CO3409 Distributed Enterprise Systems

Lab : RESTful Webservices

Summary

Create a Servlet that provides a REST-based service to provide an exchange rate service. The first version will deliver HTML. The second version will return XML.

Purpose

On completion of this you should be able to

- Create a Servlet responding to a range of URLs defined by a pattern
- Modify the configuration information (in web.xml)
- Handle parameters from a POST HTTP method
- Use Java collection classes Hashmap and Set
- Use Java for each
- Appreciate the role of XSL in transforming XML from one format to another

Things to Remember

Some things to remember...

- Run Netbeans as "Administrator".
- When running your application, if you are using a database please ensure your database is running first (see previous lab worksheets).
- To deploy an Enterprise Application, I suggest using Clean and Build and then Deploy.

Activities

1. Background

Our service will provide the conversion rate between different currencies. It will also allow users to update the conversion rate (using simple password-based security). It will be provided as REST-based service. This means that the conversion rate is provided as a resource, identified by a URI (uniform resource identifier). To find the conversion rate from dollars to pounds, the user would retrieve information from `http://<service host address>/restservice/conversion/dollar/pound`

In HTML this would be:

```
<html>
  <body>
    <p> dollar to pound conversion: 0.65</p>
  </body>
</html>
```

In XML it would be

```
<exchange>
  <message>dollar to pound conversion: 0.65</message>
</exchange>
```

In line with the REST philosophy, the service will provide useful information in response to requests that don't give a full URL. For example: http://localhost:8080/restservice/conversion/restservice/conversion/dollar/ returns (in HTML):

```
<html>
  <body>
    <p> dollar conversions </p>
      <a href=/restservice/conversion/dollar/euro>euro</a>
      <a href=/restservice/conversion/dollar/pound>pound</a>
  </body>
</html>
```

or in XML

```
<exchange>
  <message>dollar conversions</message>
  <links>
    <link><text>euro</text><address>
http://localhost:8080/restservice/conversion/restservice/conversion/dollar/euro</ad
dress></link>
    <link><text>pound</text><address>
http://localhost:8080/restservice/conversion/restservice/conversion/dollar/pound</a
ddress></link>
  </links>
</exchange>
```

and http://localhost:8080/restservice/conversion/restservice/conversion/ returns (in XML – you should be able to work out the HTML for yourself).

```
<exchange>
  <message>conversions</message>
  <links>
    <link><text>dollar</text><address>
http://localhost:8080/restservice/conversion/serviceURL/dollar</address></link>
    <link><text>euro</text><address>
http://localhost:8080/restservice/conversion/serviceURL/euro</address></link>
    <link><text>pound</text><address>
http://localhost:8080/restservice/conversion/serviceURL/pound</address></link>
  </links>
</exchange>
```

2. Creating a RESTful Web service using the NetBeans IDE

Creating a Servlet to return a representation of a resource

1. Create a Web application called `restservice`

   > **Note:** if you use a different name, you will have to alter the links in the code below, or change the context path in the Run tab of the project properties to "restservice".

   Use Glassfish 3 with Java EE 6 Web if available, alternatively you can use TomCat. Create a servlet called rest in a package called restpackage. In the Configure Servlet Deployment Step of the New File wizard, modify the URL Pattern(s): from /rest to /conversion. This will be the URL that the client will send to the Servlet.

2. Open the Project properties. On the Run branch, enter "conversion” (No quotes) into the Relative URL. This will make the server's initial page the Servlet rather than the index.jsp page.
3. Remove the comments from the body of processRequest in the Servlet, and add the following to the body of the Servlet, to replace the statement outputting the header 1 line.

```
String pathInfo = request.getPathInfo();
  if (pathInfo == null || pathInfo.equals("/")) {
    out.println("<p>Conversions</p>");
    out.println("<a href=\"/restservice/conversion/dollar\">dollar</a>");
    out.println("<a href=\"/restservice/conversion/euro\">euro</a>");
    out.println("<a href=\"/restservice/conversion/pound\">pound</a>");
  }
  else {
    String[] stringList = pathInfo.split("/");
    for (String s : stringList) {
      out.println("<p>" + s + "</p>");
    }
    // we have one or more currencies
  }
```

4. Run the application. You will get error 404 when you click on the links generated. This is because only `/restservice/conversion` URL is passed to the servlet. Modify the annotation to pass anything starting with `/restservice/conversion` to the servlet by changing the URLPatterns annotation from `/conversion` to `/conversion/*`.

   Clean and build the application. Run it and check that the links work.
5. What do you think is being returned from `request.getPathInfo()`?
6. What is `pathInfo.split("/")` doing? Use the debugger if necessary to work this out.
7. Add the following to the class (not to a method) to create instance variables. The lookup table is initialised in the constructor.

```
String [] validCurrencies = {"dollar","euro","pound"};

HashMap lookup = new HashMap();

public rest() {
lookup.put("dollar/euro", new Float(1.1));
lookup.put("dollar/pound", new Float (0.7));
lookup.put("euro/dollar", new Float (1.4));
lookup.put("euro/pound", new Float (0.85));
lookup.put("pound/dollar", new Float (1.6));
lookup.put("pound/euro", new Float (1.2));
}
```

8. Fix up any missing imports. (Hint: lightbulb)
9. Add the following after the comment “We have one or more currencies”:

```
  if (stringList.length == 2) {
    out.println("<p>" + stringList[1] + " Conversions</p>");
    for (String s : validCurrencies) {
      if (!s.equals(stringList[1])) {
        out.println("<a href=\"/restservice/conversion/" +
             stringList[1] + "/" + s + "\">" + s + "</a>");
      }
    }
  } else if (stringList.length == 3) {
    Float result = (Float)lookup.get(stringList[1]+"/"+
        stringList[2]);
    if (result == null){
      out.println("<p>" + stringList[1] + " to " +
        stringList[2] + " conversion is not supported</p>");
      }
    else
      out.println("<p>" + stringList[1] + " to " +
         stringList[2] + " conversion: "+result+"</p>");
      out.println("<a href=\"/restservice/conversion/\">conversions</a>");
  }
```

10. Run the application and test it using the links and typing the URL directly into the browser. Remove any debugging code – i.e. the code that printed out the path components.

Modifying the state of the resource

1. The service should allow exchange rates to be modified by an authorised user. Add the following code to processRequest, just after it has determined that you have specified 2 currencies (work out what the length of the string must be) and before it calculates their exchange rate:

```
if (request.getMethod().equals("POST")){
   String user = request.getParameter("userid");
   String password = request.getParameter("password");
   String strExchangeRate = request.getParameter("exchange");
   try {
      Float exchangeRate = Float.parseFloat(strExchangeRate);
      if (user != null && password != null & exchangeRate != null) {
         if (user.equals("chris") && password.equals("password")) {
            lookup.put(stringList[1] + "/" + stringList[2], exchangeRate);
         }
      }
   } catch (NumberFormatException numberFormatException) {
      out.println(strExchangeRate + "is not a valid floating point number");
   }
}
```

2. Modify `index.jsp` to contain a form called `exchangeForm`, with an action that is `/restservice/conversion/dollar/euro` and which uses the `POST` method
3. Add three textboxes called `userid`, `password` and `exchange`.
4. Add some text to label the boxes.

5. Add a submit button, labelled `submit`.
6. Run the application. Navigate to `/restservice/conversion/dollar/euro`
7. Now navigate to `/restservice/index.jsp`
8. Set up the correct userid and password and enter a new dollar/euro exchange rate. Click Submit.

> **Note**: For this to be more useful, we would have to make it easier to select the exchange rate to be modified:

9. Add the following into the head element:

```
<SCRIPT type="text/JavaScript">
  function OnSubmitForm()
  {
    var dest = "/restservice/conversion/";
    dest = dest + document.exchangeForm.source.value + "/"
      + document.exchangeForm.destination.value;
    document.exchangeForm.action = dest;
    alert(dest)
    return true;
  }
</SCRIPT>
```

10. Add the following attribute into the Form tag:

```
onSubmit="return OnSubmitForm();"
```

11. Add the following to the form:

```
From: <input type="text" name="source" value="" size="15" /><br>
To: <input type="text" name="destination" value="" size="15" /><br>
```

Understanding the Application

1. What method do you use to compare strings? Why don't you use `==`?
2. What is the benefit of the `for each` version of the for statement?
3. What does a `Hashmap` do?
4. What does the String split method do?
5. Explain how the application works. Hint there are three key URIs: `/conversion`, `/conversion/dollar`, and `/conversion/dollar/euro`.

Further Reading

Read the following.

· Guide: When and how to use REST http://searchsoa.techtarget.com/essentialguide/Guide-When-and-how-to-use-REST

Have you understood?

1. What are the key features of a RESTful application?
2. Does this application satisfy the REST principles?
3. This application stores the information in an in-memory data structure, what would be necessary in a real application (hint: persistence).
4. What is alternative way of saving information such as the URL patterns in java web containers or application servers instead of using annotations?
5. What are the advantages and disadvantages of delivering XML rather than HTML?

Advanced Work

Delivering XML

Modify the servlet so that it generates XML rather than HTML. You will need to delete the lines that output HTML `<body>` tags etc. Replace the lines that currently output the message and the links by code that generate strings in the appropriate format. See the XML samples at the start of this lab for the format. I used two string variables, message and addresses and built up the output in them. If you do this, you may find the following method useful:

```
/* message should be a simple string, addresses should be a sequence of <link>
elements each with a text element and an address element */
String generateXML(String message, String addresses) {
  String res = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
  // "<?xml-stylesheet type=\"text/xsl\" href=\"http:/restservice/tohtml.xsl\"?>" +
     "<exchange><message>";
  res = res + message+ "</message><links>";
  res = res + addresses +"</links></exchange>";
  return res;
}
```

Transforming the XML with an XSLT File

XSLT files allow you to describe how different XML elements should be transformed. Create a new XSL Stylesheet file (in the XML category). Call it `tohtml.xsl` and put it in the Web Pages folder (You can't save it there directly, so save it in the default folder, switch to the Files tab and drag it to the Web Pages folder, alongside `index.jsp`.

Replace the default template with the following:

```
<xsl:template match="/">
  <html>
  <body>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
```

```
<xsl:template match="message">
  <p><b><xsl:value-of select="."/></b></p>
</xsl:template>

<xsl:template match="links">
  <ul><xsl:apply-templates select="link"/></ul>
</xsl:template>

<xsl:template match="link">
  <li> <a href="{address}"><xsl:value-of select="text"/></a></li>
</xsl:template>
```

Uncomment the line that generates a link to a stylesheet in generateXML() and Run the application.

Have you understood?

1. What does XSLT do?
2. What language is XSLT based on?