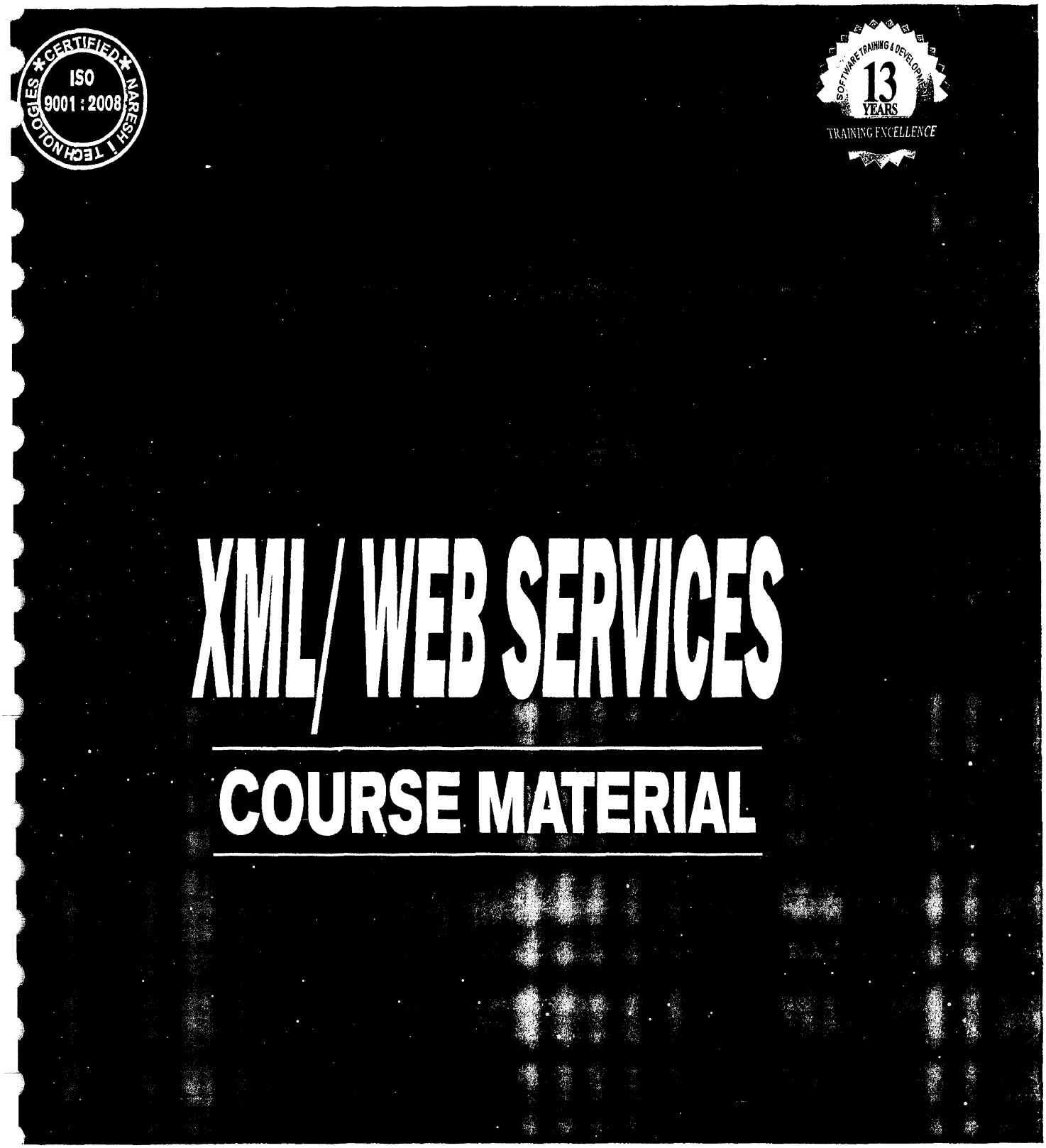


TM

NARESH technologies

SOFTWARE TRAINING & DEVELOPMENT

Opp.Satyam Theatre, Ameerpet, Hyderabad - 16
Ph: 040-23746666, 23734842 Cell: 9000994007 / 08

Branches : CHENNAI PH:044-6555 5557/581 VIJAYAWADA PH:0866-6576789

info@nareshit.com www.nareshit.com nareshit nareshitech

XML-WebServices

XML:

Xml Introduction	001-002
Attributes	002-003
Example on xml document	004-005
DTD	005-010
DTD Types Of Element	010-014
Declaring attributes	014-022
XSD , XSD DataTypes	023-027
XSD NameSpace	027-032
XSD Restrictions/facets	033-037
User Derived DataTypes	038-042
JAX-P	043-044
DOM Parser	045-054
SAX Parser	055-057
JAX-B	058-063

WebServices:

WebServices introduction	064-065
SOAP basd webservices introduction	065-068
WSDL Introduction	068-071
JAX-RPC	071-075
Steps to develop SOAP based webservices with JAX-RPC –SI	075-082
JAX-RPC Clients	081-085

JAX-RPC(Apache-axis)	086-095
WSDL	095-104
SOAP	105-114
JAX-WS	115-152
Spring +JAXWS WebServices (SOAP) Integration Example	153-159
RestFull WebServices	159-165
Jersey implementation	165-177
Jersey 2.x Example	178-188
RestEasy Implemention	188-202
Spring Rest-API Example	203-208
Response Class	208-209
Get Http header in JAX-RS	210-212
Extracting Request Parameters	212-219
Restfull webservices Security	220-228
Conclusion	228-229
JAX-RS Annotations	229-230

XML :

- XML stands for Extensible Markup Language .It is used to describe the data.
- It is designed to transport and store data.
- XML was designed to be self-descriptive.
- XML document is a text-based document with .xml extension.
- It is not a programming a language, it is a markup language.
- XML became a W3C Recommendation as early as in February 1998.
- The current version of XML is 1.0.

Markup: Enclosing textual content with in textual code(tags) is nothing but markup.

ex:-

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
<empname>rama</empname>
<salary>18000</salary>
</employee>
```

XML Purpose:

1.Used to exchange the information between applications
in different languages(java,.net ,php etc...) applications Over the Network.
2.XML Documents also used as text-based databases.

Note:

- Every XML document contains One Unique Root Element.
- The XML document begins with a PRO LOG.
- In XML The Tags are Case sensitive.
- The XML tags User-defined Tags.
- All tags Must be Close in XML document.
- To view The XML documents we can use Browser softwares (OR) XML editor softwares
- We can define the XML Document Structure in a DTD(Document Type Defination) (OR) in a XSD (XML Schema Defination).
- While Validating the XML document the Parser is using The DTD (OR) XSD.
- The XML document contains Elements, Attributes and Entity References.

Elements:

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

1.other elements(child elements)
2.text-data
3.attributes
4.mix of all

XML Elements Naming Rules:

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces
- Any name can be used, no words are reserved (except xml).

Naming Styles For Elements :

There are no naming styles defined for XML elements. some commonly used Naming Styles Are :

Style	Example	Description
Lower Case	<firstname>	All the letters are Lower Case
Upper Case	<FIRSTNAME>	All the letters are Upper Case
Underscore	<first_name>	Underscore Seperates word
Pascal Case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

Attributes :

- XML elements can have attributes in name/value pairs just like in HTML.
- Attributes provide additional information about an element.
- Attributes can occur in any order in an XML element.
- Attribute is unique in an XML element. i.e. It should not be repeated.

For Example:

<employee empNo="101" name="raja" salary="19000"/> valid

Attribute values must be enclose with single quotes (OR) double quotes

<employee empNo='101' name='raja' salary='19000'/> valid

<employee empNo=101 name=raja salary=19000/> invalid

Entity References :

- Some characters have a special meaning in XML.
- For Example If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

The following code will generate an XML error

<salary>if salary <10000 then <salary>

To avoid above error,replace the "<" character with an entity reference:

<salary>if salary < 10000 then <salary>

- The Entity References we can use as follows &EntityReferenceName;
- There are five predefined entity references in XML.

EntityReference	equal to	
< > & ' "	< > & ' "	less than greater than ampersand apostrophe quotation mark

Syntactically < and & symbols are illegal in XML but the greater than symbol and other symbols are legal. Even though Legal It is recomnded to work with Entity References.

Comments in XML:

The syntax for writing comments in XML is similar to that of HTML.
<!– This is a comment –>
Two dashes in the middle of a comment are not allowed.

White-space is Preserved in XML

XML does not truncate multiple white-spaces (But HTML truncates multiple white-spaces to one single white-space):

XML	Hello	Sathish
HTML	Hello Sathish	

An Example XML Document

The XML document contains information about books in a bookstore.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="education">
<title>java in two days</title>
<author>sathish</author>
<price>300.00</price>
</book>
<book category="education">
<title>xml in two days</title>
<author>sathish</author>
<author>jhon</author>
<price>200.00</price>
</book>
<book category="cooking">
<title>Everyday Italian</title>
<author>smith</author>
<price>230.00</price>
</book>
</bookstore>
```

Self-Describing Syntax:-

XML uses a much self-describing syntax.

In xml a prolog defines the XML version and the character encoding:

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

The XML prolog is optional. If it exists, it must come first in the document.

The next line is the root element of the document:

```
<bookstore>
The next line starts a <book> element:
<book category="cooking">
The <book> elements have 3 child elements: <title>,
<author>, <price>.
<book category="cooking">
<title>Everyday Italian</title>
<author>smith</author>
<price>230.00</price>
</book>
</bookstore>
```

The above XML document contains information about books in a bookstore.

Similarities between HTML and XML?

1. Both are languages of web.
2. Both are markup languages.
3. Both are originated from SGML. [Standardized General Markup Language]
4. Tags are basic building blocks of both HTML and XML documents.

Differences Between HTML and XML?

1. HTML tags are predefined tags where as XML tags are user defined tags.
2. HTML tags are limited number of tags where as XML tags are extensible.
3. HTML tags are case insensitive where as XML tags are sensitive.
4. HTML tags are meant for displaying the data but not for describing the data where as XML tags are meant for describing the data.
5. HTML focuses on how data looks where as XML focuses on what data is.

What is a well-formed XML document?

If an XML document confirms the syntactical rules (above specified) of XML specification is said to be well formed. An XML document is said to be well-formed if it observes the following rules.

1. It must begin with the XML declaration
2. I t must have one unique root element
3. Start-tags must have matching end-tags
4. Elements are case sensitive
5. All elements must be closed
6. All elements must be properly nested
7. All attribute values must be quoted
8. Entities must be used for special characters

DTD:

- DTD stands for Document Type Defination.
- DTD is used to define the structure of a XML document.
- DTD is a text based document with .dtd extension
- DTD is using by Parser software while validating the DTD.
- If the DTD is declared inside the XML file then that DTD is called as Internal DTD.
- Internal DTD's are not reusable. Because The DTD's are specific to a particular XML document.
- If the DTD is declared in an external file then that DTD is called as an External DTD. The External DTD's is reusable documents.

The DTD contain

- Element Declarations
- Attribute Declarations
- Entities Declarations
- PCDATA(parsed character data) Section
- CDATA (character data) Section
- PCDATA is text that will be parsed by a parser.
- CDATA is text that will not be parsed by a parser
- PCDATA is a datatype used only for Elements.
- CDATA is a datatype used only for Attributes.

Element Declarations:

To declare an Element we can use the following syntax

```
<! ELEMENT element-name (content-model) >
```

Example XML Document:

```
<employee>
<empNo>101</empNo>
<name>rama</name>
<salary>9000.0</salary>
</employee>
```

employee is a Root-Element of XML .
empNo,name and salary is called as child elements of Employee.

The Structure of Above XML document in DTD is :

```
<!ELEMENT employee (empNo,name,salary)>
<!ELEMENT empNo (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
```

#PCDATA is a data type . The Data type used only for Elements.

- If any Element content-model is #PCDATA the element allows only the Text-Data.
- The Employee element is called as child-only element . It allows Three Child Elements as empNo,name and salary.
- The empNo ,name and salary elements are text-only elements. These are allows text-data only.

Internal DTD:

- If the DTD is declared inside the XML file then that DTD is called as Internal DTD.
- To declare the DTD in XML document we can use <!DOCTYPE> Declaration .

Syntax for Internal DTD:

```
<!DOCTYPE root-element-name [Element Delcarations]>
```

XML document with an internal DTD:

Employee.xml

```
<!DOCTYPE employee [
<!ELEMENT employee (empNo,name,salary)>
<!ELEMENT empNo (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
]>
<employee>
<empNo>101</empNo>
<name>rama</name>
<salary>9000.0</salary>
</employee>
```

External DTD :

If the DTD is declared in an external file then that DTD is called as an External DTD. The External DTD's is reusable documents.

The External DTD's are two types

1) Private DTD
2) Public DTD.

- If a DTD is specific to a particular project then that DTD is called as private DTD.
- If a DTD is common to all the projects then that DTD is called as public DTD.
- Private DTD identified by the SYSTEM keyword.
- Public DTD identified by the PUBLIC keyword.
- TO link the DTD with XML document use <!DOCTYPE> declaration in the XML document.

Private DTD :

<!DOCTYPE root_element SYSTEM "DTDFILE_LOCATION">

Example :

<!DOCTYPE employee SYSTEM "employee.dtd">

PUBLIC DTD:

<!DOCTYPE root_element

PUBLIC " The Formal Public Identifier" "Dtd_file_location">

Example :

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

1. web-app = root element
2. **PUBLIC**

   The Availability. The most common DOCTYPES you will use will be publicly available - "PUBLIC". But you can also specify a local DTD with the "SYSTEM" key word.

3. **"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"**

   The Formal Public Identifier. This entire string is what identifies the DOCTYPE.

   Registration. If there is a plus-sign (+) here, that means that the organization is registered with the ISO. A minus-sign (-) indicates that it is not registered.

Sun Microsystems, Inc.

The Organization. This is the group that owns and maintains the DOCTYPE being used.

EN :

The Language. This is the language that the DTD is written in. It is *not* the language of the content of the page.

Cardinality operators:

An XML element can occur zero or more times in the XML document. Cardinality operator specifies the no. of times an XML element can occur in XML document. In a DTD we have three cardinality operators.

1 ? (means 0to 1)
2 + (means 1to n)
3 * (means O to n)

Note:

If a cardinality operator is not associated with the XML element, that element should occur exactly once in the XML document.

Declaring only one occurrence of the same element:

```
<!ELEMENT element-name (child-name)>
```

Example:

```
<!ELEMENT books (book)>
```

The example declaration above declares that the child element book can only occur one time inside the books element.

Declaring minimum one occurrence of the same element:

<! ELEMENT element-name (child-name+) >

Example :

<! ELEMENT books (book+) >

The + sign in the above example declares that the child element book must occur one or more times inside the books element.

Declaring zero or more occurrence of the same element:

<! ELEMENT element-name (child-name) >*

E.g

```
<! ELEMENT books (book*) >
```

The * sign in the above example declares that the child element book must occur zero or more times inside the books element.

The following books.dtd Example shows how to use cardinality operators.

```
<!ELEMENT books (book*)>
<!ELEMENT book (title,publications?,author+,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT publications (#PCDATA)>
```

Books.xml

```
<!DOCTYPE books SYSTEM "books.dtd">
<books>
<book>
<title>java in two days </title>
<publications>nit</publications>
<author>sathish</author>
<author>jhon</author>
<price>200</price>
</book>
<book>
<title>DTD in two days </title>
<author>sathish</author>
<price>100</price>
</book>
</books>
```

Types of Elements :

In DTD it is possible to declare the following five Types of Elements.

1)Text-Only Elements
2)Child Only Elements
3)Mixed Elements
4)Empty Elements
5) ANY Elements

Text-Only Elements :

- If an element allows only the text-data then that element is called as Text-only element.
- To declare text-only element we can use #PCDATA as a datatype in the Content Model.

Syntax :

<!ELEMENT element-name (#PCDATA) >

Example :

<!ELEMENT title (#PCDATA)>

Child Only Elements :

If an element is allows only the child elements then that element is called as Child-only element.

Syntax :

<!ELEMENT element-name (child1,child2,child3.....) >

Example :

```
<!ELEMENT book (title,price,author)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

book is a child-only element.
title,price ,author is called as Text-only Elements

Defining the order of occurrence :

- When there are a multiple number of child elements, you must designate the order of occurrence. There are two ways to notate the order of occurrence.
- Using a comma (,) between the child element name and the next child element name indicates that the child elements will occur in the order given.
- Using a vertical line (|) means that either one or the other child element will occur.
- "," Occurs in the order given
- "|" Either one or the other child element occurs

In the following DTD example, the content of "book" is the "title" and "price" elements.

```
<!ELEMENT book (title,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

The following is a valid element description for this type of Element Type Declaration:

```
<book>
<title>AdvJava</title>
<price>300</price>
</book>
```

The following is a invalid element description for this type of Element Type Declaration:

```
<book>
<price>300</price>
<title>AdvJava</title>
</book>
```

Because "," defines the order of occurrence as the order in which the child element was written.

To describe an Element Type Declaration where either the "title" or "book-name" element (child elements of "book") occurs:

```
<!ELEMENT book (title|book-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT book-name (#PCDATA)>
```

valid xml :

```
<book>
<title>core java</title>
</book>
```

valid xml :

```
<book>
<book-
name>java</book-
name>
```

invalid xml:

```
<book>
<title>php</title>
<book-name>java</book-name>
</book>
```

Inside the book either title (OR) book-name is allows.

The following examples shows how to use comma (,) and vertical lines :

DTD:

```
<!ELEMENT books (book+)>
<!ELEMENT book ((title|book-name),author,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT book-name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

XML:

```
<!DOCTYPE books SYSTEM "books.dtd">
<books>
<book>
<title>java in two days </title>
<author>sathish</author>
<price>200</price>
</book>
</books>
(OR)
```

```
<!DOCTYPE books SYSTEM "books.dtd">
<books>
<book>
<book-name>java in two days </book-name>
<author>sathish</author>
<price>200</price>
</book>
</books>
```

Mixed Elements :

If an element is allows text-data and child elements then that element is called as Mixed element.

books.xml

```
<!DOCTYPE books [
<!ELEMENT books (book)>
<!ELEMENT book (#PCDATA|isbn|title|price)*>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
<books>
<book>
The book isbn number is <isbn>101</isbn>
The book title is <title>corejava</title>
The book price is <price>900</price>.
</book>
</books>
```

Empty Elements :

If an element is not allows text-data and child elements then that element is called as an Empty Element.

books.xml

```
<!DOCTYPE books [
<!ELEMENT books (book+)>
<!ELEMENT book EMPTY >
]>
<books>
<book></book>
<book/>
</books>
```

Any Element:

If an element is allows text-data (OR) child elements (OR) empty content (OR) mix of all then that element is called as any element.

books.xml

```
<!DOCTYPE books [
<!ELEMENT books (book*)>
<!ELEMENT book ANY>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
<books>
<book>
this is <title>corejava</title> book.
</book>
<book>this is advjava book</book>
<book></book>
<book/>
<book>
<isbn>9001</isbn>
<price>800</price>
<title>corejava</title>
</book>
</books>
```

Declaring Attributes :

An attribute declaration has the following syntax:-

<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

<payment type="check" />

The attribute-value can be one of the following:

Value	Explanation
Value	The default value of the attribute
#REQUIRED	The attribute is mandatory attribute.
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value:

DTD:

```
<!ELEMENT course EMPTY>
<!ATTLIST course name  CDATA "java">
```

Valid XML:

<course name="java" />
(OR)
<course name="php" />
In the above example , the "course" element is defined to be an empty element with a "name" attribute of type CDATA. If course name is not specified, it has a default value as "java" .

#REQUIRED

#REQUIRED is indicates that an attribute is a manadatory attribute.

Syntax

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

Example

DTD:

```
<!ELEMENT course EMPTY>
<!ATTLIST course cid CDATA #REQUIRED>
<!ATTLIST course name  CDATA "java">
```

Valid XML:

<course cid="cj101" name="corejava"/>

Valid XML:

<course cid="cj101" />

Invalid XML:

<course />

#IMPLIED:

#IMPLIED is indicates that an attribute is an optional attribute.

Syntax

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example

DTD:

```
<!ELEMENT course EMPTY>
<!ATTLIST course cid CDATA
#REQUIRED>
<!ATTLIST course name  CDATA
#IMPLIED>
```

Valid XML:

<course cid="cj101" name="corejava"/>

Valid XML:

<course cid="cj101" />

Invalid XML:

<course />

#FIXED:

#FIXED is indicates that an attribute has fixed value.

Syntax

<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example

```
<!ELEMENT course EMPTY>
<!ATTLIST course cid CDATA #REQUIRED>
<!ATTLIST course name  CDATA #IMPLIED>
<!ATTLIST course fee  CDATA #FIXED "1000">
```

Valid XML:

<course cid="cj101" name="corejava" fee="1000"/>

Valid XML:

<course cid="cj101" name="corejava" />
Using of fixed attribute is also an optional.

Invalid XML:

<course cid="cj101" name="corejava" fee="1200" />

The attribute-type can be one of the following:-

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

Enumerated type Attribute :

Syntax

<!ATTLIST element-name attribute-name (en1|en2|..) default-value>

Example

DTD:

<!ATTLIST payment type (check|cash) "cash">

XML example:

<payment type="check" />
or
<payment type="cash" />

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

ID type Attribute :

- The ID type attribute value must be unique in the XML.
- ID type value should not be start with digit.

```
<!DOCTYPE students [
<!ELEMENT students (student+)>
<!ELEMENT student EMPTY>
<!ATTLIST student sid ID #REQUIRED>
<!ATTLIST student sname CDATA #REQUIRED>
]>
<students>
<student sid="N101" sname="rama"/>
<student sid="N102" sname="rama"/>
</students>
```

IDREF type attribute :

- IDREF type attribute value must be the id of another element.
- IDREF type is allows single value.
- The IDREFS type is allows list of values.

IDREFExample.dtd

```
<!ELEMENT students (course*,student*)>
<!ELEMENT course (#PCDATA)>
<!ATTLIST course cid ID #REQUIRED>
<!ATTLIST course cname CDATA #REQUIRED>
<!ATTLIST course fee CDATA #IMPLIED>
<!ELEMENT student EMPTY>
<!ATTLIST student sid ID #REQUIRED>
<!ATTLIST student sname CDATA #REQUIRED>
<!ATTLIST student  student-course IDREF #IMPLIED>
<!ATTLIST student student-courses IDREFS #IMPLIED>
```

IDREFExample.xml

```
<!DOCTYPE students PUBLIC  "-//sathish//DTD 1.0 //EN"
"E://IDREFExample.dtd">
<students>
<course cid="j101" cname="corejava" fee="1000">this is corejava
course</course>
<course cid="j102" cname="AdvJava" fee="2000">this is Adv java
course</course>
<course cid="c103" cname="c" fee="800">this is c lang course</course>
<course cid="p104" cname="php" fee="1200">this is php course</course>
<student sid="s101" sname="rama" student-course="j102"/>
<student sid="s102" sname="roja" student-course="j101"/>
<student sid="s103" sname="raja" student-courses="j101 j102 c103"/>
<student sid="s104" sname="3sha" student-courses="c103  p104"/>
</students>
```

NMTOKEN type Attribute :

It is similar to CDATA but this attribute value is a valid XML name.

```
<!DOCTYPE courses [
<!ELEMENT courses (course*)>
<!ELEMENT course EMPTY>
<!ATTLIST course cid ID #REQUIRED>
<!ATTLIST course cname NMTOKEN #REQUIRED>
<!ATTLIST course fee CDATA #REQUIRED>
]>
<courses>
<course cid="J103" cname="java" fee="2000"/>
<course cid="C101" cname="c" fee="900"/>
<course cid="N102" cname=".net" fee="1800"/>
</courses>
```

NOTATION type :

- The attribute type of NOTATION allows you to use a value that has been declared as a *notation* in the DTD.
- A notation is used to specify the format of non-XML data. A common use of notations is to describe MIME types such as image/gif, image/jpeg etc.

Syntax:

To declare a notation:

```
<!NOTATION name SYSTEM "external_id">
```

To declare the attribute:

<!ATTLIST element_name attribute_name NOTATION default_value>

Example:

```
<!DOCTYPE photos [
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION GIF SYSTEM "image/gif">
<!ELEMENT photos (photo+)>
<!ELEMENT photo (#PCDATA) >
<!ATTLIST photo type NOTATION (JPG|GIF) #REQUIRED>
]>
<photos>
<photo type="JPG">this is Jpeg format</photo>
<photo type="GIF">this is gif format</photo>
</photos>
```

ENTITY type attribute :

The attribute type of ENTITY is used for referring to the name of an entity you've declared in your DTD.

Syntax:

<!ATTLIST element_name attribute_name ENTITY default_value>

Example:

```
<!ELEMENT mountains (mountain+)>
<!ELEMENT mountain (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST mountain photo ENTITY #IMPLIED>
<!ENTITY mt_cook_1 SYSTEM "mt_cook1.jpg">
```

The following XML document would be valid, as it conforms to the above DTD:

```
<mountains>
<mountain photo="mt_cook_1">
<name>Mount Cook</name>
</mountain>
</mountains>
```

Invalid XML - The following XML document would be invalid. Because the photo attribute of the second element contains a value that hasn't been declared as an entity:

```
<mountains>
<mountain photo="mt_cook_1">
<name>Mount Cook</name>
</mountain>
<mountain photo="None">
<name>Cradle Mountain</name>
</mountain>
</mountains>
```

ENTITIES type attribute

The attribute type of ENTITIES allows you to refer to multiple entity names, separated by a space.

Syntax:

<!ATTLIST element_name attribute_name ENTITIES default_value>

Example:

```
<!ATTLIST mountain photo ENTITIES #IMPLIED>
<!ENTITY mt_cook_1 SYSTEM "mt_cook1.jpg">
<!ENTITY mt_cook_2 SYSTEM "mt_cook2.jpg">
```

Valid XML - The following XML document would be valid, as it conforms to the above DTD:

```
<mountains>
<mountain photo="mt_cook_1 mt_cook_2">
<name>Mount Cook</name>
</mountain>
</mountains>
```

Invalid XML - The following XML document would be invalid. This is because in the first element, a comma is being used to separate the two values of the photo attribute (a space should be separating the two values):

```
<mountains>
<mountain photo="mt_cook_1,mt_cook_2">
<name>Mount Cook</name>
</mountain>
</mountains>
```

Entities for Parsed Content :

- Entities are used to define shortcuts to special characters.
- Entities can be declared internal or external.

An Internal Entity Declaration

Syntax

<!ENTITY entity-name "entity-value">

Example
DTD Example:

```
<!ELEMENT author (#PCDATA)>
<!ENTITY writer "sathish Bandi.">
<!ENTITY copyright "Copyright@Nareshit 2016.">
```

XML example:

Note: when we are using entity for parsed content the syntax is &entityName;

Example:

<author>&writer;&copyright;</author>

An External Entity Declaration:
Syntax

<!ENTITY entity-name SYSTEM "URI/URL">

Example
DTD Example:

XML example:

<author>&writer;&copyright;</author>

XSD (XML Schema Definition) :

what is an XSD?

--> XSD stands for XML Schema Definiton
--> XML Schema is used to define the structure of an XML document.
--> Schemas are another approach used by the XML parser to validate an XML document.
--> It is a text file with .xsd extension.

what xsd defines?

1. It defines elements that can appear in a document
2. It defines attributes that can appear in a document
3. It defines the number of child elements and their sequence
4. It defines whether an element is empty or can include text
5. It defines data types for elements and attributes
6. It defines default and fixed values for elements and attributes

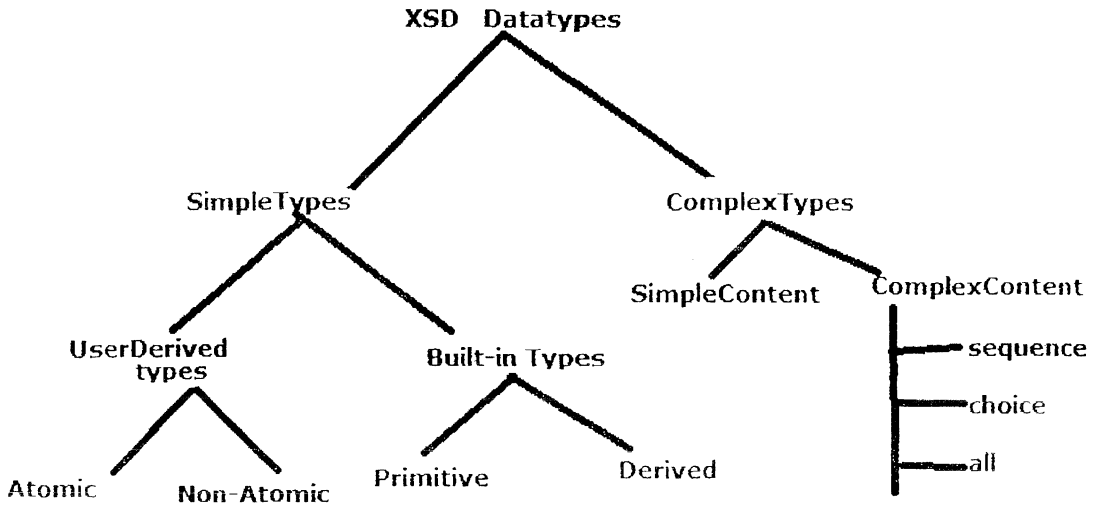
Difference Between DTD and XSD?

DTD	XSD
1. Used for legacy XML documents	1. Used for current XML documents
2.Limited data types so not type safe	2 . Rich in data types, So it is type safe.
3. Cardinality control over the elements is constraints Limited.	3 .XSD provides much more cardinality
4. Not supporting to create our own Datatypes Datatypes	4. Supporting to create Our own
5 It Doesn't support namespace	5. It support namespaces

XSD DataTypes :

There are Two Types of DataTypes.

1)SimpleTypes
2)ComplexTypes



SimpleTypes :

There are two types of Simple Types

1)Built-in types
(i) primitive types
(ii) Built-in-Derived types
2)User-Derived Types
(i)Atomic Types
(ii) Non-Atomic Types

Primitive Types(19)

string ,boolean, decimal , float ,double ,duration, dateTime, time ,date ,gYearMonth , gYear,gMonthDay,gDay, gMonth, hexBinary, base64Binary, anyURI, QName, NOTATION

Built-in Derived Data Types(25):

normalizedString,token,language,NMTOKEN,NMTOKENS,Name,NCName,ID,IDREF,

IDREFS,ENTITY,ENTITIES,integer,nonPositiveInteger,negativeInteger,long,int ,short,byte,nonNegativeInteger,unsignedLong,unsingnedInt, unsignedShort,unsignedByte,positiveInteger.

Types of Elements :

There are two types of Elements in XSD
1)Simple Element
2)Complex Element

What is a simple Element?

If any XML element allows only text data then that element is called as SimpleElement. The Simple Element shouldn't contain any sub elements or attributes.

What is Complex Element?

If an Element is allows sub elements (OR) attributes (OR) both then that element is called as Complex Element.

The complex Elements we can create as follows

- Elements with Child Elements And/OR Attributes
- Elements with Mixed Content And/OR attributes
- Elements with Text-Data and attributes
- Elements with Empty Content and attributes

To create a simple element we can use the following syntax in the XSD

syntax:

<element name="element-name" type="element-type"/>

where element-name is the name of the element and element-type is the data type of the element

example:

<element name="salary" type="decimal"/>

To create Complex Element we can use the following syntax.

Syntax1 :

```
<element name="element-name">
<complexType > ➔it makes an element is a complex type element.
<sequence> (OR) <all>
<element name="childElementName1" type="datatype"/>
<element name="childElementName2" type="datatype"/>
</sequence>(OR) </all>
</complexType>
</element>
<sequence> is a complex datatype.
```

- In the case of <sequence> all the child elements with the same order using is an important.
- <all> is a complex datatype.

- In the case of <all> the child elements with the same order using is not important.

Example :

```
<element name="employee">
<complexType >
<sequence> (OR) <all>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>(OR) </all>
</complexType>
</element>
```

Syntax2 :

```
<complexType name="element-name">
<sequence> (OR) <all>
<element name="childElementName1 " type="datatype1"/>
<element name="childElementName2" type="datatype2"/>
</sequence> (OR) </all>
</complexType>
```

Example:-

```
<complexType name="personType">
<sequence>
<element name="name" type="string"/>
<element name="age" type="integer"/>
<element name="mobile" type="string"/>
</sequence>
</complexType>
</element>
<element name="person" type="personType"/>
<element name="student" type="personType"/>
<element name="customer" type="personType"/>
```

- In xsd <schema> element is a root-element.
- The elements we can declare inside the <schema> element.

Employee.xsd

```
<schema>
<element name="employee">
<complexType >
<sequence>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>
</complexType>
</element>
</schema>
```

Employee.xml

```
<employee>
<empNo>101</empNo>
<name>sathish</name>
<salary>18000</salary>
</employee>
```

XSD Names Spaces :-

XSD name Space has two cases

1) Declaring the Namespace in the XSD document using targetNamespace declaration
2) Using the elements that are declared under a namespace in XML doc.

TargetNamespace:-

- targetNamespace declaration is similar to a package declaration of java ,
- in java package concept and in xsd namespace concept's are given to avoid ambigity problems.
- When ever the java classes are defined in a package while using that classes we will use fully qualified Name.Similarly when ever an element is defined
- Under a namespace while using of that element we will use fully qualified Name (tns:elementName).
- In order to declare a package we can use package keyword followed by name of the package. Similarly To declare a namespace in XSD we need to use targetNamespace attribute at the <schema> tag level followed by targetNamespace label as show below.
- In java we can have only one package declaration at a class level,similarly In the Same way we can have only one targetNamespace declaration at an XSD document.

Example:

```
<schema targetNamespace="http://empinfo.com">
<element name="employee">
<complexType >
<sequence>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>
</complexType>
</element>
</schema>
```

The employee and it's child elements are binded to the namespace http://empinfo.com

So,while using "employee" element we should use fuly qualified Name.
(Fully Qualified Name means namespace:element name)

But the name space labels could be any of the characters in length,so if we use fully qualified Name it would become tough to read.So, to avoid this problem XML has introduced a concept called short name for that namespace label using xmlns declaration.

Example:

```
<employee xmlns="http://empinfo.com">
<empNo>101</empNo>
<name>sathish</name>
<salary>19000</salary>
</employee>
```

(OR)

Example:

```
<e:employee xmlns:e="http://empinfo.com">
<e:empNo>101</e:empNo>
<e:name>sathish</e:name>
<e:salary>19000</e:salary>
</e:employee>
```

- Iin XSD also the <schema> element and it's child element's are required to be write with fully qualified Name.
- But writing of all the elements with fully qualified name is increasing the burden.
- So, to avoid this prcblem in XSD also we can use xmlns declaration.
- The <schema> element is given by W3C under the following Namespace

http://www.w3.org/2001/XMLSchema

Example:

```
<schema targetNamespace="http://empinfo.com"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<element name="employee">
<complexType >
<sequence>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>
</complexType>
</element>
</schema>
(OR)
<xs:schema targetNamespace="http://empinfo.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="employee">
<xs:complexType >
<xs:sequence>
<xs:element name="empNo" type="xs:int"/>
<xs:element name="name" type="string"/>
<xs:element name="salary" type="decimal"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

ElementFormDefault values:

ElementFormDefault takes either qualified (OR) unqualified.
The default value is "unqualified".

qualified–indicates that the elements are in the targetNamespace of the schema are qualified (i,e elements have a namespace)

unqualified –indicates that the elements are in the targetNamespace of the schema are qualified(i,e elements do not have a namespace)

2)Using the elements that are declared under a namespace in XML doc:

- While working with an xml document ,you need to link it to XSD to indicate it is following the structure defined in that XSD.In order to link an XML to an XSD we use an attribute "schemaLocation".
- schemaLocation attribute declaration has two pieces of information.First part is

representing the namespace from which you are using the elements.

- Second is the document which contains this namespace as show below.

Example:

```
<employee xmlns="http://empinfo.com"
schemaLocation="http://empinfo.comE://xml/employee.xsd">
</employee>
```

Note:

- schemaLocation attribute is given by W3C in the following namespace

http://www.w3.org/2001/XMLSchema-instance

- to write schemaLocation attribute with shortName recomnded to use "xmlns" declaration at root tag level.

Example:

```
<employeexmlns="http://empinfo.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://empinfo.comE://xml/employee.xsd">
</employee>
```

Note :

- If we want to include two XSD documents in the XML then you need to declare in schemaLocation tag as follows

<namespace1><schemalocation1><namespace2><schemalocation2>

<employee xmlns="http://empinfo.com"
xmlns:xsi="http://www/w3.org/2001/XMLSchema-Instance"
xsi:schemaLocation="http://empinfo.com E://xml/employee.xsd
http://deptinfo.com E://xml/department.xsd">
</employee>

Attributes Declaration :

Syntax:

<attribute name ="attribute-name" type="attribute-type"/>

Ex:-

```
<attribute name="empNo" type="int"/>
<attribute name="name" type="string"/>
<attribute name="salary" type="decimal"/>
```

Mandatory attribute:

<attribute name="empNo" type="int" use="required"/>

Default Attribute :

<attribute name="name" type="string" default="guest"/>

Fixed Attribute:

<attribute name="salary" type="decimal" fixed="10000"/>

The following example shows how create attributes in XSD:-

```
<schema>
<element name="course">
<complexType>
<attribute name="courseId" type="int"/>
<attribute name="courseName" type="string"/>
<attribute name="courseFee" type="decimal"/>
</complexType>
</element>
</schema>
```

In xml :

<course courseId="101" courseName="java" courseFee="2000"/>

The following example shows how to create child elements and attributes :

books.xsd

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://booksinfo.com/nit"
elementFormDefault="qualified">
<xs:element name="books">
<xs:complexType>
<xs:sequence>
<xs:element name="book" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="author" type="xs:string"
minOccurs="1" maxOccurs="unbounded"/>
<xs:element name="publications"
type="xs:string"/>
</xs:sequence>
<xs:attribute name="isbn" type="xs:string"
use="required"/>
<xs:attribute name="title" type="xs:string"/>
<xs:attribute name="price" type="xs:double"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Note : MinOccurs and maxOccurs are called as Cardinality Constraints.

books.xml

```
<books xmlns="http://booksinfo.com/nit"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://booksinfo.com/nit
 E://xml/books.xsd">
<book isbn="NIT101" title="corejava" price="400">
<author>sathish</author>
<author>hari</author>
<publications>nareshit</publications>
</book>
<book isbn="NITJ102" price="500" title="advjava" >
<author>Guptha</author>
<author>Venkatesh</author>
<publications>nit</publications>
</book>
</books>
```

XSD Restrictions/facets:

- Restrictions are used to define acceptable values for XML elements or attributes Restrictions on XML elements are called facets
- To define the Restrictions the following **constraints** are given in XSD.

S.No.	Restriction & Description
1	**enumeration** Defines a list of values which are acceptable.
2	**fractionDigits** Defines the maximum number of decimal places allowed(zero or more).
3	**length** Defines length in terms of characters of string or items in a list(zero or more).
4	**maxExclusive** Defines upper bounds for numeric values excluding this number.
5	**maxInclusive** Defines upper bounds for numeric values including this number.
6	**maxLength** Defines maximum length in terms of characters of string or items in a list(zero or more).
7	**minExclusive** Defines lower bounds for numeric values excluding this number.
8	**minInclusive** Defines lower bounds for numeric values including this number.
9	**minLength** Defines minimum length in terms of characters of string or items in a list(zero or more).
10	**pattern** Defines the exact sequence of characters identified by the pattern that are acceptable
11	**totalDigits** Defines the exact number of digits allowed in the number(always greater than zero)
12	**whiteSpace** Defines the way in which white space characters (line feeds, tabs, spaces, and carriage returns) are handled

Syntax for Restrictions:

<restriction base = "element-type"> restrictions </restriction>

Base — Type of the Element on which restriction is to be applied. For example,

<restriction base = "xs:integer">

specifies that this restriction is specific to an element of type int.

restriction is normally a range of conditions to be applied on the element's value. In this example, we've set a restriction on marks that marks should be in range of 0 to 100 with both values are included.

restriction

<minInclusive value = "0"/>

<maxInclusive value = "100"/>

Examples : Restriction on Value.

Example 1 :

Condition – Marks should be in range of 0 to 100.

```
<xs:element name = "marks">
<xs:simpleType>
<xs:restriction base = "xs:integer">
<xs:minInclusive value = "0"/>
<xs:maxInclusive value = "100"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 2:

Condition – firstname should be one of the lowercase letters form a to z .

```
<xs:element name = "firstname">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:pattern value = "[a-z]"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 3:

Condition : Acceptable value is three of the LOWERCASE (OR) UPPERCASE letters

```
<xs:element name="name">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 4:

Condition : Acceptable value is six digits only

```
<xs:element name="zipcode">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[0-9][0-9][0-9][0-9][0-9][0-9]"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
        (OR)
<xs:element name="zipcode">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[0-9]{6}"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 5:

Condition : Acceptable value is zero or more occurrences of lowercase letters form a to z

```
<xs:element name="name">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="([a-z])*"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 6:

Condition : Acceptable value is one of the following letters M (OR) F:

```
<xs:element name="gender">
        <xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:pattern
value="[MF]"/>
                </xs:restriction>
</xs:simpleType>
        </xs:element>
```

Example 7 :

Condition : Acceptable value is male or female

```
<xs:element name="gender">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="male|female"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 8 :

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<element name="person">
<complexType>
<sequence>
<element name="pid">
<simpleType>
<restriction base="string">
<pattern value="[0-9]{4}"/>
</restriction>
</simpleType>
</element>
<element name="name" >
```

```
<restriction base="string">
<pattern value="([a-z])*"/>
</restriction>
</simpleType>
</element>
<element name="gender">
<simpleType>
<restriction base="string">
<pattern value="male|female"/>
</restriction>
</simpleType>
</element>
</sequence>
</complexType>
</element>
</schema>
```

Example 9:

Condition – Grades should only be A, B or C.

```
<xs:element name = "grades">
<xs:simpleType>
<xs:restriction base = "xs:string">
<xs:enumeration value = "A"/>
<xs:enumeration value = "B"/>
<xs:enumeration value = "C"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Example 10 :

```
<schema
xmlns="http://www.w3.org/2001/XMLSchema">
<element name="employee">
<complexType>
<sequence>
<element name="empNo" type="int"/>
<element name="name" >
<simpleType>
<restriction base="string">
<whiteSpace value="collapse"/>
</restriction>
</simpleType>
</element>
<element name="job">
<simpleType>
<restriction base="string">
<enumeration value="sales"/>
<enumeration value="admin"/>
```

```
<enumeration value="hr"/>
<enumeration value="developer"/>
</restriction>
</simpleType>
</element>
</sequence>
</complexType>
</element>
</schema>
```

User Derived Datatypes :

User derived types are two types

1)Atomic types
2)Non-Atomic types

Atomic Types :

- Applying the Restrictions for every element is increasing the burden.
- To reuse Same Restrictions for Multiple Element's create User-Derived Datatype (atomic type) with all the required restrictions.
- And where ever the restrictions are required then use the atomic Type. .

Example: The following Example Showing How to create Atomic DataType.

In the following example NumberType, JobType are Atomic Types

```
<schema targetNamespace="http://empinfo.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:emp="http://empinfo.com"
elementFormDefault="qualified">
<simpleType name="NumberType">
<restriction base="int">
<minInclusive value="101"/>
<maxInclusive value="90000"/>
</restriction>
</simpleType>
<simpleType name="JobType">
<restriction base="string">
<enumeration value="sales"/>
<enumeration value="marketing"/>
<enumeration value="admin"/>
</restriction>
</simpleType>
<element name="employee">
<complexType>
<sequence>
<element name="empNo" type="emp:NumberType"/>
<element name="name" type="string"/>
```

```
<element name="salary" type="emp:NumberType"/>
<element name="job" type="emp:JobType"/>
</sequence>
</complexType>
</element>
</schema>
```

XML Example :

```
<employee xmlns="http://empinfo.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://empinfo.com
E://xml/user-derivedtypes.xsd">
<empNo>101</empNo>
<name>sathish</name>
<salary>22000</salary>
<job>it</job>
</employee>
```

Nonatomic Datatypes Types:

The Non-Atomic Types are Two Types
1)List Type
2)Union Type

List:

List types are used to collect list of values .

The following Example is Showing How to create List Types?

```
<xs:schema xmlns:xs=
"http://www.w3.org/2001/XMLSchema"
xmlns:nit="http://empinfo.in/nit"
targetNamespace="http://empinfo.in/nit"
elementFormDefault="qualified">
<xs:simpleType name="DateList">
<xs:list itemType="xs:date"/>
</xs:simpleType>
<xs:simpleType name="contactList">
<xs:list itemType="xs:string"/>
</xs:simpleType>
<xs:element name="employee">
<xs:complexType>
<xs:sequence>
<xs:element name="empNo" type="xs:integer"/>
<xs:element name="name" type="xs:string"/>
```

```
<xs:element name="salary" type="xs:decimal"/>
<xs:element name="vacationDays" type="nit:DateList"/>
<xs:element name="phoneNumbers"
type="nit:contactList"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

XML Example:

```
<employee xmlns="http://empinfo.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://empinfo.com
E://xml/ListTypeExample.xsd">
<empNo>101</empNo>
<name>sathish</name>
<salary>79999.99</salary>
<mobile>9889898890 9889898891 9889898892 </mobile>
<vacationDays>2016-10-13
2016-10-15 2016-10-14</vacationDays>
</employee>
```

Union Types : Union Types are used to combine two /more DataTypes as a Single Type.

The following XSD Example is Showing How to create List Types?

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="RunningRace">
 <xs:restriction base="xs:string">
 <xs:enumeration value="100 meters"/>
 <xs:enumeration value="10 miles"/>
 <xs:enumeration value="10 kilometers"/>
  </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="Gymnastics">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Floor"/>
   <xs:enumeration value="Rings"/>
   <xs:enumeration value="Beam"/>
   </xs:restriction>
  </xs:simpleType>
 <xs:simpleType name="EventType">
   <xs:union memberTypes="RunningRace Gymnastics"/>
  </xs:simpleType>
<xs:element name="program">
    <xs:complexType>
```

```
      <xs:sequence>
   <xs:element name="event" type="EventType"
  minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

XML :

```
<program
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="E://xml/UnionTypesE
xample.xsd">
<event>10 miles</event>
<event>Floor</event>
</program>
```

Complex Type Elements

The Complex Type Elements Are Possible to create As Follows

- Elements with Child Elements And/OR Attributes
- Elements with Mixed Content And/OR attributes
- Elements with Text-Data and attributes
- Elements with Empty Content and attributes

1) Elements with Child Elements:

```
<schema>
<element name="employee">
<complexType>
<sequence>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>
</complexType>
</element>
</schema>
```

In XML Write the Elements as Follows :

```
<employee>
<empNo>101</empNo>
<name>sathish</name>
<salary>12000.0</salary>
```

2) Elements with Mixed Content:

```
<schema>
<element name="employee">
<complexType  mixed="true">
<sequence>
<element name="empNo" type="int"/>
<element name="name" type="string"/>
<element name="salary" type="decimal"/>
</sequence>
</complexType>
</element>
</schema>
```

In XML File:

```
<employee>
The empNo is <empNo>101</empNo> and the name is
<name>sathish</name > and the salary is
<salary>12000.0</salary>
</employee>
```

3) Elements with Text-Data and attributes :

```
<schema>
<element name="employee">
 <complexType>
  <simpleContent>
   <extension base="string">
   <attribute name="empNo" type="int" /
<attribute name="name" type="string" />
   </extension>
  </simpleContent>
 </complexType>
</element>
</schema>
```

In XML:

```
<employee empNo="101" name="sathish" >
This is Nit Employee Information
</employee>
```

4) Elements with Empty Content and attributes :

```
<schema>
<element name="employee">
<complexType>
<attribute name="empNo" type="int"/>
<attribute name="name" type="string"/>
<attribute name="salary" type="decimal"/>
</complexType>
</element>
</schema>
```

In XML :

```
<employee empNo="101" name="sathish" salary="12000.0"/>
```

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)

JAXP Versions	Introduced
1.1	JDK 1.4
1.3	JDK5
1.4	JDK 6
1.5	JDK 7
1.6	JDK 8

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)form 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 xcerses.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)

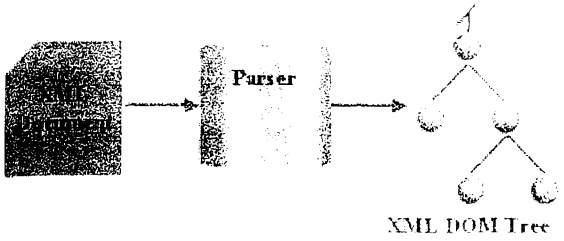
Responsibilities of parsers:

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 DocumentBuildFactory object

To create DocumentBuilderFactory object we can use the following methodpublic static DocumentBuilderFactorynewInstance()

EX: DocumentBuildFactory dbf = DocumentBuilderFactory.new Instance();

- DocumentBuilderFactory is an abstract class implemented by using AbstractFactory 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 DocumentBuildFactory 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 odject

Document doc=documentBuilder.parse("emp.xml");
Element rootElement= documentBuilder.getDocumentElement();
NodeList rootchildnodes=rootelement.getChildNodes();

here child node or elements index wil 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:-

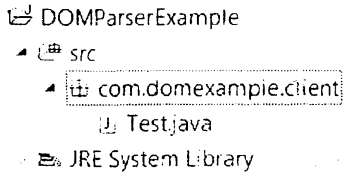
```
if(rootElement.hasChildNodes()){
   NodeList nodeList= rootElement.getChildNodes();
for(int i=0;i<nodeList.getLength();i++){
  Node node=nodeList.item(i);
if(node.getNodeType()==Node.ELEMENT_NODE){
 System.out.println(node.getNodeName()+"--
>"+node.getTextContent());
   }
   }
   }
```

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;
publicclass Test {
publicstaticvoid 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>
```

DTD

```
<!ELEMENT employees (employee*)>
<!ELEMENT employee(deptNo,deptName)>
<!ELEMENT deptNo (#PCDATA)>
<!ELEMENT deptName (#PCDATA)>
<!ATLIST employee empno CDATA #REQUIRED>
<!ATLIST employee name CDATA #IMPLIED>
<!ATLIST employee salary CDATA #IMPLIED>
```

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;
publicclass Test {
publicstaticvoid 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();
}
        }
}
```

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>
```

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;
publicclass ValidateXmlAsPerDTDTest {
publicstaticvoid 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());
}
}
}
```

throws
parse(file)---------->SAXParseException

it is handled by parser by using default ErrorHandler to print first 10 error msg's

parse---------------------> MyErrorHandler(custom Error-handler)
throws SAXException
}ctch(SAXException se){
sysout(se);
}

- 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;
publicclass MyErrorHandler implements ErrorHandler{

        publicvoid warning(SAXParseException exception)
                        throws SAXException {
        System.out.println("warning");
                thrownew SAXException(exception);
        }
        publicvoid error(SAXParseException exception) throws SAXException {
                System.out.println("error");
                thrownew SAXException(exception);
        }
        publicvoid fatalError(SAXParseException exception) throws SAXException {
                System.out.println("fatalError");
        thrownew SAXException(exception);
        }
}
```

public Document parse(File file)throws SAXEception{
try{
internal code of parse method
---code--

}
catch(SAXParserEcxception se){
errorHAndler.error(se);
}
}

class MyErrorHAndlerimplements ErrorHAndler{
public void error(SAXpEse){
trows new SAXExcpetion(se);
}

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>
```

Java Code to parse the above xml example :

```
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;
publicclass ValidateXmlAsPerXSDTest {
publicstaticvoid 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 {
```

```
 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;
publicclass WriteXmlTest{
publicstaticvoid 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);

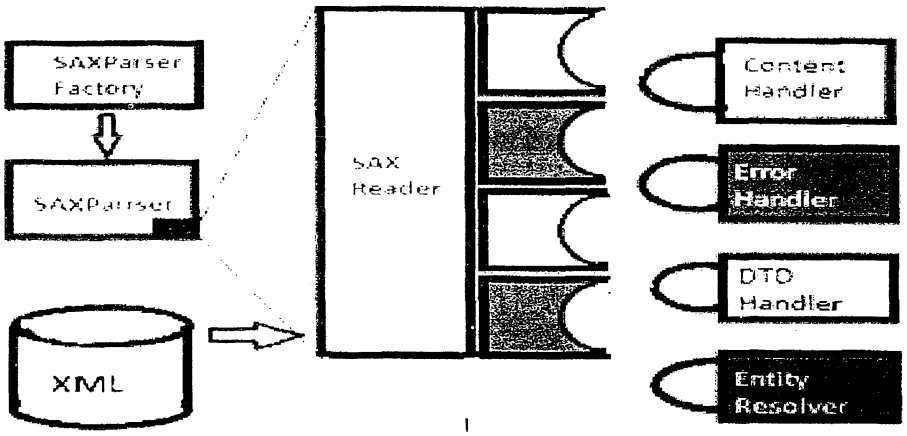
--> How to write MyHandler class?

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.
- enDocument()-To receive notification at the end of the document.
- StartElement(String uri, String localeName, String qualifiedName, Attributes attribute): To receive nofification 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;
publicclass MyHandler extends DefaultHandler {
publicvoid startDocument(){
        System.out.println("Document started");
}
publicvoid endDocument(){
        System.out.println("Document Ended");
}
publicvoid 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);
}
}
}
publicvoid endElement(String uri,String localName,String qName){
        System.out.println(" End Element Name -->"+qName);
}

publicvoid 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);
}
  }
publicvoid error(SAXParseException e)throws SAXException{
        thrownew 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-Notes

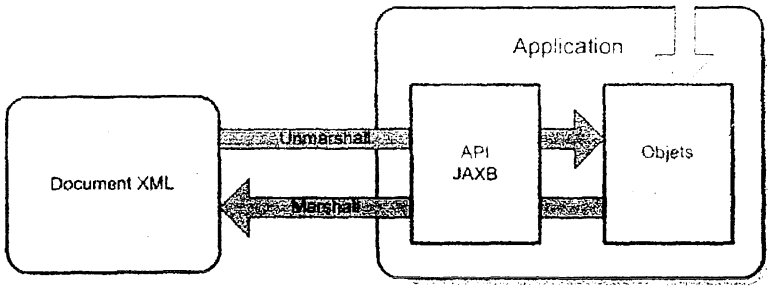
JAX-B

- 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 JAX-B 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.

JAX-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

2publicclassEmployee implementsSerializable {

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
2publicString getName() {
3 returnname;
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 filed and method (getter method) level annotation.
Used to declare the attribute for this complex/simpleType

```
@XmlAttribute
publicString getName() {
 returnname;
        }
```

4)@XmlType

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

1@XmlRootElement

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

3publicclass Employee implementsSerializable {

4...

5}

Steps:-

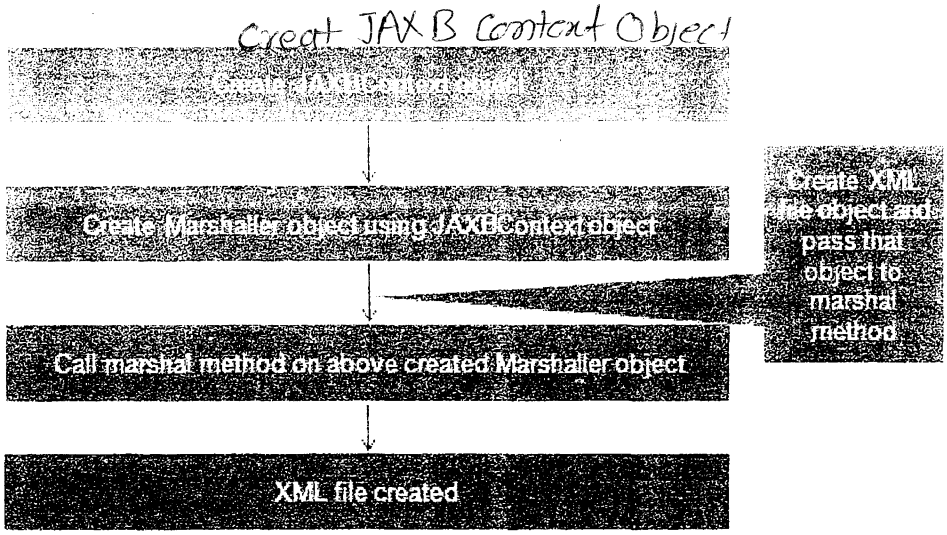
Step1:- create a Binding class(It is a Java Bean based class) and denote the required annotations

Step2:-in the client application

(i) Create JAXBContext object

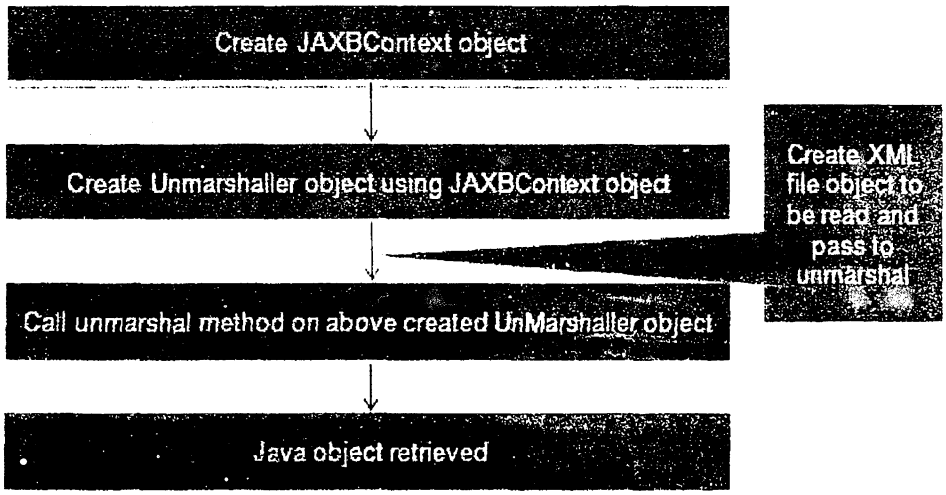
(ii) Perform the Marshalling and UnMarshalling operations as per your requirements

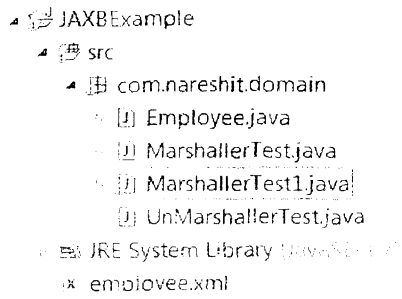
For Marshalling:



For Unmarshalling:

The





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")
publicclass Employee {
privateint empNo;
private String name;
privatedouble salary;
@XmlAttribute(name="empNo")
publicint getEmpNo() {
        return empNo;
}
publicvoid setEmpNo(int empNo) {
        this.empNo = empNo;
}
@XmlElement(name="name")
public String getName() {
        return name;
}
publicvoid setName(String name) {
        this.name = name;
}
@XmlElement
publicdouble getSalary() {
        return salary;
}
publicvoid 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 the 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;
publicclass MarshallerTest1 {
publicstaticvoid 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;
publicclass UnMarshallerTest {
        publicstaticvoid 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.

WebServices

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 its services to another application but not to end users directly.
- For webapplications 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.
- Every Programming language supporting to develop distributed
- applications.
- Java Has various api's like socketProgramming,CORBA,RMI and EJB.
- The RMI and EJB will provide the communication between java application to java application.
- Webservices will provide the communication between any lang application to any lang application.

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 4 Versions of WebServices specifications.that is

1) Basic Profile 1.0(B.P 1.0)
2) Basic Profile 1.1 (B.P 1.1)
3) Basic Profile 1.2(B.P 1.2)
4) Basic Profile 2.0 (B.P 2.0)

-->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

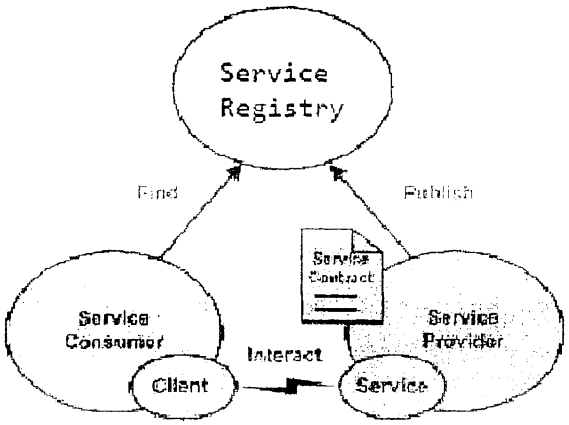
WS-I given standards

-->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 well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

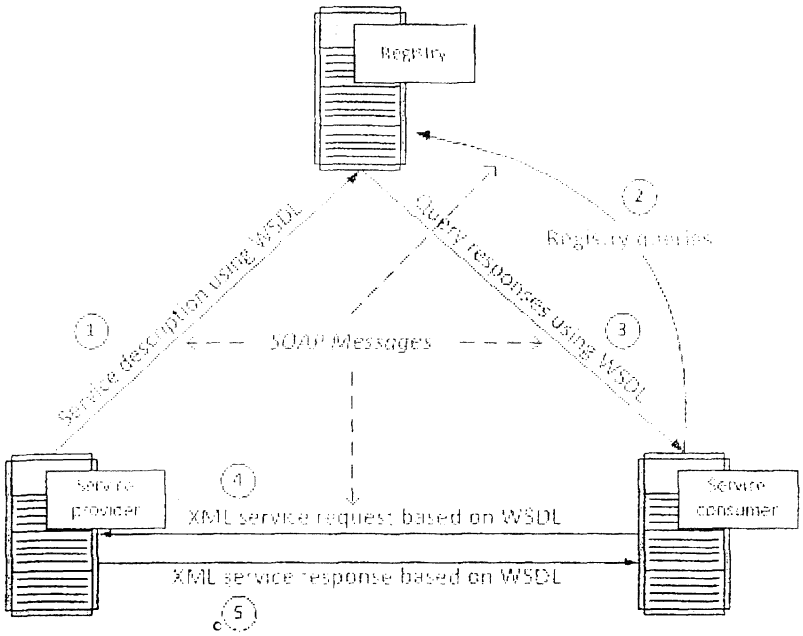
service: Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)

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 skeleton objects will prepare SOAP request and response message in the middle.
- SOAP messages are languageindependent,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

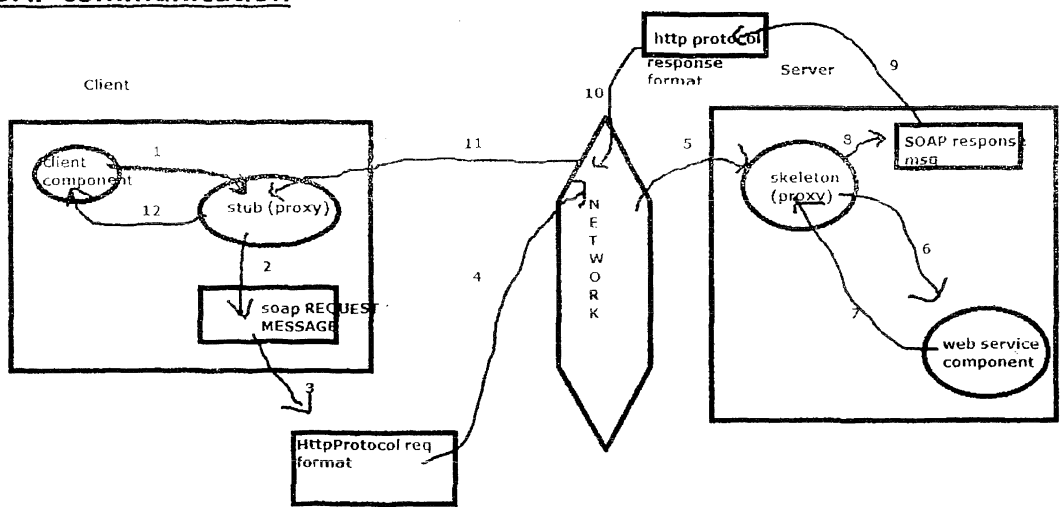
->**WS Security:** SOAP defines its **own security known as WS Security.**

->**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



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 :-

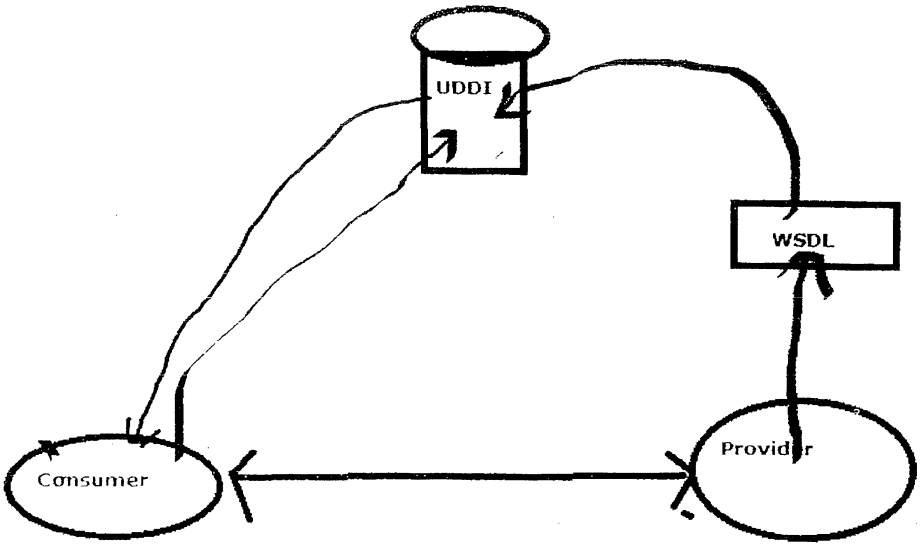
- Web service Description lang. "WSDL" is an xml document, because it should be Lang independent, so that any type of client can understand the services of the provider.
- WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.
- WSDL 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 responsecomes 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 bather 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

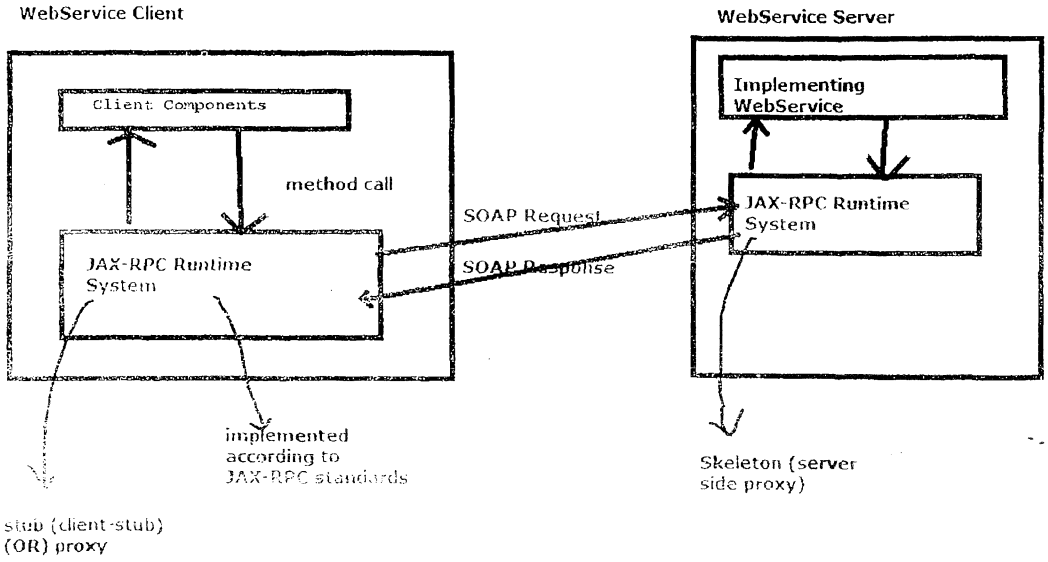
1. JAX-RPC-SI (Standard Implementation) (from sun)
2. Apache Axis1 (from Apache Software Foundation)
3. Web logic web services (from oracle)
4. Web Sphere web services (From IBM)

- 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 JEEversion: 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 theSOAP 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 supporting to work with all transport protocols

jax rpc supporting to develop Servlet or EJB endpoint based app's. jaxws supporting to develop Servlet/EJB/Webservices endpoint based app's . jax rpc not supporting work with annotations but jaxws supporting to work with annotation.

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: In 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.

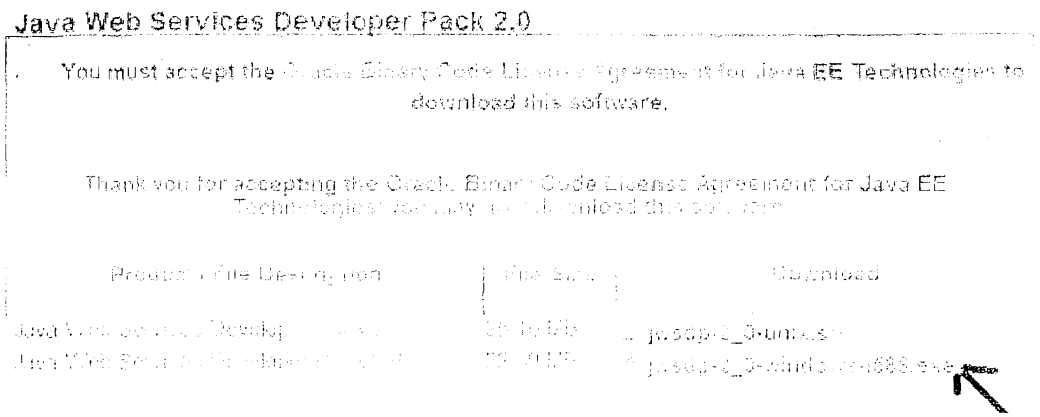
JAXRPC-SI implementation

i. This implementation is given by sun to develop web services.
ii. By using this implementations to develop webservices we required two tools
1) wscompile 2) wsdeploy

iii. 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.

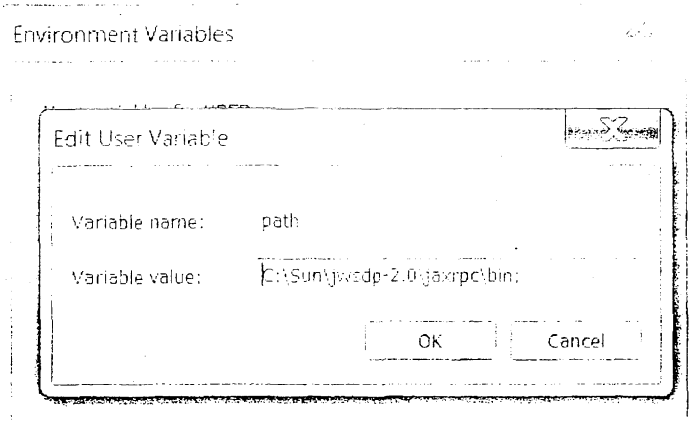
JWSDP S/w Download Link:

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

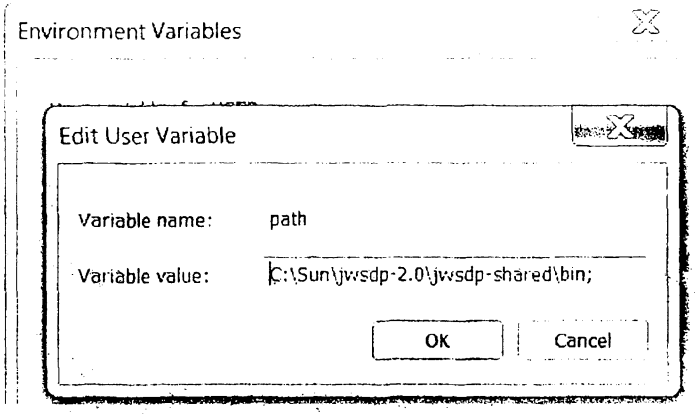


After installing JWSDP 2.0 we can set the path Environment variables as follows

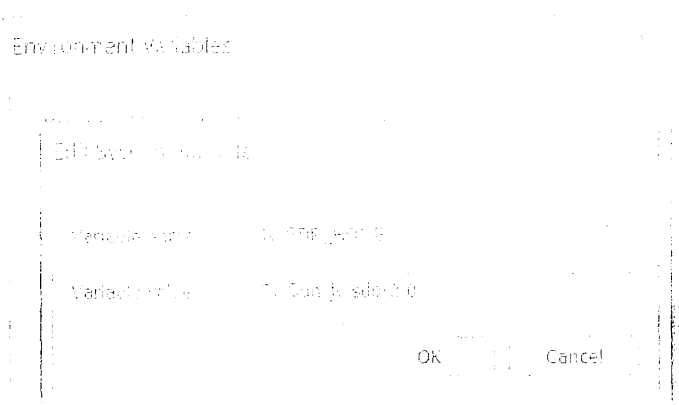
Step1: Edit the path Environment variables then set jwsdp-2.0/jax-rpc path environment variables



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.

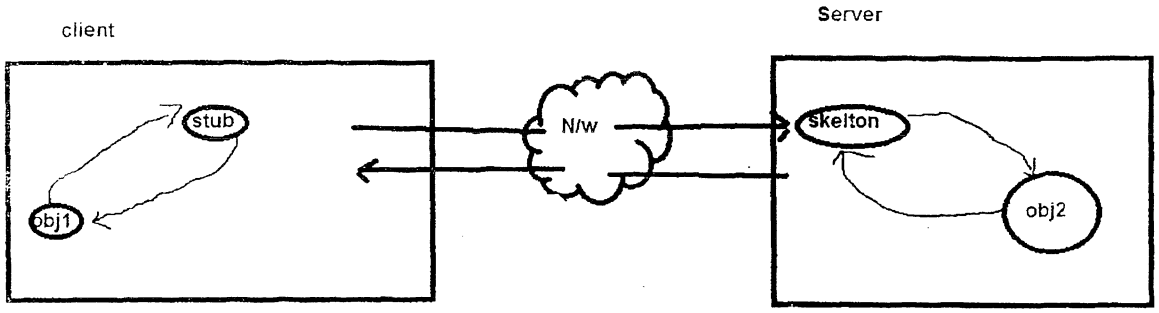
How Object at Different JVMs will communicate ?

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 isAlways accessed remotely by clients.

Rules:

i) Your SEI interface must extends java.rmi.Remoteinterface.
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;
publicinterface IHelloService extends Remote{
public String sayHello(String name)throws RemoteException;
}
```

Step2:- create SEI Implemented Class

sei interfaces make simple java methods as webservice methods

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

SEI methods are exposing dataover the network
SEI methods are accessing by client over the network. because of this reason SEI interface extends remote interface
remote is a marker interface.
serialazable,clonable,randomaccess interface also called as marker interface.
when we are extending or implementing marker interfaces jvm will provide some kinds of abilities for our java class objects.

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.

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:

generate the server side artifide(skelton)

- **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.

the model file maintaining the internal structure of WSDL document.

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

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

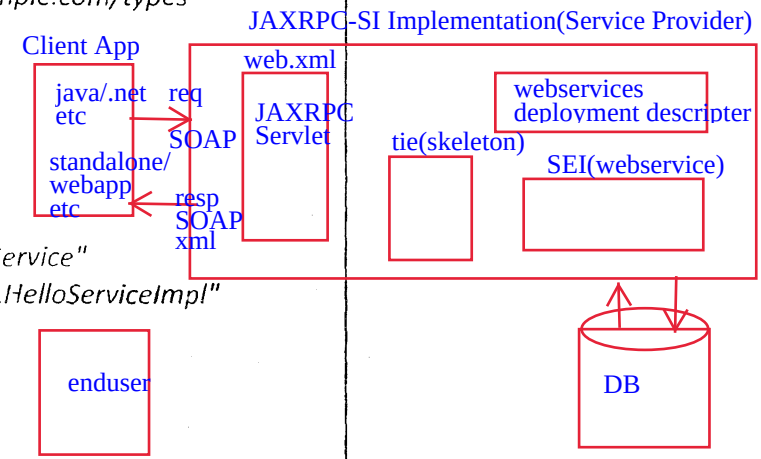
- After generating the necessary binding classes to expose our class as service, we need to
- Configure the endpoint information of our service in a configuration file called as jaxrpc-ri.xml

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>
```



Note:- the webservice deployment description is using by servlet whenever request is coming from client application.To identify the appropiate webservice for the given request.

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.

In jaxrpc-ri.xml file target namespace base and type namespace base attributes are optional if model file created with wscompile tool. if model file is not created with wscompile tool than writing is mandatory.

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 deployent descriptor (in the new war file) that carries the information describing the service, which is used for servicing the request from the consumer.

The wsdeploy tool will configure jax-rpc-servlet in web.xml file

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.

jaxrpc contextListener will load webservices deployment descriptor file from web-INF folder and passing the deployment description to servlet context obj . The JAXRPCServlet is using deployment description from servletContext obj when ever the request is comming from client to identify a suitable skeleton and webservice.

Note: the contextListener's are executing when ever the event's are raised by servletContext obj

To extract we can use jar command as follows

- Jar –xvf target.war
- Tie classes
- The tie classes which are used to communicate with clients.

Skeleton

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 Servelt 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

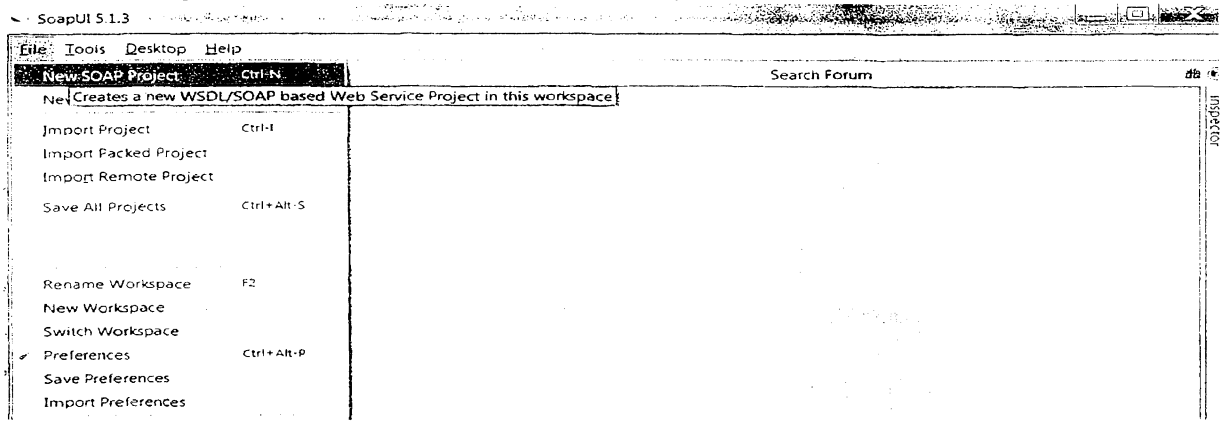
- SoapUI is the world leading Open Source Functional Testing Tool, mainly it is used for API Testing .SoapUI is a free and open source cross-platform Functional Testing solution
- SoapUI enables you to create advanced Performance Tests very quickly and run Automated Functional Tests.

The Current version of SOAP UI tool is SoapUI ~~5.2.1~~ 5.3.1

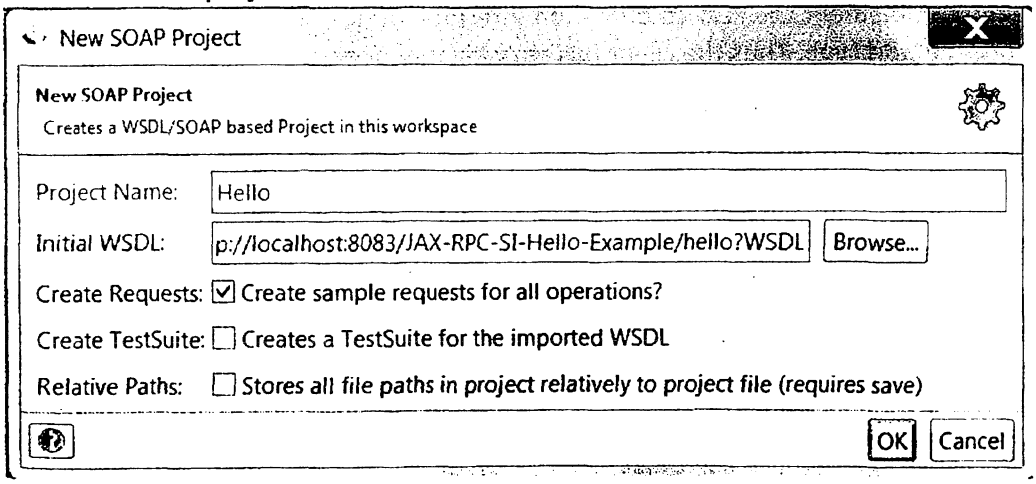
We can Download the The SOAP UI tool from the following link

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

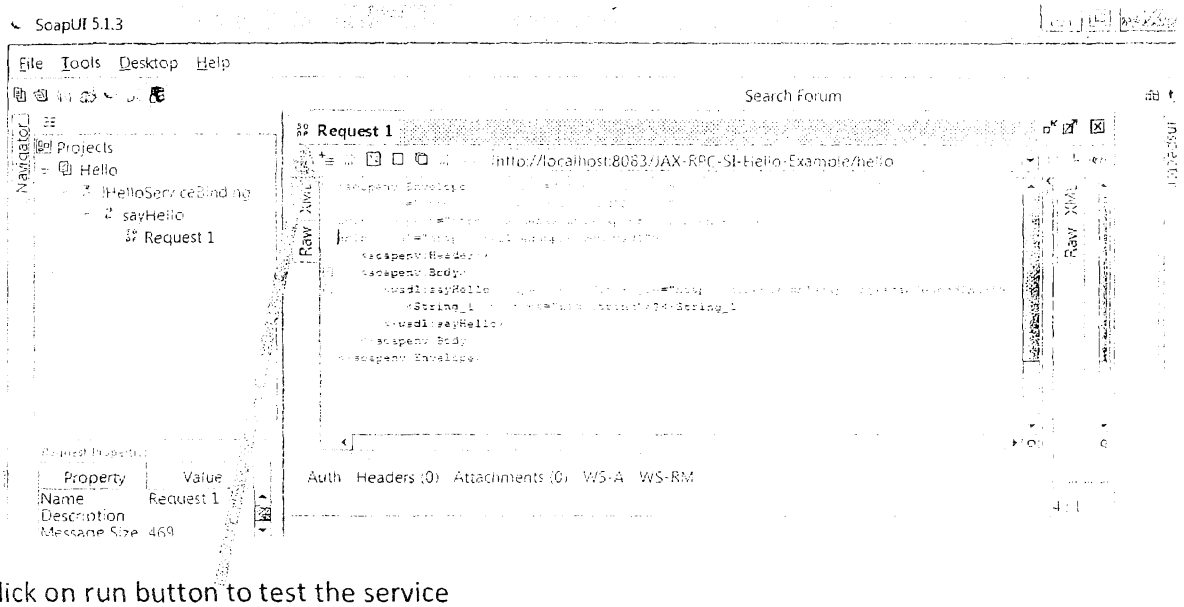
After installing SOAP UI tool create the SOAP Project to test the SOAP webservices.



Enter project Name and WSDL Location



Then click on OK button



Click on run button to test the service

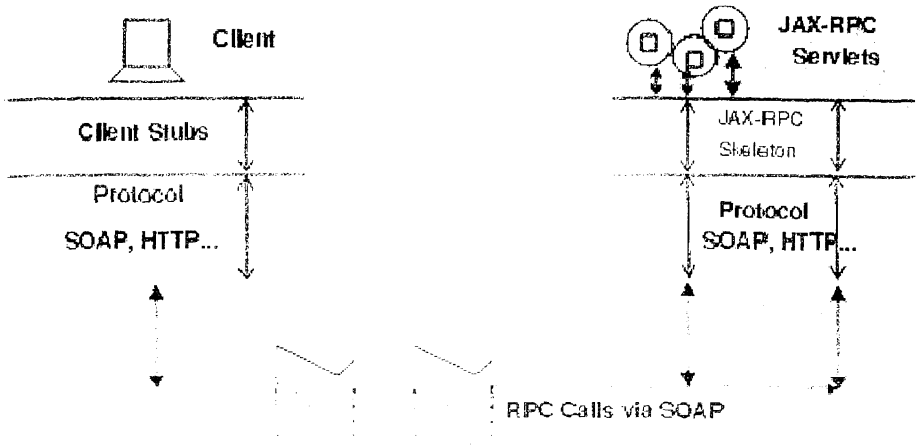
JAXRPC-Clients

- 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 static client
2) Dynamic Proxy
3) Dynamic InvocationInterface (DII)

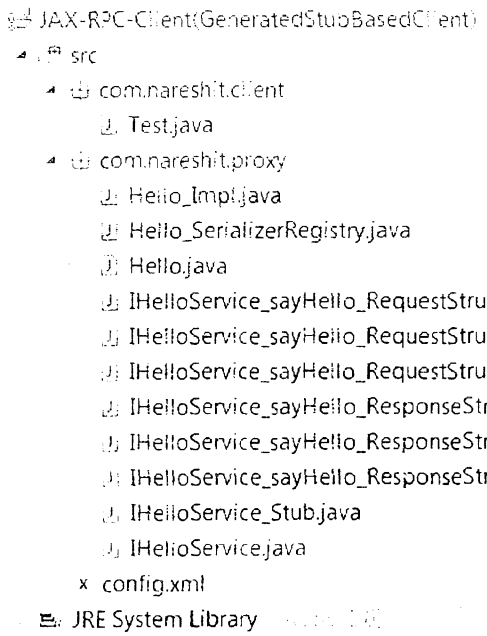
- A stub class is a mapping of a port in the WSDL that describes the Web service. It must therefore implement the service definition interface that reflects the methods of the associated portType. Thus the client has strongly typed, early-bound access to the Web service endpoint.
- The stub must also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



jaxrpc client(generated stub based)

step1- create java project
step2:- write the configuration file with wsdl info
step3:- generate the client side artifacts(like stub) by running wscompile tool
wscompile -gen:client -d src -keep src/config.xml
step4:- write the logic to consume webservice
---create service obj(service obj will provide port obj(sei obj))
---get the sei obj(stub obj) from service obj
---call the webservice methods

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:/Webserviceexamples/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 sidebinding 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.

publicclass IHelloService_Stub **extends**com.sun.xml.rpc.client.StubBase

implements com.nareshit.proxy.IHelloService {

IHelloService_Stub
Implements SEI interface

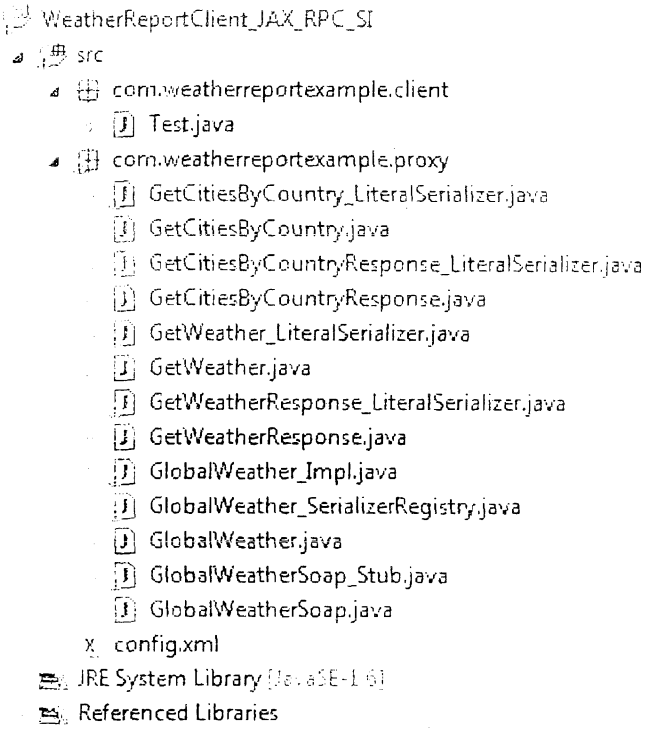
--- }

- 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();   service object
   IHelloService sei=hello.getIHelloServicePort();  sei object get from service
   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

config.xml

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

Test.java

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

}
```

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.

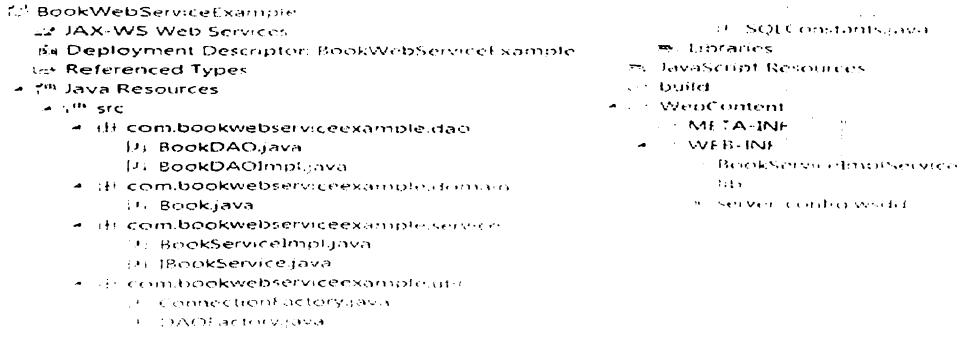
In order work on Apache Axis, First the developer has to download the Apache distribution from http://www.apache.org/dyn/closer.cgi/ws/axis/1_4.

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

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

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;
}
}
```

BookDAO.java

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

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;
}
}
```

factory Design pattern is best for developing project

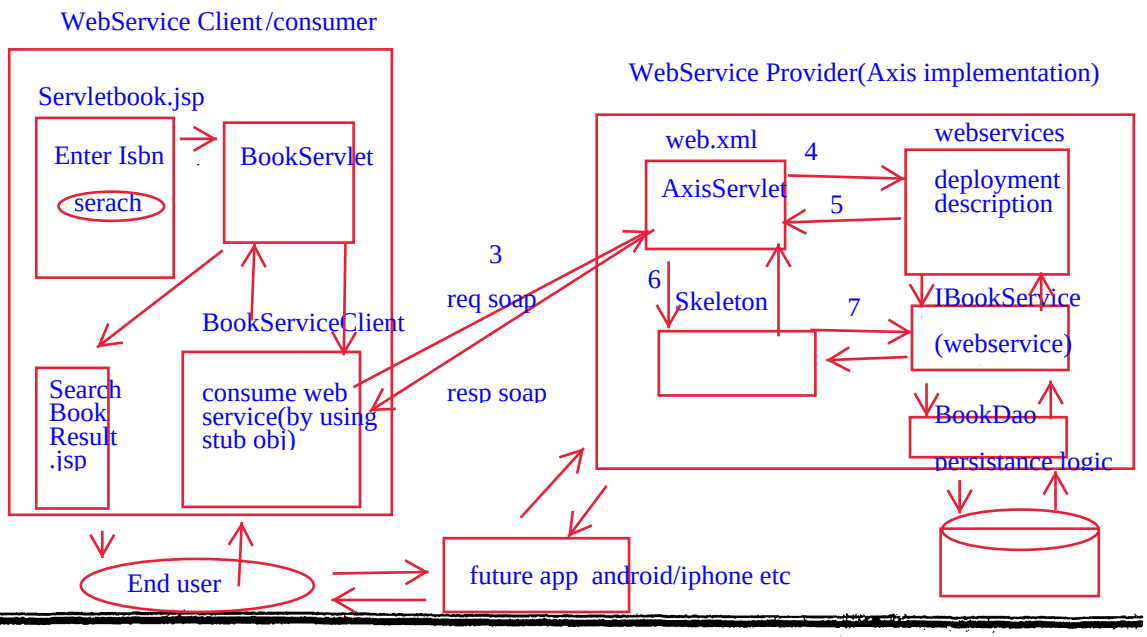
when getting data from .properties file its recommnded to use ResourceBundle bundle=ResourceBundle.getBundle("com/ck/resources/database");

use resource package for xml file and properties file
com.ck.resourse
.properties
.xml
etc

if we are using .properties file in spring bean then use Placeholder(use .properties extension)
in jsp(spring) use ResourceBundleMessageSource (not use .properties extension)

java bean obj/domain obj/Bussiness obj/real world obj

first create in project start
java bean --dao----service----controller----html



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;
}
}
```

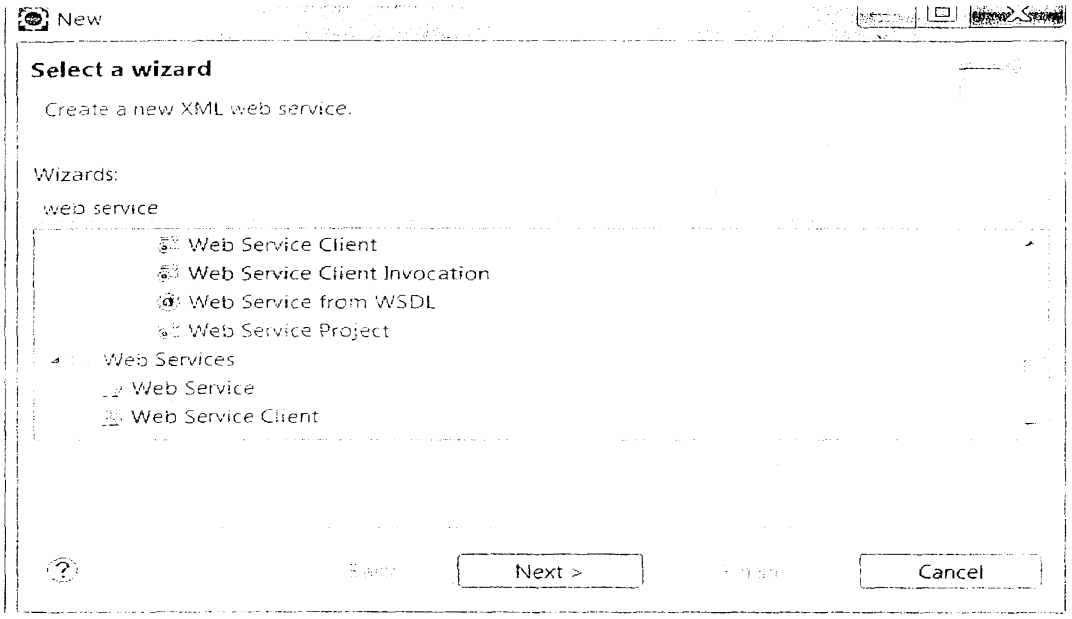
SQLConstants.java

```
package com.bookwebserviceexample.util;

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

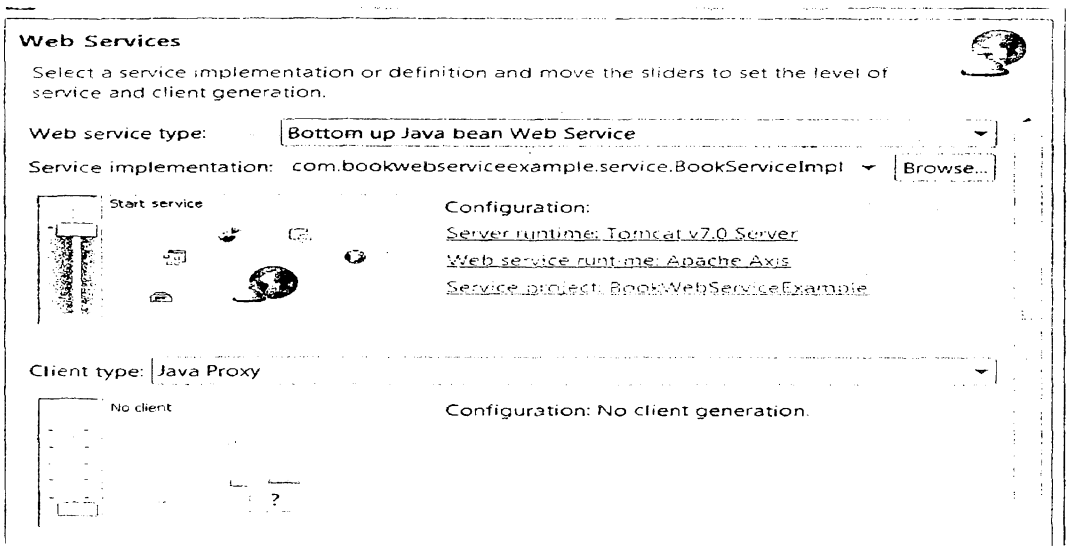
web.xml

```
<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



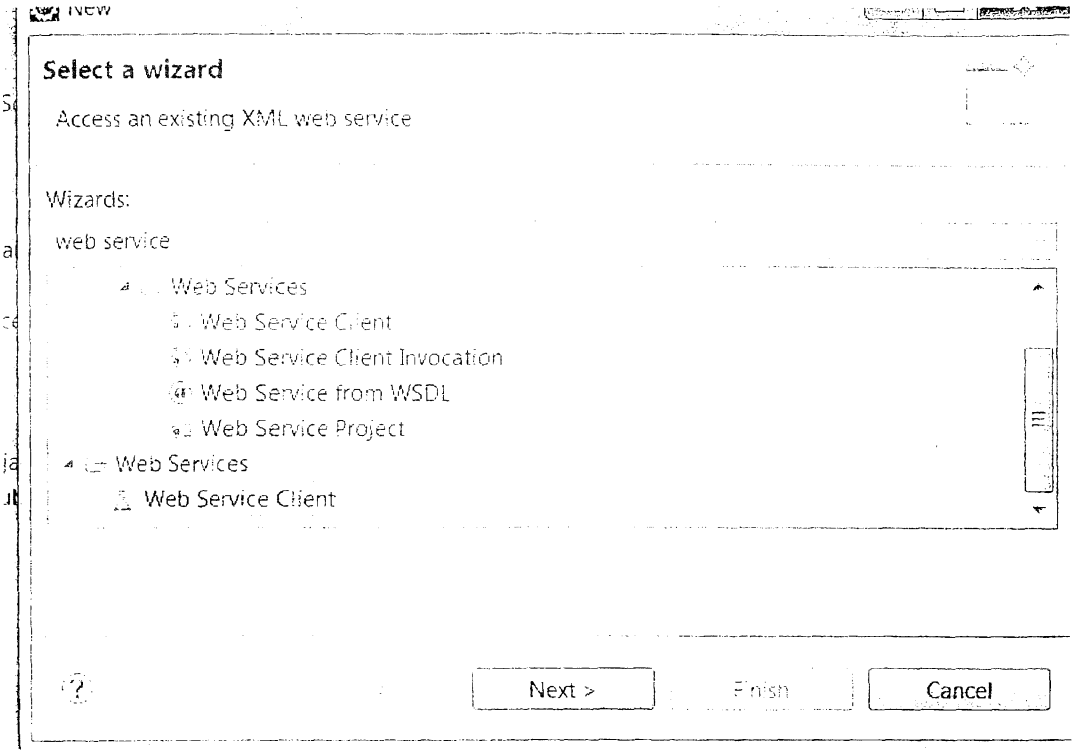
- The click on Click on Next button→Finish button.
- The WSDL and web.xml files are generating By Eclipse.

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:-

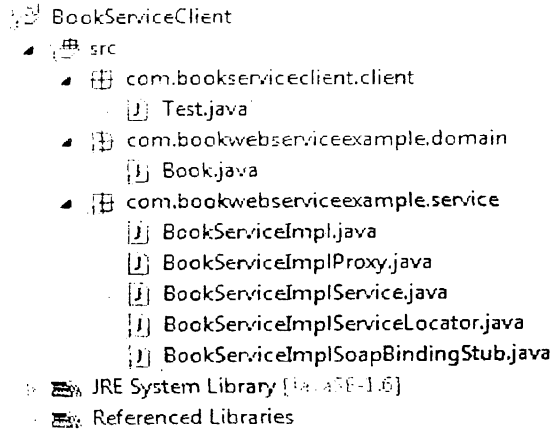
Step1: create Java Project
Step2:- right on the Project →select New →Search for web service Client



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

Test.java

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

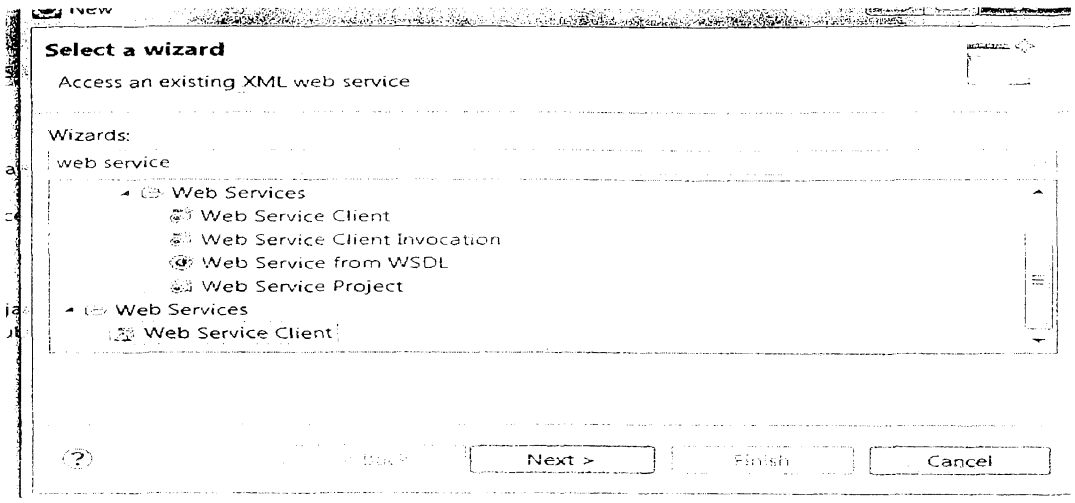
Create A Web Application as Client Application to Consume the Same above Service

- 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 }
```

Facet design pattern:- the facet design pattern is used to group multiple number of interfaces as a single interfaces.
examle:

```
public interface ICustomerService extends Remote{
}
public interface IAccountService extends Remote{
}
public interface IPaymentService extends Remote{
}
public interface IBankService extends Remote{
  public ICustomerService getCustomerService();
  public IAccountService getAccountService();
  public IPaymentService getPaymentService();
}
```

BookServlet.java

```
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
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app >
  <servlet>
    <servlet-name>bookServlet</servlet-name>
    <servlet-class>com.nareshit.client.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>/pages/searchBook.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

searchBook.jsp

```
<html>
<head><h2>SearchBooks</h2></head><br/>
<form action="searchBooks" method="post">
Enter Book Isbn :<input type="text" name="isbn"/>
<input type="submit" value="search"/>
</form>
</html>
```

searchBooksResults.jsp

```
<%@page isELIgnored="false" %>
<%@include file="searchBook.jsp" %>
<table border="1">
<tr><th>ISBN</th><th>Title</th><th>Author</th>
<th>Price</th>
</tr><tr><td>
${book.isbn}</td><td>
${book.title}</td><td>
${book.author}</td><td>
${book.price}</td></tr>
</table>
```

Note: instead of writing the xsd schema's within the wsdl document
better to write xsd schema's as an external files and import into
wsdl type section.(Real time use in project)

```
<wsdl:definitions>
--
</wsdl:type>
<schema>
</complexType>
</sequence>
<element name="isbn" type="string"/>
<element name="title" type="string"/>
<element name="author" type="string"/>
<element name="price" type="double"/>
</sequence>
<complexType name="book">
<element name="registerBookReturn" type="int"/>
<element name="book" type="b:book"/>
<element name="searchBookReturn" type="b:book"/>
<element name="isbn" type="string"/>
elementFormDefault="qualified">
xmlns:b="http://service.nareshit.com"
xmlns="http://w3.org/2001/XMLSchema"
targetNameSpace="http://service.nareshit.com"
<schema
<wsdl:types>
<wsdl:definitions>
}
//setter and getters
private double price;
private String isbn,title,author;
public class Book implements Serializable{
package com.nareshit.service;
}
public String updateBook(Book book) throws REX;
public String registerBook(Book book) throws REX;
public String searchBook(String isbn) throws REX;
public interface IBookService extends Remote{
package com.nareshit.service;
SEI:-
</wsdl:type>
<schema>
<element name="sayHelloReturn" type="string"/>
<element name="name" type="string"/>
xmlns="http://w3.org/2001/XMLSchema">
targetNameSpace="http://service.nareshit.com"
<schema
<wsdl:types>
<wsdl:definitions>
}
public String sayHello(String name) throws REX;
public interface IHelloService extends Remote{
package com.nareshit.service;
```

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>
```

Book.xsd

```
<schema targetNameSpace="http://domin.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<complexType name="book">
<sequence>
<elecment name="isbn" type="string"/>
<elecment name="title" type="string"/>
<elecment name="author" type="string"/>
<elecment name="price" type="double"/>
</sequence>
</complexType>
<schema>
```

SearchBook.xsd

```
<schema targetNameSpace="http://service.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:b="http://domain.nareshit.com"
elementFormDefault="qualified">
<complexType name="book">
<import schemaLocation="D://wsdlExamlpe/book.xsd"
namespace="http://domain.nareshit.com"/>
<elecment name="isbn" type="string"/>
<elecment name="searchBookReturn" type="b:book"/>
</schema>
```

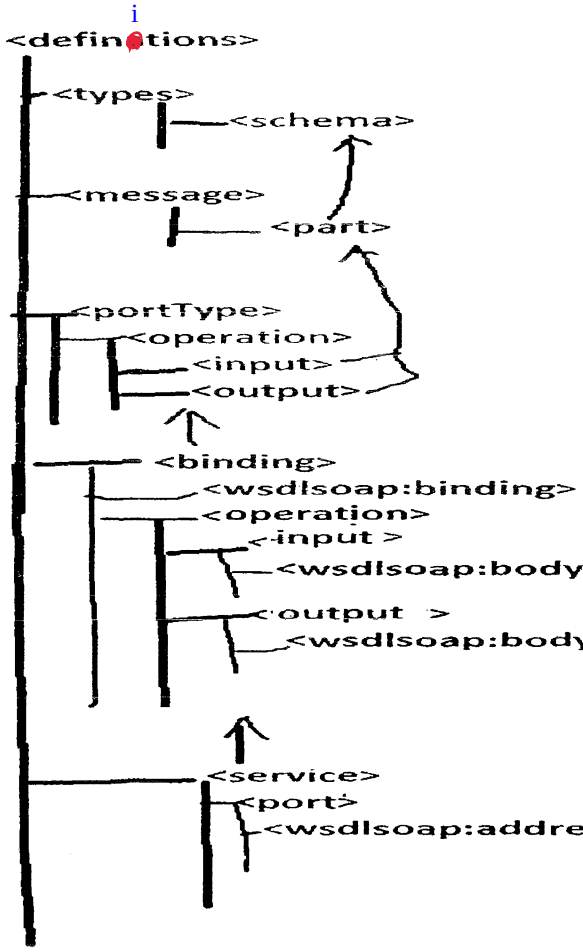
Business delegate;- in general business delegate classes contain business logic exceptional logic , transaction management logic, securiti and logger. the business delegate is a design pattern it comes under j2ee business tier design pattern
VO(TO):-VO stands for value object TO stands for transfer object .VO is a Java bean used to hold the data. in general VO object is using by presentation tier and business tier. the VO contain only string type variables. the VO is not using by integration tier components.
BO:--BO stands for bussiness object. BO is a design pattern .BO is also a java bean class. BO contain variables those are comfortable with db column types. in general bo is using by business tier and integration tier components.the VO and BO both we can use for loose coupling perpose.In general the VO and BO we can maintain as a method parameters or method return type.VO to BO object and BO to VO type conversion we can perform in a bussiness delegate class.

How to create webservice in .net by using asp.net?
step 1:- open visual studio step2:- create asp.net webservice--file--new--website--- left side--visual c# ---right side asp.net webservice--.net version 3.0 ::-enter project name and click ok button
step3::-open service.cs file and write the webservice method(to open project exprolorer(solution Explorer)--click on view menu select solution explorer) webmethod code::

```
[webservice(Namespace="http://tempuri.org/")]
[webserviceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class service :system.web.services.webservice{
public string sayHello(string name)
{
 return "hello" +name+"welcome to .net webservice";
}
}
```

step4:- start the server(f5) or select Debug--start debugging get the wsdl document and text by writing java webservice client application.

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.



registerBook.xsd

```
<schema targetNameSpace="http://service.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:b="http://domain.nareshit.com"
elementFormDefault="qualified">
<complexType name="book">
<import schemaLocation="D://wsdlExamlpe/book.xsd"
namespace="http://domain.nareshit.com"/>
<elecment name="book" type="b:book"/>
<elecment name="registerBookReturn" type="int"/>
</schema>
```

UpdateBook.xsd

```
<schema targetNameSpace="http://service.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:b="http://domain.nareshit.com"
elementFormDefault="qualified">
<complexType name="book">
<import schemaLocation="D://wsdlExamlpe/book.xsd"
namespace="http://domain.nareshit.com"/>
<elecment name="book" type="b:book"/>
<elecment name="updateBookReturn" type="int"/>
</schema>
```

BookService.wsdl

```
<wsl:definition>
<wsdl:type>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<import schemaLocation="D://wsdlExamlpe/searchbook.xsd"
 namespace="http://service.nareshit.com"/>
<import schemaLocation="D://wsdlExamlpe/registerbook.xsd"
 namespace="http://service.nareshit.com"/>
 <import schemaLocation="D://wsdlExamlpe/updatebook.xsd"
  namespace="http://service.nareshit.com"/>
 </schema>
<wsdl:type>
<wsdl:definition>
```

<types>

Its contain XSD schema definition.

- 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. by using xsd schema tags
- Let's take one IHelloService as SEI.

```
package com.nareshit.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
publicinterface 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:defination>
```

<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>

as for the coding standerd recommnded to message as sei name_operationname_request or input similarly seiname_operationnam_response or output we can write for output message

For above SEI the wsdl message tags generating as follows.

```
<wsdl:defination>
<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:defination>
```

If the provider method is taking two parameters, the consumer cannot send those two values independently, rather those has to be wrapped under single message as two parts. 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 Example The provider method is taking two parameters as show:

Ex:

```
public interface PolicyRegistration{
 public MemberCard register(MemberInfo memberInfo,PolicyInfo
policyInfo)throws RemoteException;
}
```

In the above , the register method is taking two parameters , in order to call this method , the consumer shouldn't pass two request to send the inputs to the method rather both the parameters has to be wrapped under one input <message> as two parts, where each part represents one parameter of the method . And the output of the method is represented with a different

```
<message>
<message name="PolicyRegitration_Register">
<part name="memberInfo" type="pt:MemberInfo"/>
<part name="policyInfo" type="pt:PolicyInfo"/>
</message>
```

- A message has name which allows to rerer/identify it. Generally it would be named as SEI Interface_<method> -input message and SEI
- Interface_<method> Response- output message.

<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:definations>
<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:definations>
```

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.

if SEI interface contain 100 method then we required to write 100 operation tag. each operation tag contain one input tag and one output tag

<binding>:

- Binding describeshow 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:definations>
<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:definations>
```

In the above binding we define the transport type as soap over http. Along with this we are specifying the style and use model as document and literal respectively.
In the binding the definition of the operations will specify soap:action for each operation. SoapAction is used for resolving an input request to a method on the provider.

SoapAction: while consumer sending the request to the provider, he will place sapAction in the HTTP Request header. Once the provider has received it he will identify the respective method to invoke based on the soapAction value in the request header.

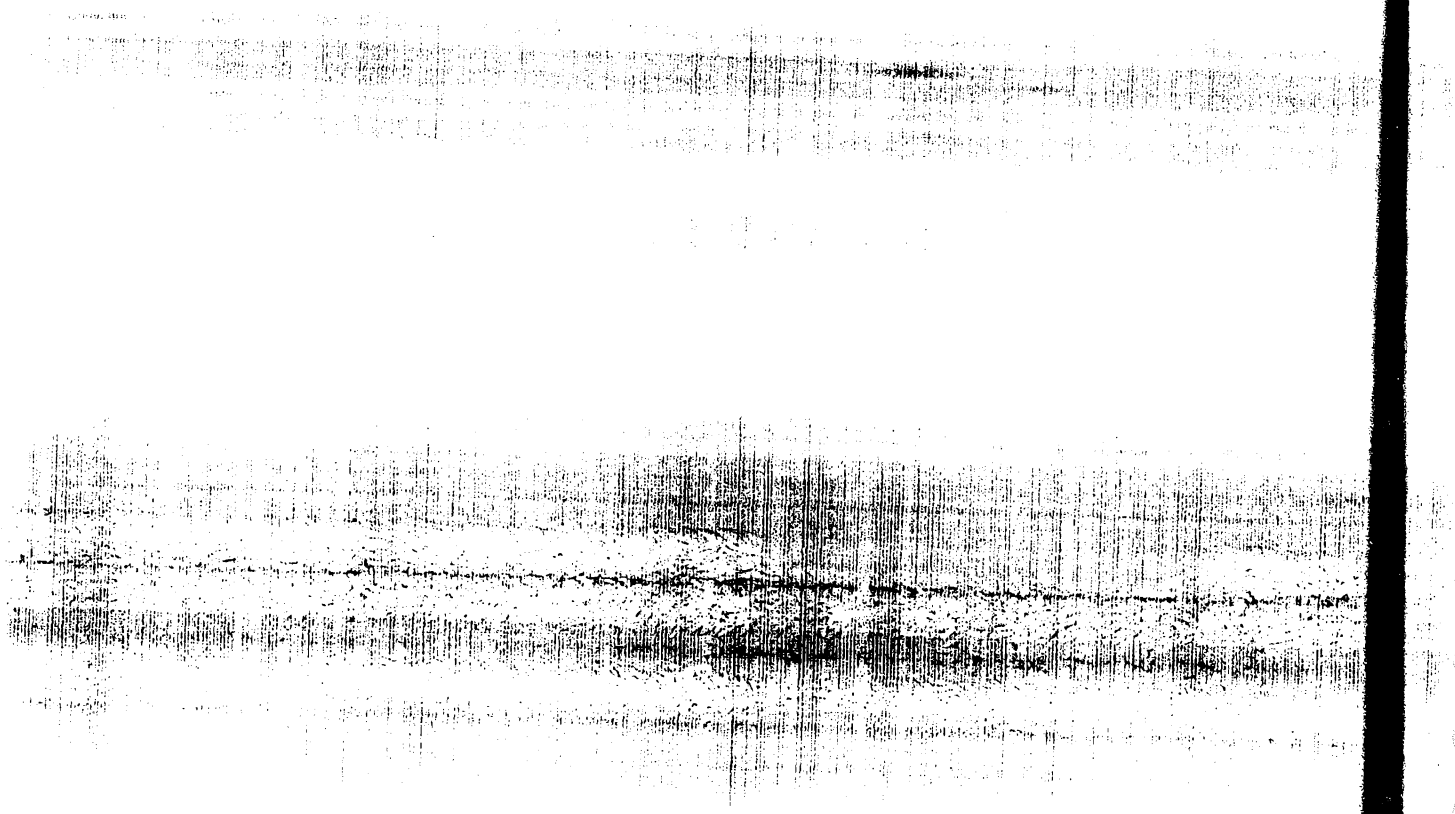
<Service>:

- 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:definations>
<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>
```

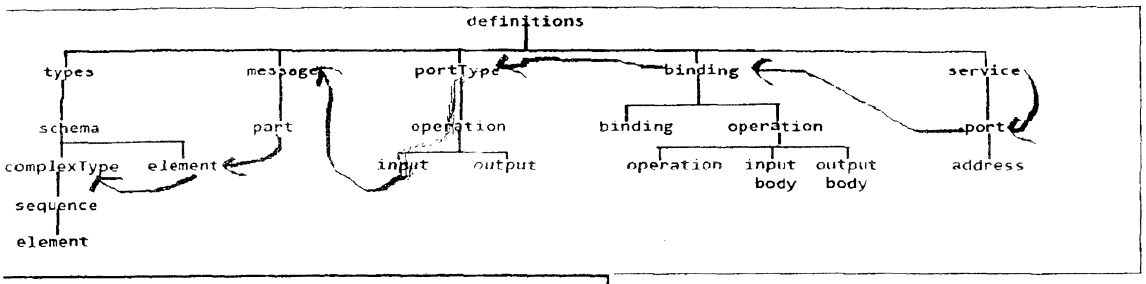
Complete WSDL document for IHelloService Example (document/literal)

```
<?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:impl=http://service.nareshit.com
 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/Hell
oService" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Wsdl elements:-



1.definition :- To declared Namespaces for all WSDL elements.
2.types :- XSD's information,DataTypes.
3.message :- Number of request elements & response elements[parts].
4.portType :- interface name,API.
5.binding :- MEF and Security Information.
6.service :- endpoint url..

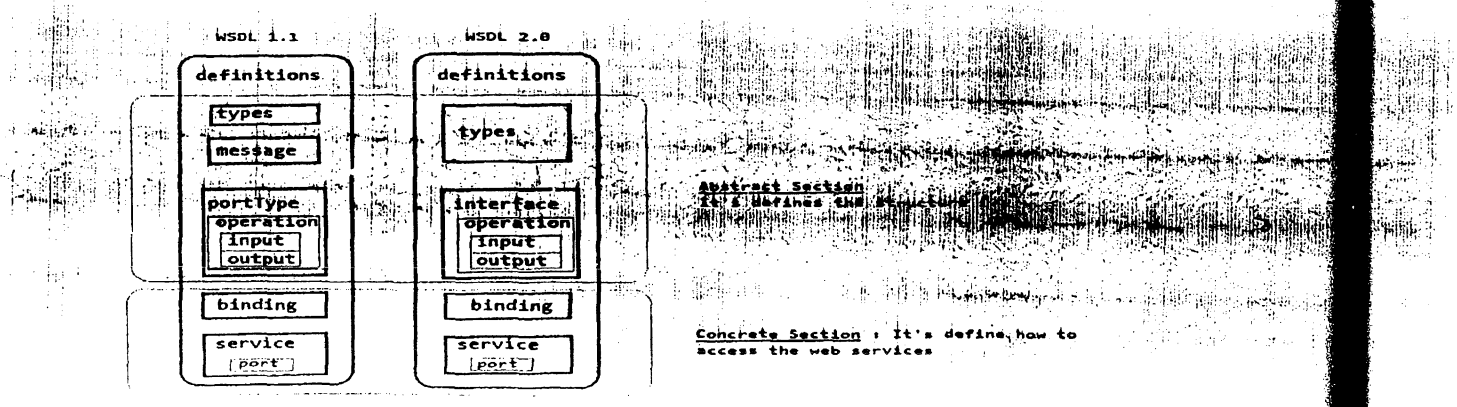
```
public interface OrderService {

  OrderResp getOrder(OrderReq req, String name);
    output      method/api        inputs
}
```

```
<wsdl:definitions
  targetNamespace="http://service.nareshit.com"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nt="http://service.nareshit.com">

  <types>
   <xs:schema>
    <xs:complexType name="OrderType">
      <xs:sequence>
       <xs:element name="name" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="OrderRequest">
      <xs:sequence>
       <xs:element name="req" type="nt:OrderReq"/>
      </xs:sequence>
    </xs:complexType>
   </xs:schema>
  </types>
```

```
  <message name="get_Order_RequDest">
    <part name="OrderReq" type="nt:OrderType"/>
    <part name="name"     type="xs:string"/>
  </message>
  <message name="get_Order_Response">
    <part name="OrderResp" type="nt:OrderType"/>
  </message>

  <portType name="OrderService">
    <operation name="getOrder"/>
      <input  message="get_Order_Request"/>
      <output message="get_Order_Response"/>
    </operation>
  </portType>

  <binding name="OrderBinding" type="nt:OrderService">
    //MEF note : once the xml file is validate,
    </binding>      then format the request

  <service name="OnlineService">
        <port name="OrderPort" binding="nt:OrderBinding"/>
          <soap:address="http:localhost:7000/Proj-107/order"/>
        </port>
  /service>
</definitions>
```

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

Difference Between Wsdl 1.1 and WSDL 2.0 version



SOAP

- SOAP(Simple Object Access Protocol) is 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

A Soap Message contains the following tags

```
<soap:Envelope>
<soap:Header>
<soap:Body>
<soap:Envelop> 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

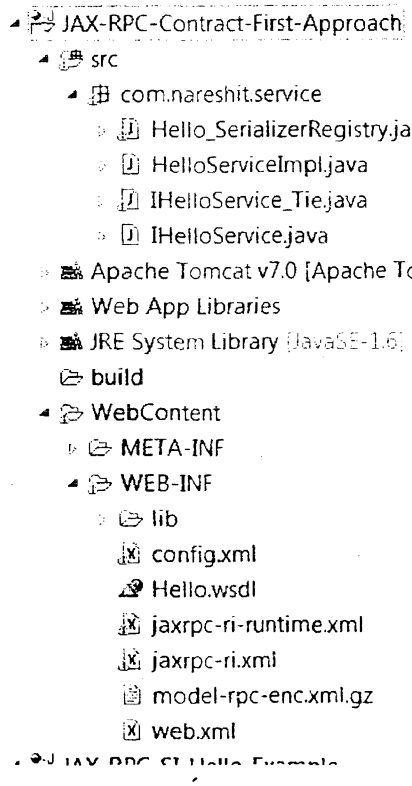
- **Faultcode** – Code to classify the fault.
- **faultstring** – fault message that can be easily understandable
- **faultactor** – the actor in the system who is the cause of this error
- **detail** – detailed error parameters

Contract First Approach :

Approach : Contract First Approach
API :JAXRPC
Implementation :JAXRPC-SI
JDK: JDK6
EndpointServlet :
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 webservices 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:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
```

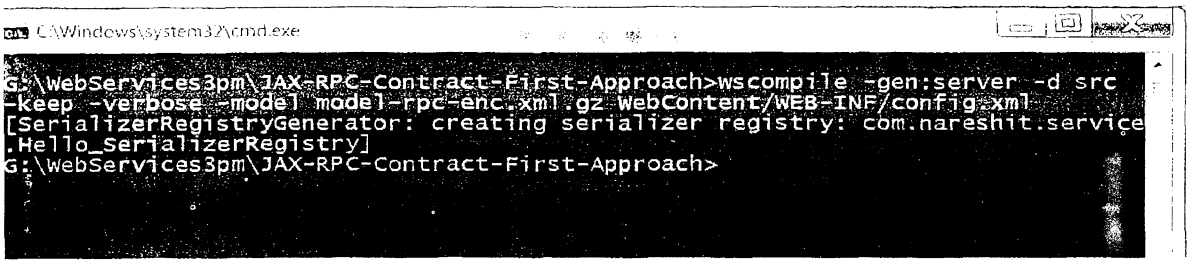
```
<schema elementFormDefault="qualified" targetNamespace="http://service.nareshit.com"
xmlns="http://www.w3.org/2001/XMLSchema">
<element name="name" type="xsd:string"/>
<element name="sayHelloReturn" type="xsd:string"/>
</schema>
</wsdl:types>
<wsdl:message name="sayHelloResponse">
<wsdl:part element="impl:sayHelloReturn"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

tep2:

x config.xml

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

tep3:



tep4:

HelloServiceImpl.java

```
package com.nareshit.service;

import java.rmi.RemoteException;

public class HelloServiceImpl implements IHelloService {

    public String sayHello(String name) throws RemoteException {

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

}
```

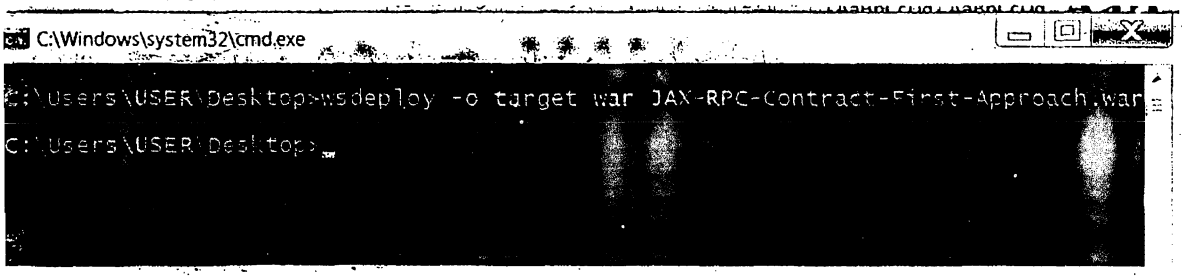
:ep5:

jaxrpc-ri.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<j2ee:webServices xmlns:j2ee="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"/>
    <endpointMapping endpointName="Hello" urlPattern="/hello"/>
</j2ee:webServices>
```

:ep6: create Project as War file

:ep7: run wsdeploy tool



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

Note:refer page no:86 also

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 Consumenr 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 is developed, we need to use the following steps in the Client Application.

1) **CreateServiceFactory object:**

ServiceFactory factory=ServiceFactory.newInstance();

2) **Create Service from the ServiceFactory-**

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

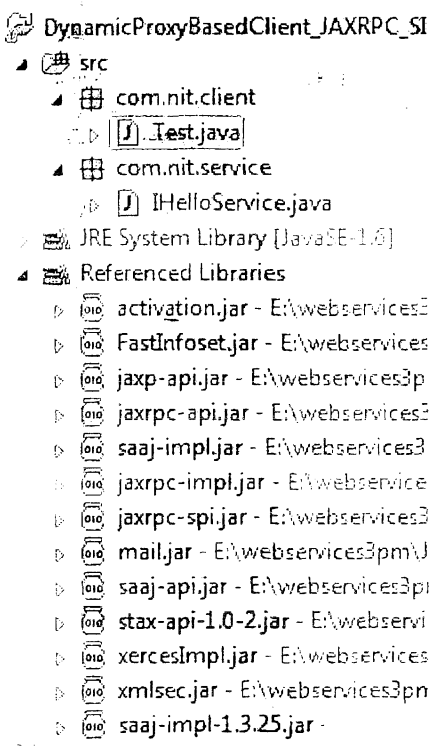
1) WSDL URL
2) QName of the service.
   So, while creating the service object, it loads the WSDL document and searches for the existence of the service qName in wsdl document. If available, with the definations of that Service (reference to the Port) will creates the service.

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

3) *call getPort() method:*
   the getPort() method will returns 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

Test.java

```
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 Obect 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

```
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.

```
package com.nareshit.client;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import javax.xml.rpc.ServiceFactory;
public class Test {
public static void main(String[] args) throws ServiceException,MalformedURLException,
RemoteException {
//create the serviceFactory obj
ServiceFactory serviceFactory=ServiceFactory.newInstance();
//create service object
String serviceName="Hello";
String tns="http://service.nareshit.com/wsdl";
String wsdl="http://localhost:8082/Hello_Service_Example_JAXRPC_SI/hello?WSDL";
 Service service=serviceFactory.createService(new QName(tns,serviceName));
 //create Call object
  Call call=service.createCall();
  call.setTargetEndpointAddress(wsdl);
  call.setPortTypeName(new QName(tns,"IHelloService"));
  call.setOperationName(new QName(tns,"sayHello"));
 call.setProperty(Call.SOAPACTION_URI_PROPERTY,"");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,"http://schemas.xmlsoap.org/so
ap/encoding/");
 String xsd_ns="http://www.w3.org/2001/XMLSchema";
 call.addParameter("String_1",new QName(xsd_ns,"string"),ParameterMode.IN);
call.setReturnType(new QName(xsd_ns,"string"));
//call object is  used to invoke a  webservice
   String msg=(String)call.invoke(new String[]{"sathish"});
   System.out.println(msg);
}
}
```

Java API for XML Web Services(JAX-WS)

The Java API for XML Web Services (JAX-WS) is a Java programming languageAPI 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.

Implementations of JAXWS API

1) **JAX-WS RI (from Sun)**
2) **Metro (from sun)**
3) **Apache CXF (from ASF)**
4) **Apache Axis2 (from ASF)**
5) **Oracle WebLogic Services**
6) **IBM WebSphere Services**

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)

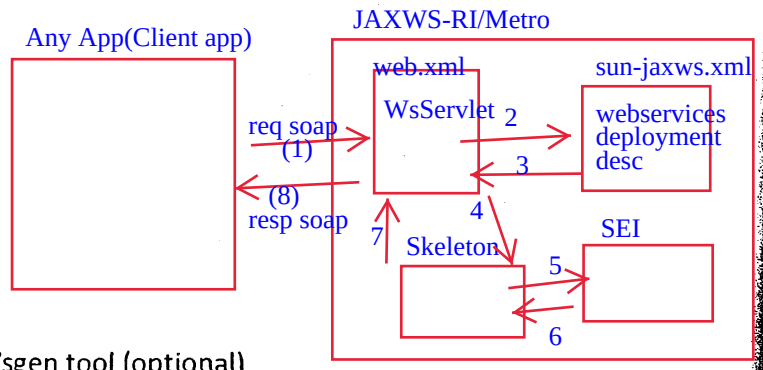
JAXWS-RI Implementation

- 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)

ContextListener

Step1 : Write SEI interface

service oriented architecture (SOA)

- 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 ContractLastApporach the development starts with SEI interface .
- To create SEI in JAXRPC-SI(JAXRPC-API) implementation we need to follows some rules
- SEI interface must extends java.rmi.Remote interface ,to indicate the methods are accessible remotely.
- Whatever the methods are we are declared under SEI All those are Exposing as a webservice methods.
- The Method Parameters and return types must be Serializable and the Methods must throws RemoteException.
- But if we are creating SEI in JAXWS-RI these rule are not required.
- SEI creation In JAX-WS-RI implementation(JAX-WS API)
- In this implementation with SEI interface no need to extends 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 contain exclude attribute.
- If execlude attribute value declared as true then the method not exposing the over the N.w .
- If execlude attribute value declared as false the the method exposed as a webservice method .
- The default value of execlude 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 throws any Exception.
- Note: In the case JAXWS API all the webservice methods throws WebserviceException. It is an UncheckedException

IHelloService.java

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

WsServletContextListener---it will load webservices deployment desc(searches in WEB-INF folder with the name of sun-jaxws.xml) and it will store the webservice desc in servletContext obj.
ContextListener is executing to handle events raised by ServletContext obj.
ServletContext obj is creating at the time of deploying an app into server.
when ever ServletContext obj is creating some event is raised, that event is handleing by context listener.

Annotation to design WebServices JAX-WS-RI:

To design the webservice we should mainly use three annotations

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

in this case the wsdl is generating by wsServlet when we are sending 1 request.
Note:- even though we are using jaxws ri/metro implementation we can use wsServlet and wsServletContextListener and the same tools like wsgen and wsimport.
The current version of metro is 2.3.1. To work with the metro implementation include the following jars in lib folder. jar name:- webservices-api.jar,webservices-extra-api.jar,webservices-extra.jar,webservices-rt.jar,webservices-tools.jar

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 @WebSevice 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 Implemenattion class itself.
- If Sei is wriitern 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 automcatically generates SEI interface based on the WSDL to access it.Even we don't provide a SEI interface the Provider Side.

HelloServiceImpl.java

```
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-RI";
7 }
8 }
```

ServletContext obj will create when ever the app deploying on server.
In this case some Event's will be rised,those Events are handling by listeners.
WsServletContextListener will handle events those are raising by servletContext obj.
--->The Listner will load webservices deployment desc(sun-jaxws.xml)

Generate Binding Classes:- (optional)

the binding classes are automatically generating at run time dynamically

- 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 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 we writing the binding classes to perform this we use the tool that is provided by
- JWSDP/JDK, nothing but wsgen. wsgen will takes 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

for maven : wsgen -d src/main/java -keep -cp target\classes com.ck.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

x 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"/>   or "/*"
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 .

servlet context object=application object

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

web.xml

```
<web-app>
<servlet>
<servlet-name>wsServlet</servlet-name>
<servlet-class>
com.sun.xml.ws.transport.http.servlet.WSServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>wsServlet</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>
<listener>
<listener-class>
com.sun.xml.ws.transport.http.servlet.WSServletContextListener
</listener-class>
</listener>
</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?wsdl

maven: wsimport -d src/main/java -keep http://localhost:8083/JAX-WS-RI-Hello-Example/hello?wsdl
we can consume webservice by using generated stub obj

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]

step: to consume the service
step1:- create service object
-- The service object will provide port obj.
the port obj is nothing but SEI obj(stub obj)
step2:- get the port obj (sei obj)
step3:- call webservice methods by using SEI.

for standalnoe client app ny jar file is not required to include in this case.

Test.java

```
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 /metro
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

sei
Step2: Generate the Java and Binding classes from WSDL by running wsimport tool

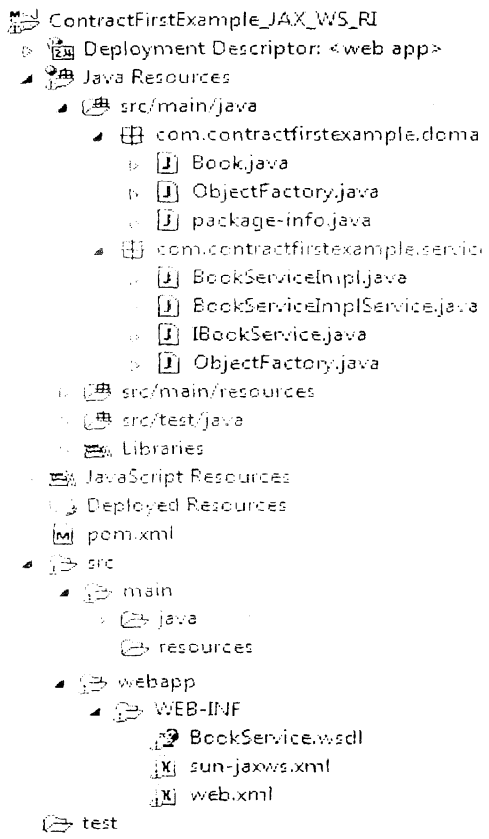
- For this we need run wsimport tool and required to pass WSDL as input and should
- Generate required classes and binding classes as an output.
- Wsimport takes directly the WSDL as input and generates the classes supporting to build both provider as well as Consumer.

Ex:-

wsimport -d src -keep WebContent\WEB-INF\BookService.wsdl
wsimport -d src -keep src/main/webapp/web-inf/bookservice.wsdl

It generates SEI interface, binding classes.

- **Step 3:** write Implementation class for SEI interface with the Required Bussiness Logic
- **Step 4:** write sun-jaxws.xml and web.xml
- **Step 5:** deploy the application into the server



BookService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://service.contractfirstexample.com"
 xmlns:impl="http://service.contractfirstexample.com"
 xmlns:tns1="http://domain.contractfirstexample.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.contractfirstexample.com"
  xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://domain.contractfirstexample.com"/>
<element name="isbn" type="xsd:int"/>
<element name="searchBookReturn" type="tns1:Book"/>
</schema>
<schema elementFormDefault="qualified"
 targetNamespace= "http://domain.contractfirstexample.com"
  xmlns="http://www.w3.org/2001/XMLSchema">
<complexType name="Book">
<sequence>
<element name="author" nillable="true" type="xsd:string"/>
<element name="bookName" nillable="true" type="xsd:string"/>
```

```
<element name="isbn" nillable="true" type="xsd:int"/>
<element name="price" nillable="true" type="xsd:double"/>
</sequence>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="searchBookResponse">
<wsdl:part element="impl:searchBookReturn" name="book"></wsdl:part>
</wsdl:message>
<wsdl:message name="searchBookRequest">
<wsdl:part element="impl:isbn" name="isbn"></wsdl:part>
</wsdl:message>
<wsdl:portType name="IBookService">
<wsdl:operation name="searchBook"  parameterOrder="isbn">
<wsdl:input message="impl:searchBookRequest" name="searchBookRequest">
</wsdl:input>
<wsdl:output message="impl:searchBookResponse" name="searchBookResponse">
</wsdl:output>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="BookServiceImplSoapBinding" type="impl:IBookService">
<wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="searchBook">
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="searchBookRequest">
<wsdlsoap:body use="literal"/>
</wsdl:input>
<wsdl:output name="searchBookResponse">
<wsdlsoap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="BookServiceImplService">
<wsdl:port binding="impl:BookServiceImplSoapBinding" name="IBookService">
<wsdlsoap:address
location="http://localhost:8082//ContractFirstExample_JAX_WS_RI/searchBook"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Generated SEI (IBookService.java)

```
package com.contractfirstexample.service;

@WebService (name = "IBookService", targetNamespace =
"http://service.contractfirstexample.com")

@SOAPBinding (parameterStyle = SOAPBinding.ParameterStyle.BARE)

@XmlSeeAlso ({
  com.contractfirstexample.domain.ObjectFactory.class,
  com.contractfirstexample.service.ObjectFactory.class
})
publicinterface IBookService {

  @WebMethod

@WebResult (name = "searchBookReturn", targetNamespace =
"http://service.contractfirstexample.com", partName = "book")

public Book searchBook(

    @WebParam(name = "isbn", targetNamespace =
"http://service.contractfirstexample.com", partName = "isbn")

int isbn);

}
```

Write BookServiceImpl.java

```
package com.contractfirstexample.service;
import javax.jws.WebService;
import com.contractfirstexample.domain.Book;
@WebService
public class BookServiceImpl implements  IBookService{
public Book searchBook(int isbn) {
Book book=null;
if(isbn==101){
book=new Book();
book.setIsbn(isbn);
book.setBookName("java");
book.setPrice(9000.0);
book.setAuthor("sathish");
}
return book;
}
}
```

sun-jaxws.xml

```
<endpoints
xmlns=http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
<endpoint name="bookservice"
implementation="com.contractfirstexample.service.BookServiceImpl"
url-pattern="/*"/>
</endpoints>
```

web.xml

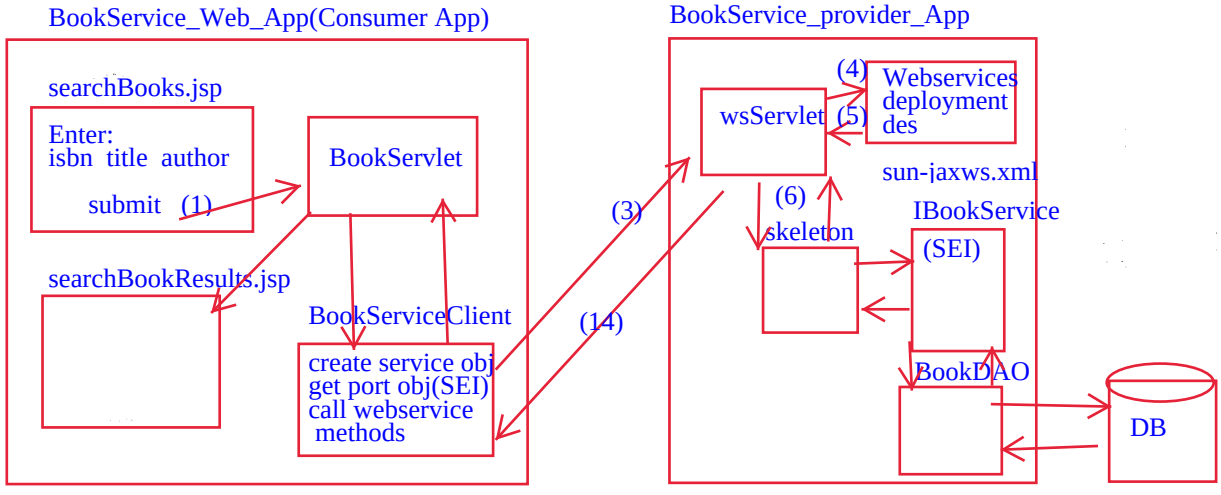
```
 1 <web-app>
 2 <servlet>
 3 <servlet-name>wsServlet</servlet-name>
 4 <servlet-class>
 5 com.sun.xml.ws.transport.http.servlet.WSServlet
 6 </servlet-class>
 7 <load-on-startup>1</load-on-startup>
 8 </servlet>
 9 <servlet-mapping>
10 <servlet-name>wsServlet</servlet-name>
11 <url-pattern>/ *          </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
```

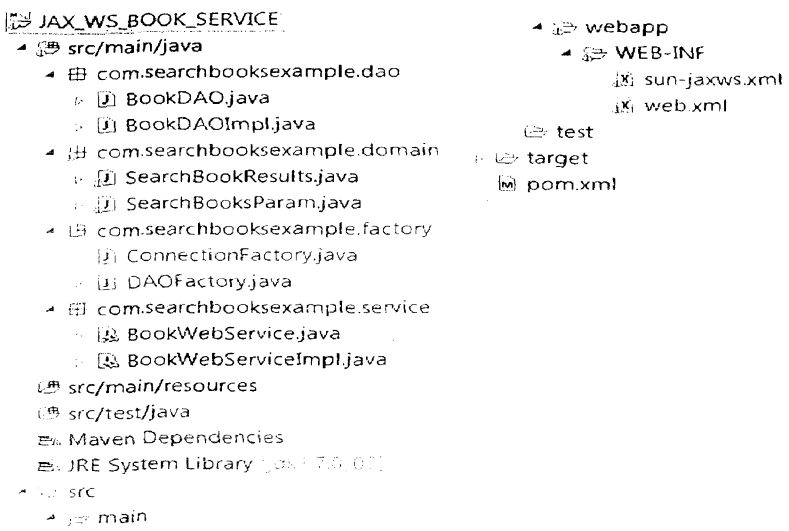
Note : Test the above Service by using SOAP UI tool.

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

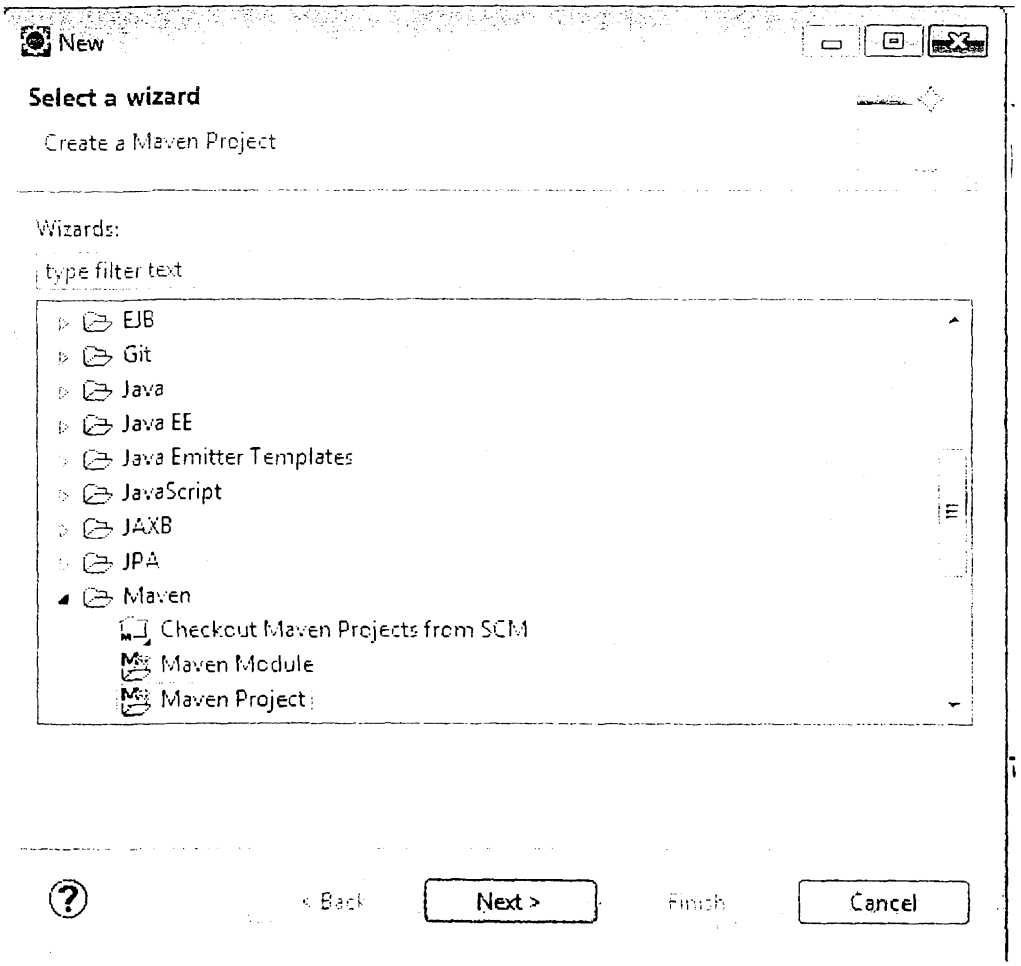
BookService Provider :

The following example is using maven tool.

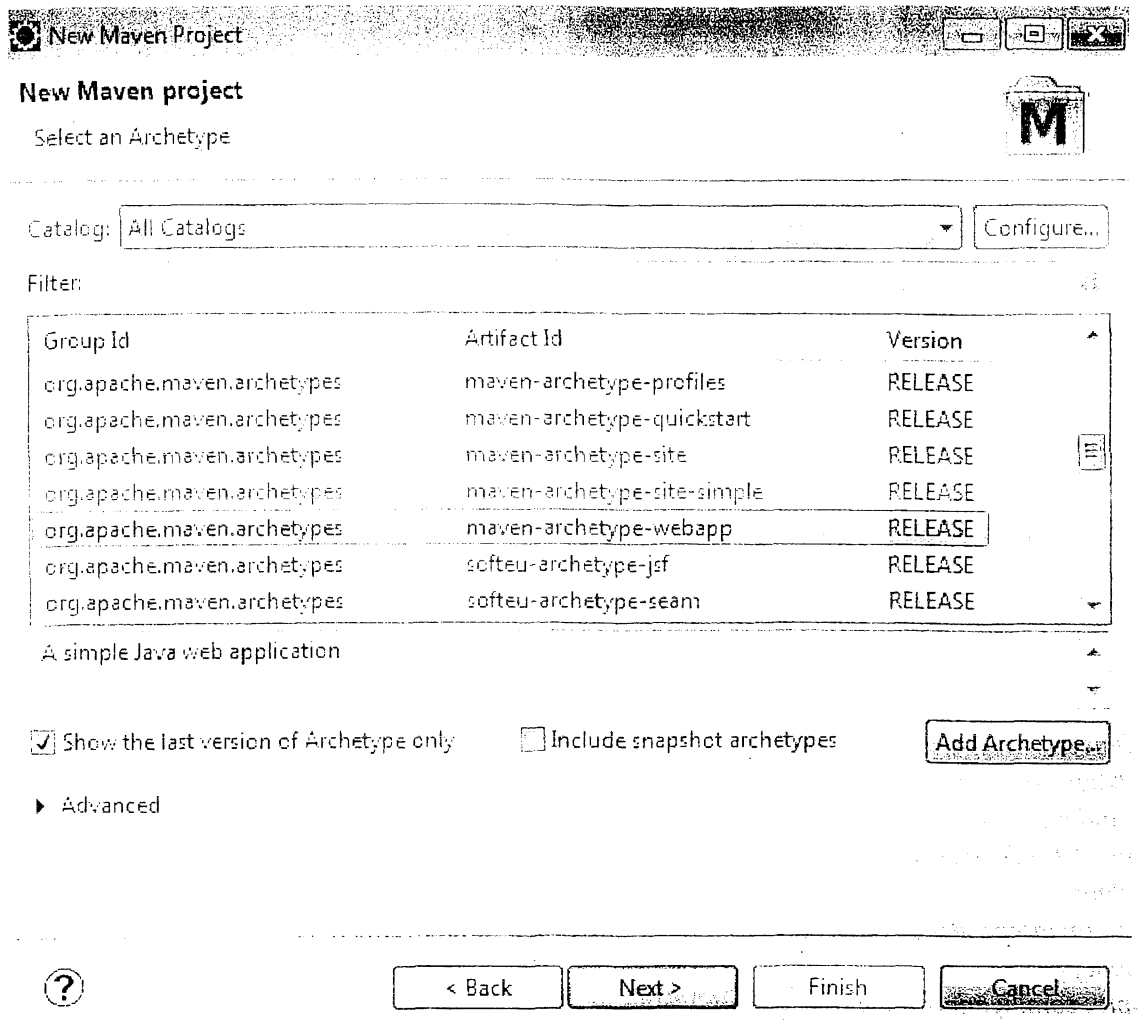




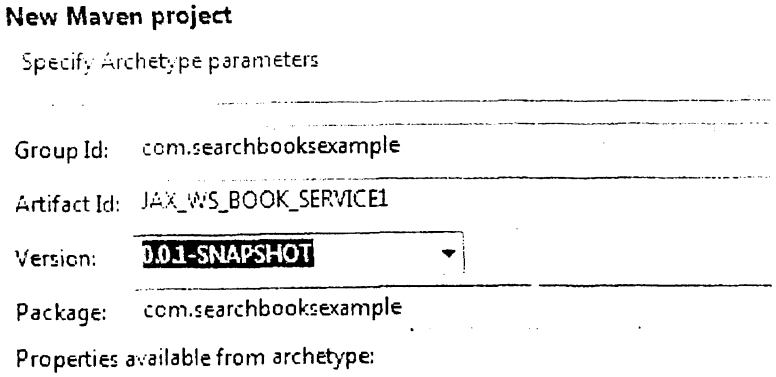
To create the project with maven →click on file menu→new →other→search for maven project→select maven project and click on next button→again click on next button.



Select maven web-app archetype to create webapplication with maven →then click on next button.



Enter groupid : uniqe identifier of project
artifactfactId : project name
version : version of your project



After entering the above details click on finish button.
Open the pom.xml file add required dependencies .
Every maven project contain pom.xml file

```
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> 2.2.6
</dependency>
<!-- https://mvnrepository.com/artifact/com.hynnet/oracle-driver-ojdbc6 -->
<dependency>
   <groupId>com.hynnet</groupId>
   <artifactId>oracle-driver-ojdbc6</artifactId>
   <version>12.1.0.1</version>
</dependency>
</dependencies>
<build>
<finalName>JAX_WS_BOOK_SERVICE1</finalName>
</build>
</project>
```

pom stands for Project Object Model.

if oracle jar's not downloading properly from central repository then install in local repository by using mvn install:install-file command as follows:
C:\>oraclexe\app\oracle\product\10.2.0\server\jdbc\lib> mvn install:install-file -Dfile="ojdbc14.jar" -DgroupId="oracle.jdbc.driver" -DartifactId="ojdbc14" -Dversion="10.2.0" -Dpackaging="jar"
after installing the ojdbc14.jar, to get this jar from local repository t oan application include the dependency as following in pom.xml file
<dependency>
<groupId>oracle.jdbc.driver</groupId>
<artifactId>ojdbc14<artifactId>
<version>10.2.0<version>
<dependency>

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 }
```

BookDAO.java

```
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;
publicclass 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,publication from
Book_Details ");
boolean flag=true;
if(searchBookParam.getIsbn()!=null
&& searchBookParam.getIsbn()>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 }
```

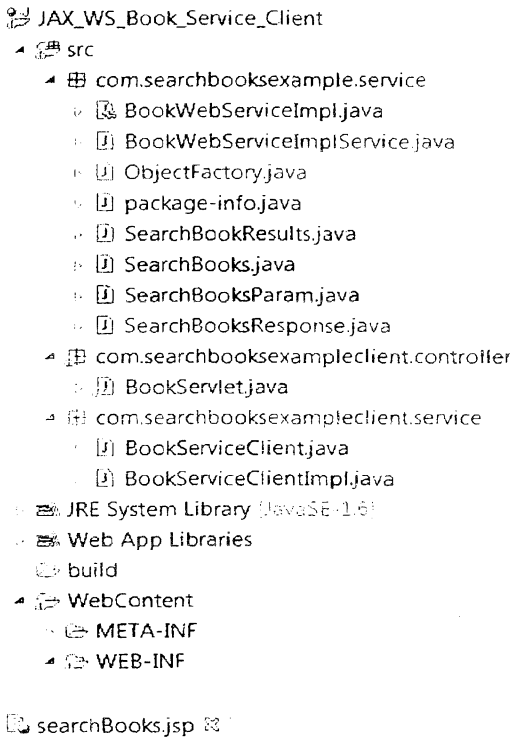
sun-jaxws.xml

```
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
```

Test the above service application by writing a client application.



if we are devloping application by using maven to get the jar files include the following dependency in pom.xml

```
<dependency>
  <groupId>javax.servlet</groupId>
<artifactId>javax.servlet-api</artifactId>
<version>3.1.0</version>
<dependency>
<dependency>
  <groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version1.2</version>
<dependency>
<dependency>
  <groupId>com.sun.xml.ws</groupId>
<artifactId>jaxws-rt</artifactId>
<version>2.1.6</version>
<dependency>
```

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>
```

To get the metro implementation jars from maven use the following dependency in pom.xml

```
<dependency>
  <groupId>org.glassfish.metro</groupId>
<artifactId>webservices-rt</artifactId>
<version>2.3.1</version>
<dependency>
```

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;
importcom.searchbooksexample.service.SearchBooksParam;
import com.searchbooksexampleclient.service.BookServiceClient;
import com.searchbooksexampleclient.service.BookServiceClientImpl;
publicclass BookServlet extends HttpServlet{
private BookServiceClient bookService;
publicvoid init(){
        bookService=new BookServiceClientImpl();
}
publicvoid 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=newSearchBooksParam(); hard coding. make searchBooksParam BO class
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

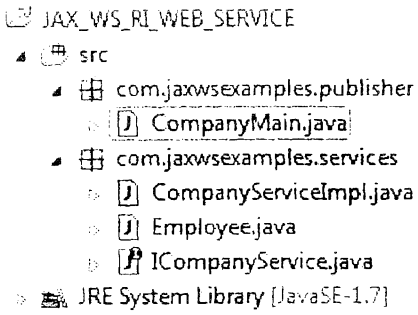
```
package com.searchbooksexampleclient.service;
import java.util.List;
import com.searchbooksexample.service.BookWebServiceImpl;
import com.searchbooksexample.service.BookWebServiceImplService;
import com.searchbooksexample.service.SearchBookResults;
import com.searchbooksexample.service.SearchBooksParam;
public class BookServiceClientImpl implements BookServiceClient {
public List<SearchBookResults>
searchBooks(SearchBooksParam searchBooksParam) {
BookWebServiceImplService service=new
BookWebServiceImplService();
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>
```

Note :

- In all the Application's we integrated Java WebApplication with Webservices Application.
- But it is possible to create webservice application without integrating with Java Web Application.
- When we are integrating the Webservice application with java web application it is required to configure a servlet in web.xml file and webservices deployment destricptor file is also required to be write.But when we are creating only webservice application these files are not required.
- The Following Example is Showing How to create only webservice application and How to publish?



ICompanyService.java

```
1  package com.jaxwsexamples.services;
2  import javax.jws.WebService;
3  @WebService
4  public interface ICompanyService {
5  public Employee getEmployeeDetails(Employee emp) ;
6
7  }
8
```

CompanyServiceImpl.java

```
1  package com.jaxwsexamples.services;
2  import javax.jws.WebService;
3  @WebService(endpointInterface="com.jaxwsexamples.services.ICompanyService")
4  public class CompanyServiceImpl implements ICompanyService{
5  public Employee getEmployeeDetails(Employee emp) {
6       if(emp.getEmpId()==201){
7          emp.setEmpName("abcEmpName from Provider **");
8          emp.setDesignation("abcDesignation from Provier **");
9    }
10       return emp;
11 }
12 }
```

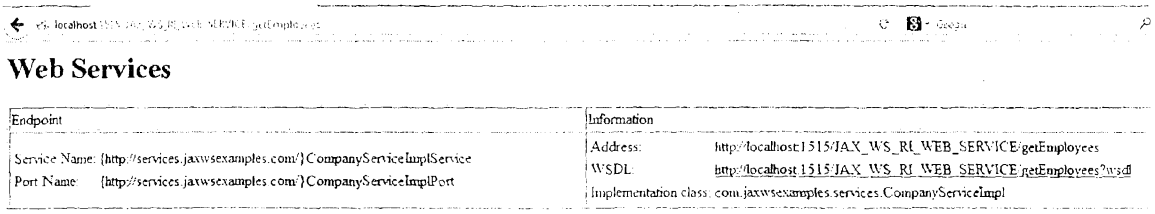
Employee.java

```
1  package com.jaxwsexamples.services;
2  public class Employee {
3     private int empId;
4     private String empName;
5     private String designation;
6      public int getEmpId() {
7        return empId;
8     }
9     public void setEmpId(int empId) {
10       this.empId = empId;
11    }
12    public String getEmpName() {
13       return empName;
14    }
15    public void setEmpName(String empName) {
16       this.empName = empName;
17    }
18    public String getDesignation() {
19       return designation;
20    }
21    public void setDesignation(String designation) {
22       this.designation = designation;
23    }
24 }
25
```

CompanyMain.java

```
1  package com.jaxwsexamples.publisher;
2  import javax.xml.ws.Endpoint;
5  public class CompanyMain {
6     public static final String endPointURI = "http://localhost:1515/JAX_WS_RI_WEB_SERVICE/getEmployees";
7  public static void main(String[] args) {
8        CompanyServiceImpl compImpl = new CompanyServiceImpl();
9        Endpoint.publish(endPointURI, compImpl);
10       System.out.println("webservice published successfully");
11 }
12 }
```

Execute the above Main method class then after open the browser and check wsdl is opening (OR) not.

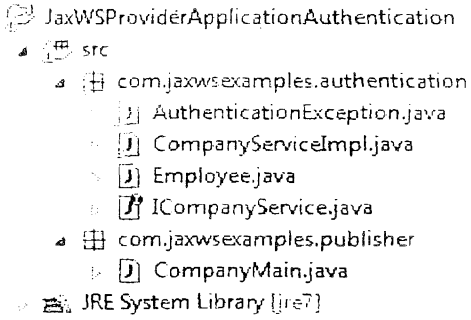


Web Services

Endpoint	Information
Service Name: {http://services.jaxwsexamples.com/}CompanyServiceImplService Port Name: {http://services.jaxwsexamples.com/}CompanyServiceImplPort	Address: http://localhost:1515/JAX_WS_RI_WEB_SERVICE/getEmployees WSDL: http://localhost:1515/JAX_WS_RI_WEB_SERVICE/getEmployees?wsdl Implementation class: com.jaxwsexamples.services.CompanyServiceImpl

Test above Webservice by using SOAP UI tool.

Securing of SOAP Webservices by using JAX-WS-RI :



spring annotation	java community
@autowired	@inject @resource
initialize the object	

Employee.java

Employee.java

```
1  package com.jaxwsexamples.authentication;
2  public class Employee {
3      private int empId;
4      private String empName;
5      private String designation;
6
7      public int getEmpId() {
8          return empId;
9      }
10     public void setEmpId(int empId) {
11         this.empId = empId;
12     }
13     public String getEmpName() {
14         return empName;
15     }
16     public void setEmpName(String empName) {
17         this.empName = empName;
18     }
19     public String getDesignation() {
20         return designation;
21     }
22     public void setDesignation(String designation) {
23         this.designation = designation;
24     }
25 }
```

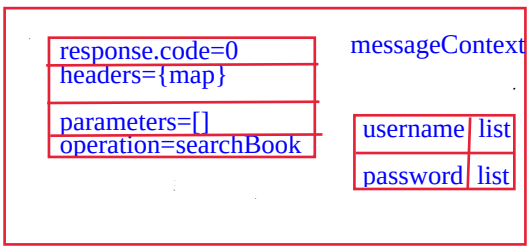
ICompanyService.java

ICompanyService.java

```
1  package com.jaxwsexamples.authentication;
2  import javax.jws.WebService;
3  @WebService
4  public interface ICompanyService {
5  public Employee getEmployeeDetails(Employee emp) throws AuthenticationException;
6
7  }
```

CompanyServiceImpl.java

```
package com.jaxwsexamples.authentication;
import java.util.*;
import javax.annotation.Resource;
import javax.jws.*;
import javax.xml.ws.WebServiceContext;
import javax.xml.ws.handler.MessageContext;
@WebService
publicclass CompanyServiceImpl implements ICompanyService{
        @Resource  it is equal to spring @autowire . it is use initialize the variable.
        private WebServiceContext webServiceContext;
publicEmployee getEmployeeDetails(Employee emp)        throws AuthenticationException{
                if (authenticate()){
                emp.setEmpName("abcEmpName from Provider **");
                        emp.setDesignation("abcDesignation from Provier **");
                        returnemp;
                } else {
thrownew AuthenticationException("you are  not authenicated to access service");
                }
        }
        @WebMethod(exclude=true)  use for not expose over the network
        privateboolean authenticate(){
                MessageContext mctx =webServiceContext.getMessageContext();
Map<String,Object>soapHeader                                                    =
(Map<String,Object>)mctx.get(MessageContext.HTTP_REQUEST_HEADERS);
                List<String>userList = (List) soapHeader.get("user");
                List<String>passwordList = (List) soapHeader.get("password");
                String username = "";
                String password = "";
                if(userList!=null){
                        username = (String)userList.get(0);
                }
                if(passwordList!=null){
                        password = passwordList.get(0).toString();
                }
                if ("jaxwsusername".equals(username) &&"jaxwspassword".equals(password)){
                        returntrue;
                } else {
                        returnfalse;
                }
        }
}
```



Every soap webservice application contain one object that is WebServiceContext object. it is similar to servletContext object of servlet application. the WebServiceContext object automatically creating whenever your webservice application deploying on the network. the WebServiceContext is an interface given by jax-ws 2.0 specification. It allows you to access several context object(like MessageContext obj,UserPrincipal obj) in your Service Endpoint Class. the context object's are holding Security related information',request url,requested method(GET/POST),serviceName,portName etc..
whenever we want to implement the security by using request Headers(SOAP Headers) then we need to work with MessageContext object. the Messagecontext object is a java map based object.it is maintaining the data key and value format.

AuthenticationException.java

AuthenticationException.java

```
1  package com.jaxwsexamples.authentication;
2  public class AuthenticationException  extends Exception{
3     public AuthenticationException(String message){
4        super(message);
5     }
6  }
7
```

CompanyMain.java

CompanyMain.java

```
1  package com.jaxwsexamples.publisher;
2  import javax.xml.ws.Endpoint;
3  import com.jaxwsexamples.authentication.CompanyServiceImpl;
4  public class CompanyMain {
5     public static final String endPointURI = "http://localhost:1414/JaxWSProviderApplicationAuthentication/getEmployees";
6  public static void main(String[] args) {
7        CompanyServiceImpl compImpl = new CompanyServiceImpl();
8        Endpoint.publish(endPointURI, compImpl);
9        System.out.println("webservice published successfully");
10 }
11 }
```

After Deploying the Above Service Then write a client application to consume .
while testing above service by using SOAP UI tool, we can send the security information(username,password)as a request Headers.

Client App:

- JaxWsClientApplicationAuthentication
  - src
    - com.jaxwsclientexample.client
      - Test.java
    - com.jaxwsexamples.authentication
      - AuthenticationException_Exception.java
      - AuthenticationException.java
      - CompanyServiceImpl.java
      - CompanyServiceImplService.java
      - Employee.java
      - GetEmployeeDetails.java
      - GetEmployeeDetailsResponse.java
      - ObjectFactory.java
      - package-info.java
  - JRE System Library [JavaSE-1.7]

Test.java

```
package com.jaxwsclientexample.client;
import java.util.*;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.MessageContext;
import com.jaxwsexamples.authentication.*;
public class Test {
public static final String WS_URL =
"http://localhost:1414/JaxWSProviderApplicationAuthentication/getEmployees?wsdl";
        public staticvoid main(String[] args) {
                CompanyServiceImplService implService = new CompanyServiceImplService();
CompanyServiceImpl sei = implService.getCompanyServiceImplPort();
                setAuthentication(sei);
                Employee emp = new Employee();
                emp.setEmpId(301);
                try{
                Employee empReturn =sei.getEmployeeDetails(emp);
                System.out.println(empReturn.getEmpName() + " .. " +
empReturn.getDesignation());
                }catch(AuthenticationException_Exception e){
                System.out.println(e.getMessage());
                }
        }
        public staticvoid setAuthentication(CompanyServiceImpl sei){
Map<String, Object>req_ctx = ((BindingProvider)sei).getRequestContext();
req_ctx.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, WS_URL);
Map<String, List<String>>headers = new HashMap<String, List<String>>();
headers.put("user", Collections.singletonList("jaxwsusername"));
headers.put("password", Collections.singletonList("jaxwspassword"));
req_ctx.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
        }
}
```

```
MAp<sting,List<String>> soapHeadersMap=new HashMap<>();
soapHeadersMap.put("user",Collection.singltonList("user1"));
soapHeaderMap.put("password",Collection.singltonList("user1"));
BindingProvider provider=(BindingProvider)sei;
//get MessageContext object by calling getRequestContext()
Map<string,object> msgContext=rovider.getRequestContext();
//keep request headers a key and map obj as a value
msgContext.put(MessageContext.Http_ReQUEST_HEADERS,soapHeadersMap);
}
```

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

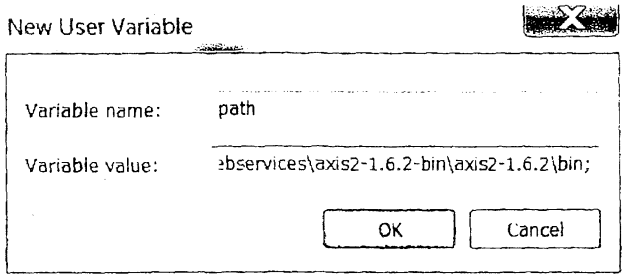
1) Binary Distribution
2) War Distribution

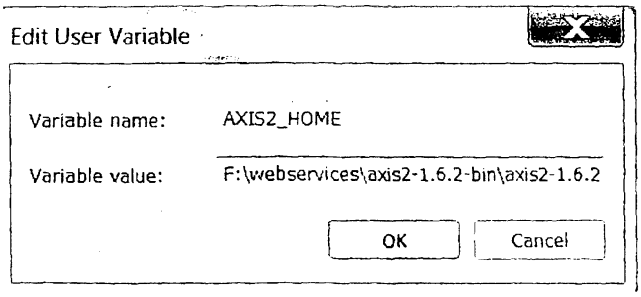
1) Binary Distribution -->which contains the Jar's and tools(Java2WSDL,WSDL2Java) that facilitate the development of Web Services.

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

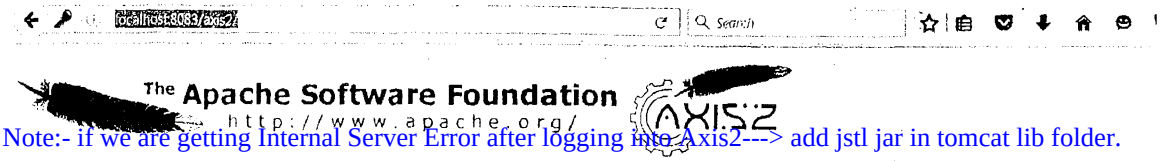
We can download axis2 software distributions from
https://axis.apache.org/axis2/java/core/download.cgi.

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





- **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.**



Note:- if we are getting Internal Server Error after logging into Axis2---> add jstl jar in tomcat lib folder.

Welcome!

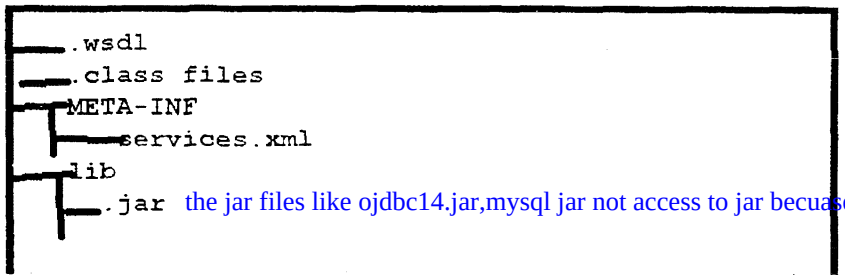
Welcome to the new generation of Axis. If you can see this page you have successfully deployed the Axis2 Web Application. However, to ensure that Axis2 is properly working, encourage you to click on the validate link.

- 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 ContractFirstApproach

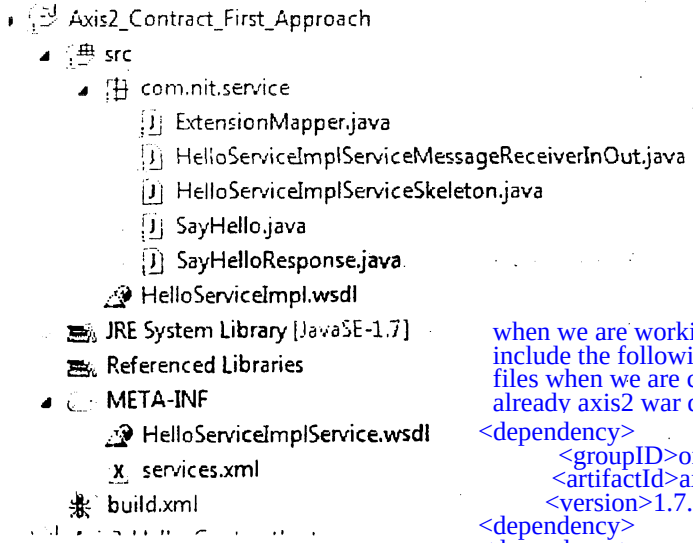
Step1: write the wsdl document
Step2: Generate the service classes and services.xml file by using wsdl2java tool
Step3: write the Bussiness logic in the generated Service class. (skeleton class)
Step4: create the project as an archive file (.aar extension)
Step5:- Deploy project into the axis2 war distribution

Axis2ProjectDirectoryStructure



the jar files like ojdbc14.jar,mysql jar not access to jar becuase axis jar already availble in axis2 war distribution

ContractFirstApproach Application:



maven---structure

src/main/java
--com.nareshit.services
hello.wsdl
maven dependency
jre
--META-INF
Hello.wsdl
services.xml
--src
target
pom.xml

when we are working with maven just to compile the generated axis2 service classes include the following two dependencies in pom.xml(the jars not comming in the .aar files when we are creating our project as an archive file and also not required. Beacuse already axis2 war distribution contain axis2 jar's)

```
<dependency>
        <groupID>org.apache.axis2</groupID>
       <artifactId>axis2-kernal</artifactId>
      <version>1.7.4<version>
<dependency>
<dependency>
        <groupID>org.apache.axis2</groupID>
       <artifactId>axis2-adb</artifactId>
      <version>1.7.4<version>
<dependency>
```

Step1: write the wsdl document

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
 targetNamespace="http://service.nit.com"
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: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

-ss means Generate Server Side Artifacts.

-sd means Generate Service Descriptor File.

Maven:-wsdl2java -ss -sd -uri src/main/java/HelloServiceImple.wsdl .

Mote- after running the tool refresh the project. the com folder drag and drop in to src/main/java. Must be rename the resource folder name into META-INF name.

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;
  public class HelloServiceImplServiceSkeleton{
     public SayHelloResponse sayHello(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)

->right click on Project →Export→Select Jar File→Browse Location→Enter File Name with .aar extension →click on SAVE →Click on Finish button.

Step5:- Deploy into the War distribution

To Deploy into the War Distribution Login into Axis2 Administration →UploadService

File(.aar file)

(OR)

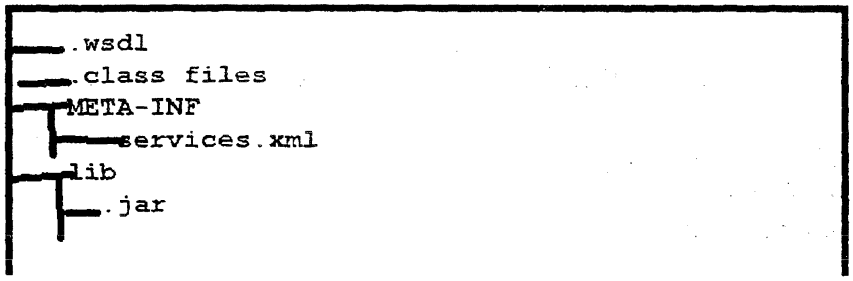
Place the *.aar file directly in the servlet engine's webapps/axis2/WEB-INF/services directory.

After Deploying the services file test by using SOAP UI tool.

Axis2 Project Contract Last Approach:

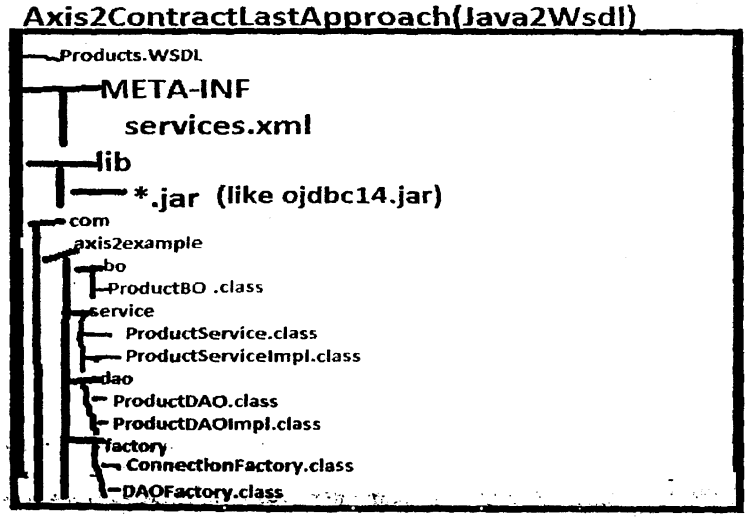
Must be maintain the Project Directory Structure as follows

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(Java2Wsdl)



ProductService.java

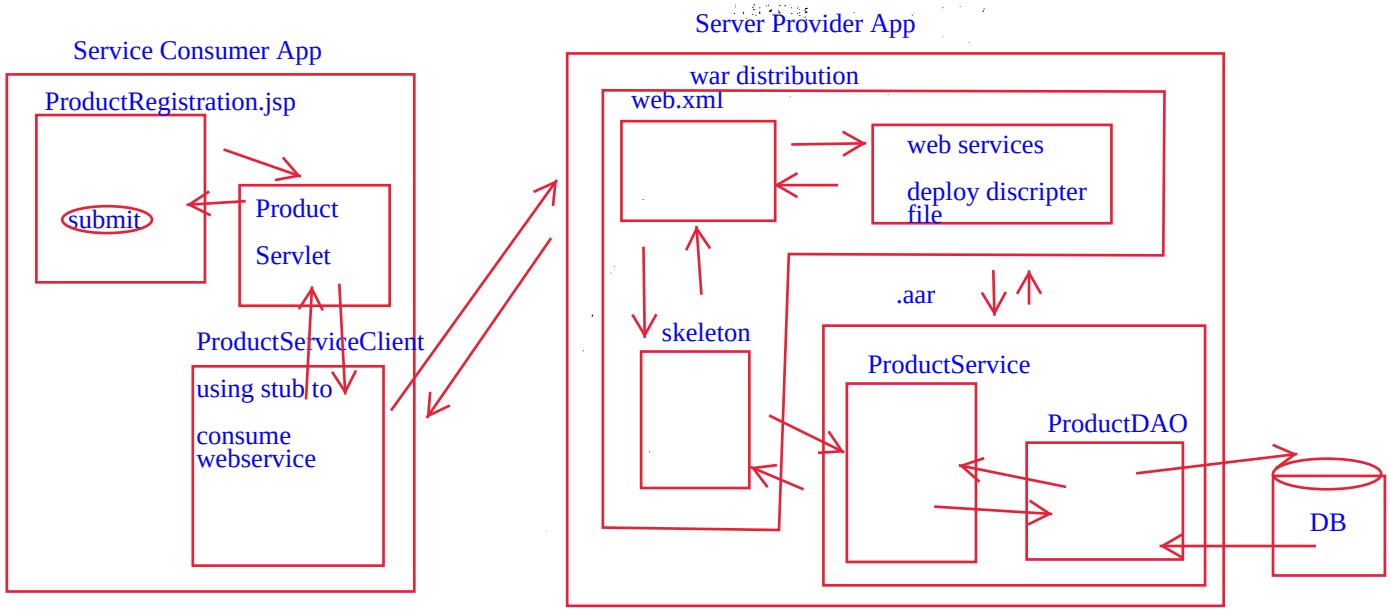
```
package com.axis2example.service;
import com.axis2example.bo.ProductBO;
public interface ProductService{
public boolean registerProduct(ProductBO productBo);
}
```

ProductServiceImpl.java

```
package com.axis2example.service;
import com.axis2example.bo.ProductBO;
import com.axis2example.factory.DAOFactory;
public class ProductServiceImpl implements ProductService{
public boolean registerProduct(ProductBO productBo){
  boolean flag=false;
  if(productBo!=null){
 int count=DAOFactory.getProductDAO().registerProduct(productBo);
  if(count>0){
  flag=true;
  }
  }
  return flag;
}
}
```

ProductDAO.java

```
package com.axis2example.dao;
import com.axis2example.bo.ProductBO;
public interface ProductDAO {
public int registerProduct(ProductBO productBO);

}
```



DAO

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(?,?,?)";
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 }
31 }
```

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

```
package com.axis2example.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) {
e.printStackTrace();
}
}
public static Connection getConnection() throws SQLException{
String url="jdbc:oracle:thin:@localhost:1521:XE";
String username="system";
String password="manager";
Connection con=DriverManager.getConnection(url,username,password);
   return con;
}
}
```

DAOFactory.java

```
package com.axis2example.factory;
import com.axis2example.dao.ProductDAO;
import com.axis2example.dao.ProductDAOImpl;
public class DAOFactory {
private static ProductDAO productDAO;
static{
 productDAO=new  ProductDAOImpl();
}
public static ProductDAO getProductDAO(){
   return productDAO;
}
}
```

Write the services.xml file as follows

services.xml

```
<service name="Product" scope="application">
<messageReceivers>
    <messageReceiver  mep="http://www.w3.org/ns/wsdl/in-only"
  class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
    <messageReceiver  mep="http://www.w3.org/ns/wsdl/in-out"
  class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>

<parameter name="ServiceClass">
    com.axis2example.service.ProductServiceImpl
  </parameter>
</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.

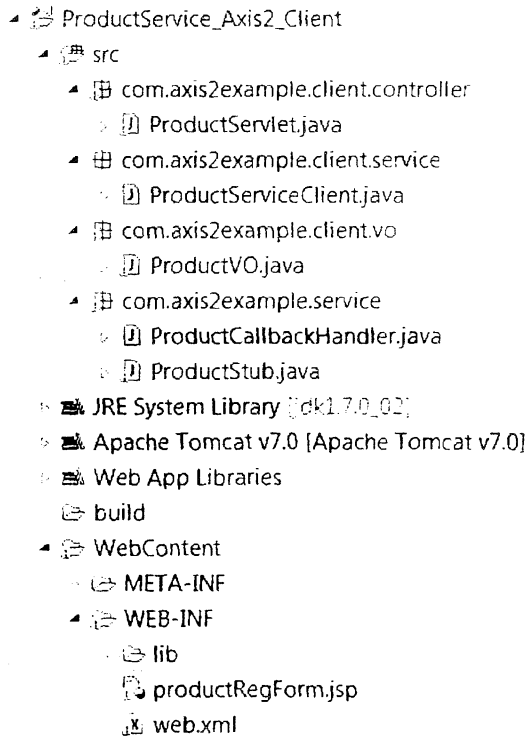
java2wsdl -cp build/classes -cn com.nit.service.ProductServiceImpl -of Product.wsdl of=output file name

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

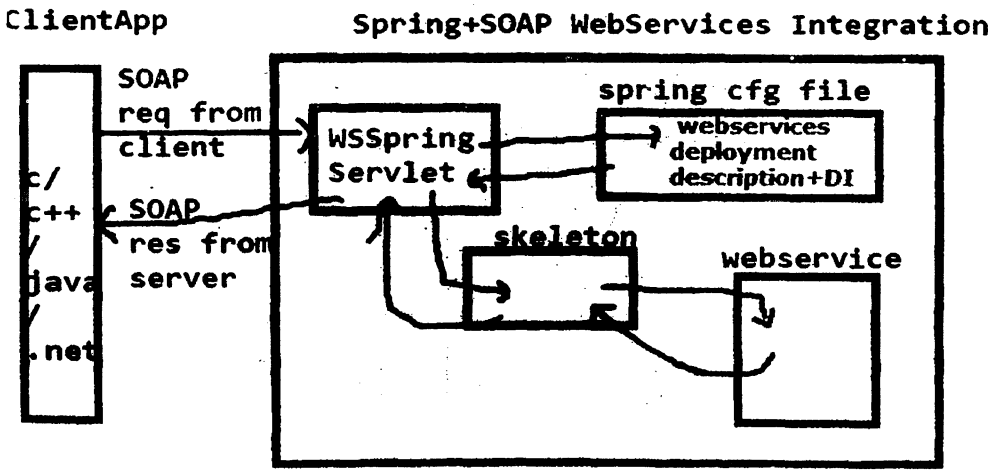
```
<web-app>
<servlet>
<servlet-name>ProductServlet</servlet-name>
<servlet-class>com.axis2example.client.controller.ProductServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>ProductServlet</servlet-name>
<url-pattern>/registerProduct</url-pattern>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

Spring +JAXWS WebServices (SOAP) Integration Example :

- SpringJaxWsExample
  - src/main/java
    - com.springjaxwsexample.dao
      - ProductDAO.java
      - ProductDAOImpl.java
    - com.springjaxwsexample.domain
      - Product.java
    - com.springjaxwsexample.service
      - ProductService.java
      - ProductServiceImpl.java
    - com.springjaxwsexample.ws
      - ProductServiceWs.java
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-1.7]
  - src
    - main
      - webapp
        - WEB-INF
          - applicationContext.xml
          - web.xml
    - test
  - target
  - pom.xml

WsServlet,WsSpringServlet given by Jax-Ws-RI/Metro

WsServlet used to integrate JavaWebApp+WebServices app
WsSpringServlet used to integrate springapp+WebServices app



pom.xml

```
<projectxmlns="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.springjaxwsexample</groupId>
<artifactId>SpringJaxWsExample</artifactId>
<packaging>war</packaging>
<version>1.0</version>
<name>SpringJaxWsExample MavenWebapp</name>
<url>http://maven.apache.org</url>
<repositories>
<repository>
<id>java.net</id>
<url>http://download.java.net/maven/2</url>
</repository>
</repositories>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-web</artifactId>
<version>3.2.0.RELEASE</version>
</dependency>
<!-- JAX-WS-RI -->
        <dependency>
                <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-rt</artifactId>
        <version>2.2.3</version>
        </dependency>
        <!-- Library from java.net, integrate Spring with JAX-WS -->
        <dependency>
                <groupId>org.jvnet.jax-ws-commons.spring</groupId>
                <artifactId>jaxws-spring</artifactId>
                <version>1.8</version>
                <exclusions>
                <exclusion>
                <groupId>org.springframework</groupId>
                        <artifactId>spring-core</artifactId>
        </exclusion>
        <exclusion>
```

```
<groupId>org.springframework</groupId>
                    <artifactId>spring-context</artifactId>
        </exclusion>
        <exclusion>
                <groupId>com.sun.xml.stream.buffer</groupId>
                    <artifactId>streambuffer</artifactId>
        </exclusion>
        <exclusion>
                <groupId>org.jvnet.staxex</groupId>
                    <artifactId>stax-ex</artifactId>
        </exclusion>
                </exclusions>
        </dependency>
        <dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.35</version>
</dependency>
<!-- https://mvnrepository.com/artifact/commons-pool/commons-pool -->
<dependency>
<groupId>commons-pool</groupId>
<artifactId>commons-pool</artifactId>
<version>1.6</version>
</dependency>
<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->
<dependency>
<groupId>commons-dbcp</groupId>
<artifactId>commons-dbcp</artifactId>
<version>1.4</version>
</dependency>
</dependencies>
<build>
<finalName>SpringJaxWsExample</finalName>
</build>
</project>
```

web.xml

```
<web-app>
<servlet>
        <servlet-name>jaxwsservlet</servlet-name>
        <servlet-class>
com.sun.xml.ws.transport.http.servlet.WSSpringServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
<servlet-name>jaxwsservlet</servlet-name>
<url-pattern>/registerProduct</url-pattern>
```

```
</servlet-mapping>
<!-- Register Spring Listener -->
        <listener>
        <listener-class>
 org.springframework.web.context.ContextLoaderListener
        </listener-class>
        </listener>
</web-app>
```

Note : ContextLoaderListener will create spring container.It searches the spring cfg file name as applicationContext.xml in WEB-INF folder

Product.java

```
package com.springjaxwsexample.domain;
import java.io.Serializable;
publicclassProductimplements Serializable{
private Integer pid;
private String productName;
private Double price;
public Integer getPid() {
        returnpid;
}
publicvoid setPid(Integer pid) {
        this.pid = pid;
}
public String getProductName() {
        returnproductName;
}
publicvoid setProductName(String productName) {
        this.productName = productName;
}
public Double getPrice() {
        returnprice;
}
publicvoid setPrice(Double price) {
        this.price = price;
}
}
```

ProductDAO.java

```
package com.springjaxwsexample.dao;
import com.springjaxwsexample.domain.Product;
publicinterface ProductDAO {
publicint registerProduct(Product product);
}
```

ProductDAOImpl.java

```
package com.springjaxwsexample.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.sql.DataSource;
import com.springjaxwsexample.domain.Product;
public class ProductDAOImpl implements ProductDAO{
private DataSource dataSource;//dependency
        public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
}
        publicint registerProduct(Product product) {
        intcount=0;
                try{
        Connection con=dataSource.getConnection();
String sql="insert into Product_Details values(?,?,?)";
PreparedStatement pst=con.prepareStatement(sql);
pst.setInt(1,product.getPid());
pst.setString(2,product.getProductName());
pst.setDouble(3,product.getPrice());
count=pst.executeUpdate();
                }catch(SQLException se){
                se.printStackTrace();
                }
                returncount;
        }
        }
```

ProductService.java

```
package com.springjaxwsexample.service;
import com.springjaxwsexample.domain.Product;
public interface ProductService {
public boolean registerProduct(Product product);
}
```

ProductServiceImpl.java

```
package com.springjaxwsexample.service;
import com.springjaxwsexample.dao.ProductDAO;
import com.springjaxwsexample.domain.Product;
/** writing of this service is an optional
 * in this class write Exception-Handling,Transaction-Management logic,loggers,Type-
Converstion,Bussiness logic's
 * if we are not writing this separate service class then we include above logic directly
in webservice class
```

```
*/
publicclass ProductServiceImpl implements  ProductService{
private ProductDAO productDAO;
public void setProductDAO(ProductDAO productDAO) {
        this.productDAO = productDAO;
}
public cboolean registerProduct(Product product) {
        booleanflag=false;
        intcount=productDAO.registerProduct(product);
        if(count>0){
                flag=true;
         }
                returnflag;
        }
}
```

ProductServiceWS.java

```
package com.springjaxwsexample.ws;
import javax.jws.WebMethod;
import javax.jws.WebService;
import com.springjaxwsexample.domain.Product;
import com.springjaxwsexample.service.ProductService;
@WebService
publicclass ProductServiceWs{
        private ProductService productService;
        @WebMethod(exclude=true)
public void setProductService(ProductService productService) {
        this.productService = productService;
}
@WebMethod(exclude=false)
        public Boolean registerProduct(Product product) {
                Boolean flag=productService.registerProduct(product);
                returnflag;
        }
}
```

applicationContext.xml

```
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ws="http://jax-ws.dev.java.net/spring/core"
xmlns:wss="http://jax-ws.dev.java.net/spring/servlet"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://jax-ws.dev.java.net/spring/core
     http://jax-ws.dev.java.net/spring/core.xsd
     http://jax-ws.dev.java.net/spring/servlet
     http://jax-ws.dev.java.net/spring/servlet.xsd">
```

```
<wss:bindingurl="/registerProduct">
<wss:service>
<ws:servicebean="#productServiceWs"/>
</wss:service>
</wss:binding>
<beanid="dataSource"class="org.apache.commons.dbcp.BasicDataSource">
<propertyname="driverClassName"value="com.mysql.jdbc.Driver"/>
<propertyname="url"value="jdbc:mysql://localhost:3306/nit"/>
<propertyname="username"value="root"/>
<propertyname="password"value="root"/>
</bean>
<beanid="productDAO"class="com.springjaxwsexample.dao.ProductDAOImpl">
<propertyname="dataSource"ref="dataSource"/>
</bean>
<beanid="productService"class="com.springjaxwsexample.service.ProductServiceImpl">
<propertyname="productDAO"ref='productDAO'/>
</bean>
<beanid="productServiceWs"class="com.springjaxwsexample.ws.ProductServiceWs">
<propertyname="productService"ref="productService"/>
</bean>
</beans>
```

Note : Test the above service by using SOAP UI tool.

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 onwords , 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

Not use real time only use spring jax rs integration using in real time

Difference between SOAP and Restfull webservices

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

What is a Resource?

In REST architecture, everything is a resource. These resources can be text files, html pages, images, videos or dynamic business data. REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs. REST uses various representations to represent a resource like text, JSON, XML. XML and JSON are the most popular representations of resources.

Representation of Resources

A resource in REST is similar Object in Object Oriented Programming. Once a resource is identified then its representation is to be decided using a standard format so that server can send the resource in above said format and client can understand the same format.

For example, in RESTful Web Services - User is a resource which is represented using **following XML format:**

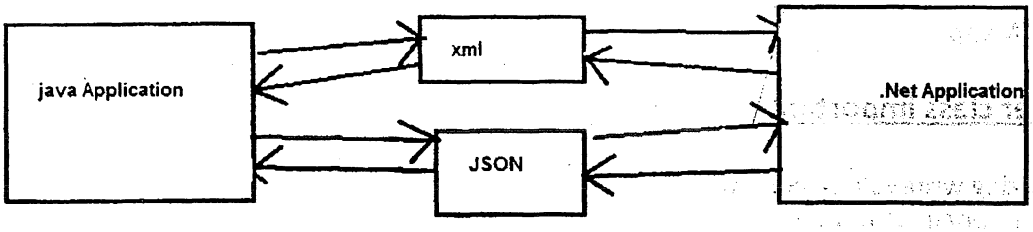
```
<user>
  <id>101</id>
  <name>Mahesh</name>
  <profession>Former</profession>
</user>
```

Same resource can be represented in JSON format:

```
{
  "id":101,
  "name":"Mahesh",
  "profession":"Former"
}
```

JSON:

- JSON stands for JavaScirpt 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
- JSONformat is very lightweight format compared to SOAP/XML.
- JSON a is text-based document

Ex:-

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

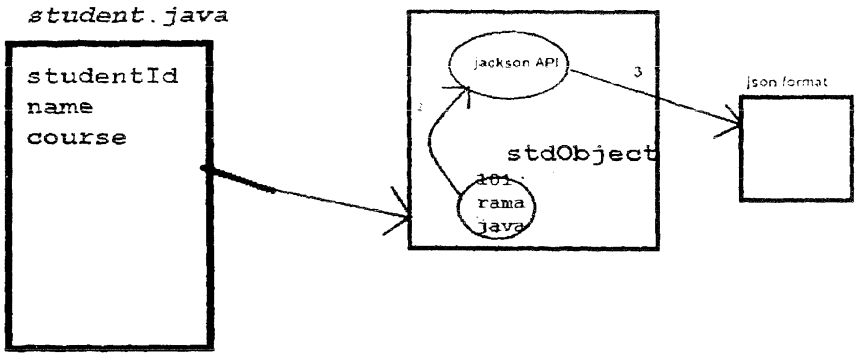
--> To read the JSON format data from the java application and to construct JSON format data by the java application we require JSON API.

Different vendorsare 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.

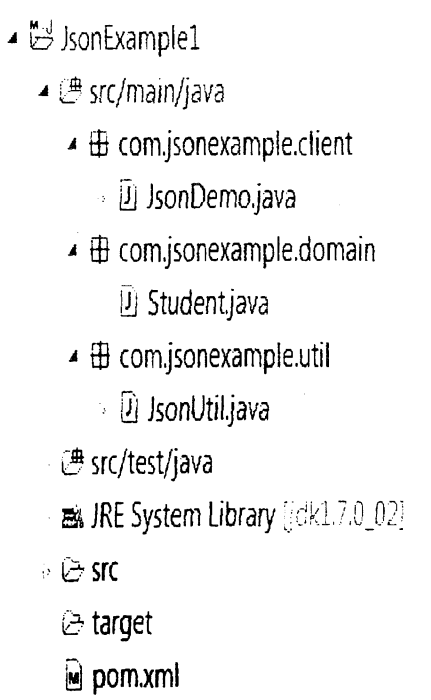
ObjectMapper class important methods

- 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>
```



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;
    }
}
```

JsonUtil.java

```
package com.jsonexample.util;
import java.io.IOException;
public class JsonUtil {                    (run time polymorphism)
public static String convertJavaToJson(Object obj){
    String json="{}";
ObjectMapper objectMapper=new ObjectMapper();
try{
json=objectMapper.writeValueAsString(obj);
}catch(JsonProcessingException e){
    e.printStackTrace();
}
return json;
}              (for downcosting using generics and it is runtime polymorphism)
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;
}
}
```

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
- 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 webservices 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 simple java classes that are annotated with @Path annotation.
- The Resource class methods are required to denote with resource method designator annotation such as @GET, @PUT, @POST, @DELETE.
- Resource methods are methods of a resource class annotated with a resource method designator annotations and exposing over the Network.

Root Resource class Example to produce JSON Response:

```
@Path("products")
public class ProductService{
@GET
@Path("getProduct")
@Produces(MediaType.APPLICATION_JSON) // indicates the java method will produces
content/response in JSON format
public Product searchProduct(){
//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;
   }
}
```

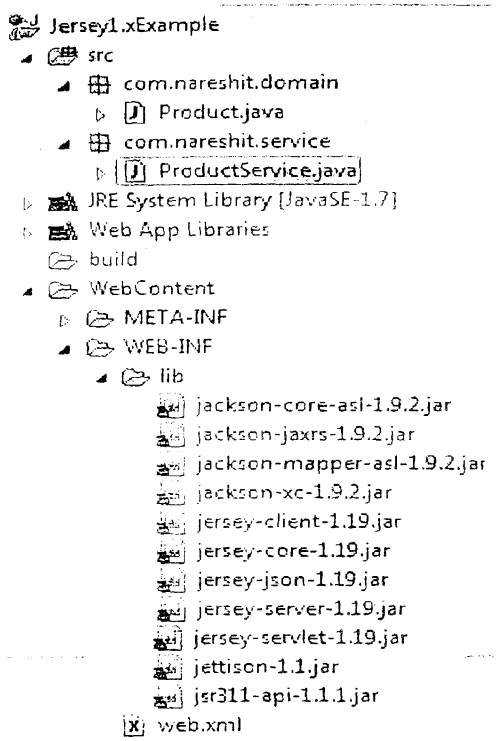
- The client is using the following URL to access the above Service
- **URL:-** http://<hostName>:<portNum>/context-root /products/searchProduct?pid=101
- The above URL shows the query param pid=101 where 101 value will be passed as an input to parameter pid of the method.
- The QueryParameters are name-value ~~paris~~ pairs.
- The QueryParameters are starts with ? in the URL and if multiple Query parameters are there separated with & symbol

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")) {

        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

→ The query parameters are ~~req~~ inputs value the query parameter are request param value



Product.java

```
package com.nareshit.domain;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Product {
private Integer pid;
private String pname;
private Double price;
public Integer getPid() {
        return pid;
}
public void setPid(Integer pid) {
        this.pid = pid;
}
public String getPname() {
        return pname;
}
public void setPname(String pname) {
        this.pname = pname;
}
public Double getPrice() {
        return price;
}
public void setPrice(Double price) {
        this.price = price;
}
}
```

ProductService.java

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
import com.nareshit.domain.Product;
@Path("products")
public class ProductService {
@GET
@Path("/getProductInXML")
@Produces(MediaType.APPLICATION_XML)
public Product  searchProduct1(@QueryParam("pid") Integer pid){
                System.out.println("Entered into searchProduct1 xml");
                Product p=null;
                if(pid!=null && pid.equals(301)){
        p=new  Product();
        p.setPid(pid);
        p.setPname("keyboard");
        p.setPrice(300.0);
        }
        return p;
}
@GET
@Path("/getProductInJson")
@Produces(MediaType.APPLICATION_JSON)
public Product  searchProduct2(@QueryParam("pid") Integer pid){
                System.out.println("Entered into searchProduct2 json");
                Product p=null;
                if(pid!=null && pid.equals(302)){
        p=new  Product();
        p.setPid(pid);
        p.setPname("mouse");
        p.setPrice(320.0);
        }
return p;
}
@GET
@Path("/getProductName")
@Produces(MediaType.TEXT_PLAIN)
public String getProductName(@QueryParam("pid") Integer pid){
        if(pid!=null && pid.equals(301)){
        return "keyboard";
        }
        return "Product not found";
        }
}
```

web.xml

```
<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>
```

Note : To Test The Above Service Application Either User SOAP UI Tool (OR) PostMan tool(OR) RestClient tool.

@PathParam:

In this case the input for the method parameters will be recived 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
```

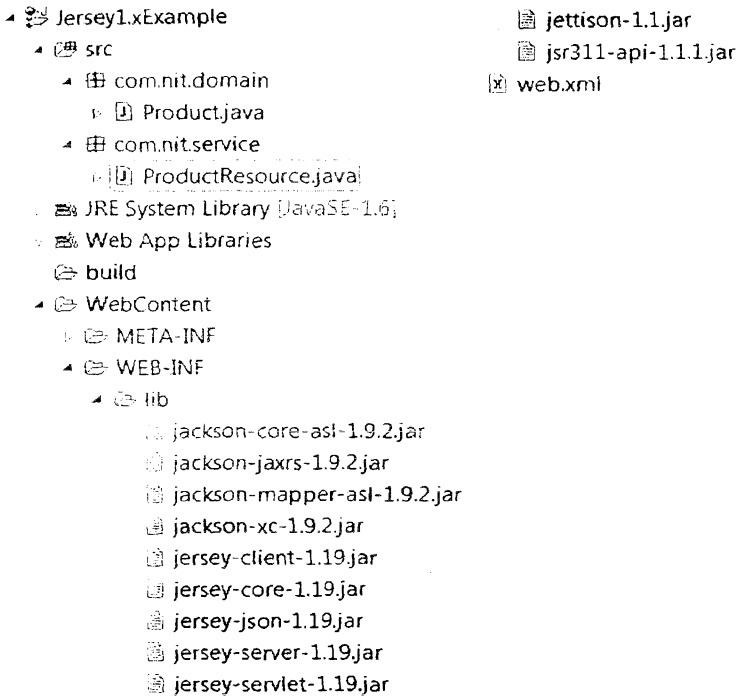
@MatrixParam:

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 starts with semicolon and if multiple parameters are there then separated with semicolon.
- The Matrix Parameters are Name-Value Pairs.

```
@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

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
```

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/RestClient tool (OR) write a client application.

The following is Example is showing how to test above webservice by using Jersey 1.x version client

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;
publicclass Test {
publicstaticvoid main(String[] args) {
String URL="http://localhost:8081/Jersey1.xExample/products/searchProductInJson;pid=101";
//create client object
Client client=Client.create();
//create WebResource object with request URL
WebResource webResource=client.resource(URL;)
//get the  Builder object fro WebResource
 Builder builder=webResource.accept(MediaType.APPLICATION_JSON);
ClientResponse response=builder.get(ClientResponse.class);
  String jsonResponse=response.getEntity(String.class);
//here entity is nothing result given by server
 System.out.println(response.getStatus()+" -->"+response.getStatusInfo());
    System.out.println(jsonResponse);
        }
}
```

```
Client client=Client.create();
string url="--------------------";
WebResource webResource=client.resource(url);
ClientResponse clientResponce=webResource.get(ClientResponse.class);
sysout(clientResponse.getStatus()+""+clientResponse.getStatusInfo());
String result=clientResponse.getEntity(String.class);
sysout(result);
```

@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.

web.xml

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>
```

ProductService.java

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";
}
}
```

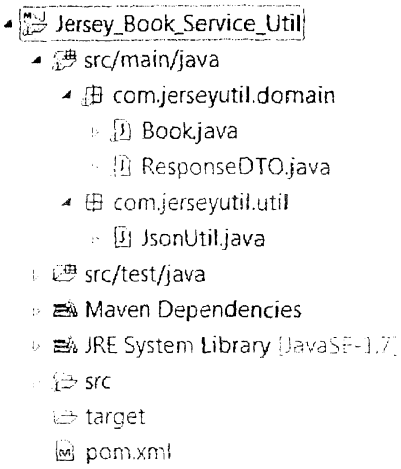
We can test above service by using the following client application As follows

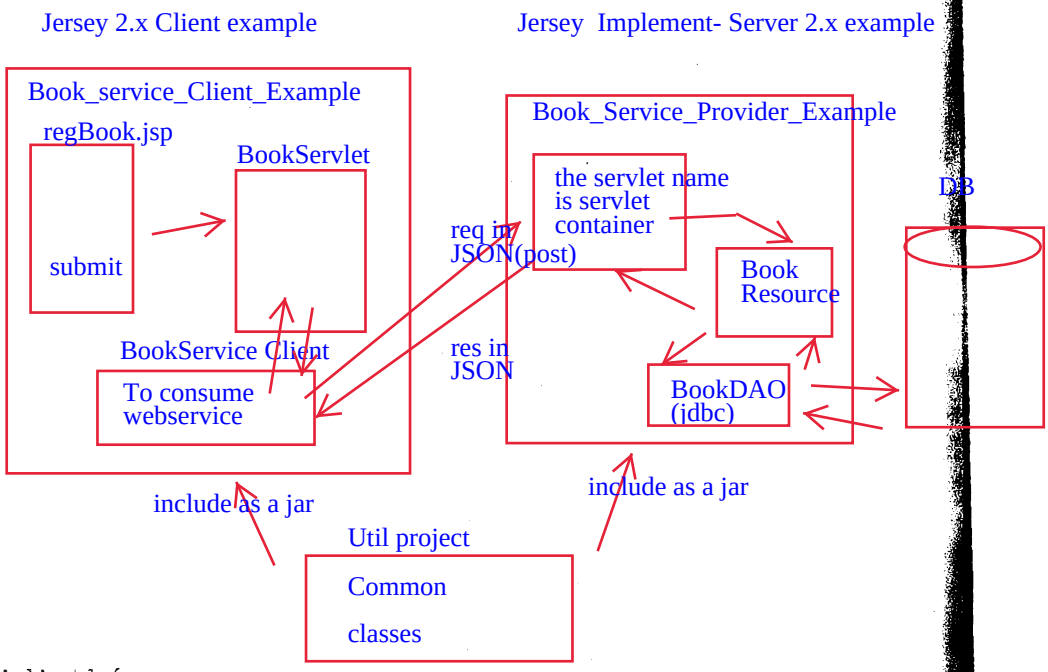
```
package com.nit.client;
import java.io.StringWriter;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import com.nit.domain.Product;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
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 = newStringWriter();
                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;
        }
}
```

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;
    }

}
```

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;
}
}
```

Jersey_Book_Service_Util/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/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>
```

- Jersey_Book_Service_Provider
  - 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 MavenWebapp</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>
<!-- https://mvnrepository.com/artifact/com.hynnet/oracle-driver-ojdbc6 -->
<dependency>
   <groupId>com.hynnet</groupId>
   <artifactId>oracle-driver-ojdbc6</artifactId>
   <version>12.1.0.1</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>Jersy</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>Jersy</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

<url-pattern>/*</url-pattern>
</servlet-mapping>

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;
}
}
```

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;
}
}
```

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;
}
}
```

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;
}
}
```

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 MavenWebapp</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);
}
}
```

Writable Smart Insert

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;
}
}
```

- The client obj is heavyweight object . It is always recomnded to close the client object after consuming the resource.
- The client obj Establish the connection b.w client app to server app.
- 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;
}
}
```

@FormParam :

In JAX-RS, you can use @FormParam annotation to bind HTML form parameters value to a Java method. The following example show you how to do it :

Note : @FormParam is used to bind html form fields to your method inputs. It works only for http method POST.

First create a HTML page with Form Parameters

a simple HTML form with "**post**" method.

UserForm.html

```
<html>
<body>
<h1>JAX-RS @FormQuery Testing</h1>
<form action="user/register" method="post">
Name : <input type="text" name="name" />
Age : <input type="text" name="age" />
<input type="submit" value="Add User" />
</form>
</body>
</html>
```

Create A Resource class to get Form Parameters

```
Example to use @FormParam and to get above HTML form parameter values.
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
@Path("user")
public class UserService {

        @POST
        @Path("/register")
    @Produces("text/plian")
        public String addUser(
                @FormParam("name") String name,
                @FormParam("age") int age) {

 return "addUser is called, name : " + name + ", age : " + age);

        }
}
```

Note :

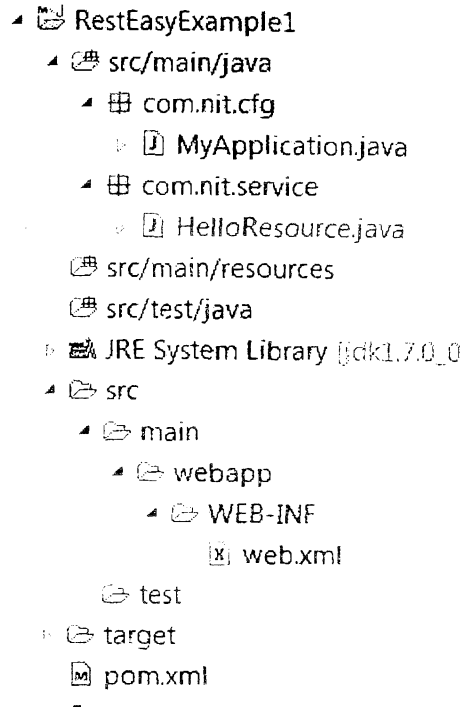
Access HTML Page. URL : *http://localhost:8080/RESTfulExample/UserForm.html*
Then click on Add User Button get the Out put.

RestEasy Implementation:

- RESTEasy 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 RESTEasy can run in any Servlet container,
- We can download the RestEasy jar's from http://resteasy.jboss.org/downloads

after downloading the jars . extract the zip file and copy the jar files from lib folder (copy all the jars except resteasy-cdi jar)

step1:- create RootResource class develop manually.(automatic created by jersey provided servlet)
step2:- create Resource Methods
step3:- create Application class ,
in this class register all the root-resources
step4:-Configure HttpServletDispatcher in web.xml file(supply the application class name as an init-param to servlet)



RestEasyExample1/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.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>
```

resteasy-jackson-provider and resteasy-jaxb-provider

3.0.10.final---used to get resteasy jar from maven central repository

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;
importjavax.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
                returnset;
}
}
```

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 with context-paramaeter in web.xml file to specify the prefix.

```
<!-- if the url pattern is except /* then we can configure with
<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

developing of the client app's by using Jersey and RestEasy little bit difficult.
to access the Restfull webservices in the easy manner spring f.w provided org.springframework.web.client.RestTemplate class
the RestTemplate class avoid memory leakeg problem and boiler plate coding problem
the RestTemplate class provide following methods to acess the restfull webservices

```
package com.nit.client;
import java.util.HashMap;
import java.util.Map;
importjavax.ws.rs.core.MediaType;
importorg.jboss.resteasy.client.ClientRequest;
importorg.jboss.resteasy.client.ClientResponse;
importorg.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);
}
}
```

public T getForObject
(String url,Class responseType)

public T getForObject
(String url,Class responseType,Object---params)

public T psotForObject(String url,Object
responseBody,Class responseType)

public T psotForObject(String url,Object
responseBody,Class responseType)

public T psotForObject(String url,Object
responseBody,Class responseType,Map params)

to get the spring jar's we can include the
following dependency in pom.xml

```
<group id> org.springFramework<>
<arti-->spring-web<>
<version>4.3.5.Release<>
```

to access restfullweb services:-----

```
public static void main(s a){
String result="";
String url="http://localhost:8082/RestEasyHelloExample2/
abc/hello/sayHello/{name}";
RestTemplate restTemplate=new RestTemplate();
try{
result=restTemplate.getForObject(url,String.class,"satish");
}catch(RestClientException e){
        result="Unable to process your request";
sysout(e);
}
sysout(result);
}
```

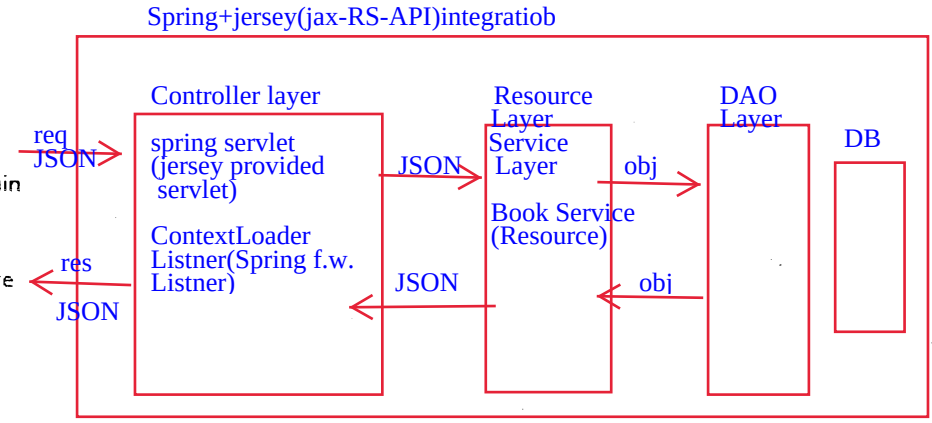
RestEasyClientApp/pom.xml

```
<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/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>
```

3.0.20.Final

Spring+RestFull Webservices (Jersey Implementation) Integration Example

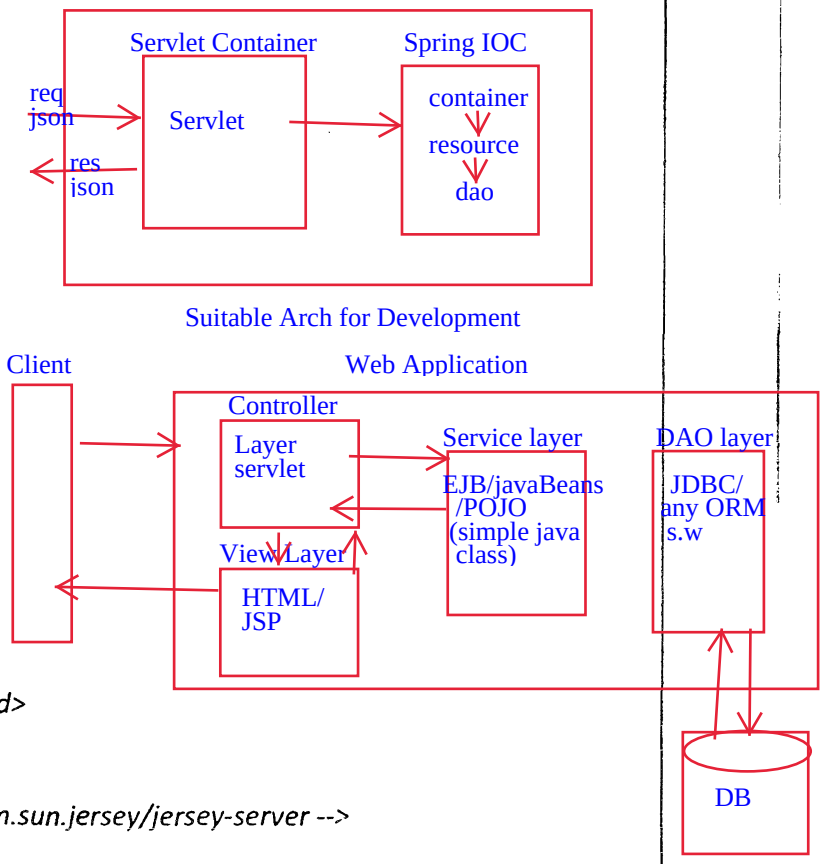
- SpringJerseyRestFullExample
  - src/main/java
    - com.jerseyexample.dao
      - BookDAO.java
      - BookDAOImpl.java
    - com.jerseyexample.domain
      - Book.java
      - ResponseDTO.java
    - com.jerseyexample.service
      - BookService.java
    - com.jerseyexample.util
      - JsonUtil.java
  - src/main/resources
  - src/test/java
  - Maven Dependencies
  - JRE System Library [jre7]
  - src
  - target
  - pom.xml



The contextloaderListner will creates spring IOC container and it store the spring IOC container in content scope.
Note:-the context loader listner by default searches the spring cfg file name applicationcontext.xml in WEB-INF folder
if the file name is different or file location is different then we an configure content parameter in web.xml

```
<projectxmlns="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.springjerseyexample</groupId>
<artifactId>SpringJerseyRestFullExample</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringJerseyRestFullExample MavenWebapp</name>
<url>http://maven.apache.org</url>
```

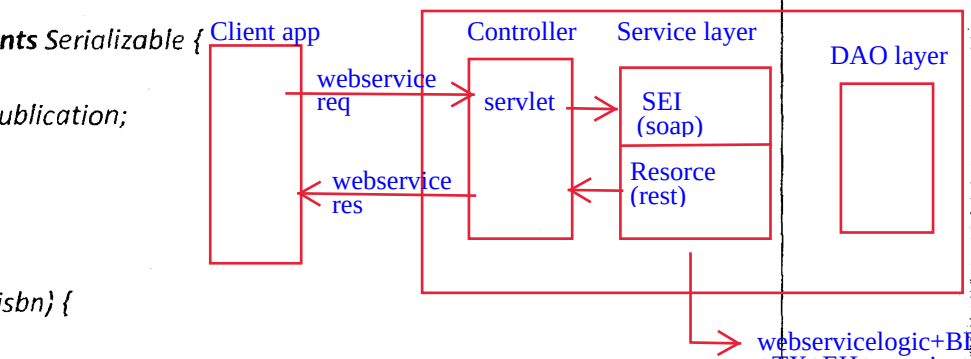
```
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-web</artifactId>
<version>3.0.5.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-jdbc</artifactId>
<version>3.0.5.RELEASE</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.35</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-server -->
<dependency>
<groupId>com.sun.jersey</groupId>
<artifactId>jersey-server</artifactId>
<version>1.8</version>
</dependency>
<dependency>
<groupId>org.codehaus.jackson</groupId>
<artifactId>jackson-mapper-asl</artifactId>
<version>1.9.13</version>
</dependency>
<dependency>
        <groupId>com.sun.jersey.contribs</groupId>
        <artifactId>jersey-spring</artifactId>
        <version>1.8</version>
        <exclusions>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
```



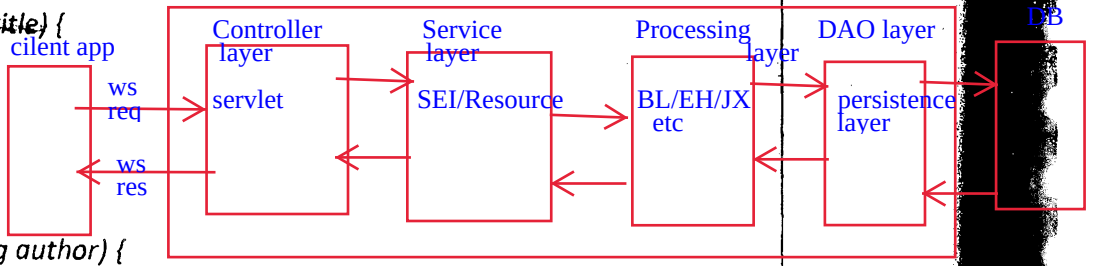
```
                        <artifactId>spring-web</artifactId>
                    </exclusion>
                    <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-aop</artifactId>
                    </exclusion>
                    <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-beans</artifactId>
                    </exclusion>
                    <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>

</dependencies>
<build>
<finalName>SpringJerseyRestFullExample</finalName>
</build>
</project>
```

Book.java

```
package com.jerseyexample.domain;
import java.io.Serializable;
public class Book implements Serializable {
private Integer isbn;
private String title,author,publication;
private Double price;
public Integer getIsbn() {
        return isbn;
}
public void setIsbn(Integer 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) {
```



To seperate webservices with other logics (BL) it recommonded to take two seperate layers
1-service layer
2-process layer/business layer



```
        this.author = author;
}
public String getPublication() {
        return publication;
}
public void setPublication(String publication) {
        this.publication = publication;
}
public Double getPrice() {
        return price;
}
public void setPrice(Double price) {
        this.price = price;
}

}
```

ResponseDTO.java

```
package com.jerseyexample.domain;
import java.io.Serializable;
public class ResponseDTO implements Serializable{
private byte status;
private String msg;
private String data;
public byte getStatus() {
        return status;
}
public void setStatus(byte status) {
        this.status = status;
}
public String getMsg() {
        return msg;
}
public void setMsg(String msg) {
        this.msg = msg;
}
public String getData() {
        return data;
}
public void setData(String data) {
        this.data = data;
}
}
```

BookDAO.java

```
package com.jerseyexample.dao;
import com.jerseyexample.domain.Book;
public interface BookDAO {
public int registerBook(Book book);
}
```

BookDAOImpl.java

```
package com.jerseyexample.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.jerseyexample.domain.Book;
@Repository
public class BookDAOImpl implements BookDAO{
        @Autowired
        private DataSource dataSource;
  public int registerBook(Book book) {
        int count=0;
         Connection con=null;
        try{
con=dataSource.getConnection();
String sql="insert into Book_Details values(?,?,?,?,?)";
PreparedStatement pst=con.prepareStatement(sql);
pst.setInt(1,book.getIsbn());
pst.setString(2,book.getTitle());
pst.setDouble(3,book.getPrice());
pst.setString(4,book.getAuthor());
pst.setString(5,book.getPublication());
count=pst.executeUpdate();
         }catch(SQLException se){
                se.printStackTrace();
         }finally{
        if(con!=null){
                try{
            con.close();
                }catch(SQLException se){
                se.printStackTrace();
                }
        }
         }
                return count;
        }
}
```

BookService.java

```
package com.jerseyexample.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;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.jerseyexample.dao.BookDAO;
import com.jerseyexample.domain.Book;
import com.jerseyexample.domain.ResponseDTO;
import com.jerseyexample.util.JsonUtil;
@Path("books")
@Service
publicclass BookService{
        @Autowired
private BookDAO bookDAO;
@POST
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/registerBook")
public String registerBook(String jsonBook){
        System.out.println("Entered into registerBook ::"+jsonBook);
Book book=JsonUtil.convertJsonToJava(Book.class,jsonBook);
String msg="Registration Is Failure!Please Try Again";
ResponseDTO responseDTO=new ResponseDTO();
responseDTO.setMsg(msg);
int count=bookDAO.registerBook(book);
if(count>0){
    msg="Registration Is Success";
        responseDTO.setMsg(msg);
        responseDTO.setStatus((byte)1);
        responseDTO.setData(book.getIsbn().toString());
  }
 System.out.println("Response of  registerBook ::"+responseDTO.getMsg());
 String  jsonResponseDTO=JsonUtil.convertJavaToJson(responseDTO);
return jsonResponseDTO;
}
}
```

JsonUtil.java

```
package com.jerseyexample.util;
import java.io.IOException;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
public class JsonUtil {
public static <T> T convertJsonToJava(Class<T> cls,String  json){
        T response=null;
        ObjectMapper mapper=new ObjectMapper();
        try {
                response=mapper.readValue(json,cls);
        } catch (JsonParseException e) {
                e.printStackTrace();
        } catch (JsonMappingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
return response;
}
public static String convertJavaToJson(Object obj){
        String jsonStr="";
        ObjectMapper mapper=new ObjectMapper();
        try {
                jsonStr=mapper.writeValueAsString(obj);
        } catch (JsonGenerationException e) {
                e.printStackTrace();
        } catch (JsonMappingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
        return jsonStr;
}
}
```

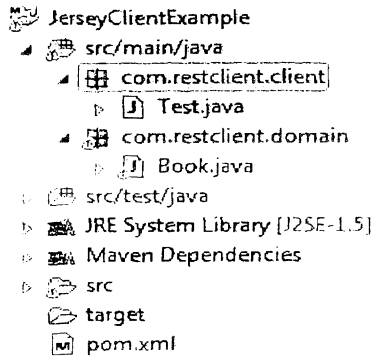
web.xml

```
<web-app>
<listener>
<listener-class>
org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
<servlet-name>jersey-serlvet</servlet-name>
<servlet-class>
com.sun.jersey.spi.spring.container.servlet.SpringServlet</servlet-class>
<init-param>
<param-name>com.sun.jersey.config.property.packages</param-name>
<param-value>com.jerseyexample.service</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>jersey-serlvet</servlet-name>
<url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>
```

applicationContext.xml

```
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<beanid="dataSource"class="org.springframework.jdbc.datasource.DriverManagerD
ataSource">
<propertyname="driverClassName"value="com.mysql.jdbc.Driver"/>
<propertyname="url"value="jdbc:mysql://localhost:3306/nit"/>
<propertyname="username"value="root"/>
<propertyname="password"value="root"/>
</bean>
<context:component-scanbase-
package="com.jerseyexample.service,com.jerseyexample.dao"/>
</beans>
```

The Following Client Application is used to Test above Webservice



pom.xml

```
<projectxmlns="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.restclient.client</groupId>
<artifactId>JerseyClientExample</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name> JerseyClientExample </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.jersey</groupId>
<artifactId>jersey-client</artifactId>
<version>1.19.1</version>
</dependency>
<dependency>
<groupId>org.codehaus.jackson</groupId>
<artifactId>jackson-mapper-asl</artifactId>
<version>1.9.13</version>
</dependency>
</dependencies>
</project>
```

Book.java

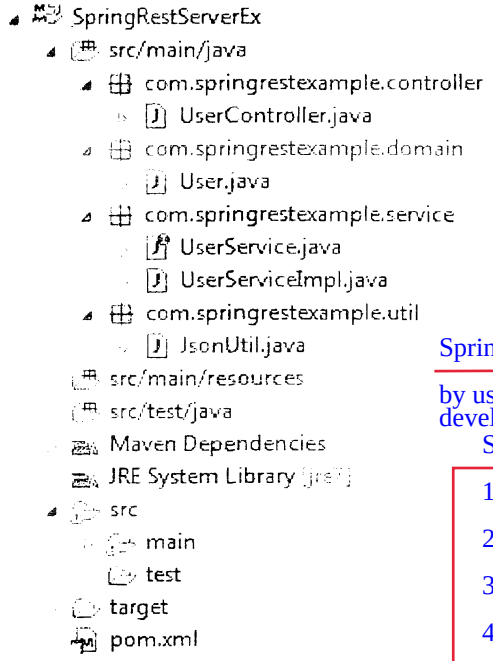
```
package com.restclient.domain;
import java.io.Serializable;
public class Book implements Serializable {
private Integer isbn;
private String title,author,publication;
private Double price;
public Integer getIsbn() {
        return isbn;
}
public void setIsbn(Integer isbn) {
        this.isbn = isbn;
}
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 String getPublication() {
        return publication;
}
public void setPublication(String publication) {
        this.publication = publication;
}
public Double getPrice() {
        return price;
}
public void setPrice(Double price) {
        this.price = price;
}
}
```

Test.java

```
package com.restclient.client;
import java.io.IOException;
import javax.ws.rs.core.MediaType;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import com.restclient.domain.Book;
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 JsonGenerationException,
JsonMappingException, IOException {
   String URL="http://localhost:8090/SpringJerseyRestFullExample/books/registerBook";
  Book book=new Book();
  book.setIsbn(302);
  book.setTitle("java");
  book.setPrice(1900.0);
  book.setPublication("nit");
  book.setAuthor("jhon");
  ObjectMapper mapper=new ObjectMapper();
  String jsonBook=mapper.writeValueAsString(book);
  System.out.println(jsonBook);
  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,jsonBook);
  System.out.println(clientResponse.getStatus()+" "+clientResponse.getStatusInfo());
System.out.println(clientResponse.getEntity(String.class));
   }
}
```

In jdbc it is possible to get the connection in two ways 1-by using drivermanager class getconnection() method
2-by using dataSource interface getconnection() method
drivermanager class getconnection() method always open a physical connection b.w java program to db. Due to this we face below problem
Each physical connection is hish expensive. hence the request processing cost is increase in this case
to open physical connection the application will spend more time. Hence performence of application is decrease in this case
To over come driverManager class getConnection(0 method problem sun micro system given DataSource interface

Spring Rest-API Examples :



JAX-RS API Annotation	SPRING_rest Annotations
@path	@RestController
@GET	@RequestMapping
@POST	
@Produces	@ResponseBody
@Consumes	@RequestBody
@QueryParam	@RequestParam
@PathParam	@PathVariable

Spring Rest:- developing of rest-full app's by using spring f.w:--

by using spring it is possible to develop web app. similarly by using spring it is possible to develop restfull webservices app..

Spring Web App annotations	Spring-restfull app annotations
1-@Controller	1-@RestController(4.0)
2-@RequestMapping	2-@RequestMapping
3-@RequestParam	3-@RequestParam (get to query-params)
4-@ModelAttribute	4-@PathVariable (get to get path -params)
	5-@RequestBody--used to get different Media type req
	6-@ResponseBody- used to produce Media type res

pom.xml

```
<projectxmlns="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.springrestexample</groupId>
<artifactId>SpringRestExample</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringRestExample MavenWebapp</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.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>4.0.5.RELEASE</version>
</dependency>
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>javax.servlet-api</artifactId>
```

```
<version>3.0.1</version>
</dependency>
<dependency>
<groupId>org.codehaus.jackson</groupId>
<artifactId>jackson-mapper-asl</artifactId>
<version>1.9.13</version>
</dependency>
</dependencies>
<build>
<finalName>SpringRestExample</finalName>
</build>
</project>
```

User.java

```
package com.springrestexample.domain;
publicclass User {
privateintuserId;
private String userName;
private String email,mobile;
publicint getUserId() {
        returnuserId;
}
publicvoid setUserId(intuserId) {
        this.userId = userId;
}
public String getUserName() {
        returnuserName;
}
publicvoid setUserName(String userName) {
        this.userName = userName;
}
public String getEmail() {
        returnemail;
}
publicvoid setEmail(String email) {
        this.email = email;
}
public String getMobile() {
        returnmobile;
}
publicvoid setMobile(String mobile) {
        this.mobile = mobile;
}
}
```

JAX-RS API Application:-

```
@Path("users")
public class UserResource{
@Path("getUser/{uid}")
@Produces(MediaType.Application_JSON)
@GET
public string getUserDetails(
@pathParam ("uid") Integer uid)
{
-----
----
return json response;
}
}
```

Spring-Rest Application:-

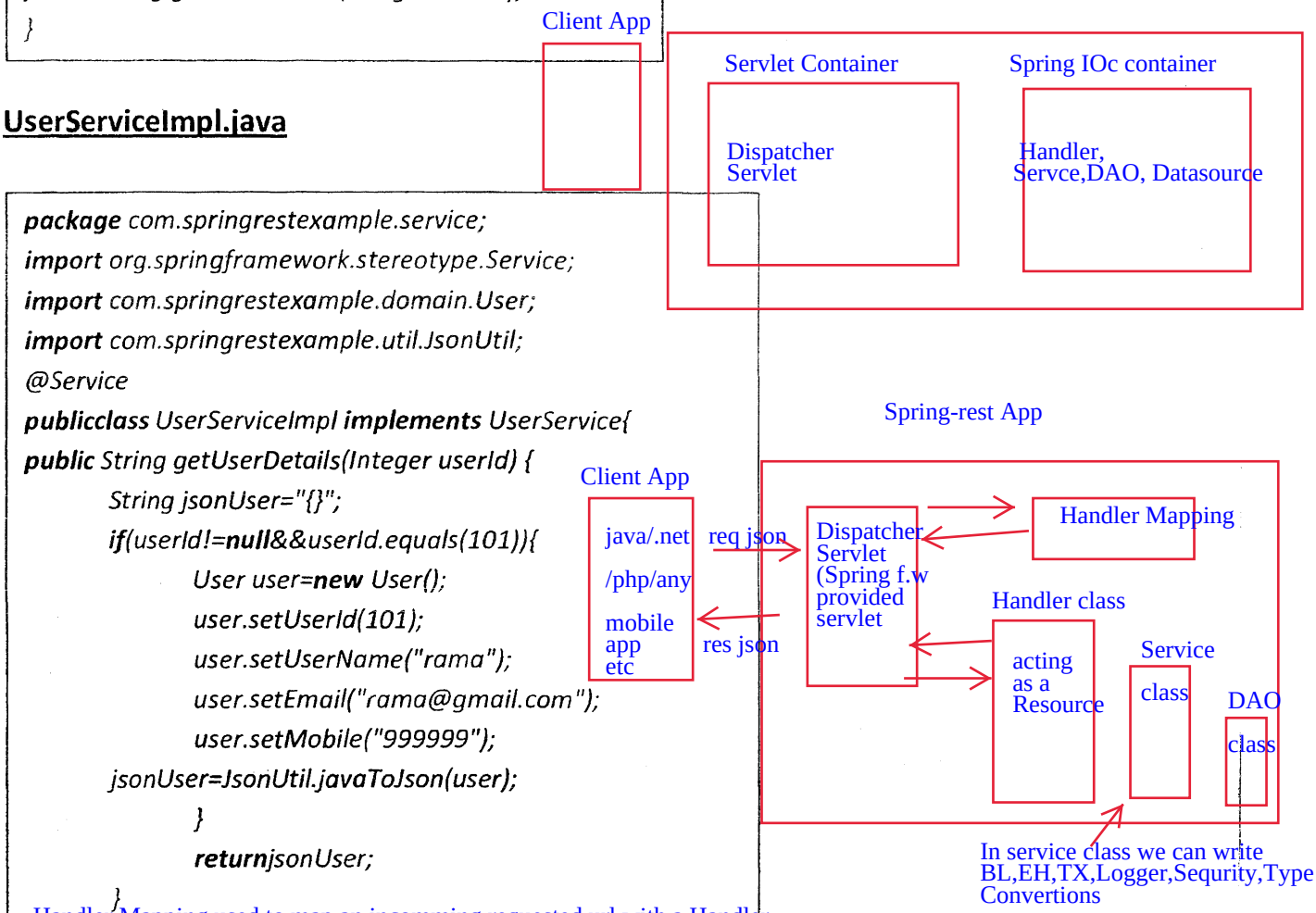
```
@RestController
@RequestMapping(value="users")
public class UserHandler{   //acting as a resource
@RequestMapping(value="getUser/{uid}" , method=RequestMethod.GET)
@ResponseBody
public String getUserDetails(
@pathVariable ("uid") Integer uid)
{
-----
----
return json response;
}
}
```

UserService.java

```
package com.springrestexample.service;
publicinterface UserService {
public String getUserDetails(Integer userId);
}
```

UserServiceImpl.java

```
package com.springrestexample.service;
import org.springframework.stereotype.Service;
import com.springrestexample.domain.User;
import com.springrestexample.util.JsonUtil;
@Service
publicclass UserServiceImpl implements UserService{
public String getUserDetails(Integer userId) {
        String jsonUser="{}";
        if(userId!=null&&userId.equals(101)){
                User user=new User();
                user.setUserId(101);
                user.setUserName("rama");
                user.setEmail("rama@gmail.com");
                user.setMobile("999999");
        jsonUser=JsonUtil.javaToJson(user);
                }
                returnjsonUser;
        }
}
```



Handler Mapping used to map an incomming requested url with a Handler

@requestMapping annotation is holding Handlermapping obj

the dispatcher servlet is using handler mapping to identify appropriate handler class for incomming handler object

the dispatcher servlet is creating the spring ioc container .a and dispacther servlet store the spring ioc containor in servlet context object.

JsonUtil.java

```
package com.springrestexample.util;
import java.io.IOException;
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
publicclass JsonUtil {
publicstatic String javaToJson(Object obj){
        String jsonStr="{}";
        ObjectMapper mapper=new ObjectMapper();
        try {
                jsonStr=mapper.writeValueAsString(obj);
        } catch (JsonGenerationException e) {
                e.printStackTrace();
        } catch (JsonMappingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }

        returnjsonStr;
}
publicstatic<T>T jsonToJava(Class<T>cls,String str){
        T response=null;
ObjectMapper mapper=new ObjectMapper();
        try {
                response=mapper.readValue(str,cls);
        } catch (JsonParseException e) {
                e.printStackTrace();
        } catch (JsonMappingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
        returnresponse;
}
}
```

UserController.java

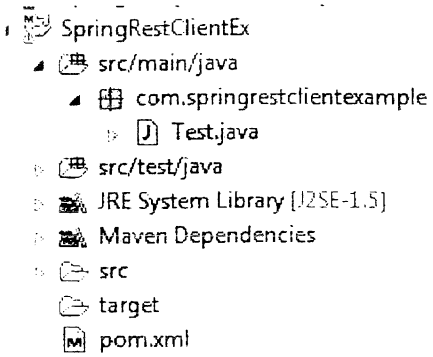
```
package com.springrestexample.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.springrestexample.service.UserService;
@RestController
publicclass UserController {
        @Autowired
        private UserService userService;
@RequestMapping(value="getUserInfo/{userId}",method=RequestMethod.GET)
@ResponseBody
public String getUserDetails(@PathVariable("userId") Integer userId){
         String jsonUser=userService.getUserDetails(userId);
        returnjsonUser;
}
}
```

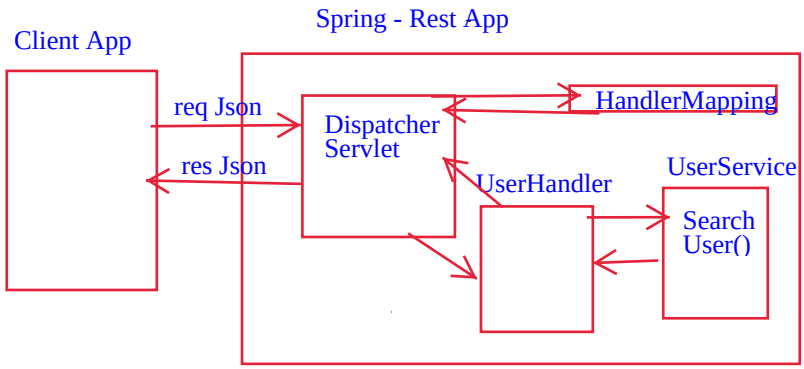
web.xml

```
<web-app>
<servlet>
<servlet-name>spring</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>spring</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

spring-servlet.xml

```
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:component-scan
base-package="com.springrestexample.controller,com.springrestexample.service"/>
</beans>
```





The RestTemplate class is given by spring framework to access the Restfull webservices in the easy manner.

Test.java

```
package com.springrestclientexample;
import java.util.HashMap;
import java.util.Map;
import org.springframework.web.client.RestTemplate;
publicclass Test{
publicstaticvoid main( String[] args ) {
  String url="http://localhost:8082/SpringRestServerEx/getUserInfo/{userId}";
   RestTemplate restTemplate=new RestTemplate();
   Map<String,Object>map=new HashMap<String, Object>();
map.put("userId",101);
   String jsonUser=restTemplate.getForObject(url,String.class,map);
   System.out.println(jsonUser);
   }
}
```

Response class:

- **Response class is present in javax.ws.rs.core package.**
- Defines the contract between a returned instance and the runtime when an application needs to provide metadata to the runtime.
- An application class can extend this class directly or can use one of the static methods to create an instance of this class .

The following Example is showing how to use Response class as a return type to resource method

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.Response;

@Path("hello")

public class HelloWorldService {

        @GET

        @Path("/sayHello/{name}")

        public Response getMsg(@PathParam("name") String name) {

                String output = "Hello  : " + name+"  Welcome ";

                return Response.status(200).entity(output).build();

        }

}
```

The following Example is showing how to use XML format data with Response obj

```
public class StudentService {
        @GET
        @Path("/getStudentDetails/{sid}")
        @Produces(MediaType.APPLICATION_XML)
        public Response getResult(@PathParam("sid) Integer sid){
                Student std=new Student();
                std.setName("sathish");
                std.setSid(sid);
        return Response.status(200).entity(std).build();
        }
}
```

Student.java

```
package com.nareshit.domain;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Student {
private Integer sid;
private String name;
//setters and getters
}
```

Get HTTP header in JAX-RS

We have two ways to get HTTP request header in JAX-RS:

1. Inject directly with @HeaderParam
2. Programatically via @Context

1. @HeaderParam Example

In this example, it gets the browser "user-agent" from request header.

```
Import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
@Path("/users")
public class UserService {
@GET
public Response  getHeadersResult(@HeaderParam("user-agent") String userAgent){
System.out.println("getHeaderResult method is calling and userAgent is :"+userAgent);
return Response.status(200).entity(userAgent).build();
}
```

2) Pragmatically via @Context Annnotation :

For @HeaderParam annotation Alternatively, you can also use @Context to get "javax.ws.rs.core.HttpHeaders" directly, see equivalent example to get browser "user-agent".

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import avax.ws.rs.core.Response;
@Path("/users")
public class UserService {
@GET
@Path("/get")
```

```
 @Produces("text/plain")
 public String getHeaders(@Context HttpHeaders headers) {
String userAgent = headers.getRequestHeader("user-agent").get(0);
return  "getHeaders is called, userAgent : " + userAgent;
}
}
```

List all request headers

You can also get list of available HTTP request headers via following code :

```
@Path("/users")
public class UserService {
 @GET
@Path("/getAll")
@Produces("text/plain")
public Response getAllHeaders(@Context HttpHeaders headers){
  MultivaluedMap<String,String> map=headers.getRequestHeaders();
    Set<String> keys=map.keySet();
  for(String key:keys){
        System.out.println("Header Name : "+key);
  List<String> headerValues=headers.getRequestHeader(key);
    for(String headerValue:headerValues){
   System.out.println(headerValue);
    }
  }
        return Response.status(200).build();
}
}
```

Note :

In general, @Context can be used to obtain contextual Java types related to the request or response.

(for example @Context HttpServletRequest request).

@CookieParam:

- Cookie is a name-value pair.
- In General Cookie is transferring from Server to client and client to server.
- The Cookie object is creating in Server System but stores in Client System.

We can retrieve the entire cookie by injecting the Cookie with

the @CookieParam annotation by providing the cookie name.

```
@GET
@Produces("text/plain")
public String getCookie(@CookieParam("cookie-name")  Cookie cookie){
System.out.println(cookie);
return "OK";
}
```

```
@Path("getAllCookies")
@GET
public Response getAllCookies(@Context HttpHeaders headers){
Map<String,Cookie> map=headers.getCookies();
Sysout(map.size());
return Responce.ok("OK").build();
}
```

We can also retrieve just the cookie value by injecting the value with the @CookieParam annotation.

```
@GET
@Produces("text/plain")
public String getCookieValue(@CookieParam("name") String cookie){
System.out.println(cookie);
return "OK";
}
```

Extracting Request Parameters:

Parameters of a resource method may be annotated with parameter-based annotations to get the values from a request obj.

You can extract the following types of parameters for use in your resource class:

- QueryParam
- URI path(PathParam)
- FormParam
- Cookie Param
- Header Param
- MatrixParam

Query parameters **are retrieved from the request URI query parameters and are specified by using the** javax.ws.rs.QueryParam **annotation in the method parameter arguments.**

The following example, from the Employee **application, demonstrates using of** @QueryParam annotation **to retrieve query parameters from the requested URL:**

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
@Path("employee")
public class EmployeeService {
 @GET
@Path("/getEmployeeDetails")
public Response getEmployee(@QueryParam("dept") String department,
@QueryParam("location") String location){
      String resp = "Query parameters are received. 'dept' value is: "
                  +department+" and location value is: "+location;
        return Response.status(200).entity(resp).build();
   }
}
```

In the above example, if you use ***"/employee/getEmployeeDetails?location=hyderabad&dept=finance"*** *URI pattern with query* parameters, getEmployee() method will be invoked, and you will get "**Query parameters as 'dept' value is: finance and location value is: hyderabad**" as a response.

To assign default values to method input variables if the query parameters are not available You can use @DefaultValue annotation to specify default value.

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
@Path("employee")
public class EmployeeService {
 @GET
@Path("/getEmployeeDetails")
public Response getEmployee (
@DefaultValue("accounts") @QueryParam("dept") String department,
@DefaultValue("bangalore") @QueryParam("location") String location){
      String resp = "Query parameters are received. 'dept' value is: "
                  +department+" and location value is: "+location;
returnResponse.status(200).entity(resp).build();
   }
}
```

In the above example, if you use "**/employee/getEmployeeDetails**" URI pattern, and you will get "**Query parameters are received. 'dept' value is: accounts and location value is: bangalore**" as a response.

Sometimes User-defined Java programming language types may be used as query parameters(OR) primitive Types used as a query parameters (OR) collection types used as a query parameters as per the requirements.

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
@Path("employee")
public class EmployeeService {
 @GET
@Path("/getEmployeeByDeptNo")
@Produces(MediaType.TEXT_PLAIN)
public String getEmployee(@QueryParam("deptNo") intdeptNo ){

return "Employee Name =raja ,Emp No = 101 , salary =2000 ,deptNo="+deptNo;
  }
 @GET
@Path("/getEmployeeByDeptName/{deptName}")
@Produces(MediaType.TEXT_PLAIN)
public String getEmployee(@PathParam("deptName") Employee emp){

return "Employee Name =raja ,Emp No = 101 , salary =2000
,deptName="+emp.getDeptName();
  }
@GET
@Path("/deleteDepartments")
@Produces(MediaType.TEXT_PLAIN)
public String deleteDepartments(@QueryParam("deptNo")  List<Integer>deptNoList){
System.out.println("List of Deprts :"+deptNoList);
return "Departments deleted ";
  }
}
```

http://localhost:8084/AppName/employee/deleteDepartments?deptNo=101&deptNo=102&deptNo=103 for collection

For Query-Parmeters and Matrix Parameter

The following code shows how to write Employee class for above example

```
public class Employee{
private String deptName;
public Employee(String deptName) {
this.deptName=deptName;
  }
public void setDeptName(String deptName){
this.deptName=deptName;
}
public String getDeptName(){
return deptName;
}
}
```

The constructor for Employee takes a single String parameter.

Note :Both @QueryParam and @PathParam can be used only on the following Java types:

- All primitive types except char
- All wrapper classes of primitive types except Character
- Any class with a constructor that accepts a single String argument
- Any class with the static method named valueOf(String) that accepts a single String argument
- List<T>, Set<T>, or SortedSet<T>, where *T* matches the already listed criteria. Sometimes, parameters may contain more than one value for the same name. If this is the case, these types may be used to obtain all values

From JAX -RS 2.0 version all primitive type including char and all wrapper types including Character also allows

Note :

If @DefaultValue is not used in conjunction with @QueryParam, and the query parameter is not present in the request, the value will be an empty collection for List, Set, or SortedSet; null for other object types; and the default for primitive types.

BeanParameters(By using @BeanParam annotation) (JAX-RS 2.0)):

```
package com.nareshit.bean;
public class EmployeeBean{
@QueryParam("salary")
private double salary;
@QueryParam("deptName")
private String deptName;
@QueryParam("location")
private String location;
//setters and getters
}
```

For Query- Parameters and matrix parameters sending of values are an optional and order is not important. Because these are name-value pairs
For Path-Parameters sending of values are mandatory and order is important

```
package com.nareshit.service;
@Path("/employee")
public class EmployeeService{
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/searchEmployees")
public String searchEmployees(@BeanParam EmployeeBean employeeBean){
return "empNo =101 ,empName=rama,salary="+employeeBean.getSalary();
}
}
```

Exception-Handling :

- When we send a request to the resource, the resource sometimes may throws Exceptions.
- If resource throws Exception , JAXRS runtime will convert that exception into an response message with status code as error response code.
- But if we want to return a custom error response rather than a default error response message we need to handle exceptions and should return as a response shown below.

UserNotFoundException.java

```
public class UserNotFoundException extends Exception {
        public UserNotFoundException() {
            super();
          }
         public UserNotFoundException (String msg) {
            super(msg);
          }
public UserNotFoundException (String msg, Exception e) {
super(msg, e);
          }
        }
```

UserService.java

```
@Path("users")
public class UserService {
 @GET
@Path("/getUser/{userId}")
@Produces(MediaType.APPLICATION_JSON)
public Response getUser(@PathParam("userId") String userId) throws
UserNotFoundException {
if(userId.equals("101")){
```

```
User user=new User();
user.setUserId(userId);
user.setName("sathish");
user.setEmail("sathish@gmail.com");
user.setMobile("8888889988");
return Response.status(200).entity(user).type(MediaType.APPLICATION_JSON).build();
 }
 else{
 throw new UserNotFoundException ("User does not exist with id " +    userId);
}
 }
}
```

This above method may throws an appropriate exception when a user cannot be found. However, we still need to create a Handler object to convert this exception into an actual JSON response so we get a nice friendly error message. The class below handles this exception and will convert the error message in the exception into a JSON response. The important annotation you'll see on this class i,e @Provider annotation.

UserNotFoundExceptionHandler :

```
@Provider
public final class UserNotFoundExceptionHandler implements
ExceptionMapper<UserNotFoundException> {
@Override
public Response toResponse(UserNotFoundException exception) {
return Response.status(Status.BAD_REQUEST)
                .entity(new ErrorMessage(exception.getMessage()))
.type(MediaType.APPLICATION_JSON).build();
        }
}
```

```
@provider
public class myexception implements
ExceptionMapper<Exception>{
@override
public Response to Response(Exception exception){
return
Response.status(Status.BAD_REQUEST).entity(exc
eption.getMesssage()).type(MediaType.APPLICAT
ION_JSON).build());
  }
}
```

the above exception handler machanism is declarative exceptional handling. if you are handaling the exception by try and catch blocks then that exception handling is called as programatic exception handaling.

Caching:

When we opening of a webpage for the first time takes some time, but the second or third time it loads faster. This happens because whenever we visit a webpage for the first time, our browser caches the content and need not have to make a call over the network to render it.

This caching ability of the browser saves a lot of network bandwidth and helps in cutting down the server load.

1.) Browser or local Caches: This is the local in-memory cache of a browser. This is the fastest cache available. Whenever we hit a webpage a local copy is stored in browser and then second time it uses this local copy instead of making a real request over the network.

2.) Proxy Caches: These are pseudo web servers that work as middlemen between browsers and websites.

These servers cache the static content and serve its clients so the client does not have to go to server for these resources.

Content Delivery Networks (CDN) are of similar concept where in they provide proxy caches from their servers , which helps in faster serving the content and sharing the load of servers.

Inconsistency and invalidation: When we are dealing with cache we need to make sure that the stale data is invalidated and it should be consistent. So let's see what all mechanisms are provided by HTTP which can help us make effective use of HTTP caching.

HTTP Headers

Before HTTP 1.1 the only way to control caching behavior was with the help of "expries" header. In this an expiry date can be specified.

But, later more powerful and extensive headers were introduced in HTTP 1.1, this was "cache-control" header. This along with "retag" gave the real power to the applications to control the behavior of caches.
JAX-RS supports these and provides APIs to use them. We will explore how we can leverage caching using HTTP response headers and the support provided by JAX-RS.

Expires Header:

In HTTP 1.0, a simple response header called Expires would tell the browser how long it can cache an object or page. It would be a date in future after which the cache would not be valid.
So, if we made an API call to retrieve data :

```
GET /users/1
The response header would be:
HTTP/1.1 200 OK
Content-Type: application/xml
Expires: Tue, 20 Dec 2016 16:00 GMT
-----
<user id="1">...</users>
```

- This means the XML data is valid until 20th Dec 2016, 16:00 hours GMT.
- JAX-RS supports this header in **javax.ws.rs.core.Response** object.

```
package com.cacheexample.service;

@Path("user")
public class UserService{
@Path("getUser/{uid}")
@GET
@Produces(MediaType.APPLICATION_XML)
```

```
public Response getUserXML(@PathParam("uid") Long uid){
User user=null;
if(uid.equals(101)){
user=new User();
user.setUserId(uid);
user.setUserName("sathish");
user.setEmail("sathish@gmail.com");
}
ResponseBuilder builder = Response.ok(user,MediaType.APPLICATION_XML); //Putting
expires header for HTTP browser caching.
Calendar cal = Calendar.getInstance();
cal.set(2016,12,25,16,0);
builder.expires(cal.getTime());
return builder.build();
}
Note : ResponseBuilder object build the Response object
```

But to support CDNs, proxy caches there was a need for more enhanced headers with richer set of features, having more explicit controls. Hence in HTTP 1.1 few new headers were introduced and Expires was deprecated. Let's explore them.

With Cache-Control :

Cache-Control has a variable set of comma-delimited directives that define who, how and for how long it can be cached. Let's explore few of them:

–**private/public** : these are accessibility directives, private means a browser can cache the object but the proxies or CDNs cannot and public makes it cacheable by all.

-no-cache, no-store,max-age are few others where name tells the story.

JAX-RS provides **javax.ws.rs.core.CacheControl** class to represent this header.

```
package com.cacheexample.service;
@Path("user")
public class UserService{

@Path("getUser/{uid}")

 @GET

@Produces(MediaType.APPLICATION_XML)

public Response getUserXMLwithCacheControl(@PathParam("uid") Long uid){
User user=null;
if(uid.equals(101)){
user=new User();
user.setUserId(uid);
user.setUserName("sathish");
user.setEmail("sathish@gmail.com");
}
```

```
CacheControl cc = new CacheControl();

cc.setMaxAge(300);// set the max-age cache control directive.

cc.setPrivate(true);

ResponseBuilder builder = Response.ok(user,MediaType.APPLICATION_XML);

 builder.cacheControl(cc);

return builder.build();

}
```

RESTful Web Service Security

Security Areas:

There are two main areas for securities.

Authentication: Process of checking the user, who they claim to be.

Authorization: Process of deciding whether a user is allowed to perform an activity within the application.

Different Authentications :

- ***BASIC Authentication***
    - It's simplest of all techniques and probably most used as well. You use login/password forms – it's basic authentication only. You input your username and password and submit the form to server, and application identify you as a user – you are allowed to use the system – else you get error.
    - The main problem with this security implementation is that credentials are propagated in a plain way from the client to the server. Credentials are merely encoded with Base64 in transit, but not encrypted or hashed in any way. This way, any sniffer could read the sent packages over the network.

DIGEST Authentication

This authentication method makes use of a hashing algorithms to encrypt the password (called **password hash**) entered by the user before sending it to the server. This, obviously, makes it much safer than the basic authentication method, in which the user's password travels in plain text that can be easily read by whoever intercepts it.

CLIENT CERT Authentication

- This is a mechanism in which a trust agreement is established between the server and the client through certificates. They must be signed by an agency established to ensure that the certificate presented for authentication is legitimate, which is known as CA.
- Using this technique, when the client attempts to access a protected resource, instead of providing a username or password, it presents the certificate to the server. The certificate contains the user information for authentication including security credentials, besides a unique private-public key pair. The server then determines if the user is legitimate through the CA. Additionally, it must verify whether the user has access to the resource. This mechanism must use HTTPS as the communication protocol as we don't have a secure channel to prevent anyone from stealing the client's identity.

Ways to Secure RestFullWeb Services:

You can secure your Restful Web services using one of the following methods to support authentication, authorization, or encryption:

1) Using **web.xml** deployment descriptor to define security configuration.

2) Using the **javax.ws.rs.core.SecurityContext** interface to implement security programmatically.

3) By Applying annotations to your JAX-RS classes.

Securing RESTful Web Services Using web.xml

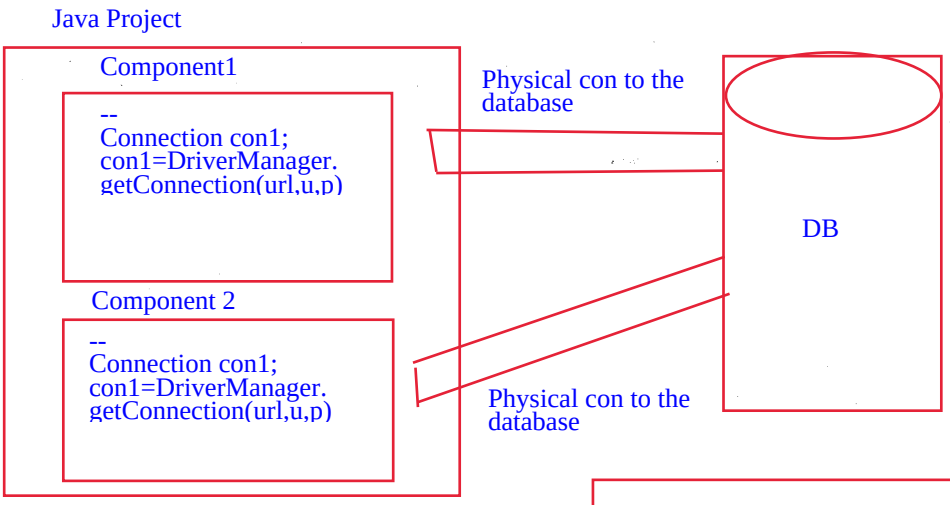
- You secure RESTful Web services using the **web.xml** deployment descriptor as you would for other Java EE Web applications.
- For example, to secure your RESTful Web service using basic authentication, perform the following steps:

1. Define a **<security-constraint>** for each set of RESTful resources (URIs) that you plan to protect.
2. Use the **<login-config>** element to define the type of authentication you want to use and the security realm to which the security constraints will be applied.
3. Define one or more security roles using the **<security-role>** tag and map them to the security constraints defined in step 1.
4. To enable encryption, add the **<user-data-constraint>** element and set the **<transport-guarantee>** subelement to **CONFIDENTIAL**

Example 1 Securing RESTful Web Services Using Basic Authentication

Step1 : create Root Resource class as for your requirement :

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
@Path("products")
public class ProductService {
@GET
@Path("/getProductName")
@Produces(MediaType.TEXT_PLAIN)
public String getProductName(
                @QueryParam("pid") Integer pid){
        if(pid!=null && pid.equals(301)){
        return "keyboard";
        }
        return "Product not found";
        }
}
```



Step2:- Configure The Security Configuration in web.xml file as follows

```
<web-app>
<servlet>
<servlet-name>RestServlet</servlet-name>
<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RestServlet</servlet-name>
<url-pattern>/*</url-pattern>
</servlet-mapping>
<security-constraint>
<web-resource-collection>
<web-resource-name>Products</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>
<login-config>
<auth-method>BASIC</auth-method>
</login-config>
<security-role>
<role-name>ADMIN</role-name>
</security-role>
</web-app>
```

Step3:- Set username and password and Role's in Tomcat Server

Changes in Tomcat Server (**tomcat-users.xml)**

As stated above we are using Tomcat 7. Now for the user to be authenticated we will specify the role 'ADMIN' (***which is the role chosen in our*** web.xml ***above***) and some username and password in our container. This username and password will have to be supplied to access the restricted resource.

tomcat-users.xml

```
<tomcat-users>
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<role rolename="admin"/>
```

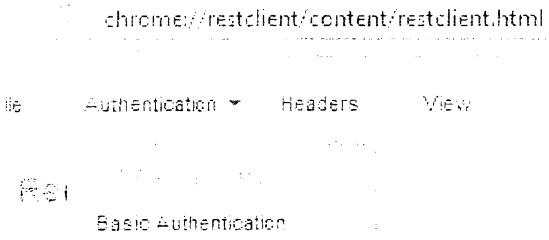
```
<role rolename="customer"/>
<user username="customer" password="customer" roles="customer"/>
<user username="admin" password="admin" roles="admin"/>
<user username="admin" password="admin" roles="admin-gui,manager-gui"/>
</tomcat-users>
```

Step4:- Deploy the Project into the server

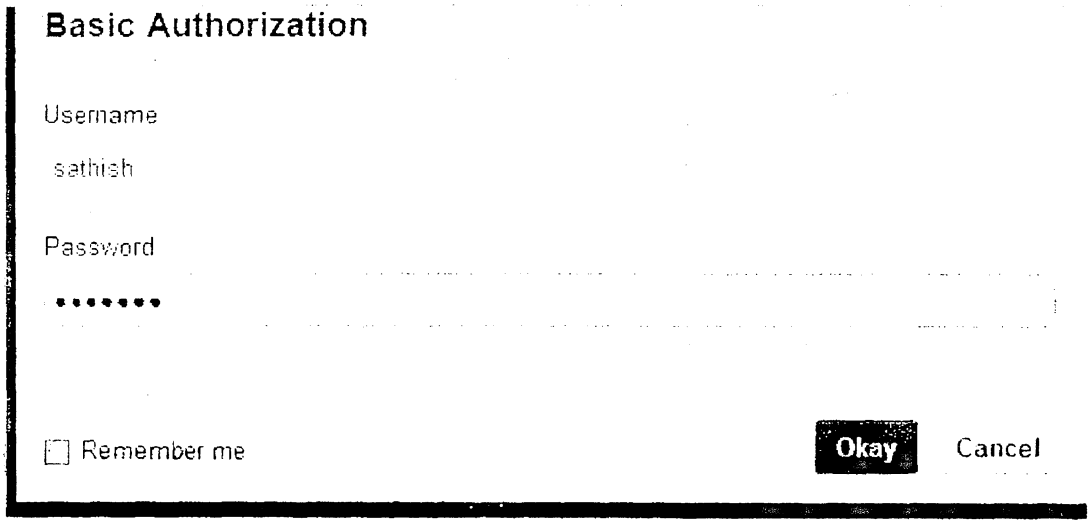
Note : Test the above Service By using Any RestClient tool.

While sending the request set username and password as follows.

First Click on Authentication in the client tool→Basic Authentication

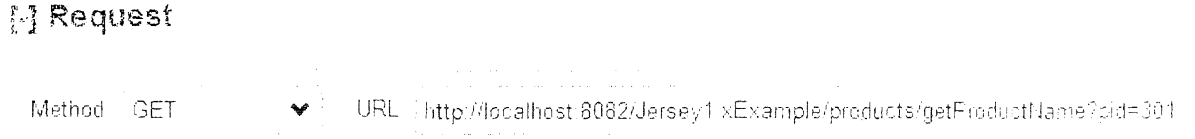


Enter UserName and password

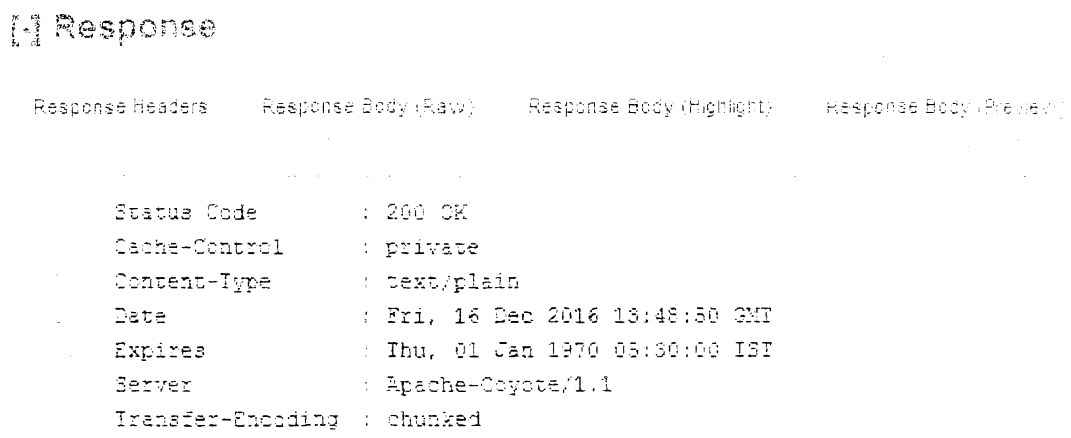


DataSource:- DataSource is an interface,present in javax.sql package .DataSource object is acting as a mediator b.w. java app and database DataSource is used to get the connection from Connection pool. for DataSource interface different vendor provided implementation provided like spring,apache ect . spring frame work provided class name:- org.springframework.jdbc.datasource.drivermanagerDataSource
If you are working with spring we can configure DataSource bean as follwos
<bean id="datasource" class="org.apache.commons.dbcp.basicDataSource">
<property name="driverClassName""value="---""/>
<property name="driverClassName""value="---""/>
<property name="driverClassName""value="---""/>
with the above cfg's connection pool and <property name="driverClassName""value="---""/> creating and managing by spring IOC container. but is also possible to work with server managed connection pool in spring and possible to get <property name="driverClassName""value="---""/> obj from JNDI Registry.
The server administrater will create the connection pool in the server. after creating the connection pool in the server servver administrator will link datasource object with the connection pool. after that datasource object binding into a jndi registry with jndi name . the application developer will lookup the datasource object from jndi registry and gets the connections with the help of datasource object
In spring to get any object from jndi registry we can use Jndi object factory bean.

Enter the URL and send



Output :



2) Securing RESTful Web Services Using SecurityContext

The **javax.ws.rs.core.SecurityContext** interface provides access to security-related information for a request. The **SecurityContext** provides functionality similar to **javax.servlet.http.HttpServletRequest**, enabling you to access the following security-related information:

- **java.security.Principal** object containing the name of the user making the request.
- Authentication type used to secure the resource, such as BASIC_AUTH, FORM_AUTH, and CLIENT_CERT_AUTH.
- Whether the authenticated user is included in a particular role.
- Whether the request was made using a secure channel, such as HTTPS.

You access the **SecurityContext** by injecting an instance into a class field, setter method, or method parameter using the **javax.ws.rs.core.Context** annotation.

The following shows how to inject an instance of **SecurityContext** into the **sc** method parameter using the **@Context** annotation, and check whether the authorized user is included in the **admin** role before returning the response.

Example 2 Securing RESTful Web Service Using SecurityContext

```
package com.nareshit.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.SecurityContext;
@Path("products")
public class ProductService {
@GET
@Path("/getProductName")
@Produces(MediaType.TEXT_PLAIN)
public String getProductName(
                @QueryParam("pid") Integer pid,
                @Context SecurityContext sc) {
        if (sc.isUserInRole("admin")){
        if(pid!=null && pid.equals(301)){
        return "keyboard";
        }
        else{
        return "Product Not Found";
        }
}
}
}
```

3) Securing RESTful Web Services Using Annotations

The **javax.annotation.security** package provides annotations, defined in Table 1, that you can use to secure your RESTful Web services.

Table 1 Annotations for Securing RESTful Web Services

DeclareRoles	Declares roles.
DenyAll	Specifies that no security roles are allowed to invoke the specified methods.
PermitAll	Specifies that all security roles are allowed to invoke the specified methods.
RolesAllowed	Specifies the list of security roles that are allowed to invoke the methods in the application.
RunAs	Defines the identity of the application during execution in a J2EE container.

In The following Example getProductName method is annotated with the **@RolesAllowed** annotation to override the default and only allow users that belong to the **ADMIN** security role.

Example 3 Securing RESTful Web Service Using Annotations(javax.annotation.security)

```
package com.nareshit.service;
import javax.annotation.security.RolesAllowed;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
@Path("products")
@RolesAllowed({"admin","customer"})
public class ProductService {
@GET
@Path("/getProductName")
@Produces(MediaType.TEXT_PLAIN)
@RolesAllowed("admin")
public String getProductName(
        @QueryParam("pid") Integer pid) {
```

```
        if(pid!=null && pid.equals(301)){
        return "keyboard";
        }
        else{
        return "Product Not Found";
        }
}
}
```

Realm :

- A realm is a repository of user information; it is an abstraction of the data store – text file, JDBC database or a JNDI resource. This has the following information: username, password and the roles which are assigned to the users.
- Both of the authentication and authorization make up the security policy of a server. Tomcat uses realms to implement container-managed security and enforce specific security policies.

Conculsion :

REST Web Service :

REST stands for Representational State Transfer. REST was a term coined by Roy Fielding in his doctoral dissertation. It is an architecture style for for creating network based applications. Key properties of REST are client-server communication, stateless protocol, cacheable, layered implementation and uniform interface

As RESTful web services work with HTTP URLs Paths so it is very important to safeguard a RESTful web service in the same manner as a website is be secured. Following are the best practices to be followed while designing a RESTful web service.

- **Validation** - Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks.
- **Session based authentication** - Use session based authentication to authenticate a user whenever a request is made to a Web Service method.
- **No sensitive data in URL** - Never use username, password or session token in URL , these values should be passed to Web Service via POST method.
- **Restriction on Method execution** - Allow restricted use of methods like GET, POST, DELETE. GET method should not be able to delete data.
- **Validate Malformed XML/JSON** - Check for well formed input passed to a web service method.
- **Throw generic Error Messages** - A web service method should use HTTP error messages like 403 to show access forbidden etc.

S.N. HTTP Code & Description

1 **200**
OK, shows success.

2	**201** **CREATED**, when a resource is successful created using POST or PUT request. Return link to newly created resource using location header.
3	**204** **NO CONTENT**, when response body is empty for example, a DELETE request.
4	**304** **NOT MODIFIED**, used to reduce network bandwidth usage in case of conditional GET requests. Response body should be empty. Headers should have date, location etc.
5	**400** **BAD REQUEST**, states that invalid input is provided e.g. validation error, missing data.
6	**401** **UNAUTHORIZED**, states that user is using invalid or wrong authentication token.
7	**403** **FORBIDDEN**, states that user is not having access to method being used for example, delete access without admin rights.
8	**404** **NOT FOUND**, states that method is not available.
9	**409** **CONFLICT**, states conflict situation while executing the method for example, adding duplicate entry.
10	**500** **INTERNAL SERVER ERROR**, states that server has thrown some exception while executing the method.

JAX-RS Annotations :

Following are the commonly used annotations to map a resource as a web service resource.

S.N.	Annotation & Description
1	**@Path** Relative path of the resource class/method.
2	**@GET** HTTP Get request, used to fetch resource.
3	**@PUT** HTTP PUT request, used to create resource.
4	**@POST** HTTP POST request, used to create/update resource.
5	**@DELETE** HTTP DELETE request, used to delete resource.
6	**@HEAD** HTTP HEAD request, used to get status of method availability.
7	**@Produces** States the HTTP Response generated by web service, for example APPLICATION/XML, TEXT/HTML, APPLICATION/JSON etc.
8	**@Consumes**

States the HTTP Request type, for example application/x-www-form-urlencoded to accept form data in HTTP body during POST request.

9 **@PathParam**
Binds the parameter passed to method to a value in path.

10 **@QueryParam**
Binds the parameter passed to method to a query parameter in path.

11 **@MatrixParam**
Binds the parameter passed to method to a HTTP matrix parameter in path.

12 **@HeaderParam**
Binds the parameter passed to method to a HTTP header.

13 **@CookieParam**
Binds the parameter passed to method to a Cookie.

14 **@FormParam**
Binds the parameter passed to method to a form value.

15 **@DefaultValue**
Assigns a default value to a parameter passed to method.