**Spring MVC (Introduction and Sample Application)**

It all starts with the client, which sends a request to a specific URL. When that request hit the web container e.g. Tomcat it look into web.xml and find the Servlet or Filter which is mapped to that particular URL. It the delegate that Servlet or Filter to process the request. Since Spring MVC is built on top of Servlet, this is also the initial flow of request in any Spring MVC based Java web application.

Remember, Web container e.g. Tomcat is responsible for creating Servlet and Filter instances and invoking their various life-cycle methods e.g. [init()](http://javarevisited.blogspot.sg/2015/02/constructor-vs-init-method-in-servlet.html), service(), destroy().

In case of Spring MVC, we need to declare the **DispatcherServlet** from Spring MVC jar into web.xml. This [Servlet](http://javarevisited.blogspot.sg/2017/05/difference-between-servlet-and-jsp.html)listens for a URL pattern \* as shown in below web.xml, which means all request is mapped to **DispatcherServlet**.  **DispatcherServlet** class works as the front controller. It is responsible to manage the flow of the spring mvc application.

there is the web.xml configuration for Spring MVC, you can see that DispatcherServlet is mapped to all request using URL pattern \*

**web.xml**

<web-app>

<!-- The front controller of this Spring Web application, responsible

for handling all application requests -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>example</servlet-name>

<url-pattern>\*</url-pattern>

</servlet-mapping>

</web-app>

Here we have <init-param> tag in where we can define custom spring application config file location with custom name else the name we give inside <servlet-name> tag, we have to give our spring application config file name with append **“-servlet.xml”**

Let’s see web and spring config file.

**Web.xml**

controller of this Spring Web application, responsible

for handling all application requests -->

<servlet>

<servlet-name>**spring**</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!—optional>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/config/web-application-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>example</servlet-name>

<url-pattern>\*</url-pattern>

</servlet-mapping>

</web-app>

Ex.

**spring-servlet.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:p="http://www.springframework.org/schema/p"

    xmlns:context="http://www.springframework.org/schema/context"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

    <context:component-scan  base-package="com.piyush" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <property name="prefix" value="/WEB-INF/jsp/" />

        <property name="suffix" value=".jsp" />

    </bean>

</beans>

URL pattern plays very important role here, if the request matches the URL pattern of **DispatcherServlet** then it will be processed by [**Spring MVC**](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html#axzz4jWEJmi6S) otherwise not.

<url-pattern>\*</url-pattern>

In above url pattern \* means all requests will be transferred to dispatcher servlet but if we want to specify some specific pattern to transfer to Dispatcher servlet then we can define url pattern as

<url-pattern>\*.spring</url-pattern>

Means all requests ends with .spring will only be transferred to Dispatcher Servlet.

**DispatcherServlet** passes requests to a mapped controller depending on the URL requested.

But How does **DispatcherServlet** knows which request needs to be passed to which controller and which controller or handler we have to access to serve the request. It is mandatory because we can have multiple handler to handle the request in controllers ie SimpleServletHandlerAdapter,RequestMappingHandlerAdapter,HttpRequestHAndlerAdapter etc.

after handler mapping DispatcherServlet does not communicate with the SimpleServletHandlerAdapter,RequestMappingHandlerAdapter,HttpRequestHAndlerAdapter used in Controller DispatcherServlet directly communicate with HandlerAdapter which is a bridge between Controller(HttpRequestHAndlerAdapter) and DespatcherServlet

**DispatcherServlet🡪HandlerMapping**

**DispatcherServlet🡨HandlerMapping (sends which handler adapter to use)**

**DispatcherServlet🡪HandlerAdapter🡪HandlerController(Controller)**

**Let’s now talk about how we map our requests with controllers**

Well, it uses the **@RequestMapping** annotation or Spring MVC configuration file to find out mapping of request URL to different controllers. It can also use specific request processing annotations e.g. @GetMapping or @PostMapping. Controller classes are also identified using @Controller and @RestController (in the case of RESTful Web Services) annotations.   
  
For example, below class is a Controller which will process any request having URI "/appointments". It also has @GetMapping, which means that method will be invoked when a GET request is received for this URL. The method annotated with @PostMapping will be invoked if the client sends a [POST request](http://www.java67.com/2014/08/difference-between-post-and-get-request.html) to the "/appointments" URI.

@Controller

@RequestMapping("/appointments")

public class AppointmentsController {

@GetMapping

public Map get() {

return appointmentBook.getAppointmentsForToday();

}

@PostMapping

public String add(@Valid AppointmentForm appointment, BindingResult result) {

if (result.hasErrors()) {

return "appointments/new";

}

appointmentBook.addAppointment(appointment);

return "redirect:/appointments";

}

}

After processing the request, Controller returns a **logical view name** and model to [**DispatcherServlet**](http://javarevisited.blogspot.sg/2016/01/solving-javalangclassnotfoundexception-org.springframework.web.servlet.DispatcherServlet.html)and it consults view resolvers until an actual View is determined to render the output. **DispatcherServlet** then contacts the chosen view e.g. Freemarker or JSP with model data and it renders the output depending on the model data.  
  
This Rendered output is returned to the client as HTTP response. On it's way back it can pass to any configured Filter as well e.g. [Spring Security filter chain](http://javarevisited.blogspot.sg/2017/05/how-to-enable-spring-security-in-java-web-application.html#axzz4gg59c400) or Filters configured to convert the response to JSON or XML.  
  
The **DispatcherServlet** from **Spring MVC** framework is an implementation of Front Controller Pattern and it's also a Single point of entry - handle all incoming requests, but again that depends upon your URL pattern mapping and your application.  
  
It delegates requests for further processing to additional components e.g. Controllers, Views, View Resolvers, handler mappers, exception handlers etc. It can also map directly to /, but then the exception for handling static resources needs to be configured. If you look at the web.xml configuration it also pre-loaded using the [load-on-startup](http://javarevisited.blogspot.sg/2011/12/load-on-startup-servlet-webxml-example.html#axzz4jWEcwHFD) tag.

**Spring MVC life cycle:**

1. DispatcherServlet receives the request.
2. DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
3. DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.
4. HandlerAdapter calls the business logic process of Controller.
5. Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
6. DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.
7. DispatcherServlet dispatches the rendering process to returned View.
8. View renders Model data and returns the response.