DZone

Spring Framework: @RestController vs. @Controller

See how you can utilize Spring's annotations to create RESTful web services.

by Srivatsan Sundararajan · Nov. 08, 16 · Java Zone · Tutorial

Spring MVC Framework and REST

Spring's annotation-based MVC framework simplifies the process of creating RESTful web services. The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created. While the traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML. For a detailed description of creating RESTful web services using the Spring framework, click here.

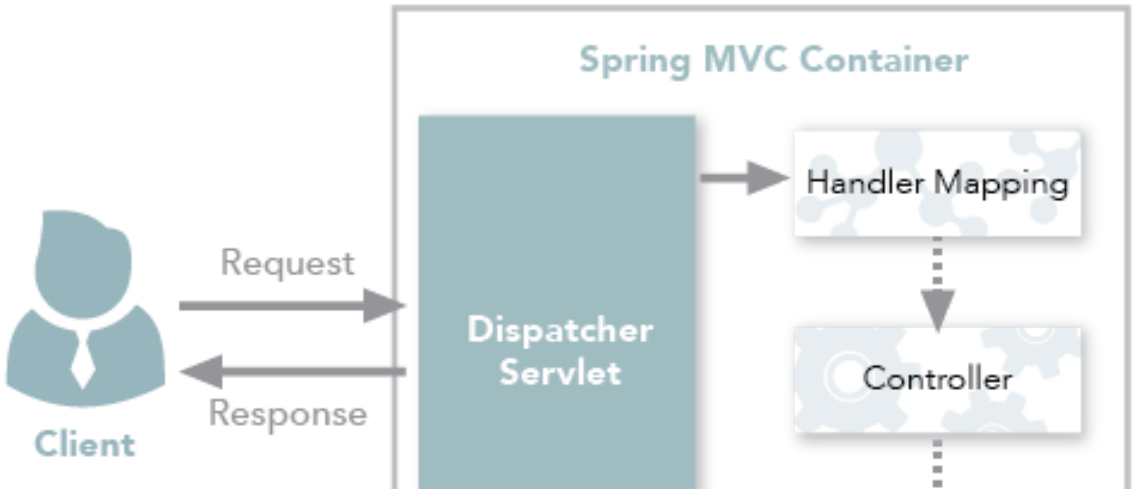


Figure 1: Spring MVC traditional workflow

Spring MVC REST Workflow

The following steps describe a typical Spring MVC REST workflow:

1. The client sends a request to a web service in URI form.
2. The request is intercepted by the DispatcherServlet which looks for Handler Mappings and its type.
    - The Handler Mappings section defined in the application context file tells DispatcherServlet which strategy to use to find controllers based on the incoming request.
    - Spring MVC supports three different types of mapping request URIs to controllers: annotation, name conventions, and explicit mappings.
3. Requests are processed by the Controller and the response is returned to the DispatcherServlet which then dispatches to the view.

In Figure 1, notice that in the traditional workflow the ModelAndView object is forwarded from the controller to the client. Spring lets you return data directly from the controller, without looking for a view, using the @ResponseBody annotation on a method. Beginning with Version 4.0, this process is simplified even further with the introduction of the @RestController annotation. Each approach is explained below.

Using the @ResponseBody Annotation

When you use the @ResponseBody annotation on a method, Spring converts the return value and writes it to the http response automatically. Each method in the Controller class must be annotated with @ResponseBody.

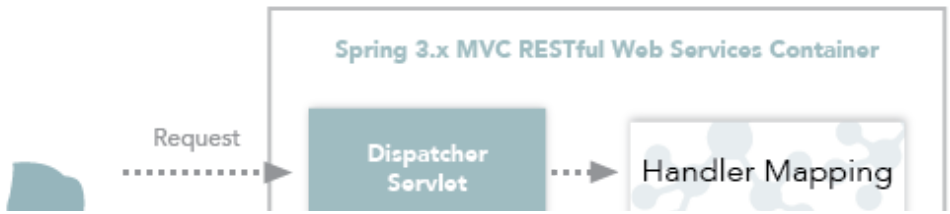


Figure 2: Spring 3.x MVC RESTful web services workflow

Behind the Scenes

Spring has a list of HttpMessageConverters registered in the background. The responsibility of the HTTPMessageConverter is to convert the request body to a specific class and back to the response body again, depending on a predefined mime type. Every time an issued request hits @ResponseBody, Spring loops through all registered HTTPMessageConverters seeking the first that fits the given mime type and class, and then uses it for the actual conversion.

Code Example

Let's walk through @ResponseBody with a simple example.

Project Creation and Setup

1. Create a Dynamic Web Project with Maven support in your Eclipse or MyEclipse IDE.
2. Configure Spring support for the project.• If you are using Eclipse IDE, you need to download all Spring dependencies and configure your pom.xml to contain those dependencies.• In MyEclipse, you only need to install the Spring facet and the rest of the configuration happens automatically.
3. Create the following Java class named Employee. This class is our POJO.

```
1  package com.example.spring.model;
2
3  import javax.xml.bind.annotation.XmlRootElement;
4
5  @XmlRootElement(name = "Employee")
6  public class Employee {
7
8      String name;
       String email;
```

```
    String email;

    public String getName() {
       return name;
    }

    public void setName(String name) {
      this.name = name;
    }

    public String getEmail() {
       return email;
    }

    public void setEmail(String email) {
      this.email = email;
    }

    public Employee() {
    }
}
```

Then, create the following @Controller class:

```
package com.example.spring.rest;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
import com.example.spring.model.Employee;

@Controller
@RequestMapping("employees")
public class EmployeeController {
    Employee employee = new Employee();

    @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces = "application/json")
    public @ResponseBody Employee getEmployeeInJSON(@PathVariable String name) {

       employee.setName(name);
       employee.setEmail("employee1@genuitec.com");

    return employee;
    }

    @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces = "application/xml")
    public @ResponseBody Employee getEmployeeInXML(@PathVariable String name) {

       employee.setName(name);
     employee.setEmail("employee1@genuitec.com");

       return employee;
    }
}
```

Notice the @ResponseBody added to each of the @RequestMapping methods in the return value. After that, it's a two-step process:

1. Add the `<context:component-scan>` and `<mvc:annotation-driven />` tags to the Spring configuration file.
    - `<context:component-scan>` activates the annotations and scans the packages to find and register beans within the application context.
    - `<mvc:annotation-driven/>` adds support for reading and writing JSON/XML if the Jackson/JAXB libraries are on the classpath.

- For JSON format, include the jackson-databind jar and for XML include the jaxb-api-osgi jar to the project classpath.

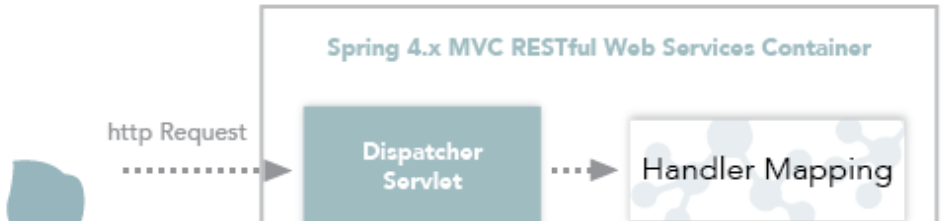
2. Deploy and run the application on any server (e.g., Tomcat). If you are using MyEclipse, you can run the project on the embedded Tomcat server.

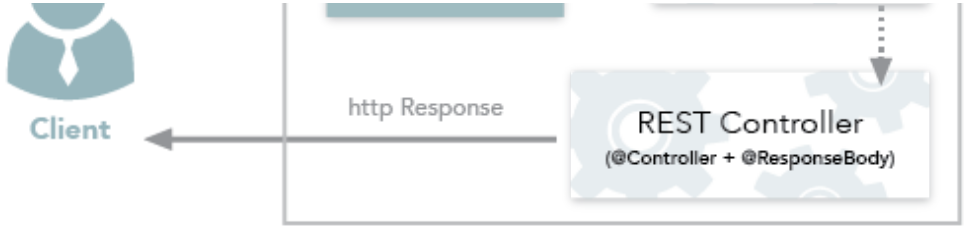
JSON—Use the URL: `http://localhost:8080/SpringRestControllerExample/rest/employees/Bob` and the following output displays:

XML — Use the
URL: `http://localhost:8080/SpringRestControllerExample/rest/employees/Bob.xml` and the following output displays:

Using the @RestController Annotation

Spring 4.0 introduced @RestController, a specialized version of the controller which is a convenience annotation that does nothing more than add the @Controller and @ResponseBody annotations. By annotating the controller class with @RestController annotation, you no longer need to add @ResponseBody to all the request mapping methods. The @ResponseBody annotation is active by default. Click here to learn more.





To use @RestController in our example, all we need to do is modify the @Controller to @RestController and remove the @ResponseBody from each method. The resultant class should look like the following:

```
package com.example.spring.rest;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.example.spring.model.Employee;

@RestController
@RequestMapping("employees")
public class EmployeeController {

    Employee employee = new Employee();

    @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces = "application/json")
    public Employee getEmployeeInJSON(@PathVariable String name) {

       employee.setName(name);
       employee.setEmail("employee1@genuitec.com");

       return employee;

```

```
    }

    @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces = "application/xml")
    public Employee getEmployeeInXML(@PathVariable String name) {

       employee.setName(name);
       employee.setEmail("employee1@genuitec.com");

    return employee;
    }
}
```

Note that we no longer need to add the @ResponseBody to the request mapping methods. After making the changes, running the application on the server again results in same output as before.

Conclusion

As you can see, using @RestController is quite simple and is the preferred method for creating MVC RESTful web services starting from Spring v4.0. I would like to extend a big thank you to my co-author, Swapna Sagi, for all of her help in bringing you this information!

Like This Article? Read More From DZone

Simple Attribute-Based Access Control With Spring Security

Applying HATEOAS to a REST API with Spring Boot

Using JAX-RS With Spring Boot Instead of MVC

Free DZone Refcard
Getting Started With Vaadin 10

Topics: REST API , CONTROLLER , JAVA , SPRING MVC

ava Partner Resources

edictive Analytics + Big Data Quality: A Love Story
elissa Data

croservices for Java Developers: A Hands-On Introduction to Frameworks & Containers
d Hat Developer Program

veloping Reactive Microservices: Enterprise Implementation in Java
ghtbend

ep insight into your code with IntelliJ IDEA.
tBrains