

PHP マニュアル

プログラミング基礎 for Windows, Linux

Smarty 編

1.MVC モデル

2.Smarty とは？

3.インストール

- template_dir
- compile_dir
- config_dir
- cache_dir

4.Smarty の基本

- assign
- display
- escape
- XSS

5.ループ処理と配列

- foreach

6.テンプレートの動的切り替え

- \$_SERVER['HTTP_USER_AGENT']
- preg_match

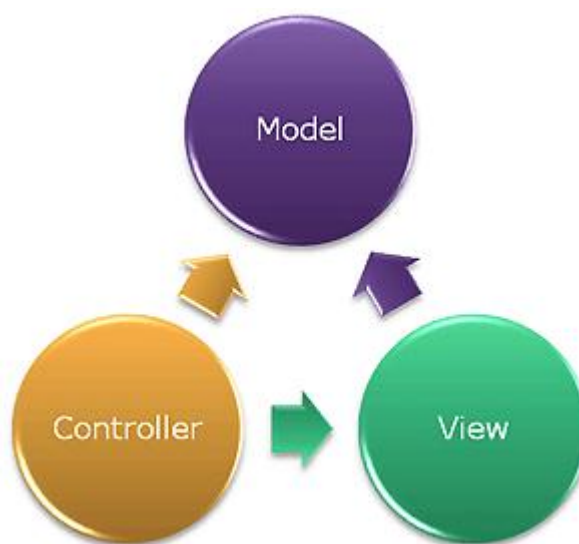
7.ブランケットをつかった効率的な配列処理

1.MVC モデル

MVCモデルとは、Model View Controller（モデル・ビュー・コントローラ）を意味します。これは、中～大規模システムを開発する場合に、処理の中核を担う「Model」、表示・出力を司る「View」、入力を受け取ってその内容に応じて View と Model を制御する「Controller」の 3 要素を機能ごとに分割する開発モデルのことを意味しています。

PHPなどではHTMLなどに直接コードを埋め込む仕様になっていますが、このような埋め込み式をとっている開発を行っている場合、規模が大きくなるにつれウェブデザイナーと共同で仕事をすることがふえてきます。コードとデザインを一緒くたにしている場合、例えばデザインの納期が遅れてしまうと、そのぶんコーディングにとりかかる時間がのびてしまいます。また、その逆でコーディングが遅れるとデザインにとりかかれれないという悪循環が発生してしまいます。また、コードを埋め込み式にしてしまうと、コード全体が複雑になったり冗長することにより可読性の低下が起きます。そのため、改修に時間がかかるなど、また、デザイナーが間違っってコードの箇所を変更してしまいバグが発生してしまう。または、その逆でプログラマーがデザインの箇所をいじったことで表示がおかしくなってしまうなどの問題がでてくるようになります。このような理由で最近では本来埋め込み式だったPHPをMVCモデルにもとづいて処理ごとに分割するスタイルが定着していきました(あくまで大規模ウェブサイト開発を前提としています)。

MVCのメリットは、デザインとロジックを平行して開発できること、また、デザイナーは本来のデザインに注力することができ、プログラマーはデザインにまで頭を悩ませることがなくなることです。もちろんこれはデザイナーとプログラマーのリソースがあるという前提での話です。小規模のシステム、または、プログラマーとデザイナーを兼任して開発している場合、MVCモデルを導入するとひとつのソースコードを分割するので逆にコード自体が短い場合、可読性が低下するなどの問題があります。



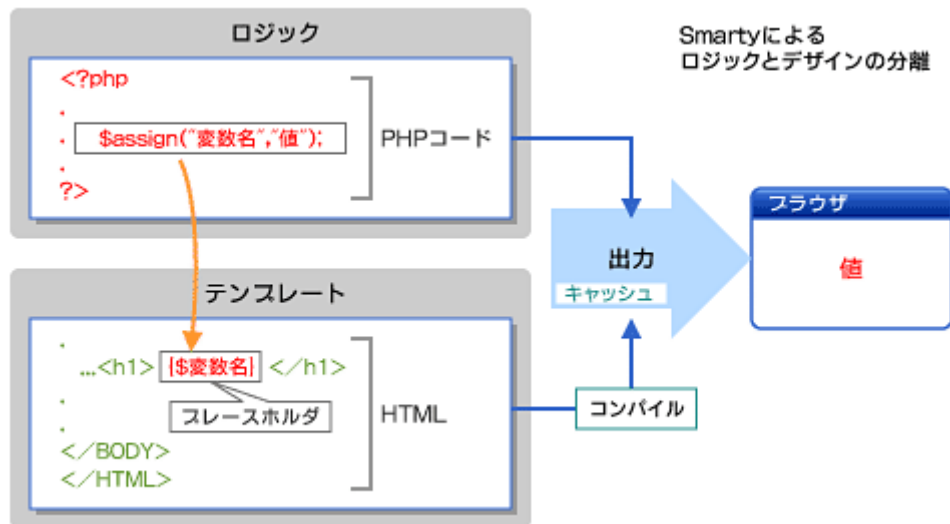
役割	概要
Model	データの管理を行うコンポーネント
View	HTML の出力を行うコンポーネント。大抵は Model のデータを元に出力
Controller	Model の操作、View へのリクエストの選択を行うコンポーネント ユーザーからのリクエストに応える処理を一元管理

2.Smarty とは？

Smartyとは、PHPでMVCモデルを実現するために開発されたロジック(PHP)とテンプレート(HTML)を分離結合することを可能にするテンプレートエンジンのことを指します。PHPでMVCを実現するというと現在の主流はSmartyを使用します。

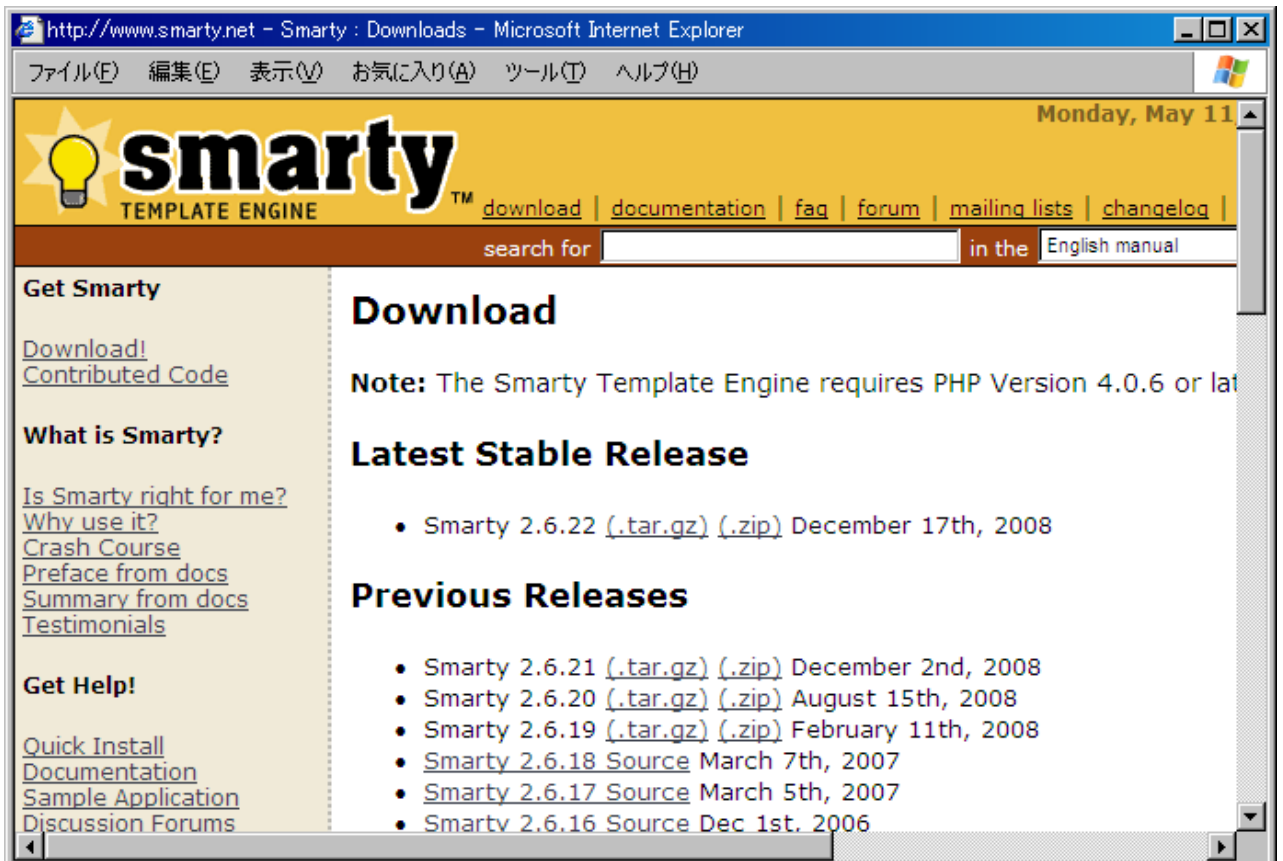
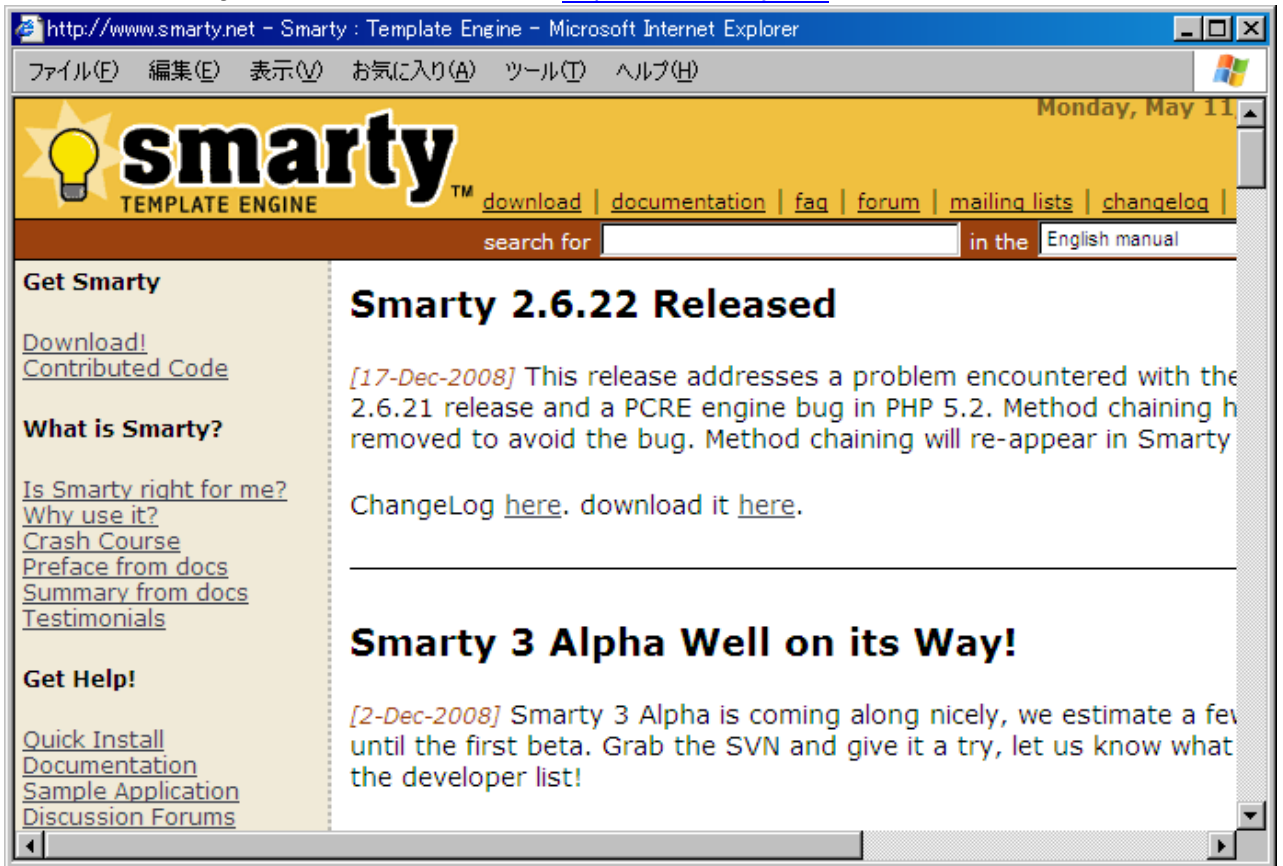
➤ Smarty の特徴

- ・非常に高速
- ・下仕事は PHP パーサが行うので能率的
- ・コンパイルは一度だけ行われるので、テンプレートのパースによるオーバーヘッドが無い
- ・再コンパイル は変更があったテンプレートファイルのみで行うのでスマート
- ・簡単に独自の 関数 や 変数の修飾子 を作成できるので、テンプレート言語を強力に拡張することが可能
- ・テンプレートの {デリミタ} タグの記法を変更し、{foo}、{{foo}}、<!--{foo}--> などを使用することが可能
- ・{if}..{elseif}..{else}..{/if} 構文は PHP パーサが処理するので、{if...} の条件式にはシンプルなものから複雑なものまで自由に指定可能
- ・sections や ifs などは無制限にネスト可能
- ・テンプレートファイル内に PHP コードを埋め込む ことも可能。しかし、エンジン自体が カスタマイズ できるので、これはおそらく不要 (そして非推奨)。
- ・組み込みで キャッシュ機能 をサポート
- ・任意の テンプレート ソース
- ・カスタム キャッシュハンドラ 関数
- ・プラグイン 機構



3.インストール

それでは、Smartyをインストールします。サイト(<http://www.smarty.net/>)からダウンロードしてください。



Latest Stable Release が最新安定バージョンです。Smarty 自体は PHP で書かれているので tar.gz (Linux でのアーカイブ形式) と zip 形式の中身は同じものです。圧縮形式が違うだけです。今回は Windows にインストールするので zip をダウンロードします。

Smarty の解凍先は、Linux の場合、/usr/local/lib/直下、Windows 環境の場合は公開フォルダ直下に解凍します。先に言ったとおり、Smarty は PHP でできているので基本的には Smarty を必要としているソース上で Smarty.class.php というファイルを require_once して、Smarty で使うフォルダのパーミッション設定しておけばすぐにつかえるようになります。Smarty ではデザイン用のテンプレートファイルと PHP のロジックファイルを結合し、一時的に保存してからそれを公開するという手順を追ってます。したがって、結合したファイルを一時的に保存するためのフォルダを用意しなければならないのです。このテンプレートファイルとロジックファイルを結合し保存する実際の処理は Apache などのサーバープログラムが行うため、一時的に保存するフォルダのパーミッションは Apache が書き込めるように設定しておかなければならないのです。

Smarty の本体は解凍した libs フォルダーのなかにある Smarty.class.php というファイルです。これをソース側で読み込みます。読み込む方法は絶対パスをそのまま書き込む方法と定数にいったん定義してそれを使うスタイルの2種類があります。一般的には後者のほうのいったん定数にしてから利用する方法をとります。こちらのほうが後でサーバー移行する際、定数の一箇所を修正するだけで済むからです。

➤ Smarty 用のディレクトリをセット

Smarty は、デフォルトで templates/、templates_c/、configs/ および cache/ と名づけられた4つのディレクトリが必要です。これらの名前は、それぞれ Smarty クラスのプロパティ \$template_dir、\$compile_dir、\$config_dir および \$cache_dir で定義することができます。Smarty を使用する各アプリケーションにおいて、これらのディレクトリを個別に設置する事を強く推奨します。

各フォルダの意味

cache	キャッシュ機能を有効にした場合に使用
config	設定ファイルを保存
templates	テンプレートファイルを保存
templates_c	コンパイル済み PHP コードを保存

Smarty ディレクトリは Smarty ライブラリによってのみアクセスされ、web ブラウザから直接アクセスされる事はありません。したがってセキュリティの心配を避けるために、これらのディレクトリをドキュメントルートの 外部 に配置する事を推奨します (ただし必須ではありません)。Smarty は \$compile_dir と \$cache_dir (templates_c/ と cache/) に 書き込み権限 でアクセスする必要があるので、web サーバのユーザがこれらに書き込める必要があります (windows ユーザはフォルダに権限を付与する必要(概念)がないのでこの話は無視してください)。

・パーミッションおよびディレクトリへの書き込み権限の付与

```
chown nobody:nobody 任意のパス/templates_c/  
chmod 770 任意のパス/templates_c/
```

```
chown nobody:nobody 任意のパス/cache/  
chmod 770 任意のパス/cache/
```

通常は、このユーザは "nobody" でグループは "nobody" です。OS X ユーザの場合は、デフォルトのユーザは "www" でグループは "www" です。もし Apache を使用しているなら、httpd.conf ファイルを見ればユーザ名とグループ名がわかります。chmod 770 は強固なセキュリティです。これは、ユーザ "nobody" とグループ "nobody" のみにディレクトリのリード/ライトアクセスを許可します。もし誰にでもリードアクセスを可能にしたい場合 (大抵はあなた自身がファイルを見るための利便性から) は、代わりに 775 を使う事が出来ます。

4.Smarty の基本

➤ Smarty の基本的な手続きの流れは以下の順序で行われます。

1) Smarty ライブラリの読み込み

```
require_once('../path/to../Smarty.class.php');
```

↓

2) Smarty インスタンスの作成

```
$smarty = new Smarty();
```

↓

3) 各ディレクトリの指定

```
$smarty->template_dir = './templates/';
```

```
$smarty->compile_dir = './templates_c/';
```

```
$smarty->config_dir = './configs/';
```

```
$smarty->cache_dir = './cache/';
```

↓

4) テンプレートの変数に値を割り当てる

```
$smarty->assign("変数名", "値");
```

↓

5) テンプレートを指定し表示

```
$smarty->display("テンプレートファイル");
```

➤ 実際のコード (ロジック側 : smarty.php)

```
<?php
```

```
//Smarty ライブラリ読み込み
```

```
//define('SMARTY_DIR', '/usr/local/lib/Smarty-2.6.22/libs/');
```

```
//define('SMARTY_DIR', 'C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\Smarty-2.6.22\libs\');
```

```
require_once('C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\Smarty-2.6.22\libs\Smarty.class.php');
```

```
//Smarty のインスタンスを作成
```

```
$smarty = new Smarty();
```

```
//各ディレクトリの指定
```

```
$smarty->template_dir = './templates/';
```

```
$smarty->compile_dir = './templates_c/';
```

```
$smarty->config_dir = './configs/';
```

```
$smarty->cache_dir = './cache/';
```

```
//キャッシュ機能の有効化
```

```
//$smarty->caching = true;
```

```
//テンプレートの変数に値を割り当てる
```

```
$smarty->assign("title", "Smarty サンプル 1");
```

```
$smarty->assign("value1", "1 つ目の値");
```

```
$smarty->assign("value2", "2 つ目の値");
```

```
...省略...
```

```
//テンプレートを指定し表示
```

```
$smarty->display("template1.tpl");
```

```
?>
```

➤ (テンプレート側 : ./templates/template1.tpl)

```
<HTML lang="ja">
<HEAD>
<META http-equiv="content-type" content="text/html; charset=UTF-8">
<TITLE>{$title|escape}</TITLE>
</HEAD>
<BODY TEXT="black" BGCOLOR="white">
{* テンプレート中のコメント *}

<!-- タイトル -->
<div align="center"><font size="5"><b>{$title|escape}</b></font></div>
<hr width="600" align="center">

<!-- テンプレートへの埋め込み -->
<div align="center">
取得した値 1:{$value1|escape}<br>
取得した値 2:{$value2|escape}
</div>

<!-- フッター -->
<hr width="600" align="center">
<table width="600" border="0" align="center">
<tr>
<td align="right"><font size="2">Smarty テスト<br>
2009/05/13 作成</font></td>
</tr>
</table>

</BODY>
</HTML>
```

PHP 側の display で結合テンプレートを指定しているのがわかると思います。assign した値をテンプレート側で変数として展開するわけです。ちなみにテンプレート側に escape とあるのは、<>のタグキャラクターを変換するコードです(エスケープするといいます)。これは、掲示板システムなどの XSS(クロスサイトスクリプティング)と呼ばれる脆弱性対策のひとつです。これをしないと掲示板システムなどをつくった場合、任意の JavaScript コードを埋め込まれてしまう恐れがあります。

5. ループ処理と配列

Smarty では、ループ処理とそれをつかった配列の展開もサポートされています。実際にコードを見たほうが早いので、以降にサンプルを掲載します。

➤ 実際のコード(ロジック側 : smarty2.php)

```
//配列の作成
$data = array("a","b","c","d");

//テンプレートの変数に値を割り当てる
$smarty->assign("title", "Smarty サンプル 2");
$smarty->assign("data", $data);

//テンプレートを指定し表示
$smarty->display("template2.tpl");
```

➤ 実際のコード(テンプレート側 : ./templates/template2.tpl)

```
{* 配列の利用 *}
{foreach from=$data item=value name=loop01}
<li>値は「{$value|escape}</li>
{foreachelse}
表示させるデータがありません。
{/foreach}
```

➤ 書式

```
{foreach from=[配列] key=[キー] item=[アイテム] name=[このループブロックの名前]}
// 処理
{foreachelse}
// [配列]が空だった場合の処理
{/foreach}
foreach の基本構文
```

例のように、\$value で配列の要素を取り出せます。また、key を foreach で設定することにより、連想配列での添え字を取得することも可能になります。

➤ ループ処理と連想配列 (ロジック側 : smarty3.php)

```
//連想配列の作成
$data = array("Sun"=>"日曜日","Mon"=>"月曜日","Tue"=>"火曜日","Wed"=>"水曜日");

//テンプレートの変数に値を割り当てる
$smarty->assign("title", "Smarty サンプル 3");
$smarty->assign("data", $data);

//テンプレートを指定し表示
$smarty->display("template3.tpl");
```

➤ ループ処理と連想配列 (テンプレート側 : ./templates/template3.tpl)

```
{* 連想配列の利用 *}
{foreach from=$data key=key item=value name=loop01}
<li>キーは「{$key|escape}」、値は「{$value|escape}」</li>
{foreachelse}
表示させるデータがありません。
{/foreach}
```

➤ foreach のネスト (ロジック側 : smarty4.php)

```
//連想配列の配列化
$data = array(
array("name"=>"MySQL","price"=>"850 円","qty"=>"5"),
array("name"=>"BIND9","price"=>"780 円","qty"=>"4"),
array("name"=>"Apache","price"=>"480 円","qty"=>"7")
);

//テンプレートの変数に値を割り当てる
$smarty->assign("title", "Smarty サンプル 4");
$smarty->assign("data", $data);

//テンプレートを指定し表示
$smarty->display("template4.tpl");
```

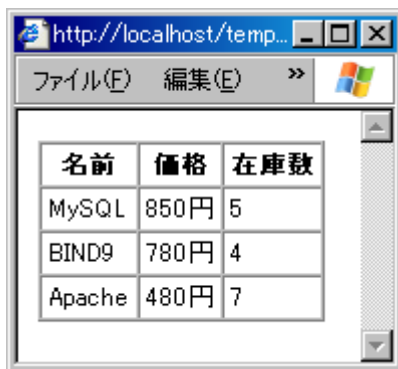
➤ foreach のネスト(テンプレート側 : ./templates/template4.tpl)

```
{* 連想配列の配列利用 *}
{foreach from=$data item=value01 name=loop01}
{foreach from=$value01 key=key item=value02 name=loop02}
{$key|escape}:{$value02|escape},
{/foreach}
<br>
{foreachelse}
表示させるデータがありません。
{/foreach}
```

例のようにネスト(入れ子)を利用して多次元配列を展開することも可能になります。foreach のネストでは多次元配列を全て展開する方法をとっています。この場合、添え字名がわかっているならばネストを添え字名に置き換えることができます。これも実際のコードを見たほうがわかりやすいと思います。また、HTML を利用して変数を展開したい場所に記述するということができます。

➤ foreach のネスト(テンプレート側 : ./templates/template4.tpl)

```
{* 連想配列の利用 *}
<table border="1" cellpadding="3" cellspacing="0">
<tr><th>名前</th>
<th>価格</th>
<th>在庫数</th>
</tr>
{foreach from=$data item=value01 name=loop01}
<tr><td>{$value01.name|escape}</td>
<td>{$value01.price|escape}</td>
<td>{$value01.qty|escape}</td>
</tr>
{foreachelse}
表示させるデータがありません。
{/foreach}
</table>
```



名前	価格	在庫数
MySQL	850円	5
BIND9	780円	4
Apache	480円	7

6. テンプレートの動的切り替え

ロジックとデザインを切り離すことで、ロジックやデザインの再利用性も高まります。例えば、PC や携帯電話などそれぞれの端末の種類に特化したテンプレートを用意しておき、ロジック側でブラウザの種類を判別し、それに基づき適切なテンプレートを選択するといった利用が可能です。

//連想配列の配列化

```
$data = array(
    array("name"=>"半熟卵の MySQL 風チキンドリア","price"=>"850 円","qty"=>"5"),
    array("name"=>"地鶏とキノコの BIND9 風フェットチーネ","price"=>"780 円","qty"=>"4"),
    array("name"=>"有機野菜の Apache 風グリーンサラダ","price"=>"480 円","qty"=>"7")
);
```

//テンプレートの変数に値を割り当てる

```
$smarty->assign("title", "Smarty サンプル 6");
$smarty->assign("data", $data);
```

//ブラウザの種類でテンプレートを切り替え

//環境変数「HTTP_USER_AGENT」でブラウザを判定

```
$agent = $_SERVER["HTTP_USER_AGENT"];
if(preg_match("/^Mozilla/", $agent) || preg_match("/^Opera/", $agent)){
    //PC 系
    $smarty->display("template6_pc.tpl");
}elseif(preg_match("/^SoftBank/", $agent) || preg_match("/^Vodafone/", $agent)){
    //SoftBank 携帯
    $smarty->display("template6_mobile.tpl");
}elseif(preg_match("/^DoCoMo/", $agent)){
    //DoCoMo 携帯
    $smarty->display("template6_mobile.tpl");
}elseif(preg_match("/^KDDI/", $agent)){
    //au 携帯
    $smarty->display("template6_mobile.tpl");
}else{
    //その他
    $smarty->display("template6_etc.tpl");
}
?>
```

ブラウザの判定には、環境変数「HTTP_USER_AGENT」を利用します。環境変数「HTTP_USER_AGENT」には各ブラウザが発行する「ユーザーエージェントタイプ」が格納されています。その中に特定の文字列が含まれているかどうかによって、各携帯キャリアや PC ブラウザの種類を区別します。

また、環境変数「HTTP_USER_AGENT」はクライアント側で簡単に偽装できるため、過度な使用には注意します。文字列の判定には preg_match() を使用しています。preg_match() では正規表現によるマッチングを利用することができ、柔軟に文字列を扱うことが可能です。サンプルでは PC 系 (template6_pc.tpl)、携帯キャリア系 (template6_mobile.tpl)、その他 (template6_etc.tpl) の 3 種類のテンプレートを用意しています。PC 系テンプレート「template6_pc.tpl」では、使用するアイコンや表示領域を一般的な PC モニタの解像度に合わせたものを利用し、携帯系テンプレート「template6_mobile.tpl」ではアイコンや表示される領域を最小限に、その他のブラウザにはテンプレート「template6_etc.tpl」を使って使用しているブラウザがコンテンツに対応していないことを表示できるようにすることができますようになります。

```

> templates/template6_pc.tpl
[* PC 用コンテンツ *}
{foreach from=$data item=value01 name=loop01}
  <table width="390" border="0" cellspacing="0" cellpadding="7" height="55" align="center">
    <tr>
      <td valign="top" width="40"></td>
      <td valign="top" width="350"><font size="2"><b>{$value01.name|escape}</b></font><br>
        <div align="right"><b>¥ {$value01.price|escape}/1 個</b>
        <b>{$value01.qty|escape}個</b></div>
      </td>
    </tr>
  </table>
{foreachelse}
表示させるデータがありません。
{/foreach}
</table>

```

```

> templates/template6_mobile.tpl
[* 携帯コンテンツ *}
{foreach from=$data item=value01 name=loop01}
  <table border="0" align="center" width="100%">
    <tr>
      <td width="20" height="30" valign="top"></td>
      <td align="left"><font size="2"><b>{$value01.name|escape}</b></font><br>
        <div align="right"><b>¥ {$value01.price|escape}/1 個 </b>
        <b>{$value01.qty|escape}個</b></div>
      </td>
    </tr>
  </table>
{foreachelse}
表示させるデータがありません。
{/foreach}
</table>

```

```

> templates/template6_etc.tpl
[* ブラウザ未対応 *}
<div align="center">
<font size="2">ご使用のブラウザには対応していません。</font>
</div>

```

このようにロジックとデザインテンプレートを切り離すことにより大規模なサイトでは埋め込み式のコードよりも記述する行数が少なくなります。また、おわかりのとおり、分岐によりテンプレートを自由に切り替えることによりロジックはロジックだけ、テンプレートはテンプレートだけと可読性が非常に向上するのが Smarty の利点です。

7. ブランケットをつかった効率的な配列処理

Smarty を使用した変数の割り当てには assign 以外に append というクラスメソッドがあります。これをつかうことでデータベースから複数行抽出し、1レコードずつ Smarty に割り当てることが可能です。Smarty 側ではこれを2次元配列で取得することになります。記述的には assign よりも見やすくなります。

➤ add.php

```
$sql1 = "SELECT * FROM tablename";

$row = NULL;
$result = mysql_query($sql) or die ("SQL 抽出エラー {$sql}");
while($row = mysql_fetch_assoc($result)){
    $smarty->append( 'row', $row );
}
```

➤ add.tpl

```
{foreach item="row" from=$row }
<tr>
    <td><input name="rec[]" value="{ $row.rec}" type="hidden"></td>
    <td><textarea name="history[]" cols=145 rows=4>{ $row.history}</textarea></td>
</tr>
{/foreach}
```

また、Smarty の出力先テンプレートの name 属性にブランケットを追記することでフォームをサブミットした場合、受信先のスクリプトでその要素を 0 からの配列で受信することが可能となります。これは不特定多数の送信要素を記述する際、大変便利な書き方になります。受信側スクリプトでは配列を取得した際に配列をカウントし、それを元に for を回すことで処理を行うことで不特定多数の要素の処理が可能となります。

➤ commit.php

```
$cnt = count($_POST['rec']);
for($i=0;$i<$cnt;$i++){
    $rec = $_POST['rec'][$i];
    $history = $_POST['history'][$i];
    echo $rec;
    echo $history;
}
```

ブランケットを使用した記述は PHP 側でのスクリプト記述でも、配列を定義し、空のブランケットに代入させることで不特定多数の配列を処理することが可能です。またこの場合でも 0 からの配列を作成することになります。

➤ add.php

```
$length = array();
$result = mysql_query($sql) or die ("SQL 抽出エラー {$sql}");
while($row = mysql_fetch_assoc($result)){
    $length[] = $row;
}
```