About viralpatel.net
Join Us
Advertise
Search

VIRALPATEL.NET

RECORDED WEBINAR
veeam
SQL Server Granular Recov
Treat Your Databases Right

Home
Android
Java
Spring
Frameworks
Database
JavaScript
Web
More...

FOLLOW:

JVMHOST
Many flavours of Java Hosting

Spring 4 MVC Tutorial Maven Example – Spring Java Configuration

BY VIRAL PATEL · JUNE 24, 2016

RECENT POSTS

Spring 4 MVC Tutorial with Eclipse, Maven – Spring 4 MVC is the newer version of our favorite Java MVC framework. A lot has improved in Spring since the Spring 3 MVC. In this tutorial we will create a simple web application from scratch in Eclipse that will use Spring's latest version 4 MVC framework and Maven configuration. Also we will use Java configuration to configure Spring instead of older XML configuration.

Getting started with Spring 4 MVC Tutorial

1. Create a new Maven project

First things first, we will bootstrap a quick Maven project in Eclipse. Follow these simple steps and create a simple webapplication.

Spring 4 MVC REST Controller Example (JSON CRUD Tutorial)

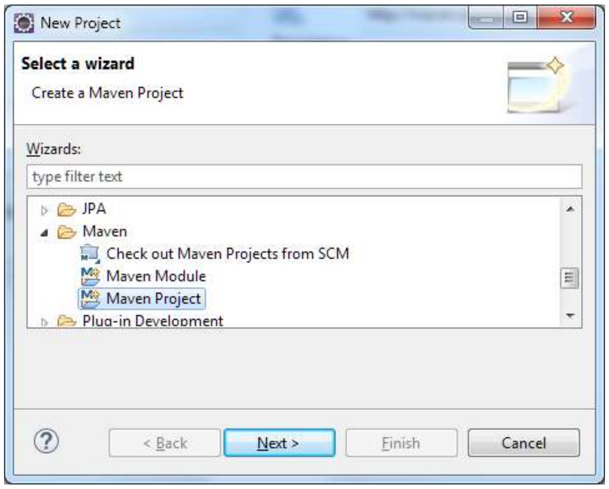
Spring 4 MVC Tutorial Maven Example – Spring Java Configuration

Find Process ID of Process using a Port in Windows

How to Embed Tomcat within Maven Project – Run Tomcat with Maven

Bootstrap Navbar Menu without JavaScript

1.1 First in Eclipse go to **File -> New** and from New project dialog select **Maven Project**.


New Project

Select a wizard

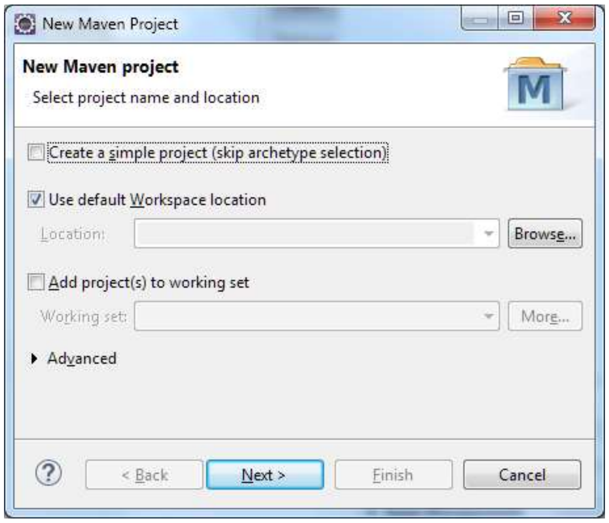
Create a Maven Project

Wizards:

type filter text

- JPA
- Maven
  - Check out Maven Projects from SCM
  - Maven Module
  - Maven Project
- Plug-in Development

1.2 From New Maven Project dialog, leave the options as shown below and press **Next**.


New Maven Project

New Maven project

Select project name and location

[ ] Create a simple project (skip archetype selection)

[✓] Use default Workspace location

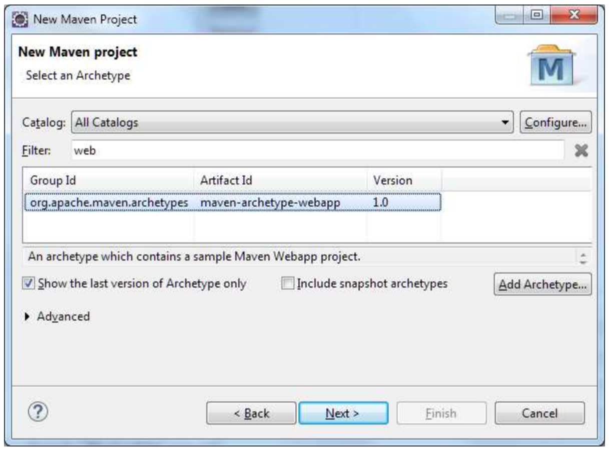
Location:

[ ] Add project(s) to working set

Working set:

Advanced

1.3 Now select the project Archetype from the options. Type *"web"* in the filter text and select **maven-archetype-webapp**.



New Maven project

Select an Archetype

Catalog: All Catalogs

Filter: web

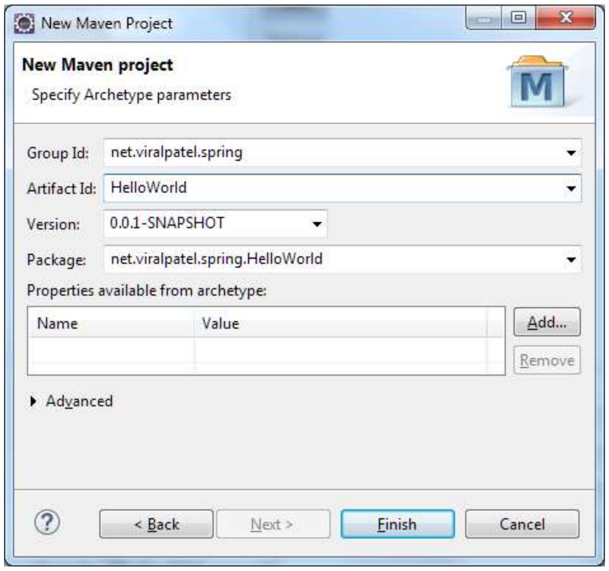
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven Webapp project.

[✓] Show the last version of Archetype only

[ ] Include snapshot archetypes

1.4 As shown below, provide the Group Id and Artifact Id. For this example I have given Group Id `net.viralpatel.spring` and Artifact Id as `HelloWorld`.



New Maven project

Specify Archetype parameters

Group Id: net.viralpatel.spring

Artifact Id: HelloWorld

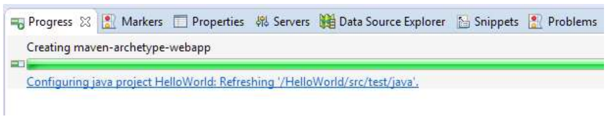
Version: 0.0.1-SNAPSHOT

Package: net.viralpatel.spring.HelloWorld

Properties available from archetype:

Name	Value

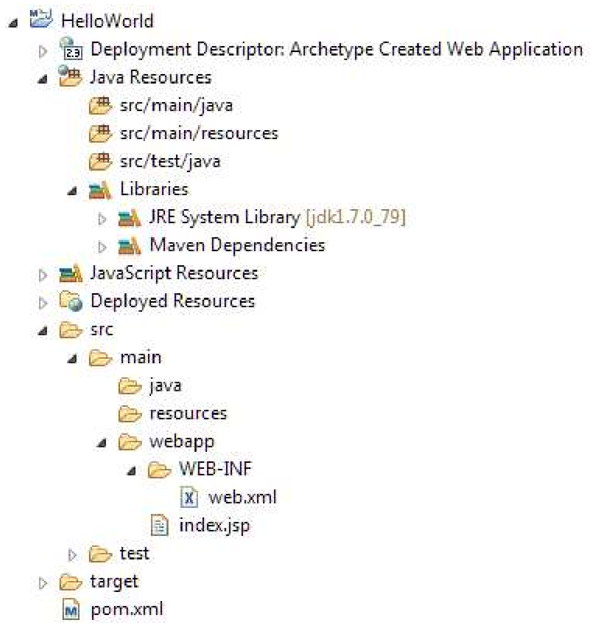
1.5 Once you press **Finish**, Eclipse should start generating Maven webapp using maven-archetype-webapp. **Progress** view should show the progress of this step.



Creating maven-archetype-webapp

Configuring java project HelloWorld: Refreshing '/HelloWorld/src/test/java'.

New project is created with `pom.xml`, `WebContent` folder and `src` folder.



```
HelloWorld
  Deployment Descriptor: Archetype Created Web Application
  Java Resources
    src/main/java
    src/main/resources
    src/test/java
    Libraries
      JRE System Library [jdk1.7.0_79]
      Maven Dependencies
  JavaScript Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        WEB-INF
          web.xml
        index.jsp
    test
  target
  pom.xml
```

2. Add Spring 4 MVC Maven dependencies

Project structure is created. Now let's start and add first the maven dependencies for Spring 4 MVC in our `pom.xml` file.

Update pom.xml file and add following dependencies.

pom.xml

```
<project xmlns="http://maven.apach
    xsi:schemaLocation="http://ma\
    <modelVersion>4.0.0</modelVers
    <groupId>net.viralpatel.spring
    <artifactId>HelloWorld</artifa
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</versi
    <name>HelloWorld Maven Webapp<
    <url>http://maven.apache.org</
    <properties>
        <java-version>1.7</java-ve
    </properties>
    <dependencies>
        <dependency>
            <groupId>javax.servlet
            <artifactId>javax.serv
            <version>3.0.1</versic
            <scope>provided</scope
        </dependency>
        <dependency>
            <groupId>javax.servlet
            <artifactId>jstl</arti
            <version>1.2</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>HelloWorld</fir
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.a
                    <artifactId>ma
                    <version>2.3.2
                    <configuration
                        <source>${
                        <target>${
                    </configuratic
                </plugin>
                <plugin>
                    <groupId>org.a
                    <artifactId>ma
                    <version>2.4</
                    <configuration
                        <warSource
                        <warName>h
                        <failOnMis
                    </configuratic
                </plugin>
            </plugins>
        </pluginManagement>
        </build>
    </project>
```

After updating pom.xml, Eclipse's maven plugin should start resolving the dependencies.

3. Set Annotation based Configuration for Spring 4 MVC tutorial

For this Spring 4 MVC tutorial we are going to use Spring's Java based configuration or annotation based configuration instead of old XML configuration. So now lets add the Java Configuration required to bootstrap Spring 4 MVC example in our webapp.

Create `AppConfig.java` file under `/src` folder. Give appropriate package name to your file.

```
/src/main/java/net/viralpatel/spring
/config/AppConfig.java
```

```
package net.viralpatel.spring.config;

import org.springframework.context.an
import org.springframework.context.an
import org.springframework.context.an
import org.springframework.web.servle
import org.springframework.web.servle
import org.springframework.web.servle
import org.springframework.web.servle
import org.springframework.web.servle
import org.springframework.web.servle
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "net.vi
public class AppConfig extends WebMvc
    @Bean
    public ViewResolver viewResolver(
        InternalResourceViewResolver
        viewResolver.setViewClass(Jst
        viewResolver.setPrefix("/WEB-
        viewResolver.setSuffix(".jsp"

        return viewResolver;
    }

    @Override
    public void configureDefaultServl
        configurer.enable();
    }

}
```

AppConfig class is annotated with Spring's annotations such as `@Configuration`, `@EnableWebMvc` and `@ComponentScan`.

The `@Configuration` annotation indicates that the class declares one or more `@Bean` methods. These methods are invoked at runtime by Spring to manage lifecycle of the beans. In our case we have defined `@Bean` for view resolver for JSP view.

The `@EnableWebMvc` is equivalent to `<mvc:annotation-driven />` in XML. It enables support for `@Controller`-annotated classes that use

`@RequestMapping` or `@GetMapping` to map incoming requests to certain methods.

The `@ComponentScan` annotation is equivalent to `<context:component-scan>` in XML. It will scan through the given package and register all the Controllers and beans.

The `configureDefaultServletHandling()` method is overridden and we enable default servlet handler. This will let other http request such as .css, .js slip through the usual DispatcherServlet and let the container process them. So now we can serve the static files css and javascript from our WebApp folder.

The above Annotation based configuration is equivalent to following XML configuration.

```
<?xml version="1.0" encoding="UTF-8"?
<beans xmlns="http://www.springframew
    xmlns:mvc="http://www.springframe
    xmlns:context="http://www.springf
    xmlns:xsi="http://www.w3.org/2001
    xsi:schemaLocation="
        http://www.springframework.or
        http://www.springframework.or
        http://www.springframework.or
        http://www.springframework.or
        http://www.springframework.or
        http://www.springframework.or

    <mvc:annotation-driven />

    <context:component-scan base-pack

    <bean id="jspViewResolver" class=
        <property name="viewClass" va
        <property name="prefix" value
        <property name="suffix" value
    </bean>

    <mvc:default-servlet-handler />

</beans>
```

4. Set Servlet 3.X Java Configuration

Create `AppInitializer` class under config package. This class will replace web.xml and it will map the spring's dispatcher servlet and bootstrap it.

```
/src/main/java/net/viralpatel/spring
/config/AppInitializer.java
```

```
package net.viralpatel.spring.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException
import javax.servlet.ServletRegistrat

import org.springframework.web.WebApp
import org.springframework.web.contex
import org.springframework.web.servle

public class AppInitializer implement

    public void onStartup(ServletCont

        AnnotationConfigWebApplicatio
        ctx.register(AppConfig.class)
        ctx.setServletContext(contain

        ServletRegistration.Dynamic s

        servlet.setLoadOnStartup(1);
        servlet.addMapping("/");
    }

}
```

We have configured the dispatcher servlet using standard Java based configuration instead of the older `web.xml`. Thus web.xml is no longer required and we can simply delete it.

5. Create the Controller

Create a sample controller `HelloController.java` under controller package. This will have a simple `hello()` method that act as starting point. Notice how we have used `@GetMapping` annotation provided as part of Spring 4 MVC. This is equivalent to @RequestMapping GET.

/src/main/java/net/viralpatel/spring
/controller/HelloController.java

```
package net.viralpatel.spring.control

import org.springframework.stereotype
import org.springframework.ui.Model;
import org.springframework.web.bind.a

@Controller
public class HelloController {

    @GetMapping("/hello")
    public String hello(Model model)

        model.addAttribute("name", "J

        return "welcome";
    }
}
```

6. Create the View and Stylesheet

6.1 Create welcome.jsp file under WEB-INF/views folder. This will be our primary view file.

```
/src/main/webapp/WEB-INF/views/welcome.jsp
```

```
<%@taglib uri="http://java.sun.com/js
<!DOCTYPE html>
<html>
<head>
    <title>Spring 4 MVC Hello World E
    <link rel='stylesheet' href='<c:u
</head>
<body>
    <h2>Hello World, Spring MVC</h2>

    <p>Welcome, ${name}</p>
</body>
</html>
```

6.2 Next creat the stylesheet for Spring MVC sample.

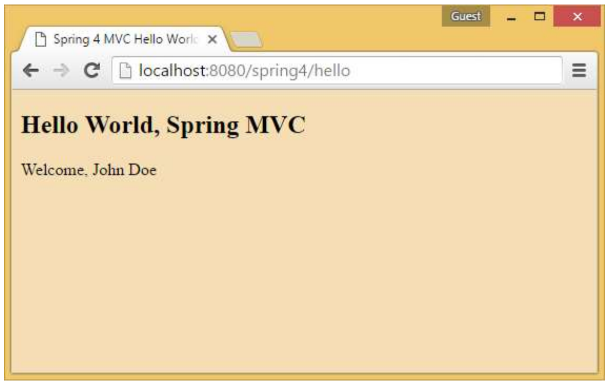
```
/src/main/webapp/resources/css/style.css
```

```
body {
    background-color: wheat;
}
```

That's All Folks

It's time to execute the project. In Eclipse you can start Tomcat and run the project inside it. Or you can run the project using Embedded Tomcat using Maven.

Once the application starts successfully, launch the browser and open `http://localhost:8080/spring4/hello`.



Hello World, Spring MVC

Welcome, John Doe

Download Source Code – Spring 4 MVC Tutorial

Source code of this Spring 4 MVC Hello World tutorial is available in Github.

Download – spring4-mvc-example.zip (6.54 KB)
Github – spring4-mvc-example.git

Related Articles

1. **Spring 4 MVC REST Controller Example (JSON CRUD Tutorial)**
2. **Spring @RequestHeader Annotation example**

3. **Change spring-servlet.xml Filename (Spring Web Contenxt Configuration Filename)**
4. **Spring 3 MVC Interceptor tutorial with example**
5. **Tutorial:Create Spring 3 MVC Hibernate 3 Example using Maven in Eclipse**
6. **Spring MVC Exception Handling using @ControllerAdvice annotation**
7. **Spring MVC Cookie example**

Tags: apache maven eclipse

PREVIOUS STORY
Find Process ID of Process using a Port in Windows

NEXT STORY
Spring 4 MVC REST Controller Example (JSON CRUD Tutorial)

YOU MAY ALSO LIKE...

Spring 3 MVC: Themes in Spring - Tutorial with Example

Spring MVC Exception Handling using @Controller Advice annotation

Spring Roo: Two Databases Configuration

Spring MVC: Multiple Row Form Submit using List of Beans

10 COMMENTS

Nodine 22 July, 2016, 20:16
Hello,
Thank you for this usefull tutorial.
Juste a question please, why do you put "spring4" in the test URL ?
http://localhost:8080/spring4/hello.

Best Regards
Nordine

Reply

Eddie
28 October, 2016, 10:32
The example is run under maven tomcat 7 plugin:

org.apache.tomcat.maven
tomcat7-maven-plugin
…

/spring4
…
You can check it out in pom.xml and the above link Embedded Tomcat using Maven.
If you build it at war and deploy to a standalone tomcat instance, you need to use the war name as context

Thanks
Eddie

Reply

Nilesh 1 September, 2016, 15:55
Hello,
This tutorial is not working for me. Can you tell me why did you put "spring" in the test URL and not the actual context name?

Thanks,
Nilesh

Reply

Sucheta Hardikar
1 September, 2016, 19:37
I have copied your complete project for Spring MVC. But value of name mapped through Model object is not accessible in the JSP view. I tried with ELIgnored attribute , by enableing EL

by using page directive in the JSP. But still value of name is not accessible through JSP . What is the reason ?

Reply

pape
16 September, 2016, 4:50

try with instead of ${name}

Reply

pape
16 September, 2016, 4:51

try with c:out value="${name}"

Reply

Aegis 28 September, 2016, 1:58

You need to include spring taglibs in the jsp to get the mapping correct.

Reply

Anjul Goel 2 October, 2016, 23:03

Hi Sucheta,

The reason the name is not accesible is because you are using an old JSP 1.2 descriptor. Look out for the solution is the below link. https://www.mkyong.com/spring-mvc/modelandviews-model-value-is-not-displayed-in-jsp-via-el/

Reply

Geetha 13 October, 2016, 18:53

Hello Sir, Thanks for sharing, Good Work.

Reply

Sandeep 28 October, 2016, 13:27

Could you please tell me why css is not loading in jsp. I am not able to see background color.

Reply

LEAVE A REPLY

Comment

Name *

Email *

Website

Post Comment