**Spring MVC vs Spring Rest**

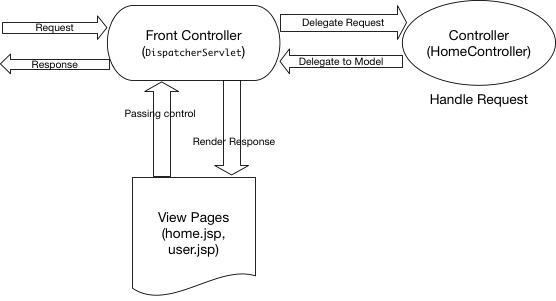
Spring’s annotation based MVC framework simplifies the process of creating RESTful web services.

The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created.

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.

**Traditional Spring MVC**

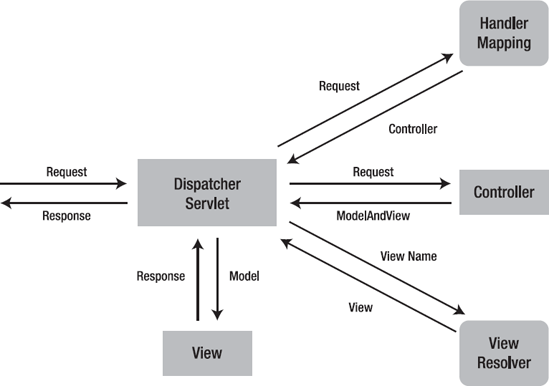
Spring MVC is based on Model-View-Controller architecture. Below image shows Spring MVC architecture at a high level.



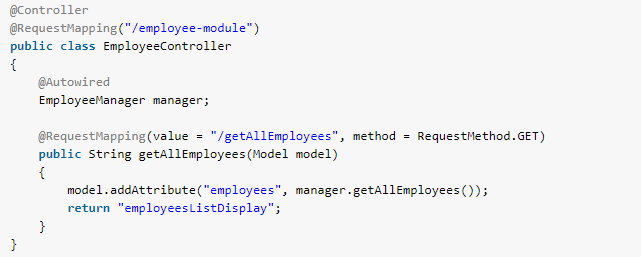
DispatcherServlet

A Spring MVC controller—often referred to as a [Dispatcher Servlet](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html) implements [front controller](https://en.wikipedia.org/wiki/Front_Controller_pattern) design pattern and every web request must go through it so that it can manage the entire request life cycle.

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring’s web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

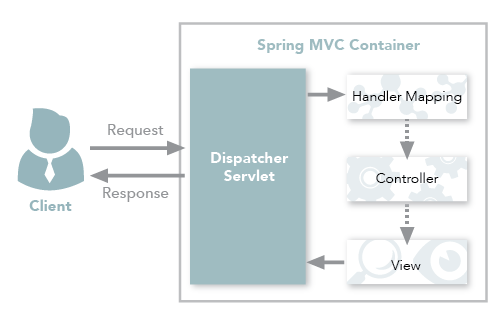


Example:



**Spring Restful Webservice**

Spring’s annotation based MVC framework simplifies the process of creating RESTful web services.



**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.

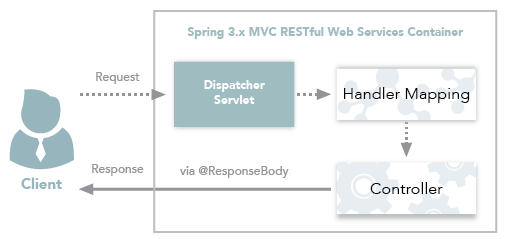
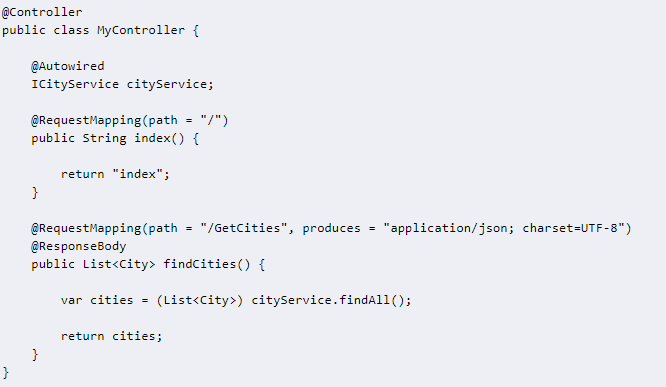


Figure 2: Spring 3.x MVC RESTful web services workflow

### 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.

**Example: With Controller and ResponseBody Annotation**



For the GetCities path, the findCities() method is called. The produces parameter indicates that the method returns JSON data; Spring RequestResponseBodyMethodProcessor handles return values from methods annotated with @ResponseBody by writing to the body of the response with an HttpMessageConverter. The message converter in our case is MappingJackson2HttpMessageConverter, which reads and writes JSON using Jackson's ObjectMapper. (Jackson is a popular Java JSON library.)

## Using the RestController Annotation

Spring 4.0 introduced @RestController, a specialized version of the controller which is a convenience annotation that is itself annotated with [@Controller](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Controller.html) and [@ResponseBody](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/ResponseBody.html).

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.  
  
Figure 5: Spring 4.x MVC RESTful Web Services Workflow

[@Target](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Target.html?is-external=true)([value](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Target.html?is-external=true#value--)=[TYPE](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/ElementType.html?is-external=true#TYPE))

[@Retention](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Retention.html?is-external=true)([value](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Retention.html?is-external=true#value--)=[RUNTIME](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/RetentionPolicy.html?is-external=true#RUNTIME))

[@Documented](https://docs.oracle.com/javase/8/docs/api/java/lang/annotation/Documented.html?is-external=true)

[@Controller](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Controller.html)

[@ResponseBody](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/ResponseBody.html)

public @interface **RestController**

**Example: With Rest Controller**

@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;

}

}

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.

**ResponseBody vs ResponseEntity**

# [**When use ResponseEntity<T> and @RestController for Spring RESTful applications**](https://stackoverflow.com/questions/26549379/when-use-responseentityt-and-restcontroller-for-spring-restful-applications)

* @RestController (I know it is the same than @Controller + @ResponseBody)
* @ResponseStatus
* @ResponseEntity