

フレームワーク作成道中 MVC 編 アドヴァ ンス (笑) Vol 1.0

古庄 道明

1 アドヴァンスに向けて

MVC の基本は以前書きましたが。それだけであれば、ぶっちゃけそんなに「うまうま」ってほどでもなかったりします。

んじゃあ使えないのかってえとなこたあなくて。

アドヴァンスでは、本当に「うまうま」な部分にちょっと切り込んでいこうかと思います。

2 遷移先 複数あると 面倒だ

時々...っていかしょちゅうなのですが。一つの form で「押したボタンごとに遷移先変えたい」とかって要求ありませんか？

「削除」と「修正」とかよくあるですよ。他にも色々ありそうですが。

そんなときこんなとき。普段だとどーゆーコーディングするかっていうと、大抵は一つの Model のなかで if 文で分岐したりして、結構長いソースになりませんか？

```

class model_hogehoge extends model_base {
function execute()
{
  // 前処理
  // 判定
  if (遷移先がこっちなら) {
    こっこの処理
  } else
  if (遷移先があっちなら) {
    あっこの処理
  } else
  if (遷移先がそこなら) {
    そこの処理
  } else
  if (遷移先がここなら) {
    ここの処理
  } else {
    どこの処理?
  }

  // 後処理
}
} // end of class

```

コメントだけだからいいですが、実際にはかな〜りわけのわかんないソースになりますっていうかそんな長いソースはいやです。

長いのは抜け始めてわかる髪だけでいいですから。

おいといて。

こんなときに。ちょっとだけ、こんな風に人は考えます。

```

class model_hogehoge extends model_base {
function execute()
{
  // 前処理
  // 判定
  if (遷移先がこっちなら) {
    $class_name = こっちの処理クラス;
  } else
  if (遷移先があっちなら) {
    $class_name = あっちの処理クラス;
  } else
  if (遷移先がそこなら) {
    $class_name = そこの処理クラス;
  } else
  if (遷移先がここなら) {
    $class_name = ここの処理クラス;
  } else {
    $class_name = どこの処理?クラス;
  }

  // とにかく処理 !!
  $obj = new $class_name;
  $obj->execute();

  // 後処理
}
} // end of class

```

すばらしいこれなら見通しがよいです.....あれ？

んとね。model クラスは、実はその中に色々情報があるので、で、そいつを controller が色々下準備後処理してくれるからこそその「楽が出来る model」なので。

...ダメですこれだとうまくいきません。せっかくいいアイデアなのにアイデア倒れです行き倒れです息にはビオレです。

...であきらめたら人間進歩がありません !!

そうだ。これならどうでしょう？

```

class model_hogehoge extends model_base {
function execute()
{
  // 前処理
  // 判定
  if (遷移先がこっちなら) {
    $class_name = こっちの処理クラス;
  } else
  if (遷移先があっちなら) {
    $class_name = あっちの処理クラス;
  } else
  if (遷移先がそこなら) {
    $class_name = そこの処理クラス;
  } else
  if (遷移先がここなら) {
    $class_name = ここの処理クラス;
  } else {
    $class_name = どこの処理?クラス;
  }

  // とにかく処理!! ...のクラス名を controller に依頼する
  $this->controller に依頼 ($class_name);

  // 後処理
}
} // end of class

```

はい、これはこれで OK なんです。案外にシンプルでしょ？

後は、controller が処理してくれる...はづです。っていうかそーゆー風に作りこめば OK です。

こういう風にするので、

- from から叩かれた時に、情報に従って処理担当の model クラス名を決定する model : 振り分け君
- 実際に処理を行う model : 実処理君

と、別々にプログラムを作成することが出来ます。こうすることで見通しとかよくなるでしょ？

こういうのを、「再帰処理」とかっていいます。

ちなみに、筆者が作っている MagicWeapon というフレームワークの場合、

```

/* 振り分け依頼をする */
// 再帰処理することを宣言する
$this->recursive_on();
// 再帰処理で動かしたいクラス名 (コマンド名) を伝えておく
$this->set_recursive_command($class_name);

// 次の処理に移るために制御を controller に戻す
return true;

```

という書式で、controller に連絡が出来ます。

この方法を使えば、1画面からどれだけ大量の分岐があっても、わりとへっちゃらです。

3 ちょっとまで ログイン処理が面倒だ

さてさて。ユーザログインの Top Page ってのがあります。

まあよくあるパターンなのですが..... ぢつはこれが結構に面倒だったりします。

Top Page は、どう冷静に考えても

- ログイン Page からログイン処理をして遷移
- 他の Page から認証処理を経て遷移

の2パターンがあるのです。

もちろん「同じ Top Page を出力するプログラムを違う model クラスに二重に書く」方法がないわけではないのですが...嫌ですよねぇ？

ここでも、前項でてきた「再帰処理」が役に立つんですねえ。

ちょっとざっくりとコードを書いてみます。

まずはログイン用。

```
class login extends base_model
{
  function execute()
  {
    // ログイン処理
    //
    if (ログイン処理が NG なら) {
      $this->set_body(エラー用の文言);
    } else {
      // ログイン処理が OK なら
      $this->recursive_on();
      $this->set_recursive_command(top page 出力用コマンド
名);
    }
    //
    return true;
  }
} // end of class
```

続いて「他 Page からの遷移用」

```

class any_to_top extends base_model
{
  function execute()
  {
    // 認証処理
    //
    if (認証処理が NG なら) {
      $this->set_body(エラー用の文言);
    } else {
      // 認証処理が OK なら
      $this->recursive_on();
      $this->set_recursive_command(top page 出力用コマンド
名);
    }
    //
    return true;
  }
} // end of class

```

ほ～ら簡単。

これで、あとは「Top Page 出力用クラス」をさくりと作ってしまえばもう
終わり。

こうやってお便利に使っていくものです。

3.1 ちなみに認証は...

はい早速ですが、上のコード、微妙に「うそ」ついてますっていうか無駄
してます。

ログイン系の Page ってほとんどが認証必要ですよ？

そーゆー時は、まずこういうクラスを用意しちゃいます。

```

class base_model_auth extends base_model
{
  abstract function execute_auth();
  function execute()
  {
    // 認証処理
    //
    if (認証処理が NG なら) {
      $this->set_body(エラー用の文言);
    } else {
      // 認証処理が OK なら
      $this->execute_auth();
    }
    //
    return true;
  }
} // end of class

```

そう。エントリーポイントを「execute」から「execute_auth」に変えちゃ
うんです !!

そうすると、先ほどの「他 Page から Top Page への遷移」プログラムは...

```
class any_to_top extends base_model_auth
{
  function execute_auth()
  {
    // 認証処理が OK 「だから」
    $this->recursive_on();
    $this->set_recursive_command(top page 出力用コマンド名);
    //
    return true;
  }
} // end of class
```

とまあ、拍子抜けするほど簡単になっちゃうんです w

ポイントは、継承元のクラスの変更と、エントリーポイントの変更。

具体的には extends が「base_model_auth」になって、function 名が「execute_auth」になってる、って部分。

このささやかな変更だけで、あっというまに「認証が OK にならなきゃこの model は動きません」が出来上がり !!

こんな風に上手に model クラスの継承を設計することで「このメニュー群だけの処理」とかっていう特別も簡単にどうにか出来ちゃったりします。

これもまた、MVC を¹上手につかった、お便利な小ネタになります。

4 きっとまだまだ

きっとまだまだ他にもお便利さんが隠れてるはず w

どこかでこっそりネタが増えるかもしれないので...なにかあったら教えてくださいね ^^

¹特に「Model がクラスである」ことを