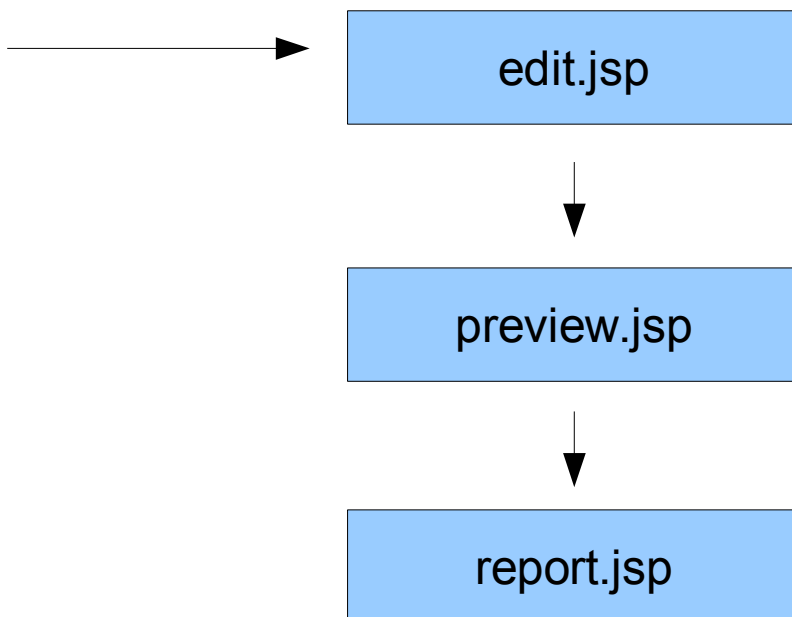


Overview

Simple Page Flow (SPF) is a small utility library you can use within your web application to control page flow.

Let's assume you have wizard like forms in your web application. Ususally you don't want users to navigate directly to the last view of your wizard by just typing in the URL in the browser. But making sure the workflow is followed correctly is not that easy. At least this is how many web developers feel.

Here is an example of such a workflow:



What prevents the user to navigate to the report.jsp view directly? Actually it's the developers responsibility to make sure this doesn't harm the system and the application still behaves correctly.

One day I was thinking about a reusable, framework independent solution to that problem. I thought of the pageflow as a state machine, where each view has transistions to other views. So a state machine has a defined starting point and with each new request the machine decides if the next step is valid or not.

But how must it be designed to be framework independent? It should work with JSP, Servlet, pure HTML, JSF, Struts, etc. The simplest solution was to design it as a servlet, that decides if the requested URL is allowed or not. In an environment where redirects are used this works pretty well, but how can we handle forwards? Of course the PageFlow object can handle this, the question is more if the filter will be enough. For JSF for instance I'm not sure this works as expected (at least when you do not use redirects). Maybe only GET calls should be filtered and posts are always accepted. Nevertheless I think the core of the SPF project can be used in a lot of applications and thanks to great open source projects (Commons Digester!) I had more or less no development time. Thanks :).

Installation and Usage

Installing the SPF and using it is very straight forward.

First we need to register the filter in our web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <filter>
    <filter-name>PageFlowFilter</filter-name>
    <filter-class>ch.menzsoft.pageflow.PageFlowFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>PageFlowFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
  </filter-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Next step is to add the required libraries to your project. Here is a list of JARs you'll need.

- Commons-digester 1.8
- commons-collections 3.2
- commons-logging 1.1
- commons-beanutils 1.7.0

At least this is the configuration I use. Note the only direct dependency used by SPF is the Digester.

Ok, now we can create our pageflow configuration file. Usually this is stored in WEB-INF/pageflow.xml. Of course you can override the location by setting up an init parameter for your filter named **CONFIG_LOCATION**.

Here is a sample configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<page-flow>
  <!--
  Define your own matcher class. Note this is optional, the RegexPatternMatcher is the default anyway.
  -->
  <pattern-matcher-class>ch.menzsoft.pageflow.RegexPatternMatcher</pattern-matcher-class>

  <!--
  here you can define any number of entry pages. Note that there are no conditions for entry pages
  -->
  <entry-views>
    <entry-view>
      /index.jsp
    </entry-view>
  </entry-views>
</page-flow>
```

```
</entry-view>
</entry-views>
```

```
<!--
```

Define your exclusion list, any regex pattern is allowed (depends on the pattern matcher!)

```
-->
```

```
<exclusions>
  <exclude-pattern>
    .*gif
  </exclude-pattern>
  <exclude-pattern>
    .*jpg
  </exclude-pattern>
</exclusions>
```

```
<!--
```

As you can see rules can also contain wildcards, if any rule fails navigation is prohibited.

*The wildcard * means everything. One could also type .* which is valid regex syntax.*

The sample below says that it is possible to navigate to /fourth.jsp no matter what the previous view was.

A custom condition checks if the navigation should be allowed.

```
-->
```

```
<rule>
  <from-view>
    *
  </from-view>
  <to-rule>
  <to-view>
    /fourth.jsp
  </to-view>
  <custom-condition>
    <custom-condition-class>
      ch.menzsoft.pageflow.DemoCustomCondition
    </custom-condition-class>
  </custom-condition>
  </to-rule>
</rule>
```

```
<!--
```

Another simple example shows how to use the attribute condition which checks if the specified attribute is in the session of the user and has the expected value. Note the expected value is optional.

```
-->
```

```
<rule>
  <from-view>
    /index.jsp
  </from-view>
  <to-rule>
  <to-view>
    /second.jsp
  </to-view>
  <attribute-condition>
    <attribute>TEST</attribute>
    <expected-value>value</expected-value>
    <scope>session</scope>
  </attribute-condition>
  </to-rule>
</rule>
```

```
<!--
```

Simple example. From the view /second.jsp you can navigate to /second.jsp (yes this is needed), /third.jsp and /first.jsp. In conjunction with the first rule (see above) also navigation to /fourth.jsp is allowed (depending on the result of the custom condition).

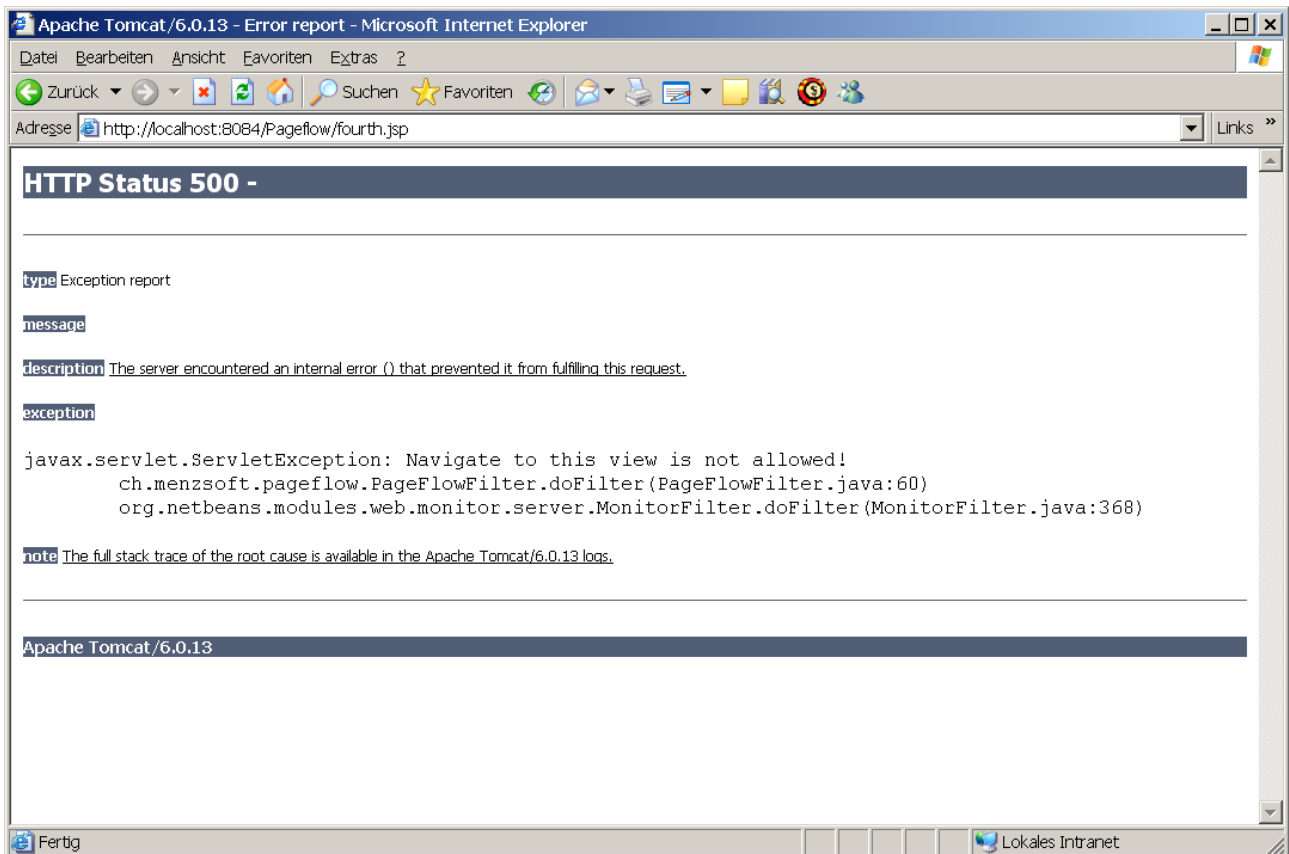
```
-->
```

```
<rule>
  <from-view>
    /second.jsp
  </from-view>
  <to-rule>
    <to-view>
      /second.jsp
    </to-view>
  </to-rule>
  <to-rule>
    <to-view>
      /third.jsp
    </to-view>
  </to-rule>
  <to-rule>
    <to-view>
      /first.jsp
    </to-view>
  </to-rule>
</rule>
</page-flow>
```

I think the XML is very easy to understand and the comments explain what happens. One speciality is that a single request can match multiple rules! All must return ok in order to navigate to the desired view.

Another thing is that a single to-view can contain more than one condition!

Having a basic setup you can try to navigate to a page that is not in the entry views and see what happens.



If you want to try the sample application with all its source code and libraries I have setup a NetBeans project (using NB6.0, not sure if you can open it with 5.5).

Note that this is not open source. If there is enough feedback and interest in the project maybe I will start an open source project. I would appreciate if you would send me a little e-mail if you intend to use the code in your own project. Thank you.

Please send me your feedback!