



Spring MVC 4.0: Consuming RESTful Web Services using RestTemplate

<http://www.programming-free.com/2014/04/spring-mvc-consuming-restful-web-services.html>



This article is a continuation of my previous article on [Spring MVC 4.0 RESTful web services](#). So far I have written two articles on how to create restful web service in Spring MVC 4.0. Last tutorial explained how to create a RESTful web service in spring that would return user information from mysql table in JSON format. Let us implement a Spring MVC application that issues web service requests and fetches the response returned by the web service.

Articles on Spring MVC RESTful Web Services

[CRUD using Spring Data Rest and AngularJS using Spring Boot](#)

[CRUD using Spring MVC 4.0 RESTful Web Services and AngularJS](#)

[Spring MVC 4.0 RESTful Web Services Simple Example](#)

[Spring MVC 4.0 RESTful Web Service JSON Response with @ResponseBody](#)

[Spring MVC 4.0: Consuming RESTful Web Services using RestTemplate](#)

In this tutorial, we will extend our previous example to include a class that fetches user data from the spring service we had already created and display it in a jsp (view). Spring MVC is a complete framework with lot of inbuilt features that facilitates easy and quick web development, including a template class - **RestTemplate** for consuming web services.

I assume that you have gone through my previous article and have created a web service as explained. I am not going to repeat anything here. Let me start from where I left in my previous tutorial. You can download the sample application provided in my [previous tutorial](#) and start from there. Make sure to create mysql table and change the connection string configuration in the downloaded application.

RestTemplate

Spring MVC provides a template class called RestTemplate to support client side access to REST web service. This class has methods to support the six HTTP methods listed below,

HTTP	RESTTEMPLATE
DELETE	<code>delete(String, String...)</code>
GET	<code>getForObject(String, Class, String...)</code>
HEAD	<code>headForHeaders(String, String...)</code>
OPTIONS	<code>optionsForAllow(String, String...)</code>
POST	<code>postForLocation(String, Object, String...)</code>
PUT	<code>put(String, Object, String...)</code>

Table Source : <https://spring.io/blog/2009/03/27/rest-in-spring-3-resttemplate>

The above link has an excellent article explaining the usage of RestTemplate for client side access.

Enough of theory, now let us proceed with the sample application.

1. Download the application we created in previous article from [here](#). Take a look at the [article](#) and create mysql table accordingly.
2. Import the downloaded project in eclipse and alter connection string configuration in 'config.properties' file to match your mysql configuration.
3. The idea is to issue request to the web service methods present in 'SpringServiceController' class and display the response in views (jsp's). In order to do this, we need to add a viewresolver to spring configuration file.

Add 'InternalResourceViewResolver' to 'rest-servlet.xml' file present in Webcontent/WEB-INF folder,

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p" xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
<context:component-scan
    base-package=
        "com.programmingfree.springservice.controller" />

<mvc:annotation-driven />
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" /> <property name="prefix" value="/WEB-INF/jsp/"
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

Configuration code to be added to existing code is highlighted above. The view-resolver will look for jsp files as the 'suffix' property contains '.jsp' in it and in the path 'WEB-INF/jsp/' as the prefix property is set as such.

2. Let us add a controller class which uses RestTemplate to query the web service we had already created. Create a new class called 'ListUsersController.java' under 'com.programmingfree.springservice.controller' and copy the below code in it.

```
01 package com.programmingfree.springservice.controller;
02
03 import java.util.LinkedHashMap;
04 import java.util.List;
05
06 import org.springframework.stereotype.Controller;
07 import org.springframework.web.bind.annotation.PathVariable;
08 import org.springframework.web.bind.annotation.RequestMapping;
09 import org.springframework.web.client.RestTemplate;
10 import org.springframework.web.servlet.ModelAndView;
11
12 import com.programmingfree.springservice.domain.User;
13
14 @Controller
15 public class ListUsersController {
16
17     @RequestMapping("/listUsers")
18     public ModelAndView listUsers() {
19         RestTemplate restTemplate = new RestTemplate();
20         String url="http://localhost:8080/SpringServiceJsonSample/service/user/";    List<LinkedHashMap>
21         users=restTemplate.getForObject(url, List.class);
22         return new ModelAndView("listUsers", "users", users);
23     }
24
25     @RequestMapping("/dispUser/{userid}")
26     public ModelAndView dispUser(@PathVariable("userid") int userid) {
27         RestTemplate restTemplate = new RestTemplate();
28         String url="http://localhost:8080/SpringServiceJsonSample/service/user/{userid}";
29         User user=restTemplate.getForObject(url, User.class,userid);
30         return new ModelAndView("dispUser", "user", user);
31     }
32 }
```

In the previous example, the url '<http://localhost:8080/SpringServiceJsonSample/service/user/>' returned list of all users in json format. This controller contains two methods, one to list all users in a table and the other to list a single user information. Each method uses different jsp pages - 'listUsers.jsp' & 'dispUser.jsp' to display data retrieved from web service. RestTemplate automatically converts the json response to Java model class, in this case 'User' class for us.

3. Create two jsp files under "WEB-INF/jsp/" folder and copy the code given below. Make sure to use the same name as given below for the jsp files as we have used the names already in the controller class.

listUsers.jsp

```
01 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02     pageEncoding="ISO-8859-1"%>
```

```

03 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
04 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
05 <html>
06 <head>
07 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
08 <title>List of Users</title>
09 </head>
10 <body>
11 <table border="1" align="center" style="width:50%">
12 <thead>
13 <tr>
14 <th>User Id</th>
15 <th>First Name</th>
16 <th>Last Name</th>
17 <th>Email</th>
18 </tr>
19 </thead>
20 <tbody>
21 <c:forEach var="users" items="${users}" >
22 <tr>
23 <td>${users.userid}</td>
24 <td>${users.firstName}</td>
25 <td>${users.lastName}</td>
26 <td>${users.email}</td>
27 </tr>
28 </c:forEach>
29 </tbody>
30 </table>
31 </body>
32 </html>

```

I have used JSTL(Java Standard Tag Libraries) to loop through the list of user objects here.

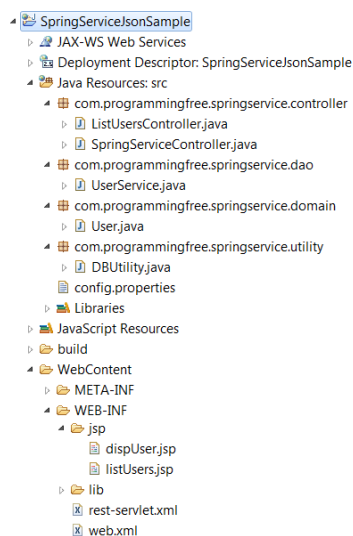
dispUser.jsp

```

01 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02 pageEncoding="ISO-8859-1"%>
03 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
04 <html>
05 <head>
06 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
07 <title>User Details</title>
08 </head>
09 <body>
10 <table border="1" align="center" style="width:50%">
11 <thead>
12 <tr>
13 <th>User Id</th>
14 <th>First Name</th>
15 <th>Last Name</th>
16 <th>Email</th>
17 </tr>
18 </thead>
19 <tbody>
20 <tr>
21 <td>${user.userid}</td>
22 <td>${user.firstName}</td>
23 <td>${user.lastName}</td>
24 <td>${user.email}</td>
25 </tr>
26 </tbody>
27 </table>
28 </body>
29 </html>

```

This is how the project structure will look after creating all necessary classes and files,




4. Finally, add the below servlet mapping to web.xml file. This is to avoid 'no mapping found error' for jsp pages, caused by the '/' mapping we have given for dispatche

servlet.

```
1 <servlet>
2   <servlet-name>jsp</servlet-name>
3   <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
4 </servlet>
5 <servlet-mapping>
6   <servlet-name>jsp</servlet-name>
7   <url-pattern>/WEB-INF/jsp/*</url-pattern>
8 </servlet-mapping>
```

That is all. Now run the application in Tomcat server and hit this url, '<http://localhost:8080/SpringServiceJsonSample/listUsers>'. This request will be routed to ListUsersController and the response will be displayed utilizing the corresponding view (listUser.jsp) such as this,



User Id	First Name	Last Name	Email
2	Priya	Balachandran	abc@abc.com
3	Anita	Lidiya	sfjdf@sdvk.com
4	Jude	Rao	sfjfsdf@sdfdgf.com
5	John	Edwin	sdjf@sdff.com
6	Mariya	Joseph	abc@abc.com
7	Sarah	Abraham	abc@def.com
8	Carolin	Stoner	asds@aedr.com
9	Angeline	Chelladurai	asds@aedr.com
10	Meena	Muthu	asds@aedr.com
11	Mariya	Magdaline	asds@aedr.com
12	Isaac	Newton	asds@aedr.com
13	Charles	Babbage	asds@aedr.com
14	Indhira	Privadarshini	asds@aedr.com

To display specific user information, hit '<http://localhost:8080/SpringServiceJsonSample/disUser/2>',



User Id	First Name	Last Name	Email
2	Priya	Balachandran	abc@abc.com

Keep yourself subscribed via email or social networking sites to get notified whenever a new article is published. Thanks for reading!

Subscribe to GET LATEST ARTICLES!

Enter Your Email Address...

Subscribe

Posted by **Priya Dai**

POST A COMMENT