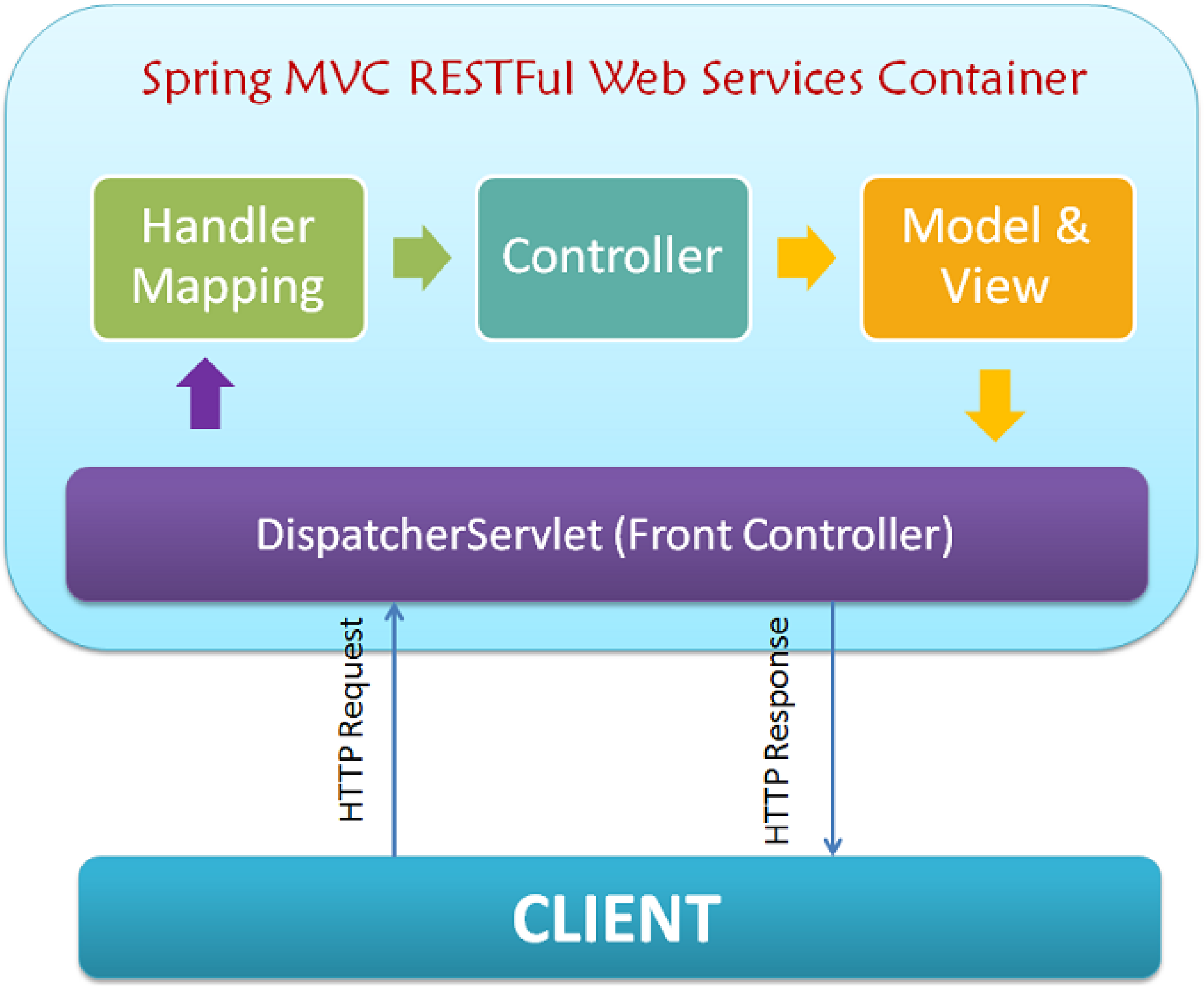
PROGRAMMINGFREE

Spring MVC 4.0 RESTFul Web Services Simple Example

http://www.programming-free.com/2014/01/spring-mvc-40-restful-web-services.html


Spring MVC RESTFul Web Services Container
Handler Mapping
Controller
Model & View
DispatcherServlet (Front Controller)
HTTP Request
HTTP Response
CLIENT

REST(Representational State Transfer) is an architectural style with which Web Services can be designed that serves resources based on the request from client. A Web Service is a unit of managed code, that can be invoked using HTTP requests. Let me put it simple for those who are new to Web Service.You develop the core functionality of your application, deploy it in a server and expose to the network. Once it is exposed, it can be accessed using URI's through HTTP requests from a variety of client applications. Instead of repeating the same functionality in multiple client (web, desktop and mobile) applications, you write it once and access it in all the applications.

Articles on Spring MVC 4

- Spring Security for Spring MVC 4 Simple Example
- 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

Spring MVC & REST

Spring MVC supports REST from version 3.0. It is easier to build restful web services with spring with it's annotation based MVC Framework. In this post, I am going to explain how to build a simple RESTFul web service using Spring MVC 4.0, that would return plain text.

Now, take a look at the architecture diagram above to understand how spring mvc restful web service handles requests from client. The request process flow is as follows,

1. Client application issues request to web service in the form of URI's.

    Example: http://example.com/service/greetings/Priya

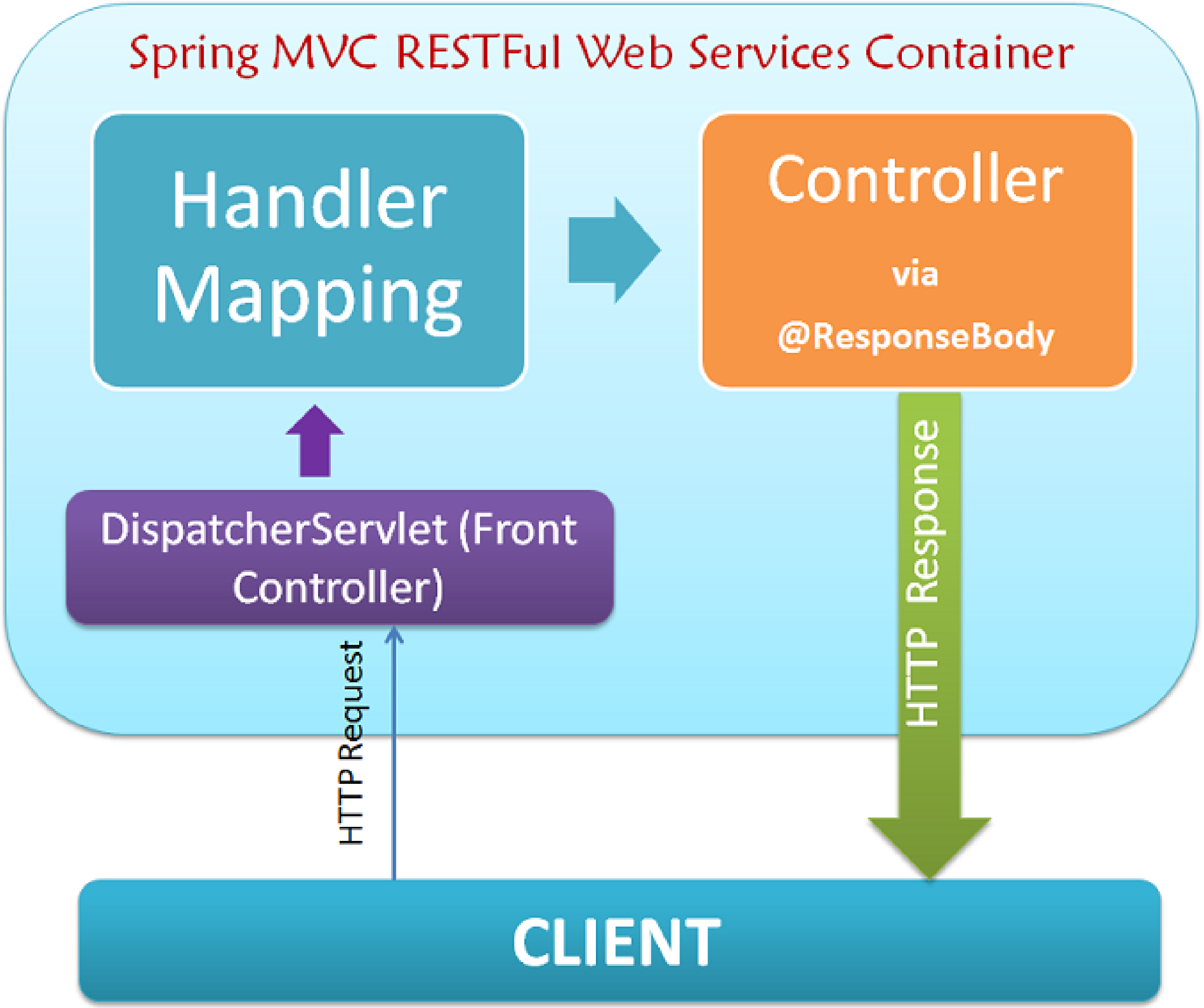
2. All HTTP Requests are intercepted by DispatcherServlet (Front End Controller). - This is defined in the web.xml file.

3. DispatcherServlet looks for Handler Mappings. Spring MVC supports three different ways of mapping request URI's to controllers : annotation, name conventions and explicit mappings. Handler Mappings section defined in the application context file, tells DispatcherServlet which strategy to use to find controllers based on the incoming request. More information on Handler Mappings can be found here.

4. Requests are now processed by the Controller and response is returned to DispatcherServlet. DispatcherServlet looks for View Resolver section in the application context file. For RESTFul web services, where your web controller returns ModelAndView object or view names, 'ContentNegotiatingResolver' is used to find the correct data representation format.

5. There is also an alternate option to the above step. Instead of forwarding ModelAndView object from Controller, you can directly return data from Controller using @ResponseBody annotation as depicted below. You can find more information on using ContentNegotiatingResolver or ResponseBody annotation to return

response here.

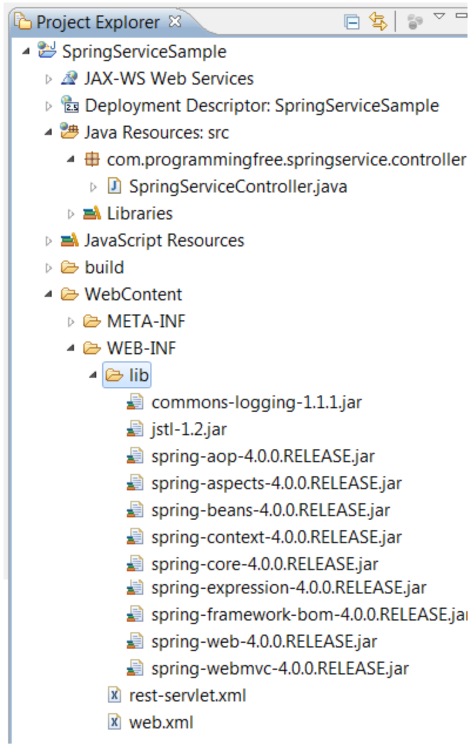

Spring MVC RESTFul Web Services Container
Handler Mapping
Controller
via
@ResponseBody
DispatcherServlet (Front Controller)
HTTP Request
HTTP Response
CLIENT

DOWNLOAD

Its time to get our hands dirty with some real coding. Follow the steps below one by one to get your first Spring MVC REST web service up and running.

1. Download Spring MVC 4.0 jar files from this maven repository here.

2. Create Dynamic Web Project and add the libraries you downloaded to WEB-INF/lib folder.



```
SpringServiceSample
  JAX-WS Web Services
  Deployment Descriptor: SpringServiceSample
  Java Resources: src
    com.programmingfree.springservice.controller
      SpringServiceController.java
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
      lib
        commons-logging-1.1.1.jar
        jstl-1.2.jar
        spring-aop-4.0.0.RELEASE.jar
        spring-aspects-4.0.0.RELEASE.jar
        spring-beans-4.0.0.RELEASE.jar
        spring-context-4.0.0.RELEASE.jar
        spring-core-4.0.0.RELEASE.jar
        spring-expression-4.0.0.RELEASE.jar
        spring-framework-bom-4.0.0.RELEASE.ja
        spring-web-4.0.0.RELEASE.jar
        spring-webmvc-4.0.0.RELEASE.jar
      rest-servlet.xml
      web.xml
```

3. Open /WEB-INF/web.xml file and add below configuration before </webapp> tag.

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

```xml
</servlet>

<servlet-mapping>
 <servlet-name>rest</servlet-name>
 <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Note that in the above code,we have named Spring Dispatcher servlet class as "rest" and the url pattern is given as "/*" which means any uri with the root of this web application will call DispatcherServlet. So what's next? DispatcherServlet will look for configuration files following this naming convention - [servlet-name]-servlet.xml. In this example, I have named dispatcher servlet class as "rest" and hence it will look for file named 'rest-servlet.xml'.

4. Now create an xml file under WEB-INF folder and name it 'rest-servlet.xml'. Copy the below in it.

```xml
<?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 />
  </beans>
<span id="ezoic-pub-ad-placeholder-110" class="ezoic-adpicker-ad" data-ezadpicker="110" data-ez-position-type="link_mid"></
```

With ComponentScan tag, Spring auto scans all elements in the provided base package and all its child packages for Controller servlet. Also, we have used <mvc:annotation-driven> tag instead of ViewResolver, with which we can directly send response data from the controller.

5. Let us create a controller servlet in the package mentioned in the component-scan tag. Copy and paste the below code in it.

```
package com.programmingfree.springservice.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.RestController;


@RestController
@RequestMapping("/service/greeting")
public class SpringServiceController {
 @RequestMapping(value = "/{name}", method = RequestMethod.GET)
 public String getGreeting(@PathVariable String name) {
  String result="Hello "+name;
  return result;
 }
}
```

'@RequestMapping' annotation is used for defining incoming request urls for class and method levels. In this case all uri's in the format `<root-url>/service/greeting/` will be routed to this Controller class. With *@RequestMapping* annotation, we can only define generic uri mappings. For dynamic uri mappings in case of passing variables along with the uri, *@PathVariable* is used. Here in this case, we pass a variable 'name' along with the uri such as, `<root-url>/service/greeting/Priya.` Here the last parameter (Priya) in the uri is retrieved using *@PathVariable.*

I explained that while using <mvc:annotation-config> tag instead of view resolver, we use *'@ResponseBody'* annotation to return response data directly from controller. But in the above code, I have not used *'@ResponseBody'*. This is because, in Spring MVC 4.0, they have introduced *'@RestController'* such that we need not use *'@ResponseBody'* tag in each and every method. *'@RestController'* will handle all of that at the type level.

This annotation simplifies the controller and it has '*@Controller*' and '*@ResponseBody*' annotated within itself.

Let us take a look at the same controller but with Spring MVC 3.0,

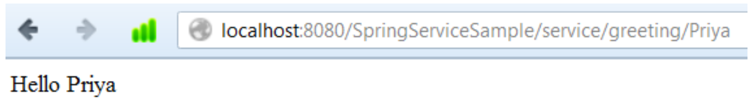
```
@Controller
@RequestMapping("/service/greeting")
public class SpringServiceController {
 @RequestMapping(value = "/{name}", method = RequestMethod.GET)
 public @ResponseBody String getGreeting(@PathVariable String name) {
  String result="Hello "+name;
  return result;
 }
}
```

Note that in Spring MVC 3.0 we had to expicitly use *@Controller* annotation to specify controller servlet and *@ResponseBody* annotation in each and every method. With the introduction of *'@RestController'* annotation in Spring MVC 4.0, we can use it in place of *@Controller* and *@ResponseBody* annotation.

That is all! Simple RESTFul Web Service returning greeting message based on the name passed in the URI, is built using Spring MVC 4.0.

So now run this application in Apache Tomcat Web Server.

Open Web Browser, and hit this url, 'http://localhost:8080/<your-application-name>/service/greeting/<anyname> . If you have downloaded this sample application, then use,
http://localhost:8080/SpringServiceSample/service/greeting/Priya . You will be seeing greeting message sent as response from server,


localhost:8080/SpringServiceSample/service/greeting/Priya

Hello Priya

DOWNLOAD

Keep yourself subscribed to programming-free.com to get these articles delivered to your inbox. Thanks for reading!

Posted by ***Priya Darshini***

POST A COMMENT