Zend_FilterChain Zend_Validator - Christopher Thompson

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_FilterChain Zend_Validator Component Proposal

Proposed Component Name	Zend_FilterChain Zend_Validator
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_FilterChain Zend_Validator
Proposers	Christopher Thompson, author Darby Felton, Zend liaison
Revision	1.1 - 25 June 2006: Created. (wiki revision: 11)

Table of Contents

1. Overview

Zend_FilterChain is a class to apply any number and/or combination of Filter objects to data in a container. Zend_Validator is a class to apply any number and/or combination of Rule objects to data in a container. Typically these are applied to the data in a Request container as part of a Input or Form Controller.

2. References

- Original Proposal for ZF 0.2

3. Component Requirements, Constraints, and Acceptance Criteria

- FilterChain class allows any number and/or combination of Filter objects to be added to the chain
- Filter classes are polymorphic allowing unlimited variety of custom filters
- FilterChain applies filters to specified data in the container passed to it

- Validator class allows any number and/or combination of Rule objects to be added to the chain
- Rule classes are polymorphic allowing unlimited variety of custom rules, each defining an error message
- Validator applies rules to specified data in the container passed to it and returns the error messages for failed rules
- Validator itself implements the Rule interface so may be used as a Rule

4. Dependencies on Other Framework Components

- Zend_Exception

5. Theory of Operation

OVERVIEW

The FilterChain/Validator style is an alternative to the simpler InputFilter style. The InputFilter is excellent where you want direct access to filtered data. Its procedural style is very easy to use. Conversely, the FilterChain/Validator style provides a foundation for things like Form and Application Controllers. These allow the programmer to be more focused on specifying the Rules and Filters needed and lets the controller deal with the actual program flow. They also ease creating scaffolding.

The thing I immediately noticed about the InputFilter class is that it is really a Container. That is one of its strengths. Containers tie frameworks together. The most common PHP Container is keyed access to data, which is the class/object form of PHP's associative arrays. Filters and Validators need a simple Container to provide structured access to data. The Container needed has only the bare bones capabilities taken from the InputFilter's constructor and get raw data methods.

The simple Container class, probably something like Zend_Http_Request would be the usual container used. This Container could be enhanced by implementing the magic accessor functions, but I have kept it simple in the example.

The FilterChain and Validator need polymorphic Filters and Rules. In the example I split-up the current Zend_Filter class into a separate class for each Filter and Rule – with doFilter() and isValid() methods respectively. These can be separated the Filters and Rules into two separate files or into a separate file for each Filter and Rule.

It should be noted that the classes presented here are the minimal case and that additional list management or other functionality may easily be added. Also note that the current Filter and InputFilter class could easily be build on top of these individual Filter and Rule classes to centralize tests.

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Filter_Abstract.php
- Zend_Filter_*.php
- Zend_Rule_Abstract.php
- Zend_Rule_Abstract.php
- Zend_Validator.php
- Zend_Filterchain.php

8. Use Cases

FILTERCHAIN

```
$filter = new Zend_FilterChain($post);
$filter->addFilter('field1', new Zend_Filter_Int());        // remove all non-numeric
characters
$filter->addFilter('field1', new Zend_Filter_Range(1, 100));       // limit to a range
of numbers
$filter->addFilter('field2', new Zend_Filter_Alpha());    // remove all non-alpha
characters
  // regexp to remove all characters not valid for basic email addresses
$filter->addFilter('field3', new Zend_Filter_Regexp('/[^a-zA-Z0-9\@\.\-\_]/', ''));
$filter->doFilter();
```

VALIDATOR

```
$validator = new Zend_Validator($post);
$isnullrule = new Zend_IsNull();
$validator->addRule('field1', $isnullrule, 'please enter field1');
$validator->addRule('field1', new Zend_Rule_Int(), 'not an integer');
$validator->addRule('field2', $isnullrule, 'please enter an email address');
$validator->addRule('field2', new Zend_Rule_Email(), 'not a valid email address');
$validator->addRule('field3', new Zend_Rule_Range(10, 20), 'number not in range
10-20');
if ($validator->isValid()) {
    echo 'VALID</br>';
} else {
    echo 'NOT VALID</br>';
}
```

- it should be noted that combining the above two example is the basis for a Form Controller class

9. Class Skeletons

```
class Zend_FilterChain
{
    public function __construct($source = NULL) {}
    public function addFilter($key, $filter) {}
    public function doFilter() {}
}
```

```
class Zend_Filter_*
{
   public function doFilter() {}
}
```

```
class Zend_Validator
{
    public function __construct($source = NULL) {}
    public function addRule($key, $obj, $msg='') {}
    public function isValid() {}
    public function getErrorMsg($key=NULL) {}
}
```

```
class Zend_Rule_*
{
    public function isValid($value) {}
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>