TapestryFasttrackForStrutsProgrammers

WARNING: This page refers to an older version of Tapestry. For Tapestry 5 see http://tapestry.apache.org/

Foreword

I am in the process of converting from Struts (1.2) to Tapestry (4.0) right now. I found the Tapestry documentation lacking in several places, namely about HiveMind, how to provide consistent desing for all pages as in Tiles etc. So I provide here the steps needed for transition from Struts to Tapestry, hoping that it will help somebody in the same situation.

(This page was started by EdYu, but left mostly empty for two years, so I am going to write it. MartinKuba)

Tapestry advantages over Struts

You need much less writing compared to Struts. Imagine the typical example of a page with a form for editing a database record. In Struts, you need to create:

1. a **form bean** for holding data entered into the form, i.e. a Java class extended from ActionForm
2. an **action** for getting the record from a database and prefiling the form bean
3. a **JSP page** for displaying the form
4. a second **action** for storing the modified data back into the database
5. a **<form-bean>** definition in struts-config.xml
6. two *<action>*definitions in struts-config.xml
7. a page **<definition>** definition in tiles-config.xml

That's a lot of typing, and in many places, which must be synchronized. In Tapestry, you need to write:

1. a form bean
2. a page template (MyPage.html)
3. a page class (MyPage.java)

That's all ! No configuration file needed, if you use annotations. How is that possible ? Because the page and the class have the same name (MyPage), they belong together, so you don't need the <action> definitions from Struts. You can place the code of both two actions, form initialization and data saving, into just one class instead of two. The form bean used is specified in the code, so you don't need the <form-bean> definition. And you don't need the Tiles definition, because page composition is given by the page template.

Application setup

Here is what you need to do to create a new web application in Tapestry. First you need the following JAR files in the WEB-INF/lib/ directory:

```
WEB-INF/lib/commons-codec-1.3.jar
WEB-INF/lib/commons-fileupload-1.1.jar
WEB-INF/lib/commons-io-1.1.jar
WEB-INF/lib/commons-logging-1.0.4.jar
WEB-INF/lib/hivemind-lib-1.1.1.jar
WEB-INF/lib/hivemind-1.1.1.jar
WEB-INF/lib/javassist-3.0.jar
WEB-INF/lib/junit-3.8.1.jar
WEB-INF/lib/log4j-1.2.13.jar
WEB-INF/lib/ognl-2.6.7.jar
WEB-INF/lib/oro-2.0.8.jar
WEB-INF/lib/tapestry-annotations-4.0.2.jar
WEB-INF/lib/tapestry-contrib-4.0.2.jar
WEB-INF/lib/tapestry-4.0.2.jar
```

Then, if you want to use the *friendly URLs* in Tapestry (and I bet you want) you need your WEB-INF/web.xml looking in this way:

```
<?xml version="1.0" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
    <servlet>
        <servlet-name>app</servlet-name>
        <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <filter>
        <filter-name>redirect</filter-name>
        <filter-class>org.apache.tapestry.RedirectFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>redirect</filter-name>
        <url-pattern>/</url-pattern>
    </filter-mapping>
    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>/app</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>*.direct</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>*.sdirect</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>*.svc</url-pattern>
    </servlet-mapping>
</web-app>
```

You also need a file WEB-INF/hivemodule.xml with the following content:

```
<?xml version="1.0"?>
<module id="mytapestryapp" version="1.0.0">
    <!-- for friendly URLs -->
    <contribution configuration-id="tapestry.url.ServiceEncoders">
        <page-service-encoder id="page" extension="html" service="page"/>
    </contribution>
    <contribution configuration-id="tapestry.url.ServiceEncoders">
        <direct-service-encoder id="direct" stateless-extension="direct" stateful-extension="sdirect"/>
    </contribution>
    <contribution configuration-id="tapestry.url.ServiceEncoders">
        <extension-encoder id="extension" extension="svc" after="*"/>
    </contribution>
</module>
```

And the last configuration file is named WEB-INF/app.application, and you need to specify here the Java class packages, which will be searched for pages and components. I also specify the template encoding here, as I need to write accented characters in my native language.

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC
        "-//Apache Software Foundation//Tapestry Specification 4.0//EN"
        "http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">

<application name="My application">
    <description>My Wonderful Application</description>
    <meta key="org.apache.tapestry.page-class-packages" value="com.mydomain.myproject.pages"/>
    <meta key="org.apache.tapestry.component-class-packages" value="com.mydomain.myproject.components"/>
    <meta key="org.apache.tapestry.template-encoding" value="iso-8859-2"/>
</application>
```

If you want to internationalize your application, it means to provide localized texts in several languages, you can create ResourceBundle files
*WEB-INF/app.properties
*WEB-INF/app_en.properties
*WEB-INF/app_cs.properties

- ...

which will hold the localized texts common for the whole application. You can also put the localized texts into files specific to each page or component.

That's the whole configuration.

Component for accessing JSPs

You can create a component which displays a JSP page, thus serving as a bridge between Tapestry and Struts. This component has no template, so it is extending AbstractComponent class. You can call it as

```
<span jwcid="@JspPage" include="/somepage.jsp" />
```

i.e. specifying the JSP page to include. Here is the code:

```
package com.mydomain.myproject.components;

import org.apache.tapestry.AbstractComponent;
import org.apache.tapestry.IMarkupWriter;
import org.apache.tapestry.IRequestCycle;
import org.apache.tapestry.annotations.ComponentClass;
import org.apache.tapestry.annotations.InjectObject;
import org.apache.tapestry.annotations.Parameter;
import org.apache.tapestry.services.RequestGlobals;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * Tapestry component for displaying a JSP page.
 *
 * @author Martin Kuba makub@ics.muni.cz
 */
@ComponentClass(allowBody = false, allowInformalParameters = false, reservedParameters = "include")
public abstract class JspPage extends AbstractComponent {

    @Parameter(name = "include", required = true)
    public abstract String getInclude();

     @InjectObject(value = "service:tapestry.globals.RequestGlobals")
    public abstract RequestGlobals getRequestGlobals();

    protected void renderComponent(IMarkupWriter writer, IRequestCycle cycle) {
        RequestGlobals globals = getRequestGlobals();
        HttpServletRequest request = globals.getRequest();
        HttpServletResponse response = globals.getResponse();
        try {
            request.getRequestDispatcher(getInclude()).include(request, response);
        } catch (ServletException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Consistent page layout as with Tiles

If you used Tiles with Struts, the first thing you want to do after application setup is to create a common layout for all pages. In Tapestry, this can be done by creating a component, which all pages will use. Usually the component is called @Border, but I prefer to call it **@MyBorder** as the layout is diferent for each application.

Create a component class MyBorder in the package specified for components:

```
package com.mydomain.myproject.components;

import org.apache.tapestry.BaseComponent;
import org.apache.tapestry.IAsset;
import org.apache.tapestry.annotations.ComponentClass;
import org.apache.tapestry.annotations.Parameter;
import org.apache.tapestry.annotations.Asset;

@ComponentClass
public abstract class MyBorder extends BaseComponent {

    @Parameter(name = "title-key")
    public abstract String getTitleKey();

    @Asset("/style.css")
    public abstract IAsset getStylesheet();

    public String getTitle() {
        if (isParameterBound("title-key")) {
            String titleKey = getTitleKey();
            return getMessages().getMessage(titleKey);
        } else {
            return "";
        }
    }
}
```

and the corresponding template in WEB-INF/MyBorder.html (I don't like JavaScript, because 15% of users have it switched off, so I use just @Shell, but not @Body):

```
<html jwcid="@Shell" title="ognl:title" stylesheet="asset:stylesheet">
 <body>
  <h1><span jwcid="@Insert" value="ognl:title">Page title</span></h1>
  <ul>
      <li><a href="#" jwcid="@PageLink" page="SomePage1">some page</a></li>
     ... whatever navigation menu you need ....
  </ul>
  <span jwcid="@RenderBody">Page content goes here.</span>
 </body>
</html>
```

In every page then use:

```
<html>
  <head>
      <link type="text/css" rel="stylesheet" href="styl.css" />
  </head>
  <body jwcid="$content$">
  <div jwcid="@MyBorder" title-key="title_for_this_page_key">
... page specific content ...
  </div>
</body>
</html>
```