**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.

### SPRING-MVCSpring 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.

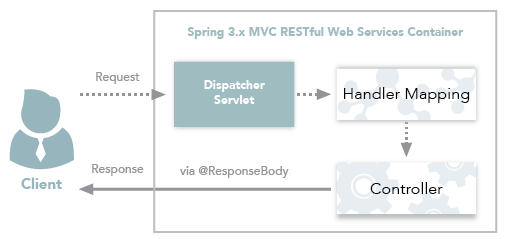


Figure 2: Spring 3.x MVC RESTful web services workflow

### 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.