

# **WEB SERVICES**

**By:**

**Mr. Satish.B**

**Naresh I Technologies**

Name: .....

Batch Time: .....

**sri raghavendra Xerox**

**All software language materials available**

**beside banglore iyyengars bakery opp:cdac balkampetroad ameerpet Hyderabad**

**cell :9951596199**



**JAXP,JAXB**  
**WEB SERVICES**  
**SOAP UI TOOL**  
**WSDL**  
**JAVA API**  
**RESTFULL WEB SERVICES**



## **JAX-P**

The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language.

JAXP leverages the parser standards Simple API for XML Parsing (SAX) and Document Object Model (DOM) so that you can choose to parse your data as a stream of events or to build an object representation of it.

JAXP also supports the Extensible Stylesheet Language Transformations (XSLT) standard, giving you control over the presentation of the data and enabling you to convert the data to other XML documents or to other formats, such as HTML.

The current version of JAXP is 1.6 that is coming with Java SE 8.0

As of version JAXP-1.4, JAXP implements the Streaming API for XML (StAX) standard.

The Java API for XML Processing (JAXP) enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation.

JAXP was developed under the Java Community Process as JSR 5 (JAXP 1.0)

<b>jdk versions</b>	<b>Jaxp- versions</b>
1.4	1.1
1.5	1.3
1.6	1.4
1.7.0	1.4.5
1.7.40	1.5
1.8	1.6

## **Overview of the JAX-P Packages**

The SAX and DOM APIs are defined by the XML-DEV group and by the W3C, respectively. The libraries that define those APIs are as follows:

- **javax.xml.parsers:** The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.
- **org.w3c.dom:** Defines the Document class (a DOM) as well as classes for all the components of a DOM.
- **org.xml.sax:** Defines the basic SAX APIs.
- **javax.xml.transform:** Defines the XSLT APIs that let you transform XML into other forms.
- **javax.xml.stream:** Provides StAX-specific transformation APIs.

Sun has given one API called JAX-P(java API for XML processing) to read and write the data from the xml document.

JAX-P API is implemented by multiple vendors

--cml4j (xml4j.jar) for IBM

--xcerser(xcerser.jar)from apache

--xmlbeans(xmlbeans.jar)from apache

--crimson(crimson.jar)from apache

--parserv2(xmlparser2.jar)from apache

From JDK1.5 version sun has given implementation for JAXP with JDK itself.

Jdk itself has xcercises.jar(Apache)implementation.

->we have 3 types of parsers

1. DOM (Document Object Model) parser

2. SAX (Simple API for XML) parser

3. STAX parser(Streaming API For Xml)

-> In addition to above parsers sun has given one API to read and write the data 'from/into' XML document in the form of java object is called as JAXB (Java Architecture for XML binding)

### **Responsibilities of parsers:**

1. Reading the content form the xml document and writing the content into the xml document.

2. Checking well form of XML document

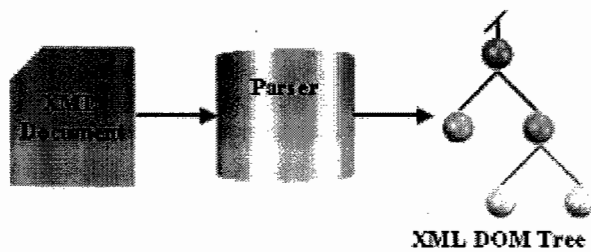
3. Verifying xml doc is valid or not against to DTD or XSD

4. Parsing The XML documents.

### **DOM (Document Object Model) Parser:**

->**DOM is an object-oriented API.**

->The DOM parser builds a tree structure which represents an XML document.



DOM parser Loads an entire XML document into memory at a time , modeling it with Object for easy nodal traversal.

DOM Parser is slower Parser and consumes a lot of memory

**DOM supports read and write operations**

**DOM supports Sequential Access and Random Access.**

### Imp Points:-

A DOM is a standard tree structure, where each node contains one of the components from an XML structure. The two most common types of nodes are element nodes and text nodes.

By Using DOM functions you create nodes, remove nodes, change their contents, and traverse the node hierarchy.

### **steps to use DOM parser for reading xml document:**

1. Create DocumentBuilderFactory object

To create DocumentBuilderFactory object we can use the following method

```
public static DocumentBuilderFactory newInstance()
```

EX: DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

->DocumentBuilderFactory is an abstract class implemented by using Abstract Factory Design Pattern.

->DocumentBuilderFactory Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

2. Create "DocumentBuilder" object from DocumentBuilderFactory object.

For this we can use the following method from DocumentBuilderFactory class

```
public DocumentBuilder newDocumentBuilder() throws ParserConfigurationException
```

ex:- DocumentBuilder db=dbf.newDocumentBuilder();

DocumentBuilder Defines the API to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a Document from XML.

Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

3. Parse the xml document with file path as a file inputstream/file obj

To parse the Xml document we can use the following method from DocumentBuilder class

```
public Document parse(File file)throws IOException,SAXException
```

```
public Document parse(InputStream is) IOException,SAXException
```

Ex:-

```
Document doc=db.parse(new File("E:\\employee.xml"));
```

**Here document object contains set of nodes in the form of tree structures and each node contains each element content.**

**After getting the Document object we can read the data in sequential Manner/RandomAccess manner as per the requirements.**

If we want to get root element from the document object then we need to invoke "getDocumentElement()" method from document object like below

Ex:-

```
Element rootElement =doc.getDocumentElement();
```

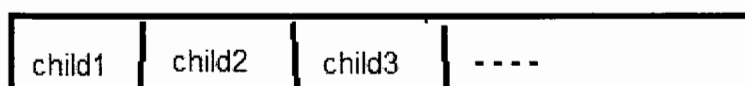
**Getting child elements from element or Node:**

1.To get the child element from Node or Element we need to invoke "getChildNodes()" method on element object or Node object

```
Document doc=documentBuilder.parse("emp.xml");
```

```
Element rootElement= documentBuilder.getDocumentElement();
```

```
NodeList rootchildnodes=rootelement.getChildNodes();
```





here child node or elements index will be start from "0".

2. Here the NodeList contains set of child nodes and to get one by one child from the NodeList we need to invoke "item(int index)" method on the NodeList by pass in NodeIndex.

Ex:-

```
if(rootElement.hasChildNodes()){
    NodeList nodeList= rootElement.getChildNodes();
    for(int i=0;i<nodeList.getLength();i++){
        Node node=nodeList.item(i);
        if(node.getNodeType()==Node.ELEMENT_NODE){
            System.out.println(node.getNodeName()+"-->"+node.getTextContent());
        }
    }
}
```

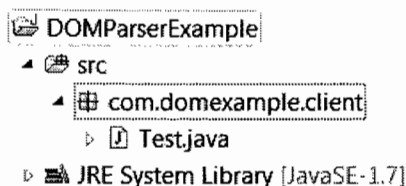
To get text data from the Element/Node first we need to travels up to this textNode and then we need to invoke "getTextContent()" on the textNode/textElement

### Example:

employee.xml file

```
=====
<employee>
<empNo>1001</empNo>
<name>rani</name>
<salary>9089</salary>
</employee>
```

**An example program to read above XML in Sequential Access By using DOM parser.**



```
package com.domexample.client;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class Test {
    public static void main(String[] args) {
        DocumentBuilderFactory builderFactory=
```

```

DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder documentBuilder
        =builderFactory.newDocumentBuilder();
    File file=new File("E://xmlexamples/employee.xml");
    Document document=documentBuilder.parse(file);
    //Document object representing the XML elements in Tree Structured Manner
    //steps --> sequential Access
    //1) get rootElement from Document obj
    Element rootElement=document.getDocumentElement();
    //Element is a child interface of Node interface
    System.out.println(rootElement.getNodeName());
    //2) get child elements from above Element
    if(rootElement.hasChildNodes()){
        NodeList nodeList= rootElement.getChildNodes();
        for(int i=0;i<nodeList.getLength();i++){
            Node node=nodeList.item(i);
            if(node.getNodeType()==Node.ELEMENT_NODE){
                System.out.println(node.getNodeName()+"-->"+node.getTextContent());
            }
        }
    }
} catch (ParserConfigurationException | IOException|SAXException e) {
    e.printStackTrace();
}
}
}

```

### **Example2:-**

```

<employees>
<employee empNo="101" name="raja" salary="19000">
<deptNo>12</deptNo>
<deptName>sales</deptName>
</employee>
<employee empNo="102" name="roja" salary="21000">
<deptNo>12</deptNo>
<deptName>sales</deptName>
</employee>
<employee empNo="103" name="rani" salary="11000">
<deptNo>13</deptNo>
<deptName>IT</deptName>
</employee>
</employees>

```

**An example program to read above XML Elements in Random Access By using DOM parser.**

```

package com.domexample.client;
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

```

```

import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class Test {
public static void main(String[] args) {
DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
try {
    builder = factory.newDocumentBuilder();
    Document document=builder.parse(new File("E://xmlexamples/employee.xml"));
    //steps for Random Access
    NodeList nodeList
    =document.getElementsByTagName("employee");
    for(int i=0;i<nodeList.getLength();i++){
    Node employeeNode=nodeList.item(i);

    System.out.println("Element :"+employeeNode.getNodeName());

    if(employeeNode.hasAttributes()){
    System.out.println(employeeNode.getNodeName()+" Attributes ");
    NamedNodeMap attributes=employeeNode.getAttributes();
    for(int k=0;k<attributes.getLength();k++){
        Node attributeNode=attributes.item(k);

    System.out.println(attributeNode.getNodeName()+"="+attributeNode.getNodeValue());
    }
    }

    if(employeeNode.hasChildNodes()){
    System.out.println(employeeNode.getNodeName()+" Child Elements ");

    NodeList childNodeList=employeeNode.getChildNodes();
    for(int j=0;j<childNodeList.getLength();j++){
        Node childNode=childNodeList.item(j);
        if(childNode.getNodeType()==Node.ELEMENT_NODE){
    System.out.println(childNode.getNodeName()+"-->"+childNode.getTextContent());
        }
    }
    }
    System.out.println("-----");
    }

} catch (ParserConfigurationException | SAXException | IOException e) {
    e.printStackTrace();
}

}

}

```

**Example 3:-**

## An example program to validate XML Elements in Against to DTD by using DOM parser

employee.dtd

```
-----
<!ELEMENT employees (employee*)>
<!ELEMENT employee(deptNo,deptName)>
<!ELEMENT deptNo (#PCDATA)>
<!ELEMENT deptName (#PCDATA)>
<!ATTLIST employee empNo ID #REQUIRED>
<!ATTLIST employee name CDATA #IMPLIED>
<!ATTLIST employee salary CDATA #IMPLIED>
```

employee.xml

```
-----
<!DOCTYPE employees SYSTEM "E://xmlexamples/employee.dtd">
<employees>
<employee empNo="E101" name="raja" salary="19000">
<deptNo>12</deptNo>
<deptName>sales</deptName>
</employee>
</employees>
```

### ValidateXmlAsPerDTDTest.java

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
public class ValidateXmlAsPerDTDTest {
public static void main(String[] args)
throws ParserConfigurationException,IOException {
DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
factory.setValidating(true);
//must be the set validating value as true to validate the Xml as per DTD
DocumentBuilder builder=factory.newDocumentBuilder();
builder.setErrorHandler(new MyErrorHandler());
try{
Document document=builder.parse(new File("E://xmlexamples/employee.xml"));
System.out.println("Document is valid");
}catch(SAXException se){
System.out.println("Document is invalid");
System.out.println(se.getMessage());
}
}
}
```

By using DefaultError Handler DOM parser will display first 10 Error Messages .

If we want to display all the Errors we can Develop CustomError Handler class By implementing ErrorHandler interface and set the CustomError Handler class object

into **DocumentBuilder** obj.

### **MyErrorHandler.java:-**

```
DocumentBuilder builder=factory.newDocumentBuilder();
builder.setErrorHandler(new MyErrorHandler());
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
public class MyErrorHandler implements ErrorHandler{
    @Override
    public void warning(SAXParseException exception)
        throws SAXException {
        System.out.println("warning");
        throw new SAXException(exception);
    }
    @Override
    public void error(SAXParseException exception) throws SAXException {
        System.out.println("error");
        throw new SAXException(exception);
    }
    @Override
    public void fatalError(SAXParseException exception) throws SAXException {
        System.out.println("fatalError");
        throw new SAXException(exception);
    }
}
```

### **Example4:**

**An example program to validate XML Elements in Against toXSD by using DOM parser**

### **employees.xml**

```
<employees
xmlns="http://www.empinfo.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.empinfo.com E://xml3pm/employees.xsd">
<employee empNo="101" name="raja" salary="19000">
<deptNo>12</deptNo>
<deptName>sales</deptName>
</employee>
</employees>
```

### **employees.xsd**

```
<schema targetNamespace="http://www.empinfo.com"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<element name="employees">
<complexType>
<sequence>
<element name="employee" maxOccurs="unbounded" minOccurs="0">
```

```

<complexType>
<sequence>
<element name="deptNo" type="integer"/>
<element name="deptName" type="string"/>
</sequence>
<attribute name="empNo" type="integer" use="required"/>
<attribute name="name" type="string"/>
<attribute name="salary" type="decimal"/>
</complexType>
</element>
</sequence>
</complexType>
</element>
</schema>

```

```

import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
public class ValidateXmlAsPerXSDTest {
public static void main(String[] args) throws ParserConfigurationException,IOException {
DocumentBuilderFactory documentBuilderFactory=DocumentBuilderFactory.newInstance();
documentBuilderFactory.setNamespaceAware(true);
//to validate the XML as per XSD must be set setNamespaceAware value as true
DocumentBuilder builderFactory=documentBuilderFactory.newDocumentBuilder();
builderFactory.setErrorHandler(new MyErrorHandler());
try {
Document document=builderFactory.parse("E://xmlexamples/employees.xml");

SchemaFactory
schemaFactory=SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
Schema schema=schemaFactory.newSchema();
Validator validator=schema.newValidator();
validator.validate(new DOMSource(document));
System.out.println("Document is valid");
} catch (SAXException e) {
System.out.println("Document is invalid");
System.out.println(e.getMessage());
}
}
}
}

```

### **Example5**

**An example program to write data to XML document By using DOM parser**

```

import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
public class WriteXmlTest{
public static void main(String[] args)
throws Exception{
DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
DocumentBuilder builder=factory.newDocumentBuilder();
Document document=builder.parse(new File("E://xmlexamples/employees.xml"));
Node employee=document.getElementsByTagName("employee").item(0);
NamedNodeMap attr=employee.getAttributes();
Node nodeAttrNode=attr.getNamedItem("id");
nodeAttrNode.setTextContent("201");
Element age=document.createElement("age");
    age.appendChild(document.createTextNode("28"));
    employee.appendChild(age);
    NodeList list=employee.getChildNodes();
    for(int i=0;i<list.getLength();i++){
        Node node=list.item(i);
        if(node.getNodeName().equals("salary")){
            node.setTextContent("200000");
        }
        if(node.getNodeName().equals("nickname")){
            employee.removeChild(node);
        }
    }
TransformerFactory tfactory=TransformerFactory.newInstance();
Transformer transformer=tfactory.newTransformer();
StreamResult result=new StreamResult(new File("E:\\xmlexamples\\ModifyEMP.xml"));
transformer.transform(new DOMSource(document), result);
    System.out.println("Done");
}
}

```

### **SAX parser(Simple API for XML)**

SAX (the Simple API for XML) is an event-based parser for xml documents. Unlike a DOM parser, a SAX parser creates no parse tree.

-->It is an event based parser.

-->it is used to perform only read operations .

→It supports only Sequential Access .

->It consumes less memory and performance is good compared to DOM.

-->SAX 2.0 validates DTO as well as schema.

#### Disadvantages of SAX

- We have no random access to an XML document since it is processed in a forward-only manner
- If you need to keep track of data the parser has seen or change the order of items, you must write the code and store the data on your own

Steps to write an application using SAX parser we will read the entire xml documents.

1. create SAXParserFactory object by using newInstance() method of SAXParserFactory class

2 create SAXParser object by using newSAXParser() method of SAXParserFactory class

Example:

```
SAXParserFactory factory = SAXParserFactory.newInstance();  
SAXParser saxParser = factory.newSAXParser();
```

3. Create a CustomHandler class and Handle the Events

```
MyHandler myHandler=new MyHandler();
```

4. Parse the XML document

For this we can use the following method

```
public void parse(InputStream is,DefaultHandler dh)throws IOEx,SAXEx
```

example:-

```
saxParser.parse(new FileInputStream("emp.xml"),myHandler);
```

#### --> How to write MyHandler class?

Ans- Our MyHandler class should subtype from org.xml.sax.helper.DefaultHandler.

--> in the Handler class override the required methods as per your requirement

startDocument()- to receive notification at the start of document.

endDocument()-To receive notification at the end of the document.

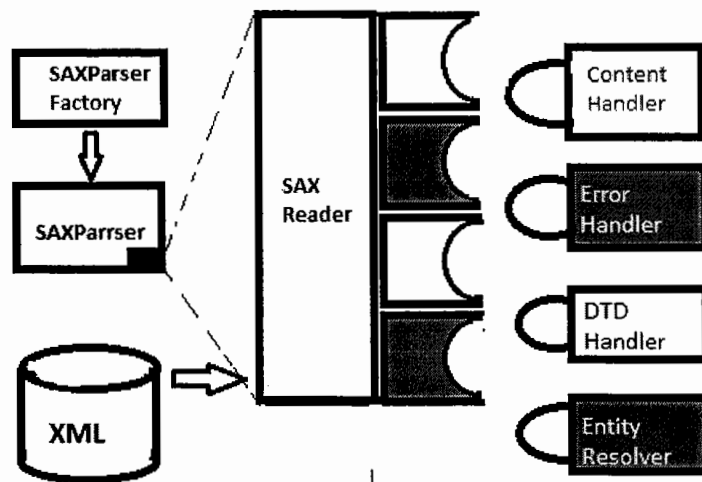
StartElement(String uri, String localeName, String qualifiedName, Attributes attribute): To receive notification at start of the element.

endElement(String uri,String localeName,String qualifiedName): To receive the notification at the end of the element.



characters(char[] c, int start, int length): To receive notification of element data.

### SAX Parser Architecture :



```
<employees>
```

```
<employee name="raja" salary="19000">
```

```
<deptNo>12</deptNo>
```

```
<deptName>sales</deptName>
```

```
</employee>
```

```
</employees>
```

#### • SAXParserExample

##### • src

##### • (default package)

▸ MyHandler.java

▸ Test.java

### MyHandler.java

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
public class MyHandler extends DefaultHandler {
public void startDocument(){
    System.out.println("Document started");
}
public void endDocument(){
    System.out.println("Document Ended");
}
```

```

public void startElement (String uri, String localName,
    String qName, Attributes attributes)
throws SAXException
{
    System.out.println("Start Element Name -->" + qName);
    if(attributes.getLength() > 0){
        System.out.println("Attributes :");
        for(int i=0; i < attributes.getLength(); i++){
            String attrName = attributes.getQName(i);
            String attrValue = attributes.getValue(i);
            System.out.println(attrName + "=" + attrValue);
        }
    }
}

public void endElement(String uri, String localName, String qName){
    System.out.println(" End Element Name -->" + qName);
}

public void characters (char ch[], int start, int length) throws SAXException {
    String s = new String(ch, start, length);
    if(s.trim().length() > 0){
        System.out.println(s);
    }
}

public void error(SAXParseException e) throws SAXException{
    throw new SAXException(e);
}

```

### Test.java

```

import java.io.File;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;

public class Test {
    public static void main(String[] args)
        throws ParserConfigurationException, IOException {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(true);
        try{
            SAXParser saxParser = factory.newSAXParser();
            MyHandler myHandler = new MyHandler();
            saxParser.parse(new File("E://xmlexamples/employee.xml"), myHandler);
            System.out.println("Document valid");
        } catch (SAXException e) {
            System.out.println("Document is invalid");
            System.out.println(e);
        }
    }
}

```



JAXB is an acronym derived from *Java Architecture for XML Binding*.

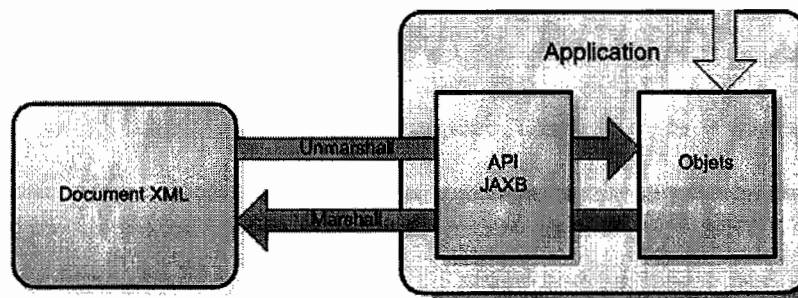
It is used to convert XML to java object and java object to XML.

**JAXB allows Java developers to access and process XML data without having to know XML or XML processing. For example, there's no need to create or use a SAX parser or write callback methods.**

The first version of JAXB is 1.0 and the current version is 2.0 version

There are two operations you can perform using JAXB

1. **Marshalling** :Converting a java object to XML .
2. **UnMarshalling** :Converting a XML to java object



### Features of JAXB 2.0

JAXB 2.0 includes several features that were not present in JAXB 1.x. They are as follows:

**1) Annotation support:** JAXB 2.0 provides support to annotation so less coding is required to develop JAXB application. The `javax.xml.bind.annotation` package provides classes and interfaces for JAXB 2.0.

**2) Support for all W3C XML Schema features:** it supports all the W3C schema unlike JAXB 1.0.

### JAXB-B Annotations

### 1) @XmlRootElement

Define the root element for the XML to be produced with @XmlRootElement JAXB annotation. The name of the root XML element is derived from the class name.

1 @XmlRootElement

2 public class Employee implements Serializable {

3       -----...

4 }

You can also specify the name of the root element of the XML using its name attribute, for example @XmlRootElement(name = "employee")

### 2) @XmlElement

This is a field and method (getter method) level annotation.

Used to declare the element for this complex/simpleType

1 @XmlElement

2 public String getName() {

3     return name;

4 }

When we are using @XmlElement at field level we can must be declare @XmlAccessorType(XmlAccessType.FIELD) at the class level.

When we are using @XmlElement getter method level @XmlAccessorType annotation is not required.

### 3) @XmlAttribute

This is a field and method (getter method) level annotation.

Used to declare the attribute for this complex/simpleType

@XmlAttribute

public String getName() {

    return name;

}

### 4) @XmlType

Specify the order in which XML elements output will be produced.

1 @XmlRootElement

2 @XmlType(propOrder = { "id", "firstName", "lastName", "email", "salary" })

3 public class Employee implements Serializable {

4...

5}

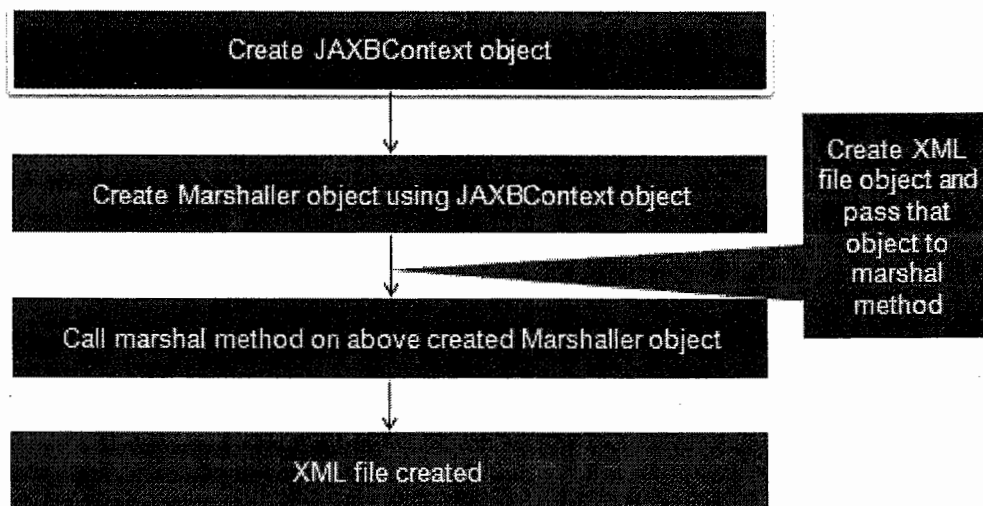
### **Steps:-**

Step1:- create a Binding class(It is a Java Bean based class) and denote the required annotations

Step2:- in the client application

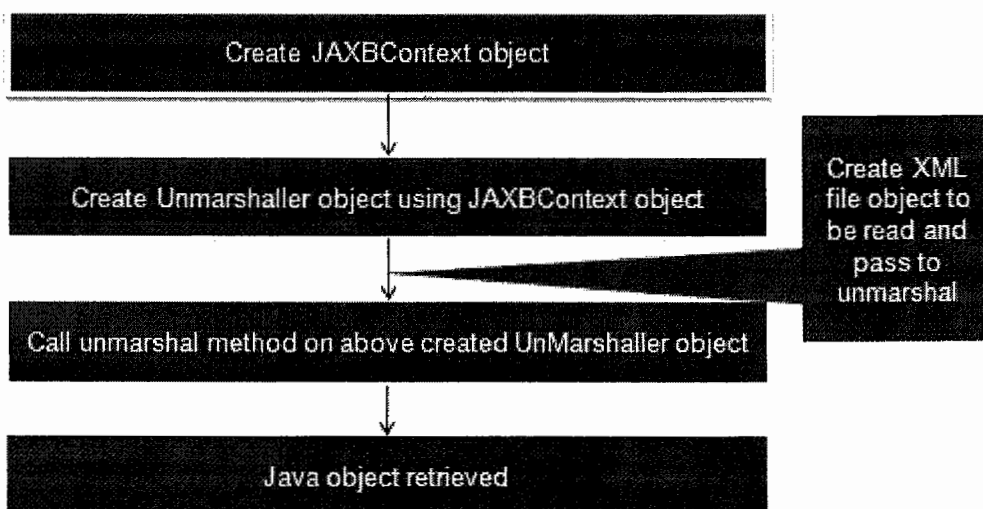
- (i) Create JAXBContext object
- (ii) Perform the Marshalling and UnMarshalling operations as per your requirements

### ***For Marshalling:***



### ***For Unmarshalling:***

The



- JAXBExample
  - src
    - com.nareshit.domain
      - Employee.java
      - MarshallerTest.java
      - MarshallerTest1.java
      - UnMarshallerTest.java
  - JRE System Library [JavaSE-1.7]
    - employee.xml

Write The Binding class as follows

Employee.java

```

package com.nareshit.domain;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name="employee")
public class Employee {
    private int empNo;
    private String name;
    private double salary;
    @XmlAttribute(name="empNo")
    public int getEmpNo() {
        return empNo;
    }
    public void setEmpNo(int empNo) {
        this.empNo = empNo;
    }
    @XmlElement(name="name")
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @XmlElement
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
}

```

### MarshallerTest.java

```

package com.nareshit.domain;

import java.io.File;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;

```

```

import javax.xml.bind.Marshaller;

public class MarshallerTest {

public static void main(String[] args) {

Employee emp=new Employee();

emp.setEmpNo(101);

emp.setName("anuska");

emp.setSalary(18000);

try {

    JAXBContext jaxbContext=

    JAXBContext.newInstance(Employee.class);

    Marshaller marshaller=jaxbContext.createMarshaller();

    marshaller.marshal(emp,new File("employee.xml"));

} catch (JAXBException e) {

    e.printStackTrace();

}

}

}

```

If you execute the above class then employee.xml file is generating and representing the Employee Object data

The following code gives Xml data in String format.

### **MarshallerTest1.java**

```

package com.nareshit.domain;
import java.io.StringWriter;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
public class MarshallerTest1 {
public static void main(String[] args) throws JAXBException{
    Employee emp=new Employee();
    emp.setEmpNo(101);
    emp.setName("anuska");
    emp.setSalary(18000);
    JAXBContext context=JAXBContext.newInstance(Employee.class);
    Marshaller marshaller=context.createMarshaller();
    StringWriter writer = new StringWriter();
    marshaller.marshal(emp, writer);
    String xml = writer.toString();
}
}

```

```

    System.out.println(xml);
}
}

```

The following example shows how to convert the XML data into java object  
**UnMarshallerTest.java**

```

package com.nareshit.domain;
import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
public class UnMarshallerTest {
    public static void main(String[] args) {
        System.out.println("Unmarshalling");
    try {
        JAXBContext jaxbContext=
            JAXBContext.newInstance(Employee.class);
        Unmarshaller unmarshaller=
            jaxbContext.createUnmarshaller();
        Employee emp=(Employee)unmarshaller.unmarshal(new File("employee.xml"));
        System.out.println(emp.getEmpNo()+" "+emp.getName()+" "+emp.getSalary());
    } catch (JAXBException e) {
        e.printStackTrace();
    }

    }
}

```

Note:- If we don't want to write the Binding class then we can write XSD and generate the Binding class with the Annotation mapping.

To generate the Binding class based on XSD we can use XJC tool from jdk s.w

Write employee.xsd and store in your project src folder

Usage : E://JAXBExamples> xjc -d <destination> schemafile

E://JAXBExamples> xjc -d src src/employee.xsd

Then after generating Binding class we can perform Marshalling and UnMarshalling operations.



A WebService is a Service which is accessible Over a network. Web service is a way of communication (OR) a technology. Web Services allows us to build interoperable Distributed applications.

for example,

With a java based application if we want to Communicate with a .Net based application we can use webservices. The communication can be done through a set of XML messages over HTTP protocol.

### **what is interoperability?**

Interoperability means language independent and platform independent

### **What is the difference between a web application and web service?**

A Web Application is a server side application which provide the services directly to enduser(customer).

A webservice is also server side application which provides it's services to another application but not to end users directly.

For web applications browser is acting as a client.

For Web Services applications an application is acts as a client.

Webservice applications are distributed applications

A Distributed System is a Collection of independent system process communicating with one another by exchanging the messages/DATA.

The Distributed applications will provide services to multiple clients.

### **WebServices :-**

Web Services also allows a program to expose objects over the Network, but the difference between other distributed Objects (RMI(OR)EJB) and web services distributed objects is these are not only platform (OS) independent, these are language independent. Which means you can write a Web Service Object using c,c++,php,java,.net and can expose over the Network.

In order to access this, the other program could be written in any of the programming languages like C(OR) c++ (OR) java (OR) .net(OR) php. Anything that is accessible irrespective of platform and programming language is called Interoperability. This means we are building distributed interoperable programs using Web Services.

WebServices are given by the open community organization, called WS-I, stands for WebService Interoperability Org. WS-I Org Given Two Versions of WebServices specifications.that is

1) Basic Profile 1.0(B.P 1.0 )

2) Basic Profile 1.1 (B.P 1.1)

-->Basic Profile is a specification document

The B.P documents contains guidelines for building webservices in any programming lang, those are not specific to java.

### **Types of WebServices**

1) Big Web Services: This types of web services uses XML based Protocols for exposing the services. Includes (SOAP and WSDL)

## 2) Restful Web Services:-

This type of web services exposes its services through a Http REST (Representational State Transfer) Architectural Style.

There So Many API's ,implementations for the

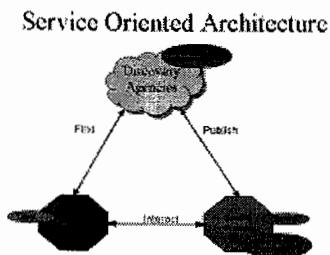
WS-I given standards

-->sun has given "JAX-RPC" API and "JAX-WS" API to develop Bigwebservices (OR) SOAP based webservices.

→sun has given "JAX-RS" API to develop Restfull webservices .

### **SOAP based webservices :**

**SOA:** A **service-oriented architecture (SOA)** is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology.



SOA provides loose coupling between software components so that they can be reused. Applications in SOA are built based on services. A service is an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

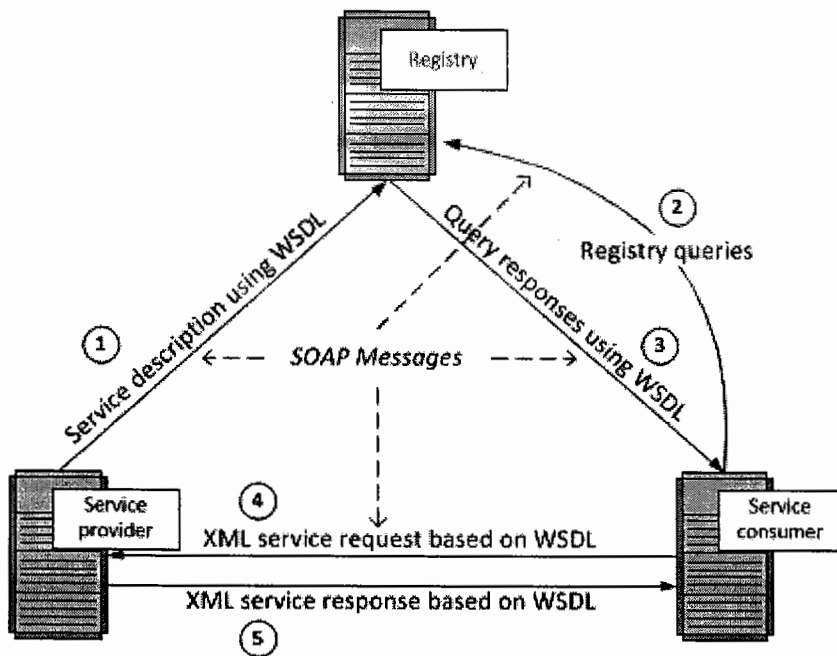
**service:** Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)

### **SOAP Based Web Service Components:-**

There are three major web service components.

- 1) SOAP
- 2)WSDL
- 3)UDDI

### SOAP webservices Arch:



### SOAP :-

Simple Object Access Protocol (SOAP) .SOAP is a XML based Data format used for Exchanging the data Over n/w.

SOAP is not really a protocol. It is a message Format.

SOAP messages are transferable across firewalls in a n/w. Because SOAP message are transferred internally as a part of http request and http response . Http is a firewall friendly protocol. SO SOAP messages are also friendly.

SOAP request and SOAP response message are transfer as a body part of http request and response.

In the application the SOAP messages are not created manually by application developers .

The mediator objects of the communication i,e stub and skelton objects will prepare SOAP request and response message in the middle.

SOAP messages are language independent , platform independent .Because it is xml format and xml is universal data transfer format.

SOAP is compatible with all transport protocols.

SOAP is having two versions 1.1 and 1.2.

### Advantages of Soap Web Services

->**WS Security:** SOAP defines its own security known as WS Security.

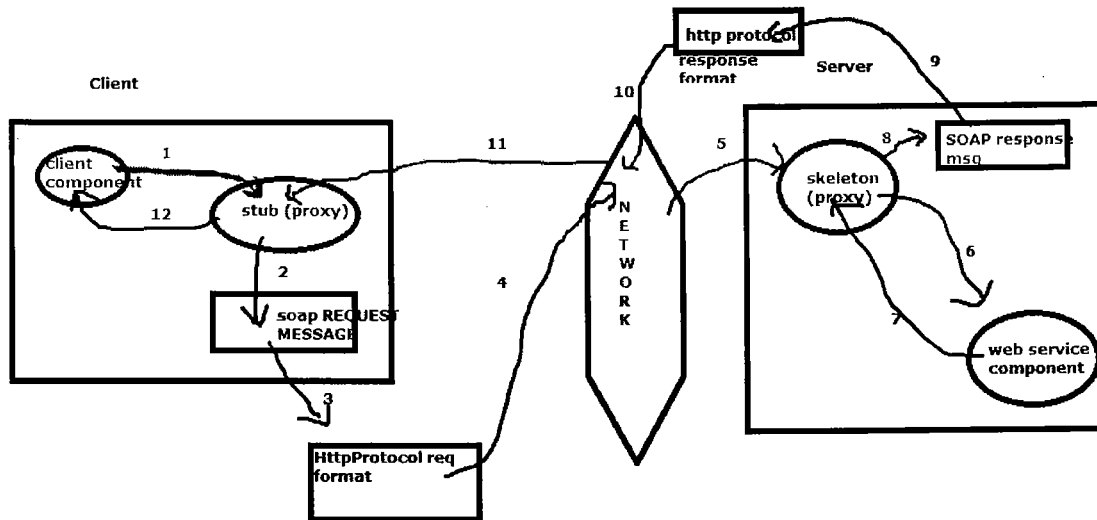
->**Language and Platform independent:** SOAP web services can be written in any programming language and executed in any platform.

### Disadvantages of Soap Web Services

- 1) **Slow:** SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
- 2) **WSDL dependent:** SOAP uses WSDL and doesn't have any other mechanism to discover the service.

### SOAP communication

*Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: [www.nareshit.com](http://www.nareshit.com)*



SOAP messages are transferable across firewalls in a n/w. Because SOAP message are transferred internally as a part of http request and http response . Http is a firewall friendly protocol. SO SOAP messages are also friendly.

### Note

Whatever the services are providing by the Provider should be explained in a document called as WSDL. Consumer cannot look into code of "provider" So consumer needs the information like what

are the services are providing, what they will take as an input, and what the

Output they will be return, what is an URL of webservice .etc.

### WSDL :-

Web service Description lang. "WSDL" is an xml document, because it should be Lang independent ,so that any type of client can understand the services of the provider.

WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is acting as a contract between client and service provider.

WSDL is pronounced as "wiz-dull".

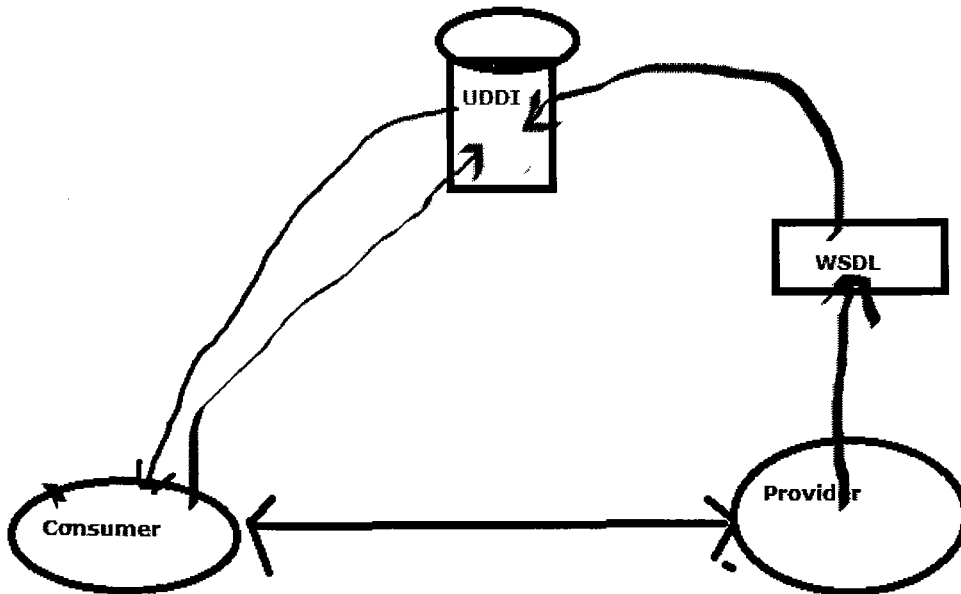
WSDL is having two versions 1.1 and 2.0.

### **How can consumer get the WSDL document,(OR) from where "Consumer get?**

WSDL document has to be placed in some location from where "Consumer" can access them,that location is nothing but "UDDI".

### UDDI

UDDI is an acronym for Universal Description Discovery and Integration. UDDI is the registry where all the WSDL documents are registered. UDDI should be interoperable, so it is also developed in XML. UDDI registry also called as XML registry.



### What is End point ?

A Server side component which receives the request from Consumer. In General we will use either Servlet (OR) EJB as an EndPoint

### Message Exchanging patterns(MEP)

In software architecture, a **messaging pattern** is a network-oriented architectural pattern which describes how two different parts of a message passing system connect and communicate with each other.

**Message exchange pattern is a template that establishes a pattern for the exchange of messages between two communicating parties.**

A Consumer can communicate with provider in Three Ways

- 1) Synchronous Request-Reply
- 2) Asynchronous Request-Reply
- 3) Fire and forget

**1) Synchronous Request -Reply :-** In this Way of Communication Consumer sends the request to Provider and Consumer waits until response comes from the Provider.

**2) Asynchronous Request -Reply: -** In this Consumer sends the **request** to Provider and Consumer will not wait until it gets the response. After request sent to the Provider it continues its flow of execution. Consumer will have Response Listener which listens the response from **the** Provider.

### **3) Fire and forget**

It is similar to Asynchronous request- reply. But in this there is no response listener.

In this way of communication once the request is sent to the provider the consumer continues its process, does not bother about response.

### **MEF (Message Exchanging Formats)**

The Message Exchanging Formats are Communication Models.

There are two communication *style* models that are used to translate a WSDL binding to a SOAP message body. They are:

- *Document, and*
- *RPC*

The advantage of using a *Document* style model is that you can structure the SOAP body any way you want it as long as the content of the SOAP message body is any arbitrary XML instance. The *Document* style is also referred to as *Message-Oriented style*.

However, with an *RPC* style model, the structure of the SOAP request body must contain both the operation name and the set of method parameters. The *RPC* style model assumes a specific structure to the XML instance contained in the message body.

Furthermore, there are two encoding *use* models that are used to translate a WSDL binding to a SOAP message. They are:

- *literal, and*
- *encoded*

When using a *literal* use model, the body contents should conform to a user-defined XML-schema(XSD) structure. The advantage is two-fold. For one, you can validate the message body with the user-defined XML-schema, moreover, you can also transform the message using a transformation language like XSLT.

With a (SOAP) *encoded* use model, the message has to use XSD datatypes, but the structure of the message need not conform to any user-defined XML schema. This makes it difficult to validate the message body or use XSLT based transformations on the message body.

The combination of the different *style* and *use* models give us four different ways to translate a WSDL binding to a SOAP message.

1. *Document/literal*
2. *Document/encoded*
3. *RPC/literal*
4. *RPC/encoded*

The final recommendation is to use *document style* with *literal use* for Web Services Interoperability (WS-I) compliance.

### **Approaches to Develop SOAP Web Services :-**

Web Services development can be done in two ways

- 1) Contract First (top-down) approach
- 2) Contract Last (bottom-up) approach

Always from a Service Oriented Architectural point of view, the Contract b/w the Consumer and provider is WSDL. But From a Java Point of View, the Contract b/w Consumer and The Provider is an Interface.

### **Contract First Approach :-**

In The Contract First Approach Always the Contract (WSDL ) would be created first and at end the Services would be generated because of this reason this approach is called as ContractFirstApproach.

WSDL-->Services

### **Contract Last Approach :-**

In The Contract Last Approach Always the Services would be Developed first and at end the Contract would be generated (WSDL doc) because of this reason this approach is called as ContractLastApproach.

Services → WSDL

### **Note :-**

The Services can be developed with any tech like .net,java,php etc..

### **JAX-RPC( Java API For XML Remote Procedure Call)**

The JAX-RPC is a high level API given by sun Microsystems for creating and accessing Big Web Services ( SOAP based) in java.

The Jax-Rpc API versions are 1.0 , 1.1 and 2.0. The JaxRpc 2.0 version renamed as JAX-WS 2.0.

### **Implementations of JAXRPC API**

1. JAX-RPC-SI (Standard Implementation) (from sun)
2. Apache Axis1 (from Apache Software Foundation)
3. Web logic web services (from oracle)
4. Web Sphere web services (From IBM)

->JAXRPC follows Basic Profile 1.0 document guidelines

->supporting only synchronous way of Communication

->The default MEF for JAXRPC applications is Rpc/encoded

-->Compatible JDK version :JDK 1.4+

--> Compatible JEE version : Jee4+

-->It supports SOAP 1.1 version (SOAP is having two versions 1.1 and 1.2)

-->It supports WSDL 1.1 version (WSDL is having two versions 1.1 and 2.0)

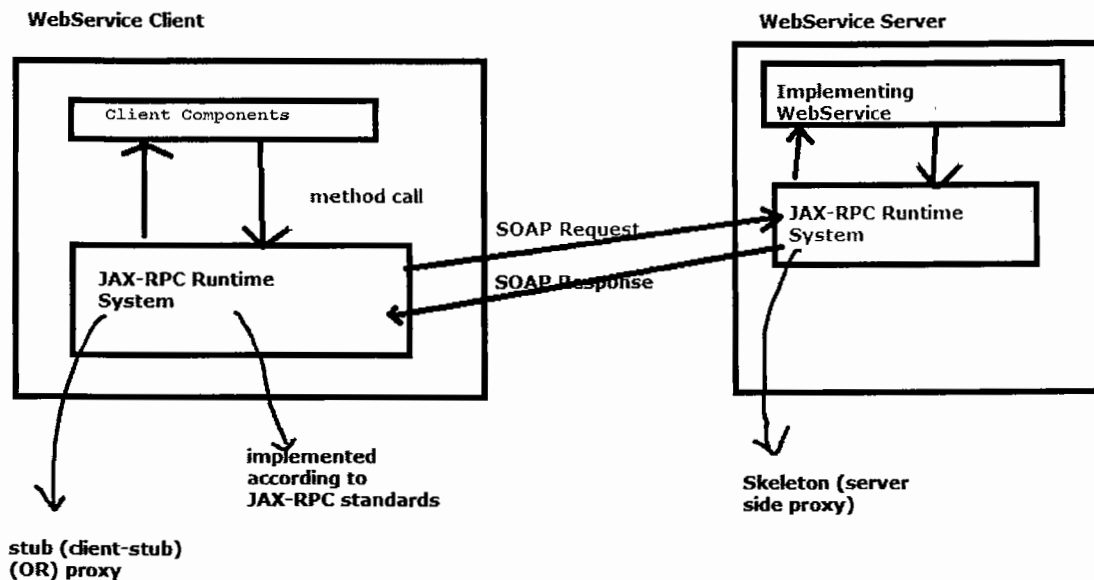
-->it is supports Any binding API for Marshling and UnMarshalling.-

> It Supports HTTP 1.1 as the transport for SOAP messages. HTTP binding for the SOAP messages is based on the SOAP 1.1 specification.

Note that the required support of HTTP 1.1 must not mean that the HTTP transport is the only transport that can be supported by a JAX-RPC runtime system implementation. JAX-RPC core APIs are designed to be transport-neutral. The JAX-RPC API Is compatible with any transport protocol that supports ability to deliver SOAP messages.

### **JAX-RPC Architecture**

The following Figure shows the architecture of the JAX-RPC implementation. The service side contains a JAX-RPC service runtime environment and a service . The client side contains a JAX-RPC client runtime environment and a client application. The JAX-RPC client and service runtime systems are responsible for sending and processing the remote method call and response, respectively. The JAX-RPC client creates a SOAP message to invoke the remote method and the JAX-RPC service runtime transforms the SOAP message to a Java method .



Note: It is not necessary to use JAXRPC both the sides.

#### JAX-RPC RS at client side

- > **Marshall the request data (java Obj) to SOAP Message**  
**Make a SOAP call to the server**
- > **Unmarshall the response SOAP message to Java Objects**

#### JAX-RPC RS in server side

- > **expose the Java Component methods as webservice methods**
- > **Unmarshall the SOAP request message to objects**
- > **Accessing the service for the request and Marshall the response to SOAP Message.**

#### JAXRPC-SI implementation

This implementation is given by sun to develop web services.

By using this implementations to develop webservices we required two tools



1) wscompile 2) wsdeploy

The above two tools are not coming along with JDK. To use the above two tools we can install JWSDP (Java Webservices Developer Pack) s/w.

#### **JWSDP S/w Download Link :**

<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jwspd-419428.html#jwspd-2.0-oth-JPR>

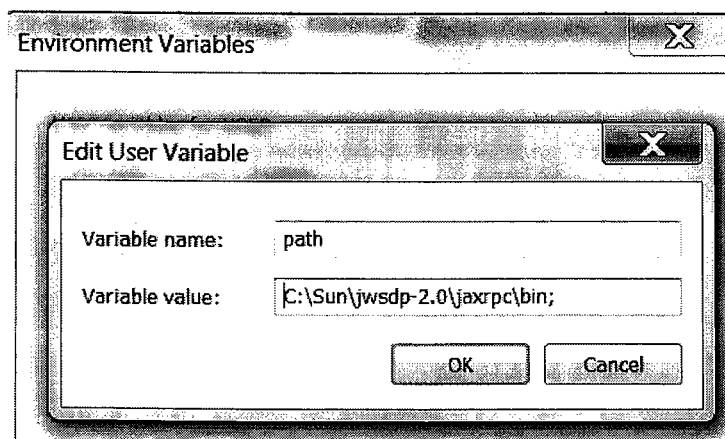


Java Web Services Developer Pack 2.0		
You must accept the Oracle Binary Code License Agreement for Java EE Technologies to download this software.		
Thank you for accepting the Oracle Binary Code License Agreement for Java EE Technologies; you may now download this software.		
Product / File Description	File Size	Download
Java Web Services Developer Pack 2.0	25.16 MB	 jwsdp-2_0-unix.sh
Java Web Services Developer Pack 2.0	22.69 MB	 jwsdp-2_0-windows-i586.exe

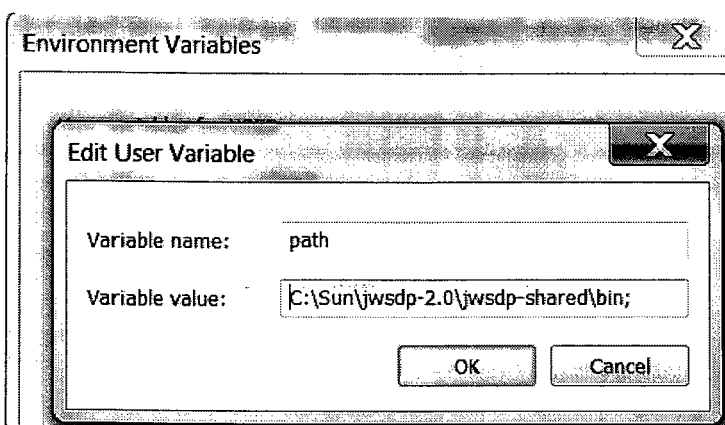
After

**Installing JWSDP S/w we can set the path Environment variables as follows**

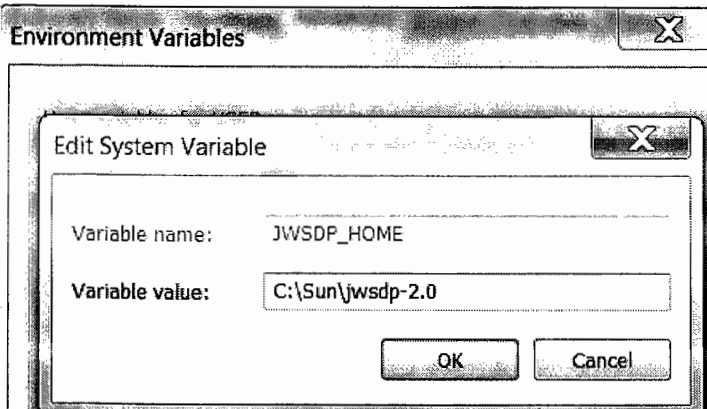
**Step1:** Edit the path Environment variables then set jwsdp-2.0/jax-rpc path environment variables



**Step2:** Edit the path Environment variables then set jwsdp-shared path environment variables



**Step3:-** set JWSDP\_HOME Environment variables



Note :- Along with the above Environment variables we can also set java path and JAVA\_HOME Environment variables .

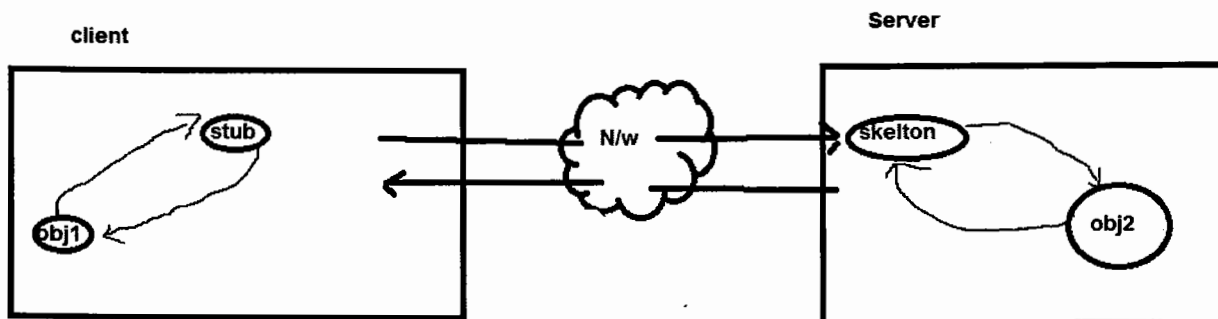
**How Object at Different JVMs will communicate ?**

If 2 java Objects are Running in a Single JVM then one Object automatically knows another. So both object's are communicate directly.

If 2 java Objects are Running in Different JVM's one object doesn't know about another. So direct communication is Not possible.

To communicate The Object Running at different JVM's in the Middle some helper Objects are required .we call these helper objects are proxy objects

The helper object resided at client JVM is called Stub and The helper Object resided at server JVM is called Skeleton.



**what is proxy ?**

A proxy is not a real object but acts as a real object .

Ex:- An ATM is not a Bank but acts as a Bank so ATM is Proxy for a bank.

### Steps to develop SOAP based webservices with JAX-RPC –SI

To develop the webservice by using JAX-RPC-SI implementation we can use the following steps.

#### Approach : ContractLastApproach

#### API : JAX-RPC

#### Implementation : JAXRPC-SI

#### Endpoint : servlet

#### JDK : JDK6

#### MEF : rpc/encoded

### **Step1: create SEI interface**

SEI stands for Service Endpoint Interface. As we Know Always From An Architectural point of view , the Contract b/w the Consumer and provider is WSDL. But From a Java Point of View, the Contract b/w Consumer and The Provider is an Interface. In Contract Last Approach the Web Service Development always starts with SEI.

The SEI is acting as a contract b/w Consumer and Provider, because All the methods declared in the SEI are exposed on the N/w as a web service methods.

- Note : The methods of SEI is Always accessed remotely by clients.

#### **Rules :**

- i) Your SEI interface must extends java.rmi.Remote interface .

**If your interface extends Remote Interface then it's get the Remote Functionality.**

- ii) The SEI interface methods must throws java.rmi.RemoteException.

- (iii) The SEI interface method parameters and return types must be serializable.

Example:

```
package com.jaxrpcexample.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface IHelloService extends Remote{
public String sayHello(String name)throws RemoteException;
}
```

### **Step2:- create SEI Implemented Class**

Write a class which implements from SEI interface, your implementation class must provides implementation for all the methods which are declared in SEI interface. Implementation class could contain some other methods also, but those will not be exposed Over the N/w. Your web service methods Overridden from SEI interface may (OR) may not throws Remote Exception.

```
package com.jaxrpcexample.service;
import java.rmi.RemoteException;
public class HelloServiceImpl implements IHelloService {
    @Override
    public String sayHello(String name) throws RemoteException {
        String msg="Hello :"+name+" Welcome to JaxRpc-SI";
        return msg;
    }
}
```

Note :- Compile the SEI and implementation classes.

#### **Write a configuration file.**

To Generate the wsdl document we will use wscompile tool. The wscompile tool directly not takes SEI name as an input.

The wscompile tool always takes configuration file as an input.

In the configuration file we can specify the SEI and SEI implemented classes and pass as an input to wscompile tool.

#### **Example configuration file for IHelloService:**

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="Hello" targetNamespace="http://service.jaxrpcexample.com/wsdl"
typeNamespace="http://service.jaxrpcexample.com/types" packageName="com.jaxrpcexample.proxy">
    <interface name="com.jaxrpcexample.service.IHelloService"
servantName="com.jaxrpcexample.service.HelloServiceImpl"/>
  </service>
</configuration>
```

<service name="Hello"> -> service name is nothing but generated wsdl file name.

<interface name="SEI" servantName="SEI-Implemented class"/>

Writing of servantName is an optional.

By convention, the configuration file is named config.xml, but this is not mandatory to maintain the same name.

#### **Step4:-Generate the wsdl and artifacts**

To generate the WSDL and artifacts we can use wscompile tool. As we know wscompile tool is not coming along with JDK. The wscompile tool is providing By JWSDP software.

Syntax to use wscompile : wscompile [options] configuration\_file.

The wscompile tool generates stubs, ties, serializers, and WSDL files used in JAX-RPC clients and services. The tool reads a configuration file, which specifies either a WSDL file, a model file, or a compiled service endpoint interface.

The following table lists the wscompile options. Note that exactly one of the -import, -define, or -gen options must be specified.

#### **Options :**

Option	Description
-classpath <path>	specify where to find input class files
-cp <path>	same as -classpath <path>
-d <directory>	specify where to place generated output files
-define	read the service endpoint interface, define a service
-f:<features>	enable the given features (See the below below table for a list of features. When specifying multiple features, separate them with commas.)
-features:<features>	same as -f:<features>
-g	generate debugging info
-gen	same as -gen:client
-gen:both	generate both client and server artifacts
-gen:client	generate client artifacts (stubs, etc.)
-gen:server	generate server artifacts (ties, etc.) and the WSDL file (If you are using wsdeploy you do not specify this option.)
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defaults to 8080 on JWSDP)
-import	read a WSDL file, generate the service endpoint interface and a template of the class that implements the interface

-keep	keep generated files
-mapping <file>	generate a J2EE mapping.xml file (This option is not available in JWSDP.)
-model <file>	write the internal model to the given file
-nd <directory>	specify where to place non-class generated files
-O	optimize generated code
-s <directory>	specify where to place generated source files
-source <version>	Generate code for the specified JAX-RPC SI version. Supported versions are: 1.0.1, 1.0.3, and 1.1 (default).
-verbose	output messages about what the compiler is doing
-version	print version information

Example:

```
wscompile -gen:server -d src -cp build\classes -keep -verbose -model model-rpc-enc.xml.gz WebContent\WEB-INF\config.xml
```

The model file contains all the artifacts generated by the wscompile tools if the -model switch is provided on the wscompile command line.

**Note :** Drag and drop model file and wsdl file into WEB-INF.

**Step5:-** write web services deployment descriptor file (jaxrpc-ri.xml)

After generating the necessary binding classes to expose our class as service, we need to

Configure the endpoint information of our service in a configuration file called as jaxrpc-ri.xml

**Note :-**

The file name should be jaxrpc-ri.xml and it is mandatory to place under WEB-INF directory.

#### **jaxrpc-ri.xml**

```
<webServices
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
  version="1.0"
  targetNamespaceBase="http://service.jaxrpcexample.com/wsdl"
  typeNamespaceBase="http://service.jaxrpcexample.com/types"
  urlPatternBase="/ws">
  <endpoint
    name="Hello"
    displayName="HelloWorld Service"
    description="A simple web service"
    wsdl="/WEB-INF/Hello.wsdl"
    interface="com.jaxrpcexample.service.IHelloService"
    implementation="com.jaxrpcexample.service.HelloServiceImpl"
    model="/WEB-INF/model-rpc-enc.xml.gz"/>
  <endpointMapping
    endpointName="Hello"
    urlPattern="/hello"/>
</webServices>
```

The file contains the following webServices attributes:

- The webServices element includes name, typeNamespace, and targetNamespace attributes.
  - The name attribute is used to generate the WSDL file for publication in a public registry.
  - The typeNamespace attribute defines the namespace in WSDL document for types generated by the wscompile tool.
  - The targetNamespace attribute is used for qualifying everything else in the WSDL document.

#### **Step 6:- create project as war file.**

To create the project as a war file we can use jar command as follows

**jar -cvf FileName.war \*.\***

cvf means create verbose file



current worling directory all the files

(OR)

In eclipse directly we can export the project as a war file.

#### **Step7: Run wsdeploy tool:-**

After creating the project as a war file ,we need to run wsdeploy tool which generates webservice deployment descriptor (in the new war file ) that carries the information describing the service, which is used for servicing the request from the consumer.

#### **SYNTAX :**

wsdeploy -o target.war MyApp.war

After generating the target.war Directly deploy the target.war into server

(OR)

Extract the target.war and copy jaxrpc-ri-runtime.xml file and web.xml files paste into Your Project WEB-INF folder. Then after deploy your project into the server.

#### **Note:**

**When we run wsdeploy tool in the web.xml file servlet and listener configurations are generating automatically.**

The Listener loads the web services deployment descriptor file and giving the the servlet.

The servlet will receive the request from client .

#### **To extract we can use jar command as follows**

Jar -xvf target.war

The classes

The tie classes which are used to communicate with clients.

**Note:- After deploying the application into the server to test the webservice we can write a client application (OR) we can use SOAP UI Tool.**

#### **Requet processing flow:-**

The consumer sends the Http request to the provider using the URL pattern specified in the `jax-rpc-ri.xml`

As we know we developed Servlet Endpoint based WebService on the Provider side , a servlet will receive the request to process it.If you look at the `web.xml` generated, it contains the JAXRPC Servlet configured to listen on the URL `"/hello"`. (this was configured when we run `wsdeploy` tool)





## SOAP UI Tool

SoapUI is the world leading Open Source Functional Testing Tool, mainly it is used for API Testing .SoapUI is a free and open source cross-platform Functional Testing solution

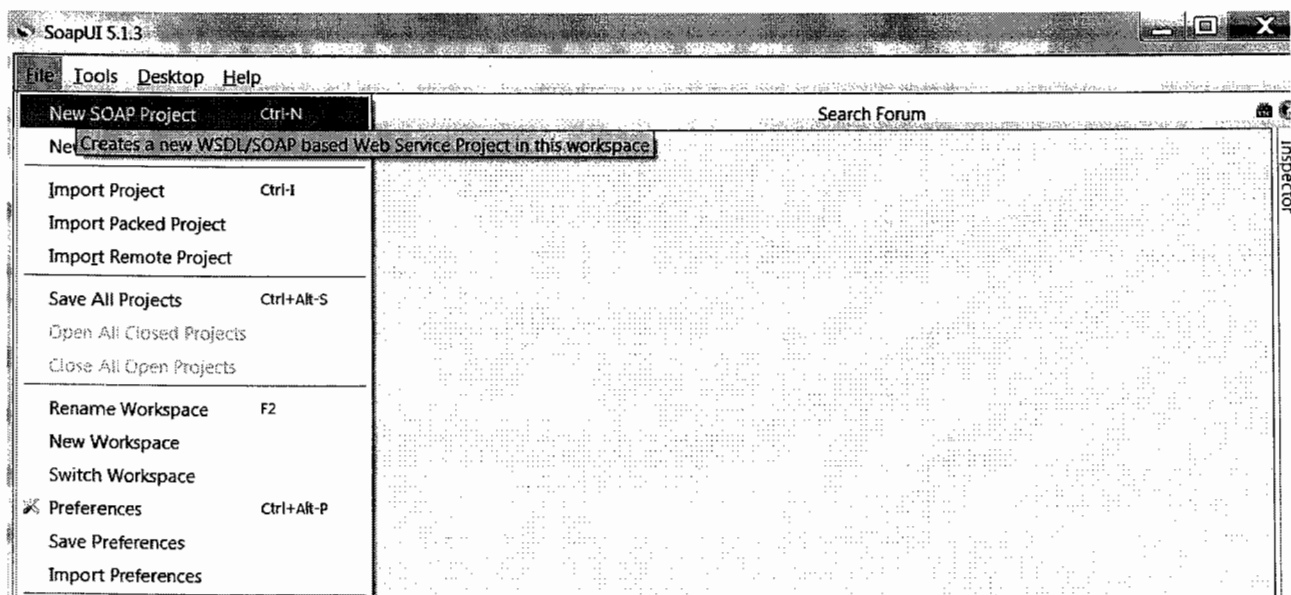
SoapUI enables you to create advanced Performance Tests very quickly and run Automated Functional Tests.

The Current version of SOAP UI tool is SoapUI 5.2.1 .

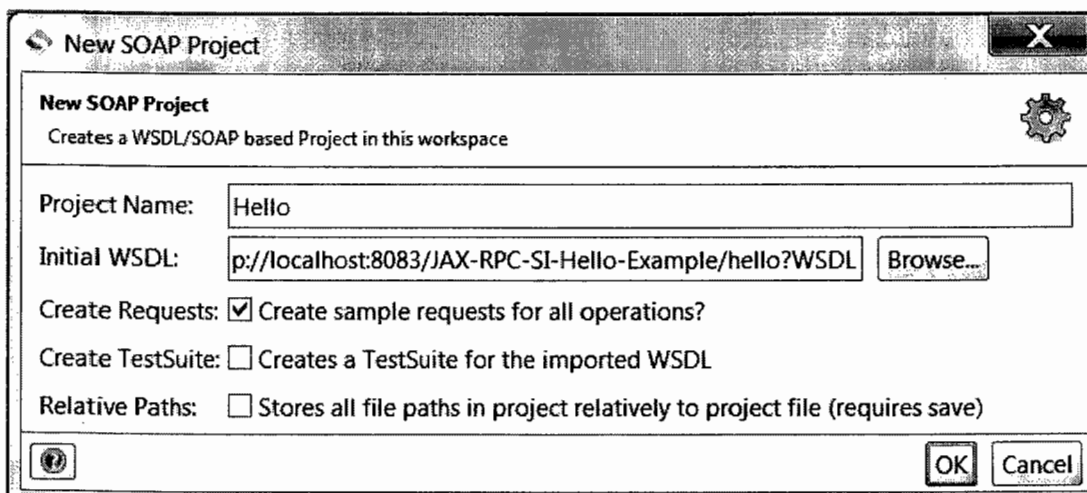
We can Download the The SOAP UI tool from the following link

<https://www.soapui.org/downloads/soapui.html>

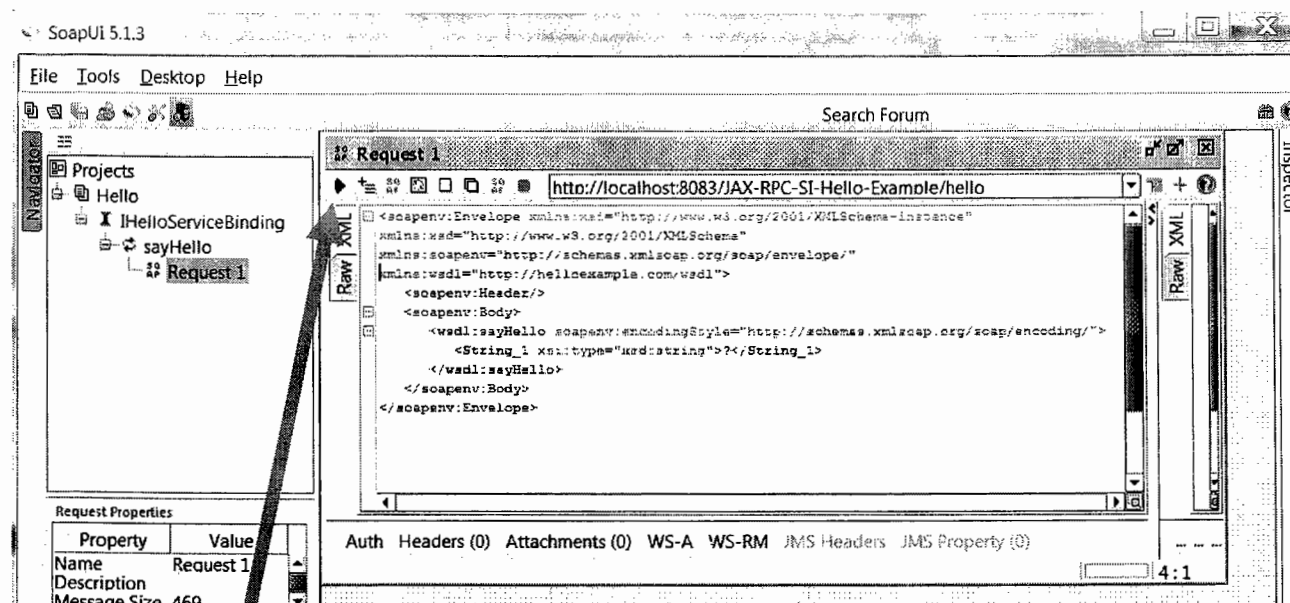
After installing SOAP UI tool create the SOAP Project to test the SOAP webservices.



Enter project Name and WSDL Location



Then click on OK button



Click on run button to test the service

## JAXRPC-Clients

By writing Client Application also we can test the webservice.

As we know JAXRPC API Facilitates in development of both Providers and Consumers as well.

JAXRPC API Supports Three Types of Clients

- 1)Generated Stub(Proxy) based Client
- 2)Dynamic Proxy
- 3)Dynamic Invocation Interface(DII)

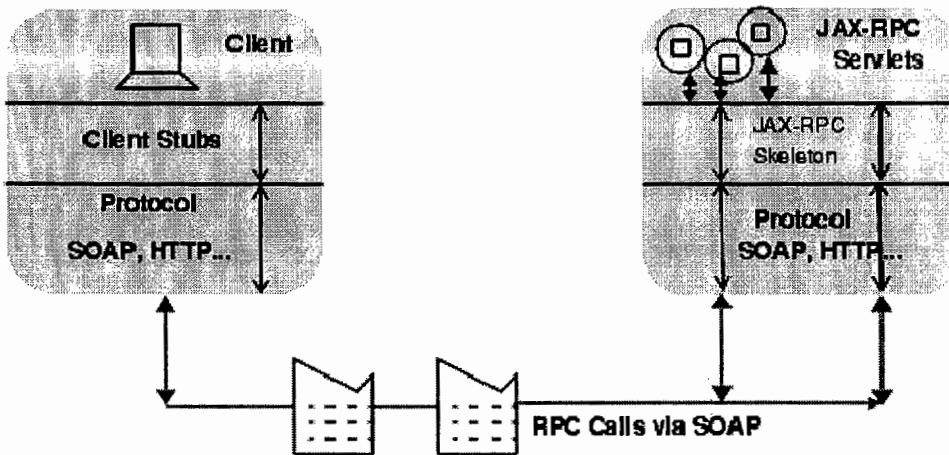
A stub class is a mapping of a port in the WSDL that describes the Web service. It must therefore implement the service definition interface that reflects the methods of the associated portType. Thus the client has strongly typed, early-bound access to the Web service endpoint.

The stub must also implement the javax.xml.rpc.Stub interface, which provides the facility for the client to configure the stub dynamically.

Typically, a JAX-RPC client performs the following steps. These steps are illustrated in the figure JAX-RPC Client Model.

1. The client calls the stub.
2. The stub redirects the call to the appropriate Web service.
3. The server catches the call and redirects it to a framework.
4. The framework wraps the actual implementation of the service, then calls the Web service on behalf of the client.

5. The framework returns the call to the server.
6. The Web service, in turn, returns the information to the originating client stub.
7. Finally, the client stub returns the information to the client application.



### 1)Generated Stub based Client

It is most widely used client, and it easy to build.

In this kind clients first we can generate all the required artifacts(stub etc..) at client side by using wscompile tool.

JAX-RPC-Client(GeneratedStubBasedClient)

```

src
├── com.nareshit.client
│   └── Test.java
├── com.nareshit.proxy
│   ├── Hello_Impl.java
│   ├── Hello_SerializerRegistry.java
│   ├── Hello.java
│   ├── IHelloService_sayHello_RequestStruct_SOAPBuilder.java
│   ├── IHelloService_sayHello_RequestStruct_SOAPSerializer.java
│   ├── IHelloService_sayHello_RequestStruct.java
│   ├── IHelloService_sayHello_ResponseStruct_SOAPBuilder.java
│   ├── IHelloService_sayHello_ResponseStruct_SOAPSerializer.java
│   ├── IHelloService_sayHello_ResponseStruct.java
│   ├── IHelloService_Stub.java
│   └── IHelloService.java
└── config.xml
JRE System Library [JavaSE-1.6]
  
```

As we know already, wscompile tool is designed to take input as configuration file and it cannot take directly the wsdl document. So, the location of the wsdl document has to be placed in config.xml and should be give it as input.

**config.xml**

```

<configuration
xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">

<wsdl location="http://localhost:8083/JAX-RPC-SI-Hello-Example/hello "
packageName="com.nareshit.proxy"></wsdl>

</configuration>

```

Now run the wscompile tool by giving the above config.xml as input.

**F:/Webseviceexamples/GeneratedStubBasedExample>wscompile -gen:client -d src -keep -verbose src\config.xml**

while running as we are developing the consumer ,we need to use -gen:client as we need consumer side binding classes .

In the Generated stub class it clearly indicates as implementing from SEI interface and it overrides the methods in the SEI interface, but those methods will not have business logic, rather they have the logic for converting object data into XML, and sends over the Http connection to the provider.

```

public class IHelloService_Stub extends com.sun.xml.rpc.client.StubBase
{
    implements com.nareshit.proxy.IHelloService {
        ---
        ---
        --- }

```

IHelloService_Stub Implements SEI interface
--

Now write the class in which create the Object of SEI , we know the Implementation of SEI interface at client side i,e IHelloService\_Stub .

Below is the code to Consume the Service.

**Test.java**

```

package com.nareshit.client;

import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.nareshit.proxy.Hello;

import com.nareshit.proxy.Hello_Impl;

import com.nareshit.proxy.IHelloService;

public class Test {

```

```
public static void main(String[] args)throws ServiceException,RemoteException {  
  
    Hello hello=new Hello_Impl();  
  
    IHelloService sei=hello.getIHelloServicePort();  
  
    String name=sei.sayHello("sathish");  
  
    System.out.println(name);  
  
}  
  
}
```

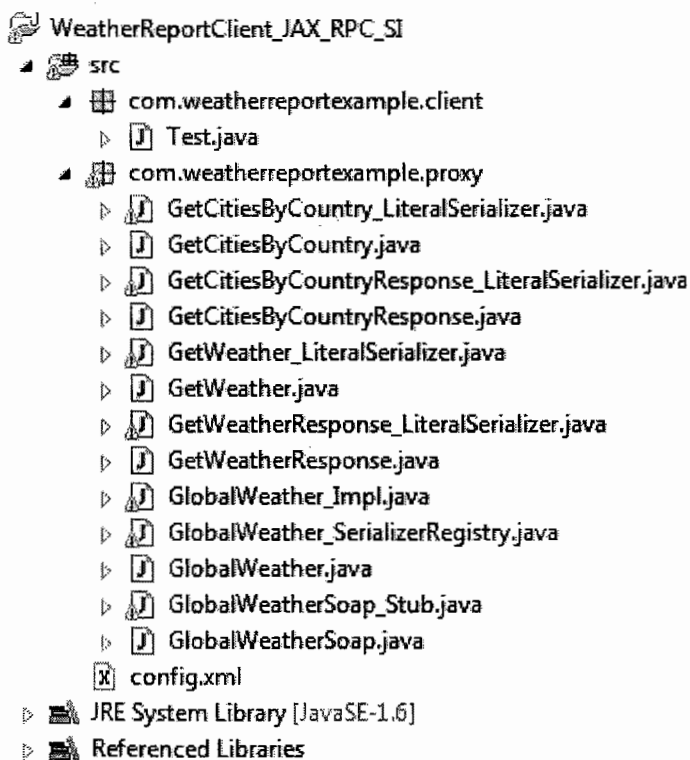
## Create a Java Client Application to Consume Weather Report Service?

**WSDL Location :** <http://www.webservicex.com/globalweather.asmx?WSDL>

**Implementation : JAXRPC-SI**

**Type : Generated Stub Based Client**

**JDK : JDK6**



As we know to create Generated Stub based Client with JAXRPC-SI Implementation it required to Work with wscompile tool.

wscompile tool will takes Configuration file as an input

So First Write config.xml file with WSDL location as follows

```
1 configuration
2 xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config"
3 <wsdl location="http://www.webservicex.com/globalweather.asmx?WSDL"
4 packageName="com.weatherreportexample.proxy"/>
5 </configuration>
```

Use wscompile tool and generate the Client Side Artifacts(like stub etc...)

**wscompile -gen:client -d src -keep -verbose src\config.xml**

Then After Generating the required Artefacts write the Following Code to Consume the Service

```
1 package com.weatherreportexample.client;
2
3 import java.rmi.RemoteException;
4
5 public class Test {
6     public static void main(String[] args) throws ServiceException, RemoteException {
7         GlobalWeather globalWeather=new GlobalWeather_Impl();
8         GlobalWeatherSoap sei=globalWeather.getGlobalWeatherSoap();
9         String result=sei.getWeather("HYDERABAD","INDIA");
10        System.out.println(result);
11    }
12 }
13
14 }
```

## JAX-RPC API(Apache Axis)

Apache Axis Stands for "Apache Extensible Interaction System" . Axis is an implementation of SOAP Protocol.

It is one of the open source implementation of JAXRPC API.

- Apache Latest Stable release Axis 1.4 final
- Axis is written completely in java
- It has the tools which can generate java classes from wsdl and wsdl from java classes.
- It has a tool for monitoring TCP/IP packets.

In order work on Apache Axis,First the developer has to download the Apache distribution from [http://www.apache.org/dyn/closer.cgi/ws/axis/1\\_4](http://www.apache.org/dyn/closer.cgi/ws/axis/1_4).

Note:- With Eclipse by –default Apache Axis implementation is coming. So directly we can also work with Eclipse to create Apache Axis Project.

### Steps to Work Apache Axis in Eclipse :

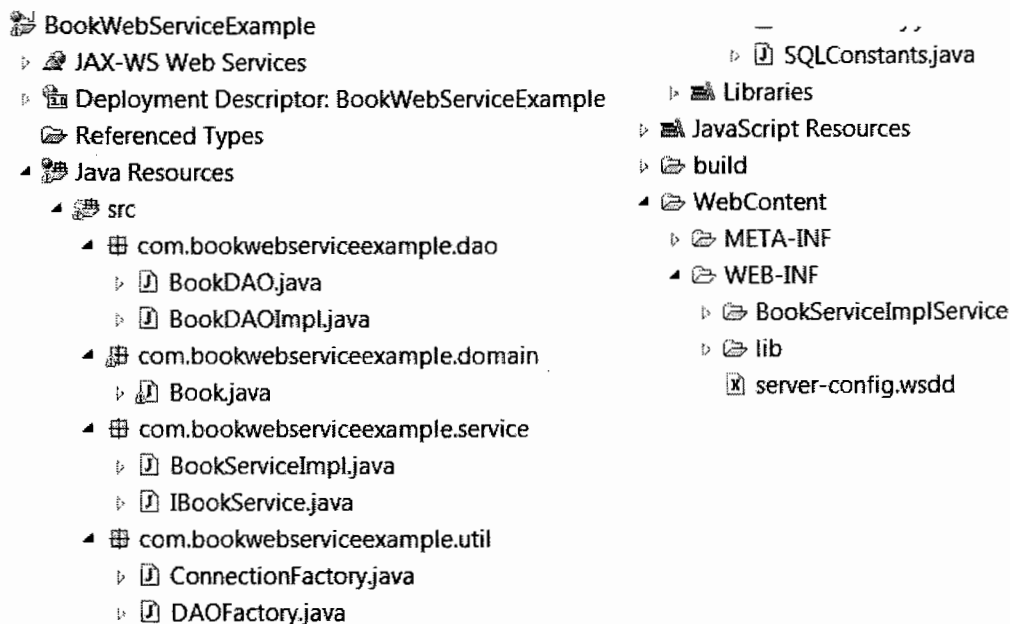
Contract Last Approach

Apache Axis1

JDK7

Servlet Endpoint

### Step1:- create Dynamic Web project in Eclipse



### Step2:- create SEI interface As follows

```

IBookService.java
package com.bookwebserviceexample.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
import com.bookwebserviceexample.domain.Book;
public interface IBookService extends Remote{
public Book searchBook(String isbn)throws RemoteException;
}
    
```

### Step3:- create SEI implementation classes as follows

```

1 BookServiceImpl.java
package com.bookwebserviceexample.service;
import java.rmi.RemoteException;
import com.bookwebserviceexample.domain.Book;
import com.bookwebserviceexample.util.DAOFactory;
public class BookServiceImpl implements IBookService{
    public Book searchBook(String isbn) throws RemoteException {
        Book book=DAOFactory.getBookDAO().searchBook(isbn);
        return book;
    }
}

```

Step4:-What Ever other classes we required create

```

1 Book.java
package com.bookwebserviceexample.domain;
import java.io.Serializable;
public class Book implements Serializable{
    private String isbn;
    private String author,title;
    private Double price;
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Double getPrice() {
        return price;
    }
    public void setPrice(Double price) {
        this.price = price;
    }
}

```

```

1 BookDAO.java
package com.bookwebserviceexample.dao;
import com.bookwebserviceexample.domain.Book;
public interface BookDAO {
    public Book searchBook(String isbn);
}

```



```

1 BookDAOImpl.java
package com.bookwebserviceexample.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.bookwebserviceexample.domain.Book;
import com.bookwebserviceexample.util.ConnectionFactory;
import com.bookwebserviceexample.util.SQLConstants;
public class BookDAOImpl implements BookDAO{
    public Book searchBook(String isbn) {
        Book book=null;
        Connection con=null;
        try{
            con=ConnectionFactory.getConnection();
            if(con!=null){
                PreparedStatement pst=
                con.prepareStatement(SQLConstants.SQL_SEARCH_BOOK);
                pst.setString(1,isbn);
                ResultSet rs=pst.executeQuery();
                if(rs.next()){
                    book=new Book();
                    book.setIsbn(isbn);
                    book.setTitle(rs.getString("title"));
                    book.setAuthor(rs.getString("author"));
                    book.setPrice(rs.getDouble("price"));
                }
            }
        }catch(SQLException se){
            System.out.println("Exeption Occured while Searching the Book :"+se.getMessage());
        }
        finally{
            if(con!=null){
                try{
                    con.close();
                }catch(SQLException se){
                    System.out.println("Exeption Occured while Closing the connection :"+se.getMessage());
                }
            }
        }
        return book;
    }
}
}

```

```

1 ConnectionFactory.java
package com.bookwebserviceexample.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class ConnectionFactory {
    static{
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException e) {
            System.out.println("Exception Occured while loading the driver class :"+e.getMessage());
        }
    }
    public static Connection getConnection() throws SQLException{
        String url="jdbc:oracle:thin:@localhost:1521:XE";
        String un="system";
        String pass="manager";
        Connection con=DriverManager.getConnection(url,un,pass);
        return con;
    }
}

```

```

DAOFactory.java
package com.bookwebserviceexample.util;
import com.bookwebserviceexample.dao.BookDAO;
import com.bookwebserviceexample.dao.BookDAOImpl;
public class DAOFactory {
    private static BookDAO bookDAO;
    static{
        bookDAO=new BookDAOImpl();
    }
    public static BookDAO getBookDAO(){
        return bookDAO;
    }
}

```

```

SQLConstants.java
package com.bookwebserviceexample.util;

public class SQLConstants {
    public static final String SQL_SEARCH_BOOK="select title,author,price from Books_Details where isbn like ?";
}

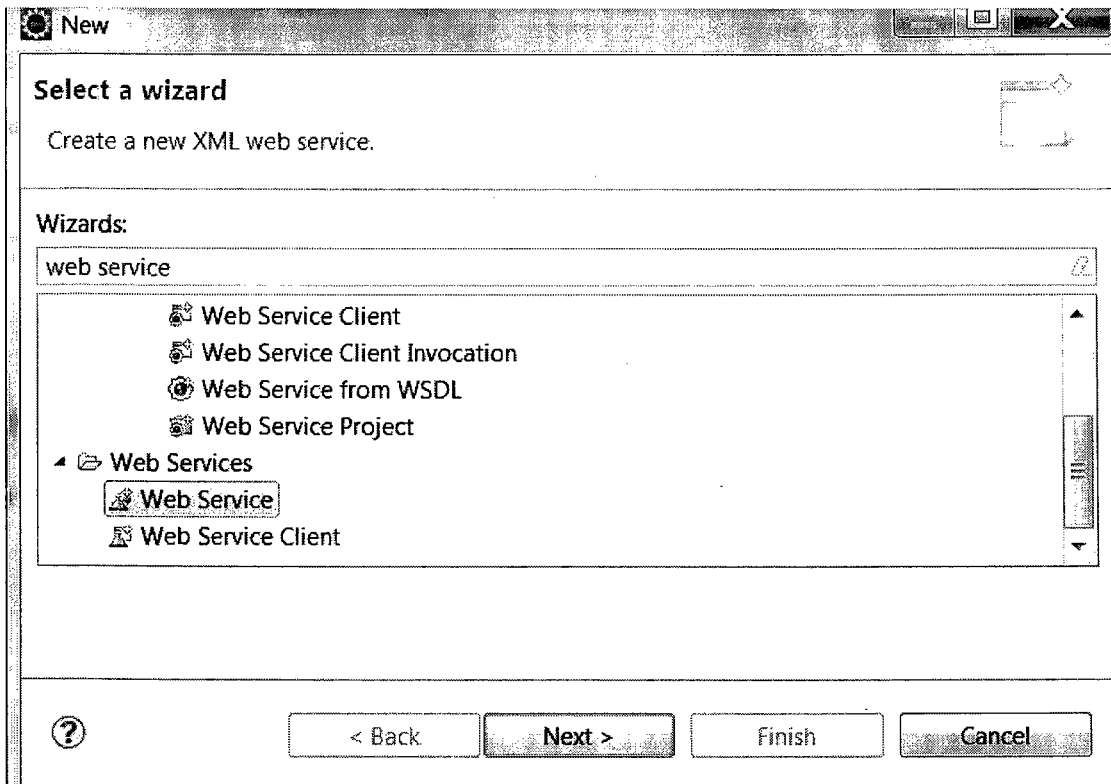
```

```

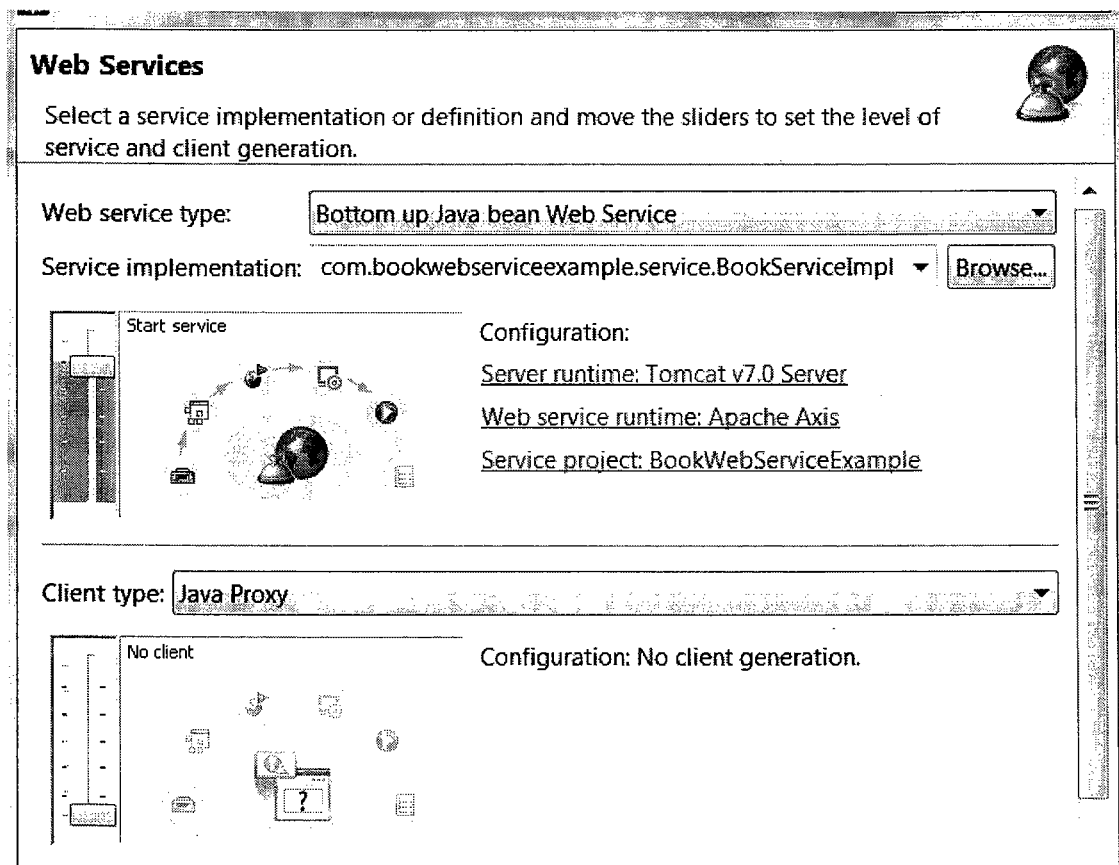
web.xml
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <servlet>
        <servlet-name>AxisServlet</servlet-name>
        <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/servlet/AxisServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>*.jws</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>AdminServlet</servlet-name>
        <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
        <load-on-startup>100</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>AdminServlet</servlet-name>
        <url-pattern>/servlet/AdminServlet</url-pattern>
    </servlet-mapping>
</web-app>

```

Step5:- After creating the required classes just right click on the project →select new →searche for Web services



The Click on Next Button → Then Select Required Approach like Bottom up Approach and SEI implementation class



The click on Click on Next button→Finish button.

The WSDL and web.xml files are generating By Eclipse.

**In the Same way we can create Generated Stub based client to Consume web Service.**

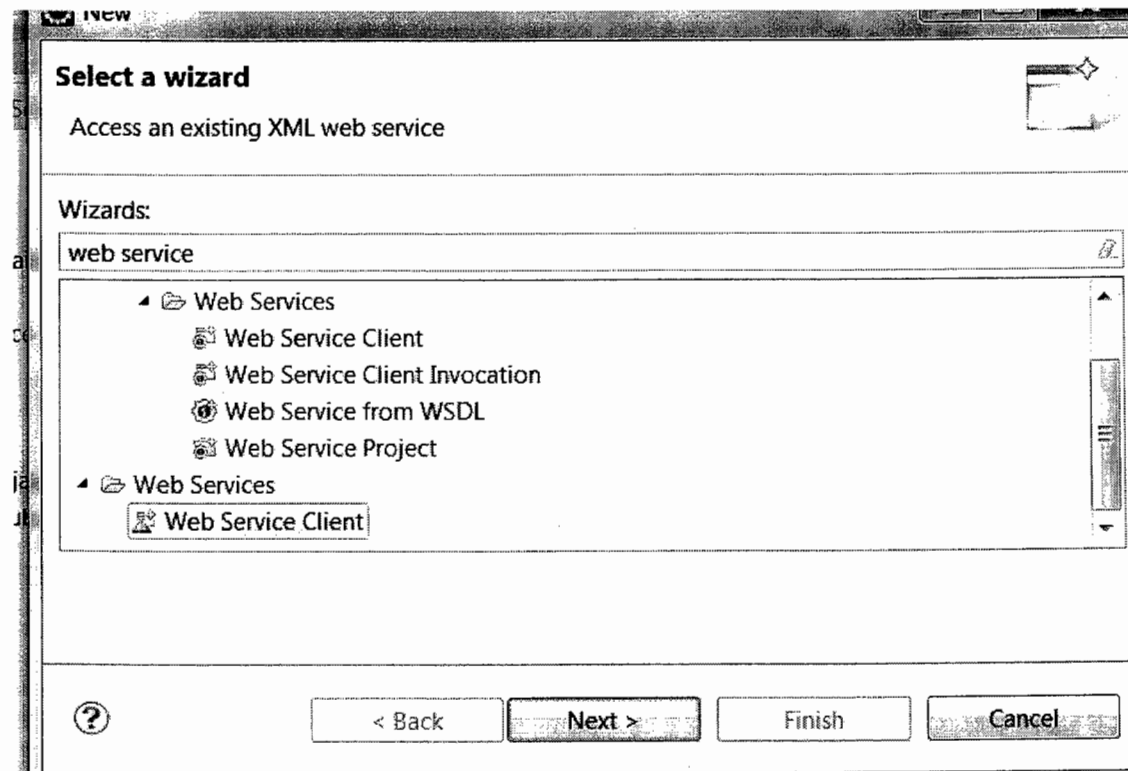
Client Side also it is not mandatory to use Axis Implementation.

The Following Example Shows How to create Client Application by Axis Implementation in Eclipse IDE

Steps:-

Step1 : create Java Project

Step2:- right on the Project →select New →Search for web service Client

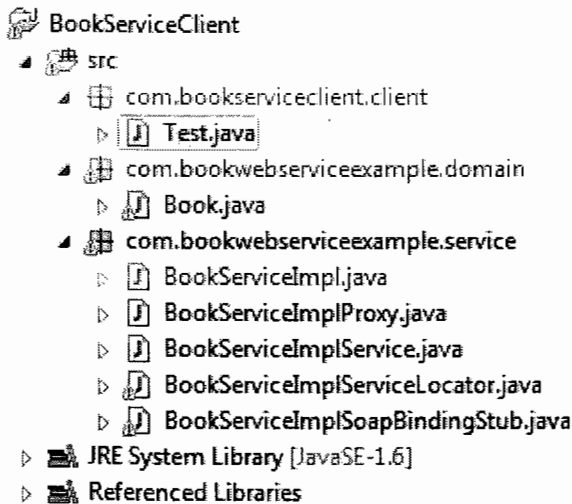


The Click on Next Button

Step3:-

Enter WSDL Location →Then Click on Next→Finish buttons

Then Client Side required Artifacts are generating



The Following Code Showing How to Consume the Service

```

1 Test.java
2
3 package com.bookserviceclient.client;
4
5 import java.rmi.RemoteException;
6
7 public class Test {
8
9     public static void main(String[] args) throws ServiceException, RemoteException {
10         BookServiceImplServiceLocator serviceImplServiceLocator=new
11             BookServiceImplServiceLocator();
12         BookServiceImpl sei=serviceImplServiceLocator.getBookServiceImpl();
13         Book book=sei.searchBook("CJ101");
14         System.out.println(book.getIsbn()+" "+book.getTitle()+" "+book.getAuthor()+" "+book.getPrice());
15     }
16 }
17
18
19
20
21
22

```

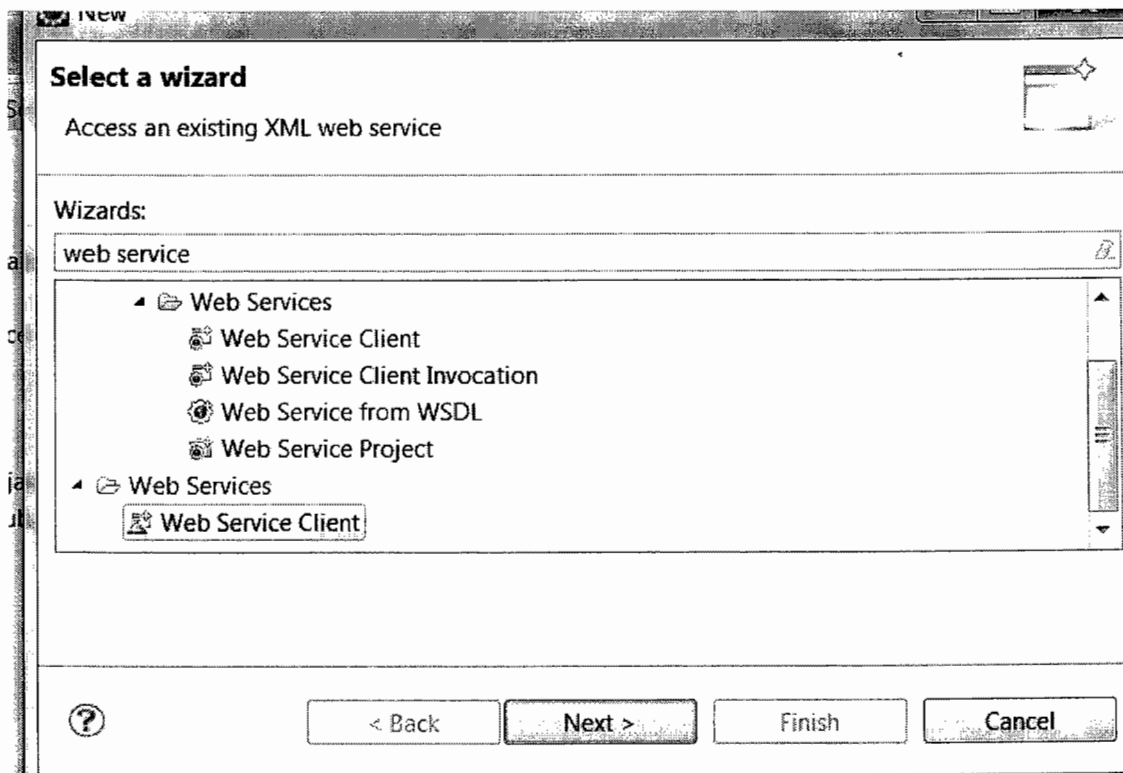
Create A Web Application as Client Application to Consume the Same Service

- BookWebServiceClient
  - src
    - com.bookwebserviceexample.domain
      - Book.java
    - com.bookwebserviceexample.service
      - BookServiceImpl.java
      - BookServiceImplProxy.java
      - BookServiceImplService.java
      - BookServiceImplServiceLocator.java
      - BookServiceImplSoapBindingStub.java
    - com.nareshit.client
      - BookServiceClient.java
      - BookServiceImplClient.java
      - BookServlet.java
  - JRE System Library [JavaSE-1.6]
  - Web App Libraries
    - build
  - WebContent
    - META-INF
    - pages
      - searchBook.jsp
      - searchBooksResults.jsp
    - WEB-INF
- lib
  - axis.jar
  - commons-discovery-0.2.jar
  - commons-logging.jar
  - jaxrpc.jar
  - jsp-api.jar
  - saaj.jar
  - servlet-api.jar
  - wsdl4j.jar
  - web.xml

Steps:-

Step1 : create Dynamic WebProject

Step2:- right on the Project →select New →Search for web service Client



The Click on Next Button

Step3:-

Enter WSDL Location → Then Click on Next → Finish buttons

Then Client Side required Artifacts are generating

Then After Write Code Access the service

The BookServiceImplClient class is having the code the access the Webservice.

```
BookServiceClient.java
1 package com.nareshit.client;
2 import com.bookwebserviceexample.domain.Book;
3 public interface BookServiceClient {
4     public Book getBook(String isbn);
5 }
6
```

```
BookServiceImplClient.java
1 package com.nareshit.client;
2
3 import java.rmi.RemoteException;
4 import javax.xml.rpc.ServiceException;
5 import com.bookwebserviceexample.domain.Book;
6 import com.bookwebserviceexample.service.BookServiceImpl;
7 import com.bookwebserviceexample.service.BookServiceImplServiceLocator;
8 public class BookServiceImplClient implements BookServiceClient {
9     public Book getBook(String isbn) {
10         Book book = null;
11         BookServiceImplServiceLocator locator = new BookServiceImplServiceLocator();
12         try {
13             BookServiceImpl sei = locator.getBookServiceImpl();
14             book = sei.searchBook(isbn);
15         } catch (ServiceException e) {
16             e.printStackTrace();
17         } catch (RemoteException e) {
18             e.printStackTrace();
19         }
20         return book;
21     }
22 }
23 }
24
```

```

1 package com.nareshit.client;
2
3 import java.io.IOException;
4 import javax.servlet.RequestDispatcher;
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import com.bookwebserviceexample.domain.Book;
10 import com.nareshit.client.BookServiceClient;
11 import com.nareshit.client.BookServiceImplClient;
12
13 public class BookServlet extends HttpServlet {
14     BookServiceClient bookServiceClient;
15
16     public void init() {
17         bookServiceClient = new BookServiceImplClient();
18     }
19
20     public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
21         String isbn = req.getParameter("isbn");
22         Book book = bookServiceClient.getBook(isbn);
23         String targetViewName = "/pages/searchBooksResults.jsp";
24         req.setAttribute("book", book);
25         RequestDispatcher rd = req.getRequestDispatcher(targetViewName);
26
27         rd.forward(req, res);
28     }
29 }
30

```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app>
3     <servlet>
4         <servlet-name>bookServlet</servlet-name>
5         <servlet-class>com.nareshit.client.BookServlet</servlet-class>
6         <load-on-startup>1</load-on-startup>
7     </servlet>
8     <servlet-mapping>
9         <servlet-name>bookServlet</servlet-name>
10        <url-pattern>/searchBooks</url-pattern>
11    </servlet-mapping>
12    <welcome-file-list>
13        <welcome-file>/pages/searchBook.jsp</welcome-file>
14    </welcome-file-list>
15 </web-app>

```

```

1 <html>
2 <head><h2>SearchBooks</h2></head><br/>
3 <form action="searchBooks" method="post">
4     Enter Book ISBN :<input type="text" name="isbn"/>
5     <input type="submit" value="search"/>
6 </form>
7 </html>

```



```
searchBooksResults.jsp
1 <%@page isELIgnored="false" %>
2 <%@include file="searchBook.jsp" %>
3 <table border="1">
4 <tr><th>ISBN</th><th>Title</th><th>Author</th>
5 <th>Price</th>
6 </tr><tr><td>
7 ${book.isbn}</td><td>
8 ${book.title}</td><td>
9 ${book.author}</td><td>
10 ${book.price}</td></tr>
11 </table>
```

## WSDL:-



## WSDL:

The Web Services are Providing by the Provider .To Know The services of “Provider” ,”Consumer” cannot look into the code of “Provider”.

So “consumer” needs the information like what are the services are providing ,what they will take as an input,and what the output they will return ,what is the URL to call a service etc.....

“Provider” will explain all these informations in one document called as “WSDL”.

“WSDL” stands for Web Services Description Language. It is used to describe the web services.

“WSDL” is an XML document,because it should be language independent ,so that any type of client can understand the services of the Provider.

From An Architectural Point of View ,WSDL is acts as a Contract Between Consumer and Provider.

The WSDL is having two versions 1.1 and 2.0.

WSDL contains total six elements. WSDL is also an XML ,Every XML will starts with PROLOG and Every Xml has a root Element .

WSDL is also starts with PROLOG and has root Element <definations>.

Including <definations> there are six elements.

The main structure of a WSDL document looks like this:

```
<definitions>
  <types>
    definition of types.....
  </types>

  <message>
    definition of a message....
  </message>

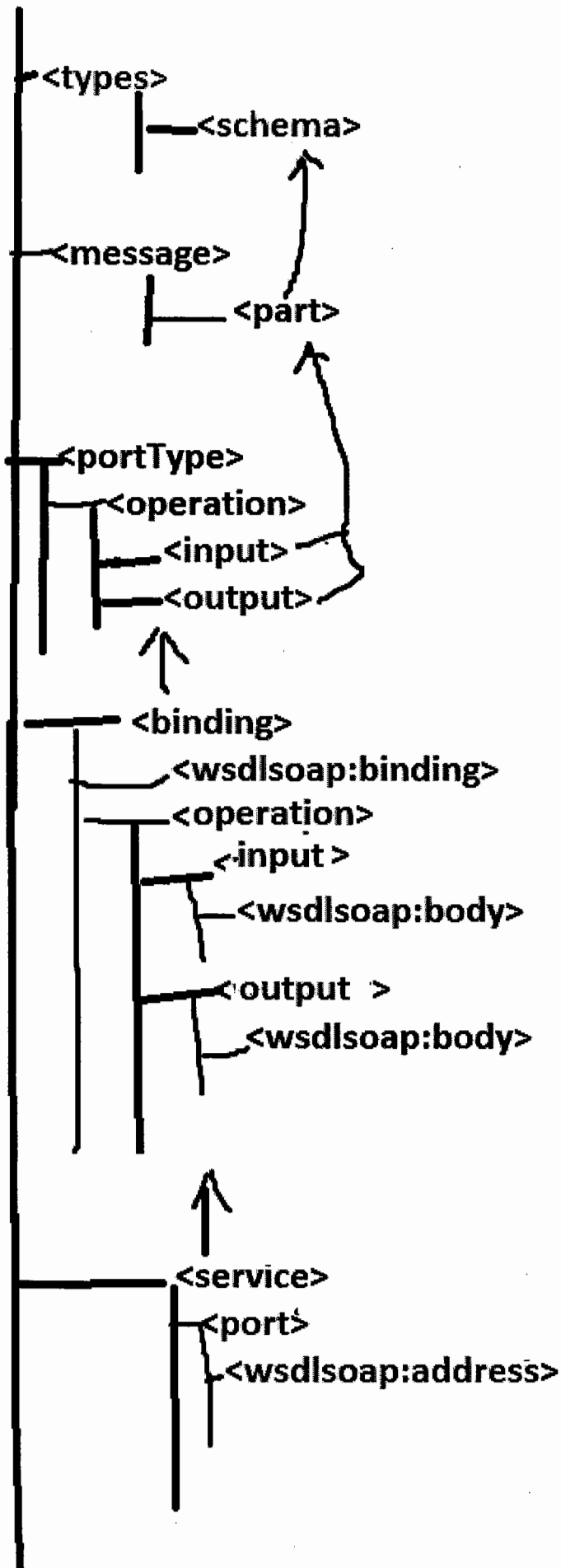
  <portType>
    <operation>
      definition of a operation.....
    </operation>
  </portType>

  <binding>
    definition of a binding....
  </binding>

  <service>
    definition of a service....
  </service>
```

</definitions>

The <**definitions**> element is the root element of WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements.

**<definitions>**

**<types>**

Types section defines the input/output types of a webservice method. If your webservice method takes any objects as an input (OR) returns any object as output, those relevant complex type

Representations are declared under <types> section of the wsdl document.

Let's take one IHelloService as SEI.

```
package com.nareshit.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface IHelloService extends Remote{
    public String sayHello(String name)throws RemoteException;
}
```

For the above SEI wsdl types are generating as follows

**WSDL document for document/literal**

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
xmlns:intf="http://service.nareshit.com"
```

```
xmlns:impl="http://service.nareshit.com"
```

```
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

```
targetNamespace="http://service.nareshit.com">
```

```
<wsdl:types>
```

```
<schema targetNamespace="http://service.nareshit.com"
```

```
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```

```
<element type="xsd:string" name="name"/>
```

```
<element type="xsd:string" name="sayHelloReturn"/>
```

```
</schema>
```

```
</wsdl:types>
```

```
<wsdl:message> -- </wsdl:message>
```

```
<wsdl:portType> ---- </wsdl:portType>
```

```
<wsdl:binding> -- </wsdl:binding>
```

```
<wsdl:service> --- </wsdl:service>
```

```
</wsdl:definitions>
```

*Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: [www.nareshit.com](http://www.nareshit.com)*

**<message>**

the next element after <types> is <message>. <message> represents the input and output of a web service method.

A web service method takes input as parameters and output as return values , to represents all of its inputs, we need one input<message> and to represents its output , we need one output <message>

For above SEI the wsdl message tags generating as follows.

```
<wsdl:definitions>
<wsdl:types> </wsdl:types>
<wsdl:message name="sayHelloResponse">
<wsdl:part name="sayHelloReturn" element="impl:sayHelloReturn"></wsdl:part>
</wsdl:message>
<wsdl:message name="sayHelloRequest">
<wsdl:part name="name" element="impl:name"></wsdl:part>
</wsdl:message>
<wsdl:portType> ---- </wsdl:portType>
<wsdl:binding> -- </wsdl:binding>
<wsdl:service> --- </wsdl:service>
</wsdl:definitions>
```

**<portType>**

PortType represents the SEI interface.

<portType> declares the operations of the service . Generally an SEI interface doesn't provider any implementation, rather it providers information like what are the methods and their parameters and return types.

Similarly <portType> describes about the service operations and their input/output types. It doesn't provide transport specific information about the service.

For the earlier show SEI interface, the portType representation as shown below.

```
<wsdl:definitions>
<wsdl:types> -- </wsdl:types>
<wsdl:message> --- </wsdl:message>
<wsdl:portType name="IHelloService">
  <wsdl:operation name="sayHello" parameterOrder="name">
```

```

    <wsdl:input message="impl:sayHelloRequest" name="sayHelloRequest">

</wsdl:input>

    <wsdl:output message="impl:sayHelloResponse" name="sayHelloResponse">

</wsdl:output>

</wsdl:operation>

</wsdl:portType>

<wsdl:binding> -- </wsdl:binding>

<wsdl:service> --- </wsdl:service>

</wsdl:definitions>

```

In this above PortType declaration, we declared one operation sayHello who has input/output as messages, which are referring to the messages we declared under messages section.

### <binding>:

Binding describes how the messages are exchanged between consumer and provider and in which format as well .i.e it will represents Message Exchange Format.

And Also It represents Transport Protocol.

In the binding the operations will specify soap:action for each operation. soapAction is used for resolving an input request to a method on the provider.

<binding> information for the above service described as follows.

```

<wsdl:definitions>

<wsdl:types> -- </wsdl:types>

<wsdl:message> --- </wsdl:message>

<wsdl:portType>---</wsdl:portType>

<wsdl:binding name="HelloServiceImplSoapBinding" type="impl:IHelloService">

    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="sayHello">

        <wsdlsoap:operation soapAction=""/>

        <wsdl:input name="sayHelloRequest">

            <wsdlsoap:body use="literal"/>

```



```

</wsdl:input>

<wsdl:output name="sayHelloResponse">

  <wsdlsoap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>
<wsdl:service> --- </wsdl:service>

</wsdl:definitions>

```

In the above binding we define the transport type as soap over http. Along with this we specifying the style and use model as document and literal respectively.

### <service>:

Service acts as a factory for create the <port> objects. Port attaches an endpoint to the binding defined above. <service> element wraps several ports under it and is responsible for creating the port objects at the consumer side.

For the above binding , the service declaration is shown below

```

<wsdl:definitions>

<wsdl:types> -- </wsdl:types>

<wsdl:message> --- </wsdl:message>

<wsdl:portType>----</wsdl:portType>

<wsdl:binding>----</wsdl:binding>

<wsdl:service name="HelloServiceImplService">

  <wsdl:port binding="impl:HelloServiceImplSoapBinding" name="impl:IHelloService">

    <wsdlsoap:address
location="http://localhost:8083/HelloServiceExample/services/HelloService"/>

  </wsdl:port>

</wsdl:service>

</wsdl:definitions>

```

**Complete WSDL document for IHelloService Example (document/literal)**

```

<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:intf="http://service.nareshit.com

xmlns:impl="http://service.nareshit.com

xmlns:apachesoap="http://xml.apache.org/xml-soap"
targetNamespace="http://service.nareshit.com">

    <wsdl:types>
        <schema elementFormDefault="qualified"
            targetNamespace="http://service.nareshit.com"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="name" type="xsd:string" />
            <element name="sayHelloReturn" type="xsd:string" />
        </schema>
    </wsdl:types>
    <wsdl:message name="sayHelloRequest">
        <wsdl:part element="impl:name" name="name">
        </wsdl:part>
    </wsdl:message>

    <wsdl:message name="sayHelloResponse">

        <wsdl:part element="impl:sayHelloReturn" name="sayHelloReturn">
        </wsdl:part>
    </wsdl:message>

    <wsdl:portType name="IHelloService">

        <wsdl:operation name="sayHello" parameterOrder="name">

            <wsdl:input message="impl:sayHelloRequest" name="sayHelloRequest">
            </wsdl:input>

            <wsdl:output message="impl:sayHelloResponse" name="sayHelloResponse">
            </wsdl:output>

        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="HelloServiceImplSoapBinding" type="impl:IHelloService">

        <wsdlsoap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />

        <wsdl:operation name="sayHello">

```

```

        <wsdlsoap:operation soapAction="" />

        <wsdl:input name="sayHelloRequest">
            <wsdlsoap:body use="literal" />
        </wsdl:input>

        <wsdl:output name="sayHelloResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>

    </wsdl:operation>

</wsdl:binding>

<wsdl:service name="HelloServiceImplService">
    <wsdl:port binding="impl:HelloServiceImplSoapBinding"
        name="impl:IHelloService">
        <wsdlsoap:address
            location="http://localhost:8083/HelloServiceExample/services/HelloService" />
    </wsdl:port>
</wsdl:service>

</wsdl:definitions>

```

# SOAP

SOAP(Simple Object Access Protocol) is a Xml based data format used to exchange the data between different applications. SOAP is a XML based messaging format which usually works on top of HTTP/HTTPS. However it can be used over other transport protocols like SMTP, SMTP etc.

SOAP is also interoperable format.

Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

## Disadvantage SOAP

SOAP messaging is slower compared to other messaging techniques like RPC, DCOM and CORBA. Because extra XML induces extra load on transport layer and extra processing on the receiver end.

## Basic building blocks of SOAP

*Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: [www.nareshit.com](http://www.nareshit.com)*

## A Soap Message Contains the following tags

**<soap:Envelope>**

**<soap:Header>**

**<soap:Body>**

<soap:Envelope> is a root element of a SOAP message

<soap:Header> is optional but each SOAP message

Contains <soap:body>

The request and response of the SOAP message will be inserted into body part of the SOAP.

Ex:- A request soap message for calling a service (method) called sayHello() method of a webservice.(document/literal)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://service.nareshit.com">
```

```
<soapenv:Header/>
```

```
<soapenv:Body>
```

```
<ser:name>?</ser:name>
```

```
</soapenv:Body>
```

```
</soapenv:Envelope>
```

The SOAP request message will be constructed by stub internally.

### Note:

The advantage of using a Document style model is that you can structure the SOAP body any way you want it as long as the content of the SOAP message body is any arbitrary XML instance. The Document style is also referred to as Message-Oriented style.

However, with an RPC style model, the structure of the SOAP request body must contain both the operation name and the set of method parameters. The RPC style model assumes a specific structure to the XML instance contained in the message body.

Ex:- A request soap message for calling a service (method) called sayHello() method of a webservice.(rpc/encoded)

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://service.nareshit.com">
```

```
<soapenv:Header/>
```

```
<soapenv:Body>
```

```
<ser:sayHello soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <name xsi:type="xsd:string">?</name>
</ser:sayHello>
</soapenv:Body>
</soapenv:Envelope>
```

**The following SOAP response message constructed by the Skeleton for returning sayHello() method result.**

### **SOAP response message(rpc/encoded):-**

---

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:sayHelloResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://service.nareshit.com">
      <sayHelloReturn xsi:type="xsd:string">Hello :sathish welcome to JAX-
RPC</sayHelloReturn>
    </ns1:sayHelloResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

### **SOAP response message(document/literal):-**

---

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <sayHelloReturn xmlns="http://service.nareshit.com">Hello :sathish welcome to JAX-
RPC</sayHelloReturn>
  </soapenv:Body>
</soapenv:Envelope>
```

### **SOAP Fault Element**

For unsuccessful SOAP response the soap body is creating with the faultCode as follows

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<soapenv:Body>
```

```
<soapenv:Fault>
```

```
<faultcode>soapenv:Server.userException</faultcode>
```

```
<faultstring>Name of the Exception : Desc of the Exception </faultstring>
```

```
<detail>
```

```
<ns1:hostname xmlns:ns1="http://xml.apache.org/axis/">Host-Name</ns1:hostname>
```

```
</detail>
```

```
</soapenv:Fault>
```

```
</soapenv:Body>
```

```
</soapenv:Envelope>
```

Lets dissect the above SOAP message

- **faultcode** – Code to classify the fault.
- **faultstring** – fault message that can be easily understandable
- **faultactor** – the actor in the system who is the cause of this error
- **detail** – detailed error parameters

### **Contract First Approach :**

Approach : Contract First Approach

API :JAXRPC

Implementation :JAXRPC-SI

JDK: JDK6

Endpoint:Servlet

MEF:document/literal

### **Steps:**

**Step1** : write WSDL document

**Step2:-** Write Configuration file with wsdl locations

**Step3:-** Generate the SEI and artifacts

For this we can use wscompile tool .

We know Already wscompile tool will takes configuration file as an input.

**Step4:-** write Service Implementation Class

**Step5:-** write webservises deployment description(jaxrpc-ri.xml)

**Step6:-** create Project as a war file

**Step7:-** run wsdeploy tool

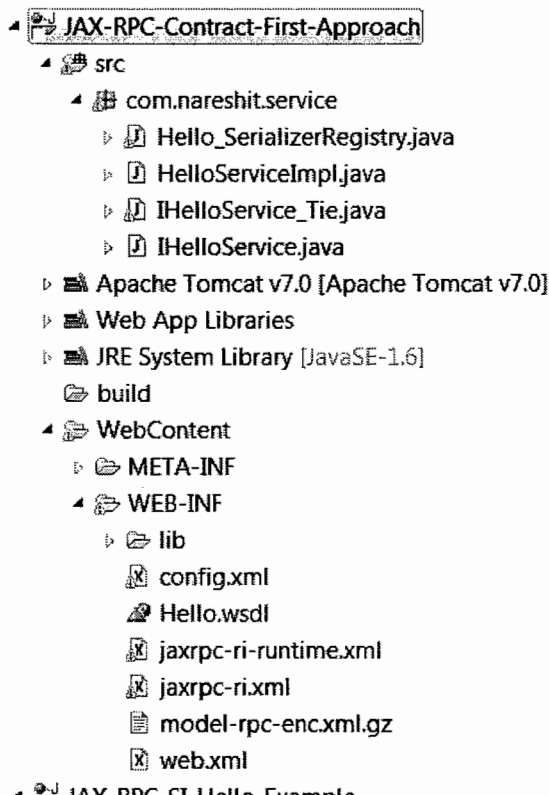
wsdeploy tool will takes your project war as an input and generates target.war

Ex:- wsdeploy -o target.war myApp.war

**Step8:-** Extract the target.war file and copy web.xml file and jaxrpc-ri-runtime.xml files

And paste into Your Project WEB-INF folder.

**Step9 :** deploy the Project into server



**Step1:**

**Hello.wsdl**

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://service.nareshit.com"
xmlns:impl="http://service.nareshit.com"
xmlns:intf="http://service.nareshit.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
```

```
<schema elementFormDefault="qualified" targetNamespace="http://service.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="name" type="xsd:string"/>
  <element name="sayHelloReturn" type="xsd:string"/>
</schema>
</wsdl:types>

<wsdl:message name="sayHelloResponse">
  <wsdl:part element="impl:sayHelloReturn" name="sayHelloReturn">
  </wsdl:part>
</wsdl:message>

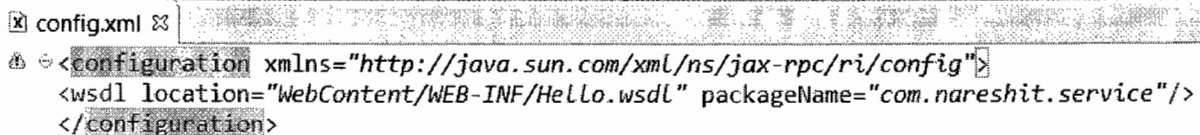
<wsdl:message name="sayHelloRequest">
  <wsdl:part element="impl:name" name="name">
  </wsdl:part>
</wsdl:message>

<wsdl:portType name="IHelloService">
  <wsdl:operation name="sayHello" parameterOrder="name">
    <wsdl:input message="impl:sayHelloRequest" name="sayHelloRequest">
    </wsdl:input>
    <wsdl:output message="impl:sayHelloResponse" name="sayHelloResponse">
    </wsdl:output>
  </wsdl:operation>
</wsdl:portType>

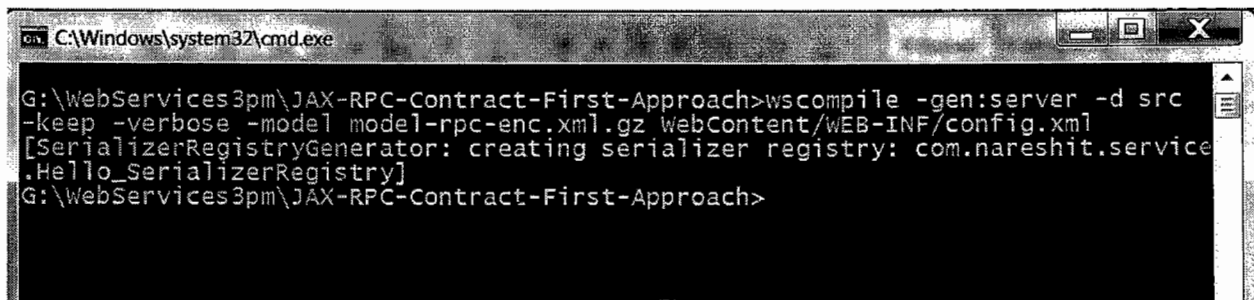
<wsdl:binding name="HelloServiceImplSoapBinding" type="impl:IHelloService">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="sayHello">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="sayHelloRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
```



```
<wsdl:output name="sayHelloResponse">
  <wsdlsoap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="Hello">
  <wsdl:port binding="impl:HelloServiceImplSoapBinding" name="IHelloService">
    <wsdlsoap:address location="http://localhost:8083/Jax-RPC-Contract-First-
Approach/hello"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

**Step2:**

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="WebContent/WEB-INF/Hello.wsdl" packageName="com.nareshit.service"/>
</configuration>
```

**Step3:**

```
C:\Windows\system32\cmd.exe
G:\WebServices3pm\JAX-RPC-Contract-First-Approach>wscompile -gen:server -d src
-keep -verbose -model model-rpc-enc.xml.gz WebContent/WEB-INF/config.xml
[SerializerRegistryGenerator: creating serializer registry: com.nareshit.service
.Hello_SerializerRegistry]
G:\WebServices3pm\JAX-RPC-Contract-First-Approach>
```

**Step4:**

```

HelloServiceImpl.java
package com.nareshit.service;

import java.rmi.RemoteException;

public class HelloServiceImpl implements IHelloService {

    @Override
    public String sayHello(String name) throws RemoteException {

        return "Hello :"+name+" welcome to JAX-RPC-Contract-First-Approach";

    }

}

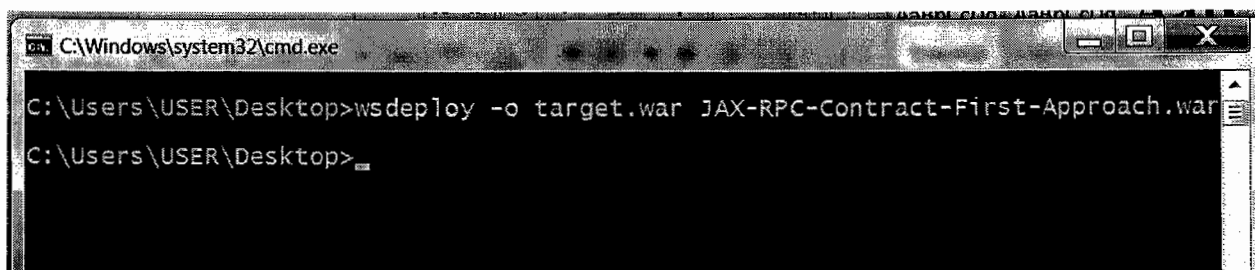
```

**Step5:**

```

jaxrpc-ri.xml
<?xml version="1.0" encoding="UTF-8"?>
<webServices xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd" version="1.0">
  <endpoint name="Hello" interface="com.nareshit.service.IHelloService"
    implementation="com.nareshit.service.HelloServiceImpl"
    wsdl="/WEB-INF/Hello.wsdl"
    model="/WEB-INF/model-rpc-enc.xml.gz">
  </endpoint>
  <endpointMapping endpointName="Hello" urlPattern="/hello"/>
</webServices>

```

**Step6: create Project as War file****Step7: run wsdeploy tool**


```

C:\Windows\system32\cmd.exe
C:\Users\USER\Desktop>wsdeploy -o target.war JAX-RPC-Contract-First-Approach.war
C:\Users\USER\Desktop>

```

**Step8:- Extract the target.war file and copy web.xml file and jaxrpc-ri-runtime.xml files**

And paste into Your Project WEB-INF folder.

**Step9 : deploy the Project into server**

## JAXRPC-Clients

### Dynamic Proxy Client

In this type of Consumer ,we don't need to generate any binding classes (OR) stub object at the

Design time. The Stub (Implementation of SEI interface at client side) will generate at runtime in the JVM.

In order to generate the stub class at runtime, we need to use the JAX-RPC API specific classes.

To develop This type Consumer first developer needs to identify the PortType, Operations and input and output messages, from which he needs to build SEI interface with the methods, parameters and return types.

Once the SEI interface has been developed, we need to use the following steps in the Client Application.

1) Create ServiceFactory object :

```
ServiceFactory factory=ServiceFactory.newInstance();
```

2) create Service from the ServiceFactory-

While creating the Service, we need to pass two parameters as input

1) WSDL URL

2) QName of the service.

So, while creating the service object, it loads the WSDL document and searches for the existence of the service QName in wsdl document. If available, with the definitions of that Service (reference to the Port) will create the service.

```
Service service=factory.createService(new URL(wsdl),new QName(tns,serviceName));
```

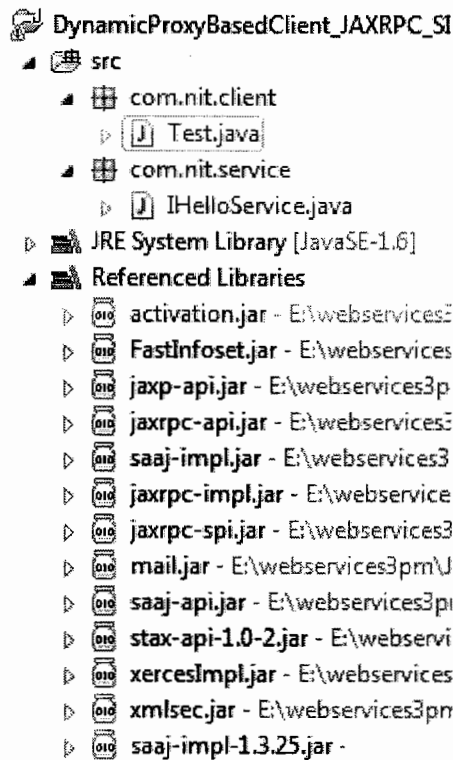
3) call getPort() method:

the getPort() method will return SEI object . i.e Stub object.

```
IHelloService sei=service.getPort(new QName(tns,portName),IHelloService.class);
```

4) then call the Service method

```
String msg=sei.sayHello(-);
```



we can develop the SEI as follows to Access a Webservice i.e IHelloService

```

IHelloService.java
1 package com.nit.service;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IHelloService extends Remote{
5     public String sayHello(String name)throws RemoteException;
6
7 }
8

```

**Write the Client application Code as follows**

```

1 package com.nit.client;
2 import java.net.MalformedURLException;
10 public class Test {
11 public static void main(String[] args)
12 throws ServiceException, MalformedURLException, RemoteException {
13 ServiceFactory serviceFactory=ServiceFactory.newInstance();
14 String tns="http://helloexample.com/wsdl";
15 String serviceName="Hello";
16 URL wsdl=new URL("http://localhost:8083/JAX-RPC-SI-Hello-Example/hello?WSDL");
17 String portName="IHelloServicePort";
18 //get service object from serviceFactory
19 Service service=serviceFactory.createService(wsdl,new QName(tns,serviceName));
20 IHelloService sei=
21 (IHelloService)service.getPort(new QName(tns,portName),IHelloService.class);
22 String data=sei.sayHello("sathish");
23 System.out.println(data);
24 }
25 }

```

### **Dynamic Invocation Interface Client :**

It is similar to Dynamic Proxy client.

In the case of Dynamic Proxy we need to develop SEI interface and method parameter and return types.

But in the case of DII client the SEI is not required to be create explicitly in the Client System.

The DII client calls any service method also dynamically by using invoke(-) method.

**Step1:-** create ServiceFactory object

**Step2:-** create Service object

**Step3:-** create Call object : Call is the Object which allows you to call a remote procedure on the Provider. To create Call object we need to call createCall() method on the service object. While creating the call object , we need to pass portName as a parameter to createCall() method.

Call call=service.createCall(new QName(tns,portName));

**Step4:-** set the Parameters to call object

i.e we need to set methodName,targetEndpointAddress(WSDL),Input and Output parameters etc..

call.setTargetEndPointAddress("WSDL URL");

call.setOperationName(new QName(tns,"sayHello"));

call.addParameter("String\_1",new QName(tns,typeName),ParameterMode.IN);

call.setReturnType(new QName(tsn,typename));

**step5:-** Along with that we need to set the below Properties

*Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: www.nareshit.com*

SOAPACTION\_URI\_PROPERTY

SOAPACTION\_USE\_PROPERTY

ENCODING\_STYLE\_PROPERTY

**Step6:-** invoke the method on the call object.

The following example shows to how access HelloService by using DII client.

```
1 HelloClient.java
2
3 1 import java.rmi.RemoteException;
4 7 public class HelloClient {
5 8 public static void main(String[] args) throws RemoteException, ServiceException {
6 9 String serviceName="Hello";
7 10 String portName="HelloServicePort";
8 11 String wsdlLoc="http://localhost:8083/JAX-RPC-SI-Hello-Example/hello?WSDL";
9 12 String tns="http://helloexample.com/wsdl";
10 13 ServiceFactory factory=ServiceFactory.newInstance();
11 14 Service service=factory.createService(new QName(tns,serviceName));
12 15 Call call=service.createCall(new QName(tns,portName));
13 16 call.setTargetEndpointAddress(wsdlLoc);
14 17 call.setProperty(Call.SOAPACTION_URI_PROPERTY,"");
15 18 call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
16 19 "http://schemas.xmlsoap.org/soap/encoding/");
17 20 call.setOperationName(new QName(tns,"sayHello"));
18 21 String xsd_ns="http://www.w3.org/2001/XMLSchema";
19 22 call.setReturnType(new QName(xsd_ns,"string"));
20 23 String msg=(String)call.invoke(new Object[]{"sathish"});
21 24 System.out.println(msg);
22 25 }
23 26 }
24 27
25 ~
```

# 6Java API for XML Web Services(JAX-WS)

The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating web services. It is part of the Java EE platform from Sun Microsystems.

--> JAX-WS API hides the complexity of web service development.

JAX-WS API versions 2.0, 2.1 and 2.2

The JAXRPC-2.0 is renamed as JAXWS-2.0 .

JAX-WS API follows Basic Profile 1.2 and 2.0 specifications

BasicProfile 1.2 --.Added support to WS-Addressing using MTOM

Basic Profile 2.0 --Added support to WS-Addressing ,MTOM and WSDL 2.0

JAX-WS provides much annotation to simplify the development and deployment for both web service clients and web service providers (endpoints).

--> If we are using JAVA 5 , then we need to physically add the jar files JAX-WS API and implementation jar files.

--> If we are using java 6 ,then as part of jdk itself we have JAX-WS API ,So just we need to add jar files of implementation.

API (From sun microsystem )	Implementation from different companies
JAX-WS	JAX-WS RI (from sun) Apache-Axis2 (from ASF) metro (from sun) CXF (from ASF) ... etc

Difference Between JAXRPC and JAXWS:

- 1) JAXRPC Follows BasicProfile 1.0 guidelines. JAXWS follows BasicProfile 1.1 guidelines
- 2) JAX-RPC supports WSDL 1.1 But JAXWS supports both WSDL 1.1 and 2.0 versions
- 3) JAXRPC supports support SOAP 1.1. But JAX WS supports both SOAP 1.1 and SOAP 1.2
- 4) WSDL 1.1 specification has support for HTTP Binding. It means You can send XML messages Over Http without SOAP. JAXRPC ignored HTTP binding, where are JAX-WS added support for Http Binding.
- 5) JAXRPC is compatible with JDK1.4 and higher where as JAX WS is compatible with JDK5 and higher.
- 6) JAXRPC does not supports Annotations But JAXWS supports annotations
- 7) JAXRPC is using Any binding API for Marshalling and UnMarshalling But JAXWS uses JAX-B binding API for Marshalling and UnMarshalling.
- 8) The Default of MEF of JAXRPC is rpc/encoded Where as JAXWS is document/literal
- 9) JAXRPC clients support only synchronous way of communication where AS JAXWS clients supports

both synchronous and asynchronous.

10) JAXRPC supports only SAAJ(SOAP with Attachments API).But JAXWS SAAJ and MTOM(Message Transmission Optimization Mechanism)

## JAXWS-RI Implementation

JAXWS RI is an implementation of JAXWS API. Used to develop/consume Big webservices in Java Platform.

The Current version of JAX-WS RI 2.2.10 .

We can download the JAXWSRI jar's from the following website

<https://jax-ws.java.net/2.2.10/>

To develop the webservices by using JAXWS-RI implementation we required to use wsimport and wsgen tools.These tools are by-default coming with JDK software only.

Steps to develop the webservice by using JAXWS-RI

### Approach : ContractLastApproach

API : JAXWS

Implementation : JAXWS-RI

Endpoint :Servlet

MEF :document/literal

Step1 : write SEI interface (optional)

Step2: write SEI implementation class

Step3: Generate the binding classes by using Wsgen tool (optional)

Step4: Write webservices deployment descriptor file i,e sun-jaxws.xml(configure the Endpoints)

Step5: write webapplication deployment descriptor file i,e web.xml (Configure the WSServlet,WSServletListener)

Step1 : Write SEI interface

As you know java programming point view and Interface acts as a contract between Provider and Consumer . SOA point of view a wsdl is acting as contract between client and service provider.In ContractLastApproach the development starts with SEI interface .



To create SEI in JAXRPC-SI(JAXRPC-API) implementation we need to follow some rules

SEI interface must extend java.rmi.Remote interface, to indicate the methods are accessible remotely.

Whatever the methods are we are declared under SEI All those are Exposed as a webservice methods.

The Method Parameters and return types must be Serializable and the Methods must throw RemoteException.

But if we are creating SEI in JAXWS-RS these rules are not required.

SEI creation In JAX-WS-RS implementation(JAX-WS API)

In this implementation with SEI interface no need to extend java.rmi.Remote interface. But to indicate as a webservice interface annotate with @WebService annotation.

As per Your requirement Declare the methods in SEI and you can choose which methods you want to expose as a webservice methods. By default all the methods are exposed as webservice methods.

For example if SEI interface is having 5 methods. Out of 5 methods if we want to expose only 1 one method as webservice method then annotate that method with @WebMethod annotation.

@WebMethod annotation contains exclude attribute.

If exclude attribute value declared as true then the method not exposing over the N.W.

If exclude attribute value declared as false then the method exposed as a webservice method.

The default value of exclude attribute value is false.

Ex:-

```
@WebMethod(exclude=false)
```

```
@WebMethod(exclude=true)
```

The SEI interface method parameters and return types are not necessary to be serializable, but as per Java coding standards recommended to be serializable.

The SEI interface methods may (OR) may not throw any Exception.

Note :In the case JAXWS API all the webservice methods throw WebServiceException. It is an UncheckedException

1 IHelloService.java

```
1 package com.nareshit.service;
2 import javax.xml.ws.WebService;
3 @WebService
4 public interface IHelloService {
5     public String sayHello(String name);
6 }
7
```

## Annotation to design WebServices JAX-WS-R1:

To design the webservice we should mainly use three annotations

1. @WebService (javax.xml.ws.WebService)
2. @WebParam(javax.xml.ws.WebParam)
3. @webMethod(javax.xml.ws.WebMethod)

1. @WebService : This annotation should be used for the webservice interface and for the webservice implementation class .

Attributes of @Webservice annotation:

--> endpoint interface

--> targetNamespace

--> service name

endpoint interface: this parameter will take the fully qualified name of the webservice interface .it is optional parametor in the @WebService this attribute we can use when we are using this annotation in implementation class level.

targetNamespace: It can be any name but it should be in the form of url .it is also an optional parameter in the @WebService

serviceName: It can be any name but we should provide meaningfull name

@WebMethod:

If we want to expose our webservice methods to the client application then we need to apply @WebMethod annotation to the methods .it is an optional annotation because by default the methods of SEI will expose to the clients as a webservice method.

@WebParam: this can be used for defining the parameters for webservice methods.

## Write SEI interface implementation class :-

The SEI Implementation class not mandatory to implement SEI interface .

The SEI implementation class we will denote with @Web Service annotation.

Implementation class methods need not throw WebService Exception.

If there is any logic throwing Exception, all such exceptions should be 'wrapped' into WebServiceException(OR) any of the sub classes of WebServiceException and should throw to the client. WebServiceException is an Unchecked Exception.

SEI is an optional:-

In JaxWs , the SEI interface is not mandatory , it is an optional. In case if you haven't written any SEI interface, all the annotations of Interface like @WebMethod, @WebParam

and @WebResult has to be declared directly in Implementation class itself.

If SEI is written you must declare them at the Interface Level. Only in the absence of Interface then only you should declare them in the class.

At the Consumer side to access the Provider , JAXWS API automatically generates SEI interface based on the WSDL to access it. Even we don't provide a SEI interface the Provider Side.

```
1 package com.nareshit.service;
2 import javax.jws.WebService;
3 @WebService
4 public class HelloServiceImpl implements IHelloService {
5     public String sayHello(String name){
6         return "Hello :"+name+" welcome to JAX-WS-RPC";
7     }
8 }
```

Generate Binding Classes :-

It is similar to the generation of binding classes in JAXRPC . Always the consumer sends XML embedded in SOAP Over Http request to Provider. The incoming XML has to be mapped to method parameters and outgoing return type has to be converted back to XML.

This means for the inputs and outputs of the methods we need to generate binding classes

Rather than writing the binding classes to perform this we use the tool that is provided by

JWSDP/JDK, nothing but wsgen. wsgen will take the input as annotated classes and generates

Binding classes to perform marshalling and unmarshalling.

Ex: wsgen -d src -keep -cp build\classes -verbose com.nareshit.service.HelloServiceImpl

# write sun-jaxws.xml:-

We need to provide the endpoint configurations, which is known as deployment descriptor .

To map our service with an URL, we need to write the deployment descriptor file called as sun-jaxws.xml and should be placed in WEB-INF directory as shown below

```
sun-jaxws.xml
1 <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
2 version="2.0">
3 <endpoint name="Hello"
4 implementation="com.nareshit.service.HelloServiceImpl"
5 url-pattern="/hello"/>
6 </endpoints>
```

write web.xml:-

As you know we are developing a Servlet Endpoint based Service, we need to configure a Servlet to handle our incoming request and to process it to return the response.

In case of JAX-WS-RI and Metro implementations we can configure WSServlet.

Here there is no tool like wsdeploy to generate the sun-jaxws.xml and web.xml the programmer has to manually configure them in these files.

While writing the WSServlet configuration in the web.xml n we need to ensure the URL pattern of the servlet and url-pattern of the endpointb in sun-jaxws.xml should be matching.

so that the servlet received the request would be able to identify the implementation class to Service the Request. Along with Servlet , we need to configure as Listener in the web.xml .

The Listener will be fired when the application Context has been initialized and checks the for the file sun-jaxws.xml under WEB-INF.

```

web.xml
1 <web-app>
2 <servlet>
3   <servlet-name>wsServlet</servlet-name>
4   <servlet-class>
5     com.sun.xml.ws.transport.http.servlet.WSServlet
6   </servlet-class>
7   <load-on-startup>1</load-on-startup>
8 </servlet>
9 <servlet-mapping>
10  <servlet-name>wsServlet</servlet-name>
11  <url-pattern>/hello</url-pattern>
12 </servlet-mapping>
13 <listener>
14  <listener-class>
15    com.sun.xml.ws.transport.http.servlet.WSServletContextListener
16  </listener-class>
17 </listener>
18 </web-app>

```

## Consumer Application:-

After Developing the provider, in order to access it, we need to build the Consumer. JAX-WS API supports both the developments of Provider as well as Consumer.

JAX-WS Supports two types of clients

- 1) Stub based
- 2) DispatchAPI

Stub Based :-

In this type of client we need to generate the stub and other artifacts by using wsimport tool.

The wsimport tool directly takes wsdl file as an input .

wsimport -d src -keep -verbose http://localhost:8083/JAX-WS-RI-Hello-Example/hello

## JAX-WS-Client(GeneratedStubBasedClient)

- src
  - com.nareshit.client
    - Test.java
  - com.nareshit.service
    - HelloServiceImpl.java
    - HelloServiceImplService.java
    - ObjectFactory.java
    - package-info.java
    - SayHello.java
    - SayHelloResponse.java
- JRE System Library [JavaSE-1.7]

```
1 package com.nareshit.client;
2 import com.nareshit.service>HelloServiceImpl;
3 import com.nareshit.service>HelloServiceImplService;
4 public class Test {
5     public static void main(String[] args) {
6         HelloServiceImplService service=new HelloServiceImplService();
7         HelloServiceImpl sei=service.getHelloServiceImplPort();
8         String msg=sei.sayHello("sathish");
9         System.out.println(msg);
10    }
11 }
```

## ContractFirstApproach:-

API: JAXWS

Implementation:JAXWSRI

JDK:JDK7

ENDPOINT:Servlet

MEF:document/literal

As we know In contract first Approach the development will starts with WSDL document.

WSDL describes the entire information about the service including the input /output complex types,port type,binding etc..

Using WSDL document we can generate the classes that are required to create Service.

Steps:

Step 1: write WSDL

Step2 : Generate the Java and Binding classes from WSDL

For this we need run wsimport tool and required to pass WSDL as input and should

generate required classes and binding classes as an output.

wsimport takes directly the WSDL as input and generates the classes supporting to build both provider as well as Consumer.

Ex:-

```
wsimport -d src -keep -verbose WebContent\WEB-INF\Hello.wsdl
```

It generates SEI interface, binding classes.

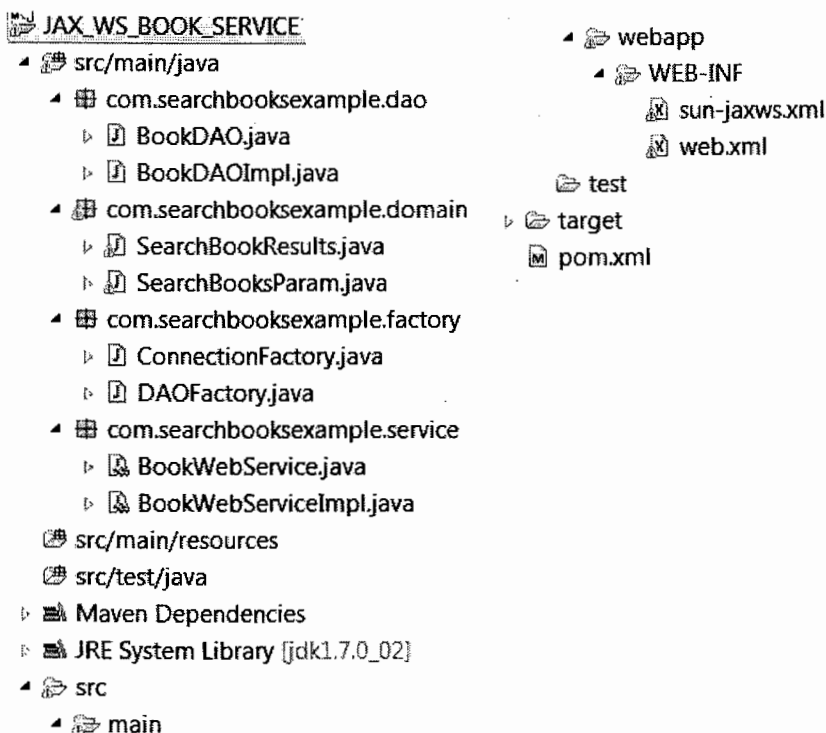
Step 3: write Implementation class for SEI interface with the Required Business Logic

Step 4: write sun-jaxws.xml and web.xml

Step5: deploy the application into the server

## BookService Example in Contract Last Approach(JAX-WS-RI\_)

BookService Provider :



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
```

```

<modelVersion>4.0.0</modelVersion>
<groupId>com.searchbooksexample</groupId>
<artifactId>JAX_WS_BOOK_SERVICE1</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>JAX_WS_BOOK_SERVICE1 Maven Webapp</name>
<url>http://maven.apache.org</url>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
<dependency>
  <groupId>com.sun.xml.ws</groupId>
  <artifactId>jaxws-rt</artifactId>
  <version>2.2.10</version>
</dependency>
<dependency>
  <groupId>oracle.jdbc.driver</groupId>
  <artifactId>ojdbc14</artifactId>
  <version>10.2.0</version>
</dependency>
</dependencies>
<build>
  <finalName>JAX_WS_BOOK_SERVICE1</finalName>
</build>
</project>

```

```

ConnectionFactory.java
1 package com.searchbooksexample.factory;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5 public class ConnectionFactory {
6     static{
7         try {
8             Class.forName("oracle.jdbc.driver.OracleDriver");
9         } catch (ClassNotFoundException e) {
10             e.printStackTrace();
11         }
12     }
13     public static Connection getConnection()
14         throws SQLException{
15         String url="jdbc:oracle:thin:@localhost:1521:XE";
16         String username="system";
17         String password="manager";
18         Connection con=DriverManager.getConnection(url,username,password);
19         return con;
20     }
21 }

```



```

1 package com.searchbooksexample.dao;
2 import java.util.List;
3 import com.searchbooksexample.domain.SearchBookResults;
4 import com.searchbooksexample.domain.SearchBooksParam;
5 public interface BookDAO {
6     public List<SearchBookResults>
7     searchBooks(SearchBooksParam searchBookParam);
8
9 }
10

```

## BookDAOImpl.java

```

package com.searchbooksexample.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import com.searchbooksexample.domain.SearchBookResults;
import com.searchbooksexample.domain.SearchBooksParam;
import com.searchbooksexample.factory.ConnectionFactory;
public class BookDAOImpl implements BookDAO {
    public List<SearchBookResults>
    searchBooks(SearchBooksParam searchBookParam){
        List<SearchBookResults> list=new ArrayList<SearchBookResults>();
        Connection con=null;
        try {
            con=ConnectionFactory.getConnection();
            StringBuilder sb=new StringBuilder("select isbn,title,author,price,publiation from Book_Details ");
            boolean flag=true;
            if(searchBookParam.getIsbn()!=null
            && searchBookParam.getIsbn().length()>0){
                sb.append(" where isbn="+searchBookParam.getIsbn());
                flag=false;
            }
            if(searchBookParam.getAuthor()!=null &&
            searchBookParam.getAuthor().trim().length()>0){
                if(flag){
                    sb.append(" where author="+searchBookParam.getAuthor()+"");
                    flag=false;
                }else{
                    sb.append(" And author="+searchBookParam.getAuthor()+"");
                }
            }
            if(searchBookParam.getTitle()!=null &&
            searchBookParam.getTitle().trim().length()>0){
                if(flag){
                    sb.append(" where title="+searchBookParam.getTitle()+"");
                }else{
                    sb.append(" And title="+searchBookParam.getTitle()+"");
                }
            }
            PreparedStatement pst=con.prepareStatement(sb.toString());
            ResultSet rs=pst.executeQuery();

```

```

while(rs.next()){
SearchBookResults searchBookResults=new SearchBookResults();
searchBookResults.setIsbn(rs.getInt("isbn"));
searchBookResults.setAuthor(rs.getString("author"));
searchBookResults.setTitle(rs.getString("title"));
searchBookResults.setPrice(rs.getDouble("price"));
searchBookResults.setPublication(rs.getString("publication"));
list.add(searchBookResults);
}
}catch (SQLException e) {
System.out.println("Exception Occured while searching the books :"+e.getMessage());
}
finally{
    if(con!=null){
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println("Exception Occured while closing the Connection :"+e.getMessage());
        }
    }
}

return list;
}
}

```

SearchBooksParam.java

```

1 package com.searchbooksexample.domain;
2 import java.io.Serializable;
3 public class SearchBooksParam implements Serializable{
4     private Integer isbn;
5     private String author,title;
6     public Integer getIsbn() {
7         return isbn;
8     }
9     public void setIsbn(Integer isbn) {
10         this.isbn = isbn;
11     }
12     public String getAuthor() {
13         return author;
14     }
15     public void setAuthor(String author) {
16         this.author = author;
17     }
18     public String getTitle() {
19         return title;
20     }
21     public void setTitle(String title) {

```

SearchBookResults.java

```
1 package com.searchbooksexample.domain;
2 import java.io.Serializable;
3 public class SearchBookResults implements Serializable{
4     private Integer isbn;
5     private String author,title,publication;
6     private Double price;
7     public Integer getIsbn() {
8         return isbn;
9     }
10    public void setIsbn(Integer isbn) {
11        this.isbn = isbn;
12    }
13    public String getAuthor() {
14        return author;
15    }
16    public void setAuthor(String author) {
17        this.author = author;
18    }
19    public String getTitle() {
20        return title;
21    }
```

DAOFactory.java

```
1 package com.searchbooksexample.factory;
2 import com.searchbooksexample.dao.BookDAO;
4 public class DAOFactory {
5     private static BookDAO bookDAO;
6     static{
7         bookDAO=new BookDAOImpl();
8     }
9     public static BookDAO getBookDAO(){
10         return bookDAO;
11     }
12 }
13
```

BookWebService.java

```
1 package com.searchbooksexample.service;
2 import java.util.List;
3 import javax.jws.WebMethod;
4 import javax.jws.WebService;
5 import com.searchbooksexample.domain.SearchBookResults;
6 import com.searchbooksexample.domain.SearchBooksParam;
7 @WebService
8 public interface BookWebService {
9     @WebMethod(exclude=false) //optional
10    public List<SearchBookResults>
11        searchBooks(SearchBooksParam searchBooksParam);
12
13 }
```

BookWebServiceImpl.java

```
1 package com.searchbooksexample.service;
2 import java.util.List;
3 import javax.jws.WebService;
4 import com.searchbooksexample.domain.SearchBookResults;
5 import com.searchbooksexample.domain.SearchBooksParam;
6 import com.searchbooksexample.factory.DAOFactory;
7 @WebService
8 public class BookWebServiceImpl implements BookWebService{
9     public List<SearchBookResults>
10    searchBooks(SearchBooksParam searchBooksParam) {
11        List<SearchBookResults> list=null;
12        if(searchBooksParam!=null){
13            list= DAOFactory.getBookDAO().searchBooks(searchBooksParam);
14        }
15        return list;
16    }
17
18 }
```

sun-jaxws.xml

```
1 <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
2 version="2.0">
3     <endpoint name="BookService"
4         implementation="com.searchbooksexample.service.BookWebServiceImpl"
5         url-pattern="/searchBooks" />
6 </endpoints>
```

web.xml

```
1 <web-app>
2 <servlet>
3   <servlet-name>wsServlet</servlet-name>
4   <servlet-class>
5     com.sun.xml.ws.transport.http.servlet.WSServlet
6   </servlet-class>
7   <load-on-startup>1</load-on-startup>
8 </servlet>
9 <servlet-mapping>
10  <servlet-name>wsServlet</servlet-name>
11  <url-pattern>/searchBooks</url-pattern>
12 </servlet-mapping>
13 <listener>
14  <listener-class>
15    com.sun.xml.ws.transport.http.servlet.WSServletContextListener
16  </listener-class>
17 </listener>
18 </web-app>
19
```

JAX\_WS\_Book\_Service\_Client

src

com.searchbooksexample.service

- BookWebServiceImpl.java
- BookWebServiceImplService.java
- ObjectFactory.java
- package-info.java
- SearchBookResults.java
- SearchBooks.java
- SearchBooksParam.java
- SearchBooksResponse.java

com.searchbooksexampleclient.controller

- BookServlet.java

com.searchbooksexampleclient.service

- BookServiceClient.java
- BookServiceClientImpl.java

JRE System Library [JavaSE-1.6]

Web App Libraries

build

WebContent

META-INF

WEB-INF

WEB-INF

lib

pages

searchBooks.jsp

searchBooksResults.jsp

web.xml

```

searchBooks.jsp
<html>
<head>
<h1 align="center">Search Books</h1><br/><hr/>
</head>
<body>
<form action="searchBooks" method="post">
<input type="text" name="isbn" placeholder="isbn"/>
<input type="text" name="title" placeholder="title">
<input type="text" name="author" placeholder="author">
<input type="submit" value="search"/>
</form>
</body>
</html>

```

```

web.xml
<web-app>
<servlet>
<servlet-name>BookServlet</servlet-name>
<servlet-class>com.searchbooksexampleclient.controller.BookServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>BookServlet</servlet-name>
<url-pattern>/searchBooks</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>/WEB-INF/pages/searchBooks.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

#### BookServlet.java

```

package com.searchbooksexampleclient.controller;
import java.io.IOException;
import java.util.List;
import javax.servlet.*;
import javax.servlet.http.*;
import com.searchbooksexample.service.SearchBookResults;
import com.searchbooksexample.service.SearchBooksParam;
import com.searchbooksexampleclient.service.BookServiceClient;
import com.searchbooksexampleclient.service.BookServiceClientImpl;
public class BookServlet extends HttpServlet{
private BookServiceClient bookService;
public void init(){
bookService=new BookServiceClientImpl();
}
public void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException{
String isbn=req.getParameter("isbn");
if(isbn==null || isbn.trim().length()==0){
isbn="0";
}
Integer isbn1=Integer.valueOf(isbn);
String author=req.getParameter("author");
String title=req.getParameter("title");
SearchBooksParam searchBooksParam=new SearchBooksParam();

```

```

searchBooksParam.setIsbn(isbn1);
searchBooksParam.setTitle(title);
searchBooksParam.setAuthor(author);
List<SearchBookResults>
searchBookResultsList=bookService.searchBooks(searchBooksParam);
req.setAttribute("searchBookResultsList",searchBookResultsList);
String target="/WEB-INF/pages/searchBooksResults.jsp";
RequestDispatcher rd=req.getRequestDispatcher(target);
rd.forward(req,res);
}
}

```

```

BookServiceClient.java
package com.searchbooksexampleclient.service;
import java.util.List;
import com.searchbooksexample.service.SearchBookResults;
import com.searchbooksexample.service.SearchBooksParam;
public interface BookServiceClient {
List<SearchBookResults> searchBooks(SearchBooksParam searchBooksParam);
}

```

```

BookServiceClientImpl.java
package com.searchbooksexampleclient.service;
import java.util.List;
import com.searchbooksexample.service.BookWebServiceImpl;
import com.searchbooksexample.service.BookWebServiceServiceImpl;
import com.searchbooksexample.service.SearchBookResults;
import com.searchbooksexample.service.SearchBooksParam;
public class BookServiceClientImpl implements BookServiceClient {
public List<SearchBookResults>
searchBooks(SearchBooksParam searchBooksParam) {
BookWebServiceServiceImpl service=new
BookWebServiceServiceImpl();
BookWebServiceImpl sei=service.getBookWebServiceImplPort();
List<SearchBookResults> list=sei.searchBooks(searchBooksParam);
return list;
}
}

```

SearchBooksResults.jsp

```

<%@page isELIgnored="false"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@include file="searchBooks.jsp" %>
<table border="1">
<tr>
<th>isbn</th><th>author</th><th>title</th>
<th>publication</th><th>price</th>
</tr>
<c:choose>
<c:when test="${searchBookResultsList.size()}>0">
<c:forEach items="${searchBookResultsList}"
var="searchBookResults">
<tr><td>
${searchBookResults.isbn}</td><td>
${searchBookResults.author}</td><td>
${searchBookResults.title}</td><td>
${searchBookResults.publication}</td><td>
${searchBookResults.price}</td></tr>

```

```
</c:forEach>
</c:when>
<c:otherwise>
<tr>
<td colspan="5">No Records Found</td>
</tr>
</c:otherwise>
</c:choose></table>
```

# Apache Axis2

Apache Axis2 is the implementation of JAXWS API.

Apache Axis2™ is a Web Services / SOAP / WSDL engine, the successor to the widely used Apache Axis.

Apache Axis2 not only supports SOAP 1.1 and SOAP 1.2, but it also has integrated support for the widely popular REST style of Web services.

Axis2 comes with many new features, enhancements and industry specification implementations. The key features are as follows:

**Speed** - Axis2 uses its own object model and StAX (Streaming API for XML) parsing to achieve significantly greater speed than earlier versions of Apache Axis.

**Asynchronous Web services** - Axis2 now supports asynchronous Web services and asynchronous Web services invocation using non-blocking clients and transports.

**MEP Support** - Axis2 now comes handy with the flexibility to support Message Exchange Patterns (MEPs) with in-built support for basic MEPs defined in WSDL 2.0.

**WSDL support** - Axis2 supports the Web Service Description Language, version 1.1 and 2.0, which allows you to easily build stubs to access remote services,

Apache Axis2 software is coming in the form of two distributions

1) Binary Distribution

2) War Distribution

1) Binary Distribution --> which contains the Jar's and tools (Java2WSDL, WSDL2Java) that facilitate the development of Web Services.

2) War Distribution → Acts as Server Side Runtime engine in which Your Services are deployed and exposed.

We can download axis2 software distributions from <https://axis.apache.org/axis2/java/core/download.cgi>.

After Downloading the Axis2 Distributions first we can set path Environment variables of Axis2 binary distributions.



**New User Variable**

Variable name: path

Variable value: %services%\axis2-1.6.2-bin\axis2-1.6.2\bin;

OK Cancel

**Edit User Variable**

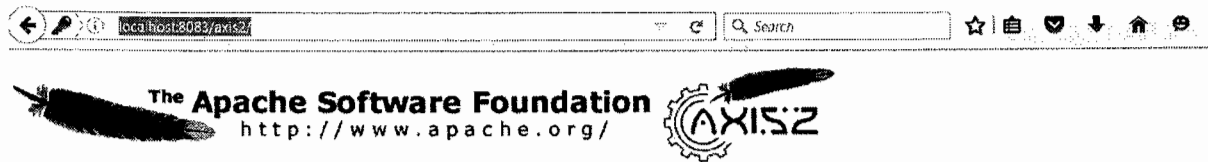
Variable name: AXIS2\_HOME

Variable value: F:\webservices\axis2-1.6.2-bin\axis2-1.6.2

OK Cancel

After setting the Environment variables for binary distribution we can deploy the axis2.war file into any server.

Then Check axis2 welcome page is opening (OR) not.



## Welcome!

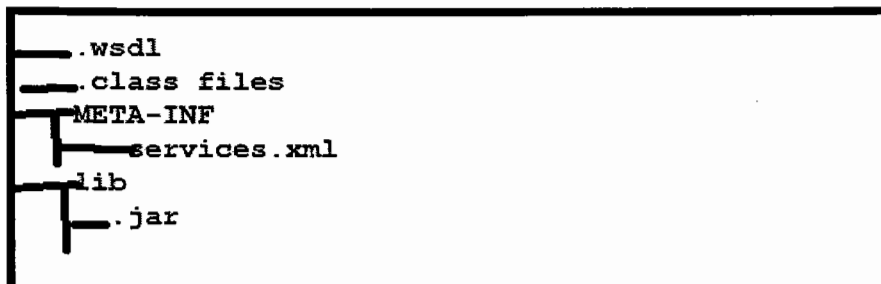
Welcome to the new generation of Axis. If you can see this page you have successfully deployed the Axis2 Web Application. However, to ensure that Axis2 is properly working, encourage you to click on the validate link.

- [Services](#)  
View the list of all the available services deployed in this server.
- [Validate](#)  
Check the system to see whether all the required libraries are in place and view the system information.
- [Administration](#)  
Console for administering this Axis2 installation.

Axis2 Project Contract Last Approach :

Must be maintain the Project Directory Structure as follows

### Axis2ProjectDirectoryStructure



Step1 : create the service classes as per your requirement

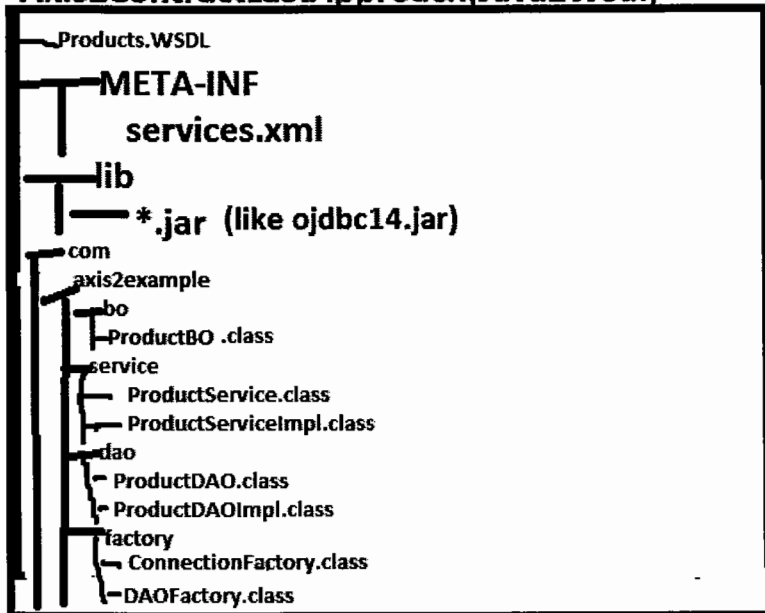
Step2: write services.xml file

Step3: Generate the WSDL document by using Java2WSDL document

Step4: create the project as an archive file (.aar extension) (Axis Archive)

Step5:- Deploy the archive file into axis2 war distribution

### Axis2ContractLastApproach(Java2Wsdll)



#### ProductService.java

```
1 package com.axis2example.service;
2 import com.axis2example.bo.ProductBO;
3 public interface ProductService{
4     public boolean registerProduct(ProductBO productBo);
5 }
```

### ProductServiceImpl.java

```
1 package com.axis2example.service;
2 import com.axis2example.bo.ProductBO;
3 import com.axis2example.factory.DAOFactory;
4 public class ProductServiceImpl implements ProductService{
5     public boolean registerProduct(ProductBO productBo){
6         boolean flag=false;
7         if(productBo!=null){
8             int count=DAOFactory.getProductDAO().registerProduct(productBo);
9             if(count>0){
10                 flag=true;
11             }
12         }
13         return flag;
14     }
15 }
```

### ProductDAO.java

```
1 package com.axis2example.dao;
2 import com.axis2example.bo.ProductBO;
3 public interface ProductDAO {
4     public int registerProduct(ProductBO productBO);
5 }
6 }
```

### ProductServiceImpl.java

```
1 package com.axis2example.dao;
2 import com.axis2example.bo.ProductBO;
3 import com.axis2example.factory.ConnectionFactory;
4 import java.sql.*;
5 public class ProductDAOImpl implements ProductDAO{
6     public int registerProduct(ProductBO productBo){
7         int count=0;
8         Connection con=null;
9         try{
10             con=ConnectionFactory.getConnection();
11             String sql="insert into Product_Details values(?,2,?)";
12             PreparedStatement pst=con.prepareStatement(sql);
13             pst.setInt(1,productBo.getProductID());
14             pst.setString(2,productBo.getProductName());
15             pst.setDouble(3,productBo.getPrice());
16             count=pst.executeUpdate();
17         }catch(SQLException se){
18             se.printStackTrace();
19         }
20     finally{
21         if(con!=null){
22             try{
23                 con.close();
24             }catch(SQLException se){
25                 se.printStackTrace();
26             }
27         }
28     }
29     return count;
30 }
```

### **ProductBO.java**

```
1 package com.axis2example.bo;
2 import java.io.Serializable;
3 public class ProductBO implements Serializable{
4     private Integer productId;
5     private String productName;
6     private Double price;
7     public void setProductId(Integer productId){
8         this.productId=productId;
9     }
10    public void setProductName(String productName){
11        this.productName=productName;
12    }
13    public void setPrice(Double price){
14        this.price=price;
15    }
16    public String getProductName(){
17        return productName;
18    }
19    public Integer getProductId(){
20        return productId;
21    }
22    public Double getPrice(){
23        return price;
24    }
25 }
```

### **ConnectionFactory.java**

```
1 package com.axis2example.factory;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5 public class ConnectionFactory {
6     static{
7         try {
8             Class.forName("oracle.jdbc.driver.OracleDriver");
9         } catch (ClassNotFoundException e) {
10             e.printStackTrace();
11         }
12     }
13     public static Connection getConnection() throws SQLException{
14         String url="jdbc:oracle:thin:@localhost:1521:XE";
15         String username="system";
16         String password="manager";
17         Connection con=DriverManager.getConnection(url,username,password);
18         return con;
19     }
20 }
21
```

#### **DAOFactory.java**

```
1 package com.axis2example.factory;
2 import com.axis2example.dao.ProductDAO;
3 import com.axis2example.dao.ProductDAOImpl;
4 public class DAOFactory {
5     private static ProductDAO productDAO;
6     static{
7         productDAO=new ProductDAOImpl();
8     }
9     public static ProductDAO getProductDAO(){
10         return productDAO;
11     }
12 }
13
```

Write the services.xml file as follows

#### **services.xml**

```
1 <service name="Product" scope="application">
2     <messageReceivers>
3         <messageReceiver mep="http://www.w3.org/ns/wsd1/in-only"
4             class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
5         <messageReceiver mep="http://www.w3.org/ns/wsd1/in-out"
6             class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
7     </messageReceivers>
8
9     <parameter name="ServiceClass">
10         com.axis2example.service.ProductServiceImpl
11     </parameter>
12 </service>
```

Generate the WSDL by using java2wsdl tool as follows

java2wsdl -cp . -cn com.axis2example.service.ProductServiceImpl -of Product.wsdl  
create the project as an archive file (.aar) and deploy into the server.

You can compress your documents into an \*.aar file, similar to a \*.jar file, and place the \*.aar file directly in the servlet engine's webapps/axis2/WEB-INF/services directory.

### **Axis2 Client Application :**

It is possible to create Axis2 clients in four ways .

- 1)building an AXIOM based client,
- 2) generating a client using Axis2 Databinding Framework (ADB),
- 3) generating a client using XMLBeans,
- 4) generating a client using JiBX.

If we are not specifying any format by default it will generate ADB client



After creating Project directory structure generate the client side artifacts by using wsdl2java tool

```
wsdl2java -uri http://localhost:8080/axis2/services/Product?wsdl
```

(OR)

```
wsdl2java -uri http://localhost:8080/axis2/services/Product?wsdl -d adb -s -o build\client
```

Then after generating the client side artifacts we can write the logic and access the service

ProductServiceClient.java

```
package com.axis2example.client.service;
import com.axis2example.service.ProductStub;
import com.axis2example.service.ProductStub.ProductBO;
import com.axis2example.service.ProductStub.RegisterProduct;
import com.axis2example.service.ProductStub.RegisterProductResponse;
import com.axis2example.client.vo.ProductVO;
import java.rmi.*;
public class ProductServiceClient{
    public boolean registerProduct(ProductVO productVO){
        boolean flag=false;
        try{
            ProductBO productBO=new ProductBO();
            productBO.setProductId(Integer.valueOf(productVO.productId));
            productBO.setProductName(productVO.productName);
            productBO.setPrice(Double.valueOf(productVO.price));
            ProductStub stub=new ProductStub();
            RegisterProduct registerProduct=new RegisterProduct();
            registerProduct.setArgs0(productBO);
            RegisterProductResponse registerProductResponse=stub.registerProduct(registerProduct);
            flag=registerProductResponse.get_return();
        }catch(RemoteException re){
            re.printStackTrace();
        }
        return flag;
    }
}
```

ProductVO.java

```
package com.axis2example.client.vo;
import java.io.*;
public class ProductVO implements Serializable
{
    public String productId,productName,price;
}
```

# ProductServlet.java

```
package com.axis2example.client.controller;
import javax.servlet.*;
public class ProductServlet extends HttpServlet {
    private ProductServiceClient productServiceClient;
    public void init() {
        productServiceClient = new ProductServiceClient();
    }
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        String status = null;
        ProductVO productVO = new ProductVO();
        productVO.productId = req.getParameter("productId");
        productVO.productName = req.getParameter("productName");
        productVO.price = req.getParameter("price");
        boolean flag = productServiceClient.registerProduct(productVO);
        if (flag == true) {
            status = "Product Registration Completed Successfully";
        } else {
            status = "Product Registration is failure";
        }
        req.setAttribute("status", status);
        String targetViewName = "/productRegForm.jsp";
        RequestDispatcher rd = req.getRequestDispatcher(targetViewName);
        rd.forward(req, res);
    }
}
```

# productRegForm.jsp

```
<%@page isELIgnored="false"%>
<html>
<head> <h1 align="center">ProductRegform</head>
<br/><hr/>
<${status}>
<form action="registerProduct" method="post">
    ProductId :<input type="text" name="productId"/>
    ProductName:<input type="text" name="productName"/>
    Price :<input type="text" name="price"/>
    <input type="submit" value="register"/>
</form>
</html>
```

# web.xml

```
<web-app>
<servlet>
    <servlet-name>ProductServlet</servlet-name>
    <servlet-class>com.axis2example.client.controller.ProductServlet
</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>ProductServlet</servlet-name>
    <url-pattern>/registerProduct</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>/productRegForm.jsp</welcome-file>
</welcome-file-list>
</web-app>
```



## Axis2 ContractFirstApproach

Step1 : write the wsdl document

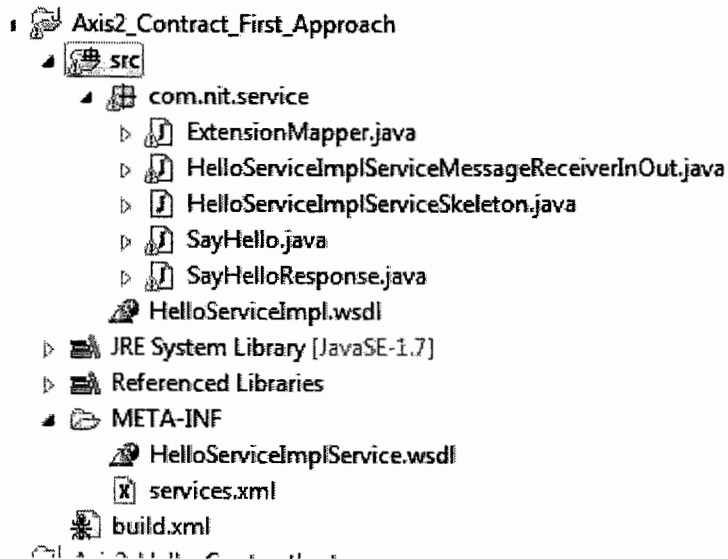
Step2: Generate the service classes and services.xml file by using wsdl2java tool

Step3: write the Bussiness logic in the generated Service class.

Step4: create the project as an archive file (.aar extension)

Step5:- Deploy project into the war distribution

### ContractFirstApproach Application :



### Step1 : write the wsdl document

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://service.nit.com" xmlns:apachesoap="http://xml.apache.org/xml-
soap" xmlns:impl="http://service.nit.com" xmlns:intf="http://service.nit.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://service.nit.com"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="sayHello">
        <complexType>
          <sequence>
            <element name="name" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="sayHelloResponse">
        <complexType>
          <sequence>
            <element name="sayHelloReturn" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
</wsdl:definitions>
```

```

    </element>
  </schema>
</wsdl:types>

  <wsdl:message name="sayHelloRequest">
    <wsdl:part element="impl:sayHello" name="parameters">
    </wsdl:part>
  </wsdl:message>

  <wsdl:message name="sayHelloResponse">
    <wsdl:part element="impl:sayHelloResponse" name="parameters">
    </wsdl:part>
  </wsdl:message>

  <wsdl:portType name="HelloService">
    <wsdl:operation name="sayHello">
      <wsdl:input message="impl:sayHelloRequest" name="sayHelloRequest">
      </wsdl:input>
      <wsdl:output message="impl:sayHelloResponse" name="sayHelloResponse">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="HelloServiceImplSoapBinding" type="impl:HelloService">
    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sayHello">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="sayHelloRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="sayHelloResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="HelloServiceImplService">

```

```

        <wsdl:port binding="impl:HelloServiceImplSoapBinding" name="HelloService">
            <wsdlsoap:address
location="http://localhost:8083/Axis2_Contract_First_Approach/services/HelloService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

**Step2: Generate the service classes and services.xml file by using wsdl2java tool**

```
wsdl2java -ss -sd -uri src/HelloServiceImpl.wsdl
```

Step3:- Write Bussiness Logic In the generated Skeleton class

```

/**
 * HelloServiceImplServiceSkeleton.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis2 version: 1.6.2 Built on : Apr 17, 2012 (05:33:49 IST)
 */
package com.nit.service;

/**
 * HelloServiceImplServiceSkeleton java skeleton for the axisService
 */
public class HelloServiceImplServiceSkeleton{

    /**
     * Auto generated method signature
     *
     * @param sayHello
     * @return sayHelloResponse
     */

    public com.nit.service.SayHelloResponse sayHello
    (

```

```
        com.nit.service.SayHello sayHello  
    )  
    {  
        String msg="Hello :"+sayHello.getName()+" welcome";  
        SayHelloResponse sayHelloResponse=new  
            SayHelloResponse();  
        sayHelloResponse.setSayHelloReturn(msg);  
        return sayHelloResponse;  
    }  
}
```

Step4:- create Project as an archive file (.aar)

Step5:- Deploy into the War distribution

**Restfullwebservices:**

REST stands for Representational State Transfer. REST is an architectural style not a protocol. REST is web standards based architecture and uses HTTP Protocol for data communication. In Rest every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

**Restfull webservices are supporting different data formats like JSON,XML,Html,PlainText**

**JAXRS API:**

**JAXRS-> JAXRS is API from sun .**

JaxRs API is used to develop/access the Restfull webservices in java platform.

**JAXRS** stands for JavaAPIForXml Restfull services.

JAXRS Versions 0.x ,1.x (1.0, 1.1), 2.x

JAX-RS 2.0 is the latest major release of JAX-RS.

JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

From version JAX-RS 1.1 onwards , JAX-RS is an official part of Java EE 6

The Java API for RESTful Web Services provides portable APIs for developing, exposing and accessing Web applications designed and implemented in compliance with principles of REST architectural style.

Packages	
Package	Description
javax.ws.rs	High-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.client	The JAX-RS client API
javax.ws.rs.container	Container-specific JAX-RS API.
javax.ws.rs.core	Low-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.ext	APIs that provide extensions to the types supported by the JAX-RS API.

**Implementations of Jax RS API:-**

- Apache CXF, an open source Web service framework
- Jersey, the reference implementation from Sun (now Oracle)
- RESteasy, JBoss's implementation
- Restlet from jerome Louvel.
- Apache Wink, Apache Software Foundation

**Difference between SOAP and Restfull webservice**

--> In soap webservices we can use only xml to make communication between the client and server provider .

--> In Restfull webservice we can make communication in between server provider and client using

--> XML

--> JSON (javascriptObjectNotation)

--> text

--> HTML

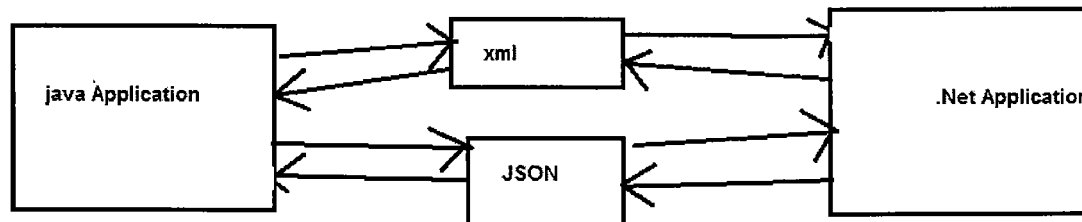
--> soap WebService supports both server level and application level security

--> But Restfull WebServices support only server level security but not for the Application level security.

**JSON:**

**JSON stands for JavaScript Object Notation**

JSON is also communicator like xml is between two language Applications.



--> we can apply Restriction xml document to accept the data by using DTD or XSD.

--> JSON we can't restrict to accept the data because it is normal text document.

JSON format is interoperable format

interoperable->lang independent and platform independent

JSON format is very lightweight format compared to SOAP/XML.

JSON is a text-based document

**ex:-**

```
{"studentId":101,"name":"raja","course":"java"}
```

--> To read the JSON format data from the java application and to construct JSON format data by the java application we require JSON API.

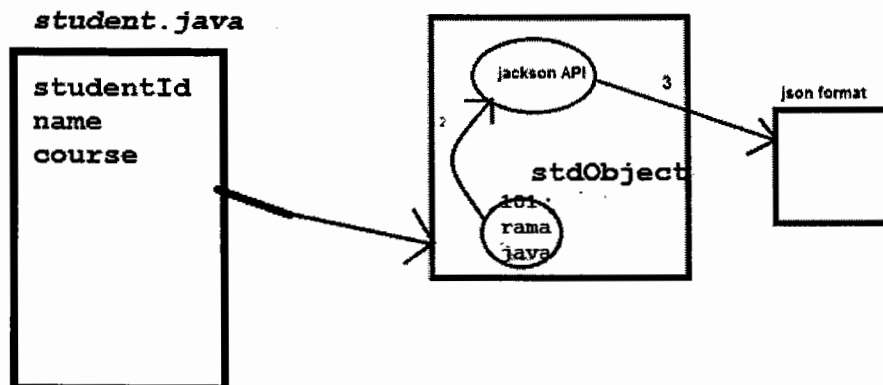
***Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: [www.nareshit.com](http://www.nareshit.com)***

Different vendors are provides JSON API for the java language.

1. jackson API
2. Gson (Google sun)API
3. JSON simple API

--> By using all the above API's from java application we can convert java object to json format and json format to java object.

### JSON Example with Jackson API:



### 1) Jackson API

Provides ObjectMapper class that is used to convert java obj into json and json into java obj.

### ObjectMapper class important methods

public String writeValueAsString(-)-->used to convert java obj into json

public T readValue(-)-->used to convert json into java obj

JACKSON API is having two versions 1.x and 2.x versions

if we are using Jackson 1.x version we can use the following jar's

jackson-core-asl jar

jackson-mapper-asl jar

the same above jar's if we want get with maven we can add the following dependencies in pom.xml

```
<dependency>
  <groupId>org.codehaus.jackson</groupId>
  <artifactId>jackson-mapper-asl</artifactId>
  <version>1.9.3</version>
</dependency>
```

if we are using jackson 2.x version we can following jar's

jackson-core jar














jackson-annotations jar



jackson-databinding jar

with maven if we want to get the jar' we can add following dependencies in pom.xml file

```
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-core</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-annotations</artifactId>
<version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.5.0</version>
</dependency>
```



- ▲  JsonExample1
  - ▲  src/main/java
    - ▲  com.jsonexample.client
      -  JsonDemo.java
    - ▲  com.jsonexample.domain
      -  Student.java
    - ▲  com.jsonexample.util
      -  JsonUtil.java
  -  src/test/java
  -  JRE System Library [jdk1.7.0\_02]
  -  src
  -  target
  -  pom.xml

 Student.java 

```
package com.jsonexample.domain;
import java.io.Serializable;
public class Student implements Serializable{
    private Integer studentId;
    private String name;
    private String course;
    public Integer getStudentId() {
        return studentId;
    }
    public void setStudentId(Integer studentId) {
        this.studentId = studentId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCourse() {
        return course;
    }
    public void setCourse(String course) {
        this.course = course;
    }
}
```

```

1 JsonUtil.java
package com.jsonexample.util;
import java.io.IOException;
public class JsonUtil {
    public static String convertJavaToJson(Object obj){
        String json="{}";
        ObjectMapper objectMapper=new ObjectMapper();
        try{
            json=objectMapper.writeValueAsString(obj);
        }catch(JsonProcessingException e){
            e.printStackTrace();
        }
        return json;
    }
    public static <T> T convertJsonToJava(String jsonStr,Class<T> targetCls){
        T response=null;
        try {
            ObjectMapper objectMapper=new ObjectMapper();
            response=objectMapper.readValue(jsonStr,targetCls);
        } catch (JsonParseException e) {
            e.printStackTrace();
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return response;
    }
}

```

```

1 JsonDemo.java
package com.jsonexample.client;
import com.jsonexample.domain.Student;
import com.jsonexample.util.JsonUtil;
public class JsonDemo{
    public static void main( String[] args) {
        Student student=new Student();
        student.setStudentId(101);
        student.setName("raja");
        student.setCourse("java");
        String jsonStudent=JsonUtil.convertJavaToJson(student);
        System.out.println(jsonStudent);
        Student std1=JsonUtil.convertJsonToJava(jsonStudent,Student.class);
        System.out.println(std1.getStudentId()+" "+std1.getName()+" "+std1.getCourse());
    }
}

```

### Jersey implementation:

In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services implementation is an open source, production quality ,used for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation.

Download the Jersey jar's from -> <https://jersey.java.net/>

Jersey implementation is given by sun and it is having two versions 1.x and 2.x

*Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23734842, website: [www.nareshit.com](http://www.nareshit.com)*

Jersey 1.x provided Servlet `com.sun.jersey.spi.container.servlet.ServletContainer`

Jersey 2.x provided Servlet `org.glassfish.jersey.servlet.ServletContainer`

### **Steps to develop the Restfull webservice application:**

step1 :- create Root Resource class

step2:- create resource methods to expose

Step3:- configure the servlet in web.xml

(The Servlet is providing by JAX-RS API implementations)

step4:- deploy the app into server

### **Root Resource**

**It is a server side class which contains the data and functionality which is exposed as Resource class to the Client.**

*Root resource classes* are POJOs (Plain Old Java Objects) that are annotated with `@Path` have at least one method annotated with `@Path` or a resource method designator annotation such as `@GET`, `@PUT`, `@POST`, `@DELETE`. Resource methods are methods of a resource class annotated with a resource method designator.

### **Root Resource class Example to produce JSON Response:**

`@Path("products")`

`public class ProductService{`

`@GET`

`@Path("/searchProducts")`

`@Produces(MediaType.APPLICATION_JSON)` // indicates the java method will produces content/response in JSON format

`public List<Product> searchAllProducts(){`

`//logic`

`}`

`}`

**Any class that is annotated with `@Path` Annotation is called as root Resource class.**

**In the Restfull services the Resources classes are associated with URI to identify and access it.**

The `@Path` annotation's value is a relative URI path. In the example above, the Java class will be hosted at the URI path /products.

### **Request Method Designator:**

In the Resource class write the methods which we want to expose the over the Http protocols as resource methods .The Methods should be annotated with relevant Http method designators like `@GET`,`@POST`,`@PUT` and `@DELETE`.

When we sent a GET request the method with `@GET` annotation would be executed to handle the request.

`@GET`, `@PUT`, `@POST`, `@DELETE` and `@HEAD` are *resource method designator* annotations defined by JAX-RS and which correspond to the similarly named HTTP methods. In the example above, the annotated Java method will process HTTP GET requests.

**Resource method:**A method in a resource class that is annotated with Request method Designator is called as Resource method.

The Resource methods can take parameters as well.In case of GET request , the data will be passed as part of QueryString(OR)Path /Cookie/MatrixParam.

### **@QueryParam:**

It is used for receiving the data as input through query String parameters as shown below.

`@Path("products")`

```
public class ProductResource {
```

```
    @GET
```

```
    @Produces(MediaType.APPLICATION_JSON)
```

```
    @Path("/searchProduct")
```

```
    public Product getProduct(@QueryParam("pid") Integer pid){
```

```
        if(pid!=null && pid.equals(101)) {
```

```
            Product product=new Product();
```

```
            product.setProductId(101);
```

```
            product.setProductName("mouse-101");
```

```
            product.setPrice(300.0);
```

```

        return product;
    }else{ return null;
    }
}

```

The client is the following URL to access the above Service

**URL:- <http://<hostName>:<portNum>/context-root /products/searchProduct?pid=101>**

**The above URL shows the query param pid=101 where 101 value will be passed as an input to parameter pid of the method.**

### **Ex2: with Multiple Query-Parameters**

```

@Path("products")

public class ProductResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/searchProduct")

    public Product getProduct(@QueryParam("pid") Integer pid, @QueryParam("pname") String
name){

        if(pid!=null && pid.equals(101) && name!=null && name.equals("mouse-101")) {

            Product product=new Product();

            product.setProductId(pid);

            product.setProductName(name);

            product.setPrice(500.0);

            return product;

        }else{

            return null;

        }

    }

}

```

**URL:** <http://<hostName>:<portNum>/context-root /products/searchProduct?pid=101&pname=mouse-101>

In the case of GET request, we don't need to mark `@Consumes` annotation because it is always consumes "plain/text" as data.

### **@PathParam:**

**In this case the input for the method parameters will be received as part of the path of the URL itself as shown below.**

`@Path("products")`

```
public class ProductResource {  
    @GET  
    @Produces(MediaType.APPLICATION_JSON)  
    @Path("/searchProduct/{pid}/{pname}")  
    public Product  
    getProduct(@PathParam("pid") Integer pid,  
               @PathParam("pname") String pname){  
        if(pid!=null && pid.equals(101)  
        && pname!=null && pname.equals("xxx"))    {  
            Product product=new Product();  
            product.setProductId(pid);  
            product.setProductName("xxx");  
            product.setPrice(500.0);  
            return product;  
        }else{  
            return null;  
        }  
    }  
}
```

URL: `http://<hostName>:<portNum>/context-root /products/searchProduct/101/xxx`

### **@MatrixParam:**

The idea of matrix parameters is that they are an arbitrary set of name-value pairs embedded in a uri path segment.

The Matrix Parameters are separated with semicolon.

```
@Path("products")
```

```
public class ProductResource {
```

```
    @GET
```

```
    @Produces(MediaType.APPLICATION_JSON)
```

```
    @Path("/searchProduct")
```

```
public Product
```

```
getProduct(@MatrixParam("pid") Integer pid,
```

```
    @MatrixParam("pname") String pname){
```

```
    if(pid!=null && pid.equals(101)
```

```
        || pname!=null && pname.equals("xxx")) {
```

```
        Product product=new Product();
```

```
        product.setProductId(pid);
```

```
        product.setProductName(pname);
```

```
        product.setPrice(500.0);
```

```
        return product;
```

```
    }else{
```

```
        return null;
```

```
    }
```

```
}
```

URL's:

```
http://<hostName>:<portNum>/context-root/products/searchProduct;pid=101
```

```
http://<hostName>:<portNum>/context-root/products/searchProduct;pid=101;pname=mouse
```

```
http://<hostName>:<portNum>/context-root/products/searchProduct;pname=mouse;pid=101
```

### @Consumes

The @Consumes annotation is used to specify which MIME media types of representations a resource can accept, or consume, from the client. If @Consumes is applied at the class level, all the response methods accept the specified MIME types by default. If @Consumes is applied at the method level, it overrides any @Consumes annotations applied at the class level.

If a resource is unable to consume the MIME type of a client request, the Jersey runtime sends back an HTTP "415 Unsupported Media Type" error.

The value of @Consumes is an array of String of acceptable MIME types. For example:

```
@Consumes({"text/plain,text/html"})
```

**The following Example is showing how to consume the data in XML by using @Consumes annotation**

```
ProductService.java
package com.nit.service;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
@Path("products")
public class ProductService {
    @POST
    @Path("/registerProduct")
    @Produces(MediaType.TEXT_PLAIN)
    @Consumes(MediaType.APPLICATION_XML)
    public String
    registerProduct( String productXML){
        System.out.println("In server :");
        System.out.println(productXML);
        return "Product Registration Completed Successfully";
    }
}
```

We can test above service by using client application As follows

```
package com.nit.client;
import java.io.StringWriter;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import com.nit.domain.Product;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
public class Test {
    public static void main(String[] args) throws JAXBException {
```



```

    String URL = "http://hostName:portNum/contextpath/products/registerProduct";
    Client client = Client.create();
    WebResource webResource = client.resource(URL);
    Builder builder = webResource.type(MediaType.APPLICATION_XML);
    builder.accept(MediaType.TEXT_PLAIN);
    Product product = new Product();
    product.setPid(201);
    product.setPname("mouse");
    product.setPrice(900.0);
    JAXBContext jaxbContext = JAXBContext.newInstance(Product.class);
    Marshaller marshaller = jaxbContext.createMarshaller();
    StringWriter writer = new StringWriter();
    marshaller.marshal(product, writer);
    String productXML = writer.toString();
    ClientResponse clientResponse = builder.post(ClientResponse.class,productXML);
    String response = clientResponse.getEntity(String.class);
    System.out.println(clientResponse.getStatus() + " "+
clientResponse.getStatusInfo());
    System.out.println(response);
}
}

```

### **Product.java**

```

package com.nit.domain;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name="product")
public class Product implements Serializable{
    private int pid;
    private String pname;
    private double price;
    @XmlAttribute(name="pid")
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    @XmlElement
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    @XmlElement
    public double getPrice() {
        return price;
    }
}

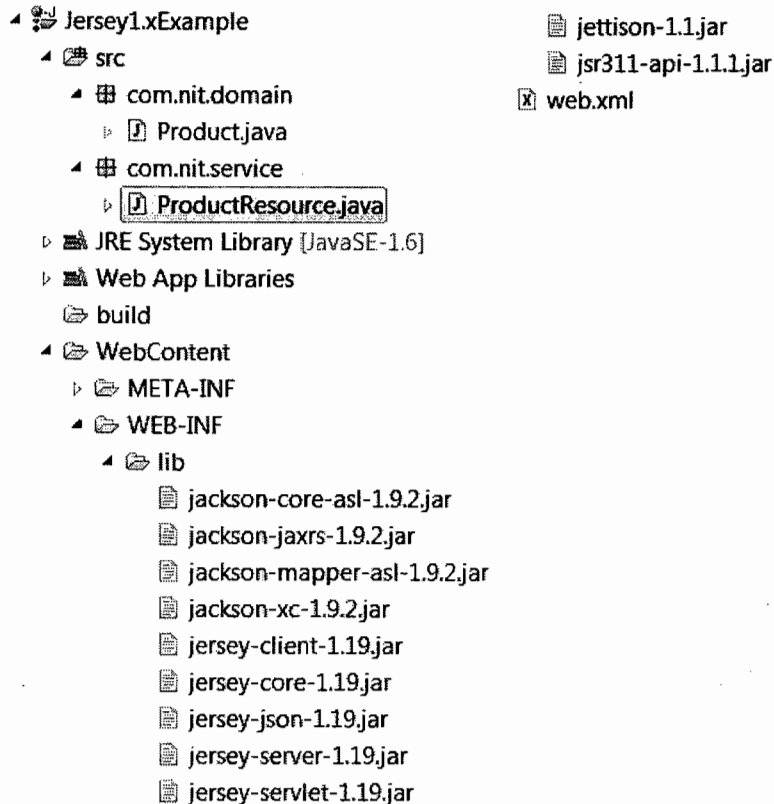
```

```

    }
    public void setPrice(double price) {
        this.price = price;
    }
}

```

The following is Example is showing how to Produce the data in Xml and JSON



```

web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <servlet>
    <servlet-name>jerseyServlet</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>jerseyServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>

```

### ProductResource.java

```
package com.nit.service;
```

```
import javax.ws.rs.GET;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import com.nit.domain.Product;

@Path("products")

public class ProductResource {

    @GET

    @Produces(MediaType.APPLICATION_JSON)

    @Path("/searchProductInJson")

    public Product getProduct(@MatrixParam("pid") Integer pid){

        if(pid!=null && pid.equals(101) ){

            Product product=new Product();

            product.setProductId(pid);

            product.setProductName("mouse");

            product.setPrice(500.0);

            return product;

        }else{

            return null;

        }

    }@GET

    @Produces(MediaType.APPLICATION_XML)

    @Path("/searchProductInXml")

    public Product searchProduct(@MatrixParam("pid") Integer pid){

        if(pid!=null && pid.equals(101) )    {

            Product product=new Product();
```

```

        product.setProductId(pid);

        product.setProductName("mouse");

        product.setPrice(500.0);

        return product;

    }else{

return null;

    }

}

}

```

Product.java

```

import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement(name="product")
public class Product implements Serializable{
    private int productId;
    private String productName;
    private double price;
    @XmlAttribute(name="pid")
    public int getProductId() {
        return productId;
    }
    public void setProductId(int productId) {
        this.productId = productId;
    }
    @XmlElement
    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    @XmlElement
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
}

```

To test the above webservice we can use SOAPUITool/PostMan tool (OR) write a client application.

The following is Example is showing how to test above webservice by using Jersey 1.x version

**Test.java**

```
package com.nit.client;
```

```
import javax.ws.rs.core.MediaType;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.WebResource.Builder;
public class Test {
public static void main(String[] args) {
String URL="http://localhost:8081/Jersey1.xExample/products/searchProductInJson;pid=101";
Client client=Client.create();
WebResource webResource=client.resource(URL);
Builder builder=webResource.accept(MediaType.APPLICATION_JSON);
ClientResponse response=builder.get(ClientResponse.class);
String jsonResponse=response.getEntity(String.class);
System.out.println(response.getStatus()+" -->" +response.getStatusInfo());
System.out.println(jsonResponse);
}
}
```

## Jersey 2.x Example



```

package com.jerseyutil.domain;
import java.io.Serializable;
public class Book implements Serializable{
    private String isbn;
    private String title;
    private String author;
    private Double price;
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public Double getPrice() {
        return price;
    }
}

```

ResponseDTO.java

```

package com.jerseyutil.domain;
import java.io.Serializable;
public class ResponseDTO implements Serializable{
    private byte status;
    private String message,data;
    public byte getStatus() {
        return status;
    }
    public void setStatus(byte status) {
        this.status = status;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public String getData() {
        return data;
    }
    public void setData(String data) {
        this.data = data;
    }
}

```

```

1 JsonUtil.java
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class JsonUtil {
    public static String convertJavaToJson(Object obj){
        String jsonStr="{}";
        ObjectMapper objectMapper=new ObjectMapper();
        try {
            jsonStr=objectMapper.writeValueAsString(obj);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return jsonStr;
    }
    public static <T> T convertJsonToJava(String jsonStr,Class<T> cls){
        T response=null;
        ObjectMapper objectMapper=new ObjectMapper();
        try {
            response=objectMapper.readValue(jsonStr,cls);
        } catch (JsonParseException e) {
            e.printStackTrace();
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return response;
    }
}

```

```

1 Jersey_Book_Service_Util/pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.jerseyutil</groupId>
    <artifactId>Jersey_Book_Service_Util</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Jersey_Book_Service_Util</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>2.5.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.5.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.5.0</version>
        </dependency>
    </dependencies>
</project>

```

- src/main/java
      - com.jerseybookserviceprovider.dao
        - BookDAO.java
      - com.jerseybookserviceprovider.factory
        - ConnectionFactory.java
        - DAOFactory.java
      - com.jerseybookserviceprovider.service
        - BookResource.java
    - src/main/resources
    - src/test/java
    - Maven Dependencies
    - JRE System Library [JavaSE-1.7]
    - src
      - main
        - webapp
          - WEB-INF
            - web.xml
        - test
      - target
      - pom.xml

### pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jerseyserverexample</groupId>
  <artifactId>Jersey_Book_Service_Provider</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>Jersey_Book_Service_Provider Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.containers</groupId>
      <artifactId>jersey-container-servlet</artifactId>
      <version>2.22.2</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>



```



```

<version>3.0.1</version>
</dependency>
<dependency>
<groupId>oracle.jdbc.driver</groupId>
<artifactId>ojdbc14</artifactId>
<version>10.2.0</version>
</dependency>
<dependency>
<groupId>org.glassfish.jersey.media</groupId>
<artifactId>jersey-media-json-jackson</artifactId>
<version>2.22.2</version>
</dependency>
<dependency>
<groupId>com.jerseyutil</groupId>
<artifactId>Jersey_Book_Service_Util</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>
</dependencies>
<build>
<finalName>Jersey_Book_Service_Provider</finalName>
</build>
</project>

```

 web.xml 

```

<web-app>
  <servlet>
    <servlet-name>Jersey</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>
        jersey.config.server.provider.packages</param-name>
      <param-value>
        com.jerseybookserviceprovider.service
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>

```

## 1 BookResource.java

```

@Path("books")
public class BookResource {
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/registerBook")
    @Consumes(MediaType.APPLICATION_JSON)
    public ResponseDTO
    registerBook(String jsonBook){
        System.out.println("Server :"+jsonBook);
        ResponseDTO response=new ResponseDTO();
        response.setMessage("Server Problem ,Book Details Could not be received ");
        if(jsonBook!=null){
            Book book= JsonUtil.convertJsonToJava(jsonBook,Book.class);
            int count=DAOFactory.getBookDAO().registerBook(book);
            if(count>0){
                response.setStatus((byte)1);
                response.setData(book.getIsbn());
                response.setMessage("Book Details saved Successfully");
            }
            else{
                response.setMessage("Book Details could not be saved,Please Try Again!");
            }
            System.out.println("Server Response :"+response.getMessage());
        }
        return response;
    }
}

```

## 2 BookDAO.java

```

package com.jerseybookserviceprovider.dao;
import java.sql.Connection;
public class BookDAO {
    private static final String SQL_REGISTER_BOOK="insert into Book_Details values(?,?,?,?)";
    public int registerBook(Book book){
        int count=0;
        try {
            Connection con=ConnectionFactory.getConnection();
            PreparedStatement pst=con.prepareStatement(SQL_REGISTER_BOOK);
            pst.setString(1,book.getIsbn());
            pst.setString(2,book.getTitle());
            pst.setString(3,book.getAuthor());
            pst.setDouble(4,book.getPrice());
            count=pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return count;
    }
}

```

## 1 DAOFactory.java

```
package com.jerseybookserviceprovider.factory;
import com.jerseybookserviceprovider.dao.BookDAO;
public class DAOFactory {
    private static BookDAO bookDAO;
    static{
        bookDAO=new BookDAO();
    }
    public static BookDAO getBookDAO(){
        return bookDAO;
    }
}
```

## 2 ConnectionFactory.java

```
package com.jerseybookserviceprovider.factory;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class ConnectionFactory {
    static{
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException e) {
            System.out.println("Exception Occured while loading the driver class :"+e.getMessage());
        }
    }
    public static Connection getConnection() throws SQLException{
        String url="jdbc:oracle:thin:@localhost:1521:XE";
        String un="system";
        String pass="manager";
        Connection con=DriverManager.getConnection(url,un,pass);
        return con;
    }
}
```

- src/main/java
      - com.jerseyclientexample.service
        - BookServiceClient.java
      - com.jerseyclientexample.servlet
        - BookServlet.java
    - src/main/resources
    - src/test/java
  - Maven Dependencies
  - JRE System Library [JavaSE-1.7]
  - src
    - main
      - webapp
        - WEB-INF
          - pages
            - regBook.jsp
          - web.xml
      - test
    - target
    - pom.xml

### pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jerseyclientexample</groupId>
  <artifactId>Jersey_Book_Service_Client</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>Jersey_Book_Service_Client Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.core</groupId>
      <artifactId>jersey-client</artifactId>
      <version>2.22.2</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.media</groupId>
      <artifactId>jersey-media-json-jackson</artifactId>

```

```

<version>2.22.2</version>
</dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
  </dependency>
  <dependency>
    <groupId>com.jerseyutil</groupId>
    <artifactId>Jersey_Book_Service_Util</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
</dependencies>
<build>
  <finalName>Jersey_Book_Service_Client</finalName>
</build>
</project>

```

regBook.jsp

```

<html>
  <%@page isELIgnored="false" %>
  <head><h2 align="center">Book Registration</h2></head>
  <br/>${responseDTO.message} <br/>
  <body>
    <form action="saveBookDetails" method="post">
      Isbn :<input type="text" name="isbn"/><br/>
      Title :<input type="text" name="title"/><br/>
      Author :<input type="text" name="author"/><br/>
      Price :<input type="text" name="price"/><br/>
      <input type="submit" value="register"/>
    </form>
  </body>
</html>

```

```

web.xml
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <servlet>
    <servlet-name>bookServlet</servlet-name>
    <servlet-class>com.jerseyclientexample.servlet.BookServlet</servlet-class>
    <load-on-startup>2</load-on-startup>

  </servlet>
  <servlet-mapping>
    <servlet-name>bookServlet</servlet-name>
    <url-pattern>/saveBookDetails</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>
      /WEB-INF/pages/regBook.jsp
    </welcome-file>
  </welcome-file-list>
</web-app>

```

```

BookServlet.java
package com.jerseyclientexample.servlet;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.jerseyclientexample.service.BookServiceClient;
import com.jerseyutil.domain.Book;
import com.jerseyutil.domain.ResponseDTO;
public class BookServlet extends HttpServlet {
  private BookServiceClient bookServiceClient;
  public void init(){
    bookServiceClient=new BookServiceClient();
  }
  public void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException{
    Book book=new Book();
    book.setIsbn(req.getParameter("isbn"));
    book.setTitle(req.getParameter("title"));
    book.setAuthor(req.getParameter("author"));
    book.setPrice(Double.parseDouble(req.getParameter("price")));
    ResponseDTO responseDTO=bookServiceClient.saveBookDetails(book);
    req.setAttribute("responseDTO",responseDTO);
    String target="/WEB-INF/pages/regBook.jsp";
    RequestDispatcher rd=req.getRequestDispatcher(target);
    rd.forward(req, res);
  }
}

```

Writahle

Smart Insert

```

1 BookServiceClient.java
package com.jerseyclientexample.service;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.Invocation.Builder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.glassfish.jersey.client.ClientConfig;
import org.glassfish.jersey.filter.LoggingFilter;
import com.jerseyutil.domain.Book;
import com.jerseyutil.domain.ResponseDTO;
import com.jerseyutil.util.JsonUtil;
public class BookServiceClient {
    public ResponseDTO saveBookDetails(Book book){
        String URL="http://localhost:8082/Jersey_Book_Service_Provider/books/registerBook";
        /*Jersey 2.x Client Code*/
        Client client = ClientBuilder.newClient( new ClientConfig().register( LoggingFilter.class ) );
        WebTarget target=client.target(URL);
        Builder builder=target.request(MediaType.APPLICATION_JSON);
        builder.accept(MediaType.APPLICATION_JSON);
        Response clientResponse=builder.post(Entity.entity(book,MediaType.APPLICATION_JSON),Response.class);
        String jsonResponse=clientResponse.readEntity(String.class);
        ResponseDTO responseDTO=JsonUtil.convertJsonToJava(jsonResponse,ResponseDTO.class);
        return responseDTO;
    }
}

```

If we are using Jersey 1.x we can use the following code in the place of above code

#### jersey 1.x version client code

```

public class BookServiceClient{

    public ResponseDTO saveBookDetails(Book book){

        StringURL="http://localhost:8082/Jersey_Book_Service_Provider/books/registerBook";

        Client client = Client.create();

        WebResource resource=client.resource(URL);

        Builder builder =resource.accept(MediaType.APPLICATION_JSON);

        builder.type(MediaType.APPLICATION_JSON);

        ClientResponse
        clientResponse=builder.post(ClientResponse.class,JsonUtil.convertJavaToJson(book));

        String jsonResponse=clientResponse.getEntity(String.class);

        ResponseDTO
        responseDTO=JsonUtil.convertJsonToJava(jsonResponse,ResponseDTO.class);

        return responseDTO;

    }

}

```

## RestEasy Implementation :

RETEasy is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. It is a **fully certified** and portable implementation of the JAX-RS 2.0 specification, a JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol. The RETEasy can run in any Servlet container,

We can download the RestEasy jar's from <http://resteasy.jboss.org/downloads>

- RestEasyExample1
  - src/main/java
    - com.nit.cfg
      - MyApplication.java
    - com.nit.service
      - HelloResource.java
  - src/main/resources
  - src/test/java
  - JRE System Library [jdk1.7.0\_0
  - src
    - main
      - webapp
        - WEB-INF
          - web.xml
      - test
    - target
    - pom.xml

```
RestEasyExample1/pom.xml
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nit</groupId>
  <artifactId>RestEasyExample1</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>RestEasyExample1 Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <repositories>
    <repository>
      <id>JBoss repository</id>
      <url>
        https://repository.jboss.org/nexus/content/groups/public-jboss/</url>
      </repository>
    </repositories>
  <dependencies>
    <dependency>
      <groupId>org.jboss.resteasy</groupId>
      <artifactId>resteasy-jaxrs</artifactId>
      <version>2.2.1.GA</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>RestEasyExample1</finalName>
  </build>
</project>
```

### HelloResource.java



```

package com.nit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
@Path("hello")
public class HelloResource {
    @Path("/sayHello/{name}")
    @Produces(MediaType.TEXT_PLAIN)
    @GET
    public String sayHello(@PathParam("name") String name){
        return "Hello"+name+"Welcome to RestEasy";
    }
}

```

### MyApplication.java

```

package com.nit.cfg;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;
import com.nit.service.HelloResource;
public class MyApplication extends Application{
    private static Set<Object> set=new HashSet<Object>();
    public MyApplication() {
        HelloResource helloResource=new HelloResource();
        set.add(helloResource);
    }
    public Set<Object> getSingletons(){//overriding method
        return set;
    }
}

```

```

web.xml
<web-app>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-class>
    <init-param>
      <param-name>javax.ws.rs.core.Application</param-name>
      <param-value>com.nit.cfg.MyApplication</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/nit/*</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>resteasy.servlet.mapping.prefix</param-name>
    <param-value>/nit</param-value>
  </context-param>
</web-app>

<!-- when the URL-Pattern is except /* then we can configure
  <context-parameter> in web.xml file to specify the prefix
  -->

```

We can access the above webservice by using a client Application (OR) SOAP UI tool

**RestEasy Client Application****Test.java**

```

package com.nit.client;
import java.util.HashMap;
import java.util.Map;
import javax.ws.rs.core.MediaType;
import org.jboss.resteasy.client.ClientRequest;
import org.jboss.resteasy.client.ClientResponse;
import org.springframework.web.client.RestTemplate;
public class Test {
public static void main(String[] args) throws Exception {
    String URL="http://localhost:8082/RestEasyExample1/nit/hello/sayHello/{name}";
    ClientRequest clientRequest=new ClientRequest(URL);
    clientRequest.accept(MediaType.TEXT_PLAIN);
    ClientResponse clientResponse=clientRequest.get(ClientResponse.class);
    String response=(String) clientResponse.getEntity(String.class);
    System.out.println(response);
}
}

```

```

RestEasyClientApp/pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.nit</groupId>
    <artifactId>RestEasyClientApp</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>RestEasyClientApp</name>
    <url>http://maven.apache.org</url>
    <repositories>
        <repository>
            <id>JBoss repository</id>
            <url>https://repository.jboss.org/nexus/content/groups/public-jboss</url>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-jaxrs</artifactId>
            <version>2.2.1.GA</version>
        </dependency>
    </dependencies>
</project>

```