Restfull Services :-

RestFull Services Works on Archtectureal Principals of World wide web (RobertFiledings)

1)Addressability

2)UnformConstraintInterfaces and never Designed to Orbitary methods.So that No need of learning any defination language(IDL(Interface Defination Language , WSDL(WebServiceDefinationLanguage)))

3)Representation Oreiented( Data Transfor should not have ay restrictions)

4)Comminication Stateless( We should not maintain any requst state every time entire data has to send means every request is new request)

5)HATEOS (Hypermedia as the Engine of Application State) is a [constraint of the REST application architecture](https://en.wikipedia.org/wiki/HATEOAS).)

-->It will provide the links to communicate with the next consucutive requests

Restful service :

==================

==> Restful service is nothing but resouce which is distributed and interoperable.which follow architectural principle.provide by web to communicate with resource.and which give scalability.

==> jax-rs api is provide by java people and it has multiple implementation

1) jersey

2) rest-easy

**Note:-**

\* we have to use @path on top of class to make class as resource which is distributed and interoperable

\* we have to use @Queryparam to get paramter value from request

\* Like above @GET and @POST and @PUT and @DELETE

Following are 5 architectural principle to use resource in rest which is distributed and inter-operable

1) Addressibility --------------> direct address to access resources.

2) uniform constrain interface ------------> following http centric methods to find appropriate method and scalability.

3) representetion oriented ---------> Resouce class give response to corresponding content type which is send by client.there is no rule to provide ony xml as content type or specific content type.resource must access any content type and could give response back to particular content type

4) communication stateless ----------> It means resource will not capture the state of request.like session in servlet.every request is like new request.it will increse

the performance as well as scalability.

5) hateoas ---------------------> Resource will send link to get access next page to communicate wih resource.

**There are 4 http centric method which is supported by jax-rs api**

1) **Get** --> (we cand send data through uri as well as by query paramter)

To access or get data from resouce we go for get method, which is safe and idompotent.idompotent means if you send 1 request or n number of request it will give same state of resouce which is safe

2) **Post** --> (we cand send data through uri , request body as well as by query paramter)

to create or update new resource we use post method.max time with help of post we create new state, which is not safe and not idompotent.means evrey time it will create new state.

3) **Put** --> (we cand send data through uri , request body as well as by query paramter)

to create or update new resource we use put method,which is safe and idompotent.example if we update mobile number also it will not change state of resource.

4) **Delete** --> (we cand send data through uri , request body as well as by query paramter)

To delete state we use this method which is also safe and idompotent.

Resorce class

\*

**Archtecture Representation**

**1)**Addresability is very easy by direct url you can send the request or you can communicate with the requested resource

2)We Designed our Resources with fixed no of resources rather tahn orbitary method interfaces so that no one required any description language to understand about our Resource methods.

incase of others (WebServices) we are writing scud and wsdl as aprt of our applications so other players in the market should no about this technology

3)Representation oriented means What format of the data you can send i will recieve it and what format of the data you want i can send it to you.that means i can handle multiple data formats but as of now i will handle limited types bcz if am ahndling all the types of data i need twirite more logic to handle so it at all i want i will wire same in case of response i can send any kind of data that information i will send as part of response header or in my application i will send setContentType

4)Communicatin stateless our resources are not maintaing any state of the request bcz it will consumes the more resources of my server memory will be increased if it continously maintain the Session Objects

**Following Are The boostraping options present in jersey to use resource**

**web.xml for rest using jersey**

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

-------------------------------------------------------------------------------

we have to provide boostrap options here

--------------------------------------------------------------------------------

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

**## There are 8 boostraping options are there in rest using jersey**

**1) Providing package to find resource :-**

In this approch we have to provide package name of resource

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</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.products</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

**2) providing sub package to find resources present in sub package along with package :**

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</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.products</param-value>

</init-param>

<init-param>

<param-name>jersey.config.server.provider.scanning.recursive</param-name>

<param-value>true</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3) providing resource name directly to init param of servlet container :**

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.classnames</param-name>

<param-value>com.products.ProductInfo</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

**4) if you want your resource to be singleton(request per object) or request scope(request per thread or prototype) then we should use following**

step 1-----> create web.xml

----------------------------

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>com.restconfig.ProductInfoApplication</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

step2 ------>

create Application class to make resource singleton or prototype(request scope) by extending Application class provide by jax-rs api

**Application class:**

public class ProductInfoApplication extends Application {

private Set<Object> getsingleton;

public ProductInfoApplication() {

// TODO Auto-generated constructor stub

getsingleton = new HashSet<Object>();

getsingleton.add(new ProductInfo());

}

@Override

public Set<Object> getSingletons() {

// TODO Auto-generated method stub

return getsingleton;

}

}

**5) without using web.xml we can bind servlet container provided by jersey to servlet container by using**

@ApplicationPath("/rest/\*") annotation

for this to work we have to write following code

**Application class:**

===================

/\*package com.restconfig;

import java.util.HashSet;

import java.util.Set;

import javax.ws.rs.ApplicationPath;

import javax.ws.rs.core.Application;

import com.products.ProductInfo;

<!-- for singleton resource -->

@ApplicationPath("/rest/\*")

public class ProductInfoApplication extends Application {

private Set<Object> getsingleton;

public ProductInfoApplication() {

// TODO Auto-generated constructor stub

getsingleton = new HashSet<Object>();

getsingleton.add(new ProductInfo());

}

@Override

public Set<Object> getSingletons() {

// TODO Auto-generated method stub

return getsingleton;

}

}

**6) with the help of web.xml but without using servletContainer servlet of jersey directly providing Application class provided by jax-rs api**

**as servlet name in web.xml**

for this approch we have to follow to steps

==============================================

**step1 --->**

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>javax.ws.rs.core.Application</servlet-name>

</servlet>

<servlet-mapping>

<servlet-name>javax.ws.rs.core.Application</servlet-name>

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

<servlet-mapping>

**step 2 ---->**

application class :

===================

public class ProductInfoApplication extends Application {

private Set<Object> getsingleton;

public ProductInfoApplication() {

// TODO Auto-generated constructor stub

getsingleton = new HashSet<Object>();

getsingleton.add(new ProductInfo());

}

@Override

public Set<Object> getSingletons() {

// TODO Auto-generated method stub

return getsingleton;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**7) with the help of web.xml but without using servletContainer servlet of jersey directly providing our application class which is extended by Application class provided by jax-rs api**

**as servlet name in web.xml**

for this approch we have to follow to steps

==============================================

**step1 --->**

\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>com.restconfig.ProductInfoApplication</servlet-name>

</servlet>

<servlet-mapping>

<servlet-name>com.restconfig.ProductInfoApplication</servlet-name>

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

<servlet-mapping>

**step 2 ---->**

**application class :**

===================

public class ProductInfoApplication extends Application {

private Set<Object> getsingleton;

public ProductInfoApplication() {

// TODO Auto-generated constructor stub

getsingleton = new HashSet<Object>();

getsingleton.add(new ProductInfo());

}

@Override

public Set<Object> getSingletons() {

// TODO Auto-generated method stub

return getsingleton;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8) without using web.xml and without using application class.in this approch directly jax-rs api provide one

class ResourceConfig class. just we have to tke one class and we have to extend from ResourceConfig class

and in constructor we have to provide resource base on requirement either singleton resource or prototype

package com.resourceconfig;

import javax.ws.rs.ApplicationPath;

import org.glassfish.jersey.server.ResourceConfig;

import com.products.ProductInfo;

@ApplicationPath("/rest/\*")

public class ProductInfoResourceConfig extends ResourceConfig {

public ProductInfoResourceConfig() {

// for singleton resouce we need to use this

//register(new ProductInfo());

// for per request new resouce you need to use this

register(ProductInfo.class);

}

}

RestEasy Implimenattion:-

## The way Jersey provided implimentation to the jax-rs api there are different different vendors provided implimentations to it.Here comes the RestEasy provided by JBoss people it will be compatable with any servletContainer as well as Jboss server it will be more compatable with the Jboss server means more no of features are provided/shifted along with the jboss server

## like how glassfish people provided impimentation to jersey the same way jboss people also did like wise apache people also provided some implimentation as well.But the features provided by Jboss people are very less when compared with the jersey provided implimentation

## Later versions of Jboss 7 they renamed it as wildfly they just renamed it apart from everything same wildfly 8 ,9 ,10 ,11 currently running. if you install wildfly server by default all theses implimentations are shifted along with that. you no need to provide any thing here.few days back resteasy 3.1.4 got released but our wilfly server contains the jars as bit low versions if you want to work with wildfly server with latest jars go to resteasy binary distribution and download latest jar files and go to server binaries and jax-rs replae with latest jars

## Here you have to provide jars which are zipped inside the latest binaries which you downloaded as part of it.it contains one binary zip file for wildfly those jars only you have to provide otherwise your server is going to corrupt.

## ResyEasy people provided 3 ways of Bootstraping the RestEasy Container but the first two ways are almost deprecated. if you want to work with those lower versions of it you have to install the old versions of the wildfly upto 10 but later versions of wildfly i.e 11 almost they deprecated it only one approach it will work. but industry mostly they may work with any of the versions bcz .in industrys that much quick adoptive ness will not be avalable

**First BootSraping option:-**

**##** incase of RestEasy HttpDispatcherServlet act as jax-rs runtime. the way jersey people provided ServletContainer resteasy people also provided one runtime with this servlet

<web-app>

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

<!-- By default it is disable we have to enable by saying that auto-scan true. it will create the every resource with request scope. but we have to tell to HttpDispatcherServlet by writing this tag -->

<context-param>

<param-name>resteasy.scan</param-name>

<param-value>true</param-value>

</context-param>

<!-- set this if you map the Resteasy servlet to something other than /\* unlike our jersey it does not support auto detection with url specified with wildcard charecter we have to

<context-param>

<param-name>resteasy.servlet.mapping.prefix</param-name>

<param-value>/resteasy</param-value>

</context-param>

-->

<servlet>

<servlet-name>Resteasy</servlet-name>

<servlet-class>

org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Resteasy</servlet-name>

<url-pattern>/resteasy</url-pattern>

</servlet-mapping> </web-app>

**Second Way of Bootstraping option:-**

## In this case we have to Provide the Application class whose implemented-class means the class who is extending the Application class we have to pass that particular class as input parameter to it

## like we have done in jersey implimentation we have to over ride those two methods getSingletones and getClasses methods. so HttpDispatcherServlet will call those methods and provides the scope to it.the difference between the jersey and resyEasy in this approach is here the parameter name will change here

**step 1-----> create web.xml**

**\*\*\*\*\*\* web.xml for this approch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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\_3\_1.xsd" id="WebApp\_ID" version="3.1">

<servlet>

<servlet-name>RestApp</servlet-name>

<servlet-class> org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher

</servlet-class>

<init-param> //Here the change will be we need to place core also here

<param-name>javax.ws.rs.core.Application</param-name>

<param-value>com.restconfig.ProductInfoApplication</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>RestApp</servlet-name>

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

<servlet-mapping>

**step2 ------>**

create Application class to make resource singleton or prototype(request scope) by extending Application class provide by jax-rs api

**Application class:**

public class ProductInfoApplication extends Application {

private Set<Object> getsingleton;

public ProductInfoApplication() {

// TODO Auto-generated constructor stub

getsingleton = new HashSet<Object>();

getsingleton.add(new ProductInfo());

}

@Override

public Set<Object> getSingletons() {

// TODO Auto-generated method stub

return getsingleton;

}

}

## From the above approach HttpDispatcherServlet(Jax-rs Runtime will be created) with the resources and their scopes

**Third Approach:-**

**##** Third Approach is same as jersery approach it is also known as j2ee5+ / servlet 3.1 or above / no web.xml

## In this case we just need to write a class that extends from Application class and need to over write this methods from it. so when ever the request sent to our resource.ServletContainer will get the request and it will forwords the request to the HttpServletDispatcher then This servlet will go to the all the resources and it will identify for the @Path annotations and it will read the meta data of the all the resources and placed inside the jax-rs runtime if it is not the case for every request it has to go to each and every resource and identifying the @Path Annotation is time taking process.So for this servletContainers will come up with the unique solution i.e ServletContext that meta data information of the our project will be placed inside the ServletContext so when ever the request comes it will identify by using servletContext. ServletContext will read the application information from the web.xml. but web.xml we are not using so for this purpose we have to go for programetic Approach so write a class that implements from one interface(mostly ServletContainerInitializer) servletContainer will call that class just after creating of this class so inside this you can write the logic to register your classes inside the servletContext object

##But the Problem with this is the person who ever all want to use this every one has to write this logic so what jax-rs people said you no need to write this logic we our self provided this logic you write a class that extends from Application and Annootated with @ApplicationPath

This Aanotation is not comming from implementation vendors so it is comming from Jax-rs.Api so every vendor must implement this Annotation as part of their implementation. if you provide this Annotation to your calss then ServletContainer will be initialized means created and then as part of Contractual Agreement it will call the Jersery or Rest Easy ServletContainerInitializer and executes the one over rided method i.e onStartUp method and it will identify the class which is Annotated with AppilcationPath and read that url and it will create or register RestEasy Servlet(HttpDispatcherServlet) with in the ServletContext with that url so by default it will make all the resource classes as Request scope.This is the Third Approach. and this is also known as independent of implementation vendors.the same code will work with other implementation vendors also bcz this annotations are comming from api not from implementation vendors

@ApplicationPath("/tour")

class TourManagerApplication extends Application{

}