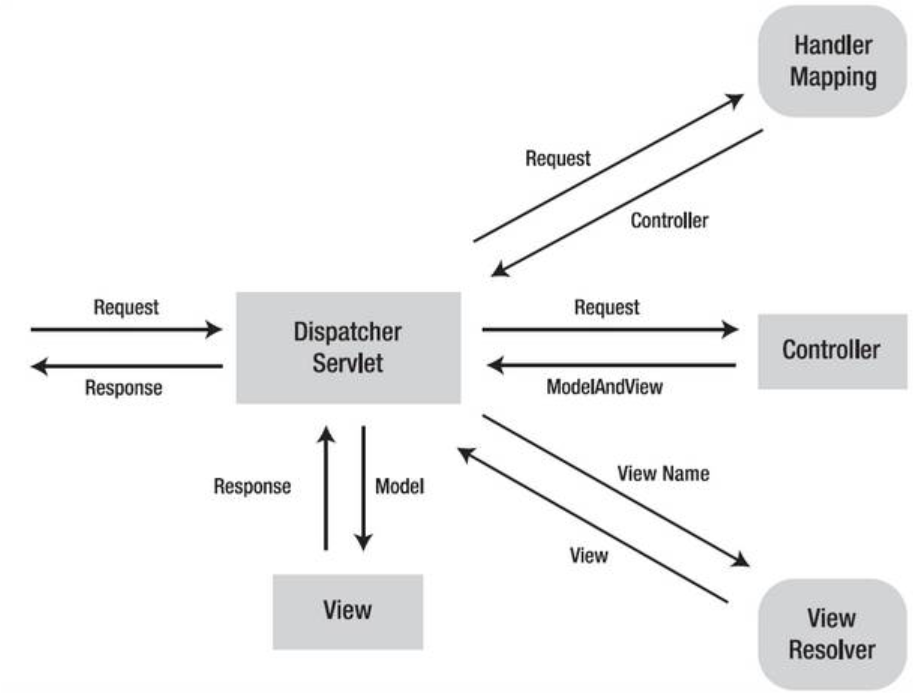
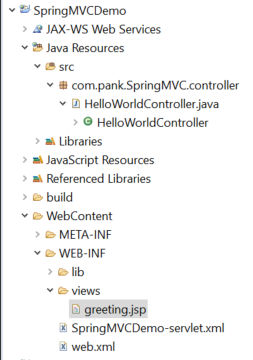
Spring MVC Flow:

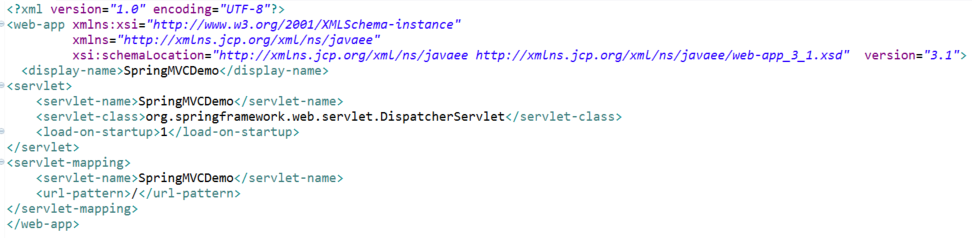


Project Structure:



web.xml is the entry point of any web program in java. Here we are telling our program that URL mapping will be handled by Dispatcher Servlet. Since we are not defining any name for the Dispatcher servlet, spring framework will look for servlet-name-servlet.xml file (in our case SpringMVCDemo-servlet-Demo.xml) inside WebContent/WEB-INF directory.

web.xml: Here we are defining the servlet. In our case it is DispatcherServlet.



<context:component-scan will activate spring annotation capability. It will tell our program where to look for annotation like @Controller, @RequestMapping etc.

InternalResourceViewResolver have the rules which help dispatcher servlet to resolve the view name. It adds prefix and suffix to the given string. Like to keyword greeting in our case, it will tell dispatcher to look for /WEB-INF/views/greeting.jsp file

SpringMVCDemo-servlet-Demo.xml:

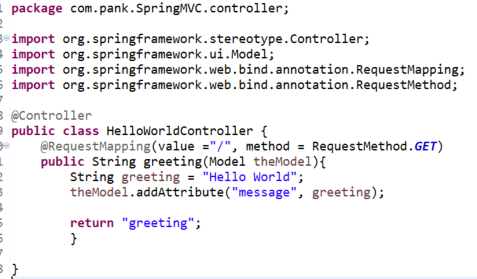


In Spring MVC, controller in the main engine, that controls the programs. It tells dispatcher which file to invoke.

@Controller indicates a particular class acts as a controller in the program

@RequestMapping helps in mapping the URL in the program.

HelloWorldController.java



Dispatcher is a java servlet.

Example:



Dispatcher will refer to many of the mapping handlers in order to identify the target controller. However if nothing is defined it will refer to BeanNameUrlHandlerMapping or DefaultAnnotationHandlerMapping.

When the target controller is identified, the DispatcherServlet sends request to it. The controller performs some work according to the request (or delegate it to the other objects), and returns back to the DispatcherServlet with the Model and the name of the View.

So, in generally DispatcherServlet capture request URI and hand over to HandlerMapping. HandlerMapping search mapping bean with method of controller, where controller returning logical name(view). Then this logical name is send to DispatcherServlet by HandlerMapping. Then DispatcherServlet tell ViewResolver to give full location of view by appending prefix and suffix, then DispatcherServlet give view to the client

**Difference between <content:annotation-config /> and <context:component-scan />**

-In case of component-scan we only need to give the package location. It will search for the @Component bean and the activate the associated annotations like @Autowired.

Whereas in case of annotation-config we need to give the bean definition explicitly like

<context:annotation-config />

<bean id="beanA" class="com.location.beans.BeanA"></bean>

<bean id="beanB" class="com.location.beans.BeanB"></bean>

It will only activate the @Autowired notation.

**Inversion Of Control:**

Outsourcing of the creation of object in spring is called IoC.

**Spring Container primary functions is:**

1. Create and manage the object(IoC)
2. Inject Object’s dependencies(DI)

**Ways to configuring Spring Container:**

1. XML configuration file
2. Java Annotations
3. Java Source Code

In Spring world, Spring container is generally known as **Application Context**

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(“applicationContext.xml”);

Coach theCoach = context.getBean(“myCoach”, Coach.class”);

**We can access the data from the front end in 2 ways.**

**Form.jsp:**

|  |
| --- |
| <!Doctype>  <html>  <head>  <title>Form</title>  </head>  <body>  <form action=*"greet"* method=*"GET"*>  Name ::: <input type = *"text"* name = *"name"* />  <input type=*"submit"* value = *"Submit"*>  </form>  </body>  </html> |

1. **HttpServletRequest method:**

|  |
| --- |
| @RequestMapping(value ="/greet")  **public** String greeting(HttpServletRequest request, Model theModel){  String name = request.getParameter("name");  String greeting = "Hello World " +name;  theModel.addAttribute("message", greeting);    **return** "greeting";  } |

1. **@RequestMapping Method:**

|  |
| --- |
| @RequestMapping(value ="/greet")  **public** String greeting(@RequestParam(value="name")String name, Model theModel){  String greeting = "Hello World " +name;  theModel.addAttribute("message", greeting);    **return** "greeting";  } |