Navigation

Wildstar Technologies Notes on the Java Platform

Sitemap

Recent site activity

Configuring JavaServer Faces 2.1 to run on the Google App Engine Using Eclipse
edited by Derek Berube
attachment from Derek Berube

HOW TO Configure Eclipse to Run With Java 7 When Java 8 Is Installed
edited by Derek Berube
attachment from Derek Berube

HOW TO Configure Eclipse to Run With Java 7
edited by Derek Berube

Mac OS X Java Development
edited by Derek Berube

HOW TO Configure Eclipse to Run With Java 7
edited by Derek Berube

View All

Configuring JSF 2.2 to run on the Google App Engine Using Eclipse

By: Derek Berube, Wildstar Technologies, LLC.
Last Update: July 10, 2014

Overview

This tutorial will guide users through the process of developing and deploying a JavaServer Faces 2.2.4 application on the 1.9.6 version of Google's App Engine for Java using the 2.2.4 version of Oracle's reference implementation of the specification. The application built out in the course of writing this tutorial is available from http://jsf22template.wildstartech.com/. The source code is available from the jsf22template Google Code project.

The complete Eclipse project is contained in the `jsf22template.zip` archive which is accessible from the following URL:

> http://www.wildstartech.com/download/jsf22template.zip

One of the key features of the JavaServer Faces 2.2 specification is the ability to use annotations to define managed beans rather than using the `faces-config.xml`. While the annotations provided by JSF 2.2 are fantastic, you may consider taking advantage of the capabilities offered by the Contexts and Dependency Injection (JSR-299) (CDI) framework (the JSR was referred to as "Web Beans" prior to January 26, 2009). CDI provides similarly named `@ApplicationScoped`, `@SessionScoped`, `@RequestScoped` annotations (part of the `javax.enterprise.context` package) but uses a `@Named` annotation in favor of the JSF 2.0 `@ManagedBean` annotation. CDI also introduces the `@ConversationScoped` annotation which is familiar to Seam aficionados and provides a variable-length scope that can span requests but isn't as long-lived as either the Session or Application scope. If you are interested in incorporating the CDI framework into your App Engine project, have a look a the "Configuring JBoss Weld to Run with JavaServer Faces on the Google App Engine" after completing this tutorial (currently in a draft state).

Software Requirements

In order to build a JavaServer Faces 2.2 application and have it run on the Google App Engine platform, you will need to have the following software downloaded and installed locally.

- The Eclipse Platform
- Google Plugin for Eclipse
- Oracle JavaServer Faces 2.2 Reference Implementation
  - javax.faces-2.2.4.jar
  - javax.faces-2.2.4-sources.jar
  - javax.faces-api-2.2-javadoc.jar
- Unified Expression Language 2.2 API (api [el-api.jar] implementation [jboss-el.jar])

Optional Software

You can download the Google AppEngine SDK v1.9.6 and install it locally on your system; however, this tutorial is written from the perspective of using the Google Plugin for Eclipse.

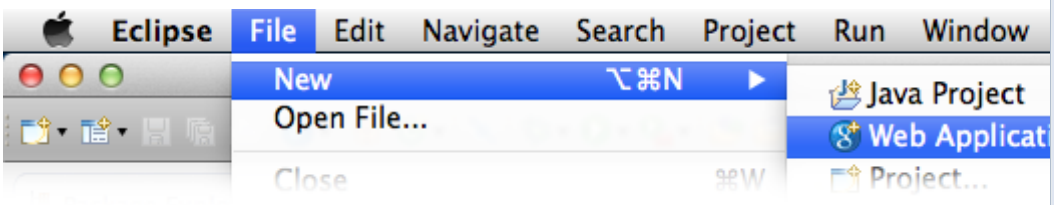
Pre-Configuration Steps

1. Download and install the Kepler release of the Eclipse SDK.

2. Follow the instructions on the "Downloading and Installing the Plugin".
3. Download the Oracle JavaServer Faces 2.2.4 (implementation [javax.faces-2.2.4.jar] and source [javax.faces-2.2.4-sources.jar]) jar files.
4. Download and uncompress version 2.3.1 (jboss-seam-2.3.1.Final.zip) of the Seam 2 framework.

Creating a New Project

1. From the 'New' sub-menu on the 'File' menu, select the "Web Application Project" menu option.



2. Give your project a name in the 'Project Name' field. For the purposes of this tutorial, we will use "Google App Engine JSF 2.2 Template".
3. Enter your project's default package name in the 'Package' field. For the purposes of this tutorial, we will use "`com.wildstartech.gae.jsf22template`".
4. Remove the check mark from the 'Use Google Web Toolkit' box in the "Google SDKs" portion of the dialog.
5. Ensure the "Use Google App Engine" option has a check mark beside it.
6. Remove the check mark beside the "Generate project sample code" option.
7. Left-click on the 'Finish' button.

Importing the Unified Expression Language Files

The following steps will guide you through the process of importing the files which provide support for the new Unified Expression Language. Prior to completing this step, you should have downloaded version 2.3.1 of the Seam 2 framework. Once you uncompress the `jboss-`

seam-2.3.1.Final.zip archive, the el-api.jar , jboss-el-api_2.2_spec.jar, and jboss-el.jar files are located in the lib folder.

1. In the "Project Explorer" tab, located on the left side of your IDE, expand your project and navigate to and left-click on the WEB-INF/lib directory.

   n

2. Left-click on the 'File' menu and select the 'Import...' menu item.
3. When presented with the "Import" dialog, left-click on the 'File System' item under the "General" option.
4. Left-click on the 'Next' button.
5. Left-click on the 'Browse...' button and select the lib folder on your local hard disk drive contained in the directory where you uncompressed the jboss-seam-2.3.1.Final.zip archive. Place check marks in the boxes to the left of the el-api.jar, jboss-el-api_2.2_spec.jar, and jboss-el.jar files.

   NOTE: Passing arguments to expression language methods works regardless of whether the jboss-el-api_2.2_spec.jar file is present. For the time being, I'm going err on the side of caution and include it in this tutorial.

6. Left-click on the 'Finish' button.

Importing the JavaServer Faces Libraries

The following steps will guide you through the process of importing the files that provide support for the reference implementation of the JavaServer Faces 2.2 specification.

1. In the "Project Explorer" tab, located on the left side of your IDE, expand your project and navigate to and left-click on the WEB-INF/lib directory.
2. Left-click on the 'File' menu and select the 'Import...' menu item.
3. When presented with the "Import" dialog, left-click on the 'File System' item under the "General" option.
4. Left-click on the 'Next' button.
5. Left-click on the 'Browse...' button and select the folder on your local hard disk drive into which you saved the javax.faces-2.2.4.jar archive.

6. Place a check mark beside the javax.faces-2.2.4.jar file and left-click not he 'Finish' button.

Configuration File Changes

appengine-web.xml

Edit the appengine-web.xml file found in the WEB-INF directory of the project to allow the web application to store data in the session created for clients visiting our site. Add the <sessions-enabled>true</sessions-enabled> line as shown below. Add the <threadsafe>true</threadsafe> setting to instruct the App Engine runtime environment to allow a single Instance to service multiple requests concurrently.

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>jsf22template</application>
  <version>1</version>
  <!--
    Allows App Engine to send multiple requests to one instance in parallel:
  -->
  <threadsafe>true</threadsafe>
  <!-- Configure java.util.logging -->
  <system-properties>
    <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
  </system-properties>
  <sessions-enabled>true</sessions-enabled>
  <async-session-persistence enabled="false" />
</appengine-web-app>
```

NOTE: We have set the enabled HTTP sessions in this web application by setting the <sessions-enabled> property to true. Experiments configuring App Engine to transfer session data from memcache to the datastore asynchronously to reduce request latency using the <async-session-persistence> parameter to true cause Task queue quotas to become exhausted. For the time being it is recommended the <async-session-persistence> property be set to false. For more information, please refer to the Enabling Sessions section of the Java Application Configuration document.

Do not forget to modify the contents of the <application></application> tag to reflect the name of your App Engine project. Save the changes to the appengine-web.xml file and close it.

web.xml

Left double-click on the contents of the `web.xml` file found in the `WEB-INF` directory and replace the contents with what is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
 <display-name>
   Wildstar Technologies, LLC. Google App Engine JSF 2.2 Template
  </display-name>
  <description>
   Template JSF 2.2 application configured to run on the Google
   AppEngine for Java.
  </description>
  <!-- ***** Designate client-side state saving. *****  -->
  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>
  <!-- Set the default suffix for JSF pages to .xhtml -->
  <context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.xhtml</param-value>
  </context-param>
  <!-- Disable use of threading for single-threaded environments such as
       the Google AppEngine. -->
  <context-param>
    <param-name>com.sun.faces.enableThreading</param-name>
    <param-value>false</param-value>
    <description>
      When enabled, the runtime initialization and default ResourceHandler
      implementation will use threads to perform their functions. Set this
      value to false if threads aren't desired (as in the case of running
      within the Google Application Engine).

      Note that when this option is disabled, the ResourceHandler will not
      pick up new versions of resources when ProjectStage is development.
    </description>
  </context-param>
  <!-- ***** Specify JBoss Expression Language Over Default -->
  <context-param>
    <param-name>com.sun.faces.expressionFactory</param-name>
    <param-value>org.jboss.el.ExpressionFactoryImpl</param-value>
  </context-param>
  <!-- ***** Load the JavaServer Faces Servlet ***** -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <!-- ***** Specify session timeout of thirty (30) minutes. ***** -->
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.xhtml</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

The `com.sun.faces.enableThreading` context parameter is set in the event you use the "Development" value for the `javax.faces.PROJECT_STAGE` context parameter. As indicated in the comments, setting `com.sun.faces.enableThreading` equal to false instructs the Mojarra implementation of the JSF 2.2 API to NOT attempt to use threads as this is not allowed by the Google App Engine platform (as described in the "Sandbox" section of the "Java Servlet Environment" article on the Google App Engine documentation site).

Modifying the Web Application's Default Page

If your web project does not already contain an `index.html` file in the war directory, then skip to the "Creating `index.jsp`" sub-section. If there is already an index.html, continue with the "Renaming `index.html` to `index.jsp` and Modifying Its Contents" section which follows.

Renaming `index.html` to `index.jsp` and Modifying Its Contents

A Google App Engine web project will create an `index.html` file as the default for your web application. In the section above, we specified a change to the `<welcome-file-list>` configuration parameter which instructs the App Engine's runtime environment to look for the following *default* pages in the specified order.

- `index.jsp`
- `index.xhtml`
- `index.html`

As such, we will rename the App Engine's default `index.html` to `index.jsp`. We will ensure that the browser's "back" button will continue to function properly by following the guidelines prescribed in the "Avoid Redirects" section of Yahoo!'s "Best Practices for Speeding Up Your Web Site" document. The `index.jsp` file will be configured to send a redirect to the browser rather than leverage an `<meta http-equiv="Refresh" content="0,welcome.jsf"/>` tag in the `<head>` of an HTML document.

1. Locate and right-click on the index.html file in the war directory shown in the "Project Explorer" window.
2. When the context-sensitive menu is displayed, left-click on the 'Rename...' menu item found on the 'Refactor' sub-menu.

3. When presented with the "Rename Resource" dialog, ensure the 'New name:' field indicates the `index.html` should be renamed to `index.jsp`.

4. Left-click on the 'OK' button to complete the file rename operation.
5. Left double-click on the newly renamed `index.jsp` file to open the file in the editor. Replace the file contents with the following:

```
<html>
  <head>
    <title>Initial Redirect Page</title>
  </head>
  <body>
    <% response.sendRedirect("welcome.jsf"); %>
  <body>
</html>
```

   NOTE: If you decide to use a page *other* than `welcome.xhtml` as your initial landing page, please make the appropriate change to the `index.jsp` shown above.

6. Save your changes to the `index.jsp` and close the file.
7. In the "Package Explorer" window, right-click on the `war` directory. Left-click on the "New" menu and then left-click on the "File" menu item.
8. When presented with the "New File" dialog, type in welcome.xhtml in the 'File name:' field and then left-click on the 'Finish' button.

9. The newly created `welcome.xhtml` file will be presented in the editor. Use the following as the contents of the file.

   NOTE: The first three lines of the file above instruct web browsers to interpret this document as a well-formed XHTML document. As described in the "The !DOCTYPE 'Switch'" section of the MSDN article entitled "CSS Enhancements in Internet Explorer 6", this text instructs versions of Internet Explorer 6 and greater to switch on standards compliance mode.

Please skip to the "Add JavaServer Faces Library to Java Build Path" section.

Creating `index.jsp`

The default welcome page for our web application will be a JSP page entitled index.jsp and we will ensure that the browser's "back" button will continue to function properly by following the guidelines prescribed in the "Avoid Redirects" section of Yahoo!'s "Best Practices for Speeding Up Your Web Site" document. The `index.jsp` file will be configured to send a redirect to the browser rather than leverage an `<meta http-equiv="Refresh" content="0,welcome.jsf"/>` tag in the `<head>` of an HTML document.

1. Locate and right-click on the war directory shown in the "Project Explorer" window.
2. Left-click on the "New" menu and then left-click on the "File" menu item.

3. When presented with the "New File" dialog, enter "`index.jsp`" as the value for the 'File name:" field and left-click on the "Finish" button.

4. The newly created `index.jsp` file will be presented in the editor. Use the following as the contents of the file.

```
<html>
   <head>
```

```
        <title>Initial Redirect Page</title>
    </head>
    <body>
        <% response.sendRedirect("welcome.jsf"); %>
    <body>
</html>
```

5. Return to the "Project Explorer" window and right-click on the war directory.
6. Left-click on the "New" menu and then left-click on the "File" menu item.
7. When presented with the "New File" dialog, enter "`welcome.xhtml`" as the value for the "File name:" field and left-click on the "Finish" button.

8. The newly created `welcome.xhtml` file will be presented in the editor. Use the following as the contents of the file.

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head id="head">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Welcome to JSF 2.2 on the Google AppEngine!</title>
  </h:head>
  <h:body id="body">
    <f:view contentType="text/html">
      <p><h:outputText value="You are now up and running with JavaServer Faces 2.2 on the Google App
Engine."/></p>
    </f:view>
  </h:body>
</html>
```

NOTE: The first three lines of the file above instruct web browsers to interpret this document as a well-formed XHTML document. As described in the "The !DOCTYPE 'Switch'" section of the MSDN article entitled "CSS Enhancements in Internet Explorer 6", this text instructs versions of Internet Explorer 6 and greater to switch on standards compliance mode.

Add JavaServer Faces Library to the Java Build Path

We are going to need to add the JavaServer Faces API to the Java Build Path of our project so we do not encounter any compilation errors.

1. Right-click on the name of the project, "Google App Engine JSF 2.2 Template", in the "Package Explorer" window.
2. When the context-sensitive menu is displayed, left-click on the 'Properties' menu item.

3. When you are presented with the "Properties for Google App Engine JSF 2.2 Template" dialog, left-click on the "Java Build Path" settings and then left-click on the 'Add JARs...' button.

4. Navigate to the `war/WEB-INF/lib` folder of the project, left-click on the `javax-faces-2.2.4.jar` file, then left-click on the 'OK' button.

5. When you are returned to the "Properties for Google App Engine JSF 2.2 Template" dialog, left-click on the 'OK' button.

Ensuring Contents of HttpSession Are Saved

Any time the Google App Engine framework detects a change in the HTTP session, that information is written out to the datastore. This works very well for immutable objects; however, it becomes more challenging when dealing with mutable objects as the GAE framework has no way of knowing when a field inside an object stored in the HTTP session is changed. A Stack Overflow question entitled "Session lost in Google App Engine using JSF" outlines a great strategy using a JSF `PhaseListener` to store the current date/time in the HTTP session at the end of each Phase.

The following process will guide you through the steps necessary to create this PhaseListener.

1. In the "Project Explorer" tab, located on the left side of your IDE, expand your project. Expand the "src" branch and left-click on the package to which you would like to add this class. For the purpose of this tutorial, we are going to select `com.wildstartech.gae.jsf22template`.
2. Right-click on the package, display the "New" menu and then left-click on the "Class" menu item.

3. Enter "SessionPhaseListener" in the "Name" field.
4. Left-click on the "Add..." button that appears to the right of the "Interfaces" section.
5. When the "Implemented Interfaces Selection" dialog appears, enter "PhaseListener" in the "Choose Interfaces" field. The "Matching items:" field should contain "**PhaseListener** - javax.faces.event" as depicted in the screen shot below. Left-click on the "OK" button.

6. When you are returned to the "New Java Class" dialog, shown in the figure below, left-click on the "Finish" button to create the class.

7. The source code for the `SessionPhaseListener.java` file will be presented and you can replace it with the code shown below.

```
package com.wildstartech.gae.jsf22template;
import java.util.Map;
import java.util.logging.Logger;

import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;
/**
 * PhaseListener to ensure the HttpSession data is written to the datastore.
 *
 * <p>If properly configured, an application deployed on the
 * <a href="http://developers.google.com/appengine">Google App Engine<a>
 * platform can leverage HttpSessions to store data between requests.  Session
 * data is stored in the datastore and memcache is also used for speed.  Session
 * data is persisted at the <strong>end</strong> of the request.</p>
 * <p>The App Engine session implementation will not recognize if properties
 * of objects stored in the session are changed which is why we have this
 * <code>PhaseListener</code> which will modify a session attribute with the
 * current date and time (in milliseconds) at the end of every phase.</p>
 *
 * @author Derek Berube, Wildstar Technologies, LLC.
 * @version 2014-01-07, 1.0
 *
 * @see
https://developers.google.com/appengine/docs/java/config/appconfig#Java_appengine_web_xml_Enabling_sessi
 * @see http://stackoverflow.com/questions/19259457/session-lost-in-google-app-engine-using-jsf
 */
public class SessionPhaseListener implements PhaseListener {
 /** Used in object serialization */
 private static final long serialVersionUID = -8246272798261076270L;
 private static final String _CLASS = SessionPhaseListener.class.getName();
 private static final Logger logger = Logger.getLogger(_CLASS);
 private static final String TIME_KEY="NOW";

 @Override
```

```
public void afterPhase(PhaseEvent event) {
logger.entering(_CLASS,"afterPhase(PhasseEvent)",event);
FacesContext ctx=null;
ExternalContext eCtx=null;
Map<String,Object> sessionMap=null;

ctx=event.getFacesContext();
eCtx=ctx.getExternalContext();
sessionMap=eCtx.getSessionMap();
sessionMap.put(TIME_KEY, System.currentTimeMillis());
logger.exiting(_CLASS,"afterPhase(PhaseEvent)");
}
@Override
public void beforePhase(PhaseEvent event) {
logger.entering(_CLASS,"beforePhase(PhaseEvent)",event);
logger.exiting(_CLASS,"beforePhase(PhaseEvent)");

}
@Override
public PhaseId getPhaseId() {
logger.entering(_CLASS,"getPhaseId(PhasseEvent)");
PhaseId phaseId=PhaseId.ANY_PHASE;
logger.exiting(_CLASS,"getPhaseId(PhaseEvent)",phaseId);
return phaseId;
}
}
```

The `getPhaseId()` method returns `PhaseId.ANY_PHASE` which tells the JSF framework the `beforePhase` and `afterPhase` methods of the listener should be called for each phase of the JavaServer Faces lifecycle. The `beforePhase` method of this class does nothing; however, the `afterPhase` method will:

1. Obtain a reference to the `FacesContext`.
2. Using the `FacesContext`, obtain a reference to the `ExternalContext`.
3. The `ExternalContext` will be used to get access to the `HttpSession` in the form as a `Map`.
4. The current date/time (expressed as milliseconds) is stored in the `HttpSession`.

Unfortunately, the JSF 2.2 API does not provide an annotation to declare a PhaseListener programmatically, so we are going to have to create a `faces-config.xml` configuration file.

1. In the "Project Explorer" tab, located on the left side of your IDE, expand your project.
2. Expand the "war" branch of your project.
3. Right-click on the "WEB-INF" folder, left-click on the "New" menu and then left-click on the "File" menu item.
4. Enter "faces-config.xml" as the value for the "File name:" field as depicted in the screen shot below.

5. Left-click on the "Finish" button.
6. Use the following as the contents for the "`faces-config.xml`" file and then close the editor view.

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    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-
facesconfig_2_2.xsd">
    <application>
    <lifecycle>
 <phase-listener>com.wildstartech.gae.jsf22template.SessionPhaseListener</phase-listener>
    </lifecycle>
    </application>
</faces-config>
```

Starting Your Google App Engine JSF Application

Now that you have completed the requisite configuration steps, you are ready to launch your first JSF application running on the App Engine platform.

1. Right-click on the "Google App Engine JSF 2.2 Template" item in the "Project Explorer" window.
2. Left-click on the "Run As" menu and then left-click on the "Web Application" menu item.

3. Open a web browser and enter "http://localhost:8888/" in the browser's address field. Your browser will be re-directed to the "http://localhost:8888/welcome.jsf" and you should see something similar to the browser window depicted below.

Additional Reading

The following articles provide additional information on JavaServer Faces technology.

- Compatibility Issues
- Configuring JBoss Weld to Run with JavaServer Faces on Google App Engine
- Using the Resource Framework with CSS Style Sheets

References

- "Best Practices for Speeding Up Your Web Site" http://developer.yahoo.com/performance/rules.html
- "CSS Enhancements in Internet Explorer 6" http://msdn.microsoft.com/en-us/library/bb250395(VS.85).aspx
- "Parsing date header as int caused NumberFormatException" https://code.google.com/p/googleappengine/issues/detail?id=8415
- "Quick Start" http://code.google.com/eclipse/docs/getting_started.html
- "Session lost in Google App Engine using JSF" http://stackoverflow.com/questions/19259457/session-lost-in-google-app-engine-using-jsf
- "Using Concurrent Requests" http://code.google.com/appengine/docs/java/config/appconfig.html#Using_Con

Copyright © 2014, Wildstar Technologies, LLC.

Comentarios