Validation Filter

From ICPCWiki

The validation filter is a Servlet Filter (http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/Filter.html) that intercepts requests for configured pages and applies a series of Validators to them.

Contents

Prerequisites

The filter relies on HTML forms having a static name and static field names. Each form to be validated must have a name atttribute.

```
<form name="SomeName" ...
```

and

```
<field name="StaticFieldName" ...
```

Configuration

Two files configure the `ValidationFilter`:

1. icpc-validation.properties
2. validation.xml

Both files must be in the JVM class path. Normally, it is suficient to place them in Tomcat's *common/classes* directory.

icpc-validation.properties

The properties file allows you to configure the external behaviors of the Filter. In other words, things like the name of the special hidden form field that is injected, where scripts are cached, which dataSource to use, and so on. The basic properties are described here:

hiddenField.name
: the name of the hidden form field injected by the filter. The default should never collide with anything on any form, so you shouldn't need to change this.

configFile
: the default is `validation.xml` but this could potentially colide with some other 3rd party validator, so you may change it.

sessionfieldsMap.var
: the name of the session variable to store the fields data in between the display of the form and validation.

cache.scripts
: whether or not to store the Javascript in separate files so that it can be cached. This should be turned on if at all possible, since it will greatly reduce your bandwidth usaged.

cache.scripts.dir.url
: the URL (relative to your server) where the scripts are stored. e.g.

```
/common/scripts
```

cache.scripts.dir.fs
: the full path to the directory on your websever where the scripts will be stored. e.g.

```
/opt/jakarta-tomcat-5.x.x/webapps/common/scripts
```

cache.scripts.prefix
: a prefix to add to the script filenames to avoid collisions with other files in that directory.

dataSource
: a data source configured in web.xml for the validators to use. The full path might be *java:comp/env/jdbc/icpc*, so you would specify *jdbc/icpc* as the data source.

validation.xml

This file (name may be different, see above) allows you to configure the internal behavior of the filter.

You file should look similar to the text below. (**NOTE:** This is a contrived example and is not necessirly consistent. There is much more to the real file.)

Validator Elements

Various Validators are configured below. We have chosen one of each class of Validator that comes packaged with the filter. The validation file starts with `<validation>`. After that, the validators are configured.

```
<validation>
    <validators>
        <validator name="IsLength1_40" trim="false"
            class="edu.baylor.icpc.validation.validators.IsLength">
            <argument value="1" />
            <argument value="40" />
        </validator>
```

Each validator tag must uniquely name the validator, provide its class, and the arguments to it's constructor, in order (All constructor arguments must be Strings). Optionally, you can configure the validator to trim it's input before validating. **NOTE:** By default, validators trim their input, however they will not alter request values, so it is up to your business logic to trim the values in the request before committing them.

```
        <validator name="IsEmail"
            class="edu.baylor.icpc.validation.validators.IsEmail">
            <message value="Please enter a valid email address." />
        </validator>
```

All validators should have a default message, but that message can be overridden by providing a `message` element to the

validator

```
        <validator name="ACMID"
            class="edu.baylor.icpc.validation.validators.MatchRegEx">
            <argument value="[0-9]+|FREE" />
            <message value="Please enter your ACMID or 'FREE' if you do not have one." />
        </validator>
        <validator name="MF"
            class="edu.baylor.icpc.validation.validators.IsInEnumeration">
            <argument value="M,F" />
            <argument value="false" />
        </validator>
        <validator name="IsMajor"
            class="edu.baylor.icpc.validation.validators.IsInTable">
            <argument value="Major" />
            <argument value="Major" />
        </validator>
        <validator name="RadioSelected"
            class="edu.baylor.icpc.validation.validators.RadioSelected" />
```

You can see the *RadioSelected* validator takes no arguments. It is a client-side only validator responsible for ensuring that a radio button has been selected.

```
    </validators>
```

Form Elements

A form must have a name, otherwise the filter will be unable to find it. The name attribute specified here should match the name attribute of the form in HTML.

```
    <forms>
        <form name="Institution">
            <field name="SchoolName">
                <validator name="IsLength1_60" />
                <message value="Please enter the name of your school." />
            </field>
```

A form is a collection of field elements. The filter will get the value for each named field and run it through each validator configured for that field. Notice that it is possible to specify field-level messages, to override the validator message(s) if a field does not pass validation. In other words, if *SchoolName* is not provided, the message displayed will not be the message configured for *IsLength*, but will be "Please enter the name of your school."

```
            <field name="SchoolShortName">
                <validator name="IsLength1_20" />
            </field>
        </form>
    </forms>
```

Page Elements

A page is the last step in configuring the filter. Most pages will only have one form that needs validation, but it is possible to have more than one form per page.

```
    <pages>
        <page name="/admin/EditInstitution\.jsp" id="EditInstitution">
            <!-- name refers to the form defined above -->
            <form name="Institution" />
        </page>
        <page name=".*/EditCoach.jsp" id="EditCoach">
            <form name="Person">
                <not-field name="ExpectedGradDate" />
            </form>
        </page>
        <page name=".*/EditPerson\.jsp|.*/ValidatePerson\.jsp" id="EditPerson">
            <form name="Person">
                <field name="Name" />
                <field name="Phone" />
            </form>
        </page>
    </pages>
</validation>
```

Validators

Built-In Validators

There are several validators that come with the framework. The most flexible is `MatchRegEx`. It takes a regular expression as the constructor argument and generates client and server side code to validate using that expression. Because it uses the same regular expression on both sides, you need to make sure that any regular expression you give it behaves the same way in Java and in Javascript. in particular, if your expression results in a Javascript error, client side validation will not occur.

Custom Validators

To implement your own Validator, you need to only implement the `Validator` interface. There is also an `AbstractValidator` base class. If you do inherit from `AbstractValidator` you will need to implement the functions

```
public void validate(String value, ValidatorResult result);
protected void recreateScript();
```

The first is the server side validate function, the second generates a `JSFunction` object from the `name` and `message` members. It will be called whenever the `name` or `message` message field is changed via their setters.

- This page was last modified 14:36, 21 February 2006.