Spring 4 MVC Hello World Tutorial – Full Example

Feb 9th, 2014 4 Comments

In this tutorial you will learn how to develop a Spring 4 MVC Hello world example. We hope this tutorial will give you a quick start with Spring MVC development using the latest Spring 4 Release.

Technologies used:

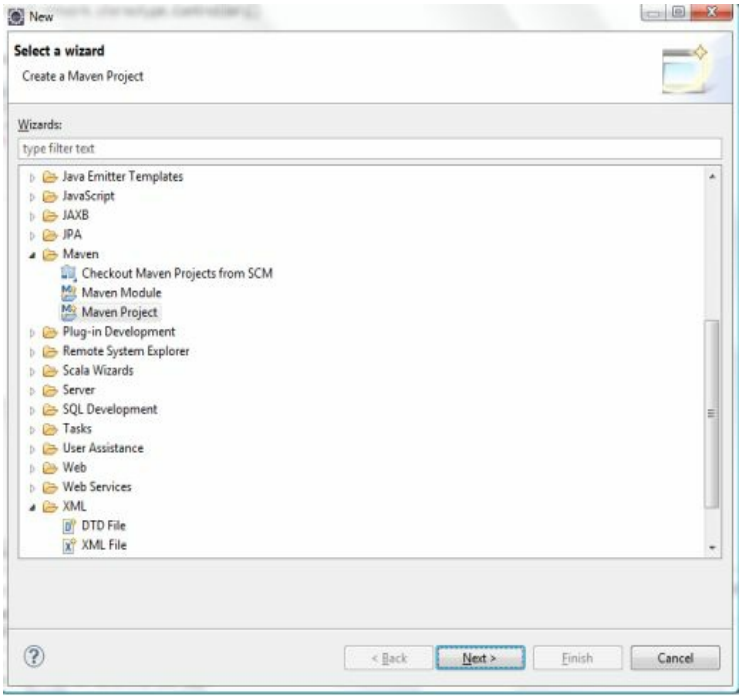
- Spring 4.0.1.RELEASE
- JDK 1.6
- Maven 3
- Eclipse Java EE IDE (Eclipse JUNO)

Updates (10 -Feb -2014): Updated the tutorial with JavaConfig. Now explains how to use **WebApplicationInitializer** and **@Configuration**

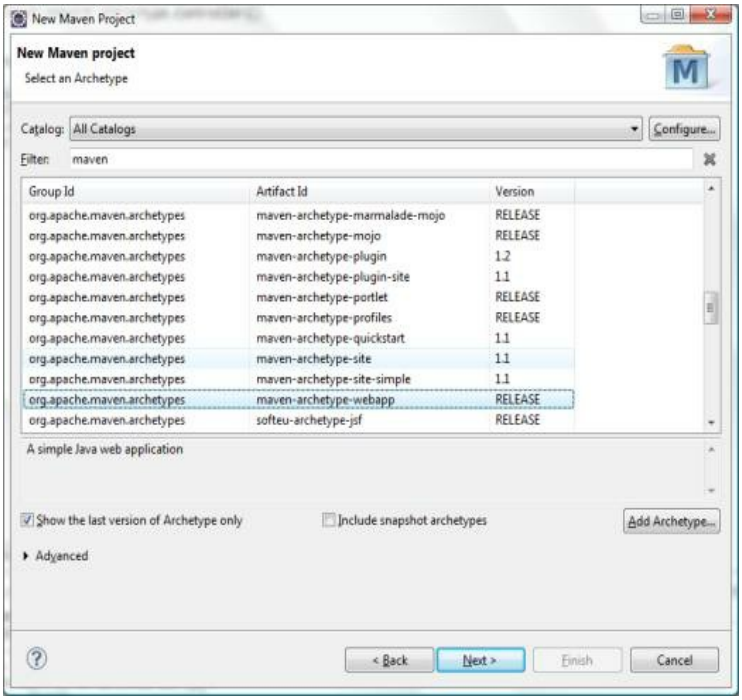
Part 1: Maven Project Setup In Eclipse

Let us start with the creation of a Maven web project in Eclipse. A maven web project archetype will create all the necessary folder structures required for a web project. We assume that you have installed the maven plugins for Eclipse. If you haven't configured it, refer our earlier Spring tutorial that has section explaining how to setup maven in eclipse.

File -> New -> Other -> Maven -> Maven Project



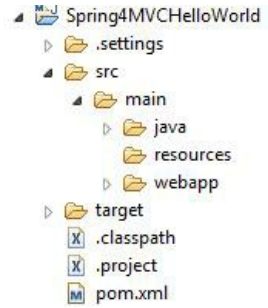
Click Next and Click Next again (If you wish to change default Workspace location , you may do so). In the next screen you should pick the maven web app archetype. Refer the screen below



Click Next and provide the following values

1. GroupId :com.javahash.web (you can change this according to your package structure)
2. Artifact Id: Spring4MVCHelloWorld
3. Version: 1.0-SNAPSHOT

Click Finish to complete the Project Setup.



Part 2: Spring Configuration

We now need to add the spring framework libraries as dependencies in mavan (pom.xml). For the sake of ease, we are going to define a maven variable to hold the spring framework version. If we need to change to a different spring release only this variable needs to be changed.

Dependencies – pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.javahash.web</groupId>
 <artifactId>Spring4MVCHelloWorld</artifactId>
 <packaging>war</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>Spring4MVCHelloWorld Maven Webapp</name>
 <url>http://maven.apache.org</url>
 <properties>
 <spring.version>4.0.1.RELEASE</spring.version>
 </properties>
 <dependencies>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>
 <!-- Spring dependencies -->
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-core</artifactId>
 <version>${spring.version}</version>
 </dependency>

 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-web</artifactId>
 <version>${spring.version}</version>
 </dependency>

 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-webmvc</artifactId>
 <version>${spring.version}</version>
 </dependency>

 </dependencies>
 <build>
 <finalName>Spring4MVCHelloWorld</finalName>
 </build>
</project>
```

Spring Beans Configuration

We need a configuration file that holds the spring configuration information. In this tutorial we will be using the Spring's auto scan feature (annotation) to detect and initialize beans. **We can name this configuration file any name**. We are using the name **dispatcher-servlet.xml** for this project. Place this file inside the **WEB-INF** folder.

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:context="http://www.springframework.org/schema/context"
 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-3.0.xsd


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


http://www.springframework.org/schema/context/spring-context-3.0.x

 <context:component-scan base-package="com.javahash.spring.con

 <bean
 class="org.springframework.web.servlet.view.InternalResourceView
 <property name="prefix">
 <value>/WEB-INF/views/</value>
 </property>
 <property name="suffix">
 <value>.jsp</value>
 </property>
 </bean>
</beans>
```

We have told spring to look at the package com.javahash.spring.controller for the beans and we have also told the framework that all views are kept

under WEB-INF/views folder.

Part 3 : Web App Configuration

Next step is to configure the web app so that it uses Spring's DispatcherServlet as the Front Controller.

Configuring web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app id="WebApp_ID" version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

 <display-name>Archetype Created Web Application</display-name

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

 <servlet-mapping>
 <servlet-name>dispatcher</servlet-name>
 <url-pattern>/</url-pattern>
 </servlet-mapping>

 <context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
 </context-param>

 <listener>
 <listener-class>
 org.springframework.web.context.ContextLoaderListener
 </listener-class>
 </listener>
</web-app>
```

Location of web.xml is inside the WEB-INF Folder

The servlet name we used in the web.xml is dispatcher. Due to this the framework will look for a file (servletname-servlet.xml) to load the Spring MVC configurations. In our case it will be dispatcher-servlet.xml. If we have used a different name for the servlet, say frontcontroller, then the framework will look for a file with name frontcontroller-servlet.xml to load MVC configurations. We can override this behavior by explicitly specifying the mvc configuration file using the parameter **contextConfigLocation.** We have used that in our web.xml

Part 4: Controller Development

From Spring 3 on wards there exists excellent support for annotations. We will use annotations to mark our class as a controller in the standard MVC design. The HelloWorldController is a very simple controller that just echoes a message. It takes a parameter and just echoes it.

```
1  package com.javahash.spring.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.ui.Model;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7
8  @Controller
9  public class HelloWorldController {
10
11 @RequestMapping("/hello")
12  public String hello(@RequestParam(value="name", required=false
13  model.addAttribute("name", name);
14  return "helloworld";
15  }
16
17 }
```

Please note the use of **@Controller** and **@RequestMapping .** The URL takes a parameter with name “name”.

View

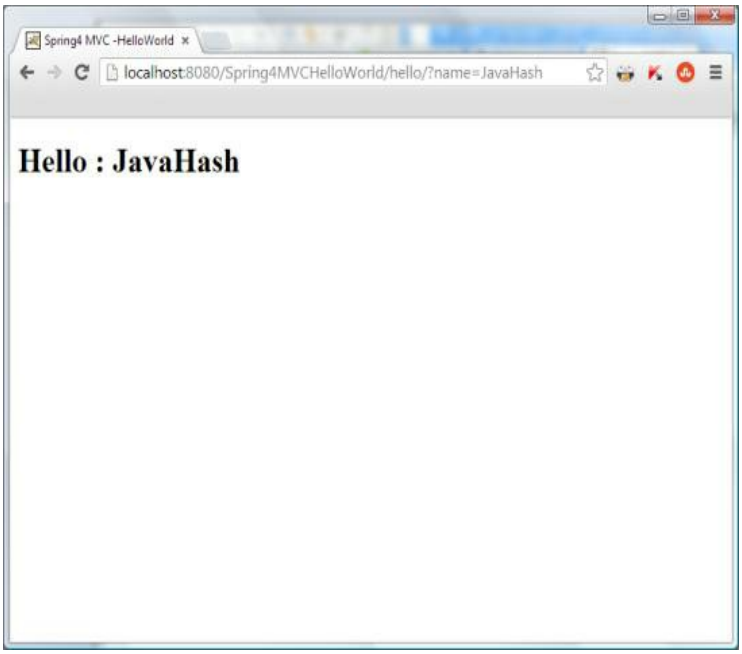
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

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>Spring4 MVC -HelloWorld</title>
</head>
<body>
<h1>Hello : ${name}</h1>
</body>
</html>
```

We can generate a War file and deploy that to a web server to test the application. In eclipse , right click the project and Click Run As – > Maven Install. This will build the project and create a war file in the target folder. In the case of this example the file will be Spring4MVCHelloWorld.war

Deploy this WAR file to a web server , say Tomcat and issue

```
1  http://localhost:8080/Spring4MVCHelloWorld/hello/?name=JavaHash
```



How to Avoid XML Files and use JavaConfig

Maintaining configuration using XML has its advantages and disadvantages. If you are not a fan of XML configuration and wish to enjoy the benefits of annotations based configuration, you can do so easily in Spring. It is your choice to go the XML path or the JavaConfig path. JavaConfig is a cool approach and helps in rapid application development and provides easy maintenance. When the number of artifacts in the project increases, JavaConfig is very handy. Also it is the developer friendly means of handling configuration. Let us see how we can replace the dispatcher-servlet.xml and the Spring Configuration defined in the web.xml to Java classes using **JavaConfig**.

Replacing dispatcher-servlet.xml with Java File

```
1  package com.javahash.spring.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.ComponentScan;
5  import org.springframework.context.annotation.Configuration;
6  import org.springframework.web.servlet.config.annotation.EnableW
7  import org.springframework.web.servlet.view.JstlView;
8  import org.springframework.web.servlet.view.UrlBasedViewResolve
9
10 @Configuration //Marks this class as configuration
11 //Specifies which package to scan
12 @ComponentScan("com.javahash.spring")
13 //Enables Spring's annotations
14 @EnableWebMvc
15 public class Config {
16
17  @Bean
18  public UrlBasedViewResolver setupViewResolver() {
19  UrlBasedViewResolver resolver = new UrlBasedViewResolver();
20  resolver.setPrefix("/WEB-INF/views/");
21  resolver.setSuffix(".jsp");
22  resolver.setViewClass(JstlView.class);
23  return resolver;
24  }
25 }
```

Moving Spring Configuration from web.xml to WebApplicationInitializer

```
1  package com.javahash.spring.config;
2
3  import javax.servlet.ServletContext;
4  import javax.servlet.ServletException;
5  import javax.servlet.ServletRegistration.Dynamic;
6
7  import org.springframework.web.WebApplicationInitializer;
8  import org.springframework.web.context.support.AnnotationConfig\
9  import org.springframework.web.servlet.DispatcherServlet;
10
11 public class WebInitializer implements WebApplicationInitializer {
12
13 public void onStartup(ServletContext servletContext) throws Serv
14
15  AnnotationConfigWebApplicationContext ctx = new AnnotationCont
16  ctx.register(Config.class);
17
18  ctx.setServletContext(servletContext);
19
20  Dynamic servlet = servletContext.addServlet("dispatcher", new Dis
21  servlet.addMapping("/");
22  servlet.setLoadOnStartup(1);
23
24  }
25
26 }
```

Download Source Code

Download – Project Source Code

References

Spring 4

Tags: S p r i n g

About prem

Prem is passionate about Java, Web technologies and massive data processing systems. He has vast amount experience in designing and architecting innovative products and projects for world wide audiences spanning airlines, retail , travel and logistics domain. He has worked with major IT companies. Visit him at Google Plus

View all posts by prem →

Related Posts

How to Inject Property Value in Spring Bean

Spring jdbc template example

Why Spring framework