The Spring web model-view-controller (MVC) framework provides a model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications.

Spring MCV basically consists of:

* The Model, that encapsulates the application data
* The View, which is responsible for rendering the model data and in general it generates HTML output that the client’s browser can interpret.
* The Controller, which processes user requests and builds appropriate model and passes it to the view for rendering.

The framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. Basically, the sequence of events corresponding to an incoming HTTP request to DispatcherServlet is performed by the following steps:

* The DispatcherServlet receives a request.
* The DispatcherServlet uses the HandlerMapping so as to call the appropriate Controller.
* The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
* The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
* The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

All the above mentioned components (HandlerMapping, Controller and ViewResolver) are parts of theWebApplicationContext which is an extension of the plain ApplicationContext with some extra features necessary for web applications.

Now, we can move on to create a simple example. We will create a Controller, a view (jsp) and we will add the necessary configuration files, and then we will use an application server to run the example.

**Create the Controller**

The Controller is where the DispatcherServlet will delegate requests. The @Controller annotation indicates that the class serves the role of a Controller. The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method. Here, it is used for both cases. The HelloWorldController.java class consists of a method,hello(ModelMap model) that will handle a GET request from the Dispatcher. The org.springframework.ui.ModelMap is used as a generic model holder. Here we set to it an attribute called name, and the value SF Hello World!.

@Controller

@RequestMapping("/helloWorld")

**public** **class** HelloWorldController {

@RequestMapping(method = RequestMethod.***GET***)

**public** String hello(ModelMap model) {

model.addAttribute("name", "SF Hello World!");

**return** "helloWorld";

}

}

## Create the View

Spring MVC supports many types of views for different presentation technologies, such as JSPs, HTML, PDF, Excel worksheets, XML etc. The view part of this MVC example is a simple jsp page, that shows the value of the attribute that was set to the Controller. It must be placed in /WEB-INF/ folder.

<html>

<body>

<h1>Spring 4.0.2 MVC HelloWorld Controller</h1>

<h3>Name : ${name}</h3>

</body>

</html>

**Configure the application**

The files that we must configure in the application are the web.xml file and the mvc-dispatcher-servlet.xml file. The web.xml file is the file that defines everything about your application that a server needs to know. It is placed in /WEB-INF/ directory of the application. The <servlet> element declares the DispatcherServlet. When the DispatcherServlet is initialized, the framework will try to load the application context from a file named [servlet-name]-servlet.xml located in/WEB-INF/ directory. So, we have created the mvc-dispatcher-servlet.xml file, that will be explained below. The <servlet-mapping> element of web.xml file specifies what URLs will be handled by the DispatcherServlet

<web-app>

<display-name>Spring MVC Basic Web Application Using XML</display-name>

<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>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

Now, let’s check on the spring-servlet.xml file. It is also placed in WebContent/WEB-INF directory. It uses the<context:component-scan> so that the Spring container will search for all annotated classes under thein.spring4buddies.application.controller package.  
The org.springframework.web.servlet.view.InternalResourceViewResolver is defined as a bean, and is used as internal resource views resolver, meaning that it will find the jsp and html files in the WebContent/WEB-INF/ folder. We can set properties such as prefix or suffix to the view name to generate the final view page URL, as shown below:

<beans>

<context:component-scan base-package=*"in.spring4buddies.application.controller"* />

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

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

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

</bean>

</beans>