


Complex Views with the Zend Framework - Part 2: View Helper Pattern

Part two of my ongoing look at the View layer of the Zend Framework turns its attention to the topic of View Helpers. The Zend Framework manual provides a fairly narrow definition of its helpers which indicates they enable complex tasks, like generating form elements, to be extracted out of views into dedicated helpers. Here I'll try to explain in greater depth the View Helper pattern which is another of those patterns in the J2EE catalog, and which adds to the range of tasks View Helpers are capable of performing.

The simplest explanation is in the separation of layers enforced by the Model-View-Controller pattern. In a typical web application using the Zend Framework, a URI is used by the framework's Controller architecture to select an action on a `Zend_Controller_Action` class to execute. This action method will then normally access the application's Model to fetch any data it requires, operate on that data, and pass a subset of it to the View (for use in rendering templates). It's clear to see so far, that the View by default relies on the Controller to pass it the data model it needs when rendering templates.

As in my previous post however, the View may also include any number of elements (header, footer, widgets, etc.) common to ALL pages. Many of these may require data of their own.

So what is the problem? Anytime a partial View needs extra data (the View's Model) it needs to push calls to extra Controllers (following the current practice for the framework) in order to get that Model. This involves yet another complete dispatch cycle, with any number of classes, plugins, and operations involved. Yet in most cases this is completely unnecessary - why not just let the View request data from the Model directly? 

In simple terms, the View Helper acts as a middle-man sitting between the View and the Model. It's job in our scenario is to replace the need to nest Controllers, and give that nesting/layout control back to where it belongs - the View layer. When a View (or partial View) requires Model data not supplied by the current controller, it calls upon a View Helper to go fetch that data independently of any Controller.

Let's take a simple example. While building a blog, we've decided to add the last 5 entry titles from Planet PHP to the bottom of the page - every page in fact. In keeping with promoting reusability we create a generic `Zps_View_Helper_Rss` class which can consume RSS, and to which we can add extra methods in the future should they be needed for other Views. This is slightly different to the current view helper description in the framework manual - here we can offer a full selection of public methods forming a fuller (read-only) interface to the underlying Model - the RSS XML.

```
require_once 'Zend/Uri.php';
require_once 'Zend/Feed/Rss.php';
class Zps_View_Helper_Rss
{
    protected $_channel = null;
    public function __construct($url)
    {
        if (!Zend_Uri::check($url)) {
            throw new Exception('RSS URL is invalid!');
        }
        $this->_channel = new Zend_Feed_Rss($url);
    }
    public function getTitles($number)
```

```

{
    $titles = array();
    $count = ;
    $number = intval($number);
    // Zend_Feed_Abstract implements Iterator!
    while ($count <= $number && $this->_channel->valid()) {
        $titles[] = $this->_channel->current()->title();
        $count++;
        $this->_channel->next();
    }
    return $titles;
}
}
}

```

Now comes our sample template - one of those recurring Views:

```


<?php $feed = new Zps_View_Helper_Rss('http://www.planet-php.net/rss/'); ?>
<div class="feedtitles">
<ul>
<?php foreach($feed->getTitles(5) as $title): ?>
<li><?php echo $title ?></li>
<?php endforeach; ?>
</ul>
</div>

```

And not a controller in sight... 

Of course the Model can equally be running from any datasource including the database, so there's a lot of flexibility for these View Helper classes. I haven't done so here, but you can probably still fit them into the current view helper convention with the single public method (in our case rss()) just returning instances of the object.

Of course, there really are times when the recommended Controller _forward() (or a viable alternative) can still be useful. Reliance on plugins was suggested on the mailing list, for ACL perhaps. But the point here is that controller nesting is not required except for those exceptional cases where a View Helper just won't cut it.

Edit: I figured it be save confusion by noting this helper could equally run from a cache of the selected RSS source which is updated separately at regular intervals (hail to cron!). Save the server making the trip on every request 


Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 06:34

You're not using the Helpers correctly. 

First off, change your __construct() method to rss(). Second, in your bootstrap or in a controller, add the path to the directory containing your custom helpers via \$view->addHelperPath() (tip: you can pass a class prefix as the second argument, which allows you to use your own namespace for helpers).

Once you've done the above steps, you can make the call much more simply in your view script:\$feed = \$this->rss('http://www.planet-php.net/rss/');


Another thing: you mention that the view sometimes needs access to the model, and you've shown one way here via a helper. Something else you can do is pass models to the view from your controller, or a plugin, or your bootstrap, or... The point is, there's no limitation on what types of variables you pass to the view, so you can pass whatever objects you need to have access to. You might even try creating a view helper that can load model objects on the fly for you... Anonymous on Apr 21 2007, 20:17

Third paragraph from the end 


. I was just demonstrating a generic View Helper which pulls from a Model instead of relying on it being pushed from the Controller level, and left it to the reader to implement it in the ZF style. While this is not in the ZF convention - crazy enough it still works... The ZF convention is just that, a convention.

I already noted the controller method. Propagating Models across nested Views is also possible from the bootstrap/other but View Helpers are a cleaner than other methods since the Views just go get data as needed in a very visible accessible fashion.

The point I was making here is that there is no reason to stop the View directly querying the Model layer when it's needed. There are a lot of people using the framework who don't see that and run off to nest controllers needlessly. A short manual note would go a long way...

Are you summarising me with that last sentence? 

Anyway, one important point is that the View Helper is needed for one important separation task (besides simplification and integration of Model interfaces which by itself is good justification for using it) - Views should not be able to write to the Model, only read from it. Keeps writing to Models isolated to Controllers where it belongs. Anonymous on Apr 22 2007, 02:28

The fact that Zend_View uses PHP as the templating language is one of its greatest strengths, I feel (and I'm sure Paul M. Jones will agree, as he developed the original implementation, basing it off of his Savant package). Because it uses PHP, you have access to everything PHP has to offer -- including libraries you've written. I was merely trying to point out a more ZF appropriate way to use helpers. 

I've typically shied away from pulling directly from models in my view layer, though Fowler clearly shows this intention in his discussion of MVC in POEAA. In some recent projects I've been working on, however, I can definitely see the use for it, and you're definitely making a good case for it (and showing how Zend_View can make it easily possible).

One note: in Fowler's discussion of MVC< he does note that typically you should not do anything that alters your model directly from the view -- i.e., you wouldn't want to do anything that would insert, update, or delete data, only pull data from the model. Anonymous on Apr 22 2007, 02:40

Should note I edited my comment after it was initially posted to improve it. I noted the same point also in the edited last paragraph. I didn't think you'd reply so fast while I was still typing the edits! 

I do agree - I very much prefer my templates in PHP. Tagsoup does have it uses when you're not fully in control of who's editing the templates - but generally I don't see that scenario very often. Anonymous on Apr 22 2007, 02:51

what if we are using something like Smarty as a template system?
and we dont want any php coding in the view at all?
how would we access the models from there if we can't initialize it from the view?

and I don't want to initialize every models i have in the controller and pass to the view.

i want to dynamically call the model from the view.

is this possible? Anonymous on Apr 28 2007, 10:45

You can achieve this within Smarty using a custom function - check the documentation for "custom functions" and check out the examples Smarty bundles in "plugins". These can be implanted into templates as function tags which the compiler translates into the relevant function call with parameters.

Adding the Model this way is possible, but not very clean since the smarty bits are procedural functions, not OO View Helpers. If you are comfortable doing so you can use them mainly to create new View Helper instances inside the function as static variables (Singleton style), which are written as in my example, or ZF's view helper outline.

The function is then just a Proxy, using a specific Smarty tag parameter to hold the method name to call on the View Helper object. If you took mt RSS helper, a Smarty function for this could be called "smarty_function_rss" and have the following tag type:

```
{rss method='getTitle' number='5' assign='titles' url='http://www.planet-php.net/rss/'}
```


Might not make perfect sense to you right now, but check the Smarty source for the function files in its plugin directory. This function would assign the 5 headlines to the template variable called \$titles as an array. Since all functions get a reference to the Smarty object, that as easy as

```
$smarty->assign('titles', $resultFromViewHelper);
```

Could write a whole blog post about this but this will get you started in the right direction. Anonymous on Apr 28 2007, 18:17

"The point I was making here is that there is no reason to stop the View directly querying the Model layer when it's needed. There are a lot of people using the framework who don't see that and run off to nest controllers needlessly."


I read directly from the model in view helpers too when necessary as otherwise you run the risk of the controller action acting as a proxy to the view for stuff that is completely unrelated to the current action. One solution is of course to use `_forward()` which requires a linear chain to be constructed and I find less flexible.


(It's just as easy in Smarty as in Zend_View too 
)

Regards,

Rob...

(finally catching up...) Anonymous on May 5 2007, 15:02

"(It's just as easy in Smarty as in Zend_View too 
)"

Yep, I have a comment for this entry giving a brief overview of adding suitable tags to the Smarty compiler . Would work for Template

Lite also. Anonymous on May 5 2007, 17:47

Could i suggest you add a link to the other parts of this series at the bottom of each part. (I getting dizzy trying to find part one)
Anonymous on Aug 21 2007, 07:41

I'm not sure if I like this. It defeats the whole purpose of MVC despite increased complexity compared to your View_Helper solution.
Anonymous on Aug 27 2007, 11:00