

ハッスリング HTML_QuickForm

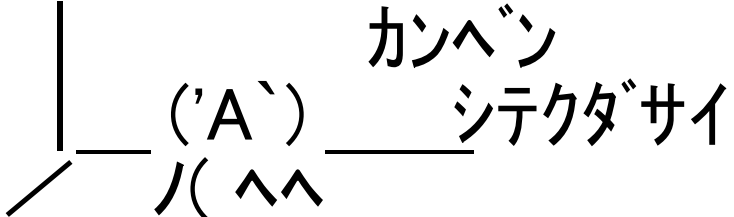
副題: メンドクサ人のためのQuickForm

熊倉 洋介 (kumakura@isite.co.jp)

2004年9月23日
第2回PHP関西セミナー

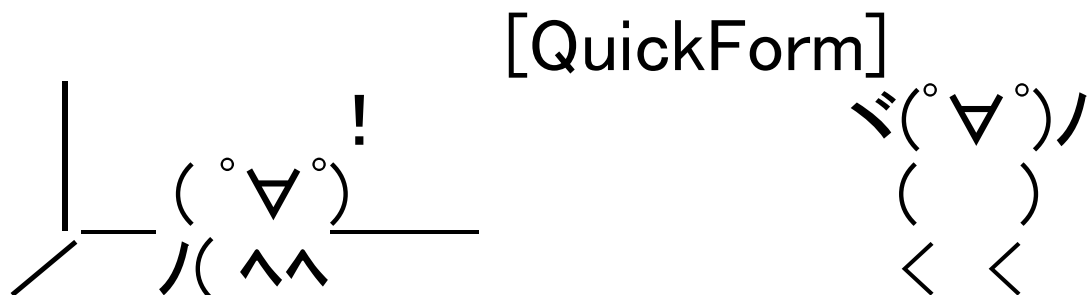
早速ですが…

Form ページ作成って何かと面倒ですよね。

- 入力値判断 （数字のみ、n文字以内などの入力制限）
 - エラーの場合はエラーメッセージ出力と共に再入力促し
 - 前回の入力値をフォームに適応
- 1つの入力個所に制限・条件が複数あったら…
- そんな入力個所が10個も20個もあったら…
- そんなページがサイト中複数あったら…
- そんなページ間の自由な移動が (ry 

そんな悩みに

HTML_QuickForm !!



- 入力値判断を一括処理
- よくある条件はすでに完備・1行で終了
- 自作条件も追加・適応可能
- 再入力時の値適応は当たり前
- 多様なレンダラーで好きな形に自動変換

フォームの要素作成 1

```
HTML_QuickForm::createElement($elementType);
```

`$elementType` :: 要素の種類

text, select,

checkbox,

radio, submit....etc.

要素それぞれに対するクラスが準備されており、
あわせて引数が引き渡される

フォームの要素作成 2

```
HTML_QuickForm::addElement ($element);
```

\$element :: 要素の種類 or 要素オブジェクト

createElement() で作った要素

※ どうこう言うより例を見たほうが早い

(1) text

```
addElement('text', $name, $label, $attributes);
```

`$name` : 要素の名称

`$label` : 要素のラベル

`$attributes` : その他 HTML で出力する際の属性

```
$form->addElement('text', 'zip', 'zip',  
                  array('size' => 7));
```



```
zip <input size="7" name="zip"  
        type="text" value="" />
```

(2) select

```
addElement('select', $name, $label,  
           $options, $attributes);
```

\$name : 要素の名称

\$label : 要素のラベル

\$options : select 内の値

\$attributes : その他 HTML で出力する際の属性

```
$form->addElement('select', 'city', '街',  
                array('1' => '大阪',  
                      '2' => '京都',  
                      '3' => '神戸'));
```



街

```
<select name="city">  
  <option value="1">大阪</option>  
  <option value="2">京都</option>  
  <option value="3">神戸</option>  
</select>
```

(3) checkbox

```
addElement('checkbox', $name, $label,  
           $text, $attributes);
```

\$name : 要素の名称

\$label : 要素のラベル

\$text : その checkbox の表記名

\$attributes : その他 HTML で出力する際の属性

```
$form->addElement('checkbox', 'cb', null, 'A');
```



```
<input name="cb" type="checkbox"  
  value="A" id="qf_12f3c9" />  
<label for="qf_12f3c9">A</label>
```

(4) radio

```
addElement('radio', $name, $label,  
           $text, $value, $attributes);
```

\$name : 要素の名称

\$label : 要素のラベル (radio の前に出力される)

\$text : 要素の表記名 (radio の後ろに出力される)

\$value : その radio の値

\$attributes : その他 HTML で出力する際の属性

```
$form->addElement('radio', 'radio', null,  
                'Yes', '1');
```



```
<input name="radio" value="1" type="radio"  
      id="qf_5b1d99" />  
<label for="qf_5b1d99">Yes</label>
```

(5) submit

```
addElement('submit', $name, $value, $attributes)
```

\$name : 要素の名称

\$value : その submit button の表記名

\$attributes : その他 HTML で出力する際の属性

```
$form->addElement('submit', 'submit', '送信');
```



```
<input name="submit" value="送信"  
      type="submit" />
```

radio ボタンとかは「[複数項目の中から1つのみ選択](#)」
なんて使い方が多い。

直接 HTML を記載する場合なら、 name を統一すれば実現。

QuickForm の場合は、QuickForm の要素グループにまとめる。

```
addGroup($elements, $name, $label, $separator)
```

\$elements : 要素の集合体 (array)

\$name : 要素グループの名称

\$label : 要素グループのラベル

\$separator : グループ内の各要素のセパレータ

(例えば、各 radio 間は改行をいれる、など)

```
$question[]  
  = &HTML_QuickForm::createElement('radio', null,  
                                     'Yes'  
                                     'Yes', '1');  
  
$question[]  
  = &HTML_QuickForm::createElement('radio', null,  
                                     'No'  
                                     'No', '2');  
  
$form->addGroup($question, 'question');
```



```
<input value="1" type="radio" id="qf_9e493d"  
  name="question" /><label for="qf_9e493d">Yes</label>  
  
<input value="2" type="radio" id="qf_451a03"  
  name="question" /><label for="qf_451a03">No</label>
```

```
<?php
require_once('HTML/QuickForm.php');
$form = new HTML_QuickForm();
$form->addElement('text', 'zip', 'zip', array('size' => 7));
$form->addElement('select', 'city', '街', array('1' => '大阪', '2' => '京都', '3' => '神戸'));
$form->addElement('checkbox', 'cb', null, 'A');
$question[] = &HTML_QuickForm::createElement('radio', null, 'Yes', 'Yes', '1');
$question[] = &HTML_QuickForm::createElement('radio', null, 'No', 'No', '2');
$form->addGroup($question, 'question');
$form->addElement('submit', 'submit', '送信');
$form->display();
?>
```

zip

街

A

Yes No

要素のルール適応

- フォームに入力された値が果たして相応しい値なのかどうか
- 相応しくない値ならば再入力をしてもらわないといけない
- 具体的な誤りの内容まで出力されたらイイ感じのページ

各要素に入力された値のルール（入力制限）決めには

`addRule()` or `addGroupRule()` ← (要素グループに対して)

QuickForm には「良く使われそうな入力制限」が予め準備済み。
これらを目的に合わせて適応するのみ！

加えてエラーメッセージを設定できる。（上から順にエラー適応）

```
$form->addElement('text', 'zip', 'zip',  
                  array('size' => 7));
```

```
$form->addRule('zip', '未入力です', 'required');
```

```
$form->addRule('zip', '数字を入力してください',  
              'numeric');
```

```
$form->addRule('zip', '郵便番号は数字7文字です',  
              'rangeLength', array(7, 7));
```

※ 予め準備されている入力制限条件はマニュアルを参照

特殊な入力制限を行いたい場合は、別途ルールの適応が可能

```
function cmpZip($fields)
{
    require_once('Zip.php');
    if (ZIP::isMatch($fields['zip'])) {
        return true;
    } else {
        return array('zip' => '不明な郵便番号です');
    }
}

$form->addFormRule('cmpZip');
```

※ Zip.php の isMatch() メソッドは郵便番号が存在するか否かを判定する自作のプログラムを想定

特殊なルールを多方面で使いたい場合、新たにルールを新設し他のルールと同じように扱うこともできる

```
<?php
require_once('HTML/QuickForm/Rule.php');
require_once('Zip.php');

class QuickForm_Rule_ZipCode extends HTML_QuickForm_Rule
{
    function validate($value)
    {    return ZIP::isMatch($value) ? true : false;    }
}
?>
```

```
require_once(CLASS_DIR . 'QuickForm/Rule/ZipCode.php');
$form->registerRule('zipcode', null,
                  'QuickForm_Rule_ZipCode');
$form->addRule('zip', '不明な郵便番号です', 'zipcode');
```

実際の入力制限の判定処理 `validate()`

このタイミングで各要素の入力値判定を行うので
`addRule()` 等での入力制限適応後に `validate()` すること

```
if ($form->validate()) {  
    // 入力値が正しい場合の処理  
} else {  
    // 再入力を促す処理  
}
```

HTML の出力

- ・ HTML 出力がいじりやすいようレンダラーが準備されている
- ・ テンプレートエンジンのレンダラーも準備済み

Smarty

PEAR_HTML_Template_Flexy

PEAR_HTML_Template_IT ...

私は Smarty 信者なので
容赦なく Smarty との連携について紹介します。

※ 皆さん一緒に Smarty を使いましょう



Smarty の基本的なお話



- 出力部分と内部処理のロジックとコンテンツの分離
- テンプレートファイルはほとんどHTMLファイルそのまま
可変箇所のみ変数を設置するイメージ
- テンプレートファイル内に条件分岐や繰り返し処理が
書けちゃう（デザイナーさん泣かせとの声も…）

PHP 側 出力処理

```
$smarty = new Smarty;  
$smarty->assign('title', 'ほげほげページ');  
$smarty->assign('name', $name);  
$smarty->display('index.tpl.html');
```

Smarty テンプレートファイル (index.tpl.html)

```
<html>  
<head><title>{$title}</title></head>  
  
<body text="#000000" bgcolor="#ffffff">  
<h1>{$title} へようこそ</h1>  
<br />  
{ $name } とかあってなんたらかんたら  
</body></html>
```

QuickForm の form を Smarty で出力

PHP 側 出力処理

```
$smarty = new Smarty;  
$smarty->assign('title', 'ほげほげページ');  
$smarty->assign('name', $name);  
  
$renderer  
    =& new HTML_QuickForm_Renderer_ArraySmarty($smarty);  
$form->accept($renderer);  
$smarty->assign('form', $renderer->toArray());  
    // QuickForm オブジェクト “$form” 内の各要素が  
    // Smarty 変数 “form” へ assign  
  
$smarty->display('index.tpl.html');
```

Smarty テンプレートファイル (index.tpl.html)

```
<body text="#000000" bgcolor="#ffffff">
<form {$form.attributes}>
{$form.zip.label}:{$form.zip.html}<br />
    (input テキスト Form)
{$form.city.label}:{$form.city.html}<br />
    (select Form)
....
</form>
```

Smarty 変数 \$form の中身は var_dump() すれば一目。
要素ごとの配列です。

(要素グループの場合は多配列になっている。アクセスは
{ \$form.question.1.html },
{ \$form.question.2.label } などなど。)

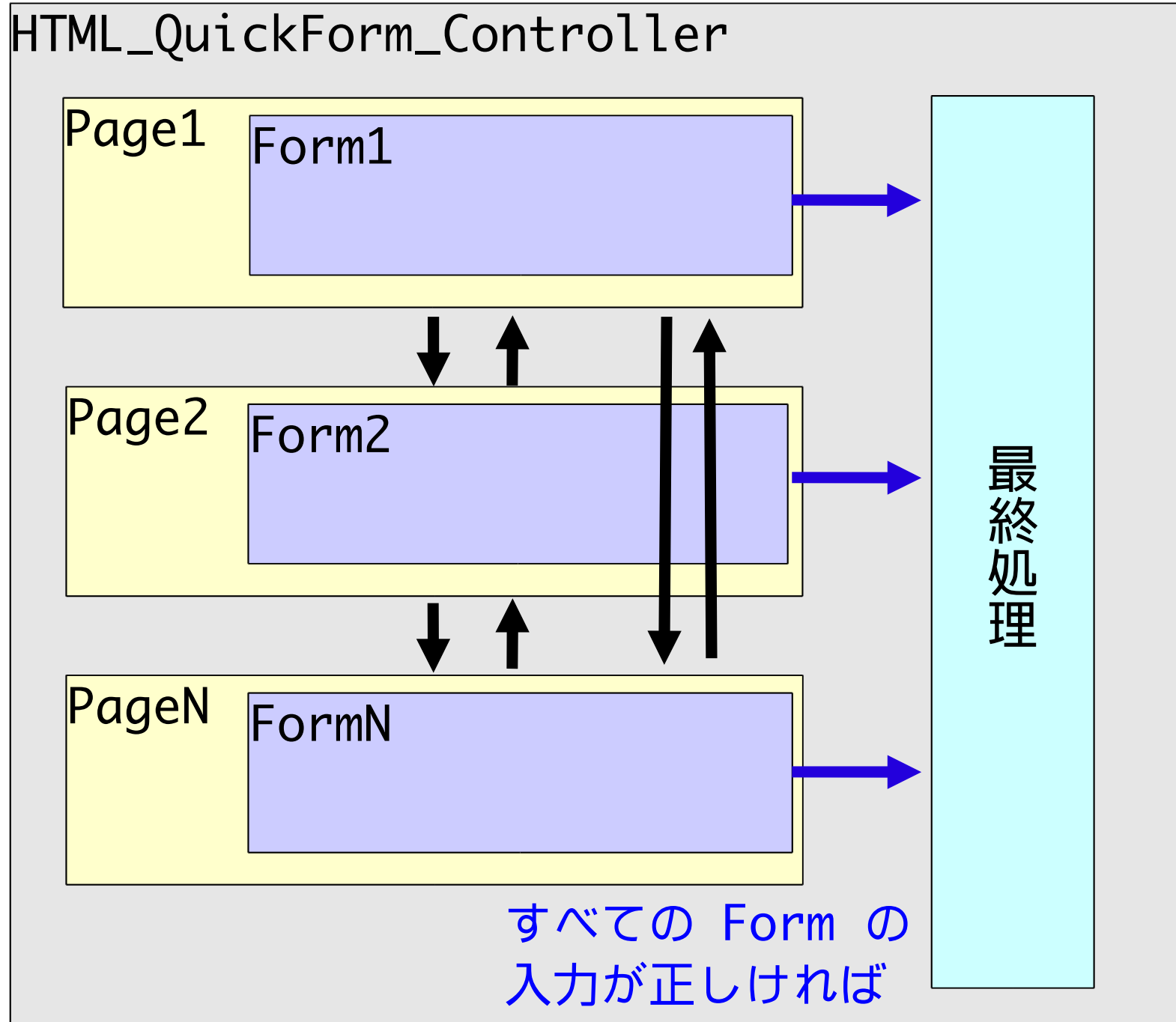
Smarty レンダラーのいい使い方

```
require_once('HTML/QuickForm/Renderer/ArraySmarty.php');
class Test_Renderer extends HTML_QuickForm_Renderer_ArraySmarty
{
    function &Test_Renderer($template)
    {
        parent::HTML_QuickForm_Renderer_ArraySmarty($template);
        $this->setRequiredTemplate(
            '{if $error}<font color="#FF0000">{$label}</font><br />
            {else}{$label}<br />
            {/if}'
        );
    }
}
```

HTML_QuickForm_Controller

- QuickForm を使用した 「複数ページ」 を操作
- 複数ページ Form の validate
- ページの遷移（「次へ」「前へ」「ジャンプ」）を管理
移動時に Action を挟むことも可能
- 超簡易 & 小規模 MVC (model-view-controller)
framework

大雑把な QFC イメージ図



QFC の使用例

(1) QFC Page

```
class Page_FormN extends HTML_QuickForm_Page
{
    function buildForm()
    {
        // addElement(), addRule など QuickForm と同様の処理

        $button[] =& $this->createElement('submit',
                                           $this->getButtonName('back'),
                                           '戻る');
        $button[] =& $this->createElement('submit',
                                           $this->getButtonName('next'),
                                           '次へ');
        $this->addGroup($button, 'button', '', '&nbsp;', false);
        $this->setDefaultAction('next');
    }
}
```

(2) QFC Action

予め準備された Action

- Next, Back, Jump (ページ遷移系)
- Submit (完全評価)
- Display (表示)
- Process (最終処理)

に加えて、サイトにあわせた自前 Action の実装が可能

例：

- page1 で user/pass を入力、ユーザ有無を DB に問い合わせ、真ならば Next
- 入力値に応じて違った Page へ Jump

などなど

例：page 遷移の途中でデータベース更新処理 Action

```
class Action_updateDB extends HTML_QuickForm_Action
{
    function perform(&$page, $actionName)
    {
        $page->isFormBuilt() or $page->buildForm();
        $pageName = $page->getAttribute('name');
        $data      =& $page->controller->container();
        $data['values'][$pageName]
            = $page->exportValues();
        $data['valid'][$pageName]
            = $page->validate();
        if ($data['valid'][$pageName]) {
            // データベース更新処理
            $action_name
                = $page->controller->getActionName();
            $values
                = $page->controller->exportValues();

            if ($this->update($values)) {
                $next
                    =& $page->controller->getPage
                        ($action_name[0]);
                $next->handle('next');
            } else {
                // error 処理
            }
        } else {
            return $page->handle('display');
        }
    }

    function update($values) {
        ....
    }
}
```

```
class Page_FormN extends HTML_QuickForm_Page
{
    function buildForm()
    {
        // addElement(), addRule など QuickForm と同様の処理

        $button[] =& $this->createElement('submit',
                                           $this->getButtonName('updateDB'),
                                           '送信');
        $this->addGroup($button, 'button', '', '&nbsp;', false);
        $this->setDefaultAction('updateDB');
    }
}
```

表示処理を行う Display Action

テンプレートを使用する場合はここへ記述する

```
class Action_Display extends HTML_QuickForm_Action_Display
{
    function _renderForm(&$page)
    {
        $smarty = new Smarty;
        $smarty->assign('title', 'ほげほげページ');

        $renderer =& new Smarty_Renderer($smarty);
        $page->accept($renderer);
        $smarty->assign('page', $renderer->toArray());
        $page_action = $page->controller->getActionName();
        $smarty->display($page_action[0] . '.tpl.html');
    }
}
```

Process Action は全 Page Form の評価が真の時に呼び出される
最終処理として行いたいものを記載する

```
class Action_Process extends HTML_QuickForm_Action
{
    function perform(&$page, $actionName)
    {
        $page->isFormBuilt() or $page->buildForm();
        $pageName = $page->getAttribute('name');
        $data      =& $page->controller->container();
        $data['values'][$pageName] = $page->exportValues();
        $data['valid'][$pageName]  = $page->validate();
        $values = $page->controller->exportValues();

        // 以下 $values を使って最終処理色々
    }
}
```

アクセスする PHP ファイルへの QFC 有効化

new() → addPage() → addAction() → run() とすれば
あとは QFC へお任せ。

```
$qfc =& new HTML_QuickForm_Controller('Forms');  
$qfc->addPage(new Page_Page1('page1'));  
$qfc->addPage(new Page_Page2('page2'));  
$qfc->addPage(new Page_Confirm('confirm'));  
  
$qfc->addAction('next', new HTML_QuickForm_Action_Next());  
$qfc->addAction('back', new HTML_QuickForm_Action_Back());  
$qfc->addAction('updateDB', new Action_UpdateDB());  
$qfc->addAction('display', new Action_Display());  
$qfc->addAction('process', new Action_Process());  
  
$qfc->run();
```

QuickForm ちょっとした使い方

`Net_UserAgent_Mobile` と組み合わせたの
PC+モバイル向けサイトでの Form 入力フォーマット指定

(1) quickform.conf

[DoCoMo]

key = istyle

hiragana = 1

kana = 2

alphabet = 3

numeric = 4

[UP.Browser]

key = format

hiragana = *M

...

[J-PHONE]

...

[Non-Mobile]

...

(2) Form 構成

```
$qf_conffile = '/path/to/quickform.conf';
$qf_conf = parse_ini_file($qf_conffile, TRUE);
$agent = &Net_UserAgent_Mobile::singleton();
if ($agent->isNonMobile()) {
    $agent_name = 'Non-Mobile';
} else {
    $agent_name = $agent->getName();
}

$form->addElement('text', 'zip', 'zip',
    array('size' => 7,
        $qf_conf[$agent_name]['key'] =>
        $qf_conf[$agent_name]['numeric']
    )
);
```