Diagram

Description automatically generated

Web Services : It is used to share data between two application.

Need : If let’s say we have two bank ATm like SBI and ICIC if sbi customer wants to take money from ICIC Bank atm then we used webservice . Icic bank will not interact with sbi bank database so that icic create web service or api for comminucated with sbi application and send details of customer to sbi application for verify after that sbi send acknowledgement and icic bank gives money to sbi cutomers.

We have old technology that perform same task:

1. SP – Socket programming :

2. RMI- Remote method invocation

3. EJB – Enterprise javabeans

This three is used to share data but only if two application is a java application.

4. Dcom - Distributed Component Object Model

This is used to share data but only if two application is a dot net application.

5. Remote procedure call

This is used to share data but only if two application is a C and C++ application.

6. Corba - Common Object Request Broker Architecture

This is used to share data with two application in any language. There is no dependency between two application.

It is failed now days because if let’s one application is made in C++ and second is in java then corba will give specific implementations of each language. Then that application can used corba implementation. But it is not providing proper implementation in some of cases it is not able to share data so we need webservies.

Webservice :

It is specification that has some set of rules and guidelines which follows to share data between two interoperable application.

It is not API.

XML – WSDL (webservices description language) – it is universal language that all application used to get webservices data from service provider.

But Who will write this XML document:

WSDL Generation tool – used to generated xml file which takes two input that your service class details and your end point url.

How to share this xml file to client like C++ application

1. By attaching email with wsdl file

If your web service is not public only want to share one application then used this.

2. Get address from UDDI

Diagram

Description automatically generated

UDDI **stands for Universal Description, Discovery, and Integration**

Here UDDI is maintain by third party. And this used by all application to get web services. If we want our web services public then used UDDI.

After getting WSDL file by c++ application -> we have Proxy generation tool will used to get input from wsdl file -> after that create SUTUBS here predefine classes for your client application. -> this class object need to create inside c++ application.

Stubs will create request xml file to destination with having soap tag – simple object access protocol it is used to define client request data inside xml – after that used transfer to destination we used HTTP protocol – in between we have class which used to get soap request from client – skeleton class used to send response back to client – again soap tag – again http protocol- In between STUBS used to generated response fro c++ object- c++ object access data.

Components:

1. Skelton class- in webservice we have only one skelton class used to manage multiple web service . It used to get request and give back response. It used to implemented by servlet need to config inside web.xml

2. WSDL – used to get your class methods data and url and used with wsdl generation tool (input – end point url and class data -used to generated xml for serviceprovider). Every webservices has unique wsdl.

3. UDDI – used to create description of wsdl file and client cn get wsdl frile from here

4. STUBS/proxies – it is depends on client if client is c++ then stubs is in c++ language. It is predefine class generated by stubs generation tools

5. SOAP – it used to created xml file for your stubs by soap tag.

6. HTTPS – this protocol used to transfer from client to destination.

API – set of interface and classes together.

Wsdl and skleteon and STUBS uses java webservices api to implemented predefine class.

* JAX-RPC (java api for xml – remote procedure call) – synch- jdk 1.4
* JAX-M(‘’’’ - message) – asynch – jdk 1.4
* JAX-WS - (“” – web services) – synch – jdk 1.5
* JAX-RS – (“” – restful webservices) – synch – jdk -1.6

This all are java webservices api but it is specification means just interface not used for implementation.

We have two types of web services like synchronized and asysnchronized web services.

Synchronized web services – when client request to service provider and client gives immedidated response then it call this.

Asychronized web services- when client send request to service provider then service wait some time mean while client is doing his work and then after some thime service provider send then it call this.

We will not used asynch web service.

Sych has two web services-

Soap – jax – rpc (Not used) , jax-ws

Restful - jax-rs

This all are the interface only. We need implementation.

Jax-rpc-si - > java api for xml remote procedure call sun implementation – need to download jar file it is not come with JDK

(SI, Axis 1, 🡪 server independent - > si given by sun and axis 1 is given apache

web logic, web server, jbose 🡪 server dependent (Not used) given by sun microsysytem)

Jax-ws :

RI- reference implementation – given by sun – it is partial

Metro – given by sun – fully implementation of jax-ws

Axis 2 – by apache – spring not supported

Apche cxf – by apache – spring supported

Web logic, web server , JBose – server dependent

Jax- rs

Jersey – from sun

Rest easy – from red hat

Rest let – from Jerome louvel developer

Apache wink – from apche – spring not supported

Apache cxf – spring supported

Jax-RPC 🡪 Use Axis 1 webservices

--🡪 Service provider creates webservices

1. create Calservices

2. Calservices - right click – webserver runtime apache axis – finish (it is generated automatically wsdl file)

3. Your skeleton - org.apache.axis.transport.http.AxisServlet

url for skeleton - > /services/\*

Text

Description automatically generated

* Client get wsdl file from service provider

We have two types of invoke webservices – jdk 1.8 install

1. proxy based web service – client is generated stubs and that stubs used to invoke webservice from service provider.

Eclipse is automatically used of stubs generation tool for creating stubs classes.

Dyamic web - Src folder java – right click – web service client – service definitaion – use web server run time - (<http://localhost:8084/Axis1CalServices/services/CalServices?wsdl>) – ok

Create TestClient - >

java.net.URL url = **new** java.net.URL("http://localhost:8084/Axis1CalServices/services/CalServices?wsdl");

org.apache.axis.client.Service service = **new** org.apache.axis.client.Service();

CalServicesSoapBindingStub stub = **new** CalServicesSoapBindingStub(url,service);

**int** result = stub.add(10, 10);

System.***out***.println("Addition is " + result);

Soap UI : It used to test api or web services or it creates soap tags.

Download - <https://www.soapui.org/downloads/soapui/>

Create new soap project – enter wsdl file/url

Graphical user interface, text, application, whiteboard

Description automatically generated

Soap request prepared by – stubs

Soap response prepared by – skeleton

Soap tag: Stubs Client request

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Soap Tag – Service provider response

Text

Description automatically generated

Disadvantages : Jax-RPC



Jax-WS🡪

Annotation : @WebService , @WebMethod , @WebParam

Every implementation used this annotation for used jax-ws

Used this annotation in interface and implementation class both.

@WebService – service name, target namespace, end point to interface.

@WebMethod(name=” ”) – give different method name in implementation class(optional)

@WebParam – give different name (optional)

Use Jax-ws-RI webservices

In lib folder

<https://repo1.maven.org/maven2/com/sun/xml/ws/jaxws-ri/2.2.10/>

<https://repo1.maven.org/maven2/javax/activation/activation/1.1.1/>

Service provider :

1. create interface

2. create implementation for interface.

3. Create web xml

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"* version=*"4.0"*>

<listener>

<listener-class>

com.sun.xml.ws.transport.http.servlet.WSServletContextListener

</listener-class>

</listener>

<servlet>

<servlet-name>WSServlet</servlet-name>

<servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>WSServlet</servlet-name>

<url-pattern>/calService</url-pattern>

</servlet-mapping>

<!-- <servlet-mapping>

<servlet-name>WSServlet</servlet-name>

<url-pattern>/fileService</url-pattern>

</servlet-mapping> -->

</web-app>

4. Create sun-jaxws.xml

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

<endpoints xmlns=*"http://java.sun.com/xml/ns/jax-ws/ri/runtime"* version=*"2.0"*>

<endpoint name=*"cService"* implementation=*"com.ingle.service.CalServicesImp"* url-pattern=*"/calService"* />

</endpoints>

Run - <http://localhost:8084/JaxWsRiCalServices/calService?wsdl>

Client service:

C:\Users\Ajay's\eclipse-workspace\servlet\JaxWsRiCalServicesClient>wsimport -keep -verbose <http://localhost:8084/JaxWsRiCalServices/calService?wsdl>

It create stubs :

TestClient:

**package** com.ingle.services;

**public** **class** TestClient {

**public** **static** **void** main(String[] arg) **throws** Exception{

java.net.URL url =**new** java.net.URL("http://localhost:8084/JaxWsRiCalServices/calService?wsdl");

javax.xml.namespace.QName q = **new** javax.xml.namespace.QName("http://services.ingle.com/","CalServiceImp");

javax.xml.ws.Service s = javax.xml.ws.Services.create(url,q);

com.ingle.services c = s.getPort(com.ingle.services.CalService.**class**);

**int** r = c.add(10,10);

System.out.println(r);

}

}

Use Jax-ws-metro webservices

same as above but jar file only change:

<https://mvnrepository.com/artifact/org.glassfish.main.packager/metro/3.1.2>

Use Jax-ws-axis2 webservices

<http://www.apache.org/dyn/closer.lua/axis/axis2/java/core/1.8.0/axis2-1.8.0-bin.zip>

create dynamic project with below 4 like select version 2.5 then only tomcat 9 is accepted.

1. install this is eclipse – preference– web service – axis 2 – select location

Service provider -

2. Calservice class

**package** com.ingle.service;

**public** **class** CalService {

**public** **int** add(**int** i, **int** j) {

**return** i+j;

}

}url - <http://localhost:8084/Axis2CalService/>

<http://localhost:8084/Axis2CalService/services/CalService?wsdl>

Client side :

Web service client – url definition – axis 2 – finish.

Test Client:

**package** com.ingle.services;

**public** **class** TestClient {

**public** **static** **void** main(String[] args) {

CalServiceStub stub = **new** CalServiceStub();

CalServiceStub.Add params = **new** CalServiceStub.Add();

params.setI(10);

params.setJ(19);

CalServiceStub.AddResponse res = stub.add(params);

**int** r = res.get\_return();

System.***out***.println(r);

}

}

Use Jax-ws-apache cxf webservices

Service provider –

CalService

**package** com.ingle.services;

**import** javax.jws.WebService;

@WebService(targetNamespace = "http://services.ingle.com/", portName = "CalServicePort", serviceName = "CalServiceService")

**public** **class** CalService {

**public** **int** add(**int** i , **int** j) {

**return** i+j;

}

}

url - [*http://localhost:8084/ApacheCXFCalService/services/CalServicePort*](http://localhost:8084/ApacheCXFCalService/services/CalServicePort)

Client Provider –

We can create with two ways

* Without creating configuration file
* With creating configuration file

Eclipse gives error so do this process –

Go to apche cxf – bin- wsdl2java

Go to project find your src location.

A picture containing text, screenshot

Description automatically generated

Clientwothoutconfig

**package** com.ingle.services;

**import** org.apache.cxf.jaxws.JaxWsProxyFactoryBean;

**public** **class** ClientWithoutConfg {

**public** **static** **void** main(String[] args) {

JaxWsProxyFactoryBean factory = **new** JaxWsProxyFactoryBean();

factory.setAddress("http://localhost:8084/ApacheCXFCalService/services/CalServicePort");

factory.setServiceClass(CalService.**class**);

Calservice service = factory.create();

System.***out***.println(service.add(12,1));

}

}

Webservice security

User level security- saml - **Security Assertion Markup Language**

Message level security-kebrose

Restful Services: using Jax-RS – by sun – jdk 1.6

Difference between soap and restful

1. soap – predefine tags from soap request and response.

2. soap – communication between client and webservice with xml of soap specification. It supports only xml.

3. Restful support all format json,xml…etc.

4. soap is time taking because when through stubs xml is created then soap request is check or validated some root tags and send response back to skeleton then again check some root tags so it’s time taking process.

5. rest is light weight and soap is heavy weight component.

6.soap is not good than performance than rest ful web services.

7. soap provides two security like user level and message level authentication but rest ful only have user level. Soap is good than rest ful in security.

8. restful is easily interact with frameworks like angular..

Restful webservice –

1. client needs convert object to json

2. service provider convert json to object

🡪 Jackson api, GSON api, Simple JSON.

1. client needs convert objects into xml= Jax-B api

2. service provider convert data from xml into objects = Jax-P api

------> using JackSon API:

Maven project –

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.12.5</version>

</dependency>

Java to json and json to java

// convert java to json

ObjectMapper mapper1 = **new** ObjectMapper();

mapper1.writeValue(**new** File("Student.json"), JavatoJson.javatojsonconvert());

System.***out***.println("Json is generated..");

// convert json to java

ObjectMapper mapper2 = **new** ObjectMapper();

Student s = mapper2.readValue(**new** File("Student.json"),Student.**class**);

System.***out***.println("id : " + s.getSid());

System.***out***.println("name : " + s.getSname());

System.***out***.println("++++++++++++++++++++++++++++++++++++++");

System.***out***.println("Your Liked subject:");

**for**(String st :s.getSubject()) {

System.***out***.println(st);

}

System.***out***.println("++++++++++++++++++++++++++++++++++++++");

System.***out***.println("Your address data");

Address a = s.getAddress();

System.***out***.println("City : " + a.getCity());

System.***out***.println("Zipcode : " + a.getZipcode());

Architecture how it is works:

Restful web service is more intelligent than soap webservice. In soap skeleton understand only one format that is xml but in restful web service skeleton understand any format.

Service provider converts json to java object through skeleton send to client. And client will converts java objects through some predefine class to json format. It is accept by skeleton again coverts into java object..repeat..

Representational state transform: It is like changing state of formats into xml,json..

Restful service is like servlet like we have used here url pattern and mapping .

Every methods has urls in rest ful apis.

There is no exception handling instead that we have response.

There is we have three support primitive type, user define type and response.

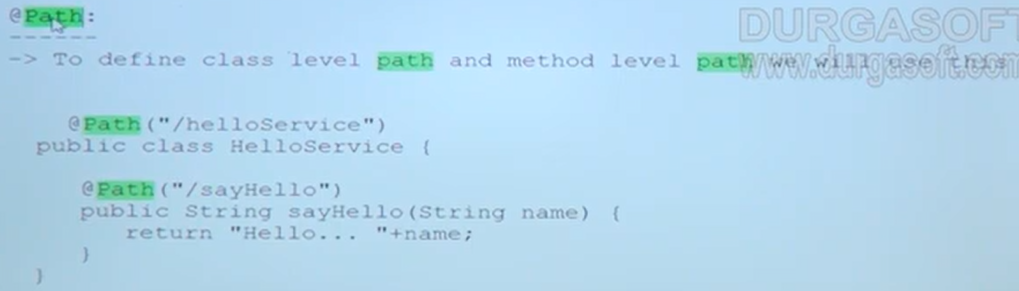
If method parameter is primitive like String then no need of any @produces annotation.

Every class has some annotation.

Text

Description automatically generated

1. @Path :



2. Parameter annotation

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text, letter

Description automatically generated

@FormParam

Graphical user interface, text, application

Description automatically generated

@Consumes –

If method parameter is user define type then we can’t used @get

Text, letter

Description automatically generated

@Produces

Graphical user interface, application

Description automatically generated

Make restful service :

* 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>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>jersey-serlvet</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>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

* HelloService.java

@Path("/helloService")

**public** **class** HelloService {

@Path("/sayHello/{age1}")

@GET

**public** String sayHello(@QueryParam("name1") String name,@PathParam("age1") **int** age) {

**return** "Hello" + name + "your age is"+age;

}

* }

TestClient :

**package** com.ingle.client;

**import** com.sun.jersey.api.client.Client;

**import** com.sun.jersey.api.client.WebResource;

**public** **class** TestClient {

**public** **static** **void** main(String[] args) {

Client c = Client.*create*();

WebResource r1 = c.resource("http://localhost:8084/restfulapijersey/rest/helloService/sayHello/21?name1=ajay");

String res = r1.get(String.**class**);

System.***out***.println(res);

}

}

Restful with xml ::

[1.@XmlRootElement](mailto:1.@XmlRootElement)

[2.@XmlElement](mailto:2.@XmlElement)

[3.@XmlAttribute](mailto:3.@XmlAttribute)