

Fundamentals of Java Enterprise Components

Session: 14

Building Web Services with JAX-WS and JAX-RS

Objectives



- ▶ Describe Web services
- ▶ Describe Java APIs used in development of Web services
- ▶ Describe types of Web services
- ▶ Understand standards used to ensure interoperability of Web services
- ▶ Understand how to build Web services using JAX-RS and JAX-WS

Web Services



Web services are applications whose components are distributed over the Internet and invoked through XML messages.

The service provider and the service consumer interact with the help of standard protocols such as SOAP.

The pattern of message exchange is defined through Web Services Description Language(WSDL) document.

The message exchange between the components can be synchronous or asynchronous.

Standard protocols and technologies used in Web services:

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)
- Electronic Business Using eXtensible Markup Language (ebXML)

Java APIs for XML and Web Services



Following are the APIs used for XML document processing and implementation of Web services:

- Java API for XML processing (JAXP)
- Java API for XML based RPC (JAX-RPC)
- Java API for XML registries (JAXR)
- SOAP with Attachments API for Java (SAAJ)
- Java API for XML Messaging (JAXM)

Types of Web Services 1-2



Web services can be technically implemented as:

- Java API for XML Web Services (JAX-WS) Web services
- Representational State Transfer (REST) or RESTful Web services

JAX-WS Web services are built by using XML messages that follow SOAP standard and XML-based message architecture and formats to build the services.

Following are the essential elements of SOAP based design:

- A formal contract which describes the service interface defined through WSDL.
- A Web service specification must be defined.
- The architecture of the service should also handle asynchronous processing and invocation.

Types of Web Services 2-2



RESTful Web services are developed through the Java API for RESTful Web Services (JAX-RS) API.

A RESTful design is used if the requirement of the service meets the following conditions:

- The Web service must be completely stateless.
- It will use caching mechanisms to improve the performance of the service.
- The service provider and service consumer are allowed to mutually agree upon the dynamic content and context of the application.
- The device accessing the Web service is a hand held device where the resources and bandwidth are limited.

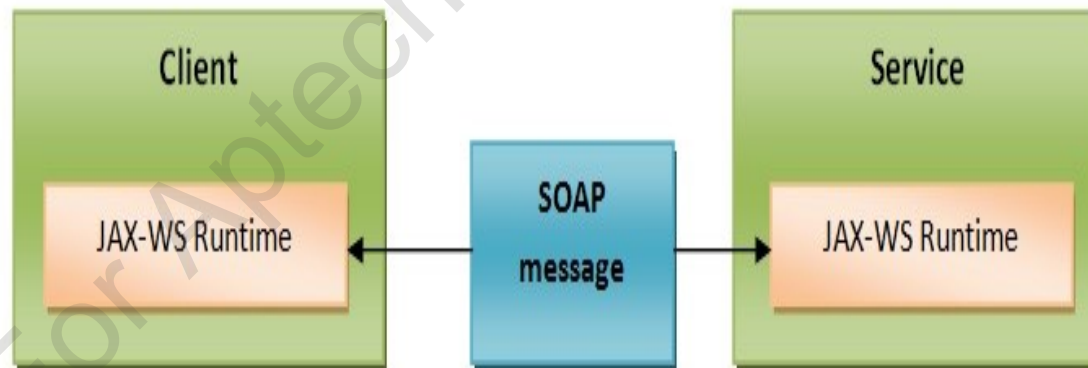
Creating a Web Service Using JAX-WS 1-2



Allows loosely coupled implementation of Web services.

The Web service is invoked through XML messages based on SOAP protocol.

Following figure shows how the Web service client and server communicate with each other:



Creating a Web Service Using JAX-WS 2-2



Following are the steps involved in creating a Web service through NetBeans IDE:

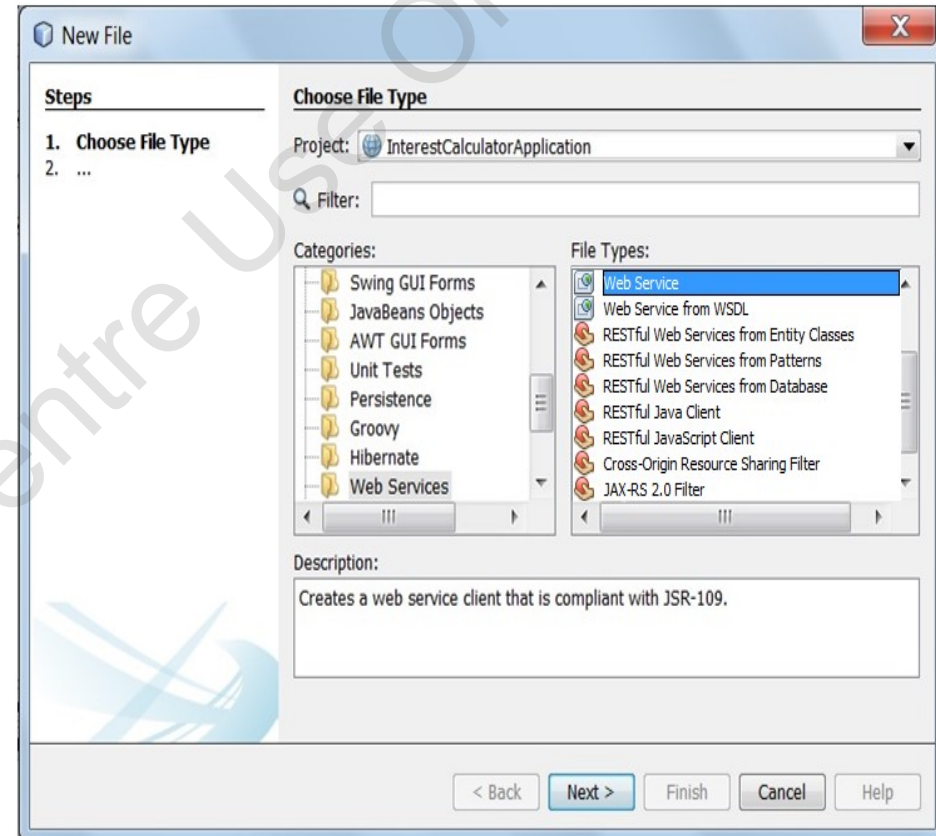
- A Web service has to be deployed in a container.
- Define the Web service by adding operations to the Web service.
- Deploy the Web service in the container and test it.
- Create a service end point to the Web service.

Creating a Web Service 1-5



In NetBeans IDE, a Web container can be created by creating a File → New project → Java Web → Web Application. Here, the Web application is named as InterestCalculatorApplication.

To create a new Web service in this Web container, right-click the Project InterestCalculatorApplication and select New → Other → Web Services → Web Service as shown in the figure.



Creating a Web Service 2-5



Click Next. The New Web Service wizard is displayed.
Provide the name of the Web service and the package in which the Web service must be created as shown in the following figure:

A screenshot of the 'New Web Service' wizard dialog box. The 'Steps' pane on the left shows '1. Choose File Type' and '2. Name and Location'. The 'Name and Location' section contains the following fields: 'Web Service Name' with the value 'InterestCalculatorWS', 'Project' with 'InterestCalculatorApplication', 'Location' with 'Source Packages', and 'Package' with 'org.me.interestcalculator'. There are two radio buttons: 'Create Web Service from Scratch' (selected) and 'Create Web Service from Existing Session Bean'. Below the radio buttons is an 'Enterprise Bean' field and a 'Browse...' button. A checkbox 'Implement Web Service as Stateless Session Bean' is checked. At the bottom are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

New Web Service

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Web Service Name: InterestCalculatorWS

Project: InterestCalculatorApplication

Location: Source Packages

Package: org.me.interestcalculator

☒ Create Web Service from Scratch

☐ Create Web Service from Existing Session Bean

Enterprise Bean: Browse...

☒ Implement Web Service as Stateless Session Bean

< Back Next > Finish Cancel Help

Select the Create Web Service from Scratch option and select the Implement Web Service as Stateless Session Bean checkbox.
Click Finish.

Creating a Web Service 3-5



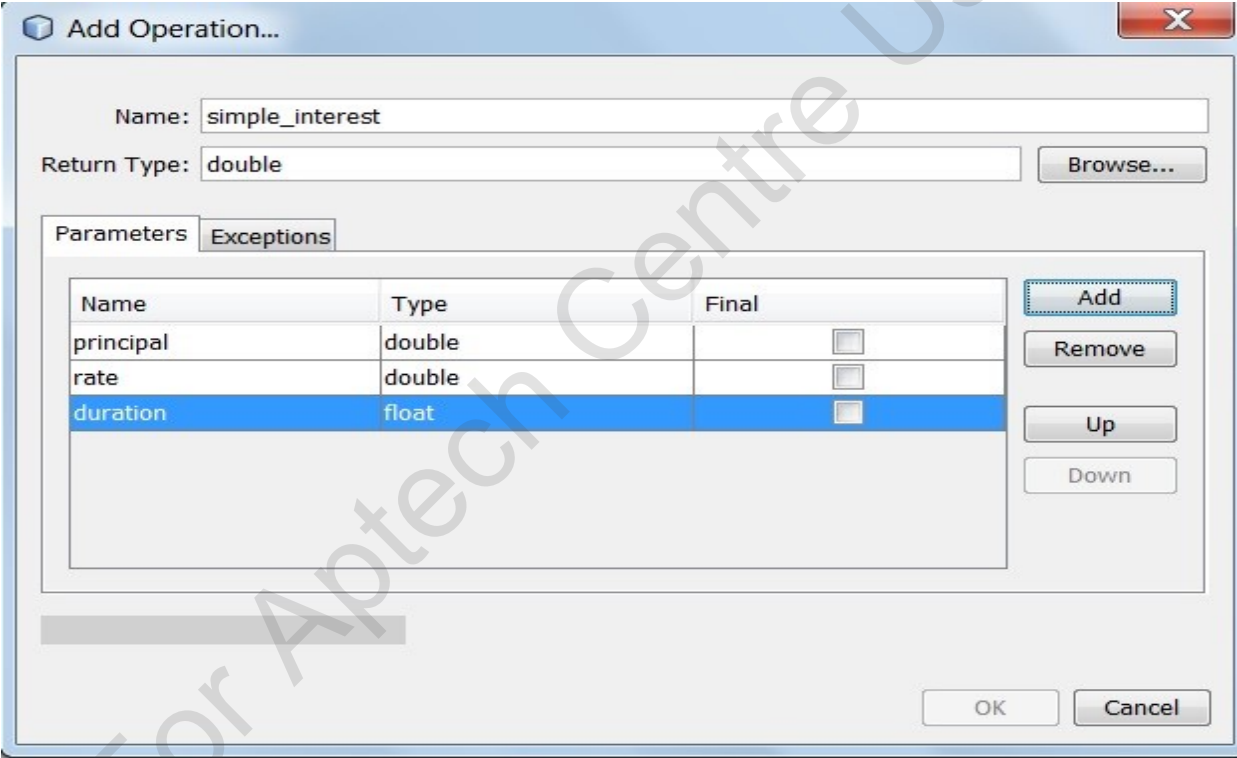
Once the Web service named InterestCalculatorWS.java is created, select the Design view of the Web service to add operations to the Web service as shown in the following figure:



Creating a Web Service 4-5



Add an operation to the Web service by clicking **Add Operation** window. This will open a wizard to add operations to the Web service as shown in the following figure:



The "Add Operation..." dialog box is shown. It has a title bar with a close button (X). The "Name" field contains "simple_interest". The "Return Type" field contains "double", with a "Browse..." button to its right. Below these fields are two tabs: "Parameters" (selected) and "Exceptions". The "Parameters" tab contains a table with three columns: "Name", "Type", and "Final". The table has three rows: "principal" (double), "rate" (double), and "duration" (float). The "duration" row is selected. To the right of the table are four buttons: "Add", "Remove", "Up", and "Down". At the bottom of the dialog are "OK" and "Cancel" buttons.

Name	Type	Final
principal	double	<input type="checkbox"/>
rate	double	<input type="checkbox"/>
duration	float	<input type="checkbox"/>

Creating a Web Service 5-5



Fill in appropriate values in the 'Add Operation' wizard to define the operation of the Web service and click OK. The Web method is added to the Web service class in the Source view as shown in the following figure:

```
@WebMethod(operationName = "simple_interest")
public double simple_interest(@WebParam(name = "principal") double principal,
    @WebParam(name = "rate") double rate, @WebParam(name = "duration") float duration) {
    //TODO write your implementation code here:
    double result = (principal*rate*duration)/100;
    return result;
}
```

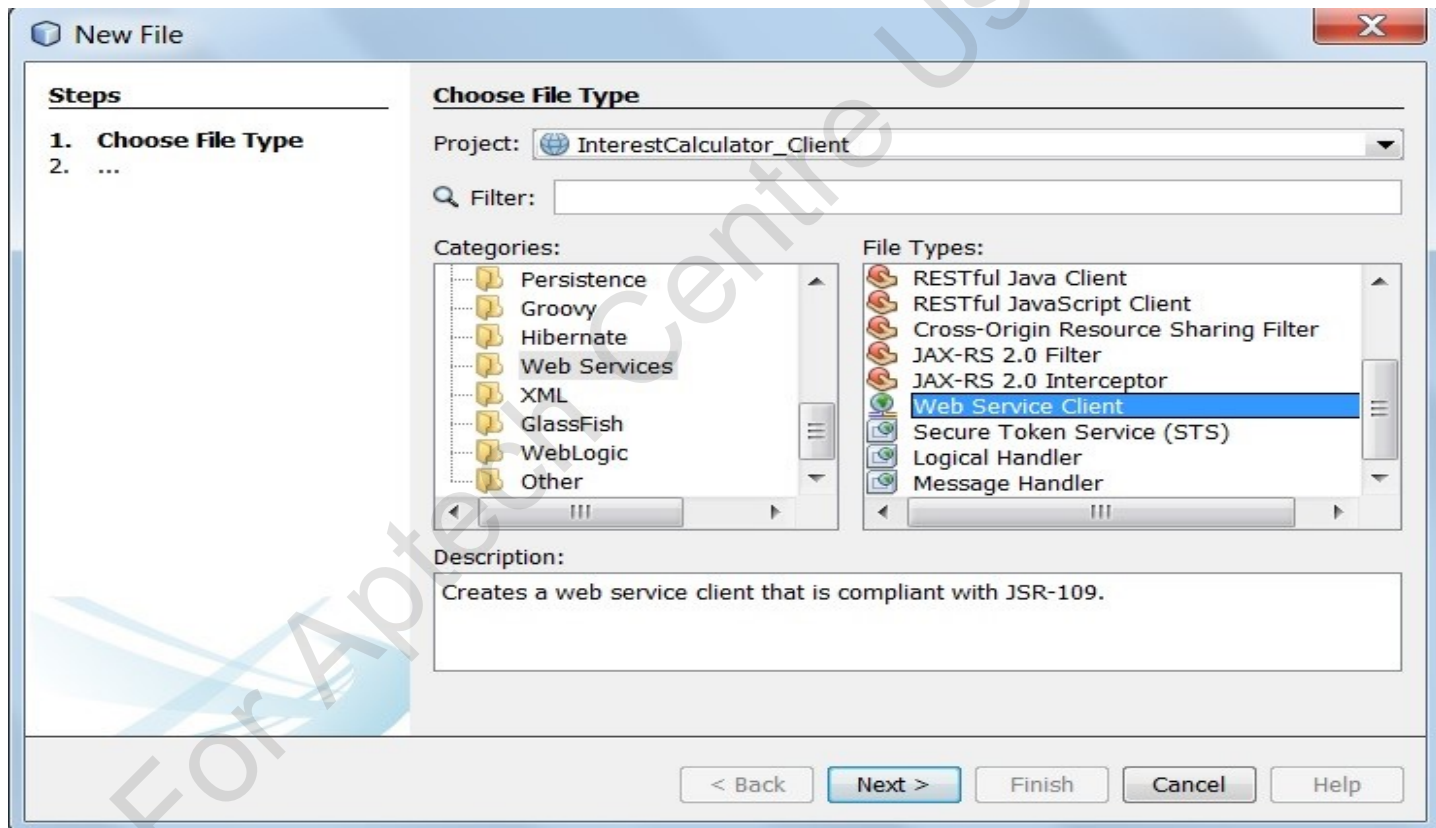
Add the following code inside the simple_interest Web method to implement the operation for calculation of simple interest:

```
.....
double result = (principal*rate*duration)/100;
return result;
.....
```


Creating a Web Service Client 1-4



Create a Web application named InterestCalculator_Client. Add a Web Service Client to the project by right-clicking the project and selecting New → Other → Web Services → Web Service Client as shown in the following figure:



Creating a Web Service Client 2-4



Selecting the Web Service Client will lead to the wizard as shown in the following figure:

The image shows a 'New Web Service Client' wizard dialog box. It has a title bar with a close button (X). On the left, a 'Steps' pane shows two steps: '1. Choose File Type' and '2. WSDL and Client Location', with the second step being the active one. The main area is titled 'WSDL and Client Location' and contains the following elements:

- A text label: 'Specify the WSDL file of the Web Service.'
- Four radio button options, each with a text label and a 'Browse...' button:
 - ☒ Project: [text field] Browse...
 - ☐ Local File: [text field] Browse...
 - ☐ WSDL URL: [text field]
 - ☐ IDE Registered: [text field] Browse...
- A text label: 'Specify a package name where the client java artifacts will be generated:'
- Two text fields:
 - Project: InterestCalculator_Client
 - Package: [dropdown menu]
- A checkbox labeled 'Generate Dispatch code' which is currently unchecked.
- An information icon (i) followed by the text: 'Enter the URL of the service you wish to use.'

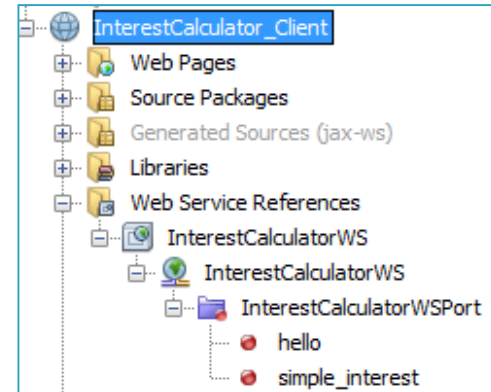
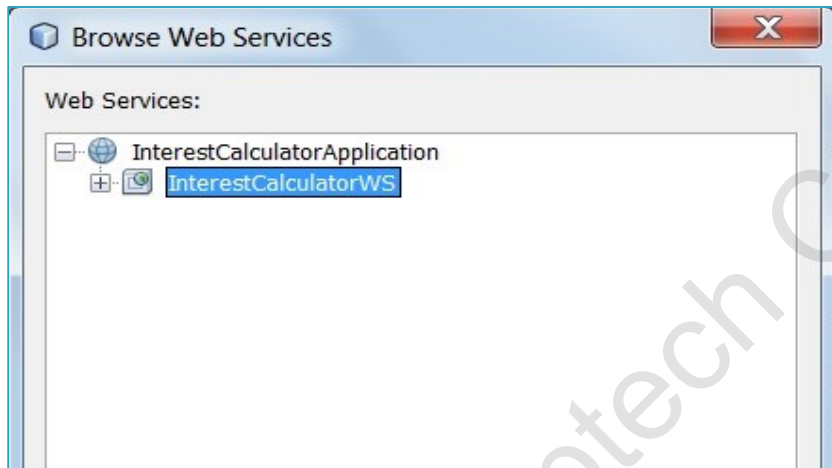
At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Creating a Web Service Client 3-4



To specify the Web Service location, click the Browse button. It displays all the available Web services as shown in the following figure:

A Web Service References directory is created in the client application folder with the reference to the Web service as shown in the following figure:



Creating a Web Service Client 4-4



Double-click InterestCalculatorWS and it will open the InterestCaculatorWS.wsdl file as shown in the following figure:

A screenshot of an IDE window titled 'InterestCalculatorWS.wsdl'. The window shows the source code of a WSDL file. The code is as follows:

```
1 <?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is 1.0.2 -->
2 <types>
3 <xsd:schema>
4 <xsd:import namespace="http://interestcalculator.me.org/" schemaLocation="http://localhost:13422/InterestCalculatorWS.xsd"/>
5 </xsd:schema>
6 </types>
7 <message name="simple_interest">
8 <part name="parameters" element="tns:simple_interest"/>
9 </message>
10 <message name="simple_interestResponse">
11 <part name="parameters" element="tns:simple_interestResponse"/>
12 </message>
13 <message name="hello">
14 <part name="parameters" element="tns:hello"/>
15 </message>
16 <message name="helloResponse">
17 <part name="parameters" element="tns:helloResponse"/>
18 </message>
19 <portType name="InterestCalculatorWS">
20 <operation name="simple_interest">
21 <input wsam:Action="http://interestcalculator.me.org/InterestCalculatorWS/simple_interestRequest" message="tns:simple_interestRequest"/>
```

The Web Service Description Language (WSDL) file holds all the information related to the Web Service whose reference was added to the client application.

Creating the JSP Page to Access the Web Service 1-3



Create a new JSP page named, index.jsp. Following code demonstrates how to modify the JSP page:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1> <!-- start web service invocation --><hr/>

    <%
    try {
      org.me.interestcalculator.InterestCalculatorWS_Service service
= new org.me.interestcalculator.InterestCalculatorWS_Service();
      org.me.interestcalculator.InterestCalculatorWS port =
service.getInterestCalculatorWSPort();
```

Creating the JSP Page to Access the Web Service 2-3



```
// TODO initialize WS operation arguments here
    double principal = 1000.0d;
    double rate = 12.0d;
    float duration = 3.0f;
    // TODO process result here
    double result = port.simpleInterest(principal, rate,
duration);
    out.println("Simple Interest is "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/></h1>
</body>
</html>
```

Creating the JSP Page to Access the Web Service 3-3



Build and run the client application. The result of running the Web service client application is shown in the following figure:



Types Supported by JAX-WS 1-3



- ▶ Java platform and XML together are ideal combination for building portable Web services and Web service application clients.
- ▶ The Java Architecture for XML Binding (JAXB) API is used for accessing XML documents from Java applications.
- ▶ JAXB handles the access of XML documents and their processing, and also handles the generation of XML documents from Java applications.
- ▶ Following table shows the mapping of XML schema to Java data types:

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	Int
xsd:long	Long
xsd:short	Short
xsd:decimal	java.math.BigDecimal
xsd:float	Float
xsd:double	Double

Types Supported by JAX-WS 2-3



XML Schema Type	Java Data Type
xsd:Boolean	Boolean
xsd:byte	Byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	Long
xsd:unsignedShort	Int
xsd:unsignedByte	Short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

Types Supported by JAX-WS 3-3



Following table shows the mapping of primitive Java data types to XML schema entities:

Java Class	XML Data Type
java.lang.String	xs:string
java.math.BigInteger	xs:integer
java.math.BigDecimal	xs:decimal
java.util.Calendar	xs:dateTime
java.util.Date	xs:dateTime
javax.xml.namespace.QName	xs:QName
java.net.URI	xs:string
javax.xml.datatype.XMLGregorianCalendar	xs:anySimpleType
javax.xml.datatype.Duration	xs:duration
java.lang.Object	xs:anyType
java.awt.Image	xs:base64Binary
javax.activation.DataHandler	xs:base64Binary
javax.xml.transform.Source	xs:base64Binary
java.util.UUID	xs:string

RESTful Services



Representational State Transfer or REST is an architectural style for designing Web services.

REST emphasizes stateless client-server architecture where Web services are seen as resources and are accessed through Universal Resource Identifiers (URIs).

Following are the important features of RESTful services:

- Using URIs for resource identification
- Using HTTP methods for resource manipulation - The RESTful design defines a one-to-one mapping between the database operations and the HTTP methods to provide a uniform interface
- The Service provider and Service consumer are stateless
- Self descriptive messages

Creating a RESTful Root Resource Class 1-3



Root resource classes are the entry point for a JAX-RS Web service.

Following are the requirements for a Java class to be a root resource class:

- The class must be annotated with `@Path` annotation.
- The class should have a public constructor which can be invoked at the runtime.
- Any one of the root resource class methods should be mapped to the HTTP methods.

Following code demonstrates a root resource class for admission service of a university:

```
package demo.jaxrs.server;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
@Path("/admissionservice/")
```

Creating a RESTful Root Resource Class 2-3



```
public class AdmissionService{
    public AdmissionService()    {
        ...
    }
    @GET
    @Path("/studentname/")
    @Produces("MediaType.TEXT_PLAIN ")
    public String getStudName(@QueryParam("Roll_no") String Roll_no){
        ...    }
    @GET
    public Student getStudent(@QueryParam("Roll_no") String Roll_no){
        ...
    }
    @DELETE
    public Response deleteStudent(@QueryParam("Roll_no") String Roll_no){
        ...    }
    @PUT
    public Response updateStudent(Student stud){
        ...    }
    @POST
    public Response addStudent(Student stud) {
        ...    }
}
```

Creating a RESTful Root Resource Class 3-3



Following table shows all the annotations used in JAX-RS Web services:

Annotation	Description
@Path	The method marked with this annotation specifies the location of the root resource class based on which the URI can be created for all the data to be accessed on the Web service.
@GET	The method marked with this annotation maps the function implemented in the method to a HTTP GET request.
@POST	The method marked with this annotation maps to the HTTP POST method and the method definition acts according to the resource which is being posted to the Web server.
@PUT	The method marked with this annotation maps to the HTTP PUT method and performs operations according to the resource which is put to the Web service.
@DELETE	The method marked with this annotation maps to the DELETE HTTP method.
@HEAD	The method marked with this annotation is used to process HTTP headers.
@PathParam	The method marked with this annotation is used to extract parameters with respect to the path of the Web service from the URI for further processing.
@QueryParam	The method marked with this annotation extracts the parameters used for querying the Web service from the URI.
@Consumes	The method marked with this annotation represents the MIME types that can be consumed by a resource in the Web service.
@Produces	The method marked with this annotation specifies the MIME types that can be produced by the resource in the Web service.

Overview of Client API for Accessing RESTful Resources with JAX-RS API



Client API is used by developers to create clients for RESTful services. This API is defined as `javax.ws.rs.client` package.

Following are the steps for creating a basic client request using the Client API:

- Import the `javax.ws.rs.client` package to the class which is supposed to be the client of the Web service and obtain an instance of `javax.ws.rs.client.Client`.
- The Client instance must be configured with the target, where the target is the Web service.
- Create a request for the Web service based on the configured target and invoke the request.

Following is an example of a request:

```
Client client = ClientBuilder.newClient();  
String name =  
client.target("http://College.com/admissionservice/studentname")  
request(MediaType.TEXT_PLAIN)  
get(String.class);
```

Summary



- ▶ Web services are Web applications whose components are distributed over the Internet and are mutually invoked through XML messages.
- ▶ JAX-WS and JAX-RS are the APIs in Java which are used to develop SOAP based and RESTful services respectively.
- ▶ SOAP based Web services have a security specification and WSDL contract which defines the service.
- ▶ Interoperability of Web services is achieved through standard protocols and standard methods of communication.
- ▶ RESTful services are stateless services and the request parameters for the service methods are passed through the URIs.
- ▶ Annotations are used in Web services for resource injection and to generate metadata.