フレームワーク作成道中 MVC編

古庄 道明

1　とりあえず 何はさておき 前口上

この文書では、フレームワークの作成の流れを追いかけていきます。んで、フレームワークって言っても色々あるので。今回は「MVC という枠組みを実現する」部分について、それがない事でどんな風に困り、それがある事でどんな風に便利になっていくのかの流れについて記述していきます。

この文書が、「MVCって美味しいの？」って思われている数多の方々の理解の一助にでもなれば、筆者望外の幸いです。

2　単純に 作ってみたのは いいけれど

ちょっと Web で Page でも作ってみようからなんとなく動的な Page を考えて PHP…ってのはまぁよくある話です。

どこぞのサイトでもお手本にしながら「今は何時何分です」とか、ちょっと頑張って入力データを受け取って「いま入力した値は＊＊です」とかって Page を作って、ちょっと「ああプログラム書いてるんだ」とか実感に浸ったりして…ある意味、一番楽しい時期かもしれません。

んでまぁ、気がつくとお仕事になっちゃったりなんかして。言われたとおりに頑張って PHP スクリプトを組んでみるです。

それがまぁ。ちょっと 1Page、Top Page を動的にごにょごにょとかいう内容だと楽なのですが……だんだん段々ダン Dadan。Page 数が増えてきて…なんか大変になってくるです。

それでもまぁ、特にプログラムの前半とか後半とか、お決まりのパターンってのはあるわけで。

```
mb_language("Japanese");
```

で日本語表示を指定するとか[1] 、なんかよくわからないけど

```
ob_start();
```

っておまじないとか[2] 。毎回使うので、コピペで作れるように「基本ファイル」とかってのをデスクトップに作っておけば、結構便利にコピペしてプログラムが量産できるです。ってか出来てたです。

[1] 日本語以外は？とかいう突っ込みはおぢちゃん嫌いだな。

[2] 賛否両論ありそうだけど。個人的には危険防止のために必須だと思う今日この頃。

そう、あの日までは。

ある日、上司の人がこういったんです。「セキュリティとかあってさ、全部のプログラムに＊＊って命令埋め込んどいて[3]」。

……え？

作ったプログラム、なんだかんだで 100 ファイルくらいあるんですが……全部修正ですか？ 本当ですか？

良くある…ととても嬉しくない話ではあるのですが。現実問題として、業務システムって日々成長するので。結果的に「全体的に変更がかかる」とかそういう類の変更要件ってのは事欠かないものです。

もちろんココで「断固として戦う」姿勢を否定はしないのですが、状況的に止むを得ないモノってのもあるので。そうすると「じゃぁどうやったら楽に変更ができるんだろう？」っていうのが発想として穏便な落としどころかと思うのですがどうでしょうか？

コラム：PHP でも cgi_request 作ろう

いやまぁあちこちで言われているような気がするのですが。

`$_REQUEST`ってのは、POST、GET 以外にも、Cookie やら file やら、色々な情報が盛りだくさんに盛り込まれています。正直、あんまり好ましい状態ではないです。

それ以外でも「POST の情報だけを使いたい」とかいう状況もあるので[a]。

ほかの言語と同様の、cgi リクエストクラスを作って、そこできちんと「GET と POST のみをあつかう」ようにしておきたいものです。

[a] クラス的にサポートしておいてなんですが。get 情報のみを使いたい、ってのは知りうるかぎり皆無に近いほどにレアケースです。

3　先頭の 定型処理は おまとめで

さて。「同じ記述は一箇所に」がプログラム界における常識で定石で通例でパターンってもんです。

幸い、PHP には require [4] というものがあります。

これでプログラムの始めの「毎度おなじみ」な、今までコピペしていた部分を差し替えてあげれば。今回はともかく、今後は随分と楽が出来そうです。

というわけでレッツ作業。……早めに気づいてるともうちょっと楽が出来たんですけどねぇ。

[3] イメージ的には「ゲートウェイ」チックなものを連想してください。

[4] include でもよいのですが、ファイルがないときにエラーにしてくれる文で require のほうがほんの少し優位だと思ってます。

4　考えりゃ 表示の処理も 定型だ

そういえば。出力部分ですが。テンプレートエンジン使ってるおかげで[5]、

```
// 設定した情報を元にして置換処理
$html_data = $convert_engine_instance->convert();
// 置換された HTML データを出力
print $html_data;
```

とかっていう随分とシンプルな…これも定型なので。こいつも一緒にまとめてどっかのファイルに書いておくことにします[6] 。

5　どこまでも 追求したい ナマケモノ

さてさて。ようやっと本題に突入してまいります。

現状、多分こーゆー感じなのではないかと思ってみたりします。

```
include('前処理');
メインの処理
include('出力処理');
include('後処理');
```

んで。

とりあえずまず、業務処理くらいはクラスにしたほうが何かとお便利な世の中なので[7] 。クラス化してみます。

```
class メインの処理クラス
{
  処理が書いてあるとかおもいねぇ
} // end of class

include('前処理');

$main = new メインの処理クラス;
$main->exec();

include('出力処理');
include('後処理');
```

概ねいけるのですが……どう考えても、ほとんどの部分が結局コピペになるのが如何にも予想つきそうなソースです。

というわけで、ここから更に縮めてみます。

---

[5] これはこれで別口に道中を書く予定です。早く読みたい人はファンレターを書きましょう(笑)。

[6] 後半へ向けての恣意的な布石ですので「なんでここだけコード書いてあるの？」とかいう不自然さを気にしてはいけません（笑)。

[7] どうお便利なのかは後ほど出すのでお楽しみに。

```
class メインの処理クラス
{
  処理が書いてあるとかおもいねぇ
} // end of class

class 色々とおまとめクラス
{
public function exec() {
  include('前処理');

  $main = new メインの処理クラス;
  $main->exec();

  include('出力処理');

  include('後処理');
}

} // end of class

$main_instance = new 色々とおまとめクラス;
$main_instance->exec();
```

うんうん。大分いい感じですねぇ。

では、おまとめクラスは良く使いそうなのでファイルに外だししてみましょう。

ついでに、include しても意味がないので、include を取り除いて、てけとうに処理を外だし、別ファイル化してみたりなんかします。

omatome.inc

```
class 色々とおまとめクラス
{
public function exec() {
  前処理

  $main = new メインの処理クラス;
  $main->exec();

  出力処理

  後処理
}

} // end of class
```

hogehoge.php

```
require_once('omatome.inc');
class メインの処理クラス
{
  処理が書いてあるとかおもいねぇ
} // end of class

$main_instance = new 色々とおまとめクラス;
$main_instance->exec();
```

さてさて。これで大分片付いたのですが…このままだと、メインの処理クラスが「常に固定のお名前」になってしまいます。それじゃわかりにくいです

よね？

折角なら、クラス名にも自由を クラス名 is ふりぃだむ。まるでどこかの革命のようですが、これで「いい感じ」になります。やっぱり理解しやすいクラス名って可読性あがりますし。

ただ、そうすると次は「どのクラスを取り込むのか？」って話になるわけでして。これには大きく２つの手法があります。混在もありなのですが。

5.1 せっかくだもの とことん追求

クラス名[8]を、URL 若しくはパラメタから受け取る手法です。

```
class 色々とおまとめクラス
{
public function exec() {
  前処理
  // パラメタを取得
  $param = $cgi_request_instance->find('param');
  // 必要なファイルを require
  require_once($param . ".inc");
  // クラスを作成して叩く!!
  $main = new $param;
  $main->exec();
  出力処理
  後処理
}
} // end of class
```

モジュールの追加が楽なのが特徴なのですが、上述のまんま実装するとロクでもなく巨大なセキュリティホールががら空きになります。

パラメタのサニタイズはきっちりと強力にしておきましょう。そこさえ OK なら、かなり楽が出来るはずです。

5.2 それでも攻撃 忘れられない

クラス名を、URL 若しくはパラメタにある key から解決します。解決するために、特別な設定ファイル[9] を用意します。

[8]大抵、require するファイル名はクラス名から作成できるような作りにするのが通例です。
[9]マップファイルとか呼称しますねぇ。

```
class 色々とおまとめクラス
{
public function exec() {
  前処理
  // パラメタを取得
  $param_name = $cgi_request_instance->find('param');
  // 設定ファイルを取得
  $config_obj = 適当な処理;
  // 必要なファイルを探し出してから require
  require_once($config_obj->get_require_file_name($param_name));
  // クラスを作成して叩く!!
  $main = new $config_obj->get_class_name($param_name);
  $main->exec();
  出力処理
  後処理
}
} // end of class
```

モジュールが増えるたびにマップファイルへの修正は入りますが、セキュリティ的には「なけりゃ動かない」分だけ楽ではあります。

さてさて。今回大活躍しております「色々とおまとめクラス」。ちょいと挙動をまとめてみます。

- 必要な前処理をする
- 必要なファイルを require して、メインの業務クラスを動かす
- 出力処理をする [10]
- 必要な後処理をする

なんか色々と頑張っておりますが。この「なんか頑張ってくれちゃう」クラスの事を、世間様の格好いい言葉で「Controller（コントローラ）」って呼称します。

そう。あの MVC の、C にあたる部分です !! [11]

ちなみに、メインの業務クラスが、MVC で言うところの「Model（モデル）」だったりします。

ほぉら。こうやって見ると簡単でしょ？

ちなみに最後の一つ、View は、実は今回、Controller に入っちゃってる「出力処理」の部分です。確かにこのままでもいいのですが…後で例題は出しますが、実は「別のほうがもっと便利」だったりするので[12] 。ちょいと切り出してみましょう。

[10] 本当はココじゃないんですが、とりあえず説明の流れ上。

[11] 大げさ。

[12] 例えば「今回の業務に限って、出力する HTML を全部ログに吐き出す」とかいう狂った仕様の時に、view クラス作っておけば継承でなんとかなったりします。

ついでに、名前も「それっぽく格好よく」してあげたりします。

```
class mvc_view
{
public function exec() {
  出力処理
}
} // end of class

class mvc_controller
{

public function exec() {
  前処理

  // パラメタを取得
  $param_name = $cgi_request_instance->find('param');

  // 設定ファイルを取得
  $config_obj = 適当な処理;

  // Model クラスの new に必要なファイルを探し出してから require
  require_once($config_obj->get_require_file_name($param_name));

  // Model クラスを作成して叩く!!
  $main = new $config_obj->get_class_name($param_name);
  $main->exec();

  // 出力処理は view だよね
  $view_obj = new mvc_view;
  $view_obj->exec();

  後処理
} // end of class

}
```

大体こんな感じでしょうか？

実際には、ここからいくつか肉付けなどをしていって、実際に「使える」クラスに仕上げていきます。

6　複雑に したいわけでは ないけれど

早々にまとめになって恐縮ですが。

MVC は、概ね上述のような流れで沸いてきたものになります。実際には、ここから更に「こーやるとお便利テクニック」とか色々あるのですが。

とりあえず、お便利テクニックや Controller クラスを「きちんと作りこむ」のは別の原稿に譲るとして。ここでは、MVC が発生した大まかな経緯のようなものを掴んでいただければと思っております。