第1章 Perl

DB接続，セッション管理，デザインとロジックの分離…モジュールで解決！

基礎編：Perl実行環境とモジュールの基礎知識

Tokyo Perl Mongers 小山浩之●OYAMA Hiroyuki●oyama@cpan.org

Perl 最もポピュラーな選択肢

Perlはその使い勝手の良さから，Webアプリケーションを開発する最もポピュラーなプログラミング言語の1つに挙げられています．Perlのプログラマは，無料で利用できる豊富なコンポーネントによって，

- Webブラウザからメールを閲覧
- ネットワーク越しにファイルをダウンロード・加工
- ネットワークプロトコルを直接操作

といったことが容易に実現できます．

Perlは「インターネットの粘着テープ」と呼ばれ，まさに粘着テープのようにあらゆるものを結び付けるために利用されています．変化の激しいインターネットの世界において，粘着テープのように簡単にコンポーネントを結び付け，短期間でWebアプリケーションをリリースできるため，今日にいたるまでPerlが選ばれ続けていると考えられます．

また，利用者と利用実績の多さも重要なポイントです．ニュースグループやメーリングリストではさまざまな情報が取り交わされ，出版される書籍の数からも利用者の多さをうかがい知ることができます．

本章では，このPerlによるWebアプリケーションの実行環境と，開発に際して浮かび上がる課題へのモジュールによる回答について述べていきます．

本特集で使用しているスクリプトは，すべて弊誌Webサイトhttp://www.gihyo.co.jp/wdpress/からダウンロードできます．ご活用ください．

Webアプリケーションの実行環境

PerlによるWebアプリケーションを実行する環境には，代表的なものとして以下の3つを挙げることができます．

- CGI
- FastCGI※1
- mod_perl※2

CGI（Common Gateway Interface）

Webアプリケーションを開発する環境としては言語を問わず，最もポピュラーなものです．ブラウザのリクエストを受け取ったhttpdがプログラムを起動して，その出力結果をコンテンツとして送信する方法です（図1）．

※1）参考URL：FastCGI Home http://www.fastcgi.com/
※2）参考URL：Apache/Perl Integration Project http://perl.apache.org/
参照書籍：『Apache拡張ガイド（上）（下）』／Lincoln Stein，Doug MacEachem著／田辺茂也監訳／田和勝訳／オライリー・ジャパン／ISBN4-87311-018-1

手軽にWebアプリケーションの開発を行うことができる反面，プロセス起動・スクリプトのコンパイルに要するオーバーヘッドからパフォーマンスを確保しにくく，トラフィックに比例して多くのリソースを消費してしまう性質を持っています．

FastCGI

httpdと共にプログラムを起動・常駐させることにより，CGIで問題になるプロセスの起動と，スクリプトのコンパイルに要するオーバーヘッドとリソース消費を解消する方法です（図2）．

Javaサーブレットのように魅力的なアーキテクチャですが，常駐プロセスとして動作するため，スクリプトを修正した場合はhttpdの再起動による一時的なサービス停止が発生します．

mod_perl

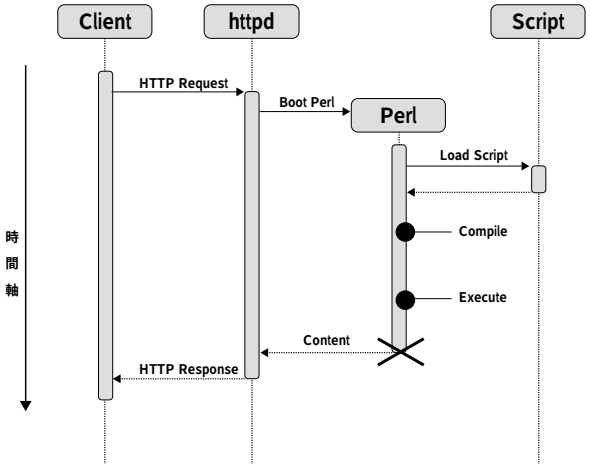
Apacheモジュールの1つで，httpdにPerlを組み込むことでhttpdが行っているURI（Uniform Resource Identifiers）からファイル名への変換，認証，コンテンツの出力，ロギングなどの処理をPerlでカスタマイズすることができます．

Apacheをカスタマイズする手段としてはもちろんですが，httpdによって直接スクリプトのコンパイルと実行が行われるためにプロセス起動のオーバーヘッドはなく，スクリプトのコンパイル結果をキャッシングするために，CGIで問題になっていた実行に要するオーバーヘッドのほとんどを解消することができます（図3）．

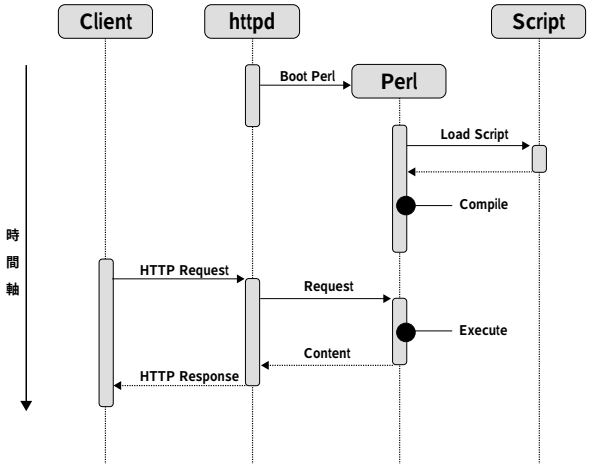
mod_perlの実行環境の1つApache::Registryは，スクリプトのコンパイル結果とグローバル変数をキャッシングして実行するため，CGI用のスクリプトをほぼ無修正のままに大幅な高速化が実現できます．ただし変数のキャッシングを行うため，グローバル変数に依存するスクリプトはうまく動作しなくなる場合があります．またhttpdにPerlを組み込むため，httpdのサイズが通常よりも大きくなる性質を持っています．

Apache::Registryのキャッシングによる動作の変化を過度に心配し，導入をためらわれている方もいらっしゃると思いますが，グローバル変数に依存せずスコ

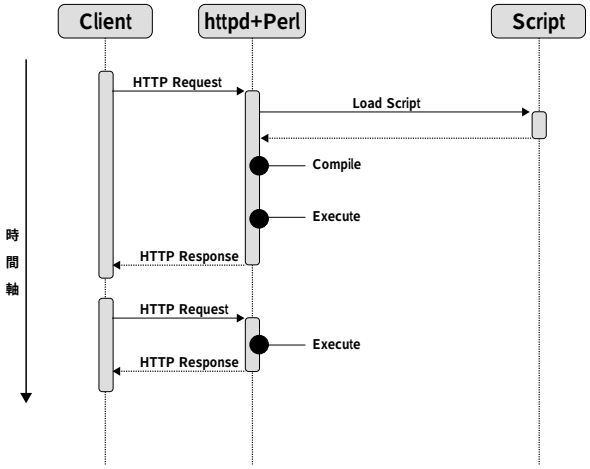
●図1　CGIの動作



●図2　FastCGIの動作



●図3　mod_perlの動作



特集3 Perlによる ハイパフォーマンスWebアプリケーションの開発
Powered by mod_perl, Apache & MySQL

ープを意識したきれいなコードを使用している限りは問題にならないので，安心してください．

ここで上げた3つの他にも，MicrosoftのIIS（Internet Information Server）へmod_perl同様にPerlを組み込むものや，ZOPE※3のように，アプリケーションサーバとしての環境を提供するMason※4など，さまざまな環境が存在し利用されています．

Webアプリケーション開発における課題

Webアプリケーションを開発する際には，解決しなければならない以下の課題があります．

- フォームの処理
- リレーショナルデータベースの操作
- セッション管理
- デザインとロジックの切り分け

Perlではこれらの複雑な問題を簡単に解決するモジュールが利用できます．各々の問題に対応するモジュールを順に説明していきます．

CGI.pm

CGI.pm※5はフォームの入力値の取得やHTTP Cookieの操作など，CGIにまつわる繁雑な処理を簡略化し，統一した見通しの良いコードを実現します．

◎問題

流通しているCGI用のスクリプトを見ると，フォームの値を取得するためのデコード処理をリスト1のように記述しているものをよく見かけます．

●リスト1

```
my $input;
my %form;
read STDIN, $input, $ENV{'CONTENT_LENGTH'};
foreach my $key_and_value (split /&/, $input) {
   my ($key, $value) = split /=/,$key_and_value;
   $form{$key} = $value;
}
```

これはPOSTしたフォームの値をハッシュに格納するコードです．ひとつ大きな問題点として，このコードは同じ名前のパラメータが複数入力された場合は正しくハンドリングできなくなります．実際に<SELECT>タグで複数の項目を選択させる場面などでは，この問題が浮上してきます．また，HTTPのGETメソッドで渡された場合と，POSTメソッドで渡された場合の2種類のデコード処理を記述する必要があります．

これらの変換処理はフォームの値を受け取るほとんどの場面で行われる処理ですが，パラメータを受け取るための前処理でしかなく，コードの空間やコーディングの手間として注力すべき処理とは言えません．

HTTP Cookieの操作やコンテンツの有効期限を設定する処理でも同様の問題が発生します．

◎解法

PerlのcoreモジュールであるCGI.pmを利用することで，ブラウザから送信されたすべての情報の取得と操作，出力する情報の操作にまつわる繁雑な処理をカプセル化し，コードを簡略化することができます．

◎結果

定型的にもかかわらず繁雑な処理を記述する必要がなくなり，同時にコードも簡潔になり見通しが良くなります．また，繁雑な処理手順を思い出し記述し直す必要もなくなります．モジュールの提供する機能とその呼び出し方を記憶するだけで良いのです．

◎利用方法

CGI.pmを利用するためには，まずインスタンスを生成します．

```
use CGI;
my $query = CGI->new;
```

クラスメソッドnew()はCGIモジュールのインスタンスを返します．フォームの値やHTTP Cookieは，こ

※3）参考URL：Zope.org http://www.zope.org/，Zope Japan http://www.zope.ne.jp/
※4）参考URL：Mason HQ http://www.masonhq.com/
※5）参考URL：CGI.pm manual page http://www.perldoc.com/perl5.6/lib/CGI.html
CGI.pmマニュアル日本語訳 http://member.nifty.ne.jp/hippo2000/perltips/Cgi.htm

の生成したインスタンス（この場合は$query）を通して操作します．

フォームの入力値は，フォームの属性名を指定するだけで参照できます．

```
$query->param('NAME');
```

CGIモジュールが提供する機能と機能を呼び出すメソッドで主に利用するものを，表1にまとめます．

◎サンプルコード

● 入力元となるHTMLフォームの定義

入力元となるHTMLフォームの定義は，リスト2を参照してください．

● フォームを受け取るスクリプトの定義

フォームを受け取るスクリプトの定義は，リスト3を参照してください．

●リスト2　入力元となるHTMLフォームの定義

```
<HTML>
<BODY>
    <FORM method="POST" action="sample.cgi">
        <INPUT type="TEXT" name="COMMENT">
        <SELECT name="LANGUAGE">
            <OPTION value="Perl">Perl
            <OPTION value="Python">Python
            <OPTION value="Ruby">Ruby
        </SELECT>
    </FORM>
</BODY>
</HTML>
```

入力された値は自動的に解析され，$queryの属性として格納されます．param()メソッドで参照したいパラメータ名を指定することで，入力された値を取得することができます．

またGETメソッドのURIで指定されたパラメータも同様の手順で取り出すことができます．

```
http://hostname/sample.cgi?COMMENT=cool
print $query->param('COMMENT');  # it's "cool"
```

92ページに掲載の“Perl mini-Cookbook”ではCGI.pmを使用したファイルアップロードに関しての情報がまとめられていますので，そちらも合わせて参照することをお勧めします．

DBI.pm

DBI.pmはPerlからリレーショナルデータベースを操作するための共通APIを提供し，異なる種類のリレーショナルデータベースも同じAPIで操作することができます（表2）．

●リスト3　フォームを受け取るスクリプトの定義

```
#!/usr/bin/perl
use CGI;

my $query = CGI->new;
print $query->param('COMMENT');
print $query->param('LANGUAGE');
__END__
```

●表1　CGI.pm簡易API表

API	内容	例
`$obj = CGI->new;`	CGI.pmのインスタンスを生成する．	
`$obj->param;` `$obj->param(NAME);`	受け取ったHTMLフォームの入力値を参照する．値の名称を指定するとその値を返す．無指定で呼び出した場合は，全フォームの属性名のリストを返す．同名の属性を複数受け取った場合は，値のリストを返す．	`my @name_list = $obj->param;` `my $value = $obj->param($name);`

`$obj->header;`
`$obj->header(CONTENT_TYPE);`
`$obj->header(LIST);`
	HTTPレスポンスヘッダを生成する．無指定の場合はContent-Type: text/htmlが出力される．	`$obj->header(`
		`  -type  => 'image/gif',`
		`  -expires => '+1m',`
		`);`

`$obj->redirect(URL);`	リダイレクションヘッダを生成する．	

`$obj->cookie(LIST)`	HTTP Cookieを生成する．Cookieの名称・値・有効期限などを指定したハッシュを引数として受け取る．生成したHTTP Cookieはheader( )メソッドの-cookieパラメータに渡すことでブラウザに送信することができる．	`my $cookie = $obj->cookie(`
		`  -name  => 'session_id',`
		`  -value  => 'id_strings',`
		`  -expires => '+1d',`
		`);`
		`print $obj->header(-cookie => $cookie);`

特集3 Perlによる ハイパフォーマンスWebアプリケーションの開発

Powered by mod_perl, Apache & MySQL

◎問題

重要なデータの保持や大量のデータを対象に操作・検索する場合，リレーショナルデータベースを利用することで，複雑な処理やパフォーマンスに関する多くの問題を解決できます．

リレーショナルデータベースを操作するSQLのインタフェースに関してはほぼ標準化されていますが，ネイティブAPIにはプロダクトごとに大きく差異があります．

それらの差異を埋めるためWindowsの世界ではODBCが，Javaの世界ではJDBCが共通APIとして広く利用されています．

◎解法

DBI.pmを使用することでPerlからさまざまなリレーショナルデータベースを操作することができます．

インタフェースを定義するDBIと実際にデータベースを操作するドライバのDBDから構成されます（図4，表3）．各種データベース用にDBDが開発されており，最新のドライバリストはDBIのWebサイト※6で参照することができます．

◎結果

Perlからリレーショナルデータベースの操作を可能にし，かつ異なるデータベースでも一貫したAPIで操作することができます．つまりテスト用にMicrosoft Access用に作成したコードをわずかな修正で，Oracle用のコードとして利用することが可能になります．

一般にリレーショナルデータベースへの接続処理に要するオーバーヘッドは大きくなります．CGIでは毎

●表2 DBI簡易API表

API	内容	例

`$handle = DBI->connect($data_source, $username, $passwd);`	Databaseへ接続しデータベースハンドルを生成する．第4引数にハッシュのリファレンスでパラメータを指定できる．RaiseError：問題発生時に例外を発生する．default off．AutoCommit：データベースへの操作を自動的にcommitする．default on．トランザクションを使用する場合はoffにする．	`my $handle = DBI->connect($data_source, $username, $passwd, {`
		`  RaiseError => 1, AutoCommit => 0`
		`});`

`$state = $handle->prepare(SQL);`	SQLをセットする．	
`$state->execute;` `$state->execute(LIST);`	SQLを実行する．	
`$state = $handle->do(SQL);`	prepare( )とexecute( )を一度に実行する．	
`$record = $state->fetchrow_arrayref;`	検索結果1レコードを配列のリファレンスで受け取る．	

`$handle->commit;`	データベースへの変更を反映する．使用するデータベースがトランザクションをサポートして，かつAutoCommitがOff(0)の場合に使用できる．	`my $handle = DBI->connect($data_source, $username, $passwd, {`
		`  RaiseError => 1, AutoCommit => 0,`
		`});`
		`eval {`
		`  $handle->do(q{`
		`  DELETE FROM table`
		`  WHERE status = 'DONE'`
		`  });`
		`};`
		`if ($@) {`
		`  $handle->rollback;`
		`  # clean up code`
		`} else {`
		`  $handle->commit;`
		`}`
		`$handle->disconnect;`

`$handle->rollback;`	データベースへの変更を中止し破棄する．使用するデータベースがトランザクションをサポートして，かつAutoCommitがOff(0)の場合に使用できる．	

※6）参考URL：DBI - A Database Interface Module for perl5 http://www.symbolstone.org/technology/perl/DBI/
DBIマニュアル日本語訳 http://member.nifty.ne.jp/hippo2000/perltips/dbimemo.htm

回接続処理を行う必要があり，実行効率が低下してしまう恐れがありますが，mod_perlの環境下ではApache::DBIを組み込むことで，スクリプトをまったく変更せずにデータベース接続を再利用し，この問題を回避することができます．

◎利用方法

DBIを使用してデータベースを操作する場合には，まずデータベースに接続します．

```
use DBI;
my $handle = DBI->connect($data_source, $username,
$password);
```

connect( )メソッドの戻り値として返されるデータベースハンドル（この場合は“$handle”）を通してデータベースの操作を行います．

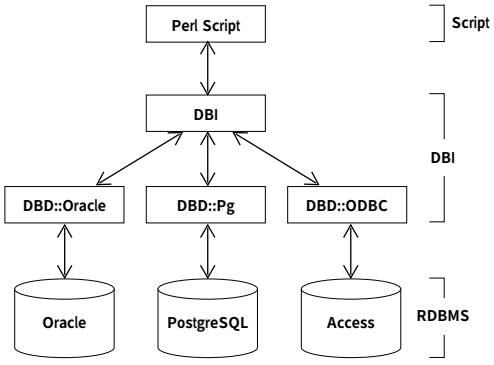
第1引数の“$data_source”では，ドライバ名とデータベース名からなる文字列で接続するデータベースを指定します．

```
dbi:DriverName:database_name
dbi:DriverName:database_name@hostname:port
dbi:DriverName:database_name;host=hostname;port=port
```

ドライバ名の後に続く文字列は使用するデータベースによって異なりますので，DBDのマニュアルを参照してください．

次にデータベースハンドルのprepare( )メソッドで，実行するSQLをセットします．

●図4　DBIのアーキテクチャ




```
my $state = $handle->prepare(q{
    SELECT column1, column2 FROM table
});
```

SQLの検索条件や値を動的に変更したい場合は，置き換える部分を“?”で指定します．この“?”の部分をプレースホルダと呼びます．

```
my $state = $handle->prepare(q{
   SELECT name, email, age FROM table WHERE age == ?
});
```

prepare( )メソッドから返されるステートメントハンドル（この場合は$state）を使用して検索の実行や結果の取得を行うことができます．

SQLの実行にはexecute( )を使用します．SQLにプレースホルダが含まれる場合は，置き換えたい値のリストを指定します．

```
$state->execute();
$state->execute($age);
```

動的にSQLを変化させる場合に毎回prepare( )でSQLをセットし使用していると，データベースエンジンがSQLを解釈しなおすオーバーヘッドが増え，パフォーマンスを落とす原因になりますので意識的にプレースホルダを使用するようにします．

検索結果の取得にはfetchrow_arrayref( )メソッドを使用します．

```
while (my $record = $state->fetchrow_arrayref) {
    print 'Name: ',  $record->[0];
    print 'Email: ', $record->[1];
    print 'Age: ',   $record->[2];
}
```

●表3　DBIドライバー覧

ADO	Informix	Solid
ASAny	Ingres	Sybase
Adabas	InterBase	Unify
Altera	ODBC	XBase
CSV	Oracle	mysql
DB2	Pg	mSQL
Empress	RAM	
Illustra	SearchServer	

fetchrow_arrayref()メソッドは検索結果を1つずつ取り出し，配列のリファレンスを返します．すべての要素を取り出し終えるとundefを返します．

データベースへの操作が終了したら，データベースハンドルのdisconnect()メソッドを呼び出して切断を行います．

```
$handle->disconnect();
```

以上が，DBIで検索処理を行う場合の例です．挿入や更新を行うINSERTやUPDATEなどのSQLを発行する場合は値を取り出す記述がなくなるだけで，ほぼ同じ手順で操作を行います．

◎サンプルコード

●使用するデータベース

MySQL※7のデータベース“customer”に含まれるテーブル“language”に，データを追加・検索する例を示します．テーブル“language”には“name”・“version”・“url”のカラムが含まれます（図5）．

このデータベースにはユーザID“user”，パスワード“password”で接続します．

●データの挿入

標準入力からタブ区切りの文字列を受取り，データベースに挿入するスクリプトをリスト4に示します．

●データの参照

テーブルの内容を表示するスクリプトをリスト5に示します．

Apache::Session.pm

Apache::Session.pm※8を使用するとセッション管理機能が利用できます（表4）．

◎問題

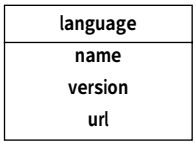
Webアプリケーションにおいて最も頭の痛い問題がセッションの維持・管理です．受け取った情報をページを跨いで引き継いだり，過去の利用状況を参照してページに反映させる場合にセッション管理機能が必要になります．

セッション管理を実現する方法にはデータをHTTP Cookieに保存したり，フォームのhiddenタグに埋め込んだり，ファイルに保存したりとさまざまな方法がありますが，どの方法も実装には大きな手間が生じます．

◎解法

Apache::Sessionのサブクラスを利用することでこ

●図5 データベースの構造



●リスト4 データ挿入スクリプト

```
#!/usr/bin/perl
use DBI;

my $handle = DBI->connect(
    'dbi:mysql:customer',
    'user', 'password'
);
my $state = $handle->prepare(q{
    INSERT INTO language (name, version, url) VALUES (?, ?, ?)
});
while (my $line = <>) {
    chomp $line;
    $state->execute(split /\t/, $line);
}
$handle->disconnect;
__END__
```

※7）参考URL：MySQL http://www.mysql.com/，Soft Agency（MySQL日本総代理店） http://www.SoftAgency.co.jp/
日本MySQLユーザ会 http://www.mysql.gr.jp/
参考書籍：『MySQL & mSQL』／Randy jay Yarger，George Reese，Tim King著／ソフトエージェンシー監訳／高見禎成，寺田美穂子訳／オライリー・ジャパン／ISBN4-87311-011-4

※8）参照URL：Apache::Session.pm manual page http://www.perldoc.com/cpan/Apache/Session.html
Apache;Sessionマニュアル日本語訳 http://member.nifty.ne.jp/hippo2000/perltips/apache/Session.htm

れらの繁雑な処理を記述することなく，セッション管理機能を利用することができます．

Apache::Sessionはセッション管理を実装するためのフレームワークで，セッション情報をMySQLやPostgreSQLなどのリレーショナルデータベースに格納するApache::Session::MySQL，Apache::Session::Postgresや，テキストファイルに格納するApache::Session::Fileなどさまざまな実装をバンドルしています．

Apache::Sessionはスカラー値をはじめとしてリファレンスによる複雑なデータ構造やオブジェクトなど，データの型を問わずに格納・復元ができます．

また名称はApache::*ですがApache専用のモジュールというわけではなく，他のhttpdでも問題なく動作します．

◎結果

セッション管理に関する繁雑な記述を簡略化できます．また，さまざまな管理方法を選択することができます．サーバサイドに情報を蓄積することで，利用者との間で不必要な情報の受け渡しが不要になります．

セッション情報をリレーショナルデータベースに格納することで，複数のWebサーバ間でセッション情報を安全に共有することができます．これはクラスタリングしたWebサーバでWebアプリケーションを実行する場合に有効です．

◎利用方法

Apache::Sessionはtiedハッシュによりセッション情報の操作と格納を行います．このtiedハッシュを操作すると，自動的にデータベースやファイルの操作が行われます．

まずセッション管理を始めるために新しいセッションを開始します．

```
use Apache::Session::DB_File;
my %session;
tie %session, 'Apache::Session::DB_File',
    $session_id, {
        FileName      => '/var/sessions.db',
        LockDirectory => '/var/lock/sessions',
};
```

ハッシュ%sessionにセッション情報を結び付けます．新しくセッションを開始する場合，セッションIDはundefを指定します．

この例ではセッション情報をdbmに格納するApache::Session::DB_Fileを使用していますので，セッション情報を格納するdbmファイルへのパス名と，排他制御用のロックファイルを作成するディレクトリを指定します．リレーショナルデータベースなど独自

●リスト5　テーブルの内容表示のスクリプト

```
#!/usr/bin/perl
use DBI;
my $handle = DBI->connect(
       'dbi:mysql:customer',
       'user', 'password'
);
my $state = $handle->prepare(q{
       SELECT name, version, url FROM language
});
$state->execute;
while (my $record = $state->fetchrow_arrayref) {
       printf "%s, %s, %s\n",
        $record->[0],    # name
        $record->[1],    # version
        $record->[3];    # url
}
$handle->disconnect;
__END__
```

●表4　Apache::Session簡易API表

API	内容	例

tie %hash, CLASS_NAME;
tie %hash, CLASS_NAME, $session_id, \%option;
	%hashを指定したクラスに結合してセッション管理の対象にします．セッションIDにundefを指定すると新しいセッションを開始します．	use Apache::Session::MySQL;
		tie %session, 'Apache::Session::MySQL', $session_id, {
		DataSource => 'dbi:mysql:sessions',
		};

$hash{_session_id};	現在のセッションIDを参照します．	
tied(%hash)->delete;	セッション情報を破棄します．	
untie %hash;	セッションを終了します．	

にロック機構を備えているものを利用する場合は，この記述は必要ありません．

後は通常のハッシュと同様に値の設定と参照を行うことができます．

```
$session{name} = 'value';
print $session{name};
```

現在のセッションIDは“_session_id”から取得できます．

```
print $session{_session_id};
```

この値をHTTP Cookieやパス情報に埋め込むことでページを跨いだ値の参照が可能になります．

セッション管理を終了する場合はtiedハッシュをuntie()します．

```
untie %session;
```

再びセッション情報を取得する場合は，セッションIDを指定してtiedハッシュを作成します．

Apache::Sessionはリファレンスやオブジェクトなどデータ型を問わずに格納と復元が可能ですが，リファレンスを格納する場合，Apache::Sessionは起点となるハッシュの値を変更したときのみデータを格納するということを意識する必要があります．

たとえばリストのリファレンスを格納している場合，リファレンス先のリストに変更を行った場合でも「リファレンス自身」は何も変化していません．

このためセッション情報を格納する場合，起点のハッシュにタイムスタンプを格納することが推奨されています．

```
$session{last_access} = time;
untie %session;
```

◎サンプルコード

● アクセスカウンタ

Apache::Sessionを使用したアクセスカウンタの実装を示します．セッション情報の管理にはdbm，セッションIDの保持にはHTTP Cookieを使用します（リスト6）．

HTML::Template.pm

HTML::Template.pm※9を使用するとテンプレートを使用して，柔軟な動的HTML生成が可能になります．置換処理に関連する記述をカプセル化，デザインとプログラムロジックの分離が実現します．

◎問題

Webアプリケーションの開発はロジックを記述するプログラマと，利用者に対するGUIをデザインするデザイナの協調作業によって進行します．コードに直接HTMLを記述するとデザインとロジックが強く結びつき，変更に対して迅速な対応が難しくなってしまいます．

PHP※10やASP（Active Server Pages）のようにHTMLへコードを埋め込む手法をとったとしても，同様の問題が起こり得ます．

◎解法

HTML::Templateモジュールを使用することでデザインとロジックの分離が実現します．

HTML::Templateは要求を処理するスクリプトとは別にHTMLの雛型を定義するテンプレートファイルを使用します．動作時にテンプレートを読み込みデータを流し込むことで動的にHTMLを出力します．

◎結果

プログラムロジックとデザインの分離が実現し，それぞれを個別に修正することができるようになります．これによりプログラマとデザイナの作業範囲が明確になります．

プログラムロジックからHTMLタグや置換処理を排除でき，見通しの良い保守しやすいコードが記述できるようになります．

※9）参照URL：HTML::Template.pm - SourceForge: CVS Repository http://sourceforge.net/cvs/?group_id=1075
HTML::Templateマニュアル日本語訳 http://member.nifty.ne.jp/hippo2000/perltips/html/template.htm
※10）参照URL：PHP: Hypertext Preprocessor http://www.php.net/，日本PHPユーザ会 http://www.php.gr.jp/

●リスト6　アクセスカウンタ

```
#!/usr/bin/perl

use Apache::Session::DB_File;
use CGI;

my $query = CGI->new;
my $session_id = $query->cookie(-name => 'session_id');
my %session
tie %session, 'Apache::Session::DB_File',
    $session_id, {
       FileName      => '/var/sessions.db',
       LockDirectory => '/var/lock/sessions',
};

++$session{access};
my $html = <<__HTML_BODY__;
<HTML>
<BODY>
    Your Access: $session{access}
</BODY>
</HTML>
__HTML_BODY__

my $state_cookie = $query->cookie(
    -name  => 'session_id',
    -value => $session{_session_id},
);
print $query->header(-cookie => $state_cookie);
print $html;
__END__
```

●リスト7　HTMLのテンプレートファイル定義

```
<HTML>
<BODY>
    <TMPL_VAR name=LIST_NAME>
    <TMPL_LOOP name=LANGUAGE>
        Name:  <TMPL_VAR name=NAME>
        URL:   <TMPL_VAR name=URL>
        <HR>
    </TMPL_LOOP>
</BODY>
</HTML>
```

●リスト8　配列のリファレンス設定

```
$html->param('LANGUAGE' => [
    {
        NAME => 'Perl',
        URL  => 'http://www.perl.com/',
    },
    {
        NAME => 'Python',
        URL  => 'http://www.python.org/',
    },
    {
        NAME => 'Ruby',
        URL  => 'http://www.ruby-lang.org/',
    },
]);
```

コード記述の視点が「HTMLタグ文字列の生成」から「値の整形と代入」に変わることで，処理の内用が明確な見通しの良いコードを記述できます．またコード修正によるデザインへの影響，デザインの修正によるコードへの影響を最小限の範囲に閉じ込めることができます．

◎利用方法

HTMLのテンプレートファイルを定義します（リスト7）．

<TMPL_VAR>と<TMPL_LOOP>はHTML::Template用の特殊タグです．

new()メソッドの引き数でテンプレートファイルへのパスを指定し，HTML::Templateのインスタンスを生成します．

```
use HTML::Template;
    my $html = HTML::Template->new(
    filename => '/path/to/template.html'
);
```

このインスタンス$htmlを通して値の設定や，置換結果の出力を行います．

<TMPL_VAR>に対応する値を設定するためにparam()メソッドを使用します．テンプレートのパラ

特集3 Perlによる ハイパフォーマンスWebアプリケーションの開発
Powered by mod_perl, Apache & MySQL

メータ名と，設定する値を引数に指定します．

```
$html->param('LIST_NAME' => 'スクリプト言語');
```

HTML::Templateはテーブルなどループした要素の出力をサポートしています．<TMPL_LOOP>タグの範囲がループ要素として扱い，ネストしたループも記述可能です．

<TMPL_LOOP>にはハッシュのリファレンスを出力順に列挙した配列のリファレンスを設定することで出力できます（リスト8）．

output( )メソッドでテンプレートへの処理結果を出力します．

```
print $html->output;
```

この出力はリスト9のようになります．

◎サンプルコード

●テンプレート“directory.html”の定義

●リスト9 テンプレートへの出力結果

```
<HTML>
<BODY>
    スクリプト言語
        Name:  Perl
        URL:   http://www.perl.com/
        <HR>
        Name:  Python
        URL:   http://www.python.org/
        <HR>
        Name:  Ruby
        URL:   http://www.ruby-lang.org/
        <HR>
    </TMPL_LOOP>
</BODY>
</HTML>
```

指定したディレクトリのファイル一覧をHTMLで出力するテンプレートをリスト10に示します．

●ディレクトリー覧を出力するスクリプトの定義

引数で指定したディレクトリの一覧をHTMLで出力するスクリプトをリスト11に示します．

まとめ

本章ではPerlの実行環境と，Webアプリケーションの実装において問題に上がるフォームの処理，リレーショナルデータベースの操作，セッション管理，デザインとロジックの分離に対応する解法をまとめました．mod_perlやFastCGIを使用すれば高速な実行環

●リスト11 指定ディレクトリー覧をHTMLで出力するスクリプト

```
#!/usr/bin/perl
use HTML::Template;

my $directory = $ARGV[0] || './';
my $html = HTML::Template->new(
    filename => 'directory.html'
);

my $file_list = [];
opendir DIR, $directory;
while (my $filename = readdir DIR) {
    push @{$file_list}, {
        NAME => $filename,
        SIZE => -s $filename,
    };
}
$html->param(FILELIST => $file_list);
$html->param(PATH => $directory);
print $html->output;
__END__
```

●リスト10 指定ディレクトリー覧をHTMLで出力するテンプレート

```
<HTML>
    <HEAD><TITLE><TMPL_VAR name=PATH></TITLE></HEAD>
<BODY>
    <TABLE>
        <TR>
           <TH>Name</TH>
           <TH>Size</TH>
        </TR>
        <TMPL_LOOP name=FILELIST>
        <TR>
           <TD><A href="<TMPL_VAR name=NAME>"><TMPL_VAR name=NAME></A></TD>
           <TD><TMPL_VAR name=SIZE></TD>
        </TR>
        </TMPL_LOOP>
    </TABLE>
</BODY>
</HTML>
```

境が手に入り，開発中に上がる課題のほとんどには「～のモジュールを利用する」というシンプルかつ確実な回答が用意されています．

加速度的に発展するWebアプリケーションの世界では，短期間に実装が完了することはもちろん，常に変化・発展するサービスに柔軟に対応する機能拡張性や保守性が要求されます．高速な実行環境と記述の簡単さ，無料で使用できる豊富なコンポーネント，利用者の多さ，オブジェクト指向のサポート等々．Perlの持つこれらの特性は，変化の激しいWebアプリケーションの世界で要求される要素と見事に合致しているのです．◆

mod_perlのインストール

●パッケージのダウンロード

mod_perlを使用するには，CPANからパッケージをダウンロードしてインストールする必要があります．http://www.cpan.org/もしくは最寄りのCPANミラーサイトにFTPで接続してパッケージをダウンロードします．

```
> ftp www.cpan.org
User: anonymous
331 Anonymous login ok, send your complete e-
mail address as password.
Password: your@email.address
ftp> cd /CPAN/modules/by-module/Apache
ftp> ls mod_perl-*.tar.gz
mod_perl-1.21.tar.gz
mod_perl-1.22.tar.gz
mod_perl-1.23.tar.gz
mod_perl-1.24.tar.gz
ftp>
```

ディレクトリ/CPAN/modules/by-module/Apacheへ移動し，mod_perl-*.tar.gzというファイルをダウンロードします．この場合，最新バージョンのmod_perl-1.24.tar.gzをダウンロードします（Apache 1.3.14を使用する場合はmod_perl-1.24_01.tar.gzをダウンロードします．/CPAN/ authors/id/D/DO/DOUGM/mod_perl-1.24_01.tar.gz）．

```
ftp> get mod_perl-1.24.tar.gz
ftp> quit
```

●パッケージの展開

ダウンロードが完了したら，Apacheのソースツリーと同じ階層のディレクトリにパッケージを展開します．

```
> cd /usr/local/src
> gzip -cd mod_perl-1.24.tar.gz | tar xvf -
mod_perl-1.24/
mod_perl-1.24/t/
mod_perl-1.24/t/docs/
...
> cd mod_perl-1.24
```

●Makefile.PLの実行

Makefile.PLファイルを実行するとMakefileの生成と，Apacheのconfigureスクリプトの実行によりhttpdにリンクするApacheモジュールリストにmod_perlが追加されます．

```
> perl Makefile.PL
Configure mod_perl with ../apache_1.3.14/src
? [y] y
Shall I build httpd in ../apache_1.3.14/src for
you? [y] y
...
```

Makefile.PLは，まずApacheのソースツリーの位置を確認を求めてくるので“y”と答えます．ソースツリーが見つからない場合はパスを入力するように求めくるので，Apacheのソースツリーへのフルパスを入力します．次にhttpdをビルドするか確認を求めてくるので“y”と答えます．

●makeとmake test

Makefile.PLが完了したら，makeを実行してPerlモジュールのビルドとmod_perlを組み込んだhttpdのビルド

を行います．

```
> make
```

makeが完了したらテストを実行します．テストの実行にはLWP::UserAgentモジュールが必要です．テストを必ず行う必要はありませんが，実行しておいた方が良いでしょう．

```
> make test
```

●インストール

テストが成功したらPerlモジュールのインストールを行います．

```
> su -
Password: xxxxxxxxx
# make install
```

この時点では新しいhttpdはインストールされていません．Apacheのソースツリーへ移動して，新しくmod_perlをリンクしたhttpdをコピーしインストールを完了します．

```
# cd ../apache_1.3.14/src
# cp -p httpd /usr/local/apache/bin/
```

●Apacheのconfigureスクリプトオプションの指定

Apacheのconfigureスクリプトをオプションを指定して実行する必要がある場合は，Makefile.PLのAPACI_ARGSオプションを指定します．

```
> perl Makefile.PL APACI_ARGS="--enable-module=rewrite,--disable-module=userdir"
```

APACI_ARGSで指定した文字列はApacheのconfigureスクリプトに渡されます．

●プラットフォームごとの注意事項

mod_perlのパッケージに含まれるINSTALLドキュメントには，インストールにおけるプラットフォームごとの注意事項がまとめられています．使用するプラットフォームについての記述に目を通しておくことをお勧めします．

mod_perlの設定

mod_perlはhttpd.confで有効にしなければ機能しません．ここではCGIの高速化のためにmod_perlを使用する設定について述べます．

●Apache::Registryの使用

Apache::RegistryはCGI用のPerlスクリプトをApacheのモジュールとして読み込み実行します．Apache::Registryは1度コンパイルしたスクリプトをキャッシングます．スクリプトを修正した場合このキャッシュは破棄され，スクリプトを再度読み込みます．

スクリプトエイリアス/cgi-bin/をApache::Registry用のディレクトリに修正する設定を行ってみましょう．

デフォルトのhttpd.confには以下のようなScriptAliasディレクティブと，そのディレクトリに対する設定の記述があります．

```
<Directory "/usr/local/apache/cgi-bin">
AllowOverride None
Options None
Order allow,deny
Allow from all
</Directory>
```

これを以下のように修正します．

```
<Directory "/usr/local/apache/cgi-bin">
SetHandler perl-script
PerlHandler Apache::Registry
AllowOverride None
Options +ExecCGI
Order allow,deny
Allow from all
PerlSendHeader Off
</Directory>
```

SetHandler，PerlHandler，PerlSendHeaderディレクティブを追加し，Optionsディレクティブに修正を加えました．これで/usr/local/apache/cgi-bin に設置したPerlスクリプトはApache::Resigtryの環境下で実行されます．URLはhttp://hostname/cgi-bin/scriptnameです．

1度目のアクセスはスクリプトの読み込みとコンパイルが行われるために，若干レスポンスが遅れますが，2回目以降のアクセスからは高速なレスポンスが帰ってくるはずです．

●Apache::DBIの追加

他のPerlモジュールを追加する場合は，PerlModuleディレクティブでモジュールを指定します．DBIによるデータベース接続を自動的に再利用するモジュールApache::DBIを追加してみましょう．

```
<Directory "/usr/local/apache/cgi-bin">
SetHandler perl-script
PerlHandler Apache::Registry
PerlModule  Apache::DBI
AllowOverride None
Options +ExecCGI
Order allow,deny
Allow from all
PerlSendHeader Off
</Directory>
```

httpd.confの修正後，httpdをリスタートすると設定が反映されます．接続処理に時間がかかるデータベースを使用する場合は，これだけで大幅な高速化が見込めます．

●Apache::PerlRunの使用

Apache::Registryはスクリプトのコンパイル結果と変数のキャッシングにより高速なレスポンスを実現しますが，グローバル変数に依存したスクリプトのほとんどは正しく動作しなくなります．

Apache::PerlRunはApache::Registryとは異なり，コンパイル結果とグローバル変数のキャッシングは行わず，CGIと同様にリクエストの度にスクリプトをロードしコンパイル・実行します．Perlインタープリタの起動に要するオーバーヘッドは発生しないため，Apache::Registry程ではありませんがパフォーマンスアップが見込めます．

Apache::PerlRunの設定はApache::Registryとほぼ同じで，以下のように記述します．

```
<Directory "/usr/local/apache/cgi-bin">
SetHandler perl-script
PerlHandler Apache::PerlRun
AllowOverride None
Options +ExecCGI
Order allow,deny
Allow from all
PerlSendHeader Off
</Directory>
```

CGI用に記述された大量のスクリプトも，Apache::PerlRunを使用すれば高速化が可能なのです．またmod_perl_traps.podにmod_perlを利用するにあたって注意すべきポイントがまとめられていますので，目を通しておくことをお勧めします．

CPANモジュールのインストール

●パッケージのダウンロード

標準添付しているcoreモジュール以外の物を使用するときには，CPANからパッケージをダウンロードしてインストールする必要があります．

```
http://www.cpan.org/
```

上記URLもしくは最寄のCPANミラーサイトにFTPで接続してパッケージをダウンロードします．

```
http://www.cpan.org/modules/by-module/Apache/mod_perl-1.24.tar.gz
http://www.cpan.org/modules/by-module/DBI/DBI-1.14.tar.gz
http://www.cpan.org/modules/by-module/Apache/Apache-Session-1.53.tar.gz
http://www.cpan.org/modules/by-module/HTML/HTML-Template-2.0.tar.gz
```

特集3 Perlによる ハイパフォーマンスWebアプリケーションの開発

Powered by mod_perl, Apache & MySQL

●インストール

それぞれのパッケージをダウンロードした後，パッケージを解凍，Makefile.PLを実行し，makeします．

```
> gzip -cd package.tar.gz | tar xvf -
> cd package/
> perl Makefile.PL
> make
> make test
```

makeとmake testが成功したらrootになってインストールします．

```
> su -
Password: xxxxxxxxxx
# make install
```

●インストールに際しての注意

パッケージによっては別途ファイルが必要な物やインストール手順が異なる物もありますので，パッケージに含まれるINSTALLやREADMEなどのドキュメントに必ず目を通してからインストール作業を行います．

以上でモジュールのインストールは完了です．また，root権限がない場合や，ホームディレクトリにインストールしたい場合はMakefile.PLのPREFIXオプションでインストール先ディレクトリを指定してからmakeします．

```
> perl Makefile.PL PREFIX=/path/to/install/directory
> make
> make test
> make install
```

●マニュアルの参照

モジュールのマニュアルはperldocで参照することができます．

```
> perldoc ModuleName
```

また，http://www.perldoc.com/ などオンラインでマニュアルを参照することも可能です．