Web Services

Tutorial

Outline

- Web Services
- Program

Web Services Overview

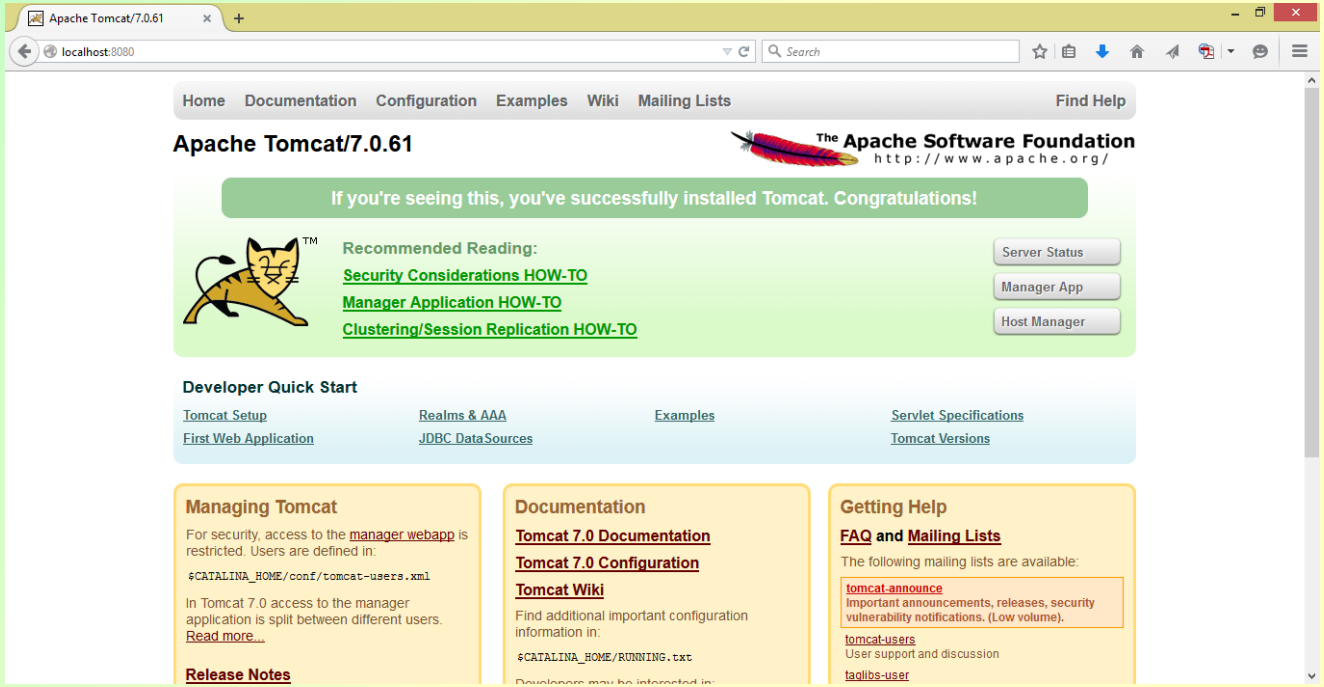
- Web services are client and server applications that communicate over HTTP to implement RPCs
  - They are language- and platform-independent
  - Data is transmitted in a standardized XML format
- Big Web Services
  - Java API for XML Web Services (JAX-WS)
  - Use XML messages following Simple Object Access Protocol (SOAP)
  - Operations offered by the service are include in the Web Services Description Language (WSDL)
- RESTful Web Services
  - Java API for RESTful Web Services (JAX-RS)
  - No requirement of XML messages or WSDL service
- Additional information can be found at
  http://docs.oracle.com/javaee/
  http://docs.oracle.com/javaee/7/tutorial/doc/webservices-intro.htm

Web Services Pre-Configuration

- Install a Java Application Server, such as Tomcat or Glassfish (comes with Java EE SDK)
  http://tomcat.apache.org/download-70.cgi (Download the core)
  http://www.oracle.com/technetwork/java/javaee/downloads/index.html
- Download the following soap-bin-2.2.zip - https://archive.apache.org/dist/ws/soap/version-2.2/
  - Xerces2_Java_2.11.0.zip - http://xerces.apache.org/mirrors.cgi
  - javax.mail.jar - https://java.net/projects/javamail/pages/Home, version 1.5.1
  - activation.jar from JavaBeans Activation Framework - http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-plat-419418.html#jaf-1.1.1-fcs-oth-JPR
- The necessary files are posted on Camino
  - Copy soap.jar, xml-apis.jar, javax.mail.jar, and activation.jar from the above files into the lib directory of your application server
  - Copy soap.war from the soap-bin-2.2.zip file into the webapps directory of your application server
- Start your application server so the web archive (WAR) file will be loaded
  - From the bin directory of Tomcat, run startup.exe or startup.sh

Application Server Installation

- If the application server was installed and started properly, when you go to http://localhost:8080 in a browser, you will see the following



JAX-WS Application Example

- Here are the steps for setting up a web service with JAX-WS

1. Write the web service server code
2. Compile the web service server
3. Create a document descriptor file for the web service
4. Deploy the web service server in an application server
5. Write the web service client code
6. Compile the web service client
7. Run the web service client

JAX-WS Application Example – Step 1

- Step 1 – Write the web service server code
  - The web service server is just a class that has one or more methods in it
  - Standard types must be used that SOAP supports
    - If SOAP doesn't support the type, you can create custom types in SOAP, though you are binding your application to languages that support the custom type then
  - Your code can be in a package

JAX-WS Application Example – Step 2

- Step 2 – Compile the web service server
  - Compile the Java code just like any other class, from the directory that contains the top-most package

```
Windows - javac coen317\SortServer.java
Mac – javac coen317/SortServer.java
```

  - Copy the compiled class into the following directory in your application server webapps\soap\WEB-INF\classes\<package>\
  - Restart the application server

SortServer.java

```
1   package coen317;
2
3   public class SortServer {
4     public int[] sort(int [] numbers) {
5       int temp;
6       for (int i=0; i < numbers.length; i++) {
7         for (int j=0; j < i; j++) {
8           if (numbers[i] > numbers[j]) {
9             temp = numbers[i];
10            numbers[i] = numbers[j];
11            numbers[j] = temp;
12          }
13        }
14      }
15      return numbers;
16    }
17  }
```

JAX-WS Application Example – Step 3

- Step 3 – Create a document descriptor file for the web service
  - The document descriptor file lets the application server know that a class is a web service, as well as what methods in the class are exposed
  - Java annotations can also be used instead of a .dd file
  - You can also deploy through an administrator interface included in soap.war – `http://localhost:8080/soap/admin/index.html`

sort.dd

```
<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment" id="urn:sort1">
  <dd:provider type="java" scope="Application" methods="sort">
    <dd:java class="coen317.SortServer" static="false" />
  </dd:provider>
  <dd:faultListener>org.apache.soap.server.DOMFaultListener</dd:faultListener>
  <dd:mappings />
</dd:service>
```

JAX-WS Application Example – Step 4

- Step 4 – Deploy the web service server in an application server
  - This will allow clients to connect to the application server and request the web service, similar to connecting to the rmiregistry in RMI or the ORB in CORBA

Windows - `java –classpath soap.jar;javax.mail.jar org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servlet/rpcrouter deploy sort.dd`

Mac - `java –classpath soap.jar:javax.mail.jar org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servlet/rpcrouter deploy sort.dd`

  - To ensure the web service is deployed correctly, you can:
    - List the deployed web services on the command line

```
java –classpath soap.jar;javax.mail.jar
org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter list
```

    - View them in a browser

Open `http://localhost:8080/soap/admin/index.html`

      - Note: You can also deploy without a document descriptor through the admin interface

JAX-WS Application Example – Step 5

- Step 5 – Write the web service client code
  - The client has the code for communicating with the web service and then making what appears to be a local call to a method

SortClient.java

```
1   import java.io.*;
2   import java.net.*;
3   import java.util.*;
4   import org.apache.soap.*;
5   import org.apache.soap.rpc.*;
6
7   public class SortClient {
8     public static void main(String [] args) {
9       try {
10        URL url = new URL("http://localhost:8080/soap/servlet/rpcrouter");
11        Call call = new Call();
12        call.setTargetObjectURI("urn:sort1");
13        call.setMethodName("sort");
14        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
15        Vector params = new Vector();
16        int [] array1 = {21, 65, 87, 78, 45};
17        params.addElement(new Parameter("numbers", int[].class, array1, null));
18        call.setParams(params);
19        Response resp = call.invoke(url, "");
20        if (resp.generatedFault()) {
21          Fault fault = resp.getFault();
22          System.out.println("fault: " + fault);
23        }
24        else {
25          Parameter result = resp.getReturnValue();
26          int [] sortedArray = (int[])result.getValue();
27          for (int i=0; i < sortedArray.length; i++) {
28            System.out.println(sortedArray[i]);
29          }
30        }
31      } catch (IOException ioe) {
32        System.out.println("IOException: " + ioe.getMessage());
33      } catch (SOAPException se) {
34        System.out.println("SOAPException: " + se.getMessage());
35      }
36    }
37  }
```

JAX-WS Application Example – Steps 6, 7

- Step 6 – Compile the web service client

Windows - `javac –classpath .;soap.jar SortClient.java`

Mac - `javac –classpath .:soap.jar SortClient.java`

- Step 7 – Run the web service client

Windows - `java –classpath .;soap.jar;javax.mail.jar SortClient`

Mac - `java –classpath .:soap.jar:javax.mail.jar SortClient`

  - › Since Tomcat is already running with the deployed web service, the client just needs to be executed