第2章

mod_perl & モジュールと Apache，MySQL で作る

実践編：ショッピングカートの実装

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

本章では，第1章で紹介したmod_perlとモジュールを活用してショッピングカートを実装し，応答速度などを検証します．

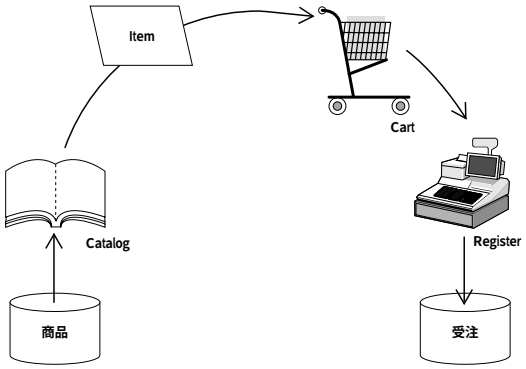
要求の定義

ショッピングカートに要求する機能を以下に示します．

- カタログからアイテムの一覧を表示
- カートにアイテムを追加
- カートからアイテムを削除
- レジスタでカートのアイテムを注文

このアプリケーションは商品を表す「アイテム」，商品情報を持つ「カタログ」，選択した商品を保持する「カート」，注文を受け付ける「レジスタ」の4つの要素から構成されます（図1）．

●図1　ショッピングカートの概要



このうちデータベースの操作を要する要素は以下の2点です．

- カタログ（商品データ）
- レジスタ（注文データ）

本章ではショッピングカートの利用者に関係する基本動作のみを実装します．注文を受けた後の処理や商品データのメンテナンス機能は割愛します．

データベースの選定

DBIを使用することで，オープンソースから商用のものまで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．既存のリレーショナルデータベースを利用する場合は別ですが，新たにWebアプリケーションを開発する場合，どのプロダクトを選定すれば良いのでしょうか．そのためにはまず開発するWebアプリケーションの特性を見極める必要があります．

今回作成するショッピングカートのアプリケーションは，シンプルなテーブル構造でかつデータベースへの要求の大半は検索処理で占められます．またテーブル構造がシンプル故に，更新処理もシンプルになります．そのため重視するポイントは「検索処理の早さ」に絞ることができ，処理をアトミックに扱うトランザクションの優先度は低くなります．これはニュースサ

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

イトやコミュニケーションサイトなどにもあてはまる条件と言えます．

今回はマルチスレッドによる軽量で高速な検索と，堅牢なデータ保持を提供するMySQLを選定しました．

複雑なテーブル構造を扱う必要がある場合は，PostgreSQL※1などが候補にあがるでしょうし，保証されたサポートサービスが必要な場合は，サポートサービス業者が存在するプロダクトか商用のプロダクトが選定対象になります．商用・非商用を問わず，プロダクトの選定に際してはその実績やサポート体制はもちろんのこと，採用しているアーキテクチャと設計理念を理解し，比較することで自然と適切なプロダクトを選定できるはずです．

データベース設計上のポイント

リレーショナルデータベースにおけるテーブル設計の手法については，書籍はもちろん，使用するリレーショナルデータベースのドキュメントで言及されているはずですのでここでは述べません．データベースの設計において，正規化を行い冗長な情報や重複した情報を排除することは基本中の基本と言えます．しかし，特定の状況ではパフォーマンスを上げるためにデータベースを冗長化させる必要が生まれます．

データベースの冗長化

頻繁に発生する要求に複雑なテーブルの結合が必要なケースでは，ディスクI/Oや計算量の増加が問題になる可能性があります．そのため，検索しやすいように必要な情報を結合したテーブルをあらかじめ作成しておくことで，この問題を回避できる場合があります．ただしデータベースに冗長な要素を増やすことは，データの更新処理に関する手続きが増え，そのパフォーマンスは低下します．また冗長化の方法によっては一時的に古い情報を検索してしまう可能性があります．この方法はデータのメンテナンスに関する手間よりも，検索性能が重要な場合に行います．

また，Webアプリケーションでは商品カテゴリの一覧や選択肢を定義したテーブルをアプリケーションから参照し，表示するケースが多く見られます．この定義用のテーブルが多数ある場合には，データベースエンジンのテーブルキャッシュの消費量も増加します．大量のトラフィックを受けるWebアプリケーションでこれら複数のテーブルを頻繁にアクセスする必要がある場合は，定義情報を単一のテーブルにまとめると，良いパフォーマンスが得ることができる場合があります．消費するテーブルキャッシュを抑えキャッシュのヒット率を上げることでパフォーマンスアップを狙います．

基本部品の開発

先に挙げた「アイテム」「カタログ」「カート」「レジスタ」の4要素を実装したモジュールを作成します．この段階では基本動作のみを実装し，GUIに関する機能は実装しません．

アイテム

商品を表すアイテムは名前・金額・写真・特徴・寸法の属性を持ちます．このアイテムのデータ構造を保持するモジュールItem.pm（**リスト1**）を作成します．

クラスメソッドnew()は新しく商品のデータ構造をインスタンス化します．new()の引数で指定したハッシュは_initialize()メソッドに渡し，解析した結果を属性に割り当てます．その他id，name，price，photo，feature，sizeは，それぞれの属性にアクセスするための属性アクセスメソッドです．

Item.pmは以下のように使用します．まず商品の属性をハッシュで与えてデータ構造を作成します．

```
use Item;
my $item = Item->new(
    id      => 1,
    name    => '商品名',
    price   => 1200,
```

※1）参考URL：PostgreSQL http://www.postgresql.org/，日本PostgreSQLユーザ会 http://www.jp.postgresql.org/

```
    photo   => 'http://hostname/image/image01.gif',
    feature => '製品の特徴',
    size    => '20x30x10'
);
```

new( )の戻り値は指定した属性のアイテム（商品）を表すオブジェクトです．各々の属性を参照する場合は，引き数を指定せずに属性アクセスメソッドを呼び出します．

```
print $item->name;
print $item->photo;
```

この例では商品名と画像へのURLを表示します．属性値を変更する場合は値を引き数に属性アクセスメソッドを呼び出します．

```
print $item->price, "\n";  # \1,200
$item->price(960);         # 2割引!
print $item->price, "\n";  #  \960
```

ショッピングカートはこの「アイテム」を受け渡しながら動作します．

カタログ

カタログはデータベースからアイテムを取り出します．全アイテムを取り出す処理と，指定したアイテムのみを取り出す処理をサポートします．カタログにアイテムを要求すると，データベースを検索し，検索結果を元にアイテムを生成します．

カタログを表すモジュールCatalog.pm（リスト2）を実装します．

Catalog.pmはDBIを使用して検索を行い，Item.pmを使用してアイテムのオブジェクトを返します．クラスメソッドnew( )はカタログを使用するために必要な

●リスト1　Item.pm

```
package Item;
use strict;

sub new
{
    my $class = shift;
    my %args = @_;
    my $self = bless {
        name    => undef,
        price   => 0,
        photo   => undef,
        feature => undef,
        size    => undef,
    }, $class;
    $self->_initialize(%args);
    $self;
}

sub _initialize
{
    my $self = shift;
    my %args = @_;
    foreach my $name (keys %args) {
        if    ($name eq 'id')      { $self->id($args{$name}) }
        elsif ($name eq 'name')    { $self->name($args{$name}) }
        elsif ($name eq 'price')   { $self->price($args{$name}) }
        elsif ($name eq 'photo')   { $self->photo($args{$name}) }
        elsif ($name eq 'feature') { $self->feature($args{$name}) }
        elsif ($name eq 'size')    { $self->size($args{$name}) }
    }
    $self;
}

sub id
{
    my $self = shift;
    if (@_) { $self->{id} = shift }
    $self->{id};
}

sub name
{
    my $self = shift;
    if (@_) { $self->{name} = shift }
    $self->{name};
}

sub price
{
    my $self = shift;
    if (@_) { $self->{price} = shift }
    $self->{price};
}

sub photo
{
    my $self = shift;
    if (@_) { $self->{photo} = shift }
    $self->{photo};
}

sub feature
{
    my $self = shift;
    if (@_) { $self->{feature} = shift }
    $self->{feature};
}

sub size
{
    my $self = shift;
    if (@_) { $self->{size} = shift }
    $self->{size};
}

1;
__END__
```

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

初期化を行います．new( )ではアイテムを識別する“アイテムID”を受け取ります．アイテムIDが指定されていない場合はすべてのアイテムを検索して出力します．アイテムIDを表す$item_idは_initialize( )メソッドに渡し，検索対象の決定に使用します．

_initialize( )メソッドでは

●データベースへの接続

●SQLのセット・実行

を行います．

handle( )メソッドはデータベースハンドルを返します．まだデータベースに接続していない場合は，接続を行い新しいデータベースハンドルを返します（リスト2-❷）．

DBIのステートメントハンドルは属性アクセスメソ

●リスト2 Catalog.pm

```
package Catalog;
use DBI;
use Item;
use strict;

sub new
{
    my $class = shift;
    my $item_id = shift;
    my $self = bless {
        handle => undef,
        state  => undef,
    }, $class;
    $self->_initialize($item_id);
    $self;
}


sub _initialize
{
    my $self = shift;
    my $item_id = shift;
    my $handle = $self->handle;
    eval {
        if (defined $item_id) {
            $self->state(
                $handle->prepare(q{
                    SELECT id, name, price, photo, feature, size
                      FROM catalog
                     WHERE id = ?
                })
            );
            $self->state->execute($item_id);
        }
        else {
            $self->state(
                $handle->prepare(q{
                    SELECT id, name, price, photo, feature, size
                      FROM catalog
                })
            );
            $self->state->execute;
        }
    };
    if ($@) { die $@ }
    $self;
}
```

```
sub get_item
{
    my $self = shift;
    my $state = $self->state;
    return undef unless my $record = $state->fetchrow_arrayref;
    return Item->new(
        id      => $record->[0],
        name    => $record->[1],
        price   => $record->[2],
        photo   => $record->[3],
        feature => $record->[4],
        size    => $record->[5],
    );
}
```

❶

```
sub close
{
    my $self = shift;
    $self->state->finish      if $self->state;
    $self->handle->disconnect if $self->handle;
}
```

```
sub handle
{
    my $self = shift;
    unless ($self->{handle}) {
        eval {
            $self->{handle} = DBI->connect(
                'dbi:mysql:webapp', 'user', 'password', {
                RaiseError => 1
            }) or die;
        };
        if ($@) { die "connect: $@" }
    }
    $self->{handle};
}
```

❷

```
sub state
{
    my $self = shift;
    if (@_) { $self->{state} = shift }
    $self->{state};
}


sub DESTROY
{
    my $self = shift;
    $self->close;
}

1;
__END__
```

ッドのstate( )に保持します．

get_item( )メソッドはデータベースに問い合わせた結果を元に，新しいアイテムのオブジェクトを返すイテレータです（リスト2-❶）．すべてのアイテムを取り出し終えるとundefを返します．

Catalog.pmを利用する場合はクラスメソッドnew( )で新しいカタログを生成して

```
use Catalog;
my $catalog = Catalog->new;
```

そこからget_item( )メソッドでアイテムを1つずつ取り出します．

```
while (my $item = $catalog->get_item) {
    push @items, $item;
}
```

使い終わったカタログは行儀よく閉じておくと，データベースとの接続も解除されます．

```
$catalog->close;
```

これで商品の一覧を提供する基本部品ができ上がりました．この時点でCatalog.pmとItem.pmの関係と使用方法さえ認識していれば，データベースから商品一覧を取り出し必要な情報を掲示することができます．

例として商品名と金額の一覧をタブ区切りで出力する例を示します（リスト3）．

検索対象のデータベースとテーブルを作成します．今回はMySQLを使用するので，mysqladminでデータベース“webapp”を作成し，mysqlクライアントでテーブル“catalog”を作成します（リスト4）．データベース“webapp”にはユーザ名“user”，パスワード“password”で接続することにします．

●リスト3　商品名と金額の一覧をタブ区切りで出力

```
#!/usr/bin/perl
use Catalog;
my $catalog = Catalog->new;
while (my $item = $catalog->get_item) {
printf "%s\t%s\n", $item->name, $item->price;
}
__END__
```

カート

カートは選択した商品を保持・削除し，その一覧を提供する機能を持ちます．カートの実装にはセッション管理機能が必要です．Apache::Sessionを組み込み，Cart.pm（リスト5）を作成します．

Cart.pmは与えられたセッションIDを元に選択したアイテムを保持します．

クラスメソッドnew( )は新しいカートを生成し，セッション情報を元にカートのアイテムを復元します．引数として指定したセッションIDは_initialize( )メソッドに渡した後，Apache::Sessionによるセッション情報の復元に使用します．新しいセッションの場合は未定義のセッションIDを指定します（リスト5-❶）．

Apache::Session::DB_Fileを使用して，dbmにセッション情報を格納します．使用するdbmのファイル名とロックファイルを作成するディレクトリを指定します．セッション情報を結び付けるハッシュにはsession( )メソッドでアクセスします（リスト5-❺）．

add_item( )メソッドは引き数で指定したアイテムをカートに追加します（リスト5-❷）．

delete_item( )メソッドは引き数で指定した番号のアイテムをカートから削除します（リスト5-❸）．

Cart.pmではアイテムを保持するために配列を使用しており，この配列にアイテムを追加することでカートに追加，配列から削除することでアイテムの削除を行います．

カートのアイテムの操作を図2に示します．

item( )メソッドはカートが保持しているアイテムをリストで返します（リスト5-❹）．

●リスト4　webapp，catalogの作成

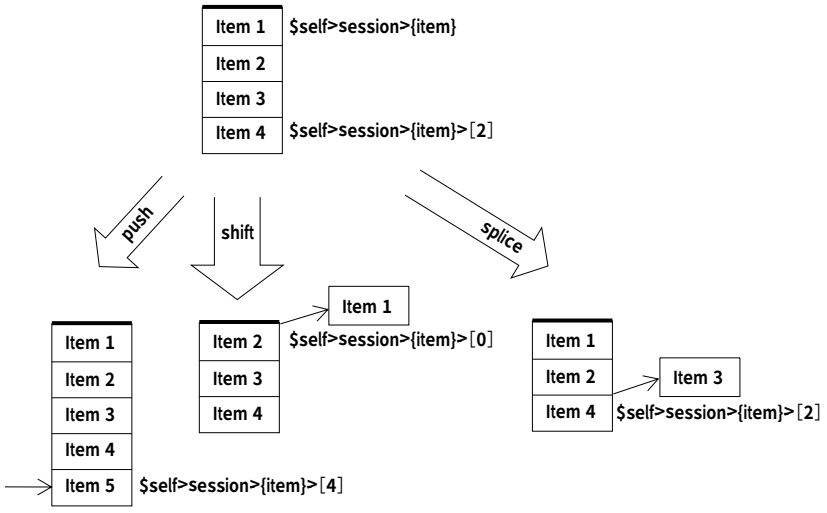
```
> mysqladmin -u user -p create webapp
Enter password: xxxxxxxxxx
Database "webapp" created.
> mysql -u user -p webapp
Enter password: xxxxxxxxxx
mysql>
mysql> CREATE TABLE catalog
    -> (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    ->  name VARCHAR(100), price INT,
    ->  photo VARCHAR(100), feature VARCHAR(255),
    ->  size VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.12 sec)
```

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

●リスト5 Cart.pm

```
package Cart;
use Apache::Session::DB_File;
use strict;

sub new
{
    my $class = shift;
    my $session_id = shift;
    my $self = bless {
        session => {}
    }, $class;
    $self->_initialize($session_id);
    $self;
}

sub _initialize
{
    my $self = shift;
    my $session_id = shift;

    eval {
        tie %{$self->session}, 'Apache::Session::DB_File',
            $session_id, {
                FileName      => '/usr/home/oyama/var/sessions.db',
                LockDirectory => '/usr/home/oyama/var/lock/sessions',
            };
    };
    if ($@) { die "create_session: $@" }
    $self;
}                                                              ❶

sub add_item
{
    my $self = shift;
    my @item = @_;
    push @{$self->session->{item}}, @item;
}                                                              ❷

sub delete_item
{
    my $self = shift;
    my $target = shift;
    splice @{$self->session->{item}}, $target, 1;
}                                                              ❸

sub item
{
    my $self = shift;
    $self->session->{item} = [] unless exists $self->session->{item};
    return (@{$self->session->{item}});
}                                                              ❹

sub session
{
    my $self = shift;
    if (@_) { $self->{session} = shift }
    $self->{session};
}                                                              ❺

sub id
{
    my $self = shift;
    $self->session->{_session_id};
}                                                              ❻

sub close
{
    my $self = shift;
    $self->{session} = {};
}                                                              ❼

sub DESTROY
{
    my $self = shift;
    $self->session->{last_access} = time;
    untie %{$self->session};
}                                                              ❽

1;
__END__
```

●図2 カートのアイテムの操作



カートからすべてのアイテムを削除する場合は，close( )メソッドを呼び出します（リスト5-❼）．

セッション情報の保存はCartオブジェクトが消滅するとき，自動的に呼び出されるDESTROYメソッドによって行われます（リスト5-❽）．

セッション情報の保存は，ハッシュのトップレベルの値を変更したときのみに行われます．アイテムを格納している$self->session->{item}は配列のリファレンスを格納しており，配列の要素を操作してもリファレンス自体は変化しません．そのためセッション情報を確実に保存させるために“last_access”に時刻を書き込んでいます．

またid( )メソッドでセッションIDを参照できます（リスト5-❻）．Cart.pmではセッションIDを“カートID”と呼ぶことにします．

レジスタ

レジスタはカートのアイテムを受け取り，注文を行います．精算に際してカートの他に利用者の住所・氏名・電話番号を受け取ることにします（リスト6，次ページ）．

クラスメソッドnew( )はカートのオブジェクトを引数として受け取り，新しいレジスタを生成します．

●リスト7　利用者の情報をハッシュで受け取る

```
$register->order(
    Address => '東京都新宿区西新宿パークタワー35F',
    Name    => '小山浩之',
    Phone   => '03-1234-5678',
);
```

```
my $register = Register->new($cart);
```

order( )メソッドは利用者の情報をハッシュで受け取り，カートの情報を元に注文情報としてデータベースに格納します（リスト7）．

注文情報を格納するテーブル“order_info”を定義します（リスト8）．

コントローラの開発

基本部品には部品間の関連やデータの受渡しに関するコードを記述していません．これらを結び付ける3つのコントローラを作成します．

- **catalog.cgi**
  カタログから商品を取り出し，一覧を表示します．
- **cart.cgi**
  カタログから指定した商品をカートに追加して，一覧を表示します．
  指定した商品をカートから削除して，残った商品

●リスト8　order_infoの定義

```
> mysql -u user -p webapp
Enter password: xxxxxxxxxx
mysql>
mysql> CREATE TABLE order_info
    -> (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    ->  session_id CHAR(32), item_id INT,
    ->  item_name VARCHAR(100), item_price INT,
    ->  address VARCHAR(255), name VARCHAR(100),
    ->  phone VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.12 sec)
```

●リスト9　catalog.html

```
<HTML>
<BODY>

<TMPL_LOOP name=ITEM_LIST>
  <TMPL_VAR name=NAME><BR>
  <TMPL_VAR name=PRICE><BR>
  <IMG src="<TMPL_VAR name=PHOTO>"><BR>
  <TMPL_VAR name=FEATURE><BR>
  <TMPL_VAR name=SIZE><BR>
  <A href="<TMPL_VAR name=ADD_CART_URI>">add cart</A><BR>
  <HR>
</TMPL_LOOP>

<A href="<TMPL_VAR name=CART_URI>">show cart infomation</A><BR>
</BODY>
</HTML>
```

●リスト6 Register.pm

```
package Register;
use DBI;
use strict;

sub new
{
    my $class = shift;
    my $cart = shift;
    my $self = bless {
        handle => undef,
        state  => undef,
        items  => [],
    }, $class;
    $self->cart($cart);
    $self;
}


sub order
{
    my $self = shift;
    my %customer = @_;
    $self->_check_customer_info(%customer)
        or die "order: Wrang customer infomation";

    my $handle = DBI->connect(
        'dbi:mysql:webapp', 'root', 'test', {
        RaiseError => 1,
      # AutoCommit => 0,
    });
    my $state = $handle->prepare(q{
        INSERT INTO order_info
                    (session_id, item_id, item_name, item_price, address, name, phone)
             VALUES (?, ?, ?, ?, ?, ?, ?)
    });
    eval {
        my $session_id = $self->cart->id;
        foreach my $item ($self->cart->item) {
            $state->execute(
                $session_id,
                $item->id,
                $item->name,
                $item->price,
                $customer{Address},
                $customer{Name},
                $customer{Phone},
            );
        }
    };
    if ($@) {
        die "order: $@"
    }
    else {
      # $handle->commit;
        $handle->disconnect;
    }
    $self;
}

sub _check_customer_info
{
    my $self = shift;
    my %customer = @_;
    return undef unless defined $customer{Address};
    return undef unless defined $customer{Name};
    return undef unless defined $customer{Phone};
    return $self;
}


sub cart
{
    my $self = shift;
    if (@_) { $self->{cart} = shift }
    $self->{cart};
}
1;
__END__
```

を表示します．

● register.cgi

注文に必要な利用者情報の入力欄を表示します．

入力欄に正しく記入した場合，カートの商品を注文します．

注文が完了したら，カートの内容を空にします．

catalog.cgi

catalog.cgiはHTML::Templateを使用してテンプレート（リスト9）を元に動的にHTMLを作成します．

Catalog.pmからアイテムを取り出し，テンプレートに値を埋め込んでいきます．

catalog.cgi（リスト10）はテンプレートを読み込み（リスト10-❶）カタログからアイテムを取り出し，テンプレートへ挿入するデータを用意します（リスト10-❷）．get_ item( )メソッドでカタログからアイテムを取り出し，対応するテンプレートのタグに属性をセットします．また，カートにアイテムを追加するURIを“ADD_CART_URI”で指定します．

最後にヘッダと，テンプレートから作成したHTMLを出力します（リスト10-❸）．

cart.cgi

cart.cgi（リスト11）も同様に，テンプレート（リスト12）を元に動的にHTMLを出力します．

cart.cgiの処理の流れはcatalog.cgiと同じですが，セッション管理を行うためにHTTP Cookieの操作を行います．

HTTP Cookieから取り出したカートID（セッションID）をCart.pmのクラスメソッドnew( )に渡し，指定してセッション管理の下でカートを生成します（リスト11-❶）．

これにより以前のカートの状態は復元されます．

アイテムの追加や削除は，URIで指定したパラメータ“mode”の値によって行います．

パラメータ“mode”の値が“add”の場合，パラメータ“item”で指定したIDのアイテムをカタログから読み込みカートに追加します（リスト11-❷）．サブルーチンload_item( )ではCatalog.pmを使用してアイテムを読み込んでいます（リスト11-❻）．

パラメータ“mode”の値が“delete”の場合，パラメータ“index”で指定した番号のアイテムをカートから削除します（リスト11-❸）．

一覧を表示するためにテンプレートへ属性をセット

●リスト10　catalog.cgi

```
#!/usr/bin/perl

use CGI;
use HTML::Template;
use Catalog;
use strict;

my $query = CGI->new;
my $html = HTML::Template->new(filename => 'catalog.html');  ❶
my $catalog = Catalog->new;
my $item_list = [];
while (my $item = $catalog->get_item) {
    push @{$item_list}, {
        NAME    => $item->name,
        PRICE   => $item->price,
        PHOTO   => $item->photo,                              ❷
        FEATURE => $item->feature,
        SIZE    => $item->size,
        ADD_CART_URI => 'cart.cgi?mode=add'. '&item='. $item->id
    };
}
$html->param(ITEM_LIST => $item_list);
$html->param(CART_URI => 'cart.cgi');

print $query->header;
print $html->output;                                          ❸
```

する処理は，catalog.cgiと同様です．セッション情報を維持するためカートIDを記録するHTTP Cookieを作成します（リスト11-❹）．

最後にテンプレートから作成したHTMLを出力して終了します（リスト11-❺）．

register.cgi

register.cgi（リスト13）では，入力フォームを表示するテンプレートリスト14と注文が完了した際に表示するテンプレートリスト15の2つを使用します．

入力フォームの値をチェックし，未記入や誤入力がある場合はサブルーチンshow_input_form()で入力フォームを表示します．また，すでに入力された値を引き継ぐため，HTML::Templateのオプション“associate”にCGI.pmのインスタンスを渡しています．associateオプションにCGI.pmのインスタンスを

●リスト11 cart.cgi

```
#!/usr/bin/perl

use CGI;
use HTML::Template;
use Catalog;
use Cart;
use strict;


my $query = CGI->new;
my $cart = Cart->new($query->cookie(-name => 'session_id'));                    ❶
my $mode = $query->param('mode');
if ($mode eq 'add') {
    my $item = load_item($query->param('item'));                                ❷
    $cart->add_item($item);
}
elsif ($mode eq 'delete') {
    $cart->delete_item($query->param('index'));                                 ❸
}

my $html = HTML::Template->new(filename => 'cart.html');
my $item_list = [];
my $index = 0;
foreach my $item ($cart->item) {
    push @{$item_list}, {
        ID      => $item->id,
        NAME    => $item->name,
        PRICE   => $item->price,
        DELETE_CART_URI => 'cart.cgi?mode=delete'. '&index='. $index++,
    };
}
$html->param(ITEM_LIST    => $item_list);
$html->param(CATALOG_URI  => 'catalog.cgi');
$html->param(REGISTER_URI => 'register.cgi');

my $state_cookie = $query->cookie(
    -name  => 'session_id',
    -value => $cart->id,                                                        ❹
    -path  => '/cgi-bin/',
);
print $query->header(-cookie => $state_cookie);                                 ❺
print $html->output;



sub load_item
{
    my $item_id = shift;
    my $catalog = Catalog->new($item_id);                                       ❻
    return $catalog->get_item;
}
```

指定すると，同名のテンプレートへ自動的に入力値をセットします（リスト13-❸）．

入力もれのある箇所をマーキングするために，パラメータをチェックして未入力の場合は“＊”を表示します（リスト13-❹）．

住所・氏名・電話番号をすべて入力している時は，

●リスト12　cart.html

```
<HTML>
<BODY>

<TMPL_LOOP name=ITEM_LIST>
  <TMPL_VAR name=ID>
  <TMPL_VAR name=NAME>
  <TMPL_VAR name=PRICE>
  <A href="<TMPL_VAR name=DELETE_CART_URI>">delete this item</A>
  <HR>

</TMPL_LOOP>

<A href="<TMPL_VAR name=CATALOG_URI>">catalog</A>
<A href="<TMPL_VAR name=REGISTER_URI>">register</A>

</BODY>
</HTML>
```

●リスト13　register.cgi

```
#!/usr/bin/perl

use CGI;
use HTML::Template;
use Register;
use Cart;
use Item;
use strict;


my $query = CGI->new;
my $cart = Cart->new($query->cookie(-name => 'session_id'));
if ($query->param('address') and
       $query->param('name') and $query->param('phone')) {
    store_order_information($query, $cart);                    ❶
    $cart->close;
    show_finish($query);
}
else {
    show_input_form($query, $cart);
}



sub store_order_information
{
    my $query = shift;
    my $cart = shift;
    my $register = Register->new($cart);
    $register->order(                                          ❷
        Address => $query->param('address'),
        Name    => $query->param('name'),
        Phone   => $query->param('phone'),
    );
}



sub show_finish
{
    my $query = shift;
    my $html = HTML::Template->new(
        filename => 'finish.html',
        associate => $query
    );
    $html->param(CATALOG_URI => 'catalog.cgi');
    my $expired_cookie = $query->cookie(
        -name    => 'session_id',
        -value   => '',
        -path    => '/cgi-bin/',
        -expires => 0,
    );
    print $query->header(-cookie => $expired_cookie);
    print $html->output;
}



sub show_input_form
{
    my $query = shift;
    my $cart  = shift;
    my $html = HTML::Template->new(
        filename => 'register.html',                           ❸
        associate  => $query
    );
    my $item_list = [];
    my $index = 0;
    foreach my $item ($cart->item) {
        push @{$item_list}, {
            NAME    => $item->name,
            PRICE   => $item->price,
            DELETE_CART_URI => 'cart.cgi?mode=delete'. '&index='. $index++,
        };
    }
    $html->param(ITEM_LIST => $item_list);
    $html->param(CATALOG_URI => 'catalog.cgi');
    $html->param(MARKING_ADDRESS => $query->param('address') ? '' : '*');
    $html->param(MARKING_NAME    => $query->param('name')    ? '' : '*'); ❹
    $html->param(MARKING_PHONE   => $query->param('phone')   ? '' : '*');

    print $query->header;
    print $html->output;
}
```

サブルーチンstore_order_information()で入力した情報とカートに含むアイテムを注文します（リスト13-❶）.

サブルーチンstore_order_information()は，CGIとカートのオブジェクトを受け取りRegister.pmのorder()メソッドで注文情報を格納します（リスト13-❷）.

注文処理が完了した後はclose()メソッドでカートの情報を破棄し，サブルーチンshow_finish()で注文完了画面を表示します．またHTTP Cookieの有効期限を切り，セッション情報をブラウザに破棄させます.

応答速度の確認

作成したコントローラはApache::Registryの環境下での実行を前提に実装していますが，CGIの環境下でも問題なく動作します．Apache※2にバンドルされているベンチマークツールApacheBenchを使用して応答速度を計測してみます.

ApacheBench

ApacheBenchは指定したURLへ指定した平行度と回数でリクエストを発行し，その応答速度を計測します．-cオプションでクライアントの数にあたる平行度，-

●リスト14 register.html

```
<HTML>
<BODY>

<TMPL_LOOP name=ITEM_LIST>
  <TMPL_VAR name=NAME>
  <TMPL_VAR name=PRICE>
  <A href="<TMPL_VAR name=DELETE_CART_URI>">delete this item</A>
  <HR>
</TMPL_LOOP>

<A href="<TMPL_VAR name=CATALOG_URI>">catalog</A>

<FORM method="GET" action="register.cgi">
  Address: <INPUT type="text" name="address" value="<TMPL_VAR name=ADDRESS>"><BLINK><FONT color="#FF0000"><TMPL_VAR
           name=MARKING_ADDRESS></FONT></BLINK><BR>
   Name: <INPUT type="text" name="name" value="<TMPL_VAR name=NAME>"><BLINK><FONT color="#FF0000"><TMPL_VAR
           name=MARKING_NAME></FONT></BLINK><BR>
   Phone: <INPUT type="text" name="phone" value="<TMPL_VAR name=PHONE>"><BLINK><FONT color="#FF0000"><TMPL_VAR
           name=MARKING_PHONE></FONT></BLINK><BR>
  <INPUT type="submit" value="submit">
</FORM>

</BODY>
</HTML>
```

●リスト15 finish.html

```
<HTML>
<BODY>

Thank you.<BR>
<TMPL_VAR name=address><BR>
<TMPL_VAR name=name><BR>
<TMPL_VAR name=phone><BR>

<A href="<TMPL_VAR name=CATALOG_URI>">catalog</A>


</BODY>
</HTML>
```

※2）参考URL：Apache Project http://httpd.apache.org/，Japanizaed Apache Server Project http://www.apache.or.jp/

tオプションで発行するリクエストの回数を指定します.

```
> ab -c 10 -t 100 http://localhost/cgi-bin/catalog.cgi
```

表1はApacheBenchを使用して，catalog.cgiの応答速度を計測した結果です.

結果Apache::Registryの環境で実行したWebアプリケーションは，CGIの約30倍の速度で実行することができました．コンパイル結果と変数のキャッシングを行わないApache::PerlRunの環境でも，スクリプトがコンパクトな場合は同等のパフォーマンスを得られることがわかります.

まとめ

データベースの操作などを行う「基礎部品」と，表示するGUIを決める「テンプレート」，それらを操作する「コントローラ」の3要素でWebアプリケーションを実装しました．一見取り扱うファイルが多く非効率的に見えるかもしれませんが，各々の要素の関連性が低いため，デザインや機能の変更や拡張に柔軟に対応することができます（図3）.

そしてmod_perlのApache::Registryを使用することで，CGIの約30倍の応答速度を実現しました.

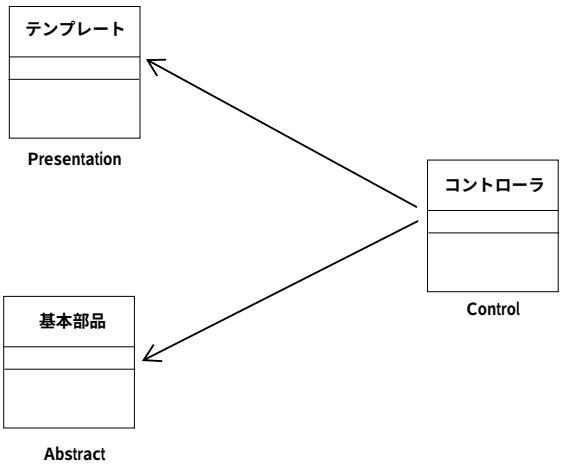
mod_perlのドキュメント“Introduction. Incentives. Credits.”※3にはmod_perlを採用している著名なWebサイトが紹介されています．それによるとMacromedia <http://www.macromedia.com/>では1ヶ月の利用者数4,273,000人，ValueClick<http://valueclick.com/>では45台のマシンで1日に8,000万件のアクセス，MP3.com <http://www.mp3.com/>では1日の利用者数408,000人にも上るトランザクションを処理しているそうです.

これらの高トラフィックなWebサイトで採用され実際に運用されていることからも，PerlによるWebアプリケーション開発の優位性やmod_perlのパフォーマンスの高さが証明できます.

そう，PerlのWebアプリケーションは速いのです.

◆

●図3 PAC関連図



●表1 catalog.cgiの応答速度

動作環境	1秒間の処理件数	転送速度
mod_perl（Apache::Registry）	20.64	14.64Kbps
mod_perl（Apache::PerlRun）	15.95	10.93Kbps
CGI	0.76	0.53Kbps

※単一のhtpdプロセスを対象に計測した結果

※3）参考URL：mod_perl guide: Introduction. Incentives. Credits. http://perl.apache.org/guide/intro.html