JAX-RS XML Example With JAXB Using Jersey

Web Services »on Jul 25, 2014 { 7 Comments } By Sivateja

In this article i will give you an example on how a RESTful web service produces XML response using Jersey. Basically JAX-RS supports conversion of java objects into XML with the help of JAXB. As Jersey it self contains JAXB libraries we no need to worry about JAXB-Jersey integration stuff.

Steps Need to be Followed

Add `jersey-server.jar` to your Maven pom.xml which includes all JAXB supporting libraries into your class path

Annotate your service method with @Produces(MediaType.APPLICATION_XML)

Required Files

pom.xml

web.xml

Customer.java

RestfulXMLExample.java

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
  <modelVersion>4.0.0</modelVersion>
  <groupId>RestfulXMLExample</groupId>
  <artifactId>RestfulXMLExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <repositories>
      <repository>
        <id>maven2-repository.java.net</id>
        <name>Java.net Repository for Maven</name>
        <url>http://download.java.net/maven/2/</url>
        <layout>default</layout>
      </repository>
   </repositories>

   <dependencies>
```

```
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-server</artifactId>
      <version>1.8</version>
    </dependency>

  </dependencies>

   <build>
    <finalName>RestfulXMLExample</finalName>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <compilerVersion>1.5</compilerVersion>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XML
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>RestPathAnnotationExample</display-name>

  <servlet>
    <servlet-name>jersey-serlvet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-clas
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.java4s</param-value>
    </init-param>
    <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>
```

Customer.java

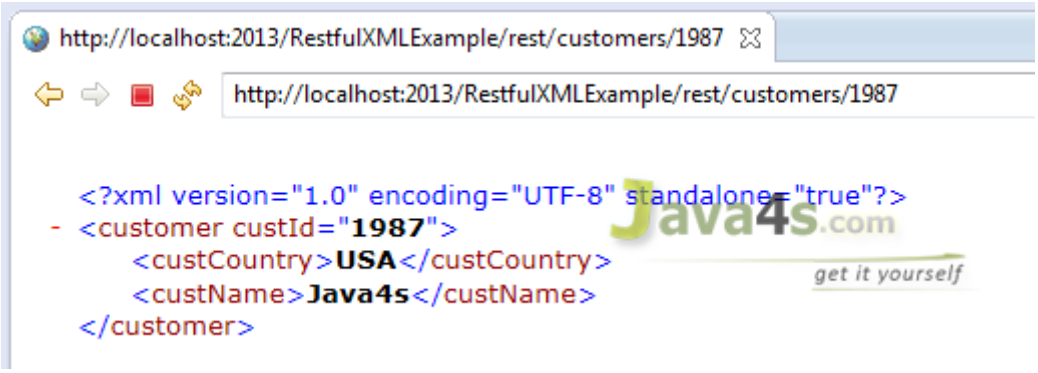
```
package com.java4s;

import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "customer")
public class Customer {

    String custName;
    String custCountry;
    int custId;

    @XmlElement
    public String getCustName() {
        return custName;
    }
    public void setCustName(String custName) {
        this.custName = custName;
    }

    @XmlElement
    public String getCustCountry() {
        return custCountry;
    }
    public void setCustCountry(String custCountry) {
        this.custCountry = custCountry;
    }

    @XmlAttribute
    public int getCustId() {
        return custId;
    }
    public void setCustId(int custId) {
        this.custId = custId;
    }
}
```

RestfulXMLExample.java

```
package com.java4s;
```

```
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.MediaType;
8
9  @Path("/customers")
10 public class RestfulXMLExample {
11
12    @GET
13    @Path("/{id}")
14    @Produces(MediaType.APPLICATION_XML)
15    public Customer getCustomerDetails(@PathParam("id") int custId) {
16
17
18     // WRITE DATABASE LOGIC TO RETRIEVE THE CUSTOMER RECORD WITH 'custID'
19
20       Customer cust = new Customer();
21       cust.setCustName("Java4s");
22       cust.setCustCountry("USA");
23       cust.setCustId(custId);
24
25       return cust;
26    }
27 }
```

Output



You Might Also Like

RESTful Java Client Example Using Jersey Client

Web Services »on Aug 5, 2014 { 11 Comments } By Sivateja

In this article i will describe how to write a JAX-RS client application using jersey client API, so far we used to call & test/read our RESTful service by its URL directly hitting in the browser [check the previous examples], but in the real time we will call the services by writing some client application logic. We have different ways to write a RESTful client. In this article i will use Jersey client [of course we are using Jersey for writing RESTful service too]

Steps need to be followed

Need to add 'jersey-client' dependency in pom.xml

As we are creating the Client application, we need to write a RESTful service to test that client, so i will take previous JSON example in order to do that

Write a client application and run it

Files Required

pom.xml

web.xml

JsonFromRestful.java

Customer.java

RESTfulClient.java

pom.xml

In the pom.xml we must include 'jersey-client' dependency [Check line numbers from 4-10]

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>JAX-RS-Client-Example</groupId>
4    <artifactId>JAX-RS-Client-Example</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6    <packaging>war</packaging>
7
8    <repositories>
9        <repository>
```

```
        <id>maven2-repository.java.net</id>
        <name>Java.net Repository for Maven</name>
        <url>http://download.java.net/maven/2/</url>
        <layout>default</layout>
      </repository>
    </repositories>

    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.8.2</version>
        <scope>test</scope>
      </dependency>

      <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-server</artifactId>
        <version>1.8</version>
      </dependency>

      <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-json</artifactId>
        <version>1.8</version>
      </dependency>

      <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-client</artifactId>
        <version>1.8</version>
      </dependency>

    </dependencies>

     <build>
      <finalName>JAX-RS-Client-Example</finalName>
      <plugins>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
            <compilerVersion>1.5</compilerVersion>
            <source>1.5</source>
            <target>1.5</target>
          </configuration>
        </plugin>
      </plugins>
    </build>
```

```
</project>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ja
  <display-name>JAX-RS-Client-Example</display-name>
  <servlet>
      <servlet-name>jersey-serlvet</servlet-name>
      <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-clas
      <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>java4s</param-value>
      </init-param>
      <init-param>
        <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
        <param-value>true</param-value>
      </init-param>
      <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>
```

JsonFromRestful.java

```
package java4s;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;

@Path("/customers")
public class JsonFromRestful {

   @GET
   @Path("/{cusNo}")
   @Produces("application/json")
   public Customer produceCustomerDetailsinJSON(@PathParam("cusNo") int no) {

      /*
       * I AM PASSING CUST.NO AS AN INPUT, SO WRITE SOME BACKEND RELATED STU
       * FIND THE CUSTOMER DETAILS WITH THAT ID. AND FINALLY SET THOSE RETRIE
       * THE CUSTOMER OBJECT AND RETURN IT, HOWEVER IT WILL RETURN IN JSON F
       * */

```

```
22        Customer cust = new Customer();
23           cust .setCustNo(no);
24           cust .setCustName("Java4s");
25           cust .setCustCountry("India");
26        return cust;
27     }
28  }
```

Customer.java

```
1  package java4s;
2
3  public class Customer {
4
5     private int custNo;
6     private String custName;
7     private String custCountry;
8
9     public int getCustNo() {
10       return custNo;
11    }
12    public void setCustNo(int custNo) {
13       this.custNo = custNo;
14    }
15    public String getCustName() {
16       return custName;
17    }
18    public void setCustName(String custName) {
19       this.custName = custName;
20    }
21    public String getCustCountry() {
22       return custCountry;
23    }
24    public void setCustCountry(String custCountry) {
25       this.custCountry = custCountry;
26    }
27
28 }
```

RESTfulClient.java

```
1  package java4s;
2
3  import com.sun.jersey.api.client.Client;
4  import com.sun.jersey.api.client.ClientResponse;
5  import com.sun.jersey.api.client.WebResource;
6
7  public class RESTfulClient {
8
9     public static void main(String[] Java4s) {
```

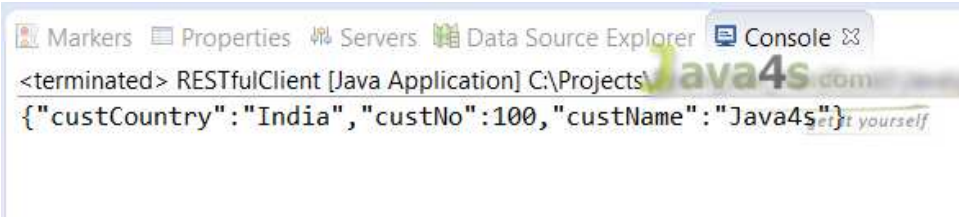
```
10
11      try {
12          Client client = Client.create();
13          WebResource resource = client.resource("http://localhost:2015/JAX-RS-Client-E
14          ClientResponse response = resource.accept("application/json").get(ClientRespor
15
16          if(response.getStatus() == 200){
17
18              String output = response.getEntity(String.class);
19              System.out.println(output);
20
21          }else System.out.println("Somthing went wrong..!");
22
23        } catch (Exception e) {
24              e.printStackTrace();
25        }
26
27      }
28  }
```

Explanation:

Line number 13, we are calling our rest service

Line number 14, we are intimating the REST Service that our client will accept JSON response

Right click on RESTfulClient.java in eclipse > Run As > Run on Server



DOWNLOAD

You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- JAX-RS XML Example With JAXB Using Jersey
- How to Test (JAX-RS) RESTful Web Services

RESTful Web Service (JAX-RS) JSON Example Using Jersey

Web Services »on Jul 19, 2014 { 16 Comments } By Sivateja

This article describes how to get a JSON response from the RESTful web services using jersey implementation. Jersey will use Jackson to convert Java objects to/form JSON, but just don't ask me what is Jackson ,as of now just remember its a high performance JSON processor, Jersey will use this API to the marshaling [converting the objects] process. Check this link if you would like to know more about Jackson.

These steps are mandatory in order to make Jersey to support with JSON mappings.

Apart from existing dependencies, add '*jersey-json.jar*' to your Maven pom.xml which includes all Jackson and other JSON supporting libraries

```
<dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-json</artifactId>
        <version>1.8</version>
</dependency>
```

In web.xml add "com.sun.jersey.api.json.POJOMappingFeature" as "init-param"

In the web service class, we need to annotate the method with @Produces(MediaType.APPLICATION_JSON). By doing so we are instructiong the service method that we are expecting the JSON output, thats it jersey will take care rest of the things

> **Note**: *In previous examples i used Tomcat 6 and JDK 1.6 but for this [JAX-RS JSON Example] example i have used JDK 1.7.*

Required Files

pom.xml

web.xml

Customer.java

JsonFromRestful.java

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
  <modelVersion>4.0.0</modelVersion>
  <groupId>JsonFromRestfulWebServices</groupId>
  <artifactId>JsonFromRestfulWebServices</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

    <repositories>
        <repository>
          <id>maven2-repository.java.net</id>
           <name>Java.net Repository for Maven</name>
           <url>http://download.java.net/maven/2/</url>
           <layout>default</layout>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-server</artifactId>
            <version>1.8</version>
        </dependency>

        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-json</artifactId>
            <version>1.8</version>
       </dependency>
     </dependencies>

    <build>
      <finalName>JsonFromRestfulWebServices</finalName>
      <plugins>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
              <compilerVersion>1.5</compilerVersion>
              <source>1.5</source>
              <target>1.5</target>
            </configuration>
        </plugin>
      </plugins>
```

```
    </build>
</project>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ja
  <display-name>JsonFromRestfulWebServices</display-name>
  <servlet>
        <servlet-name>jersey-serlvet</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-clas
        <init-param>
          <param-name>com.sun.jersey.config.property.packages</param-name>
          <param-value>java4s</param-value>
        </init-param>
        <init-param>
          <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
          <param-value>true</param-value>
        </init-param>
        <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>
```

Customer.java

```
package java4s;

public class Customer {

    private int custNo;
    private String custName;
    private String custCountry;

    public int getCustNo() {
        return custNo;
    }
    public void setCustNo(int custNo) {
        this.custNo = custNo;
    }
    public String getCustName() {
        return custName;
    }
    public void setCustName(String custName) {
        this.custName = custName;
    }
```

```
21     public String getCustCountry() {
22        return custCountry;
23     }
24     public void setCustCountry(String custCountry) {
25        this.custCountry = custCountry;
26     }
27  }
```

JsonFromRestful.java

```
1  package java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.MediaType;
8
9  @Path("/customers")
10 public class JsonFromRestful {
11
12    @GET
13    @Path("/{cusNo}")
14    @Produces(MediaType.APPLICATION_JSON)
15    public Customer produceCustomerDetailsinJSON(@PathParam("cusNo") int no) {
16
17       /*
18        * I AM PASSING CUST.NO AS AN INPUT, SO WRITE SOME BACKEND RELATED STU
19        * FIND THE CUSTOMER DETAILS WITH THAT ID. AND FINALLY SET THOSE RETRIE
20        * THE CUSTOMER OBJECT AND RETURN IT, HOWEVER IT WILL RETURN IN JSON F
21        * */
22
23       Customer cust = new Customer();
24          cust .setCustNo(no);
25          cust .setCustName("Java4s");
26          cust .setCustCountry("India");
27       return cust;
28    }
29 }
```

Output

DOWNLOAD

You Might Also Like

- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey
- How to Test (JAX-RS) RESTful Web Services

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks.
You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

Comments

16 Responses to “RESTful Web Service (JAX-RS) JSON Example Using Jersey”

Rammohan says:
August 21, 2014 at 10:50 AM

All the tutorials available in Java4S are very good and real kick start to learn java based technologies.

I would request Java4S to add Soap based web services tutorials and Spring Security tutorials as well.

Reply

Jersey Hello World Example Using JAX-RS Specification

Web Services »on Jul 6, 2014 **{ 28 Comments }** By Sivateja Like 0

In this tutorial, I will show you how to develop a RESTful hello world web application with Jersey & Maven in Eclipse. I have used Eclipse Juno to develop all web services. Make sure you have installed Maven plugin in eclipse before you start, you can check this article for help [How to Install m2eclipse (Maven) Plugin in Eclipse]

Required

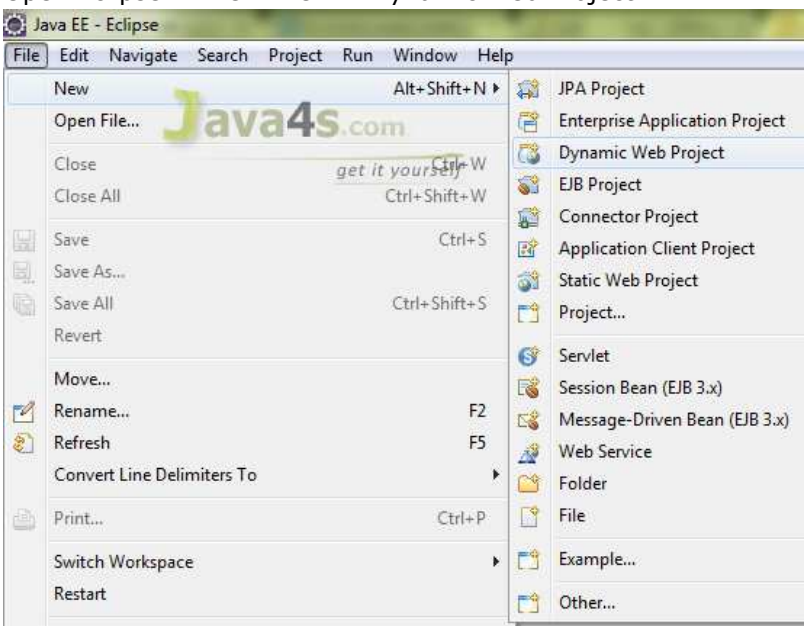
- Eclipse Juno
- JDK 1.6
- Jersey 1.8
- Maven Plugin
- Tomcat 6.0 Server

Steps

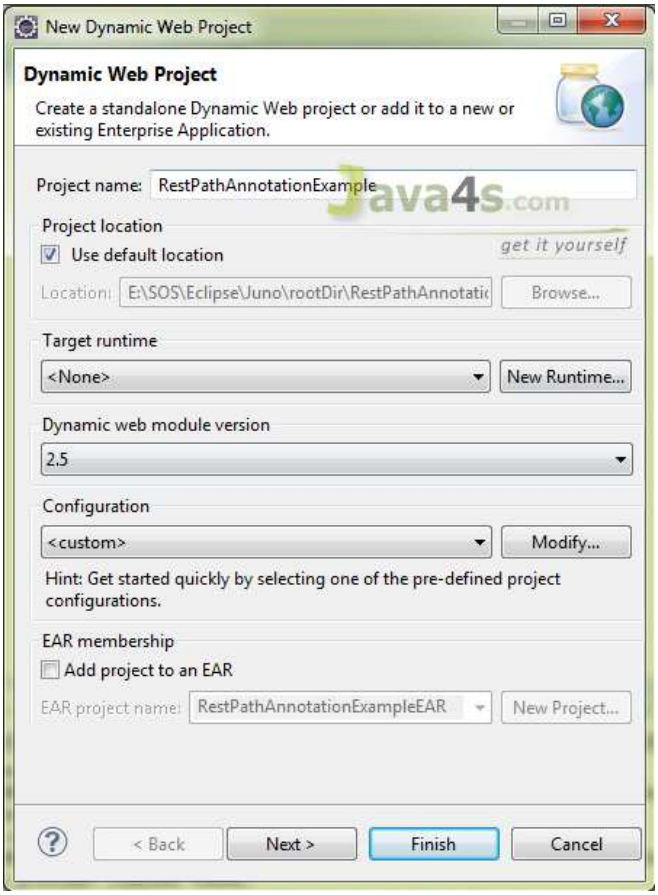
- Create a 'Dynamic Web Project'
- Convert the project into 'Maven Project' [Of course you can also create Maven project directly]
- Add required dependencies in 'pom.xml'
- Change web.xml [register *com.sun.jersey.spi.container.servlet.ServletContainer* and add related init-param]
- Run the application

Steps to Create Restful Web Services in Eclipse

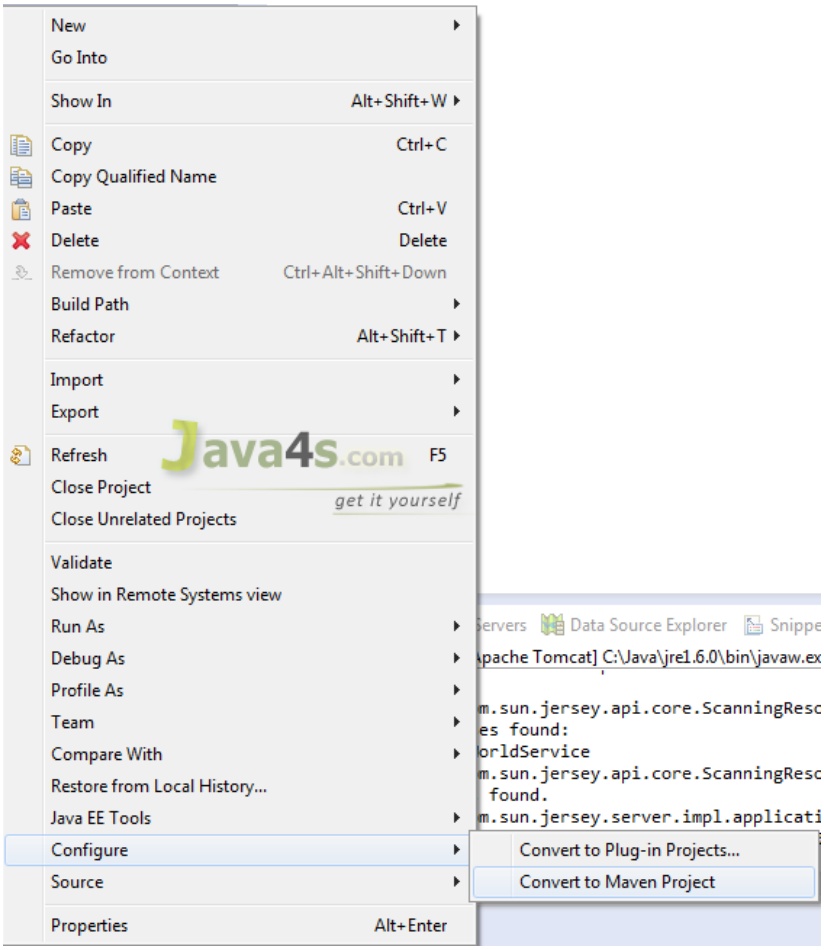
Open Eclipse > File > New > Dynamic Web Project



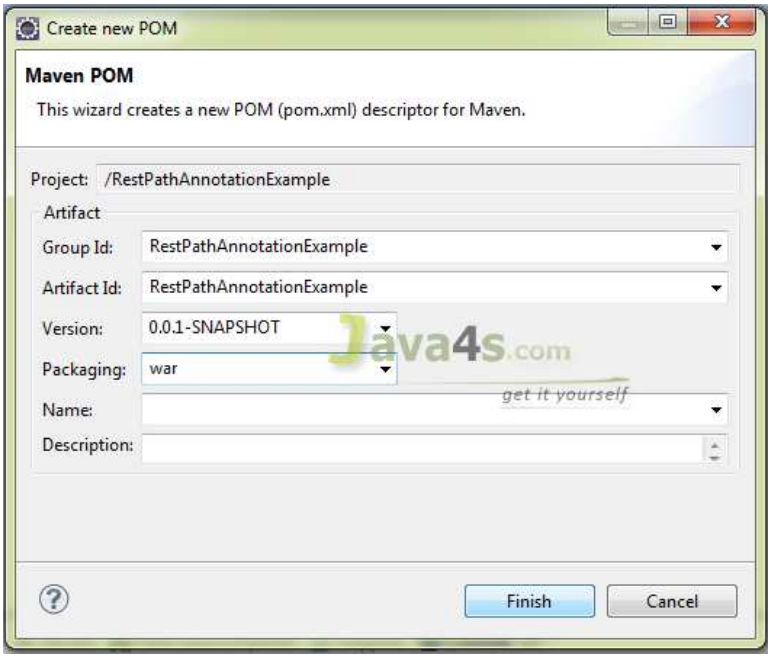
Give the project name and choose 'Dynamic web module version' as 2.5 > Finish



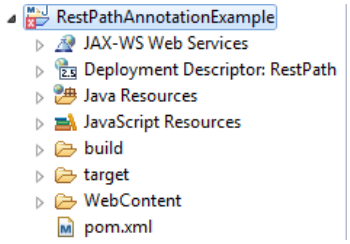
Now the project will be created in the work space, right click on the project folder > *Configure* > *Convert to Maven Project*



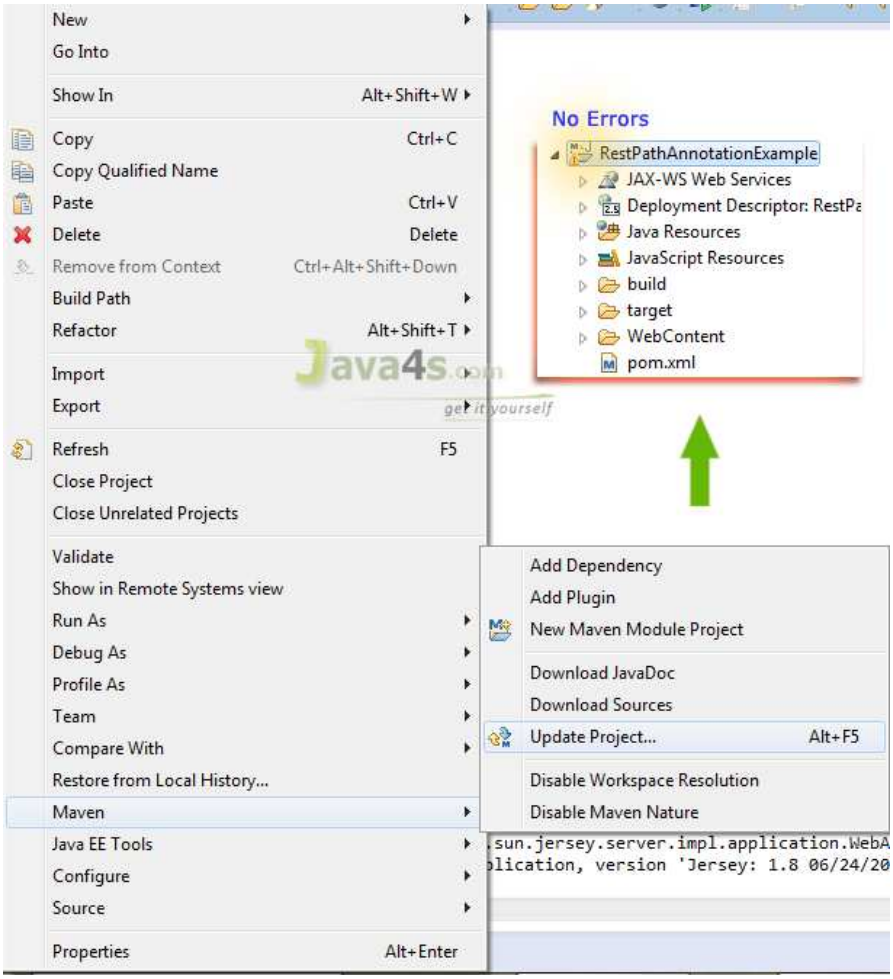
Now it will open Maven POM window, there keep everything as it is, but choose packaging to ***.war*** and click Finish



You have created a *Maven project*, finally your project looks like..



Most probably it will not show any errors, but here Its showing errors. In order to fix this right click on the project > *Maven* > *Update Project* > Now you can see the errors gone 😃



Open *pom.xml* and add the *Maven dependencies* just like bellow

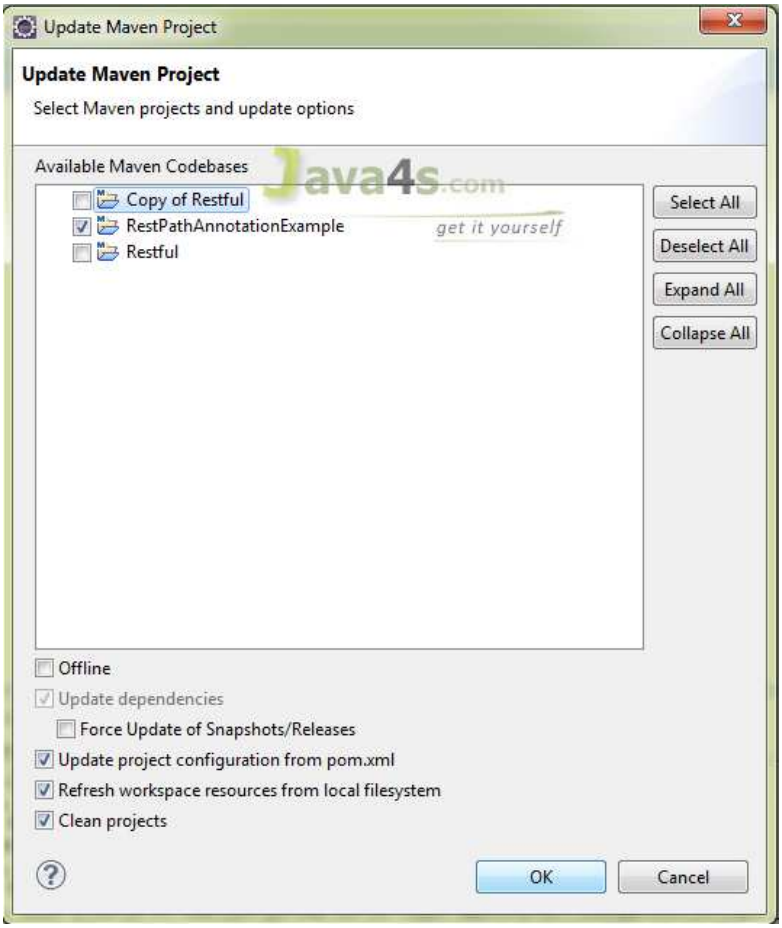
Pom.xml

What is pom.xml :

It is an XML file that contains information about the project and configuration details used by Maven to build the project. Generally at the time of developing any J2EE applications we will search and download the related jar files over the internet and we need to add them in the class path and even in the lib folder as well. But if you can install Maven plugin in your Eclipse, pom.xml will take care of adding these dependencies (`*.jars) to the project. Your work is to install Maven and update pom.xml with the required dependencies (jars). However i have already explained, how to install Maven plugin in the Eclipse.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sche
<modelVersion>4.0.0</modelVersion>
<groupId>RestPathAnnotationExample</groupId>
<artifactId>RestPathAnnotationExample</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>

<repositories>
    <repository>
      <id>maven2-repository.java.net</id>
       <name>Java.net Repository for Maven</name>
       <url>http://download.java.net/maven/2/</url>
       <layout>default</layout>
    </repository>
</repositories>

<dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-server</artifactId>
      <version>1.8</version>
    </dependency>
</dependencies>

<build>
  <finalName>RestPathAnnotationExample</finalName>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <compilerVersion>1.5</compilerVersion>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
    </plugin>
  </plugins>
</build>

</project>
```

Once you add dependencies [required libraries] in *pom.xml* > right click on the project > *Maven* > *Update Project* > Choose the current project > Ok



HelloWorldService.java

```
1  package com.java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.core.Response;
7
8  @Path("/customers")
9  public class HelloWorldService {
10
11   @GET
12   @Produces("text/html")
13   public Response getLocalCust() {
14
15         String output = "I am from 'getLocalCust' method";
16         return Response.status(200).entity(output).build();
17   }
18
19   @GET
20   @Path("/nri")
21   @Produces("text/html")
22   public Response getNriCust() {
23
24          String output = "I am from 'getNriCust' method";
25          return Response.status(200).entity(output).build();
26   }
27 }
```

web.xml

```
1  <web-app id="WebApp_ID" version="2.4"
2  xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
4  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
5  <display-name>RestPathAnnotationExample</display-name>
6
```

```
<servlet>
   <servlet-name>jersey-serlvet</servlet-name>
   <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
     <init-param>
       <param-name>com.sun.jersey.config.property.packages</param-name>
       <param-value>com.java4s</param-value>
     </init-param>
     <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>

<!-- www.Java4s.com -->
```

Now we are good to run the application, Just right click on the project >*Run As* > *Run on Server* > It will open the application *URL* like

http://localhost:2013/RestPathAnnotationExample/

But you need to satisfy the actual URL pattern, i mean change the URL to…

Main application URL:
http://localhost:2013/RestPathAnnotationExample/

Web.xml URL pattern:
/rest

@Path in HelloWorldService.java:
/customers

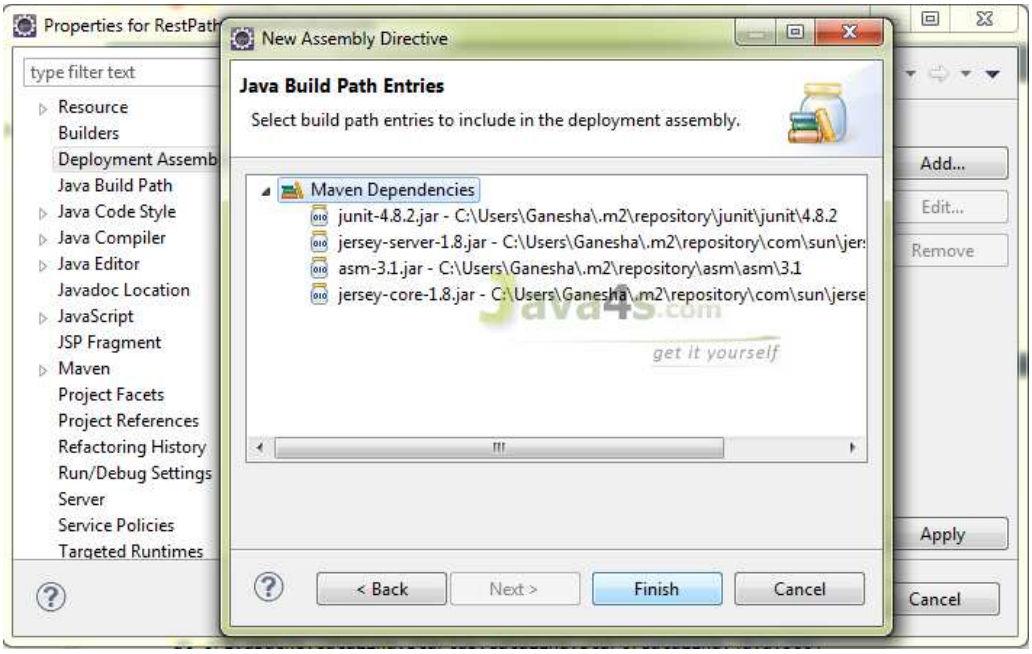
Final URL should be
http://localhost:2013/RestPathAnnotationExample/rest/customers

Hit with this final URL in your web browser or eclipse browser

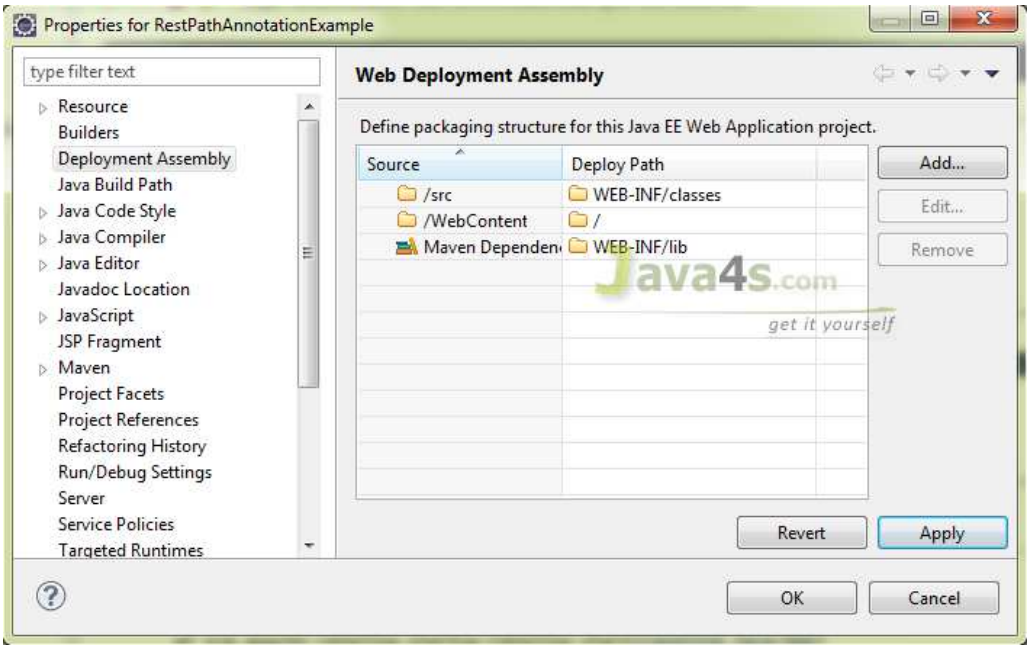
It will throw 404 error, because we forgot to add the Maven dependencies in the Deployment Assembly. So how to fix this issue ? just right click on the project > *Properties* > *Deployment Assembly* > *Add* > It will open other

window, in that choose '*Java Build Path Entries*' click Next..

Choose the Maven Dependencies root and Finish



Now it looks like..



Click Ok > and test the application with URLs

http://localhost:2013/RestPathAnnotationExample/rest/customers

http://localhost:2013/RestPathAnnotationExample/rest/customers/nri [Here /nri is the path i have mentioned in HelloWorldService.java for getNriCust() method*]*

DOWNLOAD

You Might Also Like

- Spring Boot + Maven – Hello World Example Step by Step
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- AngularJs Hello World Example
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks.
You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

Comments

28 Responses to “Jersey Hello World Example Using JAX-RS Specification”

Siva says:
August 3, 2014 at 6:09 PM

Can you please provide one JAX-RS web service example with out annotations. Actually i am learning Web services.

Reply

Andreas+Papandreas says:
October 19, 2014 at 10:17 AM

good exaple. never thought that web services are so simple 😃

Reply

Swati says:
January 17, 2015 at 6:39 AM

Hi Sivateja , thank you very much for such a simple and helpful sample code…

Reply

RESTful Web Services (JAX-RS) @FormParam Example

Web Services »on Jul 9, 2014 { 2 Comments } By Sivateja

By using @FormParam annotation, RESTful web service would accept HTML form parameters sent by the client in the POST request and bind them to the method variables. Generally @FormParam will come into picture when client send the data in POST request, if its the GET request @QueryParam would be the best choice.

Let me give you an example on usage of @FormParam in the JAX-RS.

> ***Note***:
> *If you are new to RESTful web services, first go through 'Jersey Hello World Example Using JAX-RS Specification' there you can learn each and every step to create a RESTful web service in eclipse, how to install maven and configuration settings related to JAX-RS.*

Required Files

pom.xml and web.xml are similar to the previous article

RestServiceFormParamJava4s.java

Client.html

RestServiceFormParamJava4s.java

```
1  package com.java4s;
2
3  import javax.ws.rs.FormParam;
4  import javax.ws.rs.POST;
5  import javax.ws.rs.Path;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.Response;
8
9  @Path("/customers")
10 public class RestServiceFormParamJava4s {
11
12    @POST
13    @Path("/addCustomer")
```

```
14    @Produces("text/html")
15    public Response getResultByPassingValue(
16                @FormParam("nameKey") String name,
17                @FormParam("countryKey") String country) {
18
19       String output = "<font face='verdana' size='2'>" +
20             "Web Service has added your Customer information with Name - <u>"+name
21       return Response.status(200).entity(output).build();
22
23    }
24 }
```

Client.html

```
1  <html>
2  <head>
3     <title>RESTful Web Services (JAX-RS) @FormParam Exampale</title>
4  </head>
5  <body>
6
7     <form action="http://localhost:2013/RestFormParamAnnotationExample/rest/custom
8
9        <table>
10          <tr>
11             <td><font face="verdana" size="2px">Customer Name : </font></td>
12             <td><input type="text" name="nameKey" /> </td>
13          </tr>
14
15          <tr>
16             <td><font face="verdana" size="2px">Country</font></td>
17             <td> <input type="text" name="countryKey" /> </td>
18          </tr>
19
20          <tr>
21             <td></td>
22             <td><input type="submit" value="Add Customer" /> </td>
23          </tr>
24       </table>
25
26    </form>
27
28 </body>
29 </html>
```

Explanation

Right click on your project root folder > *Run As* > *Run on Server*

Eclipse will open *http://localhost:2013/<projectRootFolder>* with 404 Error by default, forget about that

Now open *Client.html* in your web browser, enter the details and click submit [I have created this .html file to send input form parameters to our RESTful service, you no need to create & place this file in the project workspace, myself i have created client.html file in my desktop and open in Google chrome, and verified the output]

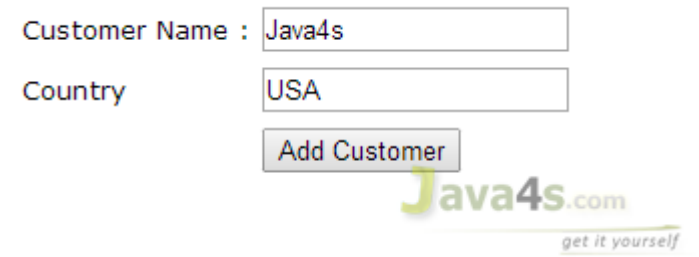
In Client.html, observe the URL in the from action [line number **7**]

Once you click on Submit, *Client.html* will POST the data to the restful service. From there REST service will retrieve those details by using @FormParam annotation.

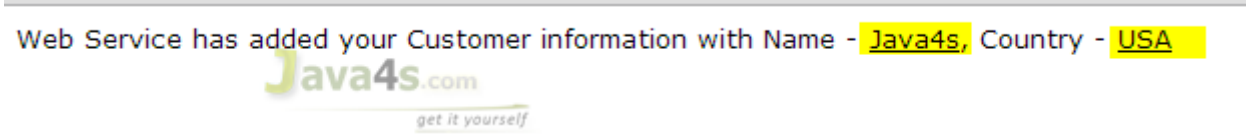
Remember: *Input field names in Client.html [line numbers 12,17] should match with @FormParam("-") parameters[line numbers 16,17] in RestServiceFormParamJava4s.java*

Output

Input:



Output:



You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

RESTful Web Services (JAX-RS) @MatrixParam Example

Web Services »on Jul 8, 2014 { 3 Comments } By Sivateja

In this article i will describe how a RESTful web services would accept multiple parameters sent by the client in the HTTP URL as Matrix Params. So what are matrix parameters ? let me give you the syntax.

Matrix Parameters Syntax

Consider this URL

http://localhost:2013/<projectRoot>/rest/customers;nameKey=*Java4s*;countryKey=*USA*

If you observe the URL, i am passing 2 parameters *nameKey=Java4s* & *countryKey=USA*. One parameter is separated from another with a semicolon, similarly you can pass any number of parameters. These type of parameters are called as Matrix Parameters. I will explain more about matrix parameters in this example.

Required Files

web.xml & pom.xml [same as previous articles]

RestServiceMatrixParamJava4s.java

RestServiceMatrixParamJava4s.java

```
1  package com.java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.MatrixParam;
5  import javax.ws.rs.Path;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.Response;
8
9  @Path("/customers")
10 public class RestServiceMatrixParamJava4s{
11
12    @GET
13    @Produces("text/html")
14    public Response getResultByPassingValue(
15                  @MatrixParam("nameKey") String name,
16                  @MatrixParam("countryKey") String country) {
17
18       String output = "Customer name - "+name+", Country - "+country+"";
19       return Response.status(200).entity(output).build();
20
```

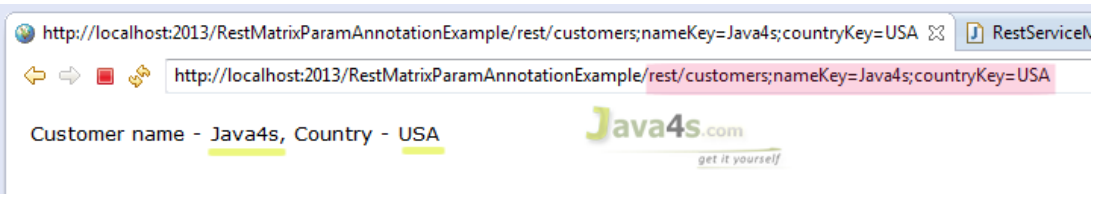
```
21     }
22 }
```

Explanation

Once you run the application, eclipse will open the following URL *http://localhost:2013/RestMatrixParamAnnotationExample/* by default

In RestServiceMatrixParamJava4s.java [line number 9] we have given class level path as /customers and we are using @MatrixParam annotation to retrieve the client inputs from the URL, so the final URL should be
http://localhost:2013/<projectRoot>/rest/*customers*;nameKey=*Java4s*;countryKey=*USA*

Output



You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks.
You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

Comments

3 Responses to “RESTful Web Services (JAX-RS) @MatrixParam Example”

vishwnath says:
June 6, 2015 at 9:22 AM

RESTful Web Services (JAX-RS) @PathParam Example

Web Services »on Jul 8, 2014 { 7 Comments } By Sivateja

In RESTful (JAX-RS) web services @PathParam annotation will be used to bind RESTful URL parameter values to the method arguments. Lets discuss with a simple example.

> **Note**:
> *If you are new to RESTful web services or if you would like to know complete step by step flow of JAX-RS, Go through this article 'Jersey Hello World example With Maven in Eclipse Juno', then only you will be able to understand this tutorial and even further web services tutorials.*

Required Files

pom.xml

web.xml

RestServicePathParamJava4s.java

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
  <modelVersion>4.0.0</modelVersion>
  <groupId>RestPathParamAnnotationExample</groupId>
  <artifactId>RestPathParamAnnotationExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <repositories>
      <repository>
        <id>maven2-repository.java.net</id>
        <name>Java.net Repository for Maven</name>
        <url>http://download.java.net/maven/2/</url>
        <layout>default</layout>
      </repository>
   </repositories>

```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.2</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-server</artifactId>
    <version>1.8</version>
  </dependency>

</dependencies>

 <build>
  <finalName>RestPathParamAnnotationExample</finalName>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <compilerVersion>1.5</compilerVersion>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>

</project>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XML
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>RestPathParamAnnotationExample</display-name>

  <servlet>
    <servlet-name>jersey-serlvet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-clas
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>com.java4s</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

```

```
17     <servlet-mapping>
18         <servlet-name>jersey-serlvet</servlet-name>
19         <url-pattern>/rest/*</url-pattern>
20     </servlet-mapping>
21 
22 </web-app>
23 <!-- www.Java4s.com -->
```

RestServicePathParamJava4s.java

```
1  package com.java4s;
2  
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.Response;
8  
9  @Path("/customers")
10 public class RestServicePathParamJava4s {
11 
12     @GET
13     @Path("{name}/{country}")
14     @Produces("text/html")
15     public Response getResultByPassingValue(
16                 @PathParam("name") String name,
17                 @PathParam("country") String country) {
18 
19         String output = "Customer name - "+name+", Country - "+country+"";
20         return Response.status(200).entity(output).build();
21 
22     }
23 }
```

Explanation

Right click on your project > *Run As* > *Run on Server*

By default eclipse will open http://localhost:2013/RestPathParamAnnotationExample/ with HTTP 404 Error

In web.xml we have specified URL pattern as /rest/* (line number 19) and in RestServicePathParamJava4s.java we specified class level @path as /customers [line number 9] and method level @path as {name}/{country} [line number 13]

So the final URL should be http://localhost:2013/RestPathParamAnnotationExample/rest/customers/Java4s/USA

Once you hit the URL, *http://localhost:2013/……/rest/../Java4s/USA* , last two parameters in this URL 'Java4s' and 'USA' are retrieved by *@PathParam("name")*, *@PathParam("country")* annotations in *RestServicePathParamJava4s.java* and will copy into *String name, String country*

respectively.

RestServiceJava4s.java | http://localhost:2013/RestPathAnnotationExample/rest/customers/Java

```
package com.java4s;

import javax.ws.rs.GET;

@Path("/customers")
public class RestServiceJava4s {

    @GET
    @Path("{name}/{country}")
    @Produces("text/html")
    public Response getResultByPassingValue(
                    @PathParam("name") String name,
                    @PathParam("country") String country) {

        String output = "Customer name is "+name+" Country is "+country+"";
        return Response.status(200).entity(output).build();

    }
}
```

Check the output

Output

http://localhost:2013/RestPathParamAnnotationExample/rest/customers/Java4s/USA

http://localhost:2013/RestPathParamAnnotationExample/rest/customers/Java4s/USA

Customer name - Java4s, Country - USA

You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

RESTful Web Services (JAX-RS) @QueryParam Example

Web Services »on Jul 8, 2014 { 6 Comments } By Sivateja

In RESTful web services (JAX-RS) @QueryParam annotation will be used to get the query parameters from the URL, Observe carefully, i am saying we will retrieve the parameters only not their values. But in case of @PathParam we will get parameter values directly.

Query Parameters Syntax

Consider this URL:

http://localhost:2013/RestPathAnnotationExample/rest/customers?nameKey=Java4s&countryKey=USA

Here query parameters are *nameKey*, *countryKey* and their values are *Java4s*, *USA* respectively, hope you understood.

Required Files

pom.xml & web.xml are similar to *previous article* no changes

RestServiceQueryParamJava4s.java

RestServiceQueryParamJava4s.java

```
1  package com.java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.QueryParam;
7  import javax.ws.rs.core.Response;
8
9  @Path("/customers")
10 public class RestServiceQueryParamJava4s {
11
12    @GET
13    @Produces("text/html")
14    public Response getResultByPassingValue(
15                  @QueryParam("nameKey") String name,
16                  @QueryParam("countryKey") String country) {
```

```
17
18        String output = "Customer name - "+name+", Country - "+country+"";
19        return Response.status(200).entity(output).build();
20
21    }
22 }
```

Explanation

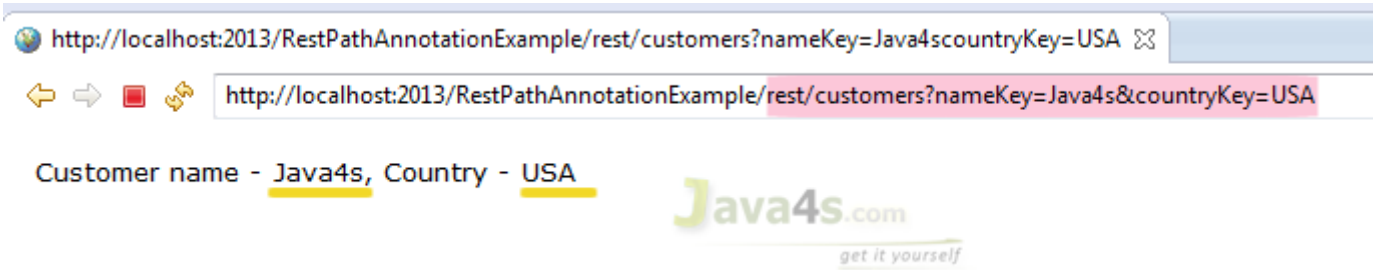
Right click on your project folder > *Run As* > *Run on Server*

Eclipse will open http://localhost:2013/RestQueryParamAnnotationExample/ with HTTP 404 Error

In web.xml we have specified the url pattern as /rest/* (line number 19) and in RestServiceQueryParamJava4s.java we specified class level @path as /customers [line number 9] and we are retrieving 2 query parameters [Line number 15,16], so our final URL should be the *http://localhost:2013/RestQueryParamAnnotationExample/rest/customers?nameKey=java4s&countryKey=USA*

Check the output by hitting the above URL

Output



You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

RESTful Web Services (JAX-RS) Annotations

Web Services »on Jul 6, 2014 { 8 Comments } By Sivateja

This tutorial explains important annotations of JAX-RS for creating RESTful web services, friends i am giving these annotations just for your understanding purpose. you better know about these annotations before we go forward with the remaining RESTful web services tutorials.

JAX-RS Annotations

@Path(`*Path*`)

@GET

@POST

@PUT

@DELETE

@Produces(MediaType.TEXT_PLAIN [, more-types])

@Consumes(type[, more-types])

@PathParam()

@QueryParam()

@MatrixParam()

@FormParam()

@Path() Annotation

Its a Class & Method level of annotation

This will check the path next to the base URL

Syntax :

Base URL :

http://localhost:(port)/<YourApplicationName>/<UrlPattern In Web.xml>/<path>

Here *<path> is the part of URI*, and this will be identified by @*path* annotation at class/method level, you will be able to understand in the next RESTful hello world tutorial.

@GET

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP GET request only, i mean if we annotate our method with @GET, the execution flow will enter that following method if we send GET request from the client

@POST

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP POST request only.

@PUT

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUT request only.

@DELETE

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETE request only.

@Produces

Its a method or field level annotation, This tells which MIME type is delivered by the method annotated with @GET. I mean when ever we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats. There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.
Remember: We will use @Produces annotation for GET requests only.

@Consumes

This is a class and method level annotation, this will define which MIME type is consumed by the particular method. I mean in which format the method can accept the input from the client.

Will discuss later regarding *@PathParam*, *@QueryParam*, *@MatrixParam*, *@FormParam* annotations , i will talk more about these annotations in the next examples.

You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client

Java4s.com

What is Web Services, Web Services Introduction

Web Services »on Jul 6, 2014 { 20 Comments } By Sivateja

What is Web Services ? Over the internet, you might have seen different kinds of definitions for Web services. My definition will almost resembles them Web Services, the name it self indicates that its a service which is available over the Web, that's it. As an example you can consider Java4s.com, When ever you hit the URL in the web browser it will gives you some output in HTML format, you can also consider this as a Web service. With web services, we can communicate different applications on different platforms, i mean a java application in Windows platform can easily communicate with the application developed using .net/php in Linux operation system.

Understanding SOAP and REST

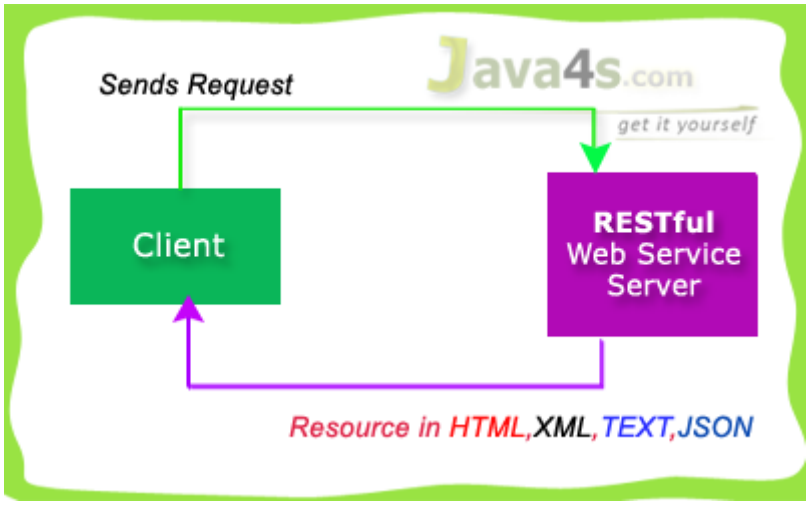
Web Services are mainly of 2 types, **SOAP** [Simple Object Access Protocol] and **REST** [Representational state transfer] based services. We have different type of specifications to implement SOAP and REST services. I believe so far you might be in confusion with these kind keywords like, JAX-RS, JAX-WS, RESTful, SOAP, Apache Axis2, Apache CXF bla bla… Let me try to bring you out of them.

> JAX-RS provides the implementation of RESTful web services, JAX-RS is a specification for RESTful Web Services with Java and it is given by Sun. Since it is a specification, other frameworks can be written to implement these specifications, and that includes *Jersey* from Oracle, *Resteasy* from Jboss, *CXF* from Apache bla bla.
>
> JAX-WS, Apache Axis2 provides the implementation for SOAP
>
> Apache CXF provides implementation for SOAP and RESTful services both.

RESTful

What ever the data/response we will get from the server is known as ***Resource*** [remember this point], Each resource can be accessed by its URI's. We can get the resource from RESTful service in different formats like, *HTML*,*XML*,*JSON*,*TEXT*,*PDF* and in the *Image* formats as well, but in real time we mainly we will prefer *JSON*. REST guidelines always talks about stateless communication between client and the Server. Stateless means, every single

request from client to server will be considered as a fresh request. Because of this reason REST always prefers to choose HTTP as it a stateless protocol.



RESTful used 4 main HTTP methods…

- **GET** - Retrieve Data
- **POST**- Create/Insert Data
- **PUT**- Update Data
- **DELETE**- Delete Data

Generally we will prefer RESTful Services in these scenarios…

- If clients require *caching*, means if you have limited bandwidth
- If you want every thing to be stateless [I have already explained about stateless]

But SOAP gives the output only in XML format. Hope you are good now 🙂 by the way we are going to use Jersey to implement JAX-RS specifications.

You Might Also Like

- Spring Boot – Introduction Tutorial (Don't Miss)
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Download Files from (JAX-RS) RESTful Web Service

Web Services »on Jul 10, 2014 { 5 Comments } By Sivateja

In this article i will show you how to download files from your JAX-RS web service. Downloading files from restful is easier compared to upload :-), however i will give you both examples. We can download any type of files from the RESTful web services, its just a matter of changing @produces annotation. For example..

We should annotate our method with

> @Produces("text/plain") If you are expecting Text file as response
> @Produces("image/your image type[.jpg/.png/.gif]") for downloading any Image files
> @Produces("application/pdf") for downloading PDF files

Lets discuss these three scenarios with an example.

Required Files

pom.xml & *web.xml* [Refer this *Restful Hello world example*, i am using the same xml's]

RestServiceFileDownloadJava4s.java

RestServiceFileDownloadJava4s.java

```
1  package com.java4s;
2
3  import java.io.File;import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.core.Response;
7  import javax.ws.rs.core.Response.ResponseBuilder;
8
9  @Path("/downnload")
10 public class RestServiceFileDownloadJava4s {
11
12    String path = "c:\\tuts\\java4s.txt";
13
14    /*    public File getCustomerDataFile() {
```

```
15          File file = new File(path);
16          return file;
17      }*/
18
19
20      @GET
21      @Path("/data")
22      @Produces("text/plain")
23      //@Produces("image/png")
24      //@Produces("application/pdf")
25      public Response getCustomerDataFile() {
26
27          File file = new File(path);
28
29          ResponseBuilder rb = Response.ok((Object) file);
30          rb.header("Content-Disposition","attachment; filename=java4sFileFromServer.txt");
31          return rb.build();
32      }
33  }
```

Explanation

Our intention is to download the TEXT file from JAX-RS, for that we need to annotate our method with @Produces(“text/plain”) [which i did in line number 24]

Once we call the RESTful service, i want to display a pop-up download box for the users to ‘download‘ that file, in order to do that we need to add ‘Content-Disposition‘ header to the response

But in the Response class we don’t have any option to add the headers, so firstly i have created ‘ResponseBuilder‘ object [line number 31, because in ResponseBuilder class we have direct method to add the *headers*], and added ‘***Content-Disposition***‘ to the header.

Finally called rb.build() [at line number 33], this will create a Response instance from the current ResponseBuilder object (rb) and returns

We can also get the output by simply writing the lines 16-19 but it wont shows download pop-up box 😀

You can enable, line numbers 25,26 if your file is Image & PDF respectively

Same thing will happen in case of ***Images/PDF*** or other file formats.

Output

DOWNLOAD

You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks.
You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

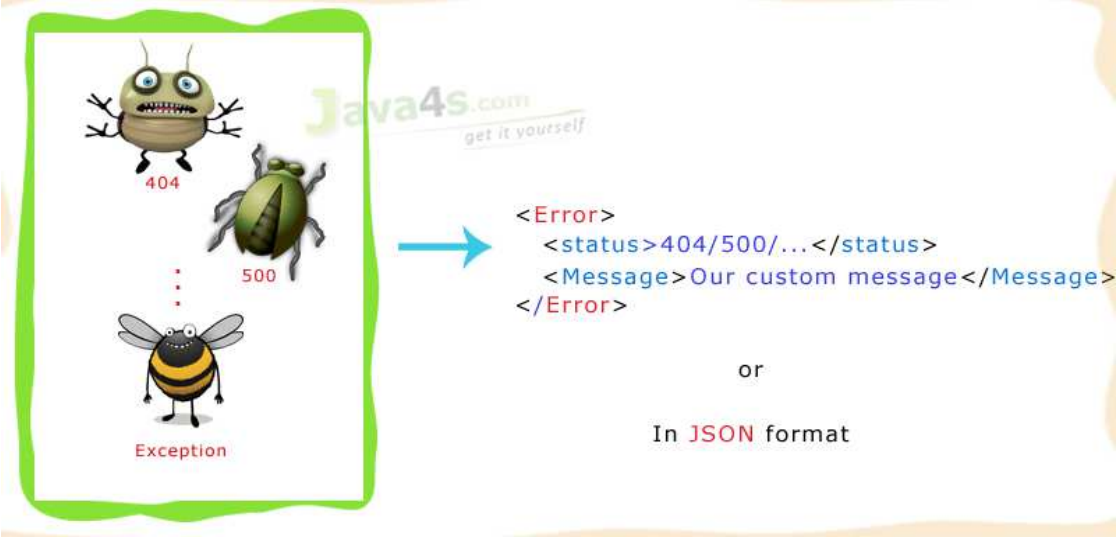
Comments

5 Responses to “Download Files from (JAX-RS) RESTful Web Service”

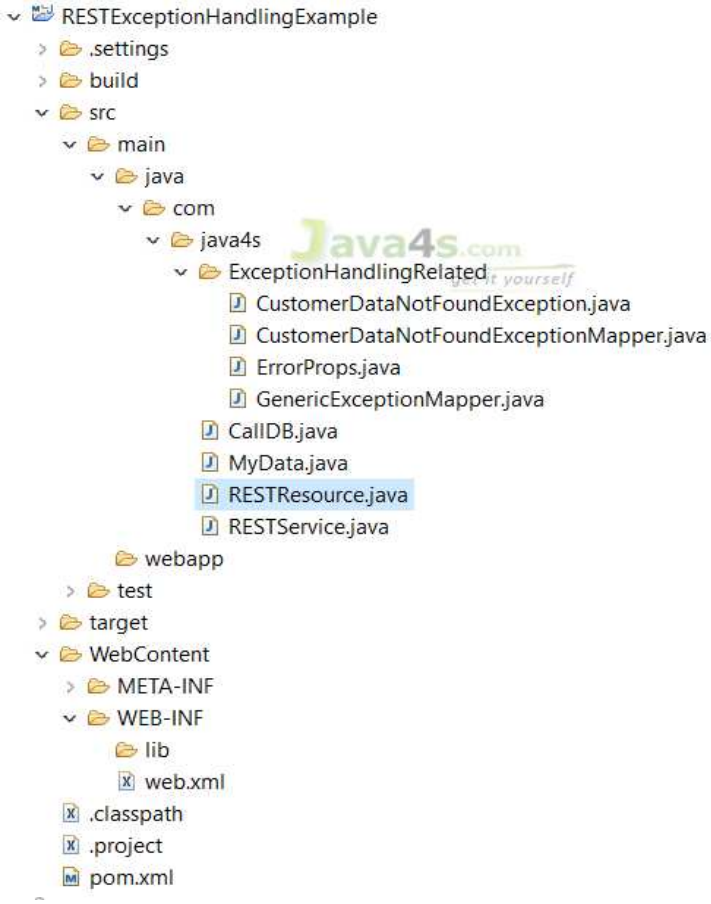
Exception Handling in RESTful Web Services (JAX-RS) with Jersey

Web Services »on May 21, 2017 { 8 Comments } By Sivateja

Exception handling in RESTful (JAX-RS) web services is a very important concept, in this article I am going to explain it step by step with an example. FYI. check this article for Creating Simple Maven RESTful Web Service Project in Eclipse. I am directly going to start with directory structure of the current example.



Directory structure



As we are dealing with exception handling, I am going to create a service which will throw an exception 😀

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/mave
<modelVersion>4.0.0</modelVersion>
<groupId>RESTExceptionHandlingExample</groupId>
<artifactId>RESTExceptionHandlingExample</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>

<repositories>
	<repository>
	  <id>maven2-repository.java.net</id>
	  <name>Java.net Repository for Maven</name>
	  <url>http://download.java.net/maven/2/</url>
	  <layout>default</layout>
	</repository>
</repositories>

<dependencies>
	<dependency>
	  <groupId>junit</groupId>
	  <artifactId>junit</artifactId>
	  <version>4.8.2</version>
	  <scope>test</scope>
	</dependency>
```

```
26
27      <dependency>
28        <groupId>com.sun.jersey</groupId>
29        <artifactId>jersey-server</artifactId>
30        <version>1.8</version>
31      </dependency>
32  </dependencies>
33
34  <build>
35    <finalName>RESTExceptionHandlingExample</finalName>
36    <plugins>
37      <plugin>
38        <artifactId>maven-compiler-plugin</artifactId>
39          <configuration>
40            <compilerVersion>1.8</compilerVersion>
41            <source>1.8</source>
42            <target>1.8</target>
43          </configuration>
44      </plugin>
45    </plugins>
46  </build>
47
48  </project>
```

web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3        xmlns="http://java.sun.com/xml/ns/j2ee"
4        xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
5        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
6    <display-name>RESTExceptionHandlingExample</display-name>
7    <servlet>
8      <servlet-name>jersey-serlvet</servlet-name>
9      <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
10     <init-param>
11       <param-name>com.sun.jersey.config.property.packages</param-name>
12       <param-value>com.java4s</param-value>
13     </init-param>
14     <load-on-startup>1</load-on-startup>
15   </servlet>
16   <servlet-mapping>
17     <servlet-name>jersey-serlvet</servlet-name>
18     <url-pattern>/*</url-pattern>
19   </servlet-mapping>
20 </web-app>
```

RESTResource.java

```
1  package com.java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
```

```
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

@Path("/customers")
public class RESTResource {

 @GET
 @Path("/checkProfile/{id}")
 public Response getAdminDetails(@PathParam("id") String id) {

 String msg = RESTService.checkCustomerStatus(id);

 return Response.status(200).entity(msg).build();
 }
}
```

RESTService.java

```
package com.java4s;

public class RESTService {

 CallDB cdb = new CallDB();

 public String checkCustomerStatus(String custId){

 MyData da = cdb.getStatus(custId);

 return da.getStatus().trim();
 }
}
```

CallDB.java

```
package com.java4s;

public class CallDB {

 public MyData getStatus(String custId) {

      // Lets say, database logic will go here and setting the output in MyData bean

        MyData da = new MyData();

           // da.setStatus("Valid");

        return da;

       }
}
```

MyData.java

```
1  package com.java4s;
2
3  public class MyData {
4
5   String status;
6
7   public String getStatus() {
8   return status;
9   }
10
11  public void setStatus(String status) {
12  this.status = status;
13  }
14 }
```

Explanation

So we are good to run the application now, if you observe I have just created 4 java classes till now..

RESTResource.java

RESTService.java

CallDB.java

MyData.java

The application will start by hitting the following URL [I am using 2017 as my server port, this might be different for your server]

```
http://localhost:2017/RESTExceptionHandlingExample/customers/checkProfile/100
```

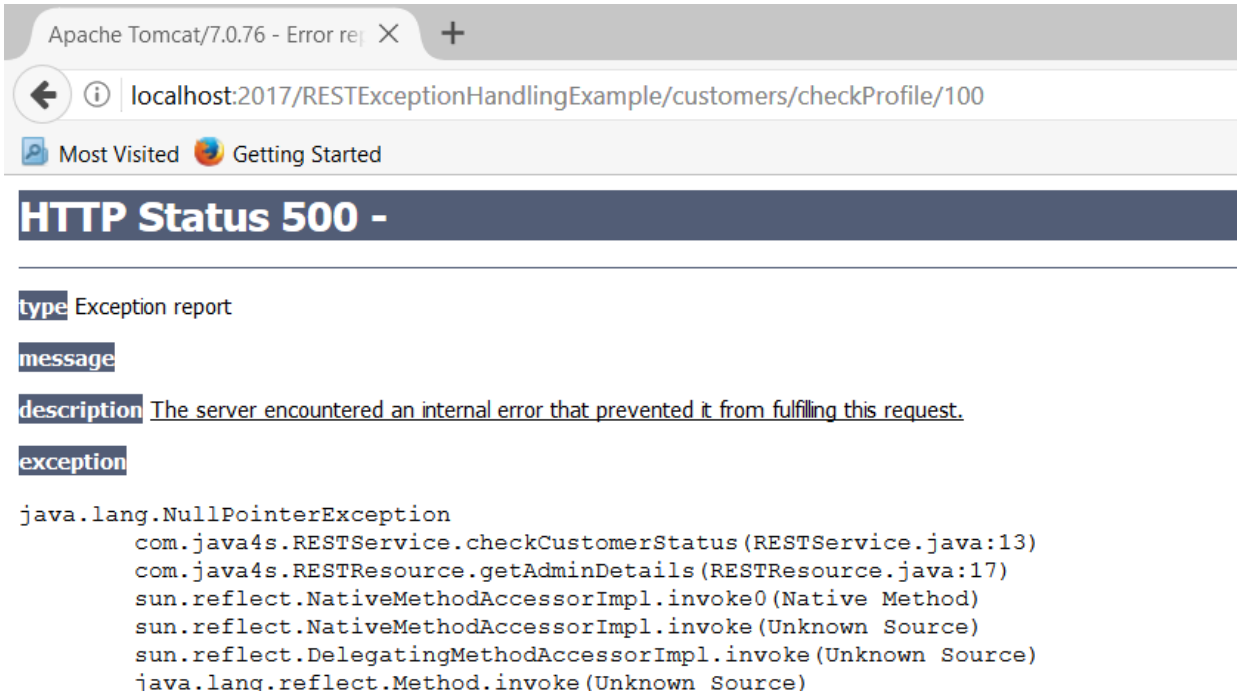
If you hit the above URL, the flow will come to RESTResource.java, in that at line number 17, I am calling checkCustomerStatus(–) function of RESTService class, by passing the id

Consider, CallDB is a class which handle all database related stuff, so in RESTService.java > line number 11, I am calling getStatus(–) of CallDB

In CallDB.java > consider we had a database call at line number 7, after that creating MyData class object and setting all the data that I retrieved from the database and returning it at line number 13, if you observe line number 11, I have commented the setter method, means I am not setting status, so by default it contains NULL value (if we call its getter method, it will return NULL)

Now the flow will come to RESTService.java > at line number 11, I am calling getStatus(), as this gives null value [as we have not set anything in CallDB > line 11], on that null I am calling .trim() again, which will give NullPointerException

Hmm.. so successfully we are able to create a service, which will throw a NullPointerException If you hit the URL, you will see..



Its the Tomcat default error page, showing the exception as NullPointerException but I don't think its the right way of displaying the errors to the consumers! lets try to display the error with some custom error message, in order to do that I am going to create a custom (user defined) run time exception, lets say my custom exception class name is CustomerDataNotFoundException.

CustomerDataNotFoundException.java

```
1  package com.java4s.ExceptionHandlingRelated;
2
3  public class CustomerDataNotFoundException extends RuntimeException{
4
5   private static final long serialVersionUID = 1L;
6
7   public CustomerDataNotFoundException(String exceptionMsg)
8   {
9   super(exceptionMsg);
10  }
11
12 }
```

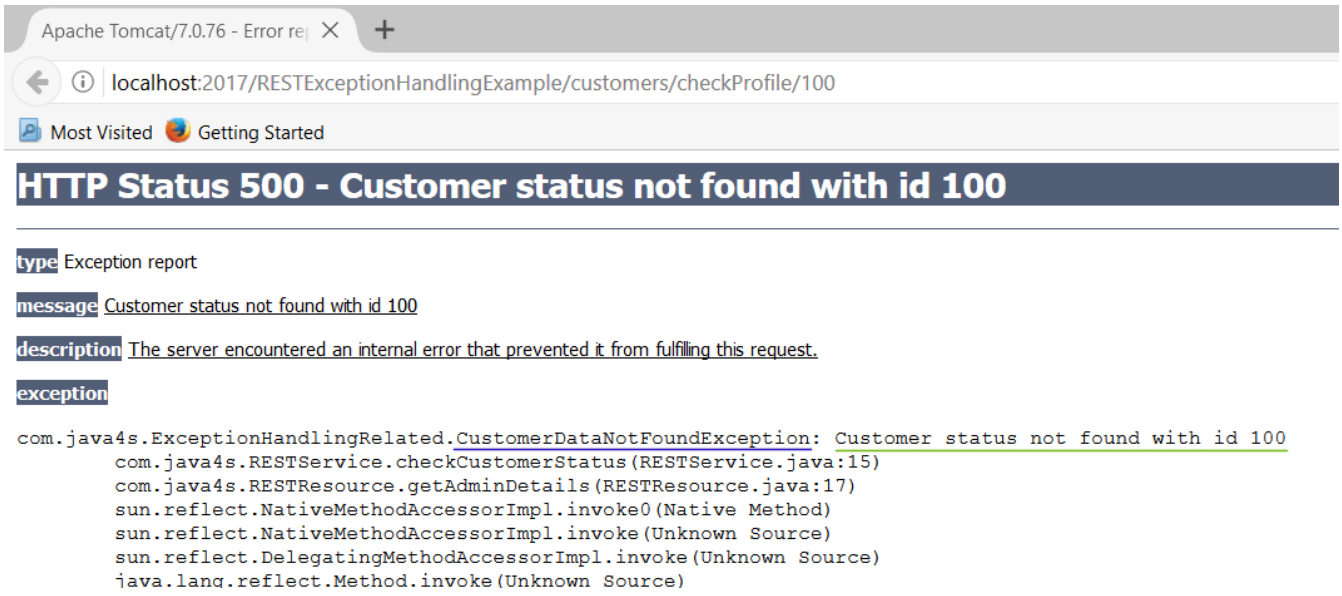
Here I have just created an exception class with constructor which takes String as an argument, now let me change RESTService.java as..

RESTService.java

```
1  package com.java4s;
2
3  import com.java4s.ExceptionHandlingRelated.CustomerDataNotFoundException;
4
5  public class RESTService {
6
7   CallDB cdb = new CallDB();
8
```

```
 9  public String checkCustomerStatus(String custId){
10
11  MyData da = cdb.getStatus(custId);
12
13  if(da.getStatus() == null)
14  {
15  throw new CustomerDataNotFoundException("Customer status not found with id "+custId);
16  }
17
18  return da.getStatus().trim();
19
20  }
21 }
```

I am checking the status with if condition. If status is NULL, throwing out the custom exception by passing some meaningful message. Lets run the application and see..



Its throwing our exception with the message we have sent, not bad 🙂 but still this is not the right way of showing the errors.

What happens behind the scenes

Actually we are throwing CustomerDataNotFoundException if the status is NULL, if you observe, we are not handling that exception in RESTService, instead simply throwing with our message. So the exception keeps bubbling up and will come to RESTResource (here also we are not handling) and so from there to JAX-RS and finally will reach Tomcat server container, and server will show its default error page, that's what we are seeing in the above image.

So in order to stop exception bubbling up to the tomcat server container, we need to create an exception mapper.

CustomerDataNotFoundExceptionMapper.java

```
1  package com.java4s.ExceptionHandlingRelated;
```

```
2
3  import javax.ws.rs.core.Response;
4  import javax.ws.rs.core.Response.Status;
5  import javax.ws.rs.ext.ExceptionMapper;
6  import javax.ws.rs.ext.Provider;
7
8  @Provider
9  public class CustomerDataNotFoundExceptionMapper implements ExceptionMapper<CustomerD
10
11  public Response toResponse(CustomerDataNotFoundException ex)
12  {
13  return Response.status(Status.NOT_FOUND)
14  .entity(new ErrorProps("404", ex.getMessage()))
15  .build();
16  }
17 }
```

ErrorProps.java

```
1  package com.java4s.ExceptionHandlingRelated;
2
3  import javax.xml.bind.annotation.XmlRootElement;
4
5  @XmlRootElement
6  public class ErrorProps {
7
8   private String status;
9   private String errorMessage;
10
11  public ErrorProps(){}
12
13  public ErrorProps(String statusFromOutside, String errorMessageFromOutside)
14  {
15  this.status = statusFromOutside;
16  this.errorMessage = errorMessageFromOutside;
17  }
18
19
20  public String getErrorMessage() {
21  return errorMessage;
22  }
23  public void setErrorMessage(String errorMessage) {
24  this.errorMessage = errorMessage;
25  }
26  public String getStatus() {
27  return status;
28  }
29  public void setStatus(String status) {
30  this.status = status;
31  }
32
33 }
```

Explanation

Created an exception mapper CustomerDataNotFoundExceptionMapper for CustomerDataNotFoundException

All exception mappers should implement ExceptionMapper interface of type generic, for now I am going to use this exception mapper only for our exception, so I have implemented ExceptionMapper of type CustomerDataNotFoundException [check at line number 9]

We need to override the toResponse method of ExceptionMapper interface, which takes exception as an argument, in this case CustomerDataNotFoundException

I want to display my exception details as an XML, so created a simple java model ErrorProps.java and annotated with @XmlRootElement

Now come back to mapper class toResponse method, there I am returning Response object

> Response.status(–) :- setting the current status
> .entity(–) :- passing ErrorProps class object by setting required values, here I am setting status as 404, and our custom exception message

Finally annotate our mapper class with @Provider annotation, so that JAX-RS will register this mapper to intercept the response when particular exception was thrown

Go ahead and run the application and see…

We did it 😀

But this is only for NullPointerException, but how about the other exceptions ? for that we need to modify the mapper. Let me do it by creating new mapper class.

GenericExceptionMapper.java

```
1  package com.java4s.ExceptionHandlingRelated;
2
3  import javax.ws.rs.core.Response;
4  import javax.ws.rs.core.Response.Status;
5  import javax.ws.rs.ext.ExceptionMapper;
6  import javax.ws.rs.ext.Provider;
7
8  @Provider
9  public class GenericExceptionMapper implements ExceptionMapper<Throwable>{
10
```

```
11  public Response toResponse(Throwable ex)
12  {
13  if(ex instanceof CustomerDataNotFoundException)
14  {
15
16  return Response.status(Status.NOT_FOUND)
17        .entity(new ErrorProps("404", ex.getMessage()))
18        .build();
19  }
20  else
21  {
22  return Response.status(Status.INTERNAL_SERVER_ERROR)
23  .entity(new ErrorProps("Some error code, 500 or somthing", ex.getMessage()))
24  .build();
25  }
26  }
27
28  }
```

What are the changes ?

Implement ExceptionMapper of type Throwable, instead of our own exception. If you check the above class, at line number 13, I am checking whether the Throwable is the instance of CustomerDataNotFoundException, if its true, I will build my Response accordingly, like this you can handle any number of exceptions in a single class, you can download and play with it.

That's it friends, hope you enjoy the article

You Might Also Like

- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey
- How to Test (JAX-RS) RESTful Web Services
- RESTful Web Service (JAX-RS) JSON Example Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks.
You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

How RESTful Web Services Extract Input Parameters

Web Services »on Jul 6, 2014 { 8 Comments } By Sivateja

In this article i will show you how a RESTful web service will extract input parameters from the client request. We have different ways of sending input values to the rest services, and RESTful web service extract those details based upon the client URL pattern. In JAX-RS we can use the following annotations to extract the input values sent by the client.

@PathParam
@QueryParam
@MatrixParam
@FormParam

@*PathParam*,@*QueryParam*,@*MatrixParam* are parameter annotations which allows us to map variable URI path fragments into your method call. Confused ? In simple words, these three annotations will come into picture in case if we are passing the input values to the restful service through the URL. After that Rest service will extract those values by using these annotations. Regarding @*FormParam,* restful web service will use this annotation to retrieve the values sent by the client through some HTML/JSP form.

@PathParam URL Syntax

```
http://localhost:7001/<Rest Service Name>/rest/customers/100/Java4s
```

Did you observe the two parameters appear in the end of the above URL [*100* & *Java4s*], which are separated by forward slash(/) are called as path parameters, as of now just remember the syntax, going forward i will give you an example on each annotation.

@QueryParam URL Syntax

```
http://localhost:7001/…/rest/customers?
custNo=100&custName=Java4s
```

If the client sends an input in the form of query string in the URL, then those parameters are called as Query Parameters. If you observe the above syntax, client passing 2 parameters *100* and *Java4s* started after question mark (?) symbol and each parameter is separated by & symbol, those parameters are called as query parameters.

@MatrixParam URL Syntax

```
http://localhost:7001/
…/rest/customers;custNo=100;custName=Java4s
```

Matrix parameters are another way defining the parameters to be added to URL. If you observe the above syntax, client is passing two parameters each are separated by semicolon, these parameters are called as matrix parameters. Remember these parameters may appear any where in the path.

@FormParam URL Syntax

Finally form parameters, if we have a HTML form having two input fields and submit button. Lets client enter those details and submit to the RESTful web service. Then the rest service will extract those details by using this @*FormParam* annotation.

For now just remember these consents, going forward i will give you an example on each annotation.

You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

How to Test (JAX-RS) RESTful Web Services

Web Services »on Jul 20, 2014 { 1 Comment } By Sivateja

In this article i will show you how to test RESTful web service (JAX-RS), so far we have learned how to create a RESTful service and testing GET and POST requests through some web browser. But in real time projects we will use different tools to test RESTful web services. If you would like to test JAX-RS with web browser you can use the following tools…

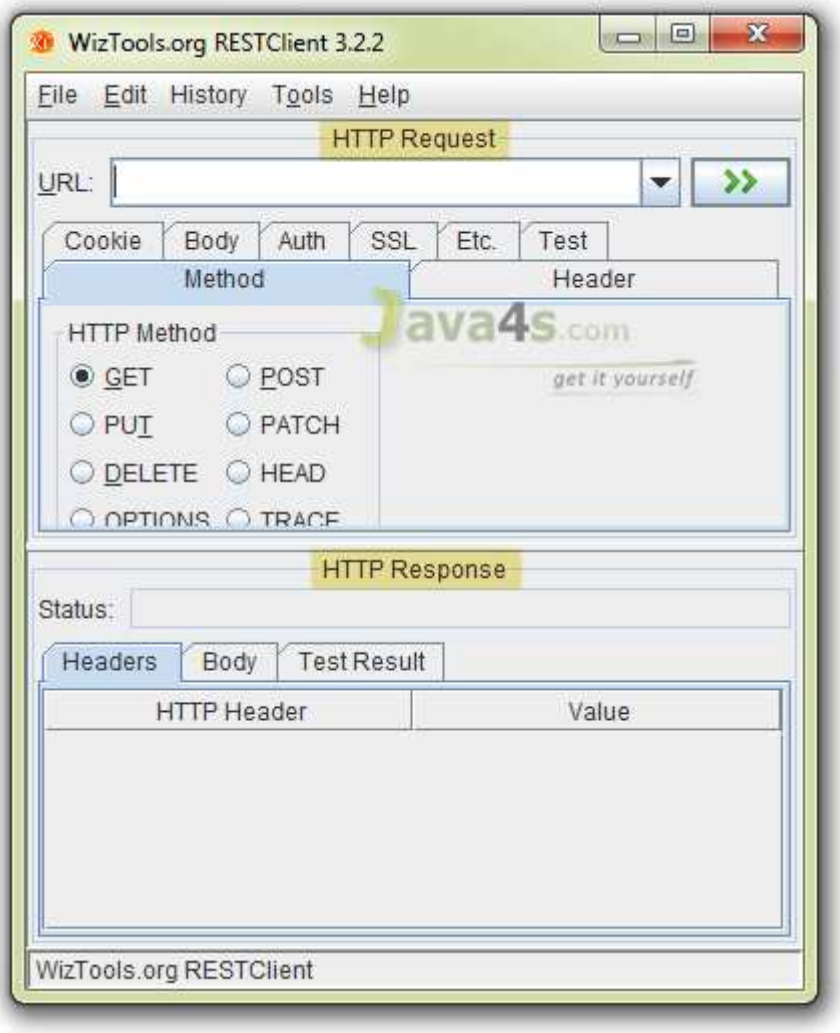
- Postman [Chrome Extension]
- REST Client [Chrome Extension]
- Advanced REST Client [Chrome Extension]
- Rest Client [Firefox Add-On]

If you would like to test JAX-RS in your local

- RESTClient UI
- SoupUi

In this tutorial i will show you how to test jax-rs with RESTClient UI

- Click here to Download RESTClient UI
- Open the above link and download *restclient-ui-3.2.2-jar-with-dependencies.jar*
- We are done, you no need to do any configurations kind of things, Just double click on the downloaded jar file to run the application, it looks like…



Lets take an example with *GET, POST, PUT, DELETE* for testing the web service.

Required Files

pom.xml

web.xml

TestingRestfulWebService.java

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
  <modelVersion>4.0.0</modelVersion>
  <groupId>TestRestfulWebServiceExample</groupId>
  <artifactId>TestRestfulWebServiceExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <repositories>
      <repository>
        <id>maven2-repository.java.net</id>
        <name>Java.net Repository for Maven</name>
        <url>http://download.java.net/maven/2/</url>
        <layout>default</layout>
      </repository>
```

```
    </repositories>

    <dependencies>
       <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.8.2</version>
          <scope>test</scope>
       </dependency>

       <dependency>
          <groupId>com.sun.jersey</groupId>
          <artifactId>jersey-server</artifactId>
          <version>1.8</version>
       </dependency>

    </dependencies>

     <build>
       <finalName>TestRestfulWebServiceExample</finalName>
       <plugins>
          <plugin>
             <artifactId>maven-compiler-plugin</artifactId>
             <configuration>
                <compilerVersion>1.5</compilerVersion>
                <source>1.5</source>
                <target>1.5</target>
             </configuration>
          </plugin>
       </plugins>
    </build>

</project>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
   xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XML
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
   <display-name>TestRestfulWebServiceExample</display-name>

   <servlet>
      <servlet-name>jersey-serlvet</servlet-name>
      <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-clas
      <init-param>
         <param-name>com.sun.jersey.config.property.packages</param-name>
         <param-value>com.java4s</param-value>
      </init-param>
      <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>
```

TestingRestfulWebService.java

```
package com.java4s;

import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

@Path("/customers")
public class TestingRestfulWebService {

    @GET
    @Produces("text/plain")
    @Path("{id}")
    public Response getCustomerDetails(@PathParam("id") String custId) {

        //CODE TO FETCH CUSTOMER DETAILS FROM THE DATABASE USING CUSTOMER I
       String output = "Customer Details With ID "+custId+" Has Been fetched Successfu
       return Response.status(200).entity(output).build();
     }

    @POST
    @Produces("text/plain")
    @Path("{id}")
    public Response insertCustomer(@PathParam("id") String custId) {

        //CODE TO INSERT CUSTOMER DETAILS USING CUSTOMER ID
       String output = "Customer Data With ID "+custId+" Has Been Saved Successfully"
       return Response.status(200).entity(output).build();
     }

    @PUT
    @Produces("text/plain")
    @Path("{id}")
    public Response updateCustomerDetails(@PathParam("id") String custId) {

```

```
39          //CODE TO UPDATE CUSTOMER DETAILS USING CUSTOMER ID
40          String output = "Customer Data With ID "+custId+" Has Been Updated Successful
41          return Response.status(200).entity(output).build();
42      }
43
44      @DELETE
45      @Produces("text/plain")
46      @Path("{id}")
47      public Response deleteCustomer(@PathParam("id") String custId) {
48
49          //CODE TO DELETE CUSTOMER DETAILS USING CUSTOMER ID
50          String output = "Customer With ID "+custId+" Has Been Deleted From the Databa
51          return Response.status(200).entity(output).build();
52      }
53
54  }
55
```

Testing JAX-RS GET Request

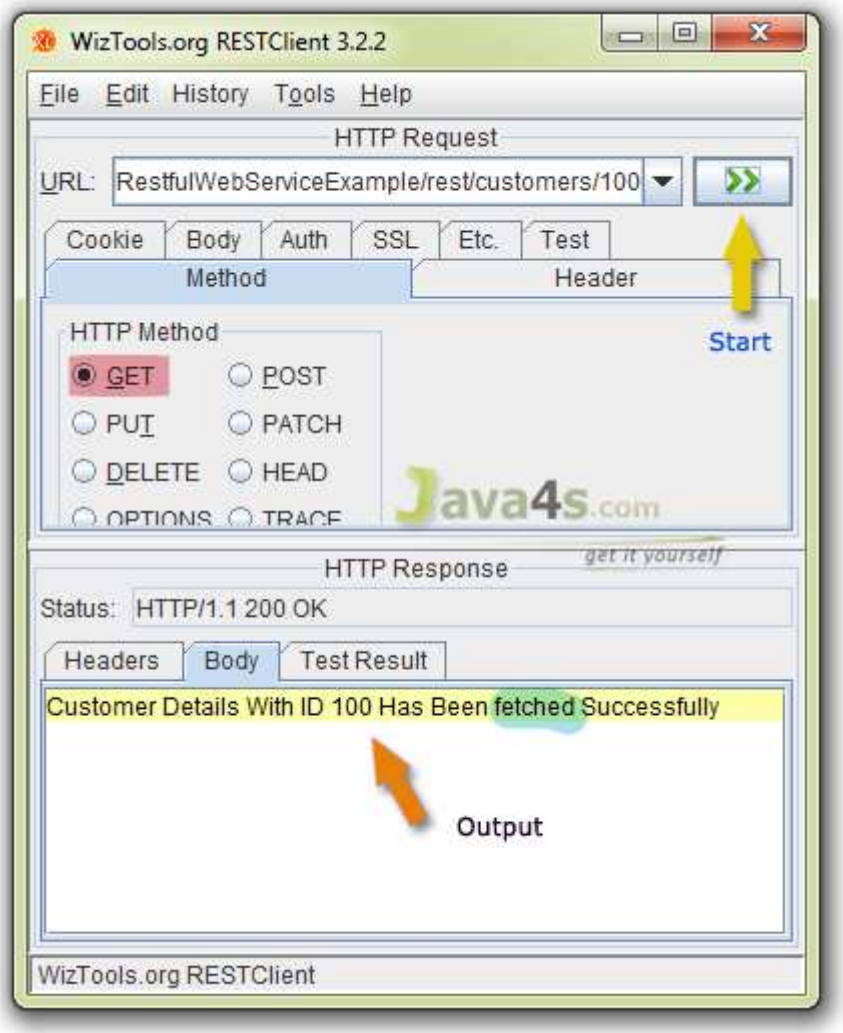
Eclipse > Run the application > Now open *Restclient UI*

In the URL field enter

http://localhost:2013/TestRestfulWebServiceExample/rest/customers/100

Choose GET method in the 'HTTP Method' options > now hit the start button and check the output

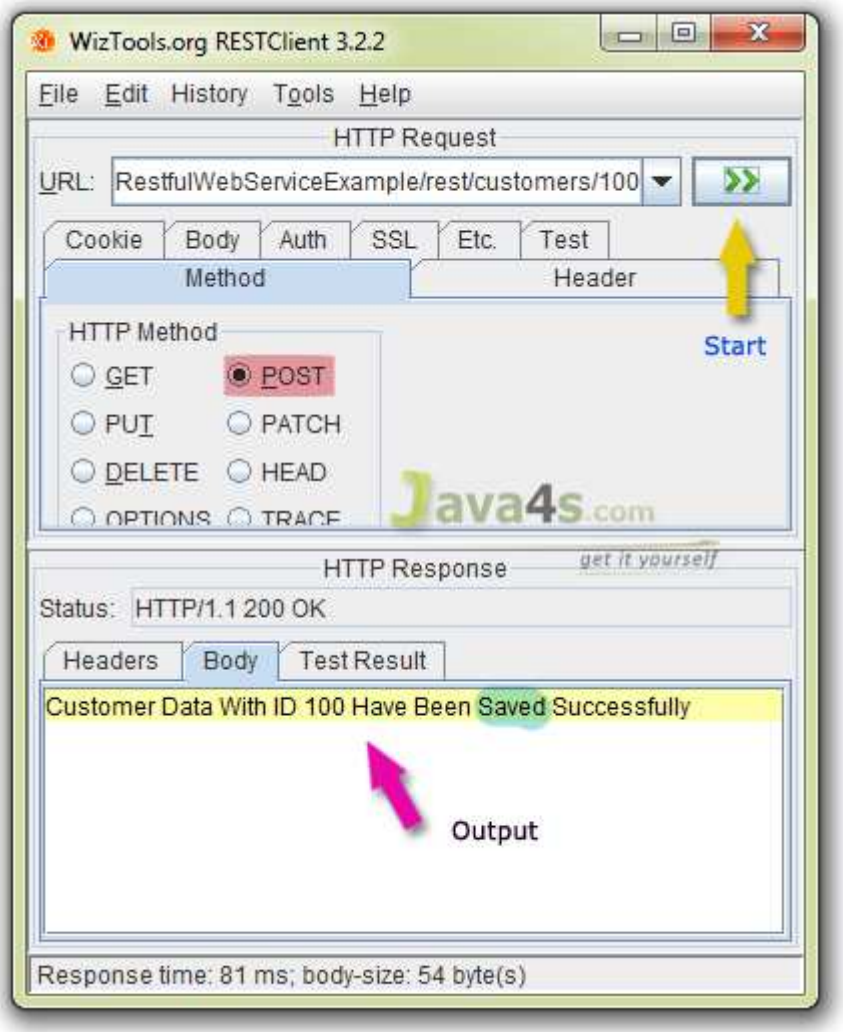
Output



Testing JAX-RS POST Request

Choose POST method in the 'HTTP Method' options > Hit the start button and check the output

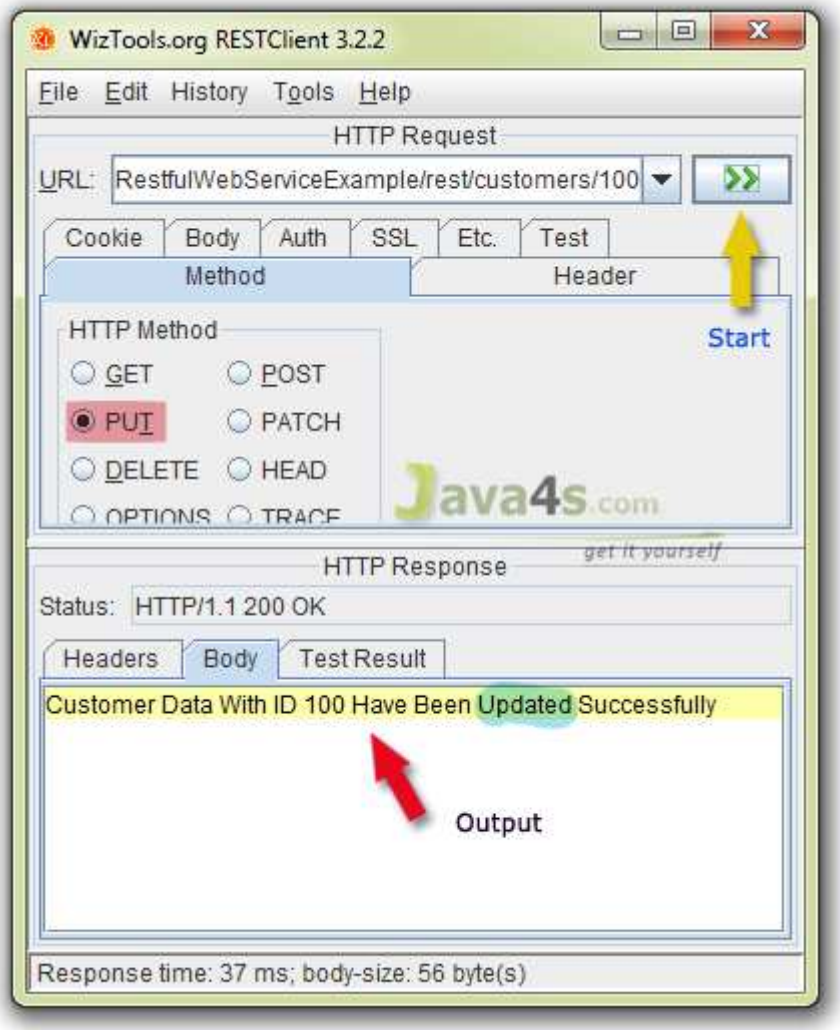
Output



Testing JAX-RS PUT Request

Choose PUT method in the 'HTTP Method' options > Hit the start button and check the output

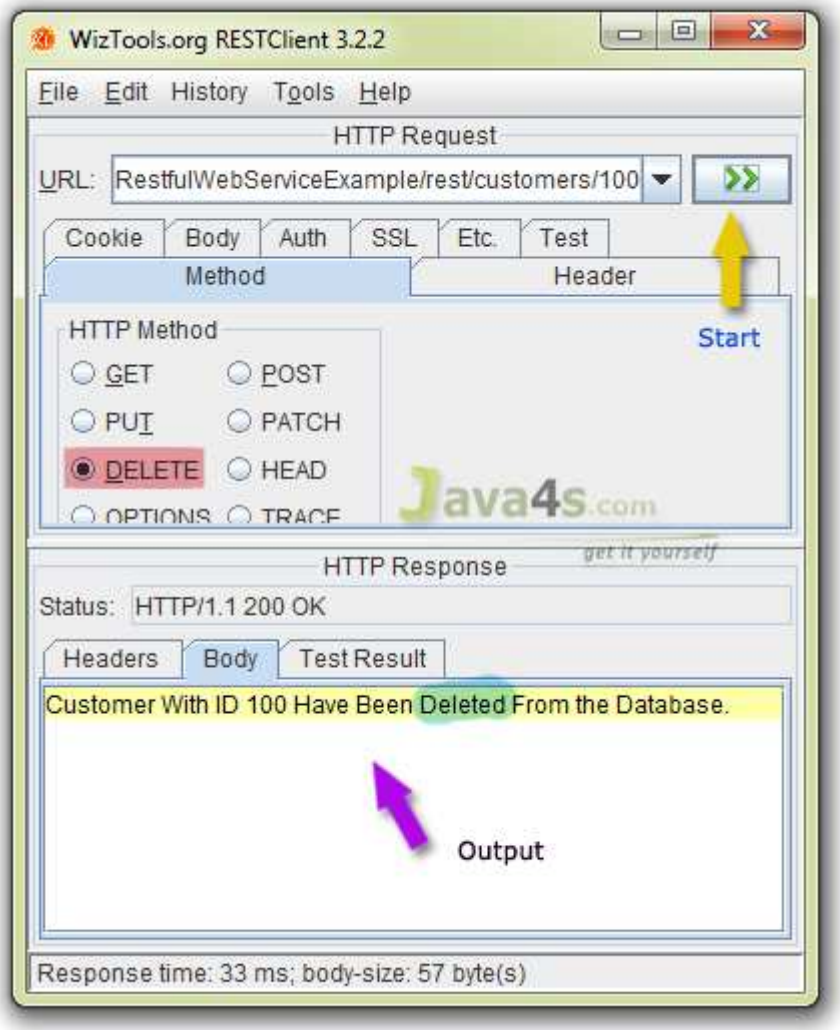
Output



Testing JAX-RS DELETE Request

Choose DELETE method in the 'HTTP Method' options > Hit the start button and check the output

Output



You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- JAX-RS Example of Multiple Resource Formats
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

JAX-RS Example of Multiple Resource Formats

Web Services »on Aug 6, 2014 { 22 Comments } By Sivateja Like 0

This article will describe how a RESTful web service produces multiple output formats. In the previous articles we came across how a RESTful service produces either XML or JSON alone as an output, but in this article i will show you what are the changes or steps need to be followed to let the rest service to produce multiple output formats from a ***single*** method, its the real time way of creating the services. Before you start reading this article, just have a look at the following articles for better understanding.

RESTful Web Service (JAX-RS) JSON Example Using Jersey

JAX-RS XML Example With JAXB Using Jersey

Exactly, exactly this current example is the combination of above two articles 😃 I am just copy & pasting those articles and doing some changes here that's it.

Required Files

pom.xml

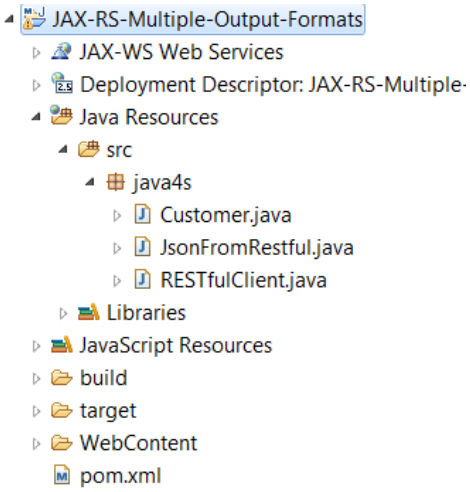
web.xml

Customer.java

JsonFromRestful.java

RESTfulClient.java

Directory Structure



pom.xml

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>JAX-RS-Multiple-Output-Formats</groupId>
4    <artifactId>JAX-RS-Multiple-Output-Formats</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6    <packaging>war</packaging>
7
```

```
    <repositories>
        <repository>
          <id>maven2-repository.java.net</id>
           <name>Java.net Repository for Maven</name>
           <url>http://download.java.net/maven/2/</url>
           <layout>default</layout>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-server</artifactId>
            <version>1.8</version>
        </dependency>

        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-json</artifactId>
            <version>1.8</version>
       </dependency>

        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-client</artifactId>
            <version>1.8</version>
        </dependency>
    </dependencies>

    <build>
      <finalName>JAX-RS-Multiple-Output-Formats</finalName>
      <plugins>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
              <compilerVersion>1.5</compilerVersion>
              <source>1.5</source>
              <target>1.5</target>
            </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/
  <display-name>JAX-RS-Multiple-Output-Formats</display-name>
  <servlet>
      <servlet-name>jersey-serlvet</servlet-name>
```

```
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
          <param-name>com.sun.jersey.config.property.packages</param-name>
          <param-value>java4s</param-value>
        </init-param>
        <init-param>
          <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
          <param-value>true</param-value>
        </init-param>
        <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>
```

Customer.java

```
package java4s;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

import com.sun.xml.txw2.annotation.XmlAttribute;

@XmlRootElement(name = "customer")
public class Customer {

   private int custNo;
   private String custName;
   private String custCountry;

   @XmlAttribute
   public int getCustNo() {
      return custNo;
   }
   public void setCustNo(int custNo) {
      this.custNo = custNo;
   }

   @XmlElement
   public String getCustName() {
      return custName;
   }
   public void setCustName(String custName) {
      this.custName = custName;
   }

   @XmlElement
   public String getCustCountry() {
      return custCountry;
   }
   public void setCustCountry(String custCountry) {
      this.custCountry = custCountry;
   }
}
```

JsonFromRestful.java

```
1  package java4s;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
6  import javax.ws.rs.Produces;
7
8  @Path("/customers")
9  public class JsonFromRestful {
10
11     @GET
12     @Path("/{cusNo}")
13     @Produces("application/xml,application/json")
14     //@Produces(MediaType.APPLICATION_JSON)
15     public Customer produceCustomerDetailsinJSON(@PathParam("cusNo") int no) {
16
17        Customer cust = new Customer();
18           cust .setCustNo(no);
19           cust .setCustName("Java4s");
20           cust .setCustCountry("India");
21        return cust;
22     }
23 }
```

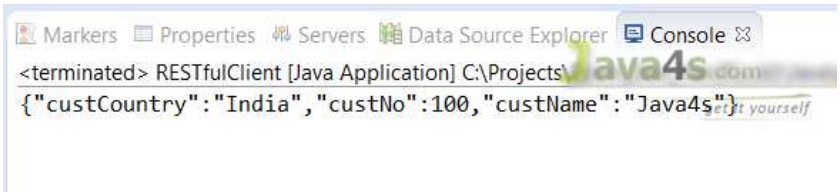
RESTfulClient.java

```
1  package java4s;
2
3  import com.sun.jersey.api.client.Client;
4  import com.sun.jersey.api.client.ClientResponse;
5  import com.sun.jersey.api.client.WebResource;
6
7  public class RESTfulClient {
8
9     public static void main(String[] Java4s) {
10
11       try {
12          Client client = Client.create();
13          WebResource resource = client.resource("http://localhost:2015/JAX-RS-Multiple-Output-Formats
14          ClientResponse response = resource.accept("application/json").get(ClientResponse.class);
15
16          if(response.getStatus() == 200){
17
18             String output = response.getEntity(String.class);
19             System.out.println(output);
20
21          }else System.out.println("Somthing went wrong..!");
22
23        } catch (Exception e) {
24              e.printStackTrace();
25        }
26
27     }
28 }
```

Execution & Output

Eclipse > right click on RESTfulClient.java > Run As > Run on Server

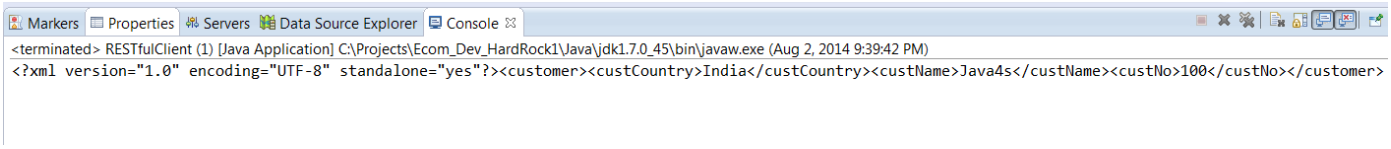
As per above program it will give you the JSON output



If you want to see the output in XML format. Open RESTfulClient.java > in line number 14 replace that line with

```
ClientResponse response =
resource.accept("application/xml").get(ClientResponse.class);
```

and the XML output will be



You Might Also Like

- Spring Boot – Creating a RESTful Web Service Example
- Exception Handling in RESTful Web Services (JAX-RS) with Jersey
- RESTful Java Client Example Using Jersey Client
- JAX-RS XML Example With JAXB Using Jersey
- How to Test (JAX-RS) RESTful Web Services

::. About the Author .::

Sivateja Kandula - Full Stack Java/J2EE & UI Web Developer

Founder of Java4s - Get It Yourself, A popular Java/J2EE Programming Blog, Love Java and UI frameworks. You can sign-up for the Email Newsletter for your daily dose of Java tutorials.

Comments

22 Responses to “JAX-RS Example of Multiple Resource Formats”

Madiraju Krishna Chaitanya says:
August 11, 2014 at 5:36 AM

Hi Sivateja Kandula Ji,Good Morning.Thanks a LOT for providing and sharing knowledge on WebServices with all of us.I was searching for an authentic content with real time