JAX-RS Redirection

JAX-RS : Redirection

With RequestDispatcherProvider

RequestDispatcherProvider is a JAXRS MessageBodyWriter which can redirect to JSP pages, named or default servlets. It can be used to serve all the responses from a given resource class or restricted to serving a limited set of classes only using a classResources map property.

Starting from CXF 2.5.0 and 2.4.4 it is also possible to specify that only responses to requests with matching URIs that will be processed.

At the moment, this provider is statically configured to support text/html content types, but it can be easily configured to support other content types if needed.

In addition to 'resourcePath' and 'dispatcherName' properties, one can set a 'scope' property which has two possible values, 'request' and 'session' with 'request' being the default value. It affects the way the JSP code can retrieve parameters passed to it by the RequestDispatcherProvider. If it is a 'request' scope then all the parameters are set as the attributes on the current HTTP request. If session scope then they're set as the attributes on the current HTTP session.

`RequestDispatcherProvider` sets the following parameters :

- JAXRS method response object. The name of this parameter is either a simple class name of this object (lower case) or a value retrieved from a beanNames map property using the fully qualified class name of this object.
- All the path, query and matrix parameters which have been initialized during the method execution
- "absolute.path", "base.path" and "relative.path" obtained from the current UriInfo

Here are some examples. Lets assume we have a book.war web application deployed.

```
<jaxrs:server id="bookservice1" address="/bookstore1">
    <jaxrs:serviceBeans>
      <bean class="org.apache.cxf.systest.jaxrs.BookStoreDispatch"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
       <ref bean="dispatchProvider"/>
    </jaxrs:providers>
</jaxrs:server>

<bean id="dispatchProvider" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
      <property name="resourcePath" value="/book.html"/>
</bean>
```

The above redirects the response to a default book.html page which is available directly in the /webapps/book folder. Typically one would do it to return some static confirmation to the client. For example, consider a POST form request that has been processed by a given JAX-RS method and the only thing that needs to be done now is to return the HTML confirmation view. Note that JAX-RS MessageBodyWriters are not invoked if the resource method returns no custom object - which is not needed in the case of the static confirmation, so for RequestDispatcherProvider be able to redirect to book.html one should simply introduce say an EmptyConfirmation bean with no properties and return it from the resource method.

Here is another example (omitting jaxrs:server declaration for brewity):

```
<bean id="dispatchProvider" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
      <property name="resourcePath" value="/book.jsp"/>
</bean>
```

The only difference from the previous example is that "/book.jsp" will be delegated to with the task of creating a view. This is a more interesting example and we presume that the resource method returns say an instance of the "org.bar.Book" bean:

```
@Path("/books")
public Resource {
    @GET
    @Produces({"text/html", "application/xml"})
    @Path("{id}")
    public Book getBook(@PathParam("id") String id) {
        // return the book
    }
}
```

Note how non-intrusive RequestDispatcherProvider is as far as writing the JAX-RS resource code is concerned, you simply list supported media types in @Produces as usual.

RequestDispatcherProvider will make an instance of Book available as an HttpServletRequest attribute named "org.bar.Book" by default. this can be customized. If a "beanName" property is set, for example to "book", then book.jsp will access a Book instance as a "book" attribute. If you have say 2 resource methods returning instances of different bean classes, possibly for different view handlers then a beanNames map property can be used, for example:

```
<bean id="dispatchProvider" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
      <property name="classResources">
          <map>
             <entry key="org.bar.Book"  value="/book.jsp"/>
             <entry key="org.bar.Customer"  value="/customer.jsp"/>
          </map>
      </property>
      <property name="beanNames">
          <map>
             <entry key="org.bar.Book"  value="book"/>
             <entry key="org.bar.Customer"  value="customer"/>
          </map>
      </property>
</bean>
```

The above configuration says that a "book.jsp" resource will handle an instance of Book by accessing it as a "book" attribute and a "customer.jsp" - an instance of Customer by retrieving it as a "customer" attribute. Note you don't need to use the "beanNames" property in such cases, a simpler "beanName" property can do unless you have a single (jsp) resource dealing with both Book and Customer.

Apart from making an instance of response class available as HttpServletRequest attribute, RequestDispatcherProvider will also make all the Path, Query and Matrix parameters available as HttpServletRequest parameters (as opposed to attributes) by default. For example, given the above code fragment, an HttpServletRequest parameter named "id" representing a @PathParam("id") available to the view handler, as well as all other query and matrix parameters. Note that RequestDispatcherProvider can be configured to save all these request parameters as HttpServletRequest attributes by setting a boolean saveParametersAsAttributes property to true.

Now, imagine a scenario like this: we have two resource methods returning a ReservationStatus bean. The first method returns a successful confirmation or uses Response.seeOther(...) to redirect to a method handling the failed reservation. So both methods return the same ReservationStatus bean but we will have two different views handling successful and failed reservations respectively. Here is one way to manage it:

```
<bean id="reserveRegistrationViews" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
         <property name="resourcePaths">
            <map>
              <entry key="/reservations/reserve/complete" value="/forms/reservationConfirm.jsp"/>
              <entry key="/reservations/reserve/failure" value="/forms/reservationFailure.jsp"/>
            </map>
         </property>
         <property name="beanName" value="data"/>
</bean>
```

Given that the same ReservationStatus bean is returned in both cases, it is actually the original request URI fragments which are used to match which view handler will deal with a given ReservationStatus, example, a response to request URI "http://localhost:8080/reservations/reserve/complete" will be handled by "/forms/reservationConfirm.jsp".

Note that RequestDispatcherProvider has a 'dispatcherName' property - that can be handy when redirecting to named servlets (example, MyServlet) including
such ones as "jsp" or "default", especially when CXFServlet handling a given invocation has a uri pattern (typically, wildcard) that may also capture the redirection request, see the next section for more information.

Next, imagine a scenario like this: we have a single resource method accepting some data and the response view will need to be different depending on the status of the request processing. Using enumerations is the most effective option in this case:

```
package resource;

public class Status {
    UPDATE_SUCCESS,
    UPDATE_FAILURE
}

@Path("/")
public class Resource {

  @PUT
  @Produces("text/html")
  public Response update(SomeData data) {
     if (update(data)) {
         return Response.ok(Status.UPDATE_SUCCESS).build();
     } else {
         return Response.ok(Status.UPDATE_FAILURE).build();
     }
  }
}
```

Next, you may have a single JSP handler which will check whether it is Status.UPDATE_SUCCESS or Status.UPDATE_FAILURE and format the response accordingly. In this case a basic RequestDispatcherProvider configuration will do:

```
<bean id="dispatchProvider" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
      <property name="resourcePath" value="/updateStatus.jsp"/>
</bean>
```

Alternatively you may have a dedicated view handler dealing with the specific status, in this case either:

```
<bean id="reserveRegistrationViews" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
         <property name="classResources">
            <map>
              <entry key="resource.Status.UPDATE_SUCCESS" value="/forms/updateSuccess.jsp"/>
              <entry key="resource.Status.UPDATE_FAILURE" value="/forms/updateFailure.jsp"/>
            </map>
         </property>
</bean>
```

or, starting from CXF 2.7.1,

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

<bean id="reserveRegistrationViews" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
         <property name="enumResources">
            <map>
              <entry
                 <key>
                    <util:constant static-field="resource.Status.UPDATE_SUCCESS"/>
                 </key>
                 <value>/forms/updateSuccess.jsp</value>
              </entry>
              <entry
                 <key>
                    <util:constant static-field="resource.Status.UPDATE_FAILURE"/>
                 </key>
                 <value>/forms/updateFailure.jsp</value>
              </entry>
            <map>
         </property>
</bean>
</beans>
```

will help.

Starting from CXF 2.6.1 it is possible to configure the provider to check if the current class has an associated view handler or not, for example:

```
<bean id="viewHandler" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
         <property name="dispatcherName" value=jsp""/>
         <property name="useClassNames" value="true"/>
</bean>
```

For example, given a simple class name such as "BookInfo", RequestDispatcherProvider will check if a "/WEB-INF/bookInfo.jsp" handler is available or not. The provider will likely be extended to check few more locations as needed.

RequestDispatcherProvider also checks a "redirect.resource.path" property on the outbound message. If this property is set then it will try to find a RequestDispatcher available on a given path.

A new property, "includeResource" is available starting from CXF 3.0.4: RequestDispatcher.include() instead of RequestDispatcher.forward() will be used if this property is set to true.

Finally, a 'servletContextPath' property can be used to have some other ServletContext (as opposed to the current one) be used for RequestDispatcher look-ups. If set then the current ServletContext.getContext(servletContextPath) will be used to get the needed ServletContext.

Logging redirects

To get RequestDispatcherProvider log the information about redirects, please set a 'logRedirects' property:

```
<bean id="reserveRegistrationViews" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
         <property name="logRedirects" value="true"/>
         <!-- other properties as needed -->
</bean>
```

You will see the logging entry like this one:

```
23-Jul-2012 11:26:13 org.apache.cxf.jaxrs.provider.RequestDispatcherProvider logRedirection

INFO: Setting an instance of "oauth2.common.ConsumerRegistration" as HttpServletRequest attribute "newClient"
and redirecting the response to "/forms/registerAppConfirm.jsp"
```

With CXFServlet

Please see the "Redirection" section on the Servlet Transport page.

Note that both CXFServlet and JAXRS RequestDispatcherProvider can work together effectively on executing the redirection requests as described at that page.

If CXFServlet URI pattern does not match the resource URIs RequestDispatcherProvider is redirecting to then there's nothing to worry about.

If you have CXFServlet listening on "/" (thus effectively catching all the requests) and also would like to use RequestDispatcherProvider, then make sure that a 'dispatcherName' property is also set, for example:

```
<bean id="dispatchProvider" class="org.apache.cxf.jaxrs.provider.RequestDispatcherProvider">
    <property name="dispatcherName" value="jsp"/>
    <property name="resourcePath" value="/WEB-INF/jsp/test.jsp"/>
</bean>
```

If resources which are redirected to can be made public (i.e, moved out of /WEB-INF) then alternative option (instead of adding a 'dispatcherName' property to RequestDispatcherProvider and still have CXFServlet listening on '/') is to configure both RequestDispatcherProvider and CXFServlet to redirect to resources such as "/jsp/test.jsp".

Also if you have many public view handlers then rather than having a "dispatcherName" property set on every dispatcher bean, you can get it set only once on CXFServlet:

```
<servlet>
        <servlet-name>RESTServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <init-param>
               <param-name>redirects-list</param-name>
            <param-value>/forms/.*</param-value>
        </init-param>
        <init-param>
               <param-name>redirect-servlet-name</param-name>
            <param-value>jsp</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>RESTServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
```

Here we have all the requests targeted at /form/* (example, /form/book.jsp, etc) handled by a jsp handler.

Custom Redirection

One can borrow some of the code from RequestDispatcherProvider and do the custom redirection from CXF in interceptors or custom invokers, if you will try to do it then you will also need to set an AbstractHTTPDestination.REQUEST_REDIRECTED property with a 'true' value on a current input message.