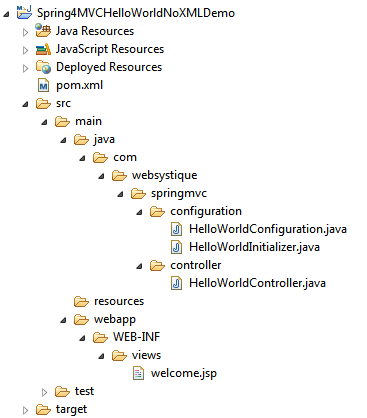
Spring MVC with Annotations Hello world Example

**Step 1: Create a project with required directory structure**

Post [Creating a maven web project with eclipse](http://websystique.com/maven/create-a-maven-web-project-with-eclipse/) contains step-by-step instruction to create a maven project with eclipse.

Following will be the final Project structure.



Now let’s add/update the content mentioned in above project structure discussing each in detail.

**Step 2: Update pom.xml with Spring and Servlet dependency**

The Spring java-based configuration we are going to discuss depends on Servlet 3.0 api, so we need to include that as a dependency in pom.xml

|  |
| --- |
| <?xml version="1.0"?>  <project xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>"      xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">        <modelVersion>4.0.0</modelVersion>      <groupId>com.websystique.springmvc</groupId>      <artifactId>Spring4MVCHelloWorldNoXMLDemo</artifactId>      <packaging>war</packaging>      <version>1.0.0</version>      <name>Spring4MVCHelloWorldNoXMLDemo</name>        <properties>          <springframework.version>4.0.6.RELEASE</springframework.version>      </properties>        <dependencies>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-webmvc</artifactId>              <version>${springframework.version}</version>          </dependency>            <dependency>              <groupId>javax.servlet</groupId>              <artifactId>javax.servlet-api</artifactId>              <version>3.1.0</version>          </dependency>          <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>javax.servlet.jsp-api</artifactId>              <version>2.3.1</version>          </dependency>          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>          </dependency>      </dependencies>        <build>          <pluginManagement>              <plugins>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-compiler-plugin</artifactId>                      <version>3.2</version>                      <configuration>                          <source>1.6</source>                          <target>1.6</target>                      </configuration>                  </plugin>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-war-plugin</artifactId>                      <version>2.4</version>                      <configuration>                       <warSourceDirectory>src/main/webapp</warSourceDirectory>                          <warName>Spring4MVCHelloWorldNoXMLDemo</warName>                          <failOnMissingWebXml>false</failOnMissingWebXml>                      </configuration>                  </plugin>              </plugins>          </pluginManagement>          <finalName>Spring4MVCHelloWorldNoXMLDemo</finalName>      </build>  </project> |

|  |
| --- |
|  |

First thing to notice here is the maven-war-plugin declaration. As we will be completely removing web.xml, we will need to configure this plugin in order to avoid maven failure to build war package. Second change is the inclusion of JSP/Servlet/Jstl dependencies which we might be needing as we are going to use servlet api’s and jstl view in our code.In general, containers already contains these libraries, so we can set the scope as provided for them in pom.xml.

Additionally, maven-compiler-plugin has been added here to explicitly specify the jdk-version we are going to use. Do note that it also forces eclipse to respect the jdk-version being used for the project. if it is not present, and you perform mvn-update from within your eclipse, eclipse switches jdk-version back to default jdk-version [1.5] which is annoying. So do add it in your project pom as well.

**Step 3: Add Controller**

Add a controller class under src/main/java as shown below:

com.websystique.springmvc.controller.HelloWorldController

|  |
| --- |
| package com.websystique.springmvc.controller;    import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;    @Controller  @RequestMapping("/")  public class HelloWorldController {        @RequestMapping(method = RequestMethod.GET)      public String sayHello(ModelMap model) {          model.addAttribute("greeting", "Hello World from Spring 4 MVC");          return "welcome";      }        @RequestMapping(value = "/helloagain", method = RequestMethod.GET)      public String sayHelloAgain(ModelMap model) {          model.addAttribute("greeting", "Hello World Again, from Spring 4 MVC");          return "welcome";      }    } |

|  |
| --- |
|  |

@Controller annotation on class name declares this class as spring bean and @RequestMapping annotation declares that this class is default handler for all requests of type ‘/’. First method does not have any mapping declared so, it will inherit the mapping from mapping declared on class level, acting as default handler for GET requests. Second method (due to additional mapping declaration with value attribute) will serve the request of form /helloagain. Attribute method says which type of HTTP request this method can serve. ModelMap is a Map implementation, which here acting as replacement of [request.getAttribute()/request.setAttribute()]setting values as request attribute. Note that we are returning “welcome” string form this method. This string will be suffixed and prefixed with suffix and prefix defined in view resolver(see spring-servlet.xml above) to form the real view file name.

**Step 4: Add View**

Create a new folder named views under WEB-INF and add in a Simple JSP page welcome.jsp (WEB-INF/views/welcome.jsp) to simply access the model value sent from controller.

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=ISO-8859-1"      pageEncoding="ISO-8859-1"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "<http://www.w3.org/TR/html4/loose.dtd>">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  <title>HelloWorld page</title>  </head>  <body>      Greeting : ${greeting}  </body>  </html> |

|  |
| --- |
|  |

**Step 5: Add Configuration Class**

Add the below mentioned class under src/main/java with specified package as shown below. This configuration class can be treated as a replacement of spring-servlet.xml as it contains all the information required for component-scanning and view resolver.

com.websystique.springmvc.configuration.HelloWorldConfiguration

|  |
| --- |
| package com.websystique.springmvc.configuration;    import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.servlet.ViewResolver;  import org.springframework.web.servlet.config.annotation.EnableWebMvc;  import org.springframework.web.servlet.view.InternalResourceViewResolver;  import org.springframework.web.servlet.view.JstlView;    @Configuration  @EnableWebMvc  @ComponentScan(basePackages = "com.websystique.springmvc")  public class HelloWorldConfiguration {      @Bean      public ViewResolver viewResolver() {          InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();          viewResolver.setViewClass(JstlView.class);          viewResolver.setPrefix("/WEB-INF/views/");          viewResolver.setSuffix(".jsp");            return viewResolver;      }    } |

|  |
| --- |
|  |

@Configuration indicates that this class contains one or more bean methods annotated with @Bean producing bean manageable by spring container. Above Configuration class is equivalent to following XML counterpart:

|  |
| --- |
| <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>"      xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans-4.0.xsd>  <http://www.springframework.org/schema/mvc> <http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd>  <http://www.springframework.org/schema/context> <http://www.springframework.org/schema/context/spring-context-4.0.xsd>">        <context:component-scan base-package="com.websystique.springmvc" />        <mvc:annotation-driven />        <bean          class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix">              <value>/WEB-INF/views/</value>          </property>          <property name="suffix">              <value>.jsp</value>          </property>      </bean>    </beans> |

|  |
| --- |
|  |

@EnableWebMvc is equivalent to mvc:annotation-driven in XML. It enables support for @Controller-annotated classes that use @RequestMapping to map incoming requests to specific method.

@ComponentScan is equivalent to context:component-scan base-package="..." providing with where to look for spring managed beans/classes.

**Step 6: Add Initialization class**

Add an initializer class implementing WebApplicationInitializer under src/main/java with specified package as shown below(which in this case acts as replacement of any spring configuration defined in web.xml). During Servlet 3.0 Container startup, this class will be loaded and instantiated and its onStartup method will be called by servlet container.

com.websystique.springmvc.configuration.HelloWorldInitializer

|  |
| --- |
| package com.websystique.springmvc.configuration;    import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.ServletRegistration;    import org.springframework.web.WebApplicationInitializer;  import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;  import org.springframework.web.servlet.DispatcherServlet;    public class HelloWorldInitializer implements WebApplicationInitializer {        public void onStartup(ServletContext container) throws ServletException {            AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();          ctx.register(HelloWorldConfiguration.class);          ctx.setServletContext(container);            ServletRegistration.Dynamic servlet = container.addServlet("dispatcher", new DispatcherServlet(ctx));            servlet.setLoadOnStartup(1);          servlet.addMapping("/");      }    } |

|  |
| --- |
|  |

The content above resembles the content of web.xml from previous tutorial as we are using the front-controller DispatherServler, assigning the mapping (url-pattern in xml) and instead of providing the path to spring configuration file(spring-servlet.xml) , here we are registering the Configuration Class. Overall, we are doing the same thing, just the approach is different.

**UPDATE:** Note that now you can write the above class even more concisely [**and it’s the preferred way**], by extendingAbstractAnnotationConfigDispatcherServletInitializer base class, as shown below:

|  |
| --- |
| package com.websystique.springmvc.configuration;    import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;    public class HelloWorldInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {        @Override      protected Class<?>[] getRootConfigClasses() {          return new Class[] { HelloWorldConfiguration.class };      }        @Override      protected Class<?>[] getServletConfigClasses() {          return null;      }        @Override      protected String[] getServletMappings() {          return new String[] { "/" };      }    } |

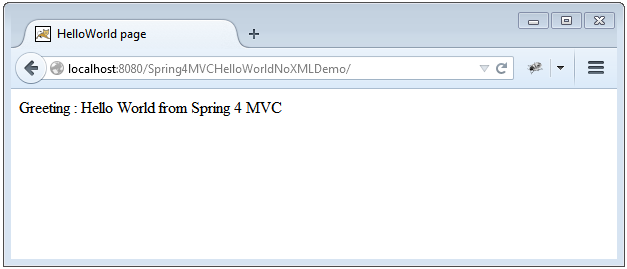
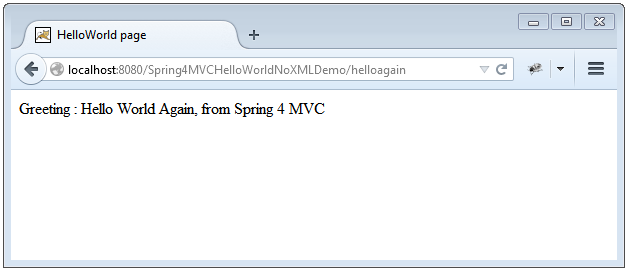
|  |
| --- |
|  |

**Step 7: Build and Deploy the application**

One thing to keep in mind that the Spring java based configuration api’s like WebApplicationInitializer depends on Servlet 3.0 containers.So make sure you don’t have any web.xml with servlet declaration less than 3.0. For our case, we have removed web.xml file from our application.

Now build the war (either by eclipse as was mentioned in last tutorial) or via maven command line( mvn clean install). Deploy the war to a Servlet 3.0 container . Since here i am using Tomcat, i will simply put this war file into tomcat webapps folder and click on start.bat inside tomcat bin directory.

Run the application

That’s it.