Jakarta RESTful Web Services 3.1 Workshop Participant

Module 6: Servlet and Jakarta Faces RESTful Clients

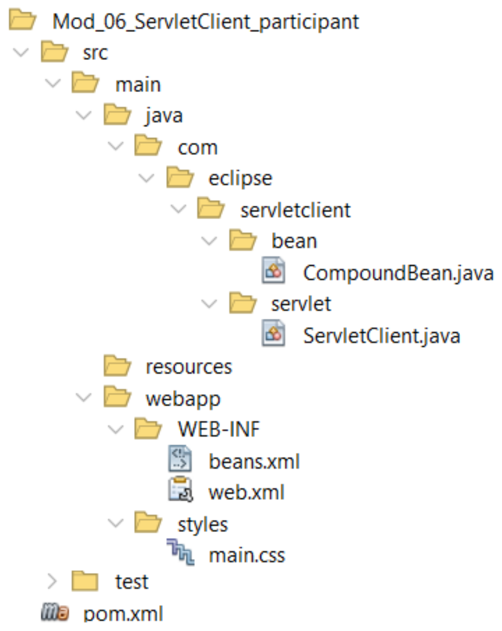
In this module we will explore RESTful clients that are server based. This means that the client can be part of a Servlet or in a POJO called upon from a Jakarta Faces page. You will be working with the following projects:

- mod_06_servletclient_participant
- mod_06_jsfclient_participant
- mod_06_restserver

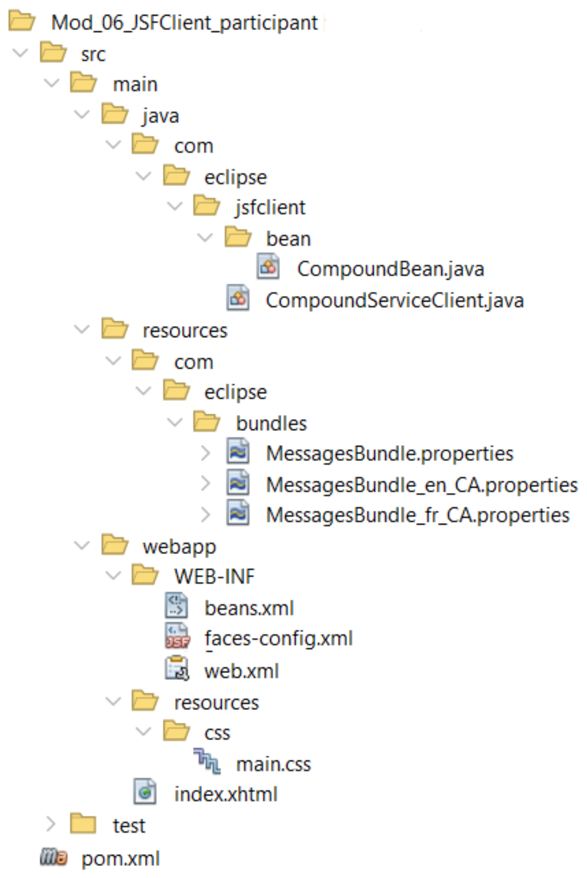
The Mod_06_RestServer is complete and is the host of the service. Have this running on the server before you work on either of the two clients.

The Projects

There are two different clients in this module. The first is the Servlet based client using an HTML page for input. The second is the Jakarta Faces based client. Here are the Maven layouts.



```
Mod_06_ServletClient_participant
  src
    main
      java
        com
          eclipse
            servletclient
              bean
                CompoundBean.java
              servlet
                ServletClient.java
      resources
      webapp
        WEB-INF
          beans.xml
          web.xml
        styles
          main.css
    test
  pom.xml
```



```
Mod_06_JSFClient_participant
  src
    main
      java
        com
          eclipse
            jsfclient
              bean
                CompoundBean.java
              CompoundServiceClient.java
      resources
        com
          eclipse
            bundles
              MessagesBundle.properties
              MessagesBundle_en_CA.properties
              MessagesBundle_fr_CA.properties
      webapp
        WEB-INF
          beans.xml
          faces-config.xml
          web.xml
        resources
          css
            main.css
        index.xhtml
    test
  pom.xml
```

The pom file

The pom file dependencies are the same for both versions, servlet and faces. As the server already has all the libraries you need the dependencies are quite simple.

```
<dependencies>
    <dependency>
        <groupId>jakarta.platform</groupId>
        <artifactId>jakarta.jakartaee-api</artifactId>
        <version>${jakartaee-api.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

There are no plugins required.

Let's look at the beans.xml

This file is the same for all examples.

Let's look at the web.xml

In the Servlet client all that you need is the name of the welcome file. In this example the welcome file is the `index.html` file.

The Jakarta Faces client's web.xml file also only needs the welcome file name that in this case is an xhtml file, `index.xhtml`. There are two optional settings. The PROJECT_STAGE determines the messages that Faces returns in the event of a problem. The FACELETS_SKIP_COMMENTS determines if comments in the xhtml file will be visible if the user uses View Source Code in their browser.

Now let us look at what you need to do to finish both clients.

Mod_06_ServletClient_participant

In this project a Jakarta Servlet will be the home of the REST services client code. The ServletClient.java is the target for a submit from the index.html page. The HTML form shows that when the Submit button is pressed the ServletClient will be called.

```
<form action="ServletClient" method="get">
```

The class constructs a `CompoundBean` using the values from the request parameters. Next is a `callCompoundService` where you will enter the REST client code.

The task in this Servlet is to display a web page. My example places the page code in the `doGet` method. The purpose of this is to display the input and the result. Normally you would rarely construct HTML in a Servlet. A more common setup is to use Jakarta Server Pages with Jakarta Servlets.

Your task is to implement the RESTful client code in the `callCompoundService` method.

Now let us look at the Faces version.

Mod_06_JSFClient_participant

In this project we are dispensing with Servlets. Instead, the file `index.xhtml` is a Jakarta Faces file. If you are not familiar with Faces, then this is a simple introduction. The Faces page looks like an HTML page. Unlike an HTML file, this page does not load into the browser. Instead, the page loads into the Faces framework servlet where it is translated into HTML that is sent to the browser.

This project also employs internationalization. This the purpose of the `faces-config.xml` file. It has the name of the package containing the bundles for each language along with the name of the identifier you can use in your Faces page. In the index.xhtml you will see references such as `#{msgs.principal}`. This means that you want the text with the key `principal` from the appropriate language bundle. You can find the bundles in the folder `src/main/resources/com/eclipse/bundles`. There is a default, English, and French bundle. The visitor's browser informs the server of your language.

Your task is to implement the RESTful client code in the `callCompoundService` method very similar to the `ServletClient`. The only difference is that JSF can access beans and call action methods directly from the page. Action methods must return a string or null. The string could be the next page to navigate to but as this is a single page app, we return null.

The code to interact with the service will be the same code as the servlet but with the result stored in the CDI bean and the return value set to null.

Where are we now?

At this point in the workshop, we have seen how we can construct REST services that can run on a server or in a desktop app. We have seen desktop, Servlet, and JSF clients. Coming up we will learn how to construct a service that accepts a binary file such as an image.