Build spring web application

. in STS, create the maven project and select webapp type project

. resolve the error which is caused by the default servlet-api version. The default servlet-api version of maven web application is 2.3, but the default version of STS is 2.5. none of version is correct for us. The correct version depends on the application server we will use. Therefore, we must import the runtime application server lib into the project lib.

manually load env. provided servlet to the library

right click on project -> Properties -> Java Build Path -> Add Library...-> Server Runtime -> Apache Tomcat -> Finish

. Add spring mvc configuration

Reference:

<http://javahash.com/spring-4-mvc-hello-world-tutorial-full-example/>

Spring 4 MVC Hello World Tutorial – Full Example

Posted on February 9, 2014 in [Hello World](http://javahash.com/category/hello-world/), [Spring Framework](http://javahash.com/category/spring-framework/)

Twitter 0Facebook 32Google+ 40

LinkedIn 4[inShare](javascript:void(0);)4Filament.io [Made with **FLARE**More Info](https://filament.io/applications/flare?utm_source=flare_wp&utm_medium=deployment&utm_content=flarebar&utm_campaign=filament)**76**FLARES

Table of Contents

1. **Technologies Used**
2. **Project Setup using Maven**
3. **Spring Bean Configuration**
4. **Web App Configuration**
5. **Developing the Controller**
6. **Developing the View**
7. **Running using browser**
8. **Using Java Config instead of XML**

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

**Technologies used**:

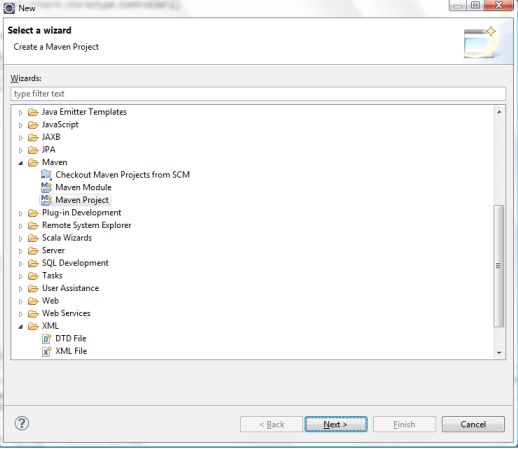
1. **Spring 4.0.1.RELEASE**
2. **JDK 1.6**
3. **Maven 3**
4. **Eclipse Java EE IDE ( Eclipse JUNO)**

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

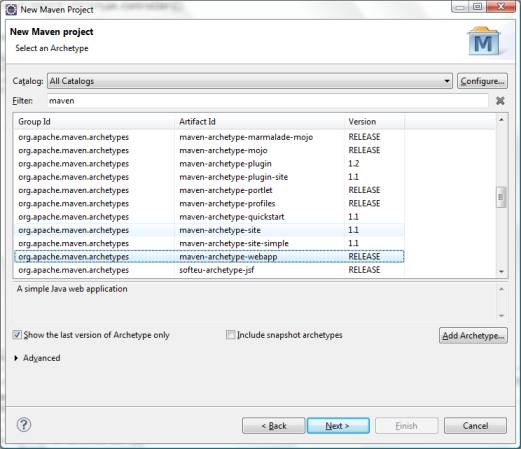
1: Maven Project Setup In Eclipse

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](http://javahash.com/spring-3-hello-world-example-quick-introductory-tutorial/) that has section explaining how to setup maven in eclipse.

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



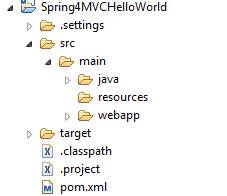
Click Next and Click Next again ( If you wish to change default Workspace location , you may do so). In the next screen you should pick the maven web app archetype. Refer the screen below

[](http://javahash.com/wp-content/uploads/2014/02/CreateMavenProjectStep2.jpg)

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.

[](http://javahash.com/wp-content/uploads/2014/02/ProjectStructure.jpg)

This completes the project setup. Make sure the project is loaded in eclipse.

2:  Spring Configuration

Activities in this section can be broadly defined as

1. Adding Spring libraries as dependencies in pom.xml
2. Defining an XML file that holds the configuration for spring beans
3. Configuring web.xml

Spring libraries are added as dependencies in maven (pom.xml).

**Dependencies – pom.xml**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">    <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>4.12</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> |

2.1 Configuring Spring Beans (dispatcher-servlet.xml)

Create an XML file and name it dispatcher-servlet.xml. T**his configuration file can be given any name**. We are using the name**dispatcher-servlet.xml** for this project. Place this file inside the **WEB-INF** folder. This xml file holds the configuration info for the view resolver that spring uses to map view names to a concrete view files.

|  |  |
| --- | --- |
| 1 | <context:component-scan base-package="com.javahash.spring.controller" /> |

Above piece of code instructs spring to scan the package (com.javahash.spring.controller) and its children to detect and auto configure components. For instance, we can annotate a class with @Controller annotation and spring will automatically configure it as a controller class. Due to this auto scanning feature, there is no need to configure controller classes in xml files.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <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-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.javahash.spring.controller" />    <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> |

3: Configuring web.xml

The  org.springframework.web.servlet.DispatcherServlet acts as the front controller for the application. All requests are intercepted by the spring servlet.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <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> |

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

The default rule is spring looks for a file honoring the contract , servletname-servlet.xml to load the spring configuration beans. Because of this, spring will look for the file ( dispatcher-servlet.xml). If we had 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 the above web.xml

4: Developing the Controller

Starting with Spring 3, there is excellent support for annotations. We can use annotations to mark our class as a controller. The HelloWorldController is a very simple controller that just echoes a message. It takes a parameter and just echoes it.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package com.javahash.spring.controller;    import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;    @Controller  public class HelloWorldController {    @RequestMapping("/hello")  public String hello(@RequestParam(value="name", required=false, defaultValue="World") String name, Model model) {       model.addAttribute("name", name);     //returns the view name     return "helloworld";    }    } |

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

**5: Developing the View**

Create a file named **helloword.jsp** inside WEB-INF/views folder.  In the dispatcher-servlet.xml, defined in section 2.1, we have specified that the jsp files will be put inside the WEB-INF/view folder.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <%@ 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> |

At run-time, spring will choose the appropriate .jsp file based on the view-name returned by the controller method.

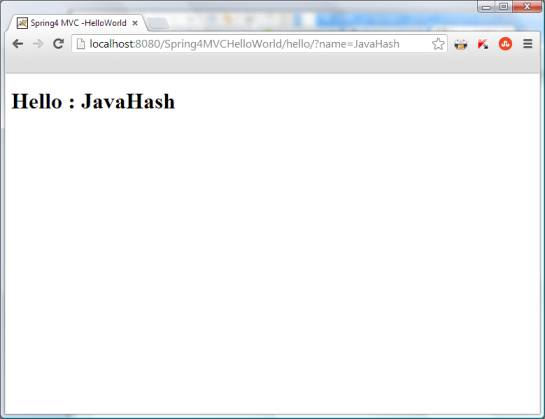
**6: Build and Package**

Right click the project , RunAs -> **maven install**

This will compile, package and install the executable as a WAR file. You can also perform **maven package** instead of maven install. Spring4MVCHelloWorld.war will be generated inside the target folder.

Deploy this WAR file to Tomcat web server or any web server you are using.

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

[](http://javahash.com/wp-content/uploads/2014/02/Results.jpg)

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

**Moving Spring Configuration from web.xml to**[**WebApplicationInitializer**](http://docs.spring.io/spring/docs/3.1.x/javadoc-api/org/springframework/web/WebApplicationInitializer.html)

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

Spring mvc configuration

From the [Spring Official Website](http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-basics):

The interface org.springframework.context.ApplicationContext represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the aforementioned beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata. The configuration metadata is represented in XML, Java annotations, or Java code.

Again from official Doc:

In the Web MVC framework, each DispatcherServlet has its own WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext. These inherited beans can be overridden in the servlet-specific scope, and you can define new scope-specific beans local to a given Servlet instance.

Now coming to your Question, as is stated [here](http://syntx.co/languages-frameworks/difference-between-loading-context-via-dispatcherservlet-and-contextloaderlistener/):

In Spring Web Applications, there are two types of container, each of which is configured and initialized differently. One is the “Application Context” and the other is the “Web Application Context”. Lets first talk about the “Application Context”. Application Context is the container initialized by a ContextLoaderListener or ContextLoaderServlet defined in the web.xml and the configuration would look something like this:

<listener>

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

</listener>

<context-param>

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

<param-value>classpath:\*-context.xml</param-value>

</context-param>

In the above configuration, I am asking spring to load all files from the classpath that match \*-context.xml and create an Application Context from it. This context might, for instance, contain components such as middle-tier transactional services, data access objects, or other objects that you might want to use (and re-use) across the application. There will be one application context per application.

The other context is the “WebApplicationContext” which is the child context of the application context. Each DispatcherServlet defined in a Spring web application will have an associated WebApplicationContext. The initialization of the WebApplicationContext happens like this:

<servlet>

<servlet-name>platform-services</servlet-name>

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

<init-param>

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

<param-value>classpath:platform-services-servlet.xml</param-value>

</init-param>

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

</servlet>

You provide the name of the spring configuration file as a servlet initialization parameter. What is important to remember here is that the name of the XML must be of the form -servlet. xml. In this example, the name of the servlet is platform-services therefore the name of our XML must be platform-service-servlet.xml. Whatever beans are available in the ApplicationContext can be referred to from each WebApplicationContext. It is a best practice to keep a clear separation between middle-tier services such as business logic components and data access classes (that are typically defined in the ApplicationContext) and web- related components such as controllers and view resolvers (that are defined in the WebApplicationContext per Dispatcher Servlet).

Usually, global scope of applicationContext declares everything under controller including dataSource, Dao, Manager or Service and helper classes. WebApplicationContext should have only the mvc related controller or helper classes. The global beans can be shared by multiple webApplicationContext and also can be overridden by WebApplicationContext. Ahm doesn’t have the best practice yet. The dataSources have been defined at the WebApplicationContext xml file.

Check these links

[difference between applicationContext.xml and spring-servlet.xml in spring](http://stackoverflow.com/questions/3652090/difference-between-applicationcontext-xml-and-spring-servlet-xml-in-spring)

<http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-basics>