Implementing RESTful Web Services in Java

JAX-RS (JSR-311) is a specification providing for RESTful services capability in a Java EE environment. It promises to give you a viable alternative to traditional, SOAP-based Web services.

Java servlets are commonly used to develop RESTful applications. There is no prescriptive pattern for using servlets. Usually, the servlet will accept the request and parse the HTTP request URI itself, looking to match the request to a known resource. For REST services development, the simple servlet model is extended in more formalized APIs. As APIs developed on top of the servlet model, though, none of them were developed as a formal standard.

What you **need in WEB-XML?**

```
<servlet>
        <description></description>
        <servlet-name>ServletAdaptor</servlet-name>
        <servlet-class>
                com.sun.jersey.server.impl.container.servlet.ServletAdaptor
        </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
<servlet-mapping>
        <servlet-name>ServletAdaptor</servlet-name>
        <url-pattern>/rs/*</url-pattern>
</servlet-mapping>
```

What JARs you need?

Actually you just need:
jsr311-api-1.0.jar
jersey-core-1.0.3.jar
jersey-server-1.0.3.jar

How do you create a Service Class? What to Extend? What to implement?

Answer is: **Nothing.**

You don't have to implement or extend anything to make a class as RESTful Web Service. Just adding @Path will make a class visible from external URL and can be called.

In the example below, CustomerService class can be approached with

http://server:post/context/rs/customerService

```
@Path("/customerService")
public class CustomerService
{

}
```

Magic of making this class available is **/rs/ and annotation @Path.**

How do you make a method as WebMethod?

Declaring a method needs three minimum annotations like in Example below.

```
@GET
@Produces("application/xml")
@Path("/<add value here>")
public String myMethod()
```

This makes a method as good as HTTP GET call (Servlet's doGet() method). Let's understand these annotations:

@GET – To create method supporting HTTP Get

@POST – To Create method supporting HTTP POST

@Produces – What is returned by this method?

@Path – How to access this method?

If @Path is set as ("myMethod"), myMethod can be accessed with following URL:

http://server:post/context/rs/customerService/myMethod

How do you pass parameters to methods?

Your Parameters declarations are depend on HTTP Method type. For

@GET – You will accept @QueryParam declared as:

```
@QueryParam("<add name here>") String data
```

So method will now look like:

```
@GET
@Produces("application/xml")
@Path("/myMethod")
public String myMethod(@QueryParam("<add name here>") String data)
```

Considering parameter name is @QueryParm("data") then you can pass this parameter as:

http://server:post/context/rs/customerService/myMethod?data=1234

What if Parameter is not passed? – No Errors, it will come as null.
What if I have to give a default value to parameter? Sure. Declare @default like below:

```
@DefaultValue("1") @QueryParam("version") int version
```

Will that work for HTTP POST as well?

No. HTTP POST parameters should be declared with @FormParam like below:

```
@POST
@Produces("application/xml")
@Path("/<add value here>")
public String myPostMethod(@FormParam("data") String data)
```

Now you can call this method with a HTML Post method:

```
<form method="POST"
action="http://server:post/context/rs/customerService/myPostMethod">
<input id="data" type="text" height="30" width="20" />
<input type="submit"/>
</form>
```

What if I have to return something else than String?

Yes. You can return whatever you want; just making sure your client understands it. String is best to return XML or JSON response which is understood by HTML or UI clients.

Does REST support @PUT, @DELETE, and @HEAD HTTP Methods?

Answer is Yes, but 99.913323% times you will be using @GET and @POST.