを推奨するが、必須というわけではない。

参照

- http://httpd.apache.org/docs/mod/mod_dir.html
- http://httpd.apache.org/docs/2.2/mod/mod_dir.html
- http://httpd.apache.org/docs/mod/mod_mime.html
- http://httpd.apache.org/docs/2.2/mod/mod_mime.html
- http://httpd.apache.org/docs/mod/mod_log_config.html

- http://httpd.apache.org/docs/2.2/mod/mod_log_config.html

レシピ 6.25 Web ディレクトリの外部にあるファイルへのアクセスを制限する

課題

Web ディレクトリの外部にあるファイルにアクセスできないようにしたい。

解決

Unix 系システムの場合は、次のように設定すればよい。

```
<Directory />
    Order deny,allow
    Deny from all
    AllowOverride None
    Options None
</Directory>
```

Windows システムの場合は、次のように設定すればよい。

```
<Directory C:/>
    Order deny,allow
    Deny from all
    AllowOverride None
    Options None
</Directory>
```

これをシステムにあるドライブ文字ごとに繰り返す。

解説

すべてのアクセスを禁止した上で、必要なところを選んでアクセス許可を与えるというのは、セキュリティ上、よい方法だ。ファイルシステム全体に対して、Deny from allディレクティブを設定すると、他の<Directory>セクションにおいて、Allow from allディレクティブで明示的に許可をしない限り、ファイルシステムのどこからもファイルをロードできなくなる。

ファイルシステム上の他のセクションへのAliasを作りたいのであれば、次のように明示的に許可を与える必要がある。

```
Alias /example /var/example
<Directory /var/example>
    Order allow,deny
```

```
    Allow from all
</Directory>
```

参照

- http://httpd.apache.org/docs/mod/mod_access.html

レシピ 6.26　ユーザごとに使えるメソッドを制限する

課題

あるユーザが特定のメソッドを使えるようにして、他のユーザには使えないようにしたい。例えば、グループAのユーザはGETとPOSTの両方を使うことができるが、それ以外の人はGETだけしか使えないようにしたい。

解決

Limitディレクティブを使って、メソッドごとにユーザ認証を適用すればよい。

```
AuthName "Restricted Access"
AuthType Basic
AuthUserFile filename
Order Deny,Allow
Allow from all
<Limit GET>
    Satisfy Any
</Limit>
<LimitExcept GET>
    Satisfy All
    Require valid-user
</Limit>
```

解説

あるHTTPメソッドを使ったアクセスは許可したいが、それ以外は禁止したい、ということはよくある。例えば、誰でもGETメソッドでドキュメントを取得できるようにしたいが、POSTメソッドでデータをアップロードすることができるのはサイトの管理者だけにしたいということがある。

見落としがないよう、すべてのメソッドを列挙するのではなく、LimitExceptディレクティブを使うことが重要だ。

参照

- http://httpd.apache.org/docs/mod/mod_auth.html
- http://httpd.apache.org/docs/mod/mod_access.html
- http://httpd.apache.org/docs/mod/core.html#limit

- http://httpd.apache.org/docs/mod/core.html#limitexcept

レシピ 6.27 Range リクエストを制限する

課題

特定のスコープにあるドキュメントについて、クライアントが部分的にダウンロードすることを禁止し代わりにドキュメント全体をリクエストするよう強制したい。

解決

ErrorDocument 403 を置き換えて、Range ヘッダ付きのリクエストを処理できるようにする。このために、httpd.conf ファイルの適切な <Directory> コンテナか、そのディレクトリの .htaccess ファイルに、次のように追加する。

```
SetEnvIf "Range" "." partial_requests
Order Allow,Deny
Allow from all
Deny from env=partial_requests
ErrorDocument 403 /forbidden.cgi
```

そして、サーバの DocumentRoot にある forbidden.cgi という名前のファイルに、次のように書く。

```
#!/usr/bin/perl -w
use strict;
my $message;
my $status_line;
my $body;
my $uri = $ENV{'REDIRECT_REQUEST_URI'} || $ENV{'REQUEST_URI'};
my $range = $ENV{'REDIRECT_HTTP_RANGE'} || $ENV{'HTTP_RANGE'};
if (defined($range)) {
    $body = "You don't have permission to access "
        . $ENV{'REQUEST_URI'}
        . " on this server.\r\n";
    $status_line = '403 Forbidden';
}
else {
    $body = "Range requests disallowed for document '"
        . $ENV{'REQUEST_URI'}
        . "'.\r\n";
    $status_line = '416 Range request not permitted';
}
print "Status: $status_line\r\n"
    . "Content-type: text/plain;charset=iso-8859-1\r\n"
    . "Content-length: " . length($body) . "\r\n"
```

```
    . "\r\n"
    . $body;
exit(0);
```

または、mod_rewriteを使って、Rangeヘッダの付いたリクエストを捕捉してもよい。そのためには、httpd.conf ファイルの適切な <Directory> コンテナか、そのディレクトリの .htaccess ファイルに、以下のように追加する。

```
RewriteEngine On
RewriteCond "%{HTTP:RANGE}" "."
RewriteRule "(.*)"          "/range-disallowed.cgi" [L,PT]
```

そして、サーバの DocumentRoot にある range-disallowed.cgi という名前のファイルに、次のように書く。

```
#!/usr/bin/perl -w
use strict;
my $message = "Range requests disallowed for document '"
    . $ENV{'REQUEST_URI'}
    . "'.\r\n";
print "Status: 416 Range request not permitted\r\n"
    . "Content-type: text/plain;charset=iso-8859-1\r\n"
    . "Content-length: " . length($message) . "\r\n"
    . "\r\n"
    . $message;
exit(0);
```

解説

どちらの解決策も、目的を達成するために少しごまかしをしている。

最初の解決策は、ErrorDocument 403 のスクリプトを書き換えて、実際の「アクセス禁止」状態と Range リクエストの両方を処理できるようにしている。リクエストヘッダにRangeフィールドが含まれていると、SetEnvIfディレクティブで、環境変数partial_requestsを設定する。この環境変数が設定されていると、Denyディレクティブによって、そのリクエストに対して403 Forbiddenステータスを返そうとする。そして、ErrorDocumentディレクティブで、403ステータスを処理するスクリプトを宣言している。このスクリプトは、リクエストヘッダにRangeフィールドがあるかどうかをチェックし、あれば「ここではRangeリクエストをすることはできません」と返し、なければ本当の「ドキュメントのアクセスが禁止されました」を返す。

2番目の解決策はmod_rewriteを使って、Rangeヘッダフィールドを含むリクエストがあれば、これを処理するカスタムスクリプトに書き換える。このスクリプトは適切なステータスコードとメッセージを返すようになっている。この解決策の「ごまかし」というのは、リクエスト自体は有効で成功するはずなのに、強引にレスポンスステータスを失敗に書き換えているところだ。

参照

- http://httpd.apache.org/docs/mod/mod_setenvif.html

- http://httpd.apache.org/docs/mod/mod_access.html
- http://httpd.apache.org/docs/mod/mod_rewrite.html

レシピ 6.28 mod_evasive を使って DoS 攻撃を防御する

課題

サーバをDoS（Denial of Service）攻撃から防御したい。

解決

http://www.zdziarski.com/projects/mod_evasive/からmod_evasiveを入手し、次のように設定すればよい。

```
DOSPageCount        2
DOSPageInterval     1
DOSSiteCount        50
DOSSiteInterval     1
DOSBlockingPeriod   10
```

解説

mod_evasiveはたった1つの簡単な作業を、とてもうまくやってくれるサードパーティ製モジュールだ。このモジュールは、サイトがDoS攻撃（サービス拒否攻撃）を受けているのを検知し、そのまま実行し続けて大きなダメージを受けるのを防いでくれる。

「サービス拒否」というのは、かなり幅広い意味で使われる用語である。ここでは、サーバが過負荷になって、正当なクライアントとの通信を含め、他に何もできなくなるようなコネクションによる攻撃を指している。Apacheの場合、通常は、1秒間に何百ものHTTPリクエストが送られると、このような状態になる。攻撃者はレスポンスを待つことさえせず、すぐに切断してまた別のリクエストを送信する。このとき、Apacheはもはや存在しないクライアントに対してレスポンスを返そうとしてしまう。

mod_evasiveは、1つのクライアントが、短時間のうちに複数のリクエストを送信するのを検知し、そのクライアントからのさらなるリクエストを拒否する。このリクエストを禁止する期間は、とても短くてもよい。リクエストが同じホストからだと検出されると、禁止する期間も更新されるようになっている。

この設定はリクエストに2つの制限を加えている。最初に、DOSPageディレクティブは、同じアドレスのクライアントが1秒間に2回以上同じURLをリクエストすると、そのクライアントをブロックすることを示している。次に、DOSSiteCountおよびDOSSiteIntervalディレクティブは、同じアドレスのクライアントが1秒間に50以上のURLをリクエストすると、そのクライアントをブロックすることを示している。この値は、比較的大きくなっている。これは、1つのページにたくさんの画像が含まれている場合にも、1つのクライアントからたくさんのリクエストが送られることがあるためだ。

DOSBlockingPeriodディレクティブは、クライアントがブロックされる期間を設定する。この場合、10秒間ブロックされることになる。この期間は短いように見えるが、同じクライアントが接続しようとする（そしてブロックされる）たびに、ブロック期間はまた最初から開始するので、永久に延長される。

参照

- http://www.zdziarski.com/projects/mod_evasive

レシピ6.29　mod_securityを使ってApacheをchrootする

課題

Apacheをもっと安全にするために、chrootしたい。

解決

Apacheをchrootするには、いろいろな方法がある。最も簡単な方法の1つは、mod_securityを使って、次のようなディレクティブを追加することだ。

```
SecChrootDir /chroot/apache
```

解説

chrootは、プログラムがjail（刑務所）の中で動くようにするUnixコマンドだ。つまり、このコマンドを実行すると、アクセス可能なファイルシステムは別のパスに置き換えられ、動作するアプリケーションは、新しいファイルシステムの外部にあるファイルにはアクセスできなくなる。こうすることによって、プログラムがアクセスできるリソースをコントロールすることができ、そのディレクトリの外部にあるファイルへ書き込んだり、そのディレクトリにないプログラムを実行したりすることを防ぐことができる。これにより、攻撃者は必要なツールにアクセスできなくなり、たくさんのセキュリティ上の弱点を防ぐことができる。

chrootを使ったときの問題は、とても面倒なことだ。例えば、Apacheをchrootすると、Apacheを動かすのに必要なすべてのライブラリやその他のファイルを、新しいファイルシステムへコピーしなければならない。mod_sslを動かしているなら、OpenSSLライブラリもすべてchroot jailにコピーする必要がある。また、Perl CGIプログラムを使っていると、Perlやそのモジュールすべてをchrootディレクトリにコピーしなければならない。

mod_securityは起動時ではなく、子プロセスをfolkする直前にApacheをchrootすることによって、このややこしい問題をうまく回避している。これは先に述べたmod_sslでの問題も解決してくれるが、Perlの問題は解決してくれない。Perl CGIプログラムはfolkした子プロセスによって実行されるためだ。しかし、chroot jailに移動したりコピーしたりする必要があるものはかなり減らすことができ、残りはCGIプログラムとして動かすものぐらいになるだろう。起動時に、Apacheが必要とするライブラリを移動したりコピーしたりする必要はない。これによって、面倒さが軽減され、実際にApacheをchrootしやすくなる。さもないと、多くの人はこの不便さに耐えてくれないだろう。

Apache 1.3を動かしているなら、LoadModuleリストの最初にmod_securityを置く必要がある。こうしないと、mod_securityはどう調整するかを制御することができなくなる。chrootは適切なタイミングで使うことが重要だ。そのためには、mod_securityは他のモジュールに制御される前に、最初に目的を達成する必要がある。

Apache 2.0 を動かしているなら、こうした考慮は不要になる。なぜなら、モジュールはいつロードされるかわかっていて、正しいタイミングで動かすことができるためだ。

参照

- http://modsecurity.org

レシピ 6.30 Apache 2.2 認証に移行する

課題

認証を使っていたが、Apache 2.2 に移行したい。しかし、認証の仕組みが全く違っている。

解決

認証はApache 2.2で再設計されて、認証と許可を別のステップとして完全に分離し、独立して設定できるようになった。最初は変化のための変化のように見えるかもしれないが、この分離を理解すると、新しい設定構文は非常に意味があり、この変化は理にかなっているとわかるだろう。

解説

認証と許可という用語は、本章の始めにやや詳しく解説している。これまでApacheはこの2つの概念の境界が曖昧で、一方の設定に制約を与えずに、もう一方を設定するのは難しかった。例えば、Digest認証を使いたいときには、ユーザのリストにテキストファイルを使う必要があった。このような制約は、もはやApache 2.2 にはない。

Apache 2.2で認証と許可を設定するには、3つの決断をしなければならない。実際には、3つの決断それぞれにつき 1 つのモジュールを選ぶ必要がある。

最初に、どの認証タイプを使いたいのか決める必要がある。Digest 認証や Basic 認証を選ぶと、mod_auth_basic や mod_auth_digest を選ぶ必要がある。2.2 以前のバージョンと同様に、AuthType ディレクティブを使うことができる。

```
AuthType Basic
```

次に、認証プロバイダを選択しなければならない。認証情報がどこに、どんな形式(テキストファイル、dbmファイルなどのデータベース)で格納するかを選択する。この選択をするためのディレクティブは、もしBasic認証を使うなら AuthBasicProvider、Digest 認証を使うなら、AuthDigestProvider になる。

```
AuthBasicProvider dbm
```

最後に、許可の要件を定義する必要がある。これは Require ディレクティブを使って設定する。

```
Require user sarah
```

このように、2.2以前と比較すると、認証、許可の設定ディレクティブには、いくらか追加されている項目がある。

```
AuthName "Private"
AuthType Basic
AuthBasicProvider dbm
AuthDBMUserFile /www/passwords/passwd.dbm
Require user sarah isaiah
```

参照

- http://httpd.apache.org/docs/2.2/howto/auth.html

レシピ6.31　mod_securityを使ってウイルスをブロックする

課題

mod_securityサードパーティ製モジュールを使って、Webサーバのページに到達する前に、よくあるセキュリティの脆弱性調査を封じたい。

解決

mod_securityをインストールしていれば(レシピ2.9を参照)、基本の「コアルール」アクセサリパッケージを使って、Webサーバを襲う最も一般的な攻撃や脆弱性調査の多くを封じることができる。コアルールパッケージは、Web上に現れた新しい問題に対応するよう、定期的に更新されている。

解説

コアルールパッケージをインストールして、READMEファイルに書かれた指示に沿って設定するのが、一番簡単だ。さらに、パッケージに含まれるファイルには、そのフォーマットについて説明があり、独自のルールを書くのを簡単にしてくれるだろう。

参照

- http://modsecurity.org/projects/rules

レシピ6.32　Subversionリポジトリに読み出しのみと書き込み可能アクセスを混在させる

課題

Subversionリポジトリにおいて、あるパスを読み出しアクセスのみとし、それ以外は書き込みアクセスもできるよう、場所によって異なる保護をしたい。

解決

簡単な解決策としては、<LimitExcept>を使って、特定のファイルやパスへの書き込みアクセスには認証を必要とすればよい。

```
<Location "/repos">
    DAV svn
    SVNParentPath "/repository/subversion"
    AuthType Basic
    AuthName "Log in for write access"
    AuthUserFile "/path/to/authfile"
    <LimitExcept GET REPORT OPTIONS PROPFIND>
        Requre valid-user
    </LimitExcept>
</Location>
```

ここで紹介した設定は、Subversionリポジトリ全体に制限を適用している。より自由度があり、きめ細かくコントロールするには、mod_authz_svnモジュールと組み合わせるとよい。

```
LoadModule authz_svn_module   modules/mod_authz_svn.so

<Location "/repos">
    DAV svn
    SVNParentPath "/repository/subversion"
    Order Deny,Allow
    Allow from all
    AuthName "Log in for write access"
    AuthType Digest
    AuthDigestDomain "/repos/"
    AuthDigestFile "/path/to/digest-file"
    AuthzSVNAccessFile "/path/to/access-file"
    <Limit GET PROPFIND OPTIONS REPORT>
        Satisfy Any
    </Limit>
    <LimitExcept GET PROPFIND OPTIONS REPORT>
        Satisfy All
        Require valid-user
    </LimitExcept>
</Location>
```

解説

最初の解決策は単純なアプローチをとっている。要するに「これらのメソッドは安全だが、それ以外のメソッドを使うときにはログインしなければならない」ということだ。

2番目の解決策は、これとmod_authz_svnモジュールの機能を組み合わせて、関連するパスに基づいて、選

択的にアクセスを許可(あるいは拒否)できるようにしている。リポジトリ全体に対してはそのまま読み出しアクセスが可能だ。使われたユーザ名に基づいて、特定のパスだけを制限したいのであれば、<LimitExcept>と</LimitExcept>の行を削除して、<Limit>コンテナを全部削除すればよい。すると、リポジトリにアクセスするときには、ユーザは常にログインを要求される。ユーザに何ができるのか(読み出し、書き込み、あるいは何もできないか)は、AuthzSVNAccessFile ディレクティブに指定した SVN auth ファイルで設定することができる。

参照

- http://httpd.apache.org/docs/2.0/mod/core.html#limitexcept
- http://svnbook.red-bean.com/en/1.4/svn.serverconfig.httpd.html

レシピ 6.33　禁止された URL を隠すために Permanent リダイレクトを使う

課題

ファイルへのアクセスが禁止されているとき、ユーザのブラウザ上には URL を見せたくない。

解決

「ドキュメントが見つかりません」というメッセージページに Permanent リダイレクトするような ErrorDocument スクリプトを追加すればよい。

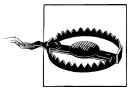
```
Alias "/not-found" "/path/to/documentroot/not-found.html"
ErrorDocument 403 "/cgi/handle-403
```

そして、cgi-bin/handle-403 スクリプトとして、次のようなものを書いておく。

```
#!/usr/bin/perl -w
#
# permanent リダイレクトを強制する
#
print "Location: http://example.com/not-found\r\n\r\n";
exit(0);
```

解説

通常、ドキュメントへのアクセスが禁止されていると、エラーが表示されてもブラウザにはそのURLが表示されたままになる。この解決策にある手順に従うと、サーバがリダイレクトを使って処理をして、禁止されている実際のドキュメントの URL を隠してしまうことができる。ブラウザは、通常のエラーとは違って、ロケーションバーの内容を変更する。



このレシピのタイトルには、「隠す」という言葉を使ったが、これには理由がある。ここで実践したのは、「隠すことによるセキュリティ」と呼ばれるものだ。「探すべきものを正確に知っていればまだ手に入れることができるかもしれないが、それを知られないようにしておきたい」ということだ。ある意味、現実から目をそらして、問題が解決するか、誰にも見つからないことを期待しているようなものである。知識のあるユーザは、ネットワークのトラフィックを調べて、禁止されているファイル名を見つけることができるだろう。

not-found.html ファイルへのリダイレクトは成功するので、ブラウザは間違ったことをしようとしたとは気付かない。リダイレクト先のスクリプトに次のような行を入れて、少しスパイスを加えることができる。

```
print "Status: 403 Forbidden\r\n\r\n";
```

こうすると、ブラウザがアクセス禁止されたファイルをリクエストすると、サーバが「向こうを見に行きなさい」と答え、ブラウザが素直に見に行くと、403 Forbiddenエラーが表示される。しかし、ブラウザのロケーションフィールドには、not-found.htmlドキュメントのURLが表示されることになる。実際にアクセス禁止されたドキュメントの URL ではない。

参照

- http://httpd.apache.org/docs/2.0/mod/core.html#errordocument
- RFC 3875 のセクション 6.3.2 と 6.3.3（ftp://ftp.rfc-editor.org/in-notes/rfc3875.txt）

7章
SSL

SSL（Secure Socket Layer）は、安全なWebサイトを構築するための標準的な方法だ。サーバとクライアント間の通信トラフィックをSSLで暗号化することによって、通信トラフィックを傍受している第三者から通信内容を保護することができる。

いったんSSLセッションが確立すると、以後のすべての通信トラフィックは暗号化される。このとき、リクエストしているURLでさえも暗号化されている。

暗号化の正確なメカニズムについては、SSLの仕様書で幅広く解説されており、http://wp.netscape.com/eng/ssl3で読むことができる。もっとわかりやすいSSLの解説として、mod_sslのマニュアルに目を通すことを勧める。これはhttp://httpd.apache.org/docs/2.2/sslで見つけることができる。このマニュアルはmod_sslのセットアップ方法について特に詳しく解説しているだけでなく、SSLの背後にある一般的な理論についても解説しており、その概念を図解している。

TLS 1.0（RFC 2246）の仕様書も見ておくとよいだろう。これはSSLの次世代のプロトコルと見なされているものだ。http://www.ietf.org/rfc/rfc2246.txtで完全な仕様書を読むこともできるし、もっとわかりやすい説明が、http://en.wikipedia.org/wiki/Transport_Layer_Security（英語）にある。

本章では、インストール方法も含め、安全なサーバなら実施しておきたい話題を紹介する。

レシピ 7.1　SSL をインストールする

課題

Apache サーバに SSL をインストールしたい。

解決

この課題に対する解決策は、最初にApacheをどうやってインストールしたのか（あるいは、SSLを組み込むためにApacheを再ビルドしたいかどうか）によって違ってくる。

Apacheのバイナリ配布を使ってインストールした場合、バイナリディストリビューションの入手先に戻って、SSLを追加するのに必要なファイルを探すことが最善の策だ。

Apacheをソースから自分でビルドした場合、Apache 1.3かApache 2.xのどちらを動かしているかによって解決策は違ってくる。

Apache 1.3では、SSLは拡張モジュールであり、Apacheの入手先とは別のところから入手してインストールしなければならない。mod_ssl(http://www.modssl.org)とApache-SSL(http://www.apache-ssl.org)という2つの選択肢がある。どちらを選ぶかによって、インストール手順は少し変わってくる。

Apache 2.xをソースからビルドした場合には、状況はまだ簡単だ。組み込みモジュールの1つとしてSSLを加えるために、Apacheのビルド時の`./configure`の引数に`--enable-ssl`を追加するだけでよい。

サードパーティ製モジュールの詳しいインストール方法について、特に、ソースコードから自分でビルドしたのではなくApacheのバイナリディストリビューションからインストールした場合には、1章と2章を参照すること。

Windows上のApache 2.0および2.2にSSLをインストールしようとする場合には、http://httpd.apache.org/docs/2.0/platform/win_compiling.htmlおよびhttp://httpd.apache.org/docs/2.2/platform/win_compiling.htmlに、Windows上でのコンパイルについての解説がある。Windows上でApache 1.3を使っていて、SSLをインストールしたい場合には、SSLディストリビューションに付属のファイルINSTALL.Win32を調べるか、http://tud.at/programm/apache-ssl-win32-howto.php3にある"The Apache + SSL on Win32 HOWTO"を参照するとよいだろう。

最後に、Apache SSLモジュールは、ApacheとOpenSSLライブラリ間のインターフェイスであり、動かすためにはOpenSSLをインストールしておかなければならないことに注意しよう。OpenSSLライブラリはhttp://www.openssl.orgから入手することができる。サーバにはすでにOpenSSLライブラリがインストール済みかもしれないが、攻撃から防御するためには、最新のセキュリティパッチを適用した最新版を入手することを推奨する。

解説

どうしてこんなに複雑なのだろうか。いろいろな理由があるが、最大の理由は、暗号化の合法性に関するものだ。長年にわたり、米国では暗号は規制された技術であった。Apacheは主に米国から配布されており、パッケージに暗号化技術を組み込んで配布することに関して、多大な警戒がなされてきた。その後、法律に大きな変更が加えられ、Apache 2.0と一緒にSSLを出荷してもよくなったが、SSLを有効にしたコンパイル済みのApacheバイナリを配布することについては、まだ疑わしいとされている。

Microsoft Windowsでは、ほとんどの人がコンパイラをすぐに使えない状況なので、さらに状況は悪い。そのため、Windows上のApacheサーバでSSLを有効にするには、サードパーティによるバイナリビルドを入手する必要があるだろう。Windows上でSSLが利用可能なApache 2.0のコンパイルについて先に紹介したURLではコンパイラを持っていることが前提になっている。また、SSLを使えるようにしたApache 1.3のビルド方法について解説したドキュメントでは、Unix系OSに匹敵するパフォーマンスが得られないためにWindowsにはApache 1.3を使わないよう推奨している。

参照

- http://httpd.apache.org/docs-2.0/platform/win_compiling.html
- http://tud.at/programm/apache-ssl-win32-howto.php3
- http://www.openssl.org
- http://www.modssl.org

- http://www.apache-ssl.org

レシピ 7.2　Windows 上に SSL をインストールする

課題

Microsoft Windows に、SSL を利用可能な Apache をインストールしたい。

解決

http://apachefriends.org から XAMPP を取得し、それをインストールする。

解説

レシピ 7.1 で説明したように、Microsoft Windows 上でソースから SSL 対応の Apache をビルドすることは確かに可能だ。しかし、正直に言って、一般人の専門知識を超えているだろう。

幸運にも、ApacheFriendsの親切な人たちがXAMPPと呼ばれるバイナリディストリビューションを作っており、mod_ssl が利用できる Apache が含まれている。このパッケージにはさらに、MySQL や PHP、Perl など、Web サイト開発でよく使うツールも含まれている。

したがって、苦労を減らすためにも、ApacheFriendsのみんなが成し遂げた偉大な仕事を利用するのがよいだろう。XAMPP パッケージをインストールしよう†。

レシピ 7.3　自己署名の SSL 証明書を生成する

課題

SSL サーバで使える自己署名の証明書を生成したい。

解決

OpenSSL に付属する openssl コマンドラインプログラムを使って、次のように実行すればよい。

```
% openssl genrsa -out server.key 1024
% openssl req -new -key server.key -out server.csr
% cp server.key server.key.org
% openssl rsa -in server.key.org -out server.key
% openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

これらのファイルを Apache サーバの設定ディレクトリ（例えば、/www/conf/）に移動し、httpd.conf 設定ファイルに、次の行を追加する。

† 現在、Apache Software Foundation では、OpenSSL を組み込んだ Windows 用 MSI パッケージも配布しており、これを利用することもできる。http://httpd.apache.org/download.cgi を参照。

```
SSLCertificateFile "/www/conf/server.crt"
SSLCertificateKeyFile "/www/conf/server.key"
```

解説

SSL証明書はSSL通信の核心部分であり、安全なサーバを動かすために必要不可欠なものだ。したがって、証明書を生成することが、安全なサーバを設定するのに必要な最初のステップになる。

鍵を生成するには何段階もの処理が必要だが、手順は非常に簡単だ。

秘密鍵の生成

最初のステップでは、秘密鍵を生成する。SSLは秘密鍵と公開鍵を使う暗号化システムだ。秘密鍵はサーバに置かれ、公開鍵はクライアントがサーバに接続するたびに送られる。クライアントは公開鍵を使ってデータを暗号化し、それをサーバに送り返す。

openssl プログラムに渡している最初の引数genrsaは、RSA鍵を生成するように指示している。これは主要なブラウザすべてでサポートされている暗号化アルゴリズムだ。

望むなら、ランダム性のソースに何を使うか指定してもよい。-rand引数には1つ以上のファイル名を指定することができ、これが乱数生成器のキーとして使われる。-rand引数を指定しなかった場合、OpenSSLは/dev/urandom があれば、これをデフォルトで使おうとする。/dev/urandom がなければ、/dev/random を使おうとする。暗号化をセキュアにするには、よいランダム性のソースを使うことが重要だ。/dev/urandomも/dev/random もないシステムでは、edg のような乱数生成器をインストールすることを検討するべきだ。edg について詳しく知りたければ、OpenSSLのWebサイトhttp://www.openssl.org/docs/crypto/RAND_egd.html を参照すること。

-out引数は、生成する鍵ファイルの名前を指定する。この引数がフルパスでなければ、コマンドを実行しているディレクトリにファイルが作られる。鍵ファイルのファイル名は実際のところ重要ではないが、使用するホスト名にちなんで名付けておくと、ファイルを整理しやすいだろう。

そして最後の 1024 は、何ビットの秘密鍵を生成するかを openssl に指定している。

出力は次のようなものになるだろう。

```
Generating RSA private key, 1024 bit long modulus
.....................................................++++++
.....++++++
e is 65537 (0x10001)
```

CSR の生成

次のステップでは、CSR(証明書署名要求)を生成する。こう呼ばれる理由は、結果として得られるファイルが署名のためにCA(認証局)に送られる、すなわち、署名を要求するからだ。(証明書は単に署名された鍵であり、その鍵が有効であり、正当な人が所有しているということを第三者が証明したものだ。)

CAとは、SSL証明書に署名することができる第三者のことだ。CAは通常、数十ある会社の1つであり、SSLサーバで使う SSL 証明書に署名することをビジネスにしている。証明書が CA によって署名されると、ブラウザは自動的にその証明書を有効なものと見なして受け入れる。証明書に署名したCAが、ブラウザにある信

頼されたCAのリストに載っていない場合、ブラウザは警告を出し、見知らぬCAが署名した証明書であることをユーザに通知し、この証明書を受け入れたいかどうか尋ねる。

これは少し手順を簡略化しすぎかもしれないが、このレシピの目的には十分だろう。

証明書を自分自身で署名することもできるが、これについては次のステップで説明する。

-key引数には、どの証明書のCSRを生成するか鍵を指定し、-out引数には、生成するファイル名を指定する。

主要なブラウザが警告やコメントなしに受け入れてくれる証明書がほしいなら、小切手かクレジットカード情報を添えて、こうしたCAの1つにcsrファイルを送ればよい。

このステップでは、たくさんの質問を尋ねられるだろう。この質問に対する答えは証明書の一部となり、証明書が信頼されたソースからやってきたということをブラウザが検証するのに使われる。エンドユーザは、Webサイトに接続するときはいつでも、その詳細を調べることができる。

質問は次のようなものだ。

```
Country Name (2 letter code) [GB]: EX
State or Province Name (full name) [Berkshire]: CO
Locality Name (eg, city) [Newbury]: Example City
Organization Name (eg, company) [My Company Ltd]: Institute of Examples
Organizational Unit Name (eg, section) []: Demonstration Services
Common Name (eg, your name or your server's hostname) []: www.example.comv
Email Address []: big-cheese@example.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

これらの値は、Common Nameを除いて、すべてオプションだ。Common Nameには正しい値を指定しなければならない。Common Nameが、この証明書を使うサーバのホスト名になる。ここに書いたホスト名はサイトにアクセスするのに使われるホスト名と正確に一致しなければならない。これが間違っていると、ユーザがWebサイトに接続するたびに、警告メッセージが出ることになる。

パスフレーズの削除

最初のステップでは、秘密鍵のパスフレーズを入力した。これによって鍵を暗号化し、パスフレーズを知っている人しか鍵の内容を読めなくしている。

これには副作用があり、Apacheサーバを起動するたびに、パスフレーズの入力が必要になる。これはかなり不便で、Webサーバの起動時には、いつも手作業のステップが必要になってしまう。パスフレーズを入力できる人が近くにいないときに、リブートやApacheサーバを自動的に再起動する場合に特に問題となる。

この問題を解消するために、鍵からパスフレーズを削除する。何か失敗した場合のため、鍵のバックアップをコピーしておくとよい。そして、パスフレーズを削除するコマンドを発行し、暗号化されていない鍵を取得する。ファイルのパーミッションを変更して、rootだけがこのファイルを読めるようにするのを忘れないこと。そうしないと、誰かがファイルを盗み見して、あなたのふりをしてWebサイトを運営することもで

きてしまう。

鍵の署名

CSRをCAに送らないのであれば、代わりに自分で自分の公開鍵に署名(自分の公開鍵に署名することは自己署名証明書になるので、「自分の証明書に署名する」と呼ばれる)することができる。これも完全に利用可能な証明書であり、お金を少し節約できる。これは、特にテスト目的で役に立つが、小さなサイトや社内ネットワークのサーバ上でSSLを動かしている場合もこれで十分だろう。

鍵に署名する手続きは、その鍵が所有者として登録された人によって実際に所有されている、ということを署名者が信頼するということを意味している。Entrustなどの商用CAにお金を払うと、実際に所有者を調査し、ある程度の確度で所有者が本当に本人であることを検証してくれる。そして、CAはその公開鍵に署名して証明書に承認のスタンプを押して送り返し、本人であることを世界に証明してくれる。

ここで取り上げた例では、鍵をその鍵自身で署名している。これは基本的に自分で自分を信頼していることになり、少しばかげている。しかし、実際のSSL暗号化の目的に対しては十分だ。望むなら、OpenSSLに付属するCA.plスクリプトを使って、自分自身のCA証明書を作成することができる。この方法の利点は、このCA証明書をユーザに配布し、ユーザがこれをブラウザにインストールできることだ。こうするとブラウザは、この証明書や同じCAで作成した他の証明書を、自動的に信頼することができる。これは、同じCAによって署名された証明書を使って複数のSSLサーバを運用しているような大きな会社で特に役立つだろう。

コマンドに与える引数の中で、最も重要なものの1つは-day引数である。これは、証明書が何日間有効であるかを指定する。商用CAから証明書を購入するつもりであれば、おそらく30日間有効な自己署名の鍵を作成しておけばよいだろう。商用の証明書が届くのを待っている間、この鍵を利用することができる。サーバで実際に使うための鍵を生成するなら、1年くらいにしてもよいだろう。そうすれば、それほど頻繁に新しい鍵を生成する必要はなくなる。

-signkey引数は、証明書を署名するのに使う鍵を指定する。これは最初のステップで生成した秘密鍵でもよいし、先に述べたCA.plスクリプトを使って生成したCA秘密鍵でもよい。このステップがうまくいくと、次のようなメッセージが出力されるはずだ。

```
Signature ok
subject=/C=US/ST=KY/L=Wilmore/O=Asbury College/OU=Information
Services/CN=www.asbury.edu/Email=rbowen@asbury.edu
Getting Private key
```

サーバの設定

鍵と証明書が生成できれば、これらをサーバで使うことができる。解決策で示したように設定に2行追加すればよい。

簡単な方法

これまで長くて大変な方法を見てきたが、もっと簡単な方法があることも知っておくべきだろう。OpenSSLには、CA.plという名前の便利なスクリプトが付属しており、鍵の生成手順を簡単にしてくれる。CA.plの使

い方は、レシピ7.4で説明するので、そこを読むと動作を理解できるだろう。しかし、スクリプトの背後で何が行われているかを知っておくことは、役に立つはずだ。少なくとも筆者たちはそう考えている。また、証明書の作り方に関しても、かなり自分でコントロールできるようになるだろう。

参照

- openssl ツールの manpage
- CA.pl スクリプトの manpage
- http://www.openssl.org/docs/apps/CA.pl.html にある CA.pl のドキュメント

レシピ 7.4　信頼された CA を生成する

課題

ブラウザが警告メッセージを出さずに受け入れてくれる SSL 鍵を生成したい。

解決

次のコマンドを実行する。

```
% CA.pl -newca
% CA.pl -newreq
% CA.pl -signreq
% CA.pl -pkcs12
```

解説

レシピ 7.3 では、鍵を生成して署名するという一連の長い手順を解説した。幸いなことに、OpenSSL にはこの手順の大部分を自動化してくれるスクリプトが付属しており、すべての引数を覚えておく必要はない。これはCA.plという名前のスクリプトで、SSLライブラリがインストールされている場所、例えば、/usr/share/ssl/misc/CA.pl にあるはずだ。

この解決策では、詳細をかなり省略しているが、鍵と証明書を生成する過程では、たくさん質問されるだろう。このレシピを成功させるには、このスクリプトがあるディレクトリの中でスクリプトを実行する必要があるということに注意しよう。

証明書のパスフレーズを省略して、サーバを起動するたびにパスフレーズが要求されないようにするには、証明書要求を生成するときに -newreq ではなく、-newreq-nodes を使えばよい。

この一連のコマンドを実行した後、-newreqと-signreqコマンドを繰り返すことで、証明書をさらに生成することができる。

これらのコマンドを実行すると、たくさんのファイルが生成される。ファイルnewcert.pemはSSLCertificateFile ディレクティブに指定するファイルになり、ファイル newreq.pem は SSLCertificateKeyFile ディレクティブに指定するファイルになる。ファイル demoCA/cacert.pem は CA 証明書ファイルであり、ユーザのブラウザにインポートする必要がある。このファイルをインポートすると、このCAによって署名された証明書を自動的に信頼してくれるようになる。また、最後に、newcert.p12はある特定のブラウザのためのファイル

であり、demoCA/cacert.pem と同じ役割を果たす。

CA 証明書のインポート

ユーザが Internet Explorer を使っているなら、インポートするには特別なファイルを作る必要がある。次のコマンドを使う。

```
openssl x509 -in demoCA/cacert.pem -out cacert.crt -outform DER
```

こうしてできた cacert.crt ファイルを、ユーザに送ればよい。

ファイルをクリックすると、SSL証明書ウィザードが立ち上がり、ブラウザにCA証明書のインストール手順を案内してくれる。

Firefoxのようなブラウザはcacert.pemファイルを直接インポートすればよい。Firefox 2の場合、メニューから［オプション］を選び、［詳細］の［証明書］の［証明書の表示］をクリックし、［認証局証明書］タブの［インポート］で証明書ファイルを選択する。

CA証明書をインポートすると、そのCAが署名したすべての証明書は、ブラウザで警告なしに利用できるようになる。

参照

- CA.pl スクリプトの manpage
- http://www.openssl.org/docs/apps/CA.pl.html にある CA.pl ドキュメント

レシピ 7.5 サイトの一部を SSL 経由で提供する

課題

サイトのある特定の部分を、SSL 経由でのみ利用できるようにしたい。

解決

これは httpd.conf ファイルを変更することで実現できる。Apache 1.3 の場合は、次のような行を追加すればよい。

```
Redirect /secure/ https://secure.example.com/secure/
```

Apache 2.0 の場合には、次のように設定すればよい。

```
<Directory /www/secure>
    SSLRequireSSL
</Directory>
```

SSLRequireSSL ディレクティブはリダイレクトを発生させないことに注意しよう。非SSLリクエストは単に禁止されるだけだ。

あるいは、Apacheのどのバージョンであっても、mod_rewriteを使って以下のように設定することができる。

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^/(.*) https://%{SERVER_NAME}/$1 [R,L]
```

解説

最善の方法は、1つではなく2つの異なるバーチャルホストを用意して、通常のページとSSLで保護されたページを別々に処理することだろう。2つのバーチャルホストは、同じコンテンツを参照していても、別のポートで、別の設定で動作する。最も重要なのは、ブラウザがこれらを完全に別のサーバであると考えることだ。自分たちもそう考えるべきだ。

特定のディレクトリだけSSLを有効にしようと考えるのではなく、あるサーバから別のSSLを有効にしたサーバへ、リクエストをリダイレクトすることを考えるべきだ。

Redirectディレクティブは、パス情報を保持することに注意しよう。/secure/something.html宛てのリクエストは、https://secure.example.com/secure/something.html にリダイレクトされる。

Redirect ディレクティブを置く場所には注意が必要だ。HTTP（非 SSL）のバーチャルホスト宣言の中にだけ置くようにすること。設定ファイルのグローバルセクションに置いてしまうと、ループを引き起こしてしまうおそれがある。新しいURLがリダイレクトの要件とマッチしてしまい、それ自身がリダイレクトしてしまうためだ。

最後に、サイト全体をSSL経由でのみ利用可能にしたいなら、特定のディレクトリではなく、すべてのURLをリダイレクトしてしまえばよい。

```
Redirect / https://secure.example.com/
```

もう一度、この Redirect ディレクティブが非 SSL のバーチャルホスト宣言の中にあることを確認しよう。

この状況に対して、RedirectMatchを使ったり、いろいろなRewriteRuleディレクティブを使ったりするなど、さまざまな解決策が提案されている。こうしたことが必要となる特別なケースもあるが、多くの場合はここで紹介した単純な解決策でうまくいくだろう。特に、メインサーバの設定ファイルにはアクセスできないが .htaccess にはアクセスできるという場合には、この解決策を使わざるを得ないだろう。

セットアップ全体は次のようなものになる。

```
NameVirtualHost *

<VirtualHost *>
    ServerName regular.example.com
    DocumentRoot /www/docs

    Redirect /secure/ https://secure.example.com/secure/
</VirtualHost>
```

```
<VirtualHost _default_:443>
    SSLEngine On
    SSLCertificateFile /www/conf/ssl/ssl.crt
    SSLCertificateKeyFile /www/conf/ssl/ssl.key

    ServerName secure.example.com
    DocumentRoot /www/docs
</VirtualHost>
```

他の2つの解決策の方がわかりやすいが、利用するにはそれぞれ少し追加の要件がある。

2番目の解決策は、SSLRequireSSLを使っており、これはApache 2.xでしか使えない。このディレクティブは、こうした課題を解決するために特別に追加されたものだ。ある<Directory>セクションにSSLRequireSSLディレクティブを指定すると、そのディレクトリに対する非SSLアクセスが禁止される。これは、ユーザをSSLホストにリダイレクトするわけではなく、単に非SSLアクセスを禁止しているだけだ。

3番目の解決策は、RewriteCondディレクティブとRewriteRuleディレクティブを使っているため、mod_rewriteがインストール済みで有効になっている必要がある。RewriteCondディレクティブを使ってクライアントがすでにSSLを使っているかどうかチェックし、SSLが使われていないときだけRewriteRuleディレクティブを呼び出す。この場合、同じコンテンツだが、HTTPでなくHTTPSを使ったリクエストにリダイレクトされる。

参照

- http://httpd.apache.org/docs-2.0/mod/mod_ssl.html
- http://httpd.apache.org/docs/mod/mod_alias.html
- http://httpd.apache.org/docs/mod/mod_rewrite.html

レシピ 7.6 クライアント証明書を使って認証する

課題

クライアント証明書を使って、サイトへのアクセスを認証したい。

解決

httpd.confファイルに、次のようにmod_sslディレクティブを追加する。

```
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile conf/ssl.crt/ca.crt
```

解説

イントラネット向けサイトや友達や家族のためのWebサイトといった、幸いにも、小さな閉じたユーザコミュニティであれば、クライアント証明書を配布してそれぞれのユーザの身元を確認することは可能だ。

クライアント証明書を作って、自分のCA証明書で署名する。そして、この解決策にあるように、SSLCACertificateFile ディレクティブに、その CA 証明書ファイルの場所を指定すればよい。

クライアント証明書は、証明書のCN(Common Name)がクライアント証明書の所有者の名前になることを除いて、サーバ証明書と同じ手順で作成することができる。

参照

- レシピ 7.3
- http://httpd.apache.org/docs-2.0/mod/mod_ssl.html

レシピ 7.7　バーチャルホストで SSL を使う

課題

1 つの IP アドレスを使って、複数の SSL ホストを動かしたい。

解決

この課題への回答はいくつかあり、考え方によって違ってくる。

まず、IP アドレスとポートごとに 1 つの SSL ホストしか動かせない、というのが公式な正しい回答だ。これは SSL の動作メカニズムによるもので、Apache に何か制約があるわけではない。同じ IP アドレスとポート上に複数のSSLホストを動かそうとすると、ブラウザに警告メッセージが表示されることになるだろう。これは、ブラウザが間違った証明書を受け取るためだ。

もう 1 つ別の回答は、ワイルドカード証明書を使うことだ。これについては次のレシピで説明する。

最後に、警告メッセージを気にしないのなら、通常通り、ネームベースのバーチャルホストをセットアップしてもよい。この場合 Apache はすべてのバーチャルホストに対して同じ証明書を使うだけだ。

近い将来、この課題に対する別の解決策ができるかもしれない。非常に多くの頭のいい人たちがこの問題に取り組んでいるが、残念ながら、まだ完全には解決されてはいない。

解説

ブラウザがhttps (SSL) リクエストを要求すると、最初に、SSL暗号化をセットアップするための証明書がブラウザに送られる。これは、リクエストするURLをブラウザがサーバに伝える前に行われるため、特定のバーチャルホストを選択することができない。したがって、特定のIPアドレスとポートに1つ以上の証明書を関係付けることはできない。

これには、基本的に 3 種類の解決策がある。問題を無視してしまうか、複数のホスト名で 1 つの証明書を使うか、複数の IP アドレスとポートを使うかのいずれかだ。それぞれ順番に解説していこう。

問題を無視する

ある状況では、問題を無視しても構わないかもしれない。テストサーバであったり、ユーザ数が少なくユーザに事情を説明できるのであれば、これは完全に許容できるシナリオになるだろう。

この場合、ネームベースのバーチャルホストをセットアップして、それぞれに同じ証明書を使う。しかし、

バーチャルホストのどれかに接続したときに、そのホスト名が証明書のCommon Nameと一致しない場合には、ブラウザは警告メッセージを表示するだろう。Firefoxでは、次のような意味の画面を表示する。

「www.example1.comに接続を確立しようとした。しかし、受け取ったセキュリティ証明書はwww.example2.comのものである。誰かがこのWebサイトの通信を傍受しようとしている可能性がある。ここで提示された証明書がwww.example1.comのものではない疑いがあるなら、接続を中断して、サイト管理者に知らせよう」

この場合もサイト管理者はすべてうまくいっていて、実際には問題がないということがわかっている。しかし、ユーザ側では、いろいろ違った反応をするだろう。パニックになる人もいれば、すぐに[キャンセル]をクリックする人もいる。メッセージを全く無視して[OK]をクリックしてしまう人もいるが、これは間違いだ。このような警告メッセージを無視していると、いつかきっと問題に巻き込まれてしまうだろう。あるいは、実際にどういう行動をとったらよいか、サイト管理者に連絡する人もいるだろう。いずれにせよ、現実に安全なWebサイトを運営して、クレジットカード取引のような業務を行っている場合には、有効な解決策ではないことは明らかだ。

複数のホストで1つの証明書を使う

すべてのホスト名が同じドメインに属しているなら、複数のホスト名で1つの証明書を使うことができる。これはワイルドカード証明書と呼ばれており、次のレシピ7.8で解説する。

複数のアドレスを使う

推奨する解決策は、SSLホストごとに異なるIPアドレスで動かすことだ。それがだめなら、同じIPアドレスだが別のポートでサイトを動かしてもよい。IPベースのバーチャルホストとポートベースのバーチャルホストについては、4章のバーチャルホストで解説した。どちらの解決策でも、警告メッセージは出なくなる。

ポートベースのバーチャルホストでホスティングするなら、うまく動かすために、URLにポート番号を指定しておこう。例えば、ポート8443でSSLを使ったホストを動かしたいなら、次のようなURLでサイトにアクセスしなければならない。

```
https://www.example.com:8443/
```

これはエンドユーザにとっては不便かもしれないが、サイトへのアクセスがすべて、リンクやフォーム要素経由であれば、とてもよい解決策になるだろう。

レシピ7.8 ワイルドカード証明書を使う

課題

同じドメインにある複数のホスト名で1つの証明書を使いたい。

解決

ワイルドカード証明書を使えばよい。“*.example.com”のような特定のドメインにあるどんなホスト名でもうまく動く。

解説

レシピ7.3で説明したテクニックを使って、Common Nameを*.example.comとして証明書を作成する。ここで、example.com は、この証明書を使いたいドメインだ。この証明書は、www.example.com や secure.example.com のような、example.com ドメインのどんなホスト名でも動かせるようになる。

しかし、多くのブラウザでは、example.comやone.two.example.comといったホスト名ではうまく動かず、厳密に *hostname*.example.com という形式のホスト名でしか動かない。

たいていの CA は、ワイルドカード証明書に署名するのにかなり高い料金をとっている。これはワイルドカード証明書に署名するのが複雑だからではなく、単純にビジネス上の判断である。ワイルドカード証明書を 1 つ購入すれば、単一ホスト用の証明書をいくつも購入する必要がなくなってしまうためだ。

参照

- レシピ 7.3

8章
動的コンテンツ

動的コンテンツを提供する仕組みがないと、ほとんどのWebサイトは生き残れないだろう。動的コンテンツとは、ユーザの必要に応じてレスポンスを生成するコンテンツのことだ。本章のレシピでは、動的コンテンツを作成するいろいろな仕組みを紹介し、遭遇するかもしれない問題を解決する手助けをする。

CGIプログラムは、Webサイトで動的コンテンツを提供する最も簡単な方法の1つである。CGIはどんなプログラミング言語でも書くことができるので、ある意味、書きやすいと言えるだろう。つまり、CGIプログラムを書くために新しいプログラミング言語を覚える必要はないということだ。本章の例は、いろいろなプログラミング言語を使って書いてあるが、これを実行できるようApacheを設定するのには、必ずしもそのプログラミング言語を知っている必要はない。

CGIは今や、動的コンテンツを生成するのに好まれている仕組みではない。しかし、CGIは最も簡単な方法であるし、CGIがどうやって動いているのかを理解しておくことは、もっと複雑な動的コンテンツの作成方法を理解するのにとても役に立つ。

PHPやmod_perlのような動的コンテンツの作成方法は非常に人気がある。これは、CGIプログラムと同様の機能をたくさん提供しながらも、一般に実行速度が速いためだ。

レシピ 8.1　CGI ディレクトリを有効にする

課題

CGI スクリプトだけが入ったディレクトリを指定したい。

解決

httpd.conf ファイルに、次の行を追加する。

```
ScriptAlias /cgi-bin/ /www/cgi-bin/
```

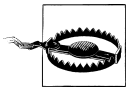
解説

CGIディレクトリは通常、Apacheをインストールしたときのデフォルト設定で指定されており、有効になっている。しかし、CGIプログラムを置くディレクトリを追加したいなら、ScriptAliasディレクティブを使え

ばよい。ScriptAlias ディレクティブは好きなだけ使うことができる。

先に紹介した1行は、次のディレクティブと同等だ。

```
Alias /cgi-bin/ /www/cgi-bin/
<Location /cgi-bin/>
    Options ExecCGI
    SetHandler cgi-script
</Location>
```



別のAliasやRewriteRuleのような他のメカニズムやURLパスを通して、URLをディレクトリに対応付けた場合には、ScriptAliasの設定が有効にならないことに注意すること。ScriptAliasの対応付けは、ディレクトリによるものではなく、URL(<Location>)によるものであるためだ。そのため、他のURLを通してこのディレクトリにあるスクリプトにアクセスしても、スクリプトが実行されるのではなく、そのコードが表示されてしまうだけだ。

対象とするディレクトリへのアクセスを許可するために、<Directory>ブロックを追加する必要があるかもしれない。cgi-binディレクトリは通常、ドキュメントディレクトリツリーの外部に置くためだ。cgi-binディレクトリでは .htaccess ファイルを使わないことを推奨する。

```
<Directory /www/cgi-bin>
    Order allow,deny
    Allow from all
    AllowOverride None
</Directory>
```

Windowsのファイル拡張子を使ってCGIプログラムを起動する方法については、レシピ8.4を参照してほしい。

参照

- 5章
- レシピ 8.2
- http://httpd.apache.org/docs/2.0/mod/mod_alias.html

レシピ 8.2 ScriptAlias 以外のディレクトリで CGI スクリプトを有効にする

課題

CGI 以外のドキュメントが入ったディレクトリに CGI プログラムを置きたい。

解決

AddHandler ディレクティブを使って、実行したいファイルに CGI ハンドラを対応付ける。

```
<Directory "/foo">
    Options +ExecCGI
    AddHandler cgi-script .cgi .py .pl
</Directory>
```

解説

多くの理由から、任意のドキュメントディレクトリでCGIの実行を許可するよりも、ScriptAliasディレクティブを通してCGIを実行できるようにする方が望ましい。主な理由は、セキュリティ監査だ。CGIプログラムがどこにあるのかわかっていると監査が簡単になるし、それが1つのディレクトリに格納されていれば、もっと簡単になる。

しかし、別の場所で CGI 機能を使いたい場合もある。例えば、静的ドキュメントとスクリプトによって1つのアプリケーションが構成されているときには、それらを1つのディレクトリにまとめておきたいだろう。

AddHandler ディレクティブを使って、特定のファイル拡張子を cgi-script ハンドラに対応付けると、そのファイルをCGIプログラムとして実行できるようになる。この例の場合、.cgi、.py、.plというファイル拡張子を持つプログラムはCGIプログラムとして扱われ、このディレクトリにあるその他のファイルはすべて通常の MIME タイプで扱われる。

Options ディレクティブには、ExecCGI ではなく +ExecCGI という引数を渡している（つまり + 符号を付けている）ことに注意しよう。+符号を付けると、すでに指定されているオプションに追加することになる。+符号を付けなければ、すでに指定されているオプションを置き換えることになる。このディレクトリにCGIプログラムだけを置くつもりなら、+符号を付けずに使うべきだ。同じディレクトリにCGI以外のドキュメントも置くつもりなら、+ 符号を付ける必要がある。

参照

- レシピ 8.1

レシピ8.3　CGIディレクトリにデフォルトドキュメントを指定する

課題

CGI ディレクトリがリクエストされたときに、デフォルトドキュメントを提供したい。

解決

ScriptAlias を使って CGI ディレクトリを有効にする代わりに、次のように設定する。

```
Alias /cgi-bin /www/cgi-bin
<Directory /www/cgi-bin>
    Options ExecCGI
```

```
    SetHandler cgi-script
    DirectoryIndex index.pl

    Order allow,deny
    Allow from all
    AllowOverride none
</Directory>
```

解説

ScriptAliasを使うと、ディレクトリのデフォルトドキュメントを提供してくれるDirectoryIndexが使えなくなる。ScriptAliasで指定されたディレクトリからデフォルトドキュメントを取得しようとすると、エラーログには次のようなエラーメッセージが記録されるだろう。

```
"attempt to invoke directory as script".
```

そして、ユーザのブラウザ上には、次のようなメッセージが表示されるだろう。

> 禁止されています。このサーバの /cgi-bin/ へのアクセス許可がありません。

デフォルトドキュメントが取得できるようにするには、ScriptAliasを使わずに、レシピ8.2で解説したもう1つの方法を使って、CGIが有効なディレクトリを作る必要がある。

ScriptAliasを使わずにCGIディレクトリを作成すると、DirectoryIndexディレクティブを使うことにより、ディレクトリがリクエストされたときにデフォルトドキュメントを表示することができる。

何か理由があって、このテクニックではなくScriptAliasを使いたいのであれば、別の方法もある。RedirectMatchディレクティブかRewriteRuleディレクティブのどちらかを使えば、CGIディレクトリ宛てのリクエストを好きなファイル名に対応付けることができる。

```
ScriptAlias /cgi-bin /www/cgi-bin
RedirectMatch "^/cgi-bin/?$" "http://server.example.com/cgi-bin/index.pl"
```

または、

```
ScriptAlias /cgi-bin /www/cgi-bin
RewriteEngine On
RewriteRule "^/cgi-bin/?$" "/cgi-bin/index.pl" [PT]
```

この2つの例は、メインサーバの設定ファイルに記述する必要がある。通常、ScriptAliasされたディレクトリの中では.htaccessファイルは有効にはならない。しかし、もしScriptAliasディレクトリの中で.htaccessファイルを有効にして、その中でRewriteRuleテクニックを使うのであれば、RewriteRuleが適用される前にリクエストされたURIからディレクトリパスが取り除かれてしまうことを覚えておこう。したがって、ルールセットは次のようなものになるだろう。

```
RewriteEngine On
RewriteBase /cgi-bin/
RewriteRule "^$" "index.pl" [R]
```

参照

- レシピ 8.2

レシピ 8.4　Windows のファイル拡張子を使って CGI プログラムを起動する

課題

Windows 上の CGI プログラムを、ファイル拡張子に関連付けられたプログラムによって実行したい。例えば、スクリプトの#!の行を変更して perl.exe の正しい場所を指定しなくても、.pl スクリプトが perl.exe によって実行されるようにしたい。

解決

httpd.conf ファイルに、次の行を追加すればよい。

```
ScriptInterpreterSource registry
```

解説

ApacheはUnixの世界からやってきたため、WindowsであってもUnixの流儀になっている部分が多い。CGI の実行もその 1 つだが、ScriptInterpreterSource ディレクティブを使うと、Windows ユーザが慣れ親しんだ方法で Apache を動かすことができる。

Windowsでは通常、ファイルタイプが拡張子によって示されている。例えば、example.plという名前のファイルは、Perl実行ファイルと関連付けられており、ユーザがExplorer上でファイルをクリックすると、このスクリプトを実行するために Perl が呼び出される。このような関連付けは、Perl や MS Word のような特定のプログラムをインストールするときに作られる。この関連付けは、Windowsのレジストリに格納されている。

これに対して、Unix系システムでは、ファイルの最初の行にインタプリタの場所を指定しているスクリプトが多い。この行は、#!という文字列で始まり、shebang行と呼ばれることが多い(sharp bangを短くしたもので、2 つの文字の省略名)。例えば、Perl プログラムは次のような行で始まる。

```
#!/usr/bin/perl
```

スクリプトを動かそうとするシェルは、この最初の行を見て、ここに指定されたパスにあるプログラムを使って、スクリプトを解釈、実行する。こうして、任意のファイル拡張子の付いた(あるいは、全く拡張子がなくてもよい)ファイルは、望みのインタプリタを呼び出すことができる。この例にあるPerlの場合、複数のバージョンのPerlがインストールされているかもしれない。ある特定のバージョンを呼び出したいならば、#!

行を適切に書けばよい。

しかし、Windowsのファイル拡張子に基づいたプログラム実行に慣れ親しんだ人には、Unix流のやり方は少し直感的でないかもしれない。そこで、Windows に Apache が導入された初期の段階で、Windows ユーザが期待するように動くよう、ScriptInterpreterSource ディレクティブが追加された。

ScriptInterpreterSource には、3つの値のどれか1つを設定することができる。デフォルト値である script に設定すると、Apache はスクリプト自体を調べて、使うべきインタプリタの場所を探す。

registryに設定すると、レジストリにあるファイル拡張子の関連付けを調べて、これを使ってスクリプトを実行する。

registry-strict に設定すると、レジストリのサブキー Shell\ExecCGI\Command だけが検索されることを除いて、registryと同じ効果になる。これは手作業によるレジストリ設定が必要なので、意図しないコマンドが実行されることを防ぐことができる。

この機能は、Unix 系 OS と Windows の両方で複数のサーバを運用しており、どちらでも同じ CGI プログラムを実行したいというユーザにとって、非常に役に立つ。例えば、Windowsマシンでは、Perlが/usr/bin/perlにあることはまずないだろう。ScriptInterpreterSourceディレクティブを使えば、Windows上のスクリプトを編集しなくても、ファイル拡張子を .pl にするだけで、そのまま実行することが可能だ。

参照

- レシピ 8.2
- レシピ 8.5
- http://httpd.apache.org/docs/2.2/mod/core.html#ScriptInterpreterSource

レシピ 8.5 CGI スクリプトの拡張子を指定する

課題

特定の拡張子のファイルをすべて、CGI スクリプトとして扱うように Apache を設定したい。

解決

httpd.conf ファイルにおいて、適用したい領域を含むスコープか適切なディレクトリの .htaccess ファイルに、次の行を追加する。

```
AddHandler cgi-script .cgi
```

解説

この解決策にある AddHandler ディレクティブは、.cgi 拡張子を持ったファイルを CGI スクリプトとして扱うよう指示している。これにより、ファイルはコンテンツとして送るのではなく、スクリプトとして実行される。

このディレクティブの効果があるのは、設定されているスコープにある、指定した拡張子のファイルだけだ。拡張子として .cgi を使うのが一般的だが、別のものに変えてもよいし、スペースで区切った複数の拡張

子を含むリストを与えてもよい。

サフィックスではなく拡張子という用語を使っていることに注意しよう。foo.cgi.enという名前のファイルは、拡張子 .en のハンドラが上書きしていない限り、CGI として扱われる。

これを実現する別の方法もある。次のようにすると、ファイルシステム上のどこにあっても、特定の拡張子を持ったファイルを CGI プログラムとして実行することができる。

```
<FilesMatch \.cgi(\.|$)>
    Options +ExecCGI
    SetHandler cgi-script
</FilesMatch>
```

FilesMatchディレクティブを使うと、指定したパターンにマッチしたファイルに対して、ディレクティブを適用することができる。この場合、.cgiというファイル拡張子を持つファイルが対象になる。これまで述べたように、ファイルはいくつかの拡張子を持つ可能性がある。パターン\.cgi$は.cgiで終わるファイル名にしかマッチしないので、ここでは、パターンとして \.cgi(\.|$)を使っている。(\.|$)という正規表現パターンを使うことによって、.cgi の後に、"." が続くか、文字列の終端のどちらでもマッチするようにしている。

参照

- レシピ 8.2

レシピ 8.6　CGI が正しくセットアップできたかテストする

課題

CGIが正しく有効になっているかテストしたい。あるいは、CGIスクリプトを動かそうとしてエラーメッセージを受け取った場合、スクリプトに問題があるか調べる前に、Web サーバの問題ではないことを確認しておきたい。

解決

次のような CGI スクリプトを動かしてみるとよい。

```
#!/usr/bin/perl
print "Content-type: text/plain\n\n";
print "It's working.\n";
```

うまく動かなければ、エラーログを調べる。

解説

Perl はおそらくどんな Unix 系システムにもインストールされているので、この CGI プログラムは CGI が正しく設定されているかテストするのに、かなり確実な方法だろう。Perl がインストールされていない場合には、次のようなシェルプログラムで代用できる。

図 8-1 CGI プログラムが正しく動いている

```
#!/bin/sh
echo Content-type: text/plain
echo
echo It\'s working.
```

Windows 上で Apache を動かしているなら、どちらも動かないかもしれない。そのときには、次のようなバッチファイルを試してみるとよい。

```
@echo off
echo Content-type: text/plain
echo.
echo It's working.
```

句読点やスラッシュなど、プログラムコードを正確にコピーしているか確認してほしい。そうしないと、プログラム自身のトラブルシューティングで、問題がさらにややこしくなってしまう。

いずれの場合も、プログラムがいったん動いてしまえば、図 8-1 のような画面が表示されるはずだ。

この背景にある考えは、問題が複雑なコードにあるのではないということを確認するために、できるだけ簡単な CGI プログラムで始めるということだ。CGI が適切に設定されているかを確認したいのであって、特定の CGI プログラムが正しいかどうかを確認したいわけではない。

特定の CGI プログラムが動かないのには、いろいろな原因が考えられる。大雑把に言うと、原因は次の 3 つのカテゴリのいずれかだ。3 つのカテゴリとは、Web サーバの設定ミス、プログラム自身のエラー、対象となるファイルやディレクトリのパーミッションの誤りである。

幸い、CGIプログラムで何かが悪いとき、その記録がエラーログに残る。エラーログがどこにあるかを知っておくのは、Apacheサーバで問題を解決するときの必須条件だ。ブラウザに返ってくるエラーメッセージもなんとなく役に立つように見えるが、雑多なメッセージになりがちで、実際の問題に特有の情報が入っていないことが多い。

理想的には、本章でこれまで解説したレシピに従っていれば、CGI プログラムの設定で問題は発生しないはずだ。したがって、原因としては、残りの 2 つのカテゴリが考えられる。

問題がパーミッションに関するものなら、ログファイルには次のようなエントリが記録されているはずだ。

```
[Sun Dec  1 20:31:16 2002] [error] (13)Permission denied: exec of /usr/local/apache/
    cgi-bin/example1.cgi failed
```

この問題の解決策は、スクリプト自体が実行可能になっているかどうか確認することだ。

```
# chmod a+x /usr/local/apache/cgi-bin/example1.cgi
```

問題がプログラム自体のエラーなら、プログラムが失敗する要因は無限にあるため、解決策も無限にある。ここで紹介したプログラム例が正しく動いているなら、問題は環境にあるのではなくプログラムにあると言ってよいだろう。

「Premature end of script headers（スクリプトヘッダが正しく終了していない）」というエラーメッセージをこれまで頻繁に見たことがあるだろうが、これ自体にはあまり意味がない。常に、このメッセージに付随した他のエラーメッセージを調べなければならない。CGIプログラム中でエラーが発生すると、HTTPヘッダを正しく作成する前に警告やエラーメッセージを出してしまう。サーバはこれを不正な形式のヘッダだと判断して、このメッセージを出してしまうのだ。suexecラッパーを使用したときにも、このような混乱が生じることがある。

次に示したエラーメッセージは特によく見かけるものであり、探しているものが何かわからないなら、この原因を突き止めるのはかなり難しいだろう。

```
[Sat Jul 19 21:39:47 2003] [error] (2)No such file or directory:
    exec of /usr/local/apache/cgi-bin/example.cgi failed
```

このエラーメッセージの原因はほとんどの場合、パスが間違っているかファイルが壊れているかのどちらかだ。多くの場合、特に他人からスクリプトを入手したとき、スクリプトの#!の行が間違った場所を指していることがある（例えば、perlが/usr/bin/perlにあるのに、#!/usr/local/bin/perlとなっているなど）。確認するには、whichコマンドの出力を#!の行と比較すればよい。例えば、Perlの正しい場所を見つけるには、次のようにコマンドを実行する。

```
% which perl
```

もう1つの原因は、ファイルがどこか壊れていて、#!の行を判読できないというものだ。このような状態を引き起こす理由のうち最もよくあるものは、スクリプトファイルをWindowsマシンからUnix系マシンにFTPで転送するときに、ASCIIモードでなくバイナリモードを使ってしまった場合だ。こうすると、ファイルの行末文字が間違った種類のファイルになるため、Apacheはスクリプトインタプリタの場所を正しく読むことができなくなる。

これを修正するには、コマンドラインから次のような1行を実行すればよい。

```
% perl -pi.bak -le 's/\r$//;' example.cgi
```

こうすると、Windowsスタイルの行末文字をすべて削除して、ファイルを実行可能にしてくれるだろう。念

のため、ファイル拡張子を.bakとしたファイルのバックアップコピーを作っておくとよいだろう。何らかの理由で、この変換が問題を引き起こすかもしれないためだ。

参照

- 付録 B

レシピ 8.7 フォームのパラメータを読み出す

課題

CGI プログラムで使うために、Web フォームから入力された値をプログラム中から読み出したい。

解決

最初に、Perl で書かれた例を見てみよう。ここでは人気のある CGI.pm モジュールを使っている。

```
#!/usr/bin/perl
use CGI;
use strict;
use warnings;

my $form = CGI->new;

# パラメータからいろいろロードする
my $name = $form->param('name') || '-';

# 複数選択可能なリストでは、リストが返る
my @foods = $form->param('favorite_foods');

# 役に立つ情報を出力
print "Content-type: text/html\n\n";
print 'Name: ' . $name . "<br />\n";
print "Favorite foods: <ul>\n";
foreach my $food (@foods) {
    print " <li>$food</li>\n";
}
print "</ul>\n";
```

次に、Cで書かれたプログラムを見てみよう。これは前の例とほとんど同じことをやっていて、cgicというC ライブラリを使っている。

```
#include "cgic.h"
/* Boutell.com の cgic ライブラリ */
```

```
int cgiMain() {
    char name[100];

    /* コンテンツタイプの送信 */
    cgiHeaderContentType("text/html");

    /* 特定の変数をロード */
    cgiFormStringNoNewlines("name", name, 100);
    fprintf(cgiOut, "Name: ");
    cgiHtmlEscape(name);
    fprintf(cgiOut, "\n");

    return 0;
}
```

この例では、Makefileも必要になる。次のようなものになるだろう。

```
CFLAGS=-g -Wall
CC=gcc
AR=ar
LIBS=-L./ -lcgic

libcgic.a: cgic.o cgic.h
 TAB rm -f libcgic.a
 TAB $(AR) rc libcgic.a cgic.o

example.cgi: example.o libcgic.a
 TAB gcc example.o -o example.cgi $(LIBS)
```

解説

この課題に対する解決策は、プログラミング言語によって異なっている。ここでは2つのプログラミング言語で書いた例を示した。これらの例では、フォームの内容を実際に解析するのに外部ライブラリを使っていることに注意しよう。これは間違ったフォームの解析を容易にするために重要なことだ。このようなライブラリを使うと、フォームエンコードされた文字をすべて正しく使用可能な値に変換することができるし、コードも読みやすく単純にすることができる。自分でこうした機能を再実装するよりも、既存のライブラリを使う方がよい場合が多い。

Perlで書いた例では、Lincoln SteinのCGI.pmを使っている。これはPerlディストリビューションに標準で含まれており、Perlをインストールすると一緒にインストールされる。ライブラリはuseキーワードを使ってロードされ、オブジェクト指向(OO)インターフェイスを通して利用する。

paramメソッドはフォームフィールドの値を返す。引数なしで呼び出すと、params()はフォームフィールド名のリストを返す。複数選択可能なフォームフィールドの名前を指定して呼び出すと、選択された値のリス

トを返す。この例では、favorite_foods という名前のフィールドで使われている。

Cで書いた例では、cgic というCライブラリを使っている。このライブラリは、http://boutell.com から入手することができる。上記のコードをコンパイルするためには、このライブラリを入手してインストールしておく必要がある。ここで紹介したMakefileを使うと、ソースコードをビルドして、実行可能なバイナリファイルを作るのを助けてくれる。コンパイルするには、make example.cgiを実行すればよい。Windows上で動かしたいなら、おそらくこの例の Makefile の .cgi を .exe に置き換える必要があるだろう。

どちらの場合も、この CGI プログラムを指す HTML フォームには name という名前のフォームフィールドがあり、そのフィールドに入力した値がブラウザ上に表示される。これらのプログラムをテストするのに必要な HTML は、次のようなものになる。

```
<html>
 <head>
   <title>Example CGI</title>
 </head>
 <body>

   <h3>Form:</h3>

   <form action="/cgi-bin/example.cgi" method="post">
   Name: <input name="name">
   <br />
   <input type="submit">
   </form>

 </body>
</html>
```

このレシピで紹介した例では、実際にHTMLフォームの内容を解析するために、CGIライブラリやCGIモジュールを使っている。Webにはたくさんの CGIチュートリアルがあり、自分で解析する方法を説明しているものもあるが、ここでは推奨しない。プログラマにとって偉大な美徳の1つは怠慢である。車輪を再発明するよりもモジュールを利用することは、最も重要な怠惰の表れの1つだ。また、こうしたモジュールは正しく動く可能性が高いので、これを利用するのは理にかなっている。フォームの内容を間違って解析してしまうのは簡単で、不完全なフォームエンコーディングに変換したり、明らかに間違ったデータになってしまうおそれがある。これらのCGIモジュールは、長年にわたって開発され、広くテストされており、考えもつかないようないろいろなケースを正しく処理することができる。

さらにこうしたモジュールは、ファイルのアップロードや複数選択のリスト、Cookieの読み出しと設定、正しくフォーマットしたエラーメッセージをブラウザへ返すことなど、自分で作ろうとすると見落としてしまうような各種機能を実現してくれる。さらに、よいプログラミングテクニックの精神からすると、既存のコードを再利用することは、時間の節約とエラーの防止につながる。

参照

- http://search.cpan.org/author/LDS/CGI.pm/CGI.pm
- http://www.boutell.com/cgic

レシピ8.8　特定のコンテンツタイプ用のCGIプログラムを呼び出す

課題

特定のドキュメントタイプに対して、一種のコンテンツフィルタとして動作する CGI プログラムを呼び出したい。例えば、写真家であれば、Web サイトで提供する写真に透かし（ウォーターマーク）を追加するためのカスタムハンドラを作成したいことがあるだろう。

解決

Action ディレクティブを使って、カスタムハンドラを作る。カスタムハンドラは CGI プログラムで実装することができる。そして、AddHandler ディレクティブを使って、特定のファイル拡張子をこのハンドラに関連付ける。

```
Action watermark /cgi-bin/watermark.cgi
AddHandler watermark .gif .jpg
```

あるいは、ファイル名ではなくデータの種類に基づいてサーバが適切なハンドラを選択するようにしたいのであれば、次のようにすればよい。

```
Action image/gif /cgi-bin/watermark.cgi
Action image/jpeg /cgi-bin/watermark.cgi
```

解説

このレシピでは、.gif や .jpg ファイルがリクエストされたときに呼び出される透かしハンドラを作成している。

CGI プログラム watermark.cgi は、入力として画像ファイルを受け取り、写真表面に透かしを貼り付ける。元の URL でリクエストされた画像ファイルのパスは、PATH_TRANSLATED 環境変数として利用することができる。プログラムは、そのファイルをロードして、必要な加工を施し、その結果得られたコンテンツを、適切な HTTP ヘッダを付けてクライアントに送る。

CGI プログラムは、このディレクティブの適用スコープ内でリクエストされたすべての .gif や .jpg ファイルに対して呼び出され、これを回避する方法はないことに注意しよう。

同じテクニックを使うと、ファイルに SSI ディレクティブのようなものを追加しなくても、HTML ページにヘッダやフッタを自動的に追加することもできる。この方法は、CGI プログラムを起動する必要があるため、かなり処理が遅く非効率だ。しかし、この方法を知っておくことは将来役に立つかもしれない。以下は、フッタを挿入する非常に単純なスクリプトの実装例だ。

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";

my $file = $ENV{PATH_TRANSLATED};

open FILE, "$file";
print while <FILE>;
close FILE;
print qq~

<p>
FOOTER GOES HERE
</p>
~;
```

同じことをPHPスクリプトで書くと、次のようになる。

```
#!/usr/bin/php
$fh = fopen($_SERVER['PATH_TRANSLATED'], 'r');
fpassthru($fh);
print "\n\n<p>\n"
    . "FOOTER GOES HERE\n"
    . "</p>\n";
return;
```

まずリクエストされたファイル(環境変数PATH_TRANSLATEDから取得できる)を読み込み、修正せずにそのまま出力している。そして、最後にフッタを数行追加して出力している。同じようなテクニックは、ページ自体のコンテンツをフィルタリングするのにも使えるだろう。Apache 2.0では、mod_ext_filterを使うと、もっとうまくやることができる。

このスクリプトはテクニックを解説することを目的としたものであり、実際にWebページにフッタを付けるのに使うべきではない。実際にこうしたタスクをするのに必要なチェック(「これはHTMLファイルなのか?」「コンテンツの後にHTMLを追加して安全なのか?」など)は、何もしていないためだ。

参照

- レシピ 8.11
- レシピ 10.7

レシピ 8.9　SSI を使用可能にする

課題

HTML ドキュメントをもっと動的にするために、SSI（Server-Side Include）を有効にしたい。

解決

少なくとも 2 つの方法がある。

.shtml のようなファイル拡張子を使って、どのファイルを SSI で解釈すべきか指定する。Apache 1.3 では、httpd.conf の適切なスコープに、次のディレクティブを追加すればよい。

```
<Directory /www/html/example>
    Options +Includes
    AddHandler server-parsed .shtml
    AddType "text/html; charset=ISO-8859-1" .shtml
</Directory>
```

Apache 2.0 以降では、次のように設定すればよい。

```
<Directory /www/html/example>
    Options +Includes
    AddType text/html .shtml
    AddOutputFilter INCLUDES .shtml
</Directory>
```

もう 1 つの方法は、httpd.conf ファイルの適切なスコープに XBitHack ディレクティブを追加して、SSI で解析すべきかどうかをファイルのパーミッションで示す。

```
XBitHack On
```

解説

SSI ディレクティブを使うと、さまざまな単純なタグで HTML ページに動的コンテンツを追加することができる。この機能は、mod_include モジュールで実装されており、そのドキュメントは、http://httpd.apache.org/docs/mod/mod_include.html にある。入門者向けの解説も、http://httpd.apache.org/docs/howto/ssi.html にある。

ここで紹介した最初の解決策では、Apache にすべての .shtml ファイルを SSI で解析するよう指示している。この解決策がうまく動いているかどうか確認するには、something.shtml という名前のファイルを作成して、次のような行を書いておけばよい。

```
File last modified at '<!--#echo var="LAST_MODIFIED" -->'
```

最後の引数("LAST_MODIFIED")と"-->"の間に空白があることに注意すること。この空白は意外にも重要だ。SSIが失敗する原因の多くは、この空白を入れるのを忘れたためだ。

サーバを通してこのドキュメントにアクセスすると、ページにはそのファイルが更新された(または作成された)日付と時刻が表示されるはずだ。

SSIを有効にしたいが、CGIスクリプトやSSIディレクティブ#exec virtualや#include virtualを使った他のコマンドの実行を許可したくない場合、この例のOptionsディレクティブに指定されているIncludesをIncludesNoExecに置き換えればよい。

Webマスターによっては、AddTypeやAddHandler、AddOutputFilterディレクティブで、.shtmlの代わりに.htmlを指定することによって、サイトにあるすべてのHTMLコンテンツをSSLで解析することを好む人もいるだろう。

ファイルに動的コンテンツを加えたいという理由だけで、ドキュメントのファイル名を.shtmlに変えたくないなら、XBitHackディレクティブを使えばうまくいく。もちろん、すべての.htmlファイルをSSIで解析することもできるが、意味もなくすべてのファイルを解析することになるので、パフォーマンスに影響を与えてしまうだろう。

XBitHackディレクティブを使うと、実行ビットが設定されているファイルだけをSSIで解析することができる。特定のディレクトリやバーチャルホストでこのXBitHackディレクティブをOnに設定しておけば、SSIディレクティブを含むファイルに実行ビットを設定するだけでよい。この方法だと、既存のドキュメントのファイル名を変えずに、SSIディレクティブを追加することができる。ファイル名を変えることによって、他のページやサイト、サーチエンジンからのリンクを壊してしまうこともない。

ファイルに実行ビットを設定する(または解除する)最も簡単な方法は、次のコマンドを実行することだ。

```
# chmod a+x foo.html   # 設定
# chmod a-x foo.html   # 解除
```

XBitHackは、ファイルへの実行アクセスという概念があるプラットフォームでしか動かない。つまり、Unix系システムでは動くが、Windowsでは動かない。

参照

- レシピ8.12
- レシピ8.11

レシピ8.10 最終更新日時を表示する

課題

Webページが最後に更新された時間を表示したいが、毎回自分で日付を変えたくない。

解決

SSI を利用して、情報を表示したい HTML ファイルに 1 行追加すればよい。

```
<!--#config timefmt="%B %e, %Y" -->
このドキュメントの最終更新日 <!--#echo var="LAST_MODIFIED" -->
```

解説

#config SSI ディレクティブを使うと、SSI の出力フォーマットを設定することができる。この場合、表示される日付と時刻のフォーマットを設定している。デフォルトの日付出力フォーマットは04-Dec-2037 19:58:15 ESTという形式で、ユーザにとってわかりやすい書式ではない。このレシピではこのフォーマットを少し読みやすく、December 4, 2002 という形式に変更した。別の出力フォーマットがよければ、timefmt 属性で指定することができる。この属性の引数には、C の strftime(3)関数と同じものを使うことができる。

参照

- レシピ 8.9
- strftime(3)のドキュメント

レシピ 8.11　標準ヘッダを挿入する

課題

それぞれの HTML ドキュメントに、ヘッダやフッタを挿入したい。

解決

SSI で解析するすべてのファイルに、次の 1 行を挿入すればよい。

```
<!--#include virtual="/include/headers.html" -->
```

解説

SSI の #include ディレクティブを使うと、Web サイト全体で 1 つのヘッダファイルを使うことができる。ヘッダファイルを修正しなければならないときも、1つのファイルだけ変更すればよく、サイト全体に即座に変更を反映させることができる。

virtual 属性への引数は、ローカル URI であり、通常の Alias、ScriptAlias、RewriteRule など、その他のコマンドの影響を受ける。例えば、次のように指定すると、DocumentRoot にあるファイルが挿入される。

```
<!--#include virtual="/index.html" -->
```

また次のように指定すると、サーバのScriptAliasディレクトリにあるfooスクリプトの出力が挿入される。

```
<!--#include virtual="/cgi-bin/foo" -->
```

引数が/文字で始まっていない場合は、#includeディレクティブを使っているドキュメントの場所からの相対パスとして扱われる。

#include virtualに渡されるURIは、../で始まっていないし、http://example.com/foo.htmlのような完全なURLを参照してもいないことに注意しよう。相対パス(つまり、/で始まらないパス)を使って挿入されたドキュメントは、挿入しているファイルと同じ場所か、その下位になければならない。URIをサーバが処理すると、他の場所にあるドキュメントを挿入することになるかもしれないが、#include virtualによるSSIコマンド構文は、同じ場所か下位の場所のURIしか許されないように制限されている。

参照

- レシピ 8.8
- レシピ 8.9

レシピ 8.12 CGI プログラムの出力を挿入する

課題

CGI プログラムの出力を HTML ドキュメント中に挿入したい。

解決

SSI を使って、ドキュメントに次のような行を追加すればよい(そのドキュメントに対して SSI の解析を有効にしておく必要がある)。

```
<!--#include virtual="/cgi-bin/content.cgi" -->
```

解説

SSIの#includeディレクティブを使うと、普通のテキストファイルが挿入できるだけでなく、CGIプログラムや他のSSIドキュメント、その他の方法で生成されたコンテンツなどの動的コンテンツも挿入することができる。

SSIの#execディレクティブを使っても同じ効果が得られるが、歴史的なセキュリティ関連の理由から、この使用は将来廃止される予定だ。この効果を得るには、#include ディレクティブを使うのが望ましい。

参照

- レシピ 8.9

レシピ 8.13　suexec を使って別のユーザとして CGI スクリプトを実行する

課題

CGI プログラムを nobody（または、Apache サーバを動かしているユーザ）以外のユーザで実行したい。例えば、特定のユーザ以外はアクセスできないデータベースがあり、サーバがそのデータベースにアクセスするためには一時的にそのユーザ ID を使わなければならないことがある。

解決

Apache をビルドするときに、configure に --enable-suexec 引数を付けてビルドし、suexec を有効にする。

そして、バーチャルホストセクションに、CGI プログラムの実行に使用するユーザとグループを指定すればよい。

```
User rbowen
Group users
```

あるいは、対象とするバーチャルホストのユーザ名を含む形式の URL で CGI プログラムを実行すると、suexec を呼び出すことができる。

解説

suexec ラッパーは suid プログラム（ファイルの所有者のユーザ ID で動く）であり、Apache を動かしている nobody ユーザではなく、指定したユーザで CGI プログラムを動かすことができる。suexec は Apache の標準ディストリビューションの一部であるが、デフォルトでは有効になっていない。

suexec の概念は Windows 環境にはうまく合わないため、Windows では利用できない。

suexec がインストールされると、この解決策で示したように、2 つの呼び出し方がある。

1 つは、VirtualHost コンテナに User と Group ディレクティブを指定する方法だ。こうすると、バーチャルホストのコンテキスト内で実行されるすべての CGI プログラムが、そのユーザとグループで実行される。これは、CGI プログラムにだけ適用されることに注意しよう。通常のドキュメントやその他の動的コンテンツは、バーチャルホストではなく、メインサーバの設定にある User と Group ディレクティブで指定したユーザとグループでアクセスされる。そのため、通常のドキュメントやその他の動的コンテンツは、メインサーバで設定されたユーザとグループに読み出し可能にしておく必要がある。

もう 1 つは、ユーザごとの Web ディレクトリを使う方法だ。UserDir ディレクトリから実行した CGI プログラムは、そのディレクトリの所有者の権限で動く。つまり、CGI プログラムが http://example.com/~rbowen/cgi-bin/test.cgi からアクセスされると、suexec は、*rbrowen* のユーザ ID と *rbowen* のメイングループ ID の権限でプログラムを実行する。

UserDirが標準以外の場所を指しているなら、ビルド時にこれをsuexecに教えなければならない。デフォルトの設定では、CGIプログラムが/home/*username*/public_html/のようなディレクトリの中で呼び出されると、suexecが呼び出されるようになっている。しかし、UserDirディレクトリを他の場所、例えば/home/*username*/www/などに移動した場合には、suexecがそのディレクトリで呼び出されるように設定する必要がある。Apache 1.3の場合には、ビルド時に次の引数で指定すればよい。

```
--suexec-userdir=www
```

Apache 2.0の場合には、次のように指定すればよい。

```
--with-suexec-userdir=www
```

suexecを通してCGIプログラムを動かすと、CGIプログラムにまつわるセキュリティ上の懸念をいくつか取り除くことができる。デフォルトでは、CGIプログラムはUserとGroupディレクティブで指定したユーザとグループの権限で動くので、損害を与える可能性がかなり制限される。しかし、Webサーバ上のすべてのCGIプログラムはそのサーバと同じ権限で動くので、あるプログラムが作成したり修正したりしたファイルを、他のプログラムが変更するおそれがある。

suexecの下でCGIプログラムを動かすと、各ユーザが自分のファイルパーミッションをもう少しコントロールすることができる。誰かが悪意のあるCGIプログラムを書いても、Webサーバ全体を無制限にコントロールできるわけではなく、そのユーザが所有しているファイルだけが損害を受けることになる。

mod_phpハンドラではなく、CGIプログラムとしてPHPスクリプトを動かすと、他のCGIプログラムと同様にsuexecを通して実行することができる。

suexecで何か問題が発生すると、suexecはできるだけ神経質な対応をして、ドキュメントを何も提供しない。エンドユーザにはエラーページを表示するが、実際に何が悪いのかを知るためには、サーバのエラーログを見る必要がある。メッセージは一目瞭然だ。suexecの問題はほとんどすべて、ファイルのパーミッションや所有権が間違っているのが原因である。suexecのログエントリは、それが何なのかをはっきりさせてくれるはずだ。

参照

- Userディレクティブ
  http://httpd.apache.org/docs/mod/core.html#user
  http://httpd.apache.org/docs-2.0/mod/mpm_common.html#user
- Groupディレクティブ
  http://httpd.apache.org/docs/mod/core.html#group
  http://httpd.apache.org/docs-2.0/mod/mpm_common.html#group
- suexecドキュメント
  http://httpd.apache.org/docs/programs/suexec.html

http://httpd.apache.org/docs-2.0/programs/suexec.html

レシピ8.14　CPANからmod_perlハンドラをインストールする

課題

CPANで利用できるたくさんのmod_perlハンドラモジュールのうち、いずれかのモジュールをインストールしたい。例えば、Apache::Perldocモジュールをインストールしたい（このモジュールはインストール済みのPerlモジュールのHTMLドキュメントを生成してくれる）。

解決

すでにmod_perlがインストールされていれば、CPANからモジュールをインストールして、Apacheの設定ファイルに数行追加すればよいだけだ。

モジュールをインストールするには、rootユーザでシェルから次のようなコマンドを実行すればよい。

```
# perl -MCPAN -e 'install Apache::Perldoc'
```

そして、Apacheの設定ファイルに以下を追加する。

```
<Location /perldoc>
    SetHandler perl-script
    PerlHandler Apache::Perldoc
</Location>
```

Apacheを再起動した後、http://example.com/perldoc/Apache/Perldocのようなurlで、このハンドラにアクセスすることができる。

解説

CPANシェルは、Perlをインストールすると一緒にインストールされる。これを使うと、CPANからPerlモジュールを簡単にインストールすることができる。CPANについてよく知らない人のために説明すると、CPANとはComprehensive Perl Archive Networkの略であり、http://cpan.orgにWebサイトがある。これは、Perl関係の総合的なアーカイブであり、想像できるものから想像もつかないようなものまで、あらゆる用途のPerlモジュールを提供している。ここには、かなりの数のmod_perlハンドラも含まれている。

このレシピで紹介したApache::PerlDocモジュールはとても単純なもので、インストール済みのPerlモジュールに関するHTMLドキュメントを、Apacheサーバを通してアクセスできるようにしてくれる。この他にも、フォトアルバムやブログのハンドラ、DNSゾーン管理など、いろいろなモジュールがある。

CPANシェルを初めて動かすときには、設定に関する一連の質問に答える必要がある。モジュールを入手するのに使いたいCPANサーバはどれか、FTPクライアントはどこにあるかなどに答える必要がある。この質問は初回だけで、それ以後はすぐに起動するようになる。

新しくインストールしたモジュールを使うために必要なApacheの設定は、モジュールごとに異なっている

が、多くはこの例にあるように設定すればよい。SetHandler perl-script ディレクティブは、コンテンツを mod_perl で処理するよう Apache に指示しており、PerlHandler ディレクティブはどの Perl モジュールに実際のハンドラコードが含まれているかを指定している。

参照

- CPAN―― http://cpan.org
- Apache::PerlDoc―― http://search.cpan.org/author/RBOW/Apache-Perldoc
- Apache::Gallery―― http://apachegallery.dk

レシピ 8.15 mod_perl ハンドラを書く

課題

独自の mod_perl ハンドラを書きたい。

解決

以下に、簡単なハンドラの例を示す。

```
package Apache::Cookbook::Example;

sub handler {
    my $r = shift;
    $r->send_http_header('text/plain');
    $r->print('Hello, World.');
}

1;
```

このコードを Example.pm という名前のファイルに書き込み、Apache/Cookbook/ ディレクトリのような、Perl が検索するディレクトリに置く。

解説

ここで紹介したハンドラの例は、全くつまらなくて、役に立たないものだ。もっと役に立つ例は、mod_perl のWebサイト（http://perl.apache.org）やGeoffrey Youngたちの優れた書籍『mod_perl Developer's Cookbook』（Sams発行、和書未刊）から得ることができる。また、少し古くなってしまったが、『Apache拡張ガイド（上）サーバサイドプログラミング』と『Apache 拡張ガイド（下）API リファレンス』（オライリー・ジャパン発行、原書『Writing Apache Modules with Perl and C』、Doug MacEachern、Lincoln Stein 著、O'Reilly Media 発行）は、mod_perl と Apache API に関する優れた入門書だ。

しかしここで本当に課題になるのは、作成したファイルをどこにどうやってインストールするのかということだ。この質問には 2 つの答えがある。どちらを選ぶかは、個人的な好みによるだろう。

Perlがモジュールを探すときには、@INCという名前のリストを調べる。このリストには、モジュールが置かれているディレクトリが入っている。したがって、モジュールをこれらのディレクトリのどれか1つに置くか、モジュールが置いてあるディレクトリをこのリストに追加すればよい。

Perlがモジュールを探す場所を知るには、次のようにして、@INCに格納されている値を調べればよい。

```
perl -le 'print join "\n", @INC;'
```

これは、次のようなリストを出力するだろう。

```
/usr/local/lib/perl5/5.8.0/i686-linux
/usr/local/lib/perl5/5.8.0
/usr/local/lib/perl5/site_perl/5.8.0/i686-linux
/usr/local/lib/perl5/site_perl/5.8.0
/usr/local/lib/perl5/site_perl
.
```

もちろん、出力結果はシステムやPerlのバージョンによって違ってくるが、結果は似たようなものになるだろう。

この場合、Apache::Cookbook::Exampleという名前のモジュールをインストールするには、ファイルExample.pmを/usr/local/lib/perl5/site_perl/5.8.0/Apache/Cookbook/Example.pmという場所に置けばよい。

もう1つの方法として、@INCリストに別のディレクトリを追加して、Perlがそのディレクトリを探しにいくようにしてもよい。startup.plファイルに、次の行を追加する。

```
use lib '/home/rbowen/perl_libs/';
```

そして、startup.plがApacheの起動時にロードされるように、Apacheサーバの設定ファイルに次のディレクティブを追加する。

```
PerlRequire /path/to/startup.pl
```

こうすると、Perlがモジュールを探すときに、このディレクトリも調べてくれる。Apache::Cookbook::Exampleという名前のモジュールは、/home/rbowen/perl_libs/Apache/Cookbook/Example.pmという場所に置くことができる。

参照

- 『mod_perl Developer's Cookbook』、Geoffrey Young、Paul Lindner、Randy Kobes著、Sams発行、和書未刊（http://modperlcookbook.org）

レシピ 8.16 PHP スクリプト処理を使用可能にする

課題

サーバで PHP スクリプトが使えるようにしたい。

解決

mod_php がインストールされていれば、AddHandler を使って、.php と .phtml ファイルを PHP ハンドラに対応付ければよい。設定ファイルに、次の行を追加する。

```
AddHandler application/x-httpd-php .phtml .php
```

解説

このレシピでは、.phtml と .php の付いたファイルをすべて PHP ハンドラに対応付けている。mod_php がインストールされていて、サーバにロードされていることを確認しておく必要がある。

モジュールを有効にするのに、AddHandlerを使うか、AddTypeを使うかに関して意見の相違があるかもしれないが、AddHandlerディレクティブを使うのが正しい。

参照

- レシピ 2.5
- PHP Web サイトにあるインストール手順—— http://php.net/manual/en/install.php

レシピ8.17 PHPが正しくインストールされているか確認する

課題

PHP が正しくインストールされていて、正しく設定されているか確認したい。

解決

テスト用の PHP ファイルに、次の行を書く。

```
<?php phpinfo(); ?>
```

解説

PHPスクリプトが実行できるはずのディレクトリに、something.phpという名前のファイルを作り、この1行を書き込む。このファイルにアクセスすると、設定されているPHPシステム変数の一覧が表示されるはずだ。出力の最初の部分は、図 8-2 のような画面になるはずだ。

参照

- レシピ 8.16

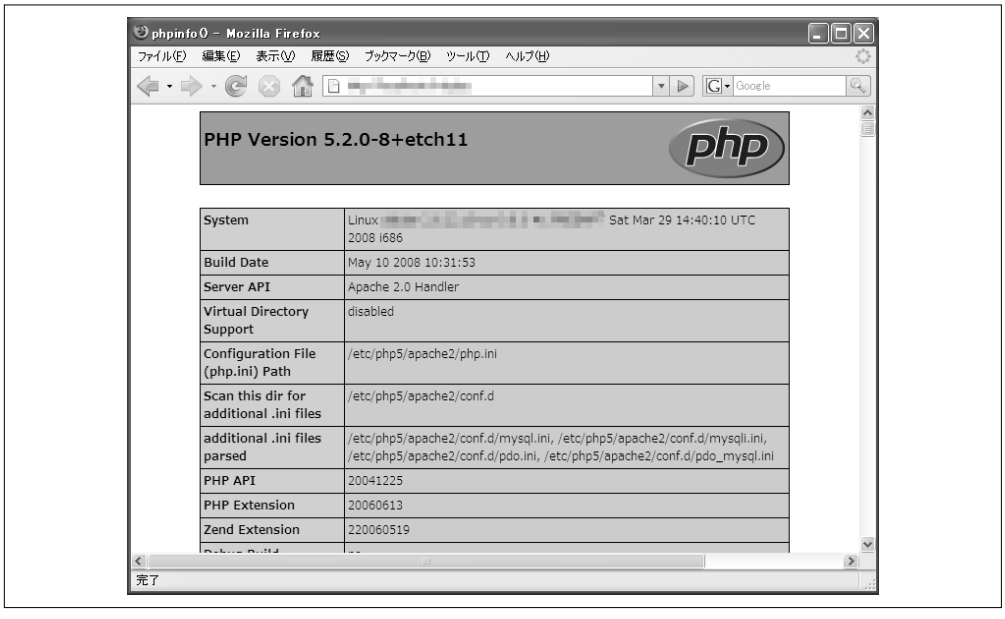


図 8-2　phpinfo()の出力例

レシピ 8.18　CGI 出力を SSI で解析する

課題

CGI スクリプトの出力に SSI ディレクティブを入れて、正しく SSI が処理されるようにしたい。

解決

これは、Apache 2.0 以降でのみ完全にサポートされている。

SSI で解析したい CGI スクリプトを含むスコープに、次の設定を追加すればよい。*.cgi* サフィックスはスクリプトが実際に使っているものに変更すること。

```
Options +Includes
AddOutputFilter INCLUDES .cgi
```

解説

この2行をサーバ全体の設定ファイルか、スクリプトと同じディレクトリにある .htaccess ファイルに書いておく。こうすると、サーバはスクリプトの出力を受け取り、クライアントに送る前にSSIディレクティブを解析してくれる。

参照

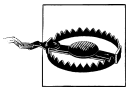
- レシピ 8.19

レシピ8.19 ScriptAliasにあるスクリプトの出力をSSIで解析する

課題

ScriptAliasディレクトリにあるスクリプトの出力にSSIディレクティブを入れて、正しくSSIが処理されるようにしたい。

解決



これは、Apache 2.0以降でのみ完全にサポートされている。

ScriptAliasディレクトリに関する<Directory>コンテナに、次の設定を追加すればよい。

```
Options +Includes
SetOutputFilter INCLUDES
```

解説

このディレクティブを使うと、サーバはScriptAliasディレクトリにあるスクリプトの出力を受け取り、クライアントに送る前にSSIディレクティブを解析してくれる。

参照

- レシピ 8.18

レシピ8.20 すべてのPerlスクリプトをmod_perlで処理する

課題

すべての*.pl*ファイルを、常にmod_perlで実行するようにしたい。

解決

httpd.confの先頭近くにあるモジュール宣言や有効化セクションの後に、次の行を追加する。

```
PerlModule Apache::Registry
```

この動作をさせたいスコープを含むhttpd.confファイルのセクション（例えば、<Directory>コンテナの中）に、以下のコードを追加する。

```
<FilesMatch \.pl$>
    SetHandler perl-script
    PerlHandler Apache::Registry
</FilesMatch>
```

mod_perl モジュールがインストールされていて、有効になっていることを確認しておくこと。

解説

PerlModule ディレクティブは、mod_perl で必要なものを確実に利用できるようにする。<FilesMatch> は *.pl* で終わるすべてのファイルに適用され、そのファイルを Apache::Registry パッケージによって CGI スクリプトとして処理するよう、サーバに指示する。

もっと詳しい情報については、mod_perl の Web サイト（http://perl.apache.org）を参照すること。これらのディレクティブは、それが実際に CGI スクリプトであってもなくても、すべての *.pl* ファイルを CGI スクリプトとして扱う。サーバが CGI スクリプトでないファイルを実行しようとすると、エンドユーザにはエラーページが表示され、サーバのエラーログに記録されることになる。最もよく記録されるエラーは、「Premature end of script headers（スクリプトヘッダが正しく終了していない）」だ。これは CGI スクリプトが壊れているか、CGI スクリプトではないファイルを実行しようとしたかのどちらかだろう。

参照

- mod_perl の Web サイト―― http://perl.apache.org
- 『mod_perl Developer's Cookbook』、Geoffrey Young、Paul Lindner、Randy Kobes 著、Sams 発行、和書未刊（http://modperlcookbook.org）

レシピ 8.21　Python スクリプト処理を使用可能にする

課題

サーバで Python スクリプトが使えるようにしたい。

解決

mod_python がインストールされていれば、次のディレクティブによって Python スクリプトがアクセスされたときに、mod_python を呼び出すことができる。

```
AddHandler mod_python .py
PythonHandler mod_python.publisher
PythonDebug On
```

解説

このレシピは、.py の付いたすべてのファイルを Python スクリプトハンドラに対応付けている。リクエストがこのディレクティブのスコープにある .py サフィックスの付いたファイルだと判断すると、サーバはその

ファイルをPythonスクリプトとして実行する。mod_pythonモジュールがインストールされていることを確認しておくこと。

参照

- レシピ8.16
- mod_pythonのWebサイトにあるインストール手順（http://modpython.org/）

9章
エラー処理

Webサイトを動かしていると、何らかの問題が発生することがある。そのときにはうまく処理して、ユーザの印象がそれほど悪くならないようにすることが重要だ。本章では、エラー状態をどうやって処理するのか、ユーザにとって有益なメッセージの返し方、問題を修正して再発を防止するのに役立つ情報をどうやって収集するかといったことを解説する。

レシピ 9.1　Host フィールドがないリクエストを処理する

課題

複数のバーチャルホストを使用しており、少なくともその1つはネームベースのバーチャルホストだとする。ネームベースのバーチャルホストを正しく動かすには、クライアントがリクエストヘッダに有効なHostフィールドを送る必要があるが、Hostフィールドが含まれていない場合にも、うまく処理したい。

解決

httpd.conf ファイルに、次の行を追加する。

```
Alias /NoHost.cgi /usr/local/apache/cgi-bin/NoHost.cgi
RewriteEngine On
RewriteCond "%{HTTP_HOST}" "^$"
RewriteRule "(.*)" "/NoHost.cgi$1" [PT]
```

NoHost.cgi には、次のようなスクリプトを用意する。

```
#!/usr/bin/perl -Tw

my $msg = "To properly direct your request, this server requires that\n"
        . "your web client include the HTTP 'Host' request header field.\n"
        . "The request which caused this response did not include such\n"
        . "a field, so we cannot determine the correct document for you.\n";
print "Status: 400 Bad Request\r\n\"
```

```
        . "Content-type: text/plain\r\n\"
        . 'Content-length: ' . length($msg) . "\r\n\"
        . "\r\n\"
        . $msg;
    exit(0);
```

解説

この解決策では、リクエストヘッダにHostフィールドのないすべてのリクエストを指定したCGIスクリプトにリダイレクトし、このスクリプトに適切な処理をさせている。

この解決策で使用しているCGIスクリプトでは、そのリクエストとサーバの環境に合わせて、レスポンスのテキストを調整すればよい。例えば、スクリプトの実行時にサーバ自身の設定ファイルを調べて、サーバ上の有効なサイトへのリンク一覧を返してもよい。「Host フィールドを付けてもう一度試してください」といったメッセージを返すだけなら、静的なHTMLファイルでも十分だろう。解決策にあるRewriteRuleディレクティブを次のように置き換えて、nohost.html を用意しておけばよい。

```
RewriteRule ".*" "/nohost.html" [PT]
```

もっとスクリプトを高度にして、例えば、httpd.confファイルからServerNameディレクティブを探して、それを一覧にし、300 Multiple Choices(複数の選択肢がある)レスポンスでそのリンクを表示することもできる。もちろん、クライアントが依然としてHostフィールドを付けずに送信してくると、うまくいかない可能性がある。

参照

- http://httpd.apache.org/docs/mod/mod_rewrite.html

レシピ9.2 CGIスクリプトのレスポンスステータスを変更する

課題

レスポンスステータスを変更したいときがある。例えば、404 Not Found(ページが見つからない)エラーの代わりに、403 Forbidden(禁止)をクライアントに返したい。

解決

ErrorDocument に静的なファイルを指定する代わりに、CGI スクリプトを指定する。CGI 仕様では、スクリプトがレスポンスステータスコードを指定できるようになっている。スクリプトでは、Content-typeフィールドのようなヘッダフィールドを返すのに加えて、Statusという名前のフィールドに返したいステータスの値とテキストを付けて返せばよい。

```
#!/bin/perl -w
print "Content-type: text/html;charset=iso-8859-1\r\n";
print "Status: 403 Access denied\r\n";
    :
```

解説

Apacheはドキュメント処理中にエラーが発生すると(例えば、ファイルが見つからないなど)、デフォルトでは、あらかじめ用意したエラーレスポンスをクライアントに返す。ErrorDocumentディレクティブを使うと、このエラーレスポンスをカスタマイズすることができる。Apacheは通常、エラーステータスを保持したまま、カスタムのエラーテキストをクライアントに送るようになっている。

しかし、ファイルが存在しないという事実を隠すために Forbidden ステータスを返すなど、ステータスを別のものに変えたいときには、Apache にその変更を指示する必要がある。

このためには、ErrorDocument を CGI スクリプトのような動的なページにする必要がある。CGI 仕様は、レスポンスのステータスコードを指定するのに非常に簡単な手段、Status CGI ヘッダフィールドを提供している。解決策ではこの使い方を示した。

参照

- 8 章
- http://httpd.apache.org/docs/mod/core.html#errordocument
- http://www.ietf.org/rfc/rfc3875.txt

レシピ 9.3　エラーメッセージをカスタマイズする

課題

デフォルトの Apache エラーページではなく、カスタマイズしたエラーメッセージを表示したい。

解決

httpd.conf で、ErrorDocument ディレクティブを使えばよい。

```
ErrorDocument 405 /errors/notallowed.html
```

解説

ErrorDocumentディレクティブを使うと、特定のエラー状態が発生したときに、独自のエラーページを作って表示させることができる。この例では、405 Method Not Allowed(許可されていないメソッド)ステータスコードが発生したとき、デフォルトの Apache のエラーページではなく、指定した URL をユーザに表示する。

このページは、Webサイトの他のページと同じ見栄えになるようにカスタマイズすることができる。エラードキュメントがサイトの他のページと全く違う見栄えだと、ユーザは混乱して別のサイトへ迷い込んだと思ってしまうかもしれないためだ。

参照

- http://httpd.apache.org/docs/mod/core.html#errordocument

レシピ 9.4 複数の言語でエラードキュメントを提供する

課題

コンテントネゴシエーションを使った複数言語対応のWebサイトにおいて、エラードキュメントにも同じようにコンテントネゴシエーションを適用したい。

解決

Apache 2.0 のデフォルト設定ファイルでは、この部分は最初はコメントになっているが、少し手を加えると、Web サイトの見栄えをカスタマイズして複数の言語でエラードキュメントを提供することができる。

設定ファイルの該当箇所のコメントを外せばよい。デフォルトの設定ファイルの場合、次のコメントを探せば、該当箇所を見つけることができるだろう。

```
# The internationalized error documents require mod_alias, mod_include
# and mod_negotiation.  To activate them, uncomment the following 30 lines.
```

Apache 1.3でこれを実現するのはかなり難しいが、本書執筆時点で進行中の解決策があり、これは2.0の実装とよく似ている。

解説

Apache 2.0 で提供されているカスタムエラードキュメントは、エラーメッセージを国際化するためのいろいろなテクニックを組み合わせたものだ。本書執筆時点では、エラーメッセージとして、日本語、ドイツ語、英語、スペイン語、フランス語、オランダ語、スウェーデン語、イタリア語、ポルトガル語などが利用可能になっている†。クライアントのブラウザに設定された言語設定に基づいて、エンドユーザの好みの言語でエラーメッセージが送られる。

コンテントネゴシエーションを使うと、ブラウザの設定に基づいて、ユーザにとって適したドキュメント(つまり適した言語)が選ばれる。コンテントネゴシエーションに関する詳しい情報は、コンテントネゴシエーションのドキュメント(Apache 2.0 の場合は http://httpd.apache.org/docs-2.0/content-negotiation.html 、Apache 1.3 の場合は http://httpd.apache.org/docs/content-negotiation.html) を参照すること。

この機能を使うと、適切な言語でエラーメッセージを送るだけではなく、エラーページの見栄えをカスタマイズして、Webサイトの他のページと合わせることもできる。この作業を簡単にするためには、top.htmlとbottom.html というファイルを error ディレクトリの include サブディレクトリに置いて、その Web サイトのコンテンツの標準的なヘッダとフッタに見えるよう変更すればよい。このヘッダとフッタの間にエラーメッセージドキュメントの本体が入ったページが作られるので、エラーページに切り替わってもユーザに不快感を与えることが少なくなる。

エラードキュメントには、SSIディレクティブを含むことができることに注意しよう。ユーザ向けのエラードキュメントをさらにカスタマイズするのにSSIを使うことができる。例えば、404(ファイルが見つからない)エラードキュメントの場合、環境変数HTTP_REFERERが定義されていれば直前のページに戻るリンクを表示し、

† 現時点(バージョン2.2.8) では、日本語も含めて16カ国語に対応している。

定義されていなければページには単に URL が見つからないことをユーザに通知することができる。このドキュメントにさらに他の SSI ディレクティブを追加して、カスタマイズしてもよい。

参照

- http://httpd.apache.org/docs/content-negotiation.html
- http://httpd.apache.org/docs-2.0/content-negotiation.html
- http://apache-cookbook.com
- レシピ 8.9

レシピ 9.5　無効な URL を他のページにリダイレクトする

課題

「ページが見つからない」というページの代わりに、他のページ、例えば、サイトのトップページを表示したい。間違った URL で、サイトの連続性を失いたくないためだ。

解決

ErrorDocument ディレクティブを使って、404 (Not Found) エラーを捕捉し、適切なページを表示すればよい。

```
ErrorDocument 404 /index.html
DirectoryIndex index.html /path/to/notfound.html
```

解説

このレシピでは、誰かが無効な URL をリクエストして 404 エラーが発生すると、すべて /index.html に移動させる。ユーザには Web サイトのトップページが表示されるので、無効な URL なのに有効なコンテンツが得られるように見える。Webサイトで無効なURLにアクセスしているユーザには、探している情報を見つけるのに有益なページが得られるだろう。

一方、このような動作は、そのURLが正しいと信じているユーザを混乱させるかもしれない。グローバルなエラードキュメントとして提供しているページが、本当にWebサイトで何かを探しているユーザの助けになっているか、単に混乱させたり惑わせたりしていないか確認しよう。この例に示したように、サイトのトップページを返すのもよいだろう。そこから、ユーザは探しているものを見つけることができるはずだ。

間違ったURLから正しいコンテンツが得られると、ユーザはブックマークを修正せずに、それが本当に無効になるまで間違ったURLを使い続けてしまう。ログファイルには404エラーが記録されるが、ユーザは無効なURLを使っているとは決して気付かない。これに対して、実際にエラードキュメントを返すと、ユーザは使っているURLが無効だとすぐにわかる。正しいURLがわかると、その新しいURLでブックマークを更新してくれるだろう。

たとえ有効なドキュメントを返したとしても、クライアントには依然としてステータスコード404が返されていることに注意しよう。Webサイトのリンクを確認するための各種ツールを使っている場合、そのツールがコンテンツ内のエラーメッセージではなくステータスコードをチェックしていれば、正しい結果を得るこ

とができる。

参照

- http://httpd.apache.org/docs/mod/core.html#errordocument
- http://httpd.apache.org/docs/mod/mod_dir.html

レシピ 9.6 Internet Explorer にエラーページを表示させる

課題

ErrorDocumentディレクティブを正しく設定したのに、IE（Internet Explorer）は設定したエラーページではなく、IE 自身のエラーページを表示してしまう。独自に設定したエラーページを表示させたい。

解決

エラードキュメントをもっと大きく、少なくとも 512 バイト以上にする。

解説

これは少し奇妙に見えるかもしれないが、事実である。きっとIEは、Webサイト管理者よりも自分の方がよく知っていると考えているのだろう。IE はエラードキュメントが 512 バイト未満なら、400 番台や 500 番台のステータスコードを受け取ると、カスタムエラーページではなくIE内部のエラーメッセージページを表示する。このサイズは実際にはブラウザで設定することができ、クライアントごとに数値を変えてもよい。「HTTPエラーメッセージを簡易表示する」機能をブラウザ全体の設定で、オフにしてもよい。IEの［**ツール**］メニューから［**インターネットオプション**］ダイアログを表示し、［**詳細設定**］タブを選んで、「**ブラウズ**」の［**HTTP エラーメッセージを簡易表示する**］を解除すればよい。

これを最初に見たときには、きっとイライラするだろう。正しく設定してあって、別のブラウザでは動いているように見えるのだ。誰かがエラードキュメントをもっと大きくする必要があると教えてくれても、あまりに信じがたく、からかわれていると思ってしまうのが普通だろう。

しかし、これは真実だ。ページをもっと大きくしよう。少なくとも 512 バイト以上にする必要がある。さもないと、IE はエラーページを無視して、代わりに IE 自身の「親切な」エラーメッセージを表示する。

ページの不足分をどうやって埋めるかは重要ではない。例えば、コメントで大きくしてもよい。次のようなコメントを 6 回繰り返せば、512 バイトを十分上回るだろう。

```
<!-- message-obscuring clients are an abomination
     and an insult to the user's intelligence -->
```

（メッセージを隠してしまうクライアントは、ユーザの知性に対する冒涜と侮辱だ。）

参照

- http://httpd.apache.org/docs/mod/core.html#errordocument

レシピ 9.7　エラー状態を通知する

課題

サーバにエラー状態が発生したとき、それを通知する電子メールを受け取りたい。

解決

ErrorDocumentディレクティブに、静的なドキュメントでなくメールを送るCGIプログラムを指定すればよい。

```
ErrorDocument 404 /cgi-bin/404.cgi
```

404.cgi は次のようなものになる。

```
#!/usr/bin/perl
use Mail::Sendmail;
use strict;

my $message = qq~
Document not found: $ENV{REQUEST_URI}
Link was from: $ENV{HTTP_REFERER}
~;

my %mail = (
            To => 'admin@server.com',
            From => 'website@server.com',
            Subject => 'Broken link',
            Message => $message,
            );
sendmail(%mail);

print "Content-type: text/plain\n\n";
print "Document not found. Admin has been notified\n";
```

解説

このレシピは、こうすることを推奨するわけではなく、ただの例として示しているだけである。かなり大きなサイズやトラフィックのあるWebサイトでこれを実践してしまうと、非常によく管理されたサイトであっても、膨大な量の電子メールが送られてしまうだろう。ユーザがURLを打ち間違ったり、自分ではコントロールできない他のサイトが自分のサイトに間違ったリンクを張っていたりするためだ。しかし、少なくとも短期間設定してみるのは、自分のWebサイトの規模を正しく理解するのに役立つかもしれない。

ErrorDocumentディレクティブでは、すべての404（ドキュメントが見つからない）リクエストを、指定したURLで処理するよう指示している。するとCGIプログラムが起動し、受け取った環境変数を調べて、どのリンクが間

違っているか、リクエストがどこからやってきたのかを判断する。

このスクリプトは電子メールでメッセージを送るために、Mail::Sendmail という Perl モジュールを使っている。このモジュールはどんな OS でもうまく動くはずだ。このモジュールは Perl の標準の一部ではないため、CPAN（http://www.cpan.org）からインストールする必要がある。もちろん、PHPや他のプログラミング言語を使って同じような効果を実現することもできる。

プログラムの最後の 2 行で、とても簡単なページをユーザに表示し、エラー状態が発生したことをユーザに教えている。この代わりに、Webサイトのもっと有益で魅力的なページにユーザを導くようなスクリプトにしてもよい。最後の 2 行を次のような設定に置き換えると、これを実現することができる。

```
print "Location: http://server.name/errorpage.html\n\n";
```

こうすると、サーバはリダイレクトヘッダをクライアントに送り、クライアントは指定されたURLをユーザに表示する。

参照

- http://httpd.apache.org/docs/mod/core.html#errordocument

10章 プロキシ

プロキシとは、他人の代わりに振る舞うということを意味している。Webサーバの場合には、クライアントの代わりに、あるサーバが別のサーバからコンテンツを取得し、それをクライアントに返すということだ。例えば、プロキシサーバの後ろに複数のWebサーバをバックエンドサーバとして隠していてもよい。プロキシサーバは正しいバックエンドサーバにリクエストを転送する責任がある。

Apacheに付属しているmod_proxyは、プロキシ動作処理を行うモジュールだ。本章のレシピでは、この機能を活用するためのいろいろなテクニックを紹介する。プロキシサーバの保護やプロキシを通したコンテンツのキャッシュ、mod_proxyでリクエストを別のポートで動いているサービスに対応付ける方法などについて解説する。

mod_proxy に関する詳しい情報は、Apache 1.3 の場合は http://httpd.apache.org/docs/mod/mod_proxy.html、Apache 2.0 の場合は、http://httpd.apache.org/docs/2.0/mod/mod_proxy.html を参照してほしい。

Apache 2.2では、mod_proxy_balancerなど、mod_proxyに機能を追加するたくさんのサブモジュールが導入された。これらについても、本章で解説する。

プロキシを使用可能にする前に、セキュリティに関する懸念を理解して、プロキシサーバを保護する対策を講じること。(詳細はレシピ 6.20 を参照)

Squid(http://www.squid-cache.org)のような専用のプロキシサーバの導入を検討するのもよいだろう。これらのソフトはプロキシ機能に特化しており、たくさんのオプションがある。

レシピ 10.1　プロキシサーバを保護する

課題

プロキシ機能を使いたいが、知らない人に使われてしまうオープンなプロキシにはしたくない。

解決

Apache 1.3 の場合は、次のように設定すればよい。

```
<Directory proxy:*>
    Order deny,allow
```

```
    Deny from all
    Allow from .yourdomain.com
</Directory>
```

Apache 2.0 の場合には、次のような設定を追加すればよい。

```
<Proxy *>
    Order Deny,Allow
    Deny from all
    Allow from .yourdomain.com
</Proxy>
```

解説

オープンなプロキシを動かすと、任意のインターネットユーザがこのプロキシサーバを経由してWebサイトにアクセスできてしまうという懸念がある。これはいろいろな理由で問題になる。ユーザは事実上、ネットワーク帯域を盗むことになるので、確かに問題だ。しかし、もっと大きな懸念は、ネットワーク管理者がかけた制限を回避するのに使われたり、ユーザが匿名でWebサイトを訪れた結果、自分のネットワークからアクセスしているように見えてしまうことだ。

このレシピにある*.yourdomain.com*は自分のドメイン名に置き換える必要がある。あるいはもっとよいのは、自分のネットワークのネットワークアドレスに置き換えることだ。(IPアドレスはホスト名やドメイン名よりもごまかすのが難しい。)例えば、このレシピの1行を、次のようにするとよい。

```
Allow from 192.168.1
```

プロキシサーバ経由のリソースへのリクエストはすべて、プロキシサーバのログファイルに記録が残ることに注意しよう。クライアントのアドレスやプロキシ経由でリクエストしたリソースが記録される。例えば、次のように記録される。

```
192.168.1.5 - - [26/Feb/2003:21:26:13 -0500] "GET http://httpd.apache.org/docs/mod/
        mod_proxy.html HTTP/1.1" 200 49890
```

このように、プロキシ経由のHTTPトラフィックはすべて記録されるので、ユーザがこのことを知ると、きっとプライバシーの侵害だと思うだろう。サーバがこうしたリクエストを記録しないように設定することもできる。プロキシリクエストのための環境変数を次のように設定する。

```
<Directory proxy:*>
    SetEnv PROXIED 1
</Directory>
```

そして、ログ用ディレクティブで、このリクエストをログに記録しないように指定すればよい。

```
CustomLog /www/logs/access_log common env=!PROXIED
```

参照

- http://httpd.apache.org/docs/mod/mod_proxy.html
- http://httpd.apache.org/docs/mod/mod_log_config.html

レシピ 10.2　プロキシサーバがオープンなメールリレーとして使われることを防ぐ

課題

Apacheサーバをプロキシとして動くようセットアップしたとき、予防措置をとらないとメールリレーとして使われる可能性がある。たとえメールサーバを安全に設定していても、システムが「オープンリレー」として機能するおそれがある。プロキシサーバがオープンリレーとして使われないようにしたい。

解決

mod_rewriteを使って、ポート25（SMTP）へのプロキシリクエストを禁止する。

```
<Directory proxy:*>
    RewriteEngine On
    RewriteRule "^proxy:[a-z]*://[^/]*:25(/|$)" "-" [F,NC,L]
</Directory>
```

解説

ApacheプロキシをSMTPリレーとして使うのは非常に簡単だが、それを防ぐのも簡単だ。この解決策では、単に、リモートメールサーバ（ポート25）に対するプロキシとして使おうとすると、403 Forbiddenのレスポンスを返している。HTTP（ポート80）やHTTPS（ポート443）、FTP（ポート20と21）などのポートは、通常通りプロキシアクセスが可能であり、影響を受けない。

参照

- http://httpd.apache.org/docs/mod/mod_proxy.html
- http://httpd.apache.org/docs/mod/core.html#directory
- http://httpd.apache.org/docs/mod/mod_rewrite.html

レシピ 10.3　別のサーバにリクエストを転送する

課題

特定のURL宛てのリクエストを、別のサーバに透過的に転送したい。

解決

httpd.confで、ProxyPassディレクティブとProxyPassReverseディレクティブを使って、次のように設定すればよい。

```
ProxyPass /other/ http://other.server.com/
ProxyPassReverse /other/ http://other.server.com/
```

解説

これらのディレクティブを使うと、*/other/* で始まる URL 宛てのリクエストは、パス情報を保持したまま *other.server.com* サーバに転送される。つまり、*http://www.server.com/other/something.html* 宛てのリクエストは、*http://other.server.com/something.html* 宛てのリクエストに変換される。他のサーバから取得したコンテンツがクライアントに返されるが、どんなテクニックが使われたのか誰にもわからないだろう。ProxyPassReverseディレクティブは、バックエンドサーバ(この場合、*other.server.com*)から送られたリダイレクトヘッダを変更して、メインサーバからやってきたように見せかける。

この方法は、サイトの動的コンテンツ部分を、mod_perlが動いている別のサーバを使って提供するときによく使われる。別のサーバではなく、同じマシンの異なるポートを使う場合もある。これに対して、サイトの静的コンテンツ部分はメインサーバから提供する。このサーバは、軽くて高速に動かすことができる。

ドキュメントに含まれるURLは、プロキシを経由しても書き換わらないことに注意しよう。ドキュメントのリンクは、絶対 URL ではなく相対 URL であるべきだ。そうしておけば、正しく機能する。

1つのフロントエンドサーバと、インターネットからアクセスできない複数のバックエンドサーバがあり、バックエンドサーバからコンテンツを提供したいという場合に、このレシピを使うとよい。この例では、*/other/*で始まるURL宛てのリクエストがやってくると、Apacheはそのリクエストを*http://other.server.com* 宛てに変換し、取得したコンテンツをクライアントに返す。例えば、*/other/example.html* という URL宛てのリクエストは、*http://other.server.com/example.html* という URL 宛てのリクエストに変換する。

ProxyPassReverseディレクティブを使うと、バックエンドサーバから返ってきたヘッダフィールドをすべて(サーバ名やLocationヘッダを含む)、エンドユーザが実際に使ったURLに書き換え、期待どおりに、リダイレクト機能を動かすことができる。

バックエンドサーバの HTML ドキュメントに含まれるすべてのリンクは、絶対 URL でなく相対 URL にしておく必要があることに注意しよう。こうしておけば、リンクはプロキシサーバ経由のコンテンツでもうまく機能する。このレシピの場合、例えば、/index.htmlへのリンクはURLに*/other/*が含まれていないので、リクエストはもはやプロキシを経由しない。

このテクニックを使うと、ユーザは 1 つの Web サイトのコンテンツを、実際には複数の Web サーバマシンから提供することができる。これは、ネットワーク境界を越える手段として使えるし、メインのWebサーバの負荷を軽減するための負荷分散テクニックとしても使える。

参照

- http://httpd.apache.org/docs/mod/mod_proxy.html
- レシピ 11.12

レシピ10.4　特定の場所に対するプロキシ経由のリクエストをブロックする

課題

プロキシサーバをコンテンツフィルタとして使って、特定の場所へのリクエストを禁止したい。

解決

httpd.conf で ProxyBlock を使って、特定のサイトへのアクセスを拒否すればよい。

```
ProxyBlock forbiddensite.com www.competitor.com monster.com
```

解説

この例は、列挙したサイトへのプロキシ経由のリクエストを禁止する。引数は部分文字列マッチであり、*example.com*は*www.example.com*にマッチし、*example*は*example.com*と*www.example.com*のどちらにもマッチする。

どのコンテンツをプロキシサーバ経由でリクエスト可能にするか、きめ細かくコントロールしたいなら、もっと高機能なソフトウェアを使った方がよいだろう。例えば、Squidのような、この分野の機能をフル装備したソフトウェアがある。

参照

- Squid プロキシサーバ―― http://www.squid-cache.org

レシピ10.5　mod_perlコンテンツを別のサーバから提供する

課題

動的コンテンツを生成するために別のサーバを動かして、動的コンテンツへのリクエストをそのサーバに透過的に対応付けたい。

解決

まず、Apacheをインストールして、ポート90のような通常とは別のポートでApacheを動かし、動的コンテンツを生成できるようにする。そして、メインサーバでは、以下のように設定する。

```
ProxyPass /dynamic/ http://localhost:90/
ProxyPassReverse /dynamic/ http://localhost:90/
```

解説

たいていの動的コンテンツ生成手法は、静的コンテンツを提供するのに比べて、かなり多くのシステムリソースを使う。このため、同じサーバにある静的コンテンツの提供に対しても影響を及ぼし、速度低下を引き起こしてしまう。子プロセスの処理が動的コンテンツの生成に費やされ、その間は静的ファイルを提供できなくなるためだ。

動的コンテンツに専用サーバを用意すると、静的コンテンツをもっと素早く提供することができる。両方

の作業を1つのサーバで行うのに比べて、どちらのサーバも限定した機能だけを実行すればよく、それぞれにインストールするモジュールも少なくて済む。

このテクニックは、mod_perlサーバや、PHPサーバ、その他の動的コンテンツを生成するサーバに利用することができる。あるいは、このテクニックを逆に適用してもよい。例えば、画像ファイルを提供する専用マシンを用意し、mod_mmap_staticを使って、メモリ内キャッシュから非常に素早く静的コンテンツを提供することも可能だ。

この例では、*/dynamic/*で始まるURL宛てのリクエストはすべて、動的コンテンツだけを処理する別のサーバに転送される。このURLにマッチしないリクエストは転送されずに、メインサーバによって処理される。

参照

- http://httpd.apache.org/docs/mod/mod_proxy.html
- 8章

レシピ 10.6 キャッシュプロキシサーバを設定する

課題

キャッシュプロキシサーバを動かしたい。

解決

サーバをプロキシサーバとして設定し、さらにキャッシュファイルを置く場所を指定する。

```
ProxyRequests on
CacheRoot /var/spool/httpd/proxy
```

解説

キャッシュプロキシサーバを動かすと、自分のネットワークのユーザは、他の人がすでにリクエストしたコンテンツにもっと素早くアクセスできるようになる。コンテンツはそのドキュメントの最新バージョンではないかもしれないが、リモートのWebサーバからでなくローカルコピーから取り出すため、かなり素早く取得することが可能だ。

Webコンテンツはますます動的になってきたため、Webコンテンツのほとんどが静的コンテンツだった頃ほど、キャッシュプロキシサーバを動かす意味は薄れてきている。しかし、mod_proxyは、どれをキャッシュしてどれをキャッシュしないか、かなり賢く機能するので、設定しておくとアクセスを高速化してくれるだろう。例えば、画像ファイルのようなドキュメントの静的部分はキャッシュして、時間とともに変化するようなドキュメントはキャッシュせずにリモートから最新バージョンを取得してくれる。

CacheRootディレクティブで指定したディレクトリが、キャッシュしたコンテンツの格納場所になる。このディレクトリは、Apacheを動かしているユーザ(通常、nobody)にとって、書き込み可能である必要がある。こうしておかないと、Apacheはその場所にキャッシュを格納することができない。

最後に注意点を述べておく。ここで解説した機能は、Apache 1.3ではmod_proxyが提供しているが、Apache

2.0ではプロキシとキャッシュ機能がそれぞれmod_proxyとmod_cacheというモジュールに分離されている。いずれにしても、これらのモジュールはデフォルトでは有効になっていない。

参照

- http://httpd.apache.org/docs/mod/mod_proxy.html
- http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
- http://httpd.apache.org/docs/2.2/mod/mod_cache.html

レシピ 10.7　プロキシ経由のコンテンツをフィルタリングする

課題

プロキシ経由のコンテンツにフィルタリングをかけたい。例えば、特定の単語を変更したい。

解決

Apache 2.0 以降では、mod_ext_filter を使って出力フィルタを作り、ユーザにコンテンツを送る前にそのフィルタを通すことができる。

```
ExtFilterDefine naughtywords mode=output intype=text/html cmd="/bin/sed s/darned/blasted/g"

<Proxy *>
    SetOutputFilter naughtywords
</Proxy>
```

解説

このレシピが提供しているのは、とてもばかばかしい「不適切な言葉」フィルタだ。いかがわしい言葉"darned"(ばかな)を、きれいな言葉"blasted"(ひどい)に置き換える。これは、もっと高度なコンテンツ変更に拡張することができる。cmd引数には任意のコマンドラインを指定することができるので、Perlスクリプトや任意のプログラムを使って、好きなようにコンテンツをフィルタリングすることができる。プロキシ経由のコンテンツはすべて、クライアントに送られる前にこのフィルタを通ることになる。

このレシピは Apache 2.0 以降でしか動かないことに注意しよう。mod_ext_filter モジュールや SetOutputFilter ディレクティブ、<Proxy> ディレクティブは、Apache 2.0 以降でしか利用できないためだ。

このようなテクニックには、対処しなければならない倫理上、法律上の問題があることにも注意してほしい。筆者らはあえてどちらの立場もとらない。特に、自分が所有していないコンテンツをプロキシ経由で修正するのは、所有者の著作権を侵害して倫理に反すると見なされるかもしれない。幸いにも、本書は哲学書ではなく技術書である。どうやれば実現できるかを解説しているが、すべきかどうかはあなたの良識と弁護士に任せよう。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

- http://httpd.apache.org/docs/2.2/mod/mod_ext_filter.html

レシピ 10.8 プロキシサーバに認証を要求する

課題

サーバのコンテンツをプロキシ経由で提供したいが、プロキシからコンテンツを提供する前に、ログインとパスワードを要求したい。

解決

標準の認証テクニックを使って、プロキシ変換のコンテンツにログインを要求することができる。

```
ProxyPass "/secretserver/" "http://127.0.0.1:8080"
<Directory "proxy:http://127.0.0.1:8080/">
    AuthName SecretServer
    AuthType Basic
    AuthUserFile /path/to/secretserver.htpasswd
    Require valid-user
</Directory>
```

解説

このテクニックは、特殊用途あるいは機能限定の Web サーバを動かしているが、Apache の豊富なアクセス制御などの機能を適用したい、という場合に役に立つ。ProxyPassディレクティブを使うと、特殊用途のサーバの URI スペースを、メインサーバの一部として扱うことができる。また、特別な `proxy:path` を指定した <Directory> コンテナ構文を使うと、対応する URI だけに Apache の設定を適用することができる。

参照

- レシピ 6.7

レシピ10.9 mod_proxy_balancerを使って負荷分散する

課題

複数のバックエンドサーバの負荷を分散したい。

解決

mod_proxy_balancer を使って、負荷分散クラスタを作る。

```
<Proxy balancer://mycluster>
    BalancerMember http://192.168.1.50:80
    BalancerMember http://192.168.1.51:80
```

```
</Proxy>
ProxyPass /application balancer://mycluster/
```

解説

mod_proxy_balancerは、複数のバックエンドサーバの負荷分散をする興味深いモジュールだ。こうした機能は、これまで高価で複雑な商用ソリューションでしか実現できなかったが、このモジュールを使うと簡単に実現することができる。このモジュールは Apache Web サーバの標準インストールに含まれている。

この例では、2つのメンバで構成される負荷分散クラスタをセットアップし、*/application*というURLをこのクラスタにプロキシしている。

mod_proxy_balancerにはいろいろなオプションがあるが、詳しくはhttp://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html にあるドキュメントを参照してほしい。

例えば、あるサーバが他のサーバよりも強力であり、クラスタに含まれる他のマシンよりも大きな負荷をかけることができるということをオプションで教えることができる。次のように設定すると、このマシンに、他のマシンの 2 倍のトラフィックを受信させるよう指示する。

```
BalancerMember http://192.168.1.51:80 loadfactor=2
```

ProxyPassディレクティブに引数を追加することによって、トラフィックをデータ量(転送バイト数)で分散させるか、リクエスト(ホスト当たりのリクエスト数)で分散させるかを指定することができる。

```
ProxyPass /application balancer://mycluster/ lbmethod=bytraffic
```

もっと詳しい情報については、mod_proxy のドキュメントを参照してほしい。

Web ベースのバランサ管理ツールもあり、次のようにして設定することができる。

```
<Location /balancer-manager>
    SetHandler balancer-manager
</Location>
```

バランサマネージャを使うと、サーバの有効 / 無効やサーバの負荷率などを、サーバを再起動することなく設定することができる。メンテナンスのためにサーバをオフラインにして、必要な処理をした後に復帰させたりする作業を、エンドユーザに影響を与えずに実行できる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html

レシピ 10.10 バーチャルホストをプロキシする

課題

バーチャルホスト全体を別のサーバにプロキシさせたい。

解決

VirtualHost の設定ブロックに、次のように ProxyPass ディレクティブを設定すればよい。

```
<VirtualHost *:80>
    ServerName server2.example.com
    ProxyPass / http://192.168.1.52:80
    ProxyPassReverse / http://192.168.1.52:80
</VirtualHost>
```

解説

このレシピは、このバーチャルホスト宛てのすべてのリクエストを、指定したバックエンドサーバに送り、このバックエンドサーバからコンテンツを提供するように設定している。ProxyPassReverseディレクティブを使うと、バックエンドサーバから発行されたリダイレクトをフロントエンドサーバに正しく書き換えてくれる。これにより、クライアントがバックエンドサーバから直接コンテンツをリクエストしないようにしている。

バックエンドサーバではなくフロントエンドサーバでログファイルを収集しておくと、役に立つだろう。バックエンドサーバ宛てのリクエストは、元のクライアントのアドレスからではなく、プロキシサーバからやってきたように見えてしまう。しかし、フロントエンドサーバで収集されるログファイルには、元のクライアントのアドレスが記録されているためだ。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

レシピ 10.11 FTP のプロキシを拒否する

課題

サーバで、FTP（あるいは、その他のプロトコル）をプロキシしないようにしたい。

解決

mod_proxy_ftp をロードしないようにすればよい。

```
# LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
```

解説

mod_proxyにはいくつかのヘルパーモジュールがあり、プロトコル固有のプロキシ機能を提供している。こうしたモジュールには、HTTP リクエストをプロキシするための mod_proxy_http、FTP リクエストをプロキシするための mod_proxy_ftp、CONNECT HTTP メソッドをサポートし、主にプロキシサーバ経由で SSL リクエストをトンネリングするのに使われる mod_proxy_connect などがある。

FTP リクエストをプロキシしないようにしたいなら、このレシピのように、mod_proxy_ftp の LoadModule ディレクティブをコメントアウトするだけでよい。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_proxy_ftp.html
- http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

11章
パフォーマンス

トレードオフを受け入れるつもりがあり、サーバを遅くしているのがどこか、少し時間をかけてベンチマークすれば、Webサイトをもっと高速化することができる。

パフォーマンスを上げるために設定可能な項目はたくさんある。しかし、他にもたくさん変更を加えなければならないことがある。何を変更するかは、何をあきらめることができて、何をトレードオフにするのかによって違ってくる。例えば、多くの場合、セキュリティとパフォーマンスのどちらかを選ばなければならない。

本章では、何を変更すればよいか提案し、何が大幅な速度低下を引き起こすのか警告する。Webサイトはそれぞれ違っており、あるWebサイトを高速化する設定が、別のサイトでは必ずしも速度向上につながらないこともある。

本章では、ハードウェアの検討、設定ファイルの変更、動的コンテンツの生成など、Webサイトのパフォーマンスに影響を与える要因に関する話題を取り上げる。

よくあるのは、アプリケーション開発者が、運用しているシステムの状況をきちんと考慮せずにプログラムを作ることだ。その結果、100件のテストデータベースでは十分速く動いていたアプリケーションが、200,000件の運用データベースではひどく遅くなってしまうことがある。
テスト環境を少なくとも運用環境と同じくらい厳しくしておけば、運用を開始したときにアプリケーションが予想外に遅くなってしまう可能性を減らすことができる。

レシピ 11.1　必要なメモリ量を測定する

課題

サーバのRAM容量が十分か確認したい。

解決

プロセスリストからApacheのプロセスを見つけて、そのプロセスの平均メモリ量を測定する。その値にピーク負荷（同時に処理するWebクライアントの最大数）を掛ければよい。

解説

ハードウェアレベルでサーバを高速にするには、高速なハードウェアを購入する以外にできることはほとんどない。したがって、必要なだけ十分なRAMがあるのかを確認しておくことは重要だ。

必要なメモリ量を測定するのは、控えめに言っても不正確な科学である。根拠のある推測をするためには、サーバの負荷を測定し、どれくらいメモリを使っているのか調べる必要がある。

1つのApacheプロセスが使うメモリ量は、サーバによってかなり違っている。これは、どんなモジュールがインストールされているか、サーバに何を要求するかによって変わってくる。自分のサーバをよく調べなければ、その環境に必要なメモリ量を正確に見積もることはできない。

topやpsのようなツールを使えば、プロセスリストを調べて、プロセスのサイズを測定することができる。また、mod_statusが提供しているserver-statusハンドラを使えば、ある時点で動いているApacheの全プロセス数を測定することができる。

例えば、Apacheプロセスがそれぞれ4MBのメモリを使っており、ピーク負荷時に125個のApacheのプロセスが動いているということがわかると、このピーク負荷を処理するには、サーバに最低でも500MBのRAMが必要だということになる。メモリは、Apacheだけでなく、OSやシステム上で動いているアプリケーションやサービスにも必要だということを忘れてはいけない。したがって実際には、このピーク負荷を処理するにはこれ以上のメモリが必要になるだろう。

これに対して、何らかの理由で、これ以上サーバのメモリを追加できないときがある。この場合には、このテクニックを使って、サーバが同時に処理できる子プロセスの最大数を把握し、MaxClientsディレクティブを使って、次のようにプロセスの最大数を制限すればよい。

```
MaxClients 125
```

参照

- http://httpd.apache.org/docs/misc/perf-tuning.html

レシピ 11.2　ab を使って Apache をベンチマークする

課題

これからやろうとする変更が、実際にパフォーマンスに影響を与えるかどうかを確認するために、ベンチマークしたい。

解決

ab（Apache bench）を使えばよい。abはApacheをインストールしたbinディレクトリにある。

```
ab -n 1000 -c 10 http://www.example.com/test.html
```

解説

Apache benchはApache付属のコマンドラインユーティリティであり、サーバの基本的なパフォーマンステストを行うことができる。このツールは、設定を少し変更して、変更前後のサーバのパフォーマンスをテ

ストするときに特に役に立つ。

この例のように引数を指定すると、ab は、リソース http://www.example.com/test.html に対して 1000 回（-n 1000 はリクエスト回数を示す）、同時に 10 リクエスト（-c 10 は並列レベルを示す）を発行する。

-h フラグを付けて ab を実行すると、指定可能な引数を調べることができる。特に興味深いのは -k フラグであり、これを指定すると KeepAlive モードを有効にする。この話題についてもっと詳しく知りたければ、次の KeepAlive に関するレシピを参照してほしい。

ab を使ってパフォーマンスを評価するときに、知っておかなければならないことがいくつかある。

Apache bench は、現実の人間がやっているような Web サイトの使い方をまねているわけではない。ある 1 つのパフォーマンスをテストするのに、同じリソースを繰り返しリクエストしているだけだ。例えば、パフォーマンス関連の変更をする前後で ab を利用すると、特定の CGI プログラムのパフォーマンスをテストすることができる。あるいは、特定のディレクトリの .htaccess ファイルやコンテントネゴシエーションの影響を測定するのに、ab を利用してもよい。もちろん、現実のユーザは同じページを繰り返しロードしたりしないので、ab によるパフォーマンス測定結果は、Web サイトの現実世界でのパフォーマンスを反映していないかもしれない。

Web サーバと ab を、同じマシンで動かすべきではない。測定に不確実性をもたらしてしまうためだ。ab と Web サーバはともにシステムリソースを消費するため、同じマシンで動かしてしまうと、ab を別のマシンで動かしてネットワーク経由でサーバにアクセスする場合よりも、かなりパフォーマンスが低下する。しかし、ab を別のマシンで動かすと、今度はネットワークの遅延が生じてしまう。これは ab をサーバと同じマシンで動かしていたときにはなかったものだ。

結局、サーバのパフォーマンスにはたくさんの要因が影響を与えるので、テストするたびに同じ値が得られるわけではない。ネットワークの状況、クライアントやサーバで動いている他のプロセスの状況など、さまざまな要因が結果に大きな影響を与えることになる。環境の違いによる影響を軽減する最善の方法は、何度もテストを実行し、その結果の平均をとることだ。また、テスト間の変更箇所をできるだけ少なく、理想的には 1 つだけにするのがよい。そうすると、その変更がどんな影響を及ぼしたのか、より確実にわかるだろう。

最後に、ab は、ある特定の変更がパフォーマンスを改善するかどうか確認するにはよいが、実際のユーザのシミュレーションをしているのではないということを理解しておく必要がある。実際のユーザは、単純に同じリソースを繰り返し取得するということはなく、サイトのいろいろな場所からいろいろなリソースを取得する。実際のサイトの利用状況は、ab によって明らかにされたものとは異なるパフォーマンス問題を引き起こすかもしれない。

参照

- ab の manpage
- http://httpd.apache.org/docs/2.2/programs/ab.html

レシピ 11.3 KeepAlive 設定を調整する

課題

KeepAlive 関連のディレクティブを、Web サイトにとってできるだけ最善となるよう調整したい。

解決

KeepAlive 設定を有効にして、関係するディレクティブに適切な値を設定する。

```
KeepAlive On
MaxKeepAliveRequests 0
KeepAliveTimeout 15
```

解説

HTTPのデフォルトの動作では、ドキュメントごとに新しいコネクションを作ってリクエストする。このコネクションのオープンとクローズには、かなり時間がかかってしまう。KeepAliveを使うと、1つのコネクションで複数のリクエストを送ることができるので、ソケット接続を確立する時間を減らすことができる。これは、サイトのコンテンツをリクエストするクライアントにとって、ロード時間の短縮になる。

KeepAlive ディレクティブで KeepAlive を有効にするのに加えて、2 つのディレクティブを使って、その動作を調整することができる。

1つは、MaxKeepAliveRequests ディレクティブで、1つのコネクション上にいくつのKeepAliveリクエストを許可するかを指定する。この値を小さく設定する理由はない。このディレクティブのデフォルト値は 100 であり、この値はたいていのサイトでかなりうまく機能しているようだ。この値を0に設定すると、1つのコネクション上に許されるリクエスト数が無制限になる。こうすると、ユーザはサイトのコンテンツをすべて1つのコネクションを使ってロードできるかもしれないが、これはKeepAliveTimeoutの値と、ユーザがどれだけ素早くサイト内を動き回るかによる。

KeepAliveTimeoutディレクティブは、次のリクエストを受信するまで、そのコネクションをどれくらいの時間、オープンしたままにするかを指定する。最適な設定はWebサイトの性質によって変わってくる。この値は、ユーザがあるページのコンテンツを理解して、次のページに移動するまでにかかる時間と考えてもよいだろう。KeepAliveTimeout が期限切れになる前に、ユーザが次のページのリンクをクリックして移動すれば、同じコネクションを使って、次のドキュメントを取得することができる。しかし、期限切れになった後だと、次のページを取得するためには、サーバに新しいコネクションを確立する必要がある。

ユーザがサイトからリソースをロードしてすぐに立ち去ってしまっても、KeepAliveTimeoutで指定した秒数の間、Apacheはコネクションをオープンしたままにする。この間、その子プロセスは他のリクエストを受け付けることはできない。したがって、KeepAliveTimeoutの値を大きくしすぎても、小さくしすぎても望ましくない。

KeepAliveTimeoutを大きくしすぎると、かなりの数のプロセスがKeepAliveモードになり、実際に使われていないのがわかるだろう（例えば、server-statusハンドラでこの様子を観測することができる。レシピ11.4を参照）。時間が経つにつれ、この使われていない子プロセスの代わりにまた新しい子プロセスが作られるので、

子プロセスの数は増え続けていく。

逆に、KeepAliveTimeout を小さくしすぎると、KeepAlive を完全に無効にしたときと同じような状態になってしまい、1つのクライアントが短時間サイトにアクセスしただけでも、たくさんのコネクションが必要になる。大きくしすぎた状態よりも、小さくしすぎた状態を検出する方が難しい。したがって、一般的には、小さくするよりも、大きくした方がまだましだ。

ドキュメントを閲覧するのにかかる時間はユーザによって違っているし、Webサイトのページによっても違っている。したがって、このディレクティブの最適値を決めるのは非常に難しい。しかし、他の対策と比べて、この設定項目がサイト全体のパフォーマンスに大きな影響を与えるとは考えにくい。デフォルト値の 5 にしておけば、たいていのサイトでかなりうまく動くだろう。

参照

- http://httpd.apache.org/docs/2.2/mod/core.html#keepalive
- http://httpd.apache.org/docs/2.2/mod/core.html#maxkeepaliverequests
- http://httpd.apache.org/docs/2.2/mod/core.html#keepalivetimeout

レシピ 11.4　サイトの動作のスナップショットを取得する

課題

サーバが何をしているのか、正確に知りたい。

解決

server-statusハンドラを有効にすると、どんな子プロセスが動いていて何をしているのか、そのスナップショットを取得することができる。もっと詳細に知りたければ、ExtendedStatus を有効にすればよい。

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 192.168.1
</Location>

ExtendedStatus On
```

http://*servername*/server-status にアクセスすると、その結果を見ることができる。

解説

mod_statusが提供しているserver-statusハンドラを使うと、サーバの動作のスナップショットを取得することができる。このモジュールはデフォルトで有効になっている。このスナップショットは、サーバを最後に再起動した日時や、動作時間、提供したデータ量など、基本的な項目を含んでいる。それに続いて、子プ

ロセスのリストと、それらが何をしているのかが表示される。ページの下部には、用語の詳細説明とテーブルの各列の意味が表示される。

サーバステータス画面はバーチャルホストを含むサーバ全体の動作を示している。他人にホスティングサービスを提供している場合は、こうした詳細情報をユーザに見せたくないだろう。

Apacheに付属しているデフォルトの設定ファイルにあるように、このハンドラへのアクセスを制限しておくのが望ましい。このページに含まれる情報の一部には、リクエストしたクライアントのアドレスやドキュメントのリストが含まれている。このような情報をWebサイトですぐに参照可能にしているのは、プライバシーの侵害だと感じる人もいるだろう。さらに、QUERY_STRING変数やPATH_INFO変数、単なるURLなど、ユーザが公開したくない情報が載ってしまうこともある。そのため、このレシピには次のような設定を加えておくのがよい。

```
Order deny,allow
Deny from all
Allow from 192.168.1
```

このように設定すると、192.168.1のネットワークからのみアクセスを許可し、それ以外のインターネット上のユーザからのアクセスを拒否することができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_status.html
- http://httpd.apache.org/server-status/

レシピ 11.5 DNS 検索を回避する

課題

DNS検索は非常に時間がかかるので、クライアントのアドレスのDNS検索が必要な状況を回避したい。

解決

常に、HostnameLookups ディレクティブを Off に設定しておく。

```
HostnameLookups Off
```

そして、Allow from ディレクティブや Deny from ディレクティブでは、できるだけホスト名でなく IP アドレスを使うようにする。

解説

DNS検索は非常に時間がかかることがあり（0秒から60秒）、是が非でも避けるべきである。クライアント

のアドレスが全く検索できないと、タイムアウトするのに 1 分ほどかかることがあり、検索を実行している子プロセスは、この間、他に何もできなくなってしまう。

Apache が DNS 検索を必要とするケースはいろいろあるが、ここでの目標はこのような状況を完全に回避することである。

HostnameLookups

Apache 1.3 以前では、HostnameLookups ディレクティブは、Apache がクライアントの IP アドレスとホスト名のどちらをログに記録するかを指定するのに使われていた。これはデフォルトでonになっており、Apache がログにエントリを記録するたびに、クライアントの IP アドレスをホスト名に変換するための DNS 検索が必要だった。幸いなことに、このディレクティブは現在デフォルトでoffになっており、このままにしておくよう注意すればよい。

アドレスをホスト名に変換する必要があるなら、別のプログラムを、できればWebサーバを運用しているマシンとは別のマシンで実行すべきだ。つまり実際には、ログファイルを別のマシンにコピーして、変換処理を行えばよい。こうすると、DNS 検索によってサーバのパフォーマンスが低下することはなくなる。

Apache には logresolve という名前のユーティリティが付属しており、ログファイルを処理して IP アドレスをホスト名に変換してくれる。さらに、たいていのログファイル分析ツールでも、ログ解析処理の一部として名前解決を実行してくれる。

Allow from ホスト名と Deny from ホスト名

Allow fromディレクティブやDeny fromディレクティブを使って、ホストベースのアクセスコントロールを実施している場合、Apacheはクライアントがホスト名を偽装していないかどうか確かめるための予防措置をとっている。具体的には、クライアントの IP アドレスに対して DNS 検索をして、アクセス制限と比較するための名前を取得する。そして、取得した名前を検索して、DNSレコードが偽造されていないことを確認する†。

したがって、パフォーマンスを上げるためには、Allowディレクティブや Denyディレクティブには、名前ではなく IP アドレスを使うのが望ましい。

参照

- 3 章

レシピ 11.6　シンボリックリンクを最適化する

課題

シンボリックリンクに関するセキュリティの必要性と、Options SymLinksIfOwnerMatch などを利用した解決策によるサーバの速度低下とのバランスをとりたい。

† 例えば、IP アドレスの所有者は、自分の逆引き DNS ゾーンに簡単に PTR レコードを設定することができる。したがって、自分の IP アドレスを他人が所有している名前を指すように変更することができてしまう。

解決

セキュリティを最も厳重にして、シンボリックリンクをほとんど利用しないのであれば、Options SymLinksIfOwnerMatch あるいは Options -FollowSymLinks を使えばよい。パフォーマンスを最大にするには、Options FollowSymLinks を使えばよい。

解説

シンボリックリンクについては、パフォーマンスとセキュリティのどちらを優先すべきか考える必要があり、自分の状況に最もふさわしい判断をする必要がある。Unix 系 OS における通常の日々の操作では、シンボリックリンクはそのリンク先のファイルと同等に見なされている†。ディレクトリに cd しても、それがシンボリックリンクであったかどうかは気にする必要はない。それは全く同じように動作する。

これに対して、サーバがシンボリックリンクをたどらないように設定してあると、Apache はファイルやディレクトリがシンボリックリンクであるかどうかを考慮しなければならない。さらに、Options SymLinksIfOwnerMatch が on になっていると、Apache は特定のファイルがシンボリックリンクであるかをチェックするだけでなく、シンボリックリンクであればリンク自体とその対象の所有者もチェックしなければならない。これは、ある種のセキュリティポリシーを強制するが、かなり時間がかかり、サーバの処理速度を低下させてしまう。

シンボリックリンクに関するセキュリティとパフォーマンスのトレードオフについて、ここにガイドラインを示しておく。

セキュリティを第一に考えるなら、シンボリックリンクをたどることを禁止すること。ドキュメントディレクトリから公開サーバに置きたくないコンテンツへのリンクを作るのは構わない。あるいは、本当にシンボリックリンクが必要な場合には、Options SymLinksIfOwnerMatch を使えばよい。これは自分が所有しているファイルにしかリンクを張れないので、ファイルシステム上でユーザがコントロールできない部分にリンクを張るのを防ぐことができる。

パフォーマンスを第一に考えるなら、常に、Options FollowSymLinks を使い、決して Options SymLinksIfOwner*Match* を使わないこと。Options FollowSymLinks を使うと、Apache はたいていの Unix 系アプリケーションと同じようにシンボリックリンクをたどることができる。つまり、Apache は対象となるファイルがシンボリックリンクであるかどうかを調べる必要がなくなる。

参照

- http://httpd.apache.org/docs/2.2/mod/core.html#options

レシピ 11.7 .htaccess ファイルのパフォーマンスへの影響を最小限にする

課題

ディレクトリごとに設定を行いたいが、.htaccess ファイルによるパフォーマンス低下を避けたい。

† もちろん、これはファイルシステムレベルでは正しくない。ここでは、実際のユーザレベルについて話している。

解決

必要なディレクトリでのみ AllowOverride ディレクティブを有効にし、Apache がその他の .htaccess ファイルを探すのに、時間を費やさないようにする。

```
AllowOverride None
```

そして、<Directory> セクションを使って、必要な .htaccess ファイルだけを有効にすればよい。

解説

.htaccessファイルを使うと、Apacheのパフォーマンスがかなり低下してしまう。リクエストされたパス上のすべてのディレクトリにある.htaccessをチェックし、関連する設定をすべて取得しなければならないためだ。Apacheの設定ディレクティブは、その設定があるディレクトリにだけ適用されるのではなく、すべてのサブディレクトリにも適用される。したがって、現在のディレクトリだけでなく、親ディレクトリにある.htaccessファイルも調べて、現在のディレクトリに至るまでのすべてのディレクティブを見つけ出す必要がある。

例えば、何らかの理由で、すべてのディレクトリに対して AllowOverride All を有効にしてあり、Document Root が /usr/local/apache/htdocs であったとする。このとき、http://example.com/events/parties/christmas.html という URL 宛てにリクエストがあると、以下のファイルを探し、あればファイルをオープンして、設定ディレクティブを検索する。

```
/.htaccess
/usr/.htaccess
/usr/local/.htaccess
/usr/local/apache/.htaccess
/usr/local/apache/htdocs/.htaccess
/usr/local/apache/htdocs/events/.htaccess
/usr/local/apache/htdocs/events/parties/.htaccess
```

AllowOverride Allをファイルシステム全体で有効にすることはまずないだろう。これは最悪のシナリオだ。しかし、この設定ディレクティブが何をしているのか正しく理解していない人が、このオプションをファイルシステム全体で有効にしてしまい、結果としてパフォーマンス低下を引き起こしてしまうことがある。

ここで推奨している解決策は、この問題を解決する最良の方法だ。<Directory>ディレクティブは、特にこの状況では重要になる。.htaccessファイルは、本当に設定変更が必要で、メインサーバの設定ファイルに簡単にアクセスできない状況でのみ使うべきだ。例えば、/usr/local/apache/htdocs/eventsに.htaccessファイルがあり、次のようなディレクティブを含んでいるとする。

```
AddEncoding x-gzip tgz
```

こうする代わりに、メイン設定ファイルに次のように設定すべきだ。

```
<Directory /usr/local/apache/htdocs/event>
    AddEncoding x-gzip tgz
</Directory>
```

つまり、.htaccessで設定する代わりに、同じディレクトリを指す<Directory>セクションで設定すればよい。

Webサイトのどこかで、やむを得ず.htaccessファイルを使う必要があるなら、必要なディレクトリだけに限定すべきだ。例えば、ディレクトリ /www/htdocs/users/leopold/ でどうしても .htaccess ファイルを使う必要があるなら、このディレクトリだけ明示的に許可すべきである。

```
<Directory /www/htdocs/users/leopold>
    AllowOverride All
</Directory>
```

AllowOverrideディレクティブについて、最後に1つ注意しておく。このディレクティブでは、.htaccessファイルに設定できるディレクティブの種類を指定することができるので、実際に必要なディレクティブだけを許可するようにしておこう。つまり、引数としてAllを使うのではなく、必要なディレクティブの種類を指定すべきだ。特に、AllowOverride の引数に Options を指定するのは、できるだけ避けるべきだ。Options ディレクティブが使えてしまうと、セキュリティの観点から使えないようにした機能を、ユーザが自分で使えるようにしてしまうおそれがある。

参照

- http://httpd.apache.org/docs/2.2/howto/htaccess.html

レシピ 11.8 コンテントネゴシエーションを使用不可にする

課題

コンテントネゴシエーションはパフォーマンスを大幅に低下させるので、使用不可にしたい。

解決

必要ないところでは、コンテントネゴシエーションを使用不可にする。コンテントネゴシエーションが必要な場合は、MultiViews オプションではなく、type-map ハンドラを使う。

```
Options -MultiViews
AddHandler type-map .var
```

解説

できれば、コンテントネゴシエーションを使用不可にする。しかし、コンテントネゴシエーションが必要な場合、例えば、多言語Webサイトの場合、MultiViews メソッドではなく、type-mapハンドラを使うべきだ。

MultiViewsを使うと、Apacheはリクエストを処理するたびにディレクトリのリストを作る必要がある。リソースがリクエストされると、ディレクトリのリストと比較して、そのリソースの言語バリエーションが存

在するか調べる。例えば、index.html がリクエストされると、index.html.en と index.html.fr という言語バリエーションが存在するかもしれない。Apacheは、見つかった言語バリエーションを、クライアントから送られたAcceptヘッダに示されたユーザの好みと比較する。これによって、Apacheは、どのリソースがユーザにふさわしいか判断する。

しかし、特にたくさんの言語バリエーションがある巨大なディレクトリやリソースの場合には、この処理にはかなり時間がかかってしまう。この情報を .var ファイルに書いておくと、代わりに type-map ハンドラを使うことができる。これを使うと、ディレクトリのリストを取得する必要がなくなるので、Apacheが適切な言語バリエーションを判断してユーザに送るのにかかる時間を、大幅に短縮することができる。

.varファイルは、特定のリソースに対する言語バリエーションのリストであり、その重要な属性を記述しておくだけでよい。

例えば、リソース index.html に対して、英語、フランス語、ヘブライ語の言語バリエーションを用意するなら、index.html.varという名前の.varファイルに、それぞれの言語情報を記述すればよい。このファイルは次のようになる。

```
URI: index.html.en
Content-language: en
Content-type: text/html

URI: index.html.fr
Content-language: fr
Content-type: text/html

URI: index.html.he.iso8859-8
Content-language: he
Content-type: text/html;charset=ISO-8859-8
```

index.html.varというファイルは、index.html.en、index.html.fr、index.html.he.iso8859-8という名前の言語バリエーションリソースと同じディレクトリに置いておく必要がある。

ドキュメントのヘブライ語バリエーションには、ファイル名と Content-type ヘッダフィールドの両方に文字セットが指定されていることに注意すること。

.var ファイルを使用可能にするには、設定ファイルに AddHandler ディレクティブを追加すればよい。

```
AddHandler type-map .var
```

設定ファイルには、ファイル名に使われているファイル拡張子に対応するディレクティブを設定しておく必要がある。ただし、通常はデフォルトの設定ファイルですでに設定されており、自分で追加する必要はないはずだ。それぞれの言語の拡張子に対応するAddLanguageディレクティブと、文字セットの拡張子に対応する AddCharset ディレクティブが設定されている。

MultiViewsと違って、このテクニックでは、効率の悪いディレクトリのリストの代わりに、.varファイルからすべての情報を取得している。

ネゴシエーションしたドキュメントをキャッシュしておくと、コンテントネゴシエーションによるパフォーマンス低下をさらに軽減することができる。キャッシュを利用するには、次のようなディレクティブを設定すればよい。

```
CacheNegotiatedDocs On
```

ネゴシエーションしたドキュメントをキャッシュしておくと、ユーザが読めない言語や、表示できないドキュメントフォーマットでファイルを取得してしまうといった、期待にそわない結果になることもある。

どんなテクニックを使ったとしても、サーバの速度をかなり低下させるおそれがあるため、できるかぎりコンテントネゴシエーションの利用を避けるべきだ。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_negotiation.html
- http://httpd.apache.org/docs/2.2/mod/mod_mime.html#addhandler
- http://httpd.apache.org/docs/2.2/mod/mod_mime.html#addcharset
- http://httpd.apache.org/docs/2.2/mod/mod_mime.html#addlanguage
- http://httpd.apache.org/docs/2.2/mod/core.html#options

レシピ 11.9　プロセスの生成を最適化する

課題

Apache 1.3を使っている場合、あるいは、Apache 2.0/2.2をprefork MPMで使っている場合、MinSpareServersディレクティブとMaxSpareServersディレクティブの値を、Webサイトにとって最適になるように設定したい。

解決

これらの最適値はサイトによって変わってくる。サイトのトラフィックをよく観察して、その結果に基づいて判断する必要がある。

解説

MinSpareServersディレクティブとMaxSpareServersディレクティブは、待機中のサーバ数の大きさをコントロールする設定値だ。やってくるリクエストに対して常に待機している子プロセスがあるように調整する。具体的には、アイドル状態のプロセス数がMinSpareServersよりも少なくなると、Apacheはその最小値に達するまでプロセスを生成する。同様に、アイドル状態のプロセス数がMaxSpareServersよりも多くなると、Apacheはその最大値を超えないようプロセスを終了する。サイトのトラフィックの変動に合わせて、このような処理が日々行われている。

特定のサイトにおいて、これらのディレクティブの最適値が何であるかは、トラフィックの変動量や変動率によって変わってくる。サイトのトラフィックが一時的に急増する傾向があるなら、MinSpareServers はその急増に耐えられるだけ十分大きくしておく必要がある。これは、リクエストがサイトにやってきたときに、アイドル状態のサーバプロセスが足らないためにリクエストが処理できないという事態を避けるためだ。サイトのトラフィックのパターンが突然急増することなく、かなりゆるやかなカーブを描くようなら、デフォルト値のままで十分だろう。

どれぐらいの負荷がかかっているか正確に観察するには、server-status ハンドラの出力を見るのが最もよい方法だ（レシピ 11.4 を参照）。

また、MaxClients には、サーバ負荷が高いときでもサーバのリソースが不足しないような値を設定するべきだ。例えば、Apache プロセスが平均で 2MB のメモリを消費し、合計で 256MB の RAM が利用可能で、他のプロセスも少しメモリを使用するなら、おそらく MaxClients を 120 よりも大きくしない方がよいだろう。RAM を使い果たしてスワップ空間を使い始めると、サーバのパフォーマンスは急激に低下してしまう。こうなると、スワップ空間を使わなくなるまでパフォーマンスは回復しない。top などのプログラムを動かしてメモリ使用量を観察し、どんなプロセスが動いていて、それぞれどれくらいメモリを使用しているのか調べるとよいだろう。

参照

- レシピ 11.10

レシピ 11.10　スレッドの生成をチューニングする

課題

スレッド対応 MPM の 1 つを使って Apache 2.0 を動かしているが、スレッド数の設定を最適化したい。

解決

設定の最適値はサーバによって変わってくる。

解説

Apache 2.0 にはいろいろなスレッド対応 MPM があり、それぞれ違ったやり方でスレッドを生成している。Apache 1.3 では、Windows 版と Netware 版はスレッドに対応しているが、Unix 版はスレッドに対応していない。スレッドの生成に関する値のチューニングは、どのバージョンを使うかによって変わってくる。

シングルプロセス MPM におけるスレッド数の設定

Windows MPM（mpm_winnt）や、Apache 1.3 の Windows 版や Netware 版のような、スレッド対応の子プロセスを 1 つだけ実行する MPM では、子プロセスは一定数のスレッドを生成する。このスレッド数は、ThreadsPerChild ディレクティブを使ってコントロールすることができる。サイトのピーク時のトラフィックが処理できるよう、この値は十分大きくしておく必要がある。Apache プロセス実行中のスレッド数は一定なので、これ以上パフォーマンスチューニングの余地はない。

worker MPMを使ったときのスレッド数の設定

worker MPMでは、子プロセスあたりのスレッド数は一定だが、子プロセスの数は可変であり、サーバ負荷の増大を吸収することができる。標準的な設定は次のようなものになるだろう。

```
StartServers 2
MaxClients 150
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25
ServerLimit 16
```

MinSpareServersディレクティブとMaxSpareServersディレクティブを使って、アイドル状態の待機スレッド数の大きさをコントロールする。これらの値は、クライアントのリクエストに対して、常にアイドル状態のスレッドが待機しているように設定すればよい。ThreadsPerChildディレクティブを使って、それぞれの子プロセスあたりのスレッド数を指定する。利用可能なアイドル状態のスレッド数がMinSpareServersを下回ると、Apacheは新しい子プロセスを起動して、その子プロセスがThreadsPerChildの数だけスレッドを生成する。同様に、サーバの負荷が減ってきて、アイドル状態のスレッド数がMaxSpareServersを上回ると、Apacheは子プロセスを終了させ、アイドル状態のスレッド数をMaxSpareServersの値以下にまで減らそうとする。

これらの値を設定する目的は、常にアイドル状態のスレッドを準備しておき、新しいクライアントからリクエストがやってきても、新しいスレッドを生成することなく処理できるようにしておくことだ。この例は、たいていのサイトでうまく動くだろう。この設定では、全く使われていない子プロセスが少なくとも1つあり、その子プロセスには25個のスレッドがあり、リクエストがやってくるのを待ち受けている。このプロセスのスレッドが1つでも使われるとすぐに、Apacheは将来のリクエストに備えて新しい子プロセスを起動する。

MaxClientsディレクティブとServerLimitディレクティブの値は、新しい子プロセスを生成しても決してRAMを使い果たさないように設定する必要がある。topなどのユーティリティを使ってプロセスリストを調べて、ServerLimitにサーバプロセスのサイズを掛けた値が利用可能なRAMサイズを超えないようにしておく必要がある。MaxClientsの値は、ServerLimitにThreadsPerChildを掛けた値と同じか、それ以下にしておく必要がある。

Netwareあるいはperchild MPMを使ったときのスレッド数の設定

ほとんどのMPMでは、MinSpareServersディレクティブとMaxSpareServersディレクティブは、サーバ全体に適用されるが、perchild MPMとnetware MPMでは、これらのディレクティブは子プロセスごとに適用される。netware MPMでは子プロセスは1つだけなので、結局同じことになる。

netware MPMでは、予備のスレッド数が、MinSpareServersとMaxSpareServersの範囲内に収まるよう、必要に応じてスレッドの生成と終了を行う。スレッドの総数は、常に、MaxClientsディレクティブに指定された上限以下でなければならない。

参照

- http://httpd.apache.org/docs/2.2/mpm.html

レシピ 11.11　頻繁に参照されるファイルをキャッシュする

課題

サイトのトップページのような、頻繁に参照されるファイルをキャッシュしたい。そうすれば、毎回ファイルシステムからロードする必要がなくなる。

解決

Apache 1.3 では mod_mmap_static を、Apache 2.0 では mod_file_cache を使って、メモリ上にファイルをキャッシュすればよい。

```
MMapFile /www/htdocs/index.html
MMapFile /www/htdocs/other_page.html
```

Apache 2.0では、MMapFileディレクティブかCacheFileディレクティブのどちらかを使用することができる。MMapFile は、メモリ上にファイルの内容をキャッシュする。これに対して、CacheFile はファイルハンドルをキャッシュするので、パフォーマンスは少し低下するが、メモリ消費は少なくて済む。

```
CacheFile /www/htdocs/index.html
CacheFile /www/htdocs/other_page.html
```

解説

ディスクアクセス時間を削減するために、頻繁にアクセスされるファイルは、何らかの方法でキャッシュするのが望ましい。MMapFileディレクティブを使うと、ファイルをRAMにロードして、以後、そのファイル宛てのリクエストがやってくると、ファイルシステムからではなくRAMから直接提供する。これに対して、CacheFileディレクティブを使うと、ファイルをオープンして、そのファイルハンドルをキャッシュし、以後のファイルオープンの時間を削減する。

Apache 1.3では、mod_mmap_staticモジュールを使うとこの機能を利用できるが、実験的なものとされており、デフォルトでは Apache に組み込まれない。このモジュールを有効にするには、Apache のビルド時、configureに--enable-module=mmap_staticを指定する必要がある。mod_mmap_staticが提供するのは、MMapFileディレクティブだけだ。

Apache 2.0では、mod_file_cacheがこの機能を提供しているが、これも実験的なものとされており、デフォルトでは Apache に組み込まれない。このモジュールを有効にするには、Apache のビルド時、configure に--enable-file-cache を指定する必要がある。mod_file_cache は、MMapFile と CacheFile ディレクティブの両方を提供している。

これらのディレクティブの引数に指定できるのは、単一のファイルだけであり、ディレクトリやディレクトリの集合を指定することはできない。あるディレクトリに含まれるコンテンツ全体をメモリ上にマップし

たい場合、ドキュメントでは次のようにするよう提案している。まず、対象とするディレクトリに対して、次のコマンドを実行する。

```
% find /www/htdocs -type f -print \
> | sed -e 's/.*/mmapfile &/' > /www/conf/mmap.conf
```

次に、メインサーバ設定ファイルに、Includeディレクティブを使って、このコマンドが生成したファイルをロードする。

```
Include /www/conf/mmap.conf
```

こうすると、そのディレクトリに含まれるファイルすべてに、MMapFileディレクティブを適用することができる。

これら2つのディレクティブを使ってファイルをキャッシュしているときには、ファイルの変更を反映するためにサーバを再起動しなければならないことに注意すること。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_mmap_static.html
- http://httpd.apache.org/docs/2.2/mod/mod_file_cache.html

レシピ 11.12　複数のサーバ間で均等に負荷を分散する

課題

複数のサーバで同じコンテンツを提供して、サーバ間で均等に負荷を分散させたい。

解決

DNS ラウンドロビンを使うと、リクエストをサーバ間で(ほぼ)均等に分散させることができる。

```
www.example.com.   86400   IN   A  192.168.10.2
www.example.com.   86400   IN   A  192.168.10.3
www.example.com.   86400   IN   A  192.168.10.4
www.example.com.   86400   IN   A  192.168.10.5
www.example.com.   86400   IN   A  192.168.10.6
www.example.com.   86400   IN   A  192.168.10.7
```

設定ファイルに次の行を追加する。

```
FileETag MTime Size
```

解説

この例は、BIND ゾーンファイルから抜粋したものだ。実際の構文は、動かしているネームサーバソフトウェアによって異なるだろう。

同じホスト名に複数のアドレスを割り当てると、リストしたサーバ間で、均等にアクセスを分散させることができる。ホスト名が要求されると、ネームサーバはラウンドロビン方式でアドレスを返す。これにより、リクエストは各サーバに順番に送られるようになる。個々のサーバは、指定されたホスト名宛てのリクエストに応答するように設定しておくだけでよい。

対象とするホスト名でhostコマンドを実行すると、可能なアドレスのリストが返ってくる。しかし、コマンドを実行するたびに、異なるリストが返ってくる、

```
% host www.example.com
www.example.com has address 192.168.10.2
www.example.com has address 192.168.10.3
www.example.com has address 192.168.10.4
www.example.com has address 192.168.10.5
www.example.com has address 192.168.10.6
www.example.com has address 192.168.10.7
% host www.example.com
www.example.com has address 192.168.10.7
www.example.com has address 192.168.10.2
www.example.com has address 192.168.10.3
www.example.com has address 192.168.10.4
www.example.com has address 192.168.10.5
www.example.com has address 192.168.10.6
```

DNSゾーンファイルを更新したときには、シリアル番号も更新して、DNSサーバを再起動または再ロードする必要がある。

キャッシュの鮮度を判断するのに使われるドキュメントの特徴の1つにETagの値があり、この値はサーバに関連している。通常、この値はドキュメントの実際のディスク位置に基づいて計算されているため、バックエンドホストごとに違う値になる。FileETagを使ってこの情報を取り除くように設定すると、ドキュメントが全く同じであれば、ETagも全く同じ値になり、キャッシュされているものと見分けがつかないようにすることができる。

参照

- 『DNS & BIND 第 4 版』（オライリー・ジャパン発行、原書『DNS and Bind』、Paul Albitz、Cricket Liu 共著、O'Reilly Media 発行）
- レシピ 10.3

レシピ 11.13 ディレクトリのリストをキャッシュする

課題

ディレクトリのリストを提供したいが、そのときのパフォーマンス低下を軽減したい。

解決

IndexOptions の引数に TrackModified を指定して、自動生成したディレクトリのインデックス結果をブラウザがキャッシュできるようにする。

```
IndexOptions +TrackModified
```

解説

ディレクトリのリストをクライアントに送るとき、Apacheはそのディレクトリをオープンし、ディレクトリのリストを取得し、その中にあるファイルのさまざまな属性を調べなければならない。この作業には非常に時間がかかるため、できるかぎり避けた方がよい。

デフォルトでは、ディレクトリのリストと一緒に送られる最終更新日時は、コンテンツを提供した日時になる。つまり、クライアントやプロキシサーバが、キャッシュにあるコピーを使うかどうか判断しようと、HEADリクエストや条件付きGETリクエストをすると、常にキャッシュではなくコンテンツの最新バージョンを取得することになる。IndexOptions に TrackModified オプションを付けると、mod_autoindex は、ディレクトリ中で一番最近更新されたファイルの最終更新日時を付けて、クライアントに送るようになる。こうすると、ブラウザやプロキシサーバは、毎回サーバからコンテンツを取得するのではなく、コンテンツをキャッシュできるようになり、キャッシュされたリストを常に最新バージョンになるよう保つことができる。

キャッシュを実装していないクライアントにとっては、このディレクティブを設定しても恩恵を受けることはないことに注意しよう。具体的に言うと、abはコンテンツをキャッシュしないので、この設定をabでテストしても改善は見られないだろう。

参照

- ab の manpage
- http://httpd.apache.org/docs/2.2/programs/ab.html

レシピ11.14 mod_perlを使ってPerl CGIプログラムを高速化する

課題

Perl CGI プログラムを動かしているが、もっと高速に動かしたい。

解決

mod_perl モジュールをインストールしてあれば、mod_cgi の代わりに、mod_perl で Perl CGI プログラムを動かすよう設定することができる。これによって、CGIプログラム自体を変更しなくても、大幅にパフォー

マンスを改善することができる。

これを実現するには、少し異なる 2 つの方法がある。

Apache 1.3 で mod_perl のバージョン 1 を使う場合には、以下のように設定する。

```
Alias /cgi-perl/ /usr/local/apache/cgi-bin/
<Location /cgi-perl>
    Options ExecCGI
    SetHandler perl-script
    PerlHandler Apache::PerlRun
    PerlSendHeader On
</Location>

Alias /perl/ /usr/local/apache/cgi-bin/
<Location /perl>
    Options ExecCGI
    SetHandler perl-script
    PerlHandler Apache::Registry
    PerlSendHeader On
</Location>
```

Apache 2.0 で mod_perl バージョン 2 を使う場合は、少し構文が変わってくる。

```
PerlModule ModPerl::PerlRun
Alias /cgi-perl/ /usr/local/apache2/cgi-bin/
<Location /cgi-perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::PerlRun
    Options +ExecCGI
</Location>

PerlModule ModPerl::Registry
Alias /perl/ /usr/local/apache2/cgi-bin/
<Location /perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    Options +ExecCGI
</Location>
```

解説

mod_perlのCGIモードを使うと、CGIプログラム自体を全く変更することなく、パフォーマンスを改善することができる。このように設定すると、これまでhttp://www.example.com/cgi-bin/program.cgiというURLでアクセスしていたCGIプログラムは、PerlRunモードでは、http://www.example.com/cgi-perl/program.cgiという URL になり、Registry モードでは、http://www.example.com/perl/program.cgi という URL になる。

PerlRun モードと Registry モードの主な違いは、Registry モードでは、コンパイル後にプログラム自体がキャッシュされるが、PerlRunモードではそうではないという点だ。RegistryモードはPerlRunモードよりも高速だが、コードの品質が高くなければならない。具体的には、グローバル変数をたくさん使ったり、不注意なコードを書いたりしてしまうと、メモリリークを引き起こすおそれがある。最終的にはサーバが利用可能なメモリを使い果たしてしまうことがある。

mod_perlで動かすPerl CGIプログラムを書く場合、あるいは、一般にPerlプログラムを書く場合には、プログラムファイルの先頭の#!の行に続いて、次の 2 行を書いておくことを推奨する。

```
use strict;
use warnings;
```

この2行を追加しても、エラーメッセージが表示されずに動いていれば、Registryモードで動かしても問題ないだろう。

strictは、Perl5以前のバージョンでは使用できない。warningsは、Perl 5.6以前では使用できない。Perl 5.6 よりも前のバージョンでは、Perl に -w フラグを指定すると、warnings と同様の動作をする。Perl プログラムの #!の行に -w を追加すればよい。

```
#!/usr/bin/perl -w
```

参照

- 『プログラミングPerl 第3版』(オライリー・ジャパン発行、Larry Wall、Tom Christiansen、Jon Orwant 共著、原書『Programming Perl』、O'Reilly Media 発行)

レシピ 11.15　動的コンテンツをキャッシュする

課題

実際にはほとんど変更しないのだが動的に生成しているドキュメントをキャッシュしたい。

解決

Apache 2.2 では、次のように設定する。

```
CacheEnable disk /
CacheRoot /var/www/cache
CacheIgnoreCacheControl On
CacheDefaultExpire 600
```

Apache 2.3 以降では、次のように設定する。

```
CacheEnable disk /
CacheRoot /var/www/cache
CacheDefaultExpire 600
CacheMinExpire 600
```

解説

キャッシュは通常、動的コンテンツに対して使うことはできない。動的コンテンツはその名前の通り、リクエストに応じて生成されるコンテンツである。つまり、リクエストされるたびに新しく作られる。したがって、それをキャッシュすることは、その本来の性質に反していることになる。

しかし、動的に生成されるコンテンツが実際には、時々刻々変化しているわけではないことが多いし、それが普通でさえある。前回リクエストされたときから実際には変わっていないのにコンテンツを生成してしまうので、ひどく時間を浪費していると言えるだろう。1秒間に数回コンテンツを生成すると、実際に必要以上の負荷をサーバにかけていることになる。

このレシピでは、少し異なる 2 つの方法で、この課題を解決している。Apache 2.3 での解決策は、よりよいものだが、本書執筆時点で、Apache 2.3 はまだリリースされていない。したがって、この非常に実用的な解決策はまだ利用できない。

Apache 2.2での解決策は、キャッシュコントロールを無効にするというアプローチをとっている。つまり、Apacheが、動的コンテンツに対するリクエストにキャッシュを使わないようにしているのを無視させて、とにかくキャッシュを使うように設定している。おまけに、デフォルトのキャッシュ有効期限を5分(600秒)に設定しているので、少なくともその期間は、キャッシュされたコンテンツは保持される。

Apache 2.3 での解決策は、もう少し簡潔だ。デフォルトのキャッシュ有効期限を設定するだけでなく、最小限のキャッシュ有効期限も 5 分に設定している。こうすると、すべてのコンテンツは少なくとも 5 分間キャッシュされるが、コンテンツ自体が望むだけ長い時間を指定することができる。Apache 2.2 での解決策との大きな違いは、2.2ではコンテンツ自身がもっと長くキャッシュされるよう要求しても無視されてしまうが、2.3 ではこれを引き受けてくれるという点だ。

Apache サーバのバージョン 2.4 がリリースされたときには、まずこの Apache 2.3 での解決策が動くかどうか確かめよう。2.3は開発ブランチであり、一般に利用する準備ができると2.4というバージョンになるはずだ。

なお、CacheRootに指定したディレクトリが存在しており、Apacheユーザが書き込み可能であることを確認しておこう。

参照

- http://httpd.apache.org/docs/2.2/caching.html

12章 ディレクトリのリスト表示

デフォルトのApache HTTPサーバには、mod_autoindexというモジュールが含まれており、これを使うとディレクトリのリストをWebページとして表示することができる。デフォルトの表示は簡潔で有用だが、このモジュールには出力を微調整してカスタマイズするさまざまな方法がある。

レシピ 12.1　ディレクトリやフォルダのリストを生成する

課題

ディレクトリがリクエストされたときに、ディレクトリのリストを表示したい。

解決

リクエストされるディレクトリに対してOptions Indexesを付ける。

```
<Directory /www/htdocs/images>
    Options +Indexes
</Directory>
```

解説

URLがファイルシステム上のディレクトリやフォルダに対応している場合、Apacheは次の3つの方法のいずれかでそのリクエストに応答しようとする。

1. mod_dirがサーバの設定に含まれており、かつ、対応するディレクトリがDirectoryIndexディレクティブの適用範囲内にあり、さらに、サーバがそのディレクティブで指定されたファイルの1つを見つけることができた場合、そのファイルを使ってレスポンスを生成する。

2. mod_autoindexがサーバの設定に含まれており、かつ、対応するディレクトリがIndexesキーワードを有効にしたOptionsディレクティブの適用範囲内にある場合、サーバは実行時にディレクトリのリストを作成し、そのリストをレスポンスとして返す。

3. サーバは404(「リソースが見つからない」)ステータスを返す。

ディレクトリのリスト表示を有効にする

ディレクトリ内にあるファイルリストをサーバに自動生成させるのに必要な鍵となるのは、設定にmod_autoindexを含み、Optionsディレクティブに Indexesキーワードを含んでいることだ。Optionsディレクティブには、絶対的な形式として次のように指定するか、

```
Options FollowSymLinks Indexes
```

もしくは、選択的、相対的な形式で、次のように指定すればよい。

```
Options -ExecCGI +Indexes
```

ディレクトリのリスト表示を有効にするときは、十分注意する必要がある。スコープの継承メカニズム（詳しくはhttp://httpd.apache.org/docs/2.2/sections.html#merginを参照）によって、この設定は、ディレクトリツリーのずっと下位のディレクトリにまで影響を与えてしまう。また、サーバは何らかのレスポンスを返そうと、このセクションのはじめに列挙した一連のルールをとにかく適用しようとするので、たった1つのファイル不足が、ファイルシステム上のコンテンツを不用意に公開してしまうことにつながるおそれがある。

リスト表示を有効にしたディレクトリのサブディレクトリで、リスト表示を無効にする

この問題をうまく解決して、ある1つのディレクトリに対してのみリスト表示を有効にするには、2つの方法がある。

- 個々のサブディレクトリの .htaccess ファイルに、"Options -Indexes" を追加する。
- すべてのサブディレクトリにマッチする <Directory> コンテナに、"Options -Indexes" を追加する。

例えば、ディレクトリ/usr/local/htdocs/archivesではリスト表示を有効にし、そのサブディレクトリではリスト表示を無効にしたいときには、次のようにすればよい。

```
<Directory /usr/local/htdocs/archives>
    Options +Indexes
</Directory>

<Directory /usr/local/htdocs/archives/*>
    Options -Indexes
</Directory>
```

もしすべてのサブディレクトリではなく、特定のサブディレクトリだけリスト表示を無効にしたいのであれば、作業はもう少し複雑になる。そのサブディレクトリのリストがある程度小さいならば、<DirectoryMatch> ディレクティブを使って、次のように実現することができる。

```
<Directory /usr/local/htdocs/archives>
    Options +Indexes
</Directory>

<DirectoryMatch /usr/local/htdocs/archives/(images|video|audio)>
    Options -Indexes
</DirectoryMatch>
```

参照

- http://httpd.apache.org/docs/2.2/mod/core.html#options
- http://httpd.apache.org/docs/2.2/mod/mod_dir.html
- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.2　標準的なヘッダとフッタを付けてディレクトリのリストを表示する

課題

ディレクトリのリストの上部にヘッダを、下部にフッタを表示したい。

解決

```
# 望むなら標準の HTML ヘッダを削除する
IndexOptions +SuppressHTMLPreamble
HeaderName /includes/header.html
ReadmeName /includes/footer.html
```

解説

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表示におけるヘッダとフッタに使うファイルの URI を指定することができる。

HeaderName ファイルに、HTML の <head> タグや <title> タグなど、HTML ドキュメントの開始に関連するタグが含まれる場合、IndexOptions +SuppressHTMLPreambleディレクティブを使って、mod_autoindexが自動的にHTMLヘッダを生成しないように設定するのがよいだろう。そうしないと、ヘッダ要素を2つ持ったHTMLドキュメントができてしまい、ヘッダで設定した属性がブラウザに無視されてしまうおそれがある。

HeaderNameディレクティブとReadmeNameディレクティブに指定する引数は、どちらも現在のディレクトリに対する相対URIである。先頭にスラッシュがなければ、現在のディレクトリに対する相対パスとして解釈される。先頭にスラッシュがあれば、DocumentRoot に対する相対 URL として解釈される。

HeaderNameとReadmeNameに指定するドキュメントはどんなに複雑であってもよく、好きなページレイアウトを使って、自動生成されたディレクトリのリスト表示のまわりを包むことができる。ページレイアウトの効果を出すために、例えば、ヘッダで<table>や セクションをオープンし、フッタでそれらをクローズするということも可能だ。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.3　スタイルシートを適用する

課題

HeaderNameドキュメントを使わずに、ディレクトリのリスト表示にCSSスタイルシートを適用したい。

解決

次のように設定すればよい。

```
IndexStyleSheet /styles/listing.css
```

解説

IndexStyleSheetディレクティブを使うと、インデックスリスト表示に使うCSSファイル名を指定することができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.4　リスト表示の一部を隠す

課題

ディレクトリのリスト表示から特定のファイルを隠したい。

解決

次のように設定すればよい。

```
IndexIgnore *.tmp *.swp .svn secret.txt
```

解説

ディレクトリのリスト表示から削除したいファイルもあるはずだ。一時的なファイルやスワップファイル、その他、自動生成されたファイルなどは、サイトを訪問するユーザには見せる必要がないものだ。CVSが作るCVSディレクトリや、Subversionが作る.svnディレクトリなど、バージョン管理用のディレクトリも、訪問者にとって有益な情報はなく、表示する必要はないだろう。

このテクニックはプライベートなドキュメントや、秘密のドキュメントを隠すのにも使うことができるが、ファイル名を知っていたり推測したりすれば、依然としてそのファイルにアクセスできるということを理解しておく必要がある。ディレクトリのリスト表示からは隠されるが、アクセスは可能なままだ。したがって、セキュリティを期待して、このテクニックを使ってはいけない。
パスワードで暗号化したファイルは、ディレクトリのリスト表示から自動的に削除される。

参照

- レシピ 12.19
- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ12.5　ディレクトリのリスト表示から特定のファイルを検索する

課題

ファイル名でリストをフィルタリングできるようにしたい。

解決

URL の QUERY_STRING として、P(パターン)引数を使えばよい。

```
http://servername/directory/?P=a*
```

あるいは、HeaderName ファイルに次のような HTML フォームを置くと、ディレクトリのリスト表示に検索機能を付けることができる。

```
<form action="" method="get">
Show files matching <input type="text" name="P" value="*" />
<input type="submit" value="Go" />
<form>
```

解説

Apache 2.0.23 では、mod_autoindex にたくさんの新しいオプションが追加され、クライアントがディレクトリのリスト表示出力をもっとコントロールできるようになった。URL の QUERY_STRING にオプションを指定することによって、ソート順や、出力フォーマット、このレシピにあるようにリスト表示するファイルを変更することができる。

?P= QUERY_STRINGを使うと、引数によってファイルのリスト表示をフィルタリングすることができる。例えば、http://*servername/directory/*?P=a* という URL にアクセスすると、a で始まるファイルがリスト表示される。この機能はApache 2.0で新しく導入された機能であり、それ以前のバージョンで同じ効果を得る方法はない。

参照

- レシピ 12.16

- レシピ 12.2

レシピ 12.6　ディレクトリのリストをソートする

課題

ディレクトリのリストをデフォルトとは違う順にソートしたい。

解決

例えば、次のように設定すればよい。

```
IndexOrderDefault Descending Date
```

解説

設定ファイルか、.htaccessファイルで、IndexOrderDefaultディレクティブを使うと、ディレクトリのリストのデフォルト表示順を指定することができる。例えば、ファイルを最新のものから順に表示したいときには、この解決策のように指定すればよい。

IndexOrderDefaultに指定できる引数には、次のようなものがある。

- Name――ファイル名またはディレクトリ名
- Date――ファイルの最終更新日時
- Size――バイト単位でのファイルサイズ
- Description――(もしあれば)ファイルの説明。AddDescriptionディレクティブを使って設定することができる。

いずれも、昇順(Ascending)と降順(Descending)を指定することができる。IndexOptions IgnoreClientで明示的に禁止されていなければ、エンドユーザがQUERY_STRINGを指定すると、IndexOrderDefaultの値を上書きすることができる。

参照

- レシピ 12.17
- レシピ 12.7

レシピ 12.7　クライアントがソート順を指定できるようにする

課題

エンドユーザがリストの表示順を指定できるようにしたい。

解決

ユーザは、QUERY_STRING 引数を使って、ソート順を変更することができる。

```
http://servername/directory/?C=D&O=D
```

あるいは、ユーザがソート順を選択できるように、HeaderName ファイルに以下を置いてもよい。

```
<form action="" method="get">
Order by by <select name="C">
<option value="N" selected="selected"> Name</option>
<option value="M"> Date Modified</option>
<option value="S"> Size</option>
<option value="D"> Description</option>
</select>
<select name="O">
<option value="A" selected="selected"> Ascending</option>
<option value="D"> Descending</option>
</select>
<input type="submit" value="Go" />
</form>
```

解説

エンドユーザが見かけをコントロールできるようにするのは、Webコンテンツをもっと有益にする強力な方法だ。IndexOptionsの引数にIgnoreClientを指定して、この機能を明示的に無効にしていない限り、ユーザは、QUERY_STRING オプションに?C= や?O= を使って、ディレクトリのリスト表示順を変更することができる。

順序O(Order)には、昇順A(Ascending)か、降順D(Descending)を指定することができる。列C(Column)には、次のいずれかを指定することができる。

- N――ファイル名またはディレクトリ名
- M――ファイルやディレクトリの最終更新日時
- S―― バイト単位でのファイルサイズ
- D―― AddDescription ディレクティブを使って設定したファイルの説明

不正な引数があると、引数の解析処理は終了してしまう。

Apache 1.3 では、代わりに次の構文を使うことができる。

```
http://servername/directory/?X=Y
```

X は先に解説した **n**、**m**、**s**、**d** のいずれかで、*Y* は昇順の **a** か、降順の **d** のどちらかになる。

参照

- レシピ 12.16
- レシピ 12.17
- レシピ 12.2

レシピ 12.8　リストの表示フォーマットを指定する

課題

リストの表示フォーマットのレベルを指定したい。

解決

設定できる表示フォーマットには、3 つのレベルがある。フォーマットしないか、フォーマットするか、HTML テーブルを使って表示するかのいずれかである。

派手なインデックスにするには、次のようにすればよい。

```
IndexOptions FancyIndexing
IndexOptions FancyIndexing HTMLTable
```

解説

"fancy" フォーマットは、たいていの Apache のデフォルト設定になっており、最もよく目にしているはずだ。HTMLTable フォーマットはそれほど使われていないが、何もしないよりも少し違った見せ方になる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#indexoptions

レシピ12.9　クライアントが表示フォーマットを指定できるようにする

課題

エンドユーザがリストの表示フォーマットを指定できるようにしたい。

解決

クエリ文字列にF引数を加えると、ユーザは使いたい表示フォーマットオプションを指定することができる。

黒丸の付いた箇条書きのリストを指定するには次のようにする。

```
http://www.example.com/icons/?F=0
```

フォーマットされたリストを指定するには次のようにする。

```
http://www.example.com/icons/?F=1
```

HTML テーブルで書かれたリストを指定するには次のようにする。

```
http://www.example.com/icons/?F=2
```

解説

IndexOptions IgnoreClientが指定されていなければ、エンドユーザはクエリ文字列の引数に数字を指定することで、レイアウトをカスタマイズすることができる。F引数によって、リストの表示フォーマットをコントロールすることができる。

参照

- レシピ 12.16

レシピ 12.10　ファイルに説明を付ける

課題

リスト表示にファイルの簡単な説明を付けたい。

解決

AddDescription ディレクティブを使うと、特定のファイルやファイル群に説明を追加することができる。

```
AddDescription "GIF image" .gif
```

解説

特定のファイルや特定のパターンにマッチしたファイルに、説明を付けることができる。AddDescription ディレクティブの最初の引数が、使用する説明になる。2番目の引数は、ファイル名と比較する部分文字列になる。このパターンにマッチしたファイルに説明が付くことになる。

説明に使える文字数は、デフォルトで 23 文字だ。IndexOptions DescriptionWidth ディレクティブで指定するか、他の列のいずれかを表示しないように設定すると、説明のスペースを変更することができる。

説明が長すぎないかよく確認しよう。そうしないと、幅の限界に達して、説明が切り詰められてしまう。説明が切り詰められて読めなくなると、ただ邪魔なものになってしまう。また、説明にはHTMLを使うこともできるが、HTML が切り詰められると HTML タグがクローズされないままになるおそれがあるので注意しよう。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#adddescription

レシピ12.11 ドキュメントのタイトルから説明を自動生成する

課題

HTML ファイルの説明を自動生成したい。

解決

説明を付けたい<Directory>スコープにおいて、次のように設定すると、HTMLファイルの<Title>タグから自動的にタイトルを読み込んで説明を付けることができる。

```
IndexOptions ScanHTMLTitles
```

解説

たくさんのHTMLファイルがあるディレクトリをリスト表示する場合、そのHTMLドキュメントのタイトルが自動的に説明の欄に表示されると便利なことがある。

ScanHTMLTitlesオプションを付けると、mod_autoindexはHTMLファイルの<Title>タグの内容を調べて、それを説明に使う。

この処理はかなり激しいファイルアクセスを伴うため、ディレクトリにあるHTMLファイルの数に比例して、かなりのパフォーマンス低下を引き起こしてしまうだろう。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#indexoptions

レシピ 12.12 リスト表示のアイコンを変える

課題

ディレクトリのリスト表示に、違うアイコンを使いたい。

解決

AddIconやその変形ディレクティブを使うと、ファイルの種類ごとに使用するアイコンを指定することができる。

```
AddIcon /icons/image.gif .gif .jpg .png
```

解説

AddIconディレクティブにはたくさんの変形があり、アイコンをファイルやファイル群、ファイルの種類などに関連付けることができる。

AddIconディレクティブを使うと、特定のパターンにマッチするファイルに使用するアイコンを設定することができる。最初の引数は、使用するアイコンのURIになる。その次に続く引数は、そのアイコンが使われるファイル拡張子や、ファイル名の一部、完全なファイル名になる。

引数として、ディレクトリのための^^DIRECTORY^^や、空行に使う^^BLANKICON^^を指定することもできる。空白が正しく入っていることを確認しよう。

親ディレクトリへのリンクに使用するアイコンを指定するには、".."という引数を使えばよい。

```
AddIcon /icons/up_one.gif ".."
```

x-gzipのような特定のエンコーディングのファイルに使用するアイコンを指定するには、AddIconByEncodingディレクティブを使えばよい。

```
AddIconByEncoding /icons/gzip.gif x-gzip
```

AddIconByTypeディレクティブを使うと、アイコンを特定のMIMEタイプに関連付けることができる。

```
AddIconByType /icons/text.gif text/*
AddIconByType /icons/html.gif text/html
```

最後に、どれにもマッチしなかったときに使用するデフォルトアイコンを指定することもできる。

```
DefaultIcon /icons/unknown.png
```

いずれのディレクティブも、画像の読み込みを無効にしているクライアントに対して表示する、代替テキストを指定することができる。次のように、括弧を付けて、アイコンのパスの前に代替テキストを書けばよい。

```
AddIcon (IMAGE,/icons/image.gif) .gif .png .jpg
```

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#addicon

レシピ 12.13　ディレクトリからリスト表示する

課題

ディレクトリのリスト表示の先頭をフォルダ(ディレクトリ)にしたい。

解決

ディレクトリのリスト表示において、アルファベット順で表示するのではなく、最初にディレクトリから表示するためには、設定ファイルに次のように指定すればよい。

```
IndexOptions FoldersFirst
```

解説

ディレクトリのリスト表示は、デフォルトでは、ディレクトリも含めたアルファベット順で表示される。しかし、最初にディレクトリを表示して、その後にファイルを表示したい人もいる。こうすると、ディレクトリ構造が深いときにも、より迅速なナビゲーションが可能だ。

FoldersFirst オプションを指定すると、リストの先頭にフォルダを表示し、その後にファイルをアルファベット順で表示する。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#indexoptions

レシピ 12.14 バージョン番号順に表示する

課題

ファイルをバージョン番号順に並べたい。1.10 は、1.2 の前ではなく、1.9 の後に表示されるようにしたい。

解決

ファイルをバージョン番号順にソートするには、設定ファイルに次の行を追加すればよい。

```
IndexOptions VersionSort
```

解説

ソフトウェアを配布しているサイトでは、1つのディレクトリに複数のバージョンのソフトウェアを置くことがよくある。このとき、アルファベット順ではなく、バージョン番号順に並んでいると便利だ。アルファベット順だと、httpd-1.10.tar.gzが、httpd-1.1.tar.gzとhttpd-1.2.tar.gzの間に表示されてしまうが、バージョン番号順にすると、httpd-1.9.tar.gz の次に httpd-1.10.tar.gz が表示されるようになる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html#addicon

レシピ 12.15 ユーザがバージョン番号によるソートを指定できるようにする

課題

ユーザがバージョン番号によるソートを有効にしたり、無効にしたりできるようにしたい。

解決

URL にクエリ文字列引数として V を追加することによって、ユーザはバージョン番号順に表示するかどう

かを指定することができる。

バージョン番号順に表示するには、次のように指定する。

```
http://www.example.com/download/?V=1
```

バージョン番号順に表示しないなら、次のように指定する。

```
http://www.example.com/download/?V=0
```

解説

F引数と同様に、V引数を使うと、ユーザはディレクトリのリスト表示のフォーマットをカスタマイズすることができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ12.16　ユーザが表示出力を完全にコントロールできるようにする

課題

これまでに説明したテクニックを組み合わせて、ユーザがディレクトリのリスト表示出力を完全にコントロールできるようにしたい。

解決

ファイルに次のような HTML を書いて、これをディレクトリのリスト表示のヘッダとして使う。

```
<form action="" method="get">
Show me a <select name="F">
<option value="0"> Plain list</option>
<option value="1" selected="selected"> Fancy list</option>
<option value="2"> Table list</option>
<select>
Sorted by <select name="C">
<option value="N" selected="selected"> Name</option>
<option value="M"> Date Modified</option>
<option value="S"> Size</option>
<option value="D"> Description</option>
<select>
<select name="O">
<option value="A" selected="selected"> Ascending</option>
<option value="D"> Descending</option>
<select>
```

```
<select name="V">
<option value="0" selected="selected"> in Normal
order</option>
<option value="1"> in Version order</option>
<select>
Matching <input type="text" name="P" value="*" />
<input type="submit" value="Go" />
<form>
```

解説

これまでのレシピでは、ユーザがクエリ文字列にフォーマットそし指定するオプションを見てきた。しかし、これについて知っている人はほとんどいないだろう。

このレシピでは、ユーザがあらゆる方法を使えるようにし、ページの中でいろいろな表示フォーマットオプションを選べるようにする。この HTML を header.html という名前で保存して、HeaderName ディレクティブに次のように指定すると、ディレクトリのリスト表示に使用することができる。

```
HeaderName /header.html
```

ユーザは、いろいろなオプションを選んだり、リストを並べ替えたり、文字列を検索するなど、思う存分、表示出力フォーマットをコントロールすることができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.17 ユーザがリスト表示を変更できないようにする

課題

ユーザがディレクトリのリスト表示の出力を変更できないようにしたい。

解決

制限したい <Directory> スコープに、次のような IndexOptions ディレクティブを設定すればよい。

```
IndexOptions +IgnoreClient
```

解説

通常は、ユーザが自分の好きなようにリスト表示の出力をコントロールできるのが望ましいが、特定のディレクトリのリスト表示を決まった方法で見せたいときもある。そのため、ユーザがその表示を変更できないようにしたい。

たいていのユーザはできること自体知らないと思われるが、デフォルトでは誰でも、QUERY_STRING引数の組み合わせでディレクトリのリスト表示の出力を変更することができる。このレシピは、この機能を無効に

する。

IgnoreClientを設定すると、SuppressColumnSortingも一緒に適用される。つまり、各列の一番上にあるクリックできるヘッダは削除されるので、ユーザがこのリンクを使ってソート順を変更できると思ってしまうことはない。

IgnoreClient は、ディレクトリのリスト表示のデフォルト順序を変更する IndexOrderDefault ディレクティブと一緒に使うことができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.18　リストの特定の列を表示しない

課題

ディレクトリのリスト表示で、特定の列を表示しないようにしたい。

解決

IndexOptions ディレクティブに Suppress* 引数のいずれかを指定すると、リストのその列を隠すことができる。例えば、最終更新日時の列を表示したくないなら、次のようにすればよい。

```
IndexOptions SuppressLastModified
```

解説

ディレクトリのリスト表示において、ファイル名以外のすべての列は、次のような IndexOptions 引数のいずれかを使うことで、隠すことができる。

- SuppressDescription――説明の列を表示しない
- SuppressIcon――通常ファイル名の隣にあるアイコンを表示しない
- SuppressLastModified――ファイルの最終更新日時の列を表示しない
- SuppressSize――ファイルサイズの列を表示しない

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ12.19　リスト表示が禁止されているファイルを表示する

課題

パスワード保護されているファイルやディレクトリは通常､ディレクトリのリストには表示されないが､これを表示するようにしたい。

解決

Apache 2.2を動かしているなら、対象とするディレクトリを指している<Directory>ブロックか、そのディレクトリにある.htaccessファイルに、次のようなIndexOptionsディレクティブを設定すればよい。

```
IndexOptions +ShowForbidden
```

Apache 2.0を動かしているなら、これに対する解決策はない。

解説

Apache 2.0におけるディレクトリのリスト表示では、保護されているドキュメントを守ろうとする。そのファイルやディレクトリにパスワード認証が必要であれば、ディレクトリのリスト表示には表示されなくなっている。

Apache 2.0では、禁止されているファイルやディレクトリを、ディレクトリのリストに表示させる方法はない。

Apache 2.2では、IndexOptionsディレクティブにShowForbiddenを付けることで、表示させることができる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html

レシピ 12.20　ディレクトリのリストにエイリアスを表示する

課題

エイリアスはディレクトリのリストには表示されないが、これを表示するようにしたい。

解決

そのディレクトリに、エイリアスと同じ名前のファイルやディレクトリを置けばよい。リストにはこのファイルやディレクトリが表示されるが、クリックするとエイリアスが呼び出される。

解説

エイリアスはディレクトリのリストには表示されない。mod_autoindexが実際にディレクトリをリスト表示するときに、ファイルシステムに問い合わせてリストを生成しているためである。ファイルシステムはエイリアスについては何も知らない。

mod_autoindexには、これらのエイリアスの存在を知ってリストに追加する方法はない。

しかし、ディレクトリにエイリアスの代わりになるアイテムを置くことはできる。エイリアスはファイルシステムより前にチェックされる。そのため、実際にファイルをクリックすると、エイリアスが呼び出されて、ファイルは無視されることになる。

参照

- http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html
- http://httpd.apache.org/docs/2.2/mod/mod_alias.html

13章
その他のトピック

何百もの設定ディレクティブと何十ものモジュールが追加機能を提供しているため、Apache Webサーバはひどく複雑になる可能性がある。したがって、Apacheの使用方法についての質問もひどく複雑になる可能性がある。本書では、最も頻繁に質問される課題をたくさん集めて分類し、関連する話題を1つの章にまとめた。

しかし、カテゴリにうまく分類できない課題もあれば、かなり基本的な課題もあった。そこで、このような課題を集めた「どこにも属さない課題」を、本章にまとめた。

レシピ 13.1　ディレクティブを適切な場所に書く

課題

必要なディレクティブはわかったが、どこに書けばいいのかわからない。

解決

ディレクティブのスコープをグローバルにしたい(つまり、Webサーバ全体のリクエストに適用したい)のであれば、設定ファイルのメイン部分に書くか、<Directory> 行で始まり </Directory> 行で終わるセクションの中に書く必要がある。

ディレクティブを特定のディレクトリに適用したいのであれば、そのディレクトリを指定した<Directory>セクションに書く必要がある。この方法で指定したディレクティブは、そのディレクトリのサブディレクトリにも適用されることに注意しよう。

同様に、ディレクティブを特定のバーチャルホストやURLの一群に適用したいのであれば、そのディレクティブを適用したいスコープを指す <VirtualHost> セクションや <Location> セクション、もしくは <Files> セクションの中に書く必要がある。

要するに、「どこに書けばいいのか」という質問の答えは、「どこに適用したいのか」ということになる。

解説

これはおそらく、Apacheについて質問する場で、最も頻繁に出てくる質問だろう。特定の状況に対して適切な方法を答えることはできるが、一般的に使える万能な方法というものはない。

設定ファイルを複数のファイルに分割して、Includeディレクティブを使って読み込むことが多いので、状況はさらに複雑になる。また、ディレクティブをどのファイルに書くかで得られる効果が違ってくると、誤解している人も多い。

ディレクティブをどこに書くべきか正しく判断するには、Apacheがセクション(例えば、<Directory>や<Location>など)をどう扱うのかを理解する必要がある。とにかくディレクティブを置けばうまく動いてくれるような、魔法のような場所はない。しかし通常、ディレクティブを置くことができる場所はたくさんあり、場所によっては望ましくない影響を引き起こすこともある。

設定ファイルにディレクティブを追加したのに、望み通りの効果が得られない場合には、2つの状況が考えられる。1つは、同じスコープ内にある別のディレクティブによって、後から上書きされている場合である。もう1つは、もっと狭いスコープに別のディレクティブが書かれている場合である。

最初の状況の場合、Apacheの設定ファイルが上から下に向かって解析されるということを理解しておくことが重要だ。Includeディレクティブを使ってファイルを取り込むと、そのIncludeディレクティブが置かれた場所に、そのファイルがそのまま書かれているものと見なされる。同じディレクティブが別の値で2回出現すると、実際に反映されるのは、最後に出現したディレクティブの値になる。

もう1つの状況の場合、あるディレクトリに指定したディレクティブはそのサブディレクトリにも適用されるが、「浅い」ディレクトリを指す<Directory>セクションよりも、もっと狭い「深い」ディレクトリを指す<Directory>セクションの方が優先されるということを理解しておく必要がある。例えば、次のような設定を考えてみる。

```
<Directory /www/docs>
    Options ExecCGI
</Directory>

<Directory /www/docs/mod>
    Options Includes
</Directory>
```

ディレクトリ/www/docs/mod/misc/にあるファイルにアクセスすると、Options ExecCGIではなくOptions Includesが適用される。これはより狭い(深い)ディレクトリのセクションの設定が適用されるためだ。

最後に、.htaccessファイルについても、同様に考慮しなければならない。このファイルは、メインサーバの設定ファイルを上書きすることができる。そのため、混乱を招き、追跡困難な状況を引き起こすおそれがある。

参照

- .htaccessファイルについて
  http://httpd.apache.org/docs/howto/htaccess.html
  http://httpd.apache.org/docs/2.2/howto/htaccess.html

- ディレクトリについて
  http://httpd.apache.org/docs/mod/core.html#directory

http://httpd.apache.org/docs/mod/core.html#directorymatch
http://httpd.apache.org/docs/2.2/mod/core.html#directory
http://httpd.apache.org/docs/2.2/mod/core.html#directorymatch

- ロケーションについて

http://httpd.apache.org/docs/mod/core.html#location
http://httpd.apache.org/docs/mod/core.html#locationmatch
http://httpd.apache.org/docs/2.2/mod/core.html#location
http://httpd.apache.org/docs/2.2/mod/core.html#locationmatch

- ファイルについて

http://httpd.apache.org/docs/mod/core.html#files
http://httpd.apache.org/docs/mod/core.html#filesmatch
http://httpd.apache.org/docs/2.2/mod/core.html#files
http://httpd.apache.org/docs/2.2/mod/core.html#filesmatch

レシピ 13.2　.htaccess ファイルの名前を変更する

課題

ディレクトリごとの設定ファイルのデフォルト名（.htaccess）を変更したい。Windowsでは、ドット（.）で始まるファイル名は問題を引き起こすことがある。

解決

AccessFileName ディレクティブを使って、新しい名前を指定すればよい。

```
AccessFileName ht.access
```

解説

サーバ全体の設定ファイルに加えて、ディレクトリごとに、特別なファイルにディレクティブを追加することができる。このファイルのデフォルト名は、.htaccess となっている。

ドットで始まるファイル名はUnixの慣例であるが、すべてのプラットフォームでうまく動くわけではない。特に Windows では、ドットで始まるファイル名のファイルを編集するのが難しい。

AccessFileNameディレクティブを使うと、ディレクトリごとの設定ファイルのファイル名を変更することができる。そのプラットフォームで有効であれば、どんなファイル名を使っても構わない。

AccessFileNameディレクティブを使うときには、Webから取得できないように<FilesMatch "^\.ht">コンテナで設定している箇所を適切に変更しなければならないことに注意しよう。例えば、.htaccessをht.accessというファイル名に変更するには、次のようにすればよい。

```
<FilesMatch "^ht\.">
    Order deny,allow
    Deny from all
</FilesMatch>
```

参照

- レシピ 11.7
- http://httpd.apache.org/docs/howto/htaccess.html
- http://httpd.apache.org/docs/2.2/howto/htaccess.html

レシピ 13.3「末尾のスラッシュ」問題を解決する

課題

URLに末尾のスラッシュを付けるとうまく読み込めるが、末尾にスラッシュを付けないと読み込めない。この問題を解決したい。

解決

ServerName が正しく設定されており、すべての Alias ディレクティブの末尾がスラッシュで終わっていないことを確認しよう。

解説

「末尾のスラッシュ」問題は、設定上の2つの問題のいずれかによって引き起こされる。1つは、ServerNameが間違っているか指定されていない場合であり、もう1つは、末尾のスラッシュの付いたAliasが設定されていて末尾にスラッシュを付けないと動かない場合である。

間違った ServerName

この問題を引き起こす最もよくある原因は、ServerNameが間違っているか、指定されていないことだ。この場合、次のような動きになる。例えば、http://example.com/something という URL をリクエストすると、somethingがディレクトリ名であれば、Apacheはクライアントにリダイレクトを送り、末尾にスラッシュを付けるよう指示する。

この時サーバは、ServerName の値とリクエストされた URL を使って、リダイレクトの URL を生成する。ServerNameが正しく設定されていないと、クライアントに生成したURLを送っても、クライアントはそのURLを見つけることができず、エラーになってしまう。

これに対して、ServerNameを全く設定していないと、Apacheは起動時に、適切な値を推測しようとする。このとき、間違った推測をしてしまうことが多く、127.0.0.1やlocalhostといった値が設定されてしまい、リモートクライアントがうまく動かないことになる。いずれの場合も、クライアントはアクセスできないURLを受け取ることになる。

不正な Alias ディレクティブ

例えば、次のようなディレクティブを考えてみよう。

```
Alias /example/ /home/www/example/
```

Alias ディレクティブは文字通り、/example/ で始まる URL をエイリアスするが、/example で始まる URL はエイリアスしない。つまり、http://example.com/example/ というURLにアクセスすると、ディレクトリ/home/www/example/ にあるデフォルトドキュメントが表示されるが、http://example.com/example というURL にアクセスすると「ファイルが見つかりません」というエラーメッセージが表示されることになる。そして、エラーログには、次のようなエントリが記録される。

```
File does not exist: /usr/local/apache/htdocs/example
```

これに対する解決策は、Aliasディレクティブの設定には、末尾にスラッシュを付けないようにすることだ。次のように設定すれば、末尾にスラッシュがあってもなくてもうまく機能する。

```
Alias /example /home/www/example
```

参照

- http://httpd.apache.org/docs/misc/FAQ-E.html#set-servername

レシピ13.4　ブラウザの能力に応じてContent-Typeヘッダを設定する

課題

ブラウザごとに別の Content-Type ヘッダを設定したい。さもないと、ブラウザはコンテンツを正しく表示できないかもしれない。

解決

RewriteCond ディレクティブを使って、Accept ヘッダを調べ、T フラグによって、Content-Type ヘッダを指定する。

```
RewriteCond "%{HTTP_ACCEPT}" "application/xhtml\+xml"
RewriteCond "%{HTTP_ACCEPT}" "!application/xhtml\+xml\s*;\s*q=0+(?:\.0*[^0-9])"
RewriteRule . - [T=application/xhtml+xml;charset=iso-8859-1]
```

解説

ブラウザごとに違ったやり方でコンテンツを処理することがあり、ときには、少し適切な指示をしてあげる必要がある。この例では、ブラウザが XHTML コンテンツを望んでいれば、提供するコンテンツがその要件を満たしていることを示す Content-Type を付けて送る。

T(Type) フラグに指定した値が、レスポンスのContent-Typeヘッダの値になる。

参照

- http://httpd.apache.org/docs/mod/mod_rewrite.html

レシピ13.5 Hostヘッダフィールドがないリクエストを処理する

課題

Hostヘッダフィールドがないすべてのリクエストに対して、別の処理をしたい。

解決

httpd.conf に、以下のように設定する。

```
SetEnvIf Host "^$" no_host=1
Order Allow,Deny
Allow from all
Deny from env=no_host
RewriteCond "%{HTTP_HOST}" "^$"
RewriteRule ".*"           -     [F,L]
```

解説

リクエストヘッダのHostヘッダフィールドは、ネームベースのバーチャルホストを正しく処理するのに必要不可欠である(レシピ4.1を参照)。クライアントがこれを付けないと、リクエストが間違ったバーチャルホストに送られてしまう可能性が高い。最近のブラウザは、すべて自動的にこのフィールドを付けてくれるので、この問題に遭遇するのは独自に書いたクライアントか、かなり古いクライアントだけだろう。

この解決策では、こうしたリクエストを403 Forbiddenステータスで拒否している。エラーページのテキストはErrorDocument 403 ディレクティブを使って設定することもできる。

ただし、ErrorDocument を指定しない方が、少し効率がよい。

参照

- レシピ 4.1

レシピ 13.6 デフォルトドキュメントを変更する

課題

デフォルトで表示されるファイルを index.html 以外のファイルにしたい。

解決

DirectoryIndex を使って、新しいファイル名を指定することができる。

```
DirectoryIndex default.htm
```

解説

ディレクトリがリクエストされたとき、つまり、ファイル名でなく/で終わるURLがリクエストされると、mod_dirは、そのディレクトリのインデックスドキュメントを選び、そのファイルをレスポンスとして提供する。デフォルトのインデックスファイルはindex.htmlという名前だが、DirectoryIndexディレクティブを使うと、別の名前に変更するができる。

DirectoryIndexには複数のファイル名を指定することができ、並び順で優先度付けされることに注意しよう。

```
DirectoryIndex index.html index.htm index.php default.htm
```

例えば、CGIプログラムのように別のディレクトリにあるコンテンツをロードしたいなら、相対URLで指定することもできることに注意しよう。

```
DirectoryIndex /cgi-bin/index.pl
```

参照

- http://httpd.apache.org/docs/mod/mod_dir.html

レシピ13.7　デフォルトの"favicon"（お気に入りアイコン）を設定する

課題

サイトに、デフォルトのお気に入りアイコン、通称"favicon"を定義したいが、個々のサイトやユーザが上書きできるようにしたい。

解決

デフォルトのfavicon.icoファイルをServerRootの/icons/サブディレクトリに置く。そして、サーバ設定ファイルの中で、そのアイコンを使いたいスコープ（例えば、特定の<VirtualHost>コンテナの中や、外部すべて）に次のような設定を追加すればよい。

```
AddType image/x-icon .ico
<Files favicon.ico>
    ErrorDocument 404 /icons/favicon.ico
</Files>
```

解説

favicon.icoファイルを使うと、Webサイトは小さな（16×16ピクセル）画像をクライアントに提供し、クライアントはその画像を使って、ページにラベルを付けることができる。例えば、Mozillaブラウザはロケーションバーやページタブにfaviconを表示している。これらのファイルは通常、サイトのDocumentRootか、faviconを参照しているページと同じディレクトリに置いておく。

この解決策では、存在しないfavicon.icoファイルを参照すると、代わりにデフォルトのfavicon.icoファイルを提供するという仕掛けになっている。期待したファイルが見つからないときにだけ、デフォルトのファイルを提供したいため、RewriteRuleの代わりにErrorDocumentディレクティブを使っている。RewriteRuleを使う場合には、注意深く設定しないと、適切なファイルが存在するのにもかかわらず、デフォルトのファイルを提供してしまうおそれがある。

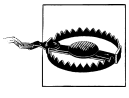
参照

- 5章

レシピ13.8 ScriptAliasされたディレクトリをリスト表示する

課題

ScriptAliasディレクティブで設定されたディレクトリで、ディレクトリインデックスを表示したい。



これはよくない考えだ。見知らぬ人に見られると致命的なスクリプト名を見せてしまうかもしれない。

解決

<Directory>コンテナに次のように追加して、ScriptAliasされたディレクトリの特性を定義すればよい。

```
<Files ".">
    Options Indexes FollowSymLinks
    SetHandler httpd/unix-directory
</Files>
```

解説

ScriptAliasディレクティブは、それを適用するディレクトリにたくさんの制限をかける。これは主に、セキュリティの観点からである。なんと言っても、このようなディレクトリには、システムで実行される任意のスクリプトが含まれているためである。よく知られた人気のあるスクリプトを使っていて、後からそのスクリプトに脆弱性が見つかると、Web上の誰もがその脆弱性を利用できてしまうおそれがある。

意図的に明示的にかけられている制限の1つは、ファイルシステム上のScriptAliasされている部分では、ディレクトリのリスト表示ができないということだ。これはいわゆる「隠すことによるセキュリティ」と呼ばれるものであり、問題を隠して、たとえ簡単にアクセスできたとしても、誰にも見つからないことを期待しているにすぎない。そうは言っても、サーバがどんなスクリプト実行しているのかを公表するよりはましだ。

しかし、ディレクトリのリスト表示、あるいは、少なくともDirectoryIndexディレクティブで指定したファイルによる擬似的なリスト表示ができるようにしたい場合もある。そのためには、この特別な保護を上書き

する必要がある。具体的には、この保護をディレクトリ自体に適用しないよう(<Files ".">コンテナによって指示する)、また、このディレクトリがスクリプトでなくディレクトリとして扱われるように(AddHandlerディレクティブによって)指示する必要がある。

これは実際のところ、Apache の内部処理を、想定外の用途で利用している。そのため、Apache の将来のバージョンでは動かなくなるかもしれない。

参照

- 12 章

レシピ 13.9　.htaccess ファイルを有効にする

課題

サーバのディレクトリで .htaccess ファイルを使えるようにしたい。

解決

httpd.confファイルにおいて、.htaccessファイルを有効にしたいディレクトリに対応するスコープに、次の行を追加すればよい。

```
AllowOverride keyword ...
```

解説

AllowOverrideディレクティブにNone以外のキーワードを指定すると、サーバは、該当するスコープにある.htaccessを処理する。どのキーワードを使ったらよいかは、.htaccessファイルを使って何がしたいかによって違ってくる。

Microsoft Windowsのように、ドットで始まるファイル名を嫌うプラットフォームもある。この特殊な性質をうまくなだめるには、次のように、AccessFileNameディレクティブを使って、サーバが別のファイル名を使うように設定すればよい。

```
AccessFileName ht.access
```

Apacheが.htaccessファイルを無視しているようにみえるなら、そのファイルにただのテキストを書いて、同じディレクトリにあるドキュメントを閲覧してみるとよい。

```
This is not an Apache directive
```

サーバが.htaccessファイルを読み込んでいれば、このテキストに関して文句を言うだろう(Apacheディレクティブではないため)。サーバのエラーログにメッセージが記録され、ブラウザにはInternal Server Errorのページが表示されるだろう。どちらも起こらなければ、ファイルは無視されている。設定したスコープと

AllowOverrideディレクティブを調べよう。

参照

- http://httpd.apache.org/docs/mod/core.html#allowoverride

レシピ13.10 IBM/LotusのSSIをApacheのSSIに変換する

課題

IBMのWeb Traffic ExpressやLotusのDomino Go Webserverで動かしていたWebサーバにあったドキュメントを、Apacheを動かしているサーバに移行したい。

解決

WTE/LDGWのSSIディレクティブの多くは、直接、Apacheのフォーマットに移植することができる。しかし、うまく動かすには修正しなければならないSSIもいくつかある。以下はそのような例外を示している。

- config cmntmsg—— Apacheには同等の設定はない。
- echoディレクティブの変数SSI_DIR、SSI_FILE、SSI_INCLUDE、SSI_PARENT、SSI_ROOT—— Apacheには、これらの自動変数に相当するものはない。
- global——このSSIディレクティブに相当するものはない。現在のファイルで設定した変数は、ファイルの後で参照してもよいが、挿入したドキュメントでは利用することができない。
- setディレクティブ—— Apacheのsetディレクティブは、SSI変数の代用である&varname;を理解することができないことを除いて、WTE/LDGWのものとほぼ同じだ。

解説

Apacheのmod_includeモジュールは、標準的な一連のSSIディレクティブを実装しているが、WTEやLDGWは、その標準に対して拡張を追加している。SSIは今や動的コンテンツを提供するのには時代遅れの技術と見なされており、同じことをしたいなら、通常は、サーブレットやテンプレートエンジン、スクリプト言語を使うことを推奨する。今後、Apacheの実装が拡張されることは、まずないだろう。したがって、WTE/LDGWの拡張を利用したいなら、ApacheのSSI実装で同じ効果を得ようとする代わりに、もっと新しい技術に移行するのに時間をかけた方がよい。

参照

- http://httpd.apache.org/docs/mod/mod_include.html

付録 A
Apache の正規表現

Apache Webサーバの設定ディレクティブには、正規表現を利用することができる(あるいは、利用しなければならない)ものも多い。正規表現は、URLやユーザ名のような文字列が、あるパターンにマッチするかどうかを判断するのに使われている。

正規表現を詳しく解説した情報源はたくさんあるので、この付録を正規表現の使い方のチュートリアルにするつもりはない。その代わりに、Apacheで使用する正規表現に特有の機能について、何ができて、何ができないのかを説明する。非常にたくさんの正規表現パッケージが存在しており、機能に違いはあるが、共通点もいくつかある。例えば、Perlプログラミング言語では、非常に豊富な正規表現を利用することができるが、Apache 2.0以降で利用されている正規表現ライブラリでも、たいていの機能が利用可能になっている。

すでに述べたように、正規表現は、特定の文字列や変数があるパターンにマッチしているかどうかを判断するのに利用する言語である。例えば、特定の文字列がすべて大文字であるかどうか、少なくとも3つの数字を含んでいるかどうか、"monkey"か"Monkey"という単語を含んでいるかなどを調べたいことがある。正規表現は、これを調べるための言語を提供している。最近のプログラミング言語には、さまざまな正規表現ライブラリが存在しており、細かい点で違いはあるが、大部分が共通になってきている。

Apache 1.3では、hsregexという名前の正規表現ライブラリを使っている。これは、Henry Spencerが開発したために、こう名付けられている。hsregexは、egrepで使われている正規表現ライブラリと同じものであることに注意しよう。

Apache 2.0では、さらに豊富な機能を持ったPCRE(Perl Compatible Regular Expressions)という名前の正規表現ライブラリが使われている。この正規表現ライブラリは、Perlプログラミング言語に付属している正規表現エンジンで利用可能な機能の多くを実装しているために、こう名付けられている。この付録では、この2つの実装の相違点をすべて説明するつもりはない。機能に関する限り、hsregexはPCREのサブセットであり、Apache 1.3の正規表現でできることは、2.0の正規表現でもすべて実現可能だ。しかし、その逆は必ずしもそうではない。

かなり大雑把に言うと、正規表現は2種類の文字を使っている。1つは、正確に文字そのものである。例えば、正規表現中に出てきたGは、通常、文字通りGを意味している。もう1つは、特別な意味のある文字だ。例えば、ピリオド(.)は、すべての文字のいずれかに一致するワイルドカード文字である。正規表現は、これら2種類の文字を組み合わせて、(ほとんど)望み通りのパターンを文字列で表現することができる。(例えば、

再帰的なパターンは利用できない）

A.1　どのディレクティブで正規表現が使えるのか？

正規表現が使えるApacheディレクティブには、主に2種類のカテゴリがある。Matchという単語が名前に含まれるディレクティブ（例えば、FilesMatch）は、その引数に正規表現が使えると考えてよい。そして、mod_rewriteモジュールが提供しているディレクティブも、その機能を実現するために正規表現を使うことができる。

mod_rewrite については、5章を参照してほしい。

Match が付くディレクティブは、Match が付かないディレクティブと同じ機能を実装している。例えば、RedirectMatch ディレクティブは、基本的に Redirect ディレクティブと同じ機能を提供している。唯一の違いは、RedirectMatch ディレクティブの最初の引数は、リテラル文字列ではなく正規表現であり、リクエストのURL を正規表現を使って比較するところだ。

A.2　正規表現の基礎

自分で正規表現を書き始めるために、いくつかの基本的な用語を知っておく必要がある。表A-1と表A-2にそれをまとめた。これらは、正規表現について知っておくべき最低限のものであり、身につけても正規表現の専門家になれるわけではないが、正規表現を使う際に直面する問題を解決するのに役立つだろう。

表 A-1　正規表現の基本的な用語

文字	意味
.	任意の文字に一致する。これはワイルドカード文字だ。
+	直前の文字の1回以上の繰り返しにマッチする。例えば、M+は1つ以上のMにマッチする。"+"は1つ以上の任意の文字にマッチする。
*	直前の文字の0回以上の繰り返しにマッチする。例えば、M*は0または複数のMにマッチする。つまり、M や MM、MMM だけではなく、M を全く含まない文字列にもマッチする。
?	直前の文字があってもなくてもマッチする。例えば、正規表現 monkeys?は、monkey と monkeys のいずれかを含む文字列にマッチする。
^	直後の文字が文字列の先頭に現れなければならないことを示す。つまり、正規表現^zimは、先頭がzimで始まる文字列でなければならない。"^"はアンカー（船を固定する「いかり」の意味）と呼ばれている。文字列の先頭に照合を固定するためだ。文字クラス（表A-2を参照）の場合、"^"文字は特別な意味になる。
$	マッチする文字が文字列の末尾に現れなければならないことを示す。つまり、正規表現gif$は、末尾がgifで終わる文字列でなければならない。"$"はアンカーと呼ばれている。文字列の末尾に照合を固定するためだ。
\	直後の文字をエスケープする。つまり、文字の特別な意味を取り除くことを意味する。例えば、正規表現に\というパターンがあれば、\は"."文字の特別な意味を取り除くので、文字"."にマッチする。

表 A-1　正規表現の基本的な用語（続き）

文字	意味
[]	文字クラスを意味する。角括弧に含まれる文字のいずれか1つにマッチする。例えば、[abc]は a か b か c のいずれかにマッチする。一方、[abc]+ は、a と b と c の任意の組み合わせにマッチする。文字クラス中では、"^"文字は通常のアンカーとしての意味ではなく、そのクラスの文字以外の文字を意味する。つまり文字クラス[^abc]は a、b、c 以外の文字にマッチする。 文字クラス中で2つの文字の間にある"-"は、その範囲にあるすべての文字を意味している。例えば、文字クラス[a-q]は小文字の a から q までのすべての文字を意味している。[a-zA-Z]は、すべての小文字と大文字のアルファベットになる。 自分で作る文字クラスに加えて、よく使われる文字のグループを表現するために、あらかじめ定義された特別な文字クラスがたくさんある。定義済みの文字クラスの一覧を表 A-2 に示す。
()	文字の集合を1つのグループにまとめる。これにより、複数の文字を1つの単位として扱うことができる。例えば、"+" や "?" を 1 文字だけでなく文字のグループ全体に適用することができる。例えば、(monkeys)?という正規表現は、monkeys という単語があってもなくてもマッチすることになる。正規表現ライブラリによっては、()によってマッチした文字列の内容を記憶し、後で使うことができるものもある。（Apache HTTP サーバで使われている正規表現ライブラリでは、これが可能だ）

よくある 2 つの間違い

正規表現（"RE" や "regex" とも呼ばれる）パターンを作るときによくはまる落とし穴は、"*" が 0 回以上の出現にマッチするということを見落としてしまうことだ。firstfield:blah:foo:blah:lastfieldのような文字列とマッチさせるつもりで、^firstfield:*lastfield$というパターンを作ったとき、このパターンは、firstfieldlastfieldという文字列にもマッチしてしまう。実際には、"*"ではなく、1 回以上の出現を意味する"+"を使えばよいことが多い。

もう 1 つよく忘れてしまうのは、可変の出現回数にマッチするパターン文字（* や +、?）は、直前の 1 つのパターン表現に対して適用されるということだ。括弧を使って 1 つにまとめていなければ、直前の 1 つの文字を意味することになる。foo+ は foo と fooooooにマッチするが、foofoofoofooにはマッチしない。foofoofoofooにマッチさせるためには、(foo)+ のように文字をグループ化しておく必要がある。

表 A-2　定義済みの正規表現文字クラス

文字クラス	意味
\d	すべての数字（"0" から "9"）
\D	数字以外のすべての文字
\s	垂直タブ以外のすべての空白文字。これはスペース、HT、CR、LF、FFとマッチする。（これは以下で説明する POSIX クラスの[:space:]とは異なることに注意しよう。[:space:]は垂直タブを含むすべての空白文字にマッチする。）
\S	すべての非空白文字（つまり、\s にマッチしないものすべて）

表 A-2　定義済みの正規表現文字クラス（続き）

文字クラス	意味
\w	単語に使う文字。つまり、アンダースコアや序数の値が255以下である文字、すなわちアルファベットや数字を含んでいる。（これは、現在のロケールによって影響を受ける制限や特性がある。詳しくは PCRE のドキュメントを参照すること。）
\W	単語に使う文字以外のすべての文字
[:alnum:]	すべての英数字
[:alpha:]	すべてのアルファベット文字
[:blank:]	スペースか水平タブ
[:cntrl:]	制御文字
[:digit:]	数字
[:graph:]	空白や制御文字でない文字
[:lower:]	小文字
[:print:]	graph と同様だが、スペースとタブも含む
[:punct:]	句読点文字
[:space:]	空白文字。改行やリターンも含む。
[:upper:]	大文字
[:xdigit:]	有効な 16 進数

[]と[::]

POSIX クラス（[::]を使ったもの）は複数の文字マッチングの省略名であり、[:と:]はこの構文の一部になる。クラスとしては、文字クラス表現の中（つまり[と]の間）で使われるだけであり、少しややこしく、紛らわしく見えるだろう。例えば、1 つの 16 進数にマッチさせるには、次のようにすればよい。

```
[[:xdigit:]]
```

1 つ以上の 16 進数にマッチさせるには、次のようにすればよい。

```
[[:xdigit:]]+
```

任意の 16 進数か空白文字にマッチさせるには、次のようにすればよい。

```
[[:xdigit:][:space:]]
```

あるいは、次のようにしてもよい。

```
[[:xdigit:]\s]
```

16 進数以外のすべての文字にマッチさせるには、次のようにすればよい。

```
[^[:xdigit:]]
```

（PCREのPOSIXクラスの実装では、次のようにすることも可能だ。

```
[[:^xdigit:]]
```

しかし、これは拡張構文であり、一般的にサポートされているわけではない。）

A.3 実際の例

これまでに説明したコンセプトを理解するには、正規表現の実例を見ていくのがよいだろう。

URLのリダイレクト

かなり単純な例から始めよう。ここでは、これまでのWebサイトにおけるカスタマサポート部分を、新しいWebサーバを使って処理したい、というシナリオを考える。これまでhttp://www.example.com/support宛てに来ていたリクエストを、今後は新しいサーバhttp://support.example.com宛てに送りたい。通常は、単純なRedirect文で実現することができるが、Webサイトの開発者が不注意で、mod_speling（レシピ5.9を参照）を使っていたとする。このため、サイト中には、http://www.example.com/support と http://www.example.com/Support へのリンクが混在しており、1つだけでなく2つのRedirect文が必要になってしまう。

2つのRedirect文を使う代わりに、次のような1つのRedirectMatchディレクティブで済ませることができる。

```
RedirectMatch ^/[sS]upport/(.*) http://support.example.com/$1
```

角括弧は文字列クラスを表しており、この1行で、sが大文字でも小文字でもリクエストをマッチさせることができる。

引数の前に^が付いていることにも注意しよう。このディレクティブは、指定したパターンが偶然どこかに含まれているようなURLではなく、このパターンで始まるURLにだけ適用される。

正規表現の最後にある括弧を使うと、リクエストされたURIの一部を覚えておき、後で変数$1として参照することができる。ここでは、リクエストされたURIの一部を保持しておき、それをリダイレクト先のURLの一部に使っている。

よくあるスペルミスへの対処

ログファイルを観察すると、supportをsuportとスペルミスしている人が多いことがわかった。RedirectMatchディレクティブを少し変更すると、このミスを簡単に修正することができる。

```
RedirectMatch ^/[sS]upp?ort/(.*) http://support.example.com/$1
```

?を使うことにより、2番目のpがあってもなくてもマッチするようにした。これにより、スペルミスしたリクエストを見つけて、とにかく適切な場所へリダイレクトしている。

A.4　参考情報

正規表現について学ぶときの最もよい情報源は、『詳説正規表現 第3版』（オライリー・ジャパン発行、原書は『Mastering Regular Expressions』、Jeffrey Friedl 著、O'Reilly Media 発行）と、『正規表現デスクトップリファレンス』（オライリー・ジャパン発行、原書『Regular Expression Pocket Reference』、Tony Stubblebind 著、O'Reilly Media発行）である。どちらもいろいろな言語における正規表現を解説しており、正規表現の背景にある一般的な理論についても解説している。

正規表現に関するフリーの情報源としては、Perlのドキュメントから関連するトピックを調べるのがよい。システムにPerlがインストールされていれば、`perldoc perlre`と入力するだけだ。あるいは、http://perldoc.perl.org/perlre.html でオンラインドキュメントにアクセスすることもできる。しかし、Perl の正規表現の用語とApacheで使われているPCREライブラリの用語には、微妙な違い（それほど微妙ではない違いも）があることに注意しよう。この違いについては、http://www.pcre.org/pcre.txt で解説されている。

付録 B
トラブルシューティング

Apache Webサーバは非常に複雑なソフトウェアだ。標準パッケージだけでも30以上の機能モジュールがあり、100以上の設定ディレクティブがある。この複雑な相互作用のために、予想も期待もしない結果が引き起こされる可能性がある。この付録では、いろいろなサポートフォーラムから抜粋した、よく問題として取り上げられる話題をいくつか紹介する。

B.1 トラブルシューティングの方法

エラーログを調査する

Apacheソフトウェアは何か問題に遭遇したときに、その詳細を報告するという、非常に良心的な仕事をしてくれる。この報告はサーバのエラーログに記録される。これは通常、次の場所のいずれかに格納されている。

- /usr/local/apache/logs/error_log
- /var/log/apache/error_log
- /var/log/httpd-error.log
- /var/log/httpd/error_log
- C:\Program Files\Apache Group\error.log
- C:\Program Files\Apache Software Foundation\Apache2.2\logs\error.log

エラーログが置かれる場所は、サーバのインストール方法や設定によって違ってくる。人気のあるパッケージ済みのインストールキット(Red HatやSuSEなど)はそれぞれ独自の場所があり、上に挙げたようにいろいろな場所になる可能性がある。もちろん最終的な場所は、httpd.confファイルの`ErrorLog`ディレクティブを調べればわかる。

Apacheがおかしな動作をしたときには、まず最初に、サーバが何かエラーメッセージを出していないか調べよう。エラーログのメッセージを見ても、問題の原因が何かすぐにはわからないことがある。問題の原因に関連していると思われるメッセージがない場合には、httpd.confファイルの`LogLevel`の設定を変更して、ログレベルを上げるとよいだろう。

```
LogLevel debug
```

debugに設定すると、サーバのエラーメッセージがすべて有効になり、かなり詳細なメッセージが出力される。したがって、問題の原因が見つかれば、設定をwarningかerrorに戻しておくのがよいだろう。

問題の特徴を把握する

問題の原因を突き止めようとするときには、自分自身に質問を問いかけてみよう。「現在どのように動作しているのか、期待している望ましい動作と比べてどこが違うのか？」

質問を問いかけると、自然と次の質問が出てくる。「何が、現在の動作を引き起こしているのか？」

この2つの質問の答えを考えているうちに、「わかった！」という瞬間がくるだろう。少なくとも調査の範囲を狭めることはできるはずだ。

B.2　設定のデバッグ

問題の原因を突き止めるためにサーバの設定を調べるときには、必ず関連するすべてのファイルを調べよう。具体的には、メインのhttpd.confファイルや.htaccessファイルだけでなく、Includeディレクティブに指定されているファイルも調べる必要がある。

サーバ全体の設定ファイルを編集したときには、その変更を有効にするために、必ずサーバを再起動する必要がある。

設定ファイルや.htaccessファイルを編集しても効果がない場合には、そのファイルにでたらめな行を書いてもう一度試してみよう。こうすることで、そのファイルの処理が実際に行われているかどうかを確認することができる。でたらめな行を追加しても.htaccessファイルが無視されているようであれば、AllowOverride Noneディレクティブのスコープ内である可能性がある。

B.3　スクリプトヘッダが正しく終了していない場合のデバッグ

CGIスクリプトを動かしていると、「Premature end of script headers（スクリプトヘッダが正しく終了していない）」というエラーメッセージをうんざりするほど目にするだろう。このときブラウザのウィンドウには、何もないページか、Internal Server Errorのページが表示される。このエラーメッセージを引き起こす原因は、いろいろと考えられる。よくある原因を以下に挙げたが、これらに限られるわけではない。

- CGIスクリプトが何も出力していない場合や、必要なヘッダを付ける前にコンテンツを出力した場合。あるいは、ヘッダとコンテンツの間に必須の空行を出力し忘れた場合
- スクリプトでエラーが発生し、期待される出力の代わりにエラーメッセージを出力した場合
- suexecを使っていて、suexecの制約のいずれかが破られた場合

問題がエラー状態なのか、CGIのレスポンスが不適切なのかを調べて判断するには、コマンドラインからスクリプトを対話形式で実行し、CGIのルールに従ってコンテンツが出力されているか確認すればよい。

suexecを使っているなら、suexecのログファイルをチェックして、セキュリティの制約が破られていないかを調べればよい。

suexec を使っているかどうかを調べるには、次のコマンドを実行する。

```
% httpd -l
Compiled-in modules:
  http_core.c
  mod_so.c
suexec: disabled; invalid wrapper /var/www/apache/bin/suexec
```

このようなメッセージが得られれば、suexecは使用不可になっており、suexecが問題の原因であるという可能性は無視してよい。

suexecが使用可能になっていれば、suexecのログファイルを調べて、より詳しい情報を入手する必要がある。suexec のログファイルは、次のコマンドで見つけることができる。

```
# suexec -V
 -D DOC_ROOT="/usr/local/apache/htdocs"
 -D GID_MIN=100
 -D HTTPD_USER="www"
 -D LOG_EXEC="/usr/local/apache/logs/suexec.log"
 -D SAFE_PATH="/usr/local/bin:/usr/bin:/bin"
 -D UID_MIN=100
 -D USERDIR_SUFFIX="public_html"
```

ここで重要な箇所は、-D LOG_EXEC="/usr/local/apache/logs/suexec.log" だ。この行には、suexecがエラーを記録している場所が示されている。

CGI と suexec についてもっと詳しく知りたければ、以下を調べてほしい。

- CGI 仕様―― http://www.ietf.org/rfc/rfc3875
- レシピ 8.13
- suexec の manpage

B.4 Windows でよくある問題

Windows 環境には、Unix 系環境では発生しない、Windows 特有の問題がある。

ホスト名を決定できない

ApacheをDOSウィンドウから起動しようとすると、次のようなメッセージを受け取ることがある。“Cannot determine hostname. Use ServerName directive to set it manually.（ホスト名を決定できません。ServerNameディレクティブを使って手動でホスト名を設定してください）”

Apacheの設定でシステムのホスト名を明示的に指定していない場合、Apacheはホスト名を自分で決定しよ

うとする。このメッセージは、その決定に失敗したことを示している。解決策は非常に簡単だ。conf\httpd.conf ファイルを開いて、文字列 ServerName を探し、次のようなコメントが外れたディレクティブがあることを確認する。

```
ServerName localhost
```

または

```
ServerName www.foo.com
```

この情報が間違っていれば、これを正しく修正する。あるいは、まだ値が設定されていなければ、正しく追加する。

また、Windows システムで DNS が有効になっていることを確認しよう。コントロールパネルの［**ネットワーク接続**］で、該当のネットワークを選び、TCP/IP のプロパティ設定を確認する。

DNSが有効になっていて、ServerNameディレクティブに正しいホスト名を設定していることを確認したら、サーバを再起動する。

Windows 上に WS2_32.DLL が見つからない

Windows 95 で Apache を起動しようとすると、"Unable To Locate WS2_32.DLL..."（WS2_32.DLL が見つからない...）というメッセージが表示されることがある。このファイルは、Apache を正しく動かすのに必要だ。

バージョン 1.3.9 より以前は、Windows 版の Apache は Winsock 1.1 を使っていた。バージョン 1.3.9 からは、ApacheはWinsock 2の機能（特に、WSADuplicateSocket()）を使い始めた。WS2_32.DLLは、Winsock 2 API を実装しているライブラリである。Winsock 2 は、Windows NT 4.0 と Windows 98 には最初から含まれている。しかし、Windows 95 の初期リリースのいくつかには、Winsock 2 が含まれていなかった。

これを修正するには、Winsock 2 をインストールすればよい。Winsock 2 は現在のところ、http://support.microsoft.com/kb/182108/から入手することができる。これをインストールしてからサーバを再起動すると、問題が解消されてるはずだ。

WSADuplicateSocket エラーを修復する

Windows 上で Apache の起動に失敗し、エラーログに次のようなメッセージが残っていることがある。

```
[crit] (10045) The attempted operation is not supported for the type of object
    referenced: Parent: WSADuplicateSocket failed for socket ###
```

これは、システムのネットワークソフトウェアにファイアウォール製品が組み込まれているが、ファイアウォールが本来のネットワーク呼び出しの機能を十分提供していないことを示している。

この問題を解消するには、Apacheサーバと同じマシンで動いているファイアウォール製品を再設定するか、無効にするか、削除する必要がある。

この問題は、Aventail Connect のような VPN（Virtual Private Networking）クライアントと一緒に Apache を

動かしているときにも発生することがある。Aventail Connect は LSP（Layered Service Provider）として、Winsock 2 API と Windows 本来の Winsock 2 実装の間にクサビのように入り込む。Aventail Connect のクサビは WSADuplicateSocket を実装しておらず、これが失敗の原因になる。

このクサビは Aventail Connect を停止するまで取り外すことができない。いったん問題が発生すると、Aventail Connect を明示的に停止するか、マシンを再起動するまで、問題は続くことになる。

もう 1 つの解決策（テストしていないが）は、apache.exe あるいは httpd.exe を Aventail Connect の除外リストに追加することだ。他のファイアウォールプログラムでも、正しく設定しておかないと、Apacheが同様の影響を受けるおそれがある。ブロックするポートのリストから Apache サーバのポート（通常、ポート 80）を除外しておく必要がある。設定方法については、それぞれのファイアウォールプログラムのドキュメントを参照すること。

システムエラー 1067 に対処する

Windows 上で Apache を起動すると、“System error 1067 has occurred. The process terminated unexpectedly.（システムエラー1067が発生し、プロセスが異常終了しました）”というメッセージを受け取ることがある。これだけだと情報不足であるが、Web サーバが何らかの理由で正しく起動できなかったことを示している。

どんなエラーであっても、解決するための最初のステップは、Apache エラーログをチェックすることだ。何も役に立つものが見つからなければ、Windowsのアプリケーションイベントログをチェックして、Apache が起動しなかった原因を調べよう。それでも、何も手がかりがなければ、次を試してみるとよい（以下は Apache 2.2 の場合）。

```
D:\>c:
C:\>cd "\Program Files\Apache Software Foundation\Apache2.2\bin"
C:\Program Files\Apache Software Foundation\Apache2.2\bin>httpd.exe
```

（プロンプトが戻らなければ、Ctrl-C を押して、Apache を終了する。）

こうすると、Apache はサービスとしてではなく、対話形式で動作する。システムエラー 1067 の警告ボックスの背後に隠れていたエラーメッセージが、すべて画面上に表示されるはずだ。

B.5　ビルド時のエラーを修正する

_inet シンボル

BIND-8をインストールしている場合に、このシンボルに関するエラーが発生するなら、インクルードファイルとライブラリの矛盾が原因だ。BIND-8はインクルードファイルとライブラリをそれぞれ/usr/local/include/と/usr/local/lib/にインストールする。一方、システムに付属しているリゾルバは、おそらく/usr/include/と /usr/lib/ にインストールされている。

システムが、/usr/include/より前に/usr/local/include/にあるヘッダファイルを探すようになっているにも関わらず、新しいリゾルバライブラリが使われていない場合には、ヘッダファイルとライブラリのバージョンが矛盾してしまう。これを解決するには、システムに付属しているインクルードファイルとライブラリを

使うか、または、新しいインクルードファイルとライブラリを使う必要がある。

Apache 2.0以降を使っている場合、または、Apache 1.3でAPACI（Apache Autoconf-style Interface）ビルドスクリプトを使用している場合には、次のように、./configureのコマンドラインでライブラリ検索リストを変更することができる。

```
% LIBS=-lbind ./configure ...
```

Apache 1.3以前を使っており、直接、設定ファイルを編集してビルドのパラメータを変更しているなら、このファイルのEXTRA_LDFLAGSの行に-lbindを追加するだけでよい。

このように、ビルド時の設定を適切に変更すれば、正しいライブラリを使ってApacheをビルドできるはずだ。

Apache 1.2以前のバージョンでは、代わりに設定ファイルにEXTRA_LFLAGSを使う。

BIND 8.1.1では、bindのライブラリとファイルがデフォルトで、/usr/local/bind以下にインストールされるので、この問題は発生しない。bindのリゾルバを使いたいなら、次のように設定、実行する必要がある。

- Apache 1.3でAPACIを使用する場合、または、2.0以降の場合、次のように実行する。

```
% CFLAGS=-I/usr/local/bin/include \
> LDFLAGS=/usr/local/bind/lib LIBS=-lbind \
> ./configure ...
```

- Apache 1.2または1.3で設定を直接編集する場合、ファイルに次のような追加、変更をする。

```
EXTRA_CFLAGS=-I/usr/local/bind/include
EXTRA_LDFLAGS=-L/usr/local/bind/lib
EXTRA_LIBS=-lbind
```

B.6　SSIを利用する

SSIが使えない場合には、Options Includesが有効になっているかどうか確認する。次に、XBitHackが有効になっているか、あるいは、使用するファイルタイプに適切なAddHandlerやAddOutputFilterディレクティブが設定されているかどうかを確認する。

レシピ8.9で解説したように、SSIを有効にするにはたくさんの方法がある。ページをロードしたとき、SSIディレクティブが解析されずにHTMLに現れてしまうのであれば、そのドキュメントに対してSSIの実行が有効になっていないことを示している。

サーバがSSIディレクティブを解析するのが困難な場合には、代わりに"An error occurred while processing this directive（このディレクティブの処理中にエラーが発生しました）"というメッセージがレスポンスの該

当箇所に入ることになる。このような状況が発生したときには、問題の原因がサーバのエラーログに記録されるはずだ。

B.7 "Not Found" エラーになるリライトのデバッグ

RewriteRule ディレクティブを使うと、いつも 404 Not Found エラーページになってしまうのであれば、PT（PassThrough）フラグを RewriteRule の行に追加する必要があるかもしれない。このフラグがない場合、Apache は、Alias の設定など、その後のいろいろな処理を実行しないためだ。これが問題の原因であるかを確認するためには、mod_rewriteのログレベルを9まで上げて、RewriteRuleに関連するログエントリに、"prefixed with document_root" という内容があることを調べればよい。

```
RewriteLog logs/rewrite-log
RewriteLogLevel 9

% tail logs/rewrite_log
ip-address - - [date] [reqid] (2) prefixed with document_root to
/usr/local/apache/htdocs/robots.text
ip-address - - [date] [reqid] (1) go-ahead with
/usr/local/apache/htdocs/robots.text [OK]
```

この調査が終わったら、RewriteLogディレクティブを無効にするか、ログレベルを下げておくことを忘れないようにしよう。さもないと、ディスクの空き容量がまるで残雪のようにどんどん消えていってしまうだろう。

RewriteRule ディレクティブに PT フラグがない場合、mod_rewrite は、このリライトが、そのリクエストに対してサーバがやるべき最後の URL 操作だと考えてしまう。mod_rewrite ディレクティブは、リクエスト処理の初期段階で適用されるため、Alias や ScriptAlias などの URL 操作が適用されないおそれがある。PT フラグを指定しておくと、mod_rewrite は処理を省略せずに、通常通り続けようとする。

B.8 .htaccess ファイルの効果がない

AllowOverrideディレクティブに適切な値が設定されているか確認する。次に、.htaccessファイルがちゃんと解析されているかどうかを確認するために、.htaccessに次のような行を追加する。そして、ブラウザ上に、サーバのエラーページが表示されるかどうか確認する。

```
Garbage Goes Here
```

.htaccessファイルは、メインサーバの設定ファイルの設定内容を上書きする。これは望まない結果になることが多いため、.htaccessファイルの使用を無効にしていることも多い。この結果、.htaccessファイルが無視されてしまうことがある。

AllowOverrideディレクティブを使うと、.htaccessファイルを有効にすることができる。さらに、.htaccess

ファイルで利用可能なディレクティブについて、そのカテゴリをリストとして指定することもできる。例えば、.htaccessファイルに、認証関連のディレクティブを設定できるようにしたいなら、メインサーバの設定ファイルには、次のような行を追加しておく必要がある。

```
AllowOverride AuthConfig
```

AllowOverride Allとすると、.htaccessファイルにはどんなディレクティブでも設定できるようになる。一方、AllowOverride Noneとすると、「.htaccess ファイルを無視してください」という意味になる。

.htaccessファイルが無視されてしまうのは、単に、設定ファイルで.htaccessを無視するように指示しているだけであることが多い。

.htaccessファイルにでたらめな行を書くと、ブラウザ上には、サーバのエラーメッセージが表示されるはずだ。これにより、Apacheが実際に、その.htaccessファイルの内容を見ているかどうかを確認することができる。このようなメッセージが表示されなければ、.htaccessファイルが完全に無視されているという確かな証拠になる。

B.9　アドレスがすでに使われている

Apache サーバを起動しようとしたときに、次のようなエラーメッセージを受け取ることがある。

```
[Thu May 15 01:23:40 2003] [crit] (98)Address already in use: make_sock: could not
    bind to port 80
```

これは、以下の 3 つのいずれかが起こっていると考えられる。

- root 以外のユーザでサーバを起動しようとしている。root ユーザになって、もう一度試してみればよい。
- すでにポート 80 を使った別のプロセス（おそらく別の Apache サーバ）が動いている。netstat を実行するかプロセスリストを見て、ポート 80 を使っていると思われるプロセスを終了すればよい。
- 設定ファイルにおいて、複数のListenディレクティブで、同じポート番号が指定されている。問題となっている重複しているディレクティブを探して、それを削除すればよい。

最初の状況の場合、Apache を起動するためには、root ユーザになる必要がある。伝統的に、1025 未満のポートをバインドできるのは、rootユーザに限られている。Apacheは通常、ポート80で起動するので、root権限が必要になる。

2番目の状況は、もう少し複雑だ。Apacheを終了した後も、子プロセスが終了せずに動き続けることがある。このような状況が発生する原因はたくさんある。多くの場合、root でログインして、kill や kill -9 を使って強制的にこのプロセスを終了すればよい。このプロセスが動いてポートを塞いでいる限り、同じポートをバインドしようとしている他のプロセスは、起動に失敗してしまう。

3番目の状況の場合、すでに最初のListenディレクティブがポート80をバインドしているのに、2番目のListenディレクティブが同じポートをバインドしようとするために発生する。単に、Listenディレクティブの1つを削除すれば、この問題を解決することができる。

索　引

記号・数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

X

あ行

か行

さ行

た行

な行

は行

ま行

や行

ら行

わ行

●著者紹介

Ken Coar（ケン・コール）

Apache Software Foundation のメンバー。『Apache Server for Dummies』（Wiley）、『Apache Server Unleashed』（Sams）の共著者である。Apacheプロジェクトに送られる電子メール処理の責任者で、そのメーリングリストに接した経験がこの本を執筆する基盤になった。

Rich Bowen（リッチ・ボーエン）

Apache Software Foundationのメンバーで、Apache Web Serverのためのドキュメンテーションの仕事をしている。ケンタッキー州のレキシントンに暮らしており、暇な時間はGeoCaching（GPSとインターネットを使ったハイテク宝探しゲーム）をして楽しんでいる。『Apache Administrator's Handbook』（Sams）の共著者。

RichもしくはDrBacchus（IRCのハンドルネーム）の名前で#apacheにおいて多くの時間をすごしている。Webサイトは、http://www.drbacchus.com/journal/

●訳者紹介

笹井 崇司（ささい たかし）

1996年、大阪大学大学院 工学研究科 情報システム工学専攻修士課程修了。現在、電機メーカーにソフトウェアエンジニアとして勤務。主にネットワーク関連のソフトウェア開発、モバイル機器の商品開発に従事。Webサイトは、http://www.textdrop.net/

● カバーの説明

表紙の動物はムース。ムースは北アメリカ、ヨーロッパ、ロシアの森林地帯に生息する、シカ科の中で最大の種である。その中でも最も体が大きいのはAlces alces gigas（ヘラジカ）で、アラスカ全土で見られる。ムースは各地に偏在しており、人と対立関係になることもある。

ムースは繁殖能力が高く、生息範囲には多くのムースが暮らしている。アラスカでは老木の伐採や山火事により、ムースのエサとなる森林が新しく形成されているが、それでもエサを求めて飛行場に立ち入ったり、市街地を歩いたりして車や列車にぶつかることもある。

ムースはアラスカらしい風景であるため、国の経済にも貢献している。ハイウェイでエサを食むムースは観光客にとって絶好のシャッターチャンスである。居住者やハンターは毎年6000～8000頭（約350万トン）を収穫する。人が生息環境の管理について学習し、またハンティングや狼、熊による捕食など、ムースの個体数に作用する要素を習得したので、アラスカにおけるムースの未来は明るいと言える。

Apacheクックブック 第2版
—Webサーバ管理者のためのレシピ集

2008年9月25日　初版第1刷発行
2010年3月16日　初版第3刷発行

著　　者　Ken Coar（ケン・コール）、Rich Bowen（リッチ・ボーエン）
訳　　者　笹井 崇司（ささい たかし）
発 行 人　ティム・オライリー
印　　刷　株式会社ルナテック
製　　本　株式会社越後堂製本
発 行 所　株式会社オライリー・ジャパン
〒160-0002　東京都新宿区坂町26番地27　インテリジェントプラザビル1F
Tel　(03)3356-5227
Fax　(03)3356-5263
電子メール　japan@oreilly.co.jp
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)3233-3440

Printed in Japan(ISBN978-4-87311-381-4)
乱本、落丁の際はお取り替えいたします。