HowToDoInJava

Spring REST Hello World XML Example

February 20, 2015 by Lokesh Gupta

In this tutorial, I am writing hello world example for RESTful APIs using Spring REST features. In this example, I will be creating two APIs which will return XML representation of resources.

Download Sourcecode

Maven Dependencies

Let’s start with runtime dependencies which you will need to write these RESTFul APIs. In fact, all you need is Spring MVC support only.

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/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.howtodoinjava.demo</groupId>
  <artifactId>springrestexample</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>springrestexample Maven Webapp</name>
  <url>http://maven.apache.org</url>
```

```
    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
      </dependency>

      <!-- Spring MVC support -->

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-webmvc</artifactId>
          <version>4.1.4.RELEASE</version>
      </dependency>

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-web</artifactId>
          <version>4.1.4.RELEASE</version>
      </dependency>

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-core</artifactId>
          <version>4.1.4.RELEASE</version>
      </dependency>

    </dependencies>
    <build>
      <finalName>springrestexample</finalName>
    </build>
  </project>
```

Note: If you please planning to include JSON support as well then all you need to do is include Jackson libraries into classpath, and same APIs will work for jackson as well.

```
<!-- Jackson JSON Processor -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.4.1</version>
</dependency>
```

Spring MVC Configuration

For creating APIs, you will need to configure your applications same as you do in Spring MVC.

web.xml

```
<!DOCTYPE web-app PUBLIC
```

```
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

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

  <servlet>
        <servlet-name>spring</servlet-name>
            <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
            </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

spring-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context/
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <context:component-scan base-package="com.howtodoinjava.demo" />
    <mvc:annotation-driven />

</beans>
```

JAXB Annotated Model Objects

You will need to annotate your model objects with jaxb annotations so that JAXB can marshal the java object into XML representation to be sent to client for that API.

EmployeeVO.java

```
package com.howtodoinjava.demo.model;

import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement (name = "employee")
@XmlAccessorType(XmlAccessType.NONE)
public class EmployeeVO implements Serializable
{
    private static final long serialVersionUID = 1L;

    @XmlAttribute
    private Integer id;

    @XmlElement
    private String firstName;

    @XmlElement
    private String lastName;

    @XmlElement
    private String email;

    public EmployeeVO(Integer id, String firstName, String lastName, String email) {
        super();
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public EmployeeVO(){

    }

    //Setters and Getters

    @Override
    public String toString() {
        return "EmployeeVO [id=" + id + ", firstName=" + firstName
                + ", lastName=" + lastName + ", email=" + email + "]";
    }
}
```

EmployeeListVO.java

```
package com.howtodoinjava.demo.model;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement (name="employees")
public class EmployeeListVO implements Serializable
{
    private static final long serialVersionUID = 1L;
    private List<EmployeeVO> employees = new ArrayList<EmployeeVO>();

    public List<EmployeeVO> getEmployees() {
        return employees;
    }

    public void setEmployees(List<EmployeeVO> employees) {
        this.employees = employees;
    }
}
```

REST Controller

This is main class which will decide that which API will behave which way.

EmployeeRESTController.java

```
package com.howtodoinjava.demo.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import com.howtodoinjava.demo.model.EmployeeListVO;
import com.howtodoinjava.demo.model.EmployeeVO;

@RestController
public class EmployeeRESTController
{
    @RequestMapping(value = "/employees")
    public EmployeeListVO getAllEmployees()
    {
        EmployeeListVO employees = new EmployeeListVO();

        EmployeeVO empOne = new EmployeeVO(1,"Lokesh","Gupta","howtodoinjava@gmail.com");
        EmployeeVO empTwo = new EmployeeVO(2,"Amit","Singhal","asinghal@yahoo.com");
        EmployeeVO empThree = new EmployeeVO(3,"Kirti","Mishra","kmishra@gmail.com");


        employees.getEmployees().add(empOne);
        employees.getEmployees().add(empTwo);
        employees.getEmployees().add(empThree);

        return employees;
    }

    @RequestMapping(value = "/employees/{id}")
    public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") int id)
    {
        if (id <= 3) {
            EmployeeVO employee = new
EmployeeVO(1,"Lokesh","Gupta","howtodoinjava@gmail.com");
            return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);
        }
        return new ResponseEntity(HttpStatus.NOT_FOUND);
    }
}
```

Let’s note down few important things.

1) We have used @RestController annotation. Till Spring 3, we would have been using @Controller annotation and in that case it was important to use @ResponseBody annotation as well. e.g.

```
@Controller
public class EmployeeRESTController
{
    @RequestMapping(value = "/employees")
    public @ResponseBody EmployeeListVO getAllEmployees()
    {
        //API code
    }
}
```

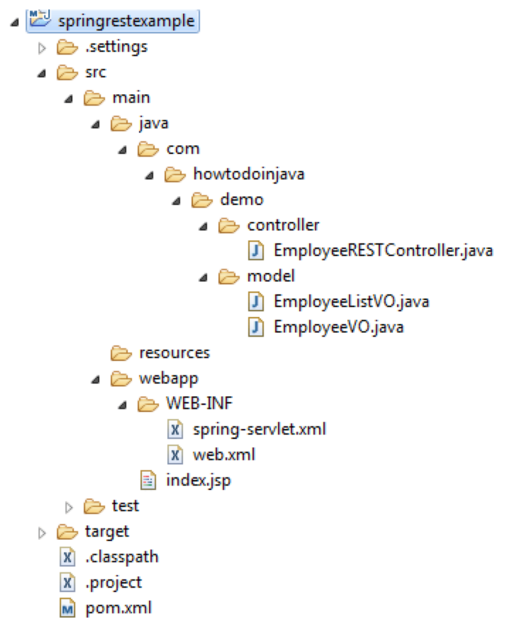
Spring 4 introduced `@RestController` which is combination of `@Controller + @ResponseBody`. So when using `@RestController`, you do not need to use `@ResponseBody`. It’s optional.

2) Here we are relying on the Spring MVC `HttpMessageConverter`s to convert an object to the xml representation requested by the user. `@ResponseBody` annotation (included through `@RestController`) tells Spring MVC that the result of the method should be used as the body of the response. As we want XML this marshaling is done by the `Jaxb2RootElementHttpMessageConverter` provided by Spring which is automatically registered in spring context if JAXB libraries are found in classpath. As I am using JRE 7 to run this application and it has JAXB inbuilt, so I do not need to add external dependency through maven.

3) Due to the `@ResponseBody` annotation, we don’t need the view name anymore but can simply return the employees object.

4) Instead of returning the java objects directly, you can wrap them inside `ResponseEntity`. The `ResponseEntity` is a class in Spring MVC that acts as a wrapper for an object to be used as the body of the result together with a HTTP status code. This provides greater control over what you are returning to client in various use cases. e.g. returning a 404 error if no employee is found for given employee id.

Project Structure



```
springrestexample
  .settings
  src
    main
      java
        com
          howtodoinjava
            demo
              controller
                EmployeeRESTController.java
              model
                EmployeeListVO.java
                EmployeeVO.java
      resources
      webapp
        WEB-INF
          spring-servlet.xml
          web.xml
        index.jsp
    test
  target
  .classpath
  .project
  pom.xml
```

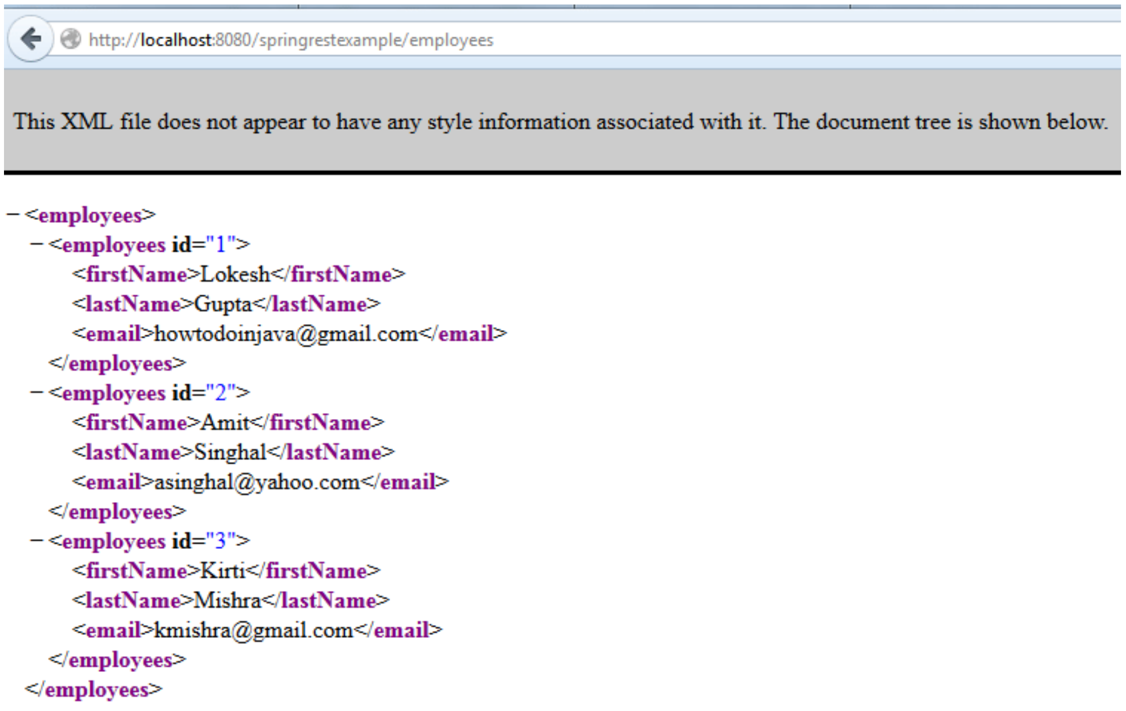
Spring REST XML Example – Project Structure

Test the APIs

Let’s test above REST APIs.

1) Hit URL : http://localhost:8080/springrestexample/employees

You can pass accept header “ `application/xml` ” as well.



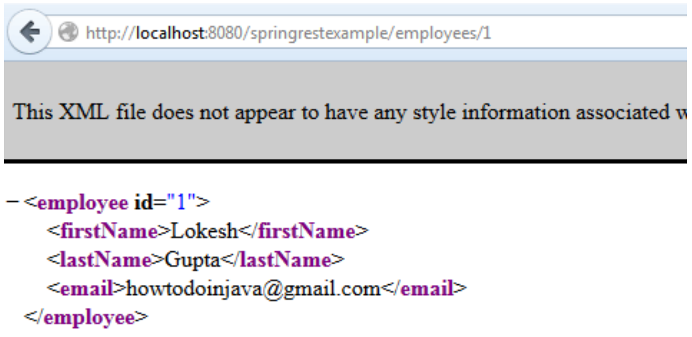
http://localhost:8080/springrestexample/employees

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <employees>
  - <employees id="1">
      <firstName>Lokesh</firstName>
      <lastName>Gupta</lastName>
      <email>howtodoinjava@gmail.com</email>
    </employees>
  - <employees id="2">
      <firstName>Amit</firstName>
      <lastName>Singhal</lastName>
      <email>asinghal@yahoo.com</email>
    </employees>
  - <employees id="3">
      <firstName>Kirti</firstName>
      <lastName>Mishra</lastName>
      <email>kmishra@gmail.com</email>
    </employees>
  </employees>
```

Spring REST XML Example – REST API for get all employees

2) Hit URL : http://localhost:8080/springrestexample/employees/1



http://localhost:8080/springrestexample/employees/1

This XML file does not appear to have any style information associated w

```
- <employee id="1">
    <firstName>Lokesh</firstName>
    <lastName>Gupta</lastName>
    <email>howtodoinjava@gmail.com</email>
  </employee>
```

Spring REST XML Example – REST API for get employee by id

3) Hit URL : http://localhost:8080/springrestexample/employees/123

```
Status Code: 404 Not Found
Content-Length: 0
Date: Fri, 18 Feb 2015 07:01:17 GMT
Server: Apache-Coyote/1.1
```

That's all for this quick hello world application for RESTFul APIs using spring mvc.

Download Sourcecode

Happy Learning !!

About Lokesh Gupta

Founded HowToDoInJava.com in late 2012. I love computers, programming and solving problems everyday. A family guy with fun loving nature. You can find me on Facebook, Twitter and Google Plus.

Feedback, Discussion and Comments

Prageeth

June 5, 2015

Short and clean article. Thanks a lot.

Reply

swati

March 16, 2015

When i hit this URL i have configured employees.jsp page and it gets redirected to this page http://localhost:8080/JAXBDemo_List_Set/employees

but still i am not able to see xml file on hitting in browser.

Reply

Binh Thanh Nguyen
March 6, 2015

Thanks, nice post

Reply

Ask Questions & Share Feedback

Your email address will not be published. Required fields are marked *

Comment

*Want to Post Code Snippets or XML content? Please use [java] … [/java] tags otherwise code may not appear partially or even fully. e.g.

```
[java]
public static void main (String[] args) {
...
}
[/java]
```

Name *

Email *

Website

Help me fight spammers. Solve this simple math. *

8 − 3 =

POST COMMENT

Search Tutorials

Type and Press ENTER...

Recommended

- 10 Life Lessons
- How to Start New Blog
- Secure Hash Algorithms
- Regular Expressions
- How Web Servers work?
- How Java I/O Works Internally?
- Best Way to Learn Java
- Java Best Practices
- Java Interview Questions
- Microservices Tutorial
- REST API Tutorial

Spring REST Tutorials

Developer Tools

JSON Formatter and Minifier
XML Formatter and Minifier
CSS Formatter and Minifier
HTML Formatter and Minifier

Meta Links

Advertise
Contact Us
Privacy policy
About Me

References

Java 8 API

Spring Framework Reference

RESTEasy Reference

Hibernate User Guide

Junit Wiki

Maven FAQs