**Spring-Boot-Tutorials**

|  |  |  |
| --- | --- | --- |
| Written by [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  |  |

* [Spring Boot – Introduction Tutorial ( Don’t Miss )](https://www.java4s.com/spring-boot-tutorials/spring-boot-introduction-tutorial-dont-miss/)
* [Spring Boot + Maven – Hello World Example Step by Step](https://www.java4s.com/spring-boot-tutorials/spring-boot-maven-hello-world-example-step-by-step/)
* [Spring Boot – Creating a RESTful Web Service Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/)
* [Spring Boot – Common Application Properties (application.properties)](https://www.java4s.com/spring-boot-tutorials/spring-boot-common-application-properties-application-properties/)
* [Spring Boot – How to Change Default Tomcat Server Port](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-change-default-tomcat-server-port/)
* [Spring Boot – How to Change Default Context Path](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-change-default-context-path/)
* [Spring Boot – How to Reload Changes Without Restarting the Server](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-reload-changes-without-restarting-the-server/)
* [Spring Boot JDBC + MySQL – How to Create/Configure a DataSource](https://www.java4s.com/spring-boot-tutorials/spring-boot-jdbc-mysql-how-to-createconfigure-a-datasource/)
* [Spring Boot JDBC + MySQL – How to Configure Multiple DataSource](https://www.java4s.com/spring-boot-tutorials/spring-boot-jdbc-mysql-how-to-configure-multiple-datasource/)
* [How to Deploy Spring Boot Applications on External Tomcat Server](https://www.java4s.com/spring-boot-tutorials/how-to-deploy-spring-boot-applications-on-external-tomcat-server/)
* [Spring Boot + Spring Security – RESTful Web Service with basic Authentication](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-security-restful-web-service-with-basic-authentication/)
* [Spring Boot + Spring Security – RESTful Web Service with Database Authentication](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-security-restful-web-service-with-database-authentication/)
* [Spring Boot + Spring MVC + JSP Hello World Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-mvc-jsp-hello-world-example/)
* [Spring Boot – Example of RESTful Web Service with XML Response](https://www.java4s.com/spring-boot-tutorials/spring-boot-example-of-restful-web-service-with-xml-response/)
* [Spring Boot – RESTful Web Service with POST Request in JSON Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-restful-web-service-with-post-request-in-json-format/)
* [Spring Boot – RESTful Web Service with POST Request in XML Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-restful-web-service-with-post-request-in-xml-example/)
* [Spring Boot – Display All Beans Available in ApplicationContext](https://www.java4s.com/spring-boot-tutorials/spring-boot-display-all-beans-available-in-applicationcontext/)
* [How to Configure Cache in Spring Boot Applications](https://www.java4s.com/spring-boot-tutorials/how-to-configure-cache-in-spring-boot-applications/)
* [Spring Boot Configure DataSource Using JNDI with Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-configure-datasource-using-jndi-with-example/)

# **Spring Boot – Introduction Tutorial ( Don’t Miss )**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Aug 27, 2017 [**{ 15 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-introduction-tutorial-dont-miss/#comments) By Sivateja

Spring Boot is a framework developed on top of core spring framework. The main aim of Spring Boot is to let developers to create spring production grade applications and services with very less effort. Did you remember, what it takes to create real-time spring applications? It includes writing many XML configurations, server setting, adding dependencies…etc. But with spring Boot we can avoid all these boilerplate code, writing XML configurations and annotations. We can create a real-time production ready applications with in minutes.

Spring Boot comes with inbuilt server, we no longer have to use any external servers like Tomcat, Glass-fish or anything else, so don’t need to deploy WAR files 🙂

So, as I said main advantage of Spring Boot is, we can create spring based applications easily in very less time, without need of any XML configurations. The main disadvantage is, it will be little tough to migrate existing spring enterprise applications to Spring Boot.

Remember, we have to use either Maven or Gradle build tool to work with Spring Boot. Just don’t worry, I will explain about the Maven/Gradle configurations while giving the example and one more thing is, Spring Boot provides command line interface tool to develop/test the Spring Boot applications from the command prompt easily.

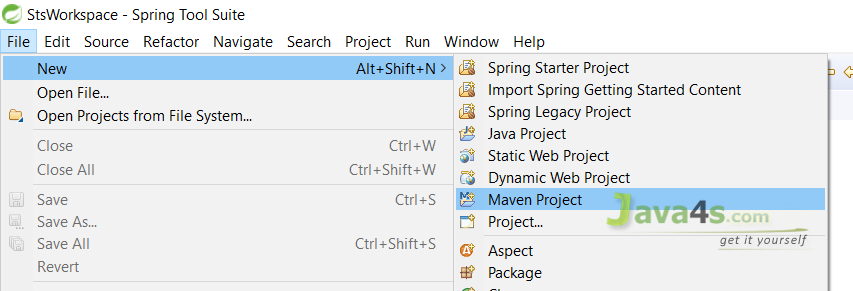
Friends, I will explain all Spring Boot concepts as easy as I can, but while you are reading just don’t skip any article as each article will have dependency with its previous article. Believe me, you can finish Spring Boot tutorials with in less than an Hour for sure, happy learning 😉

# **Spring Boot + Maven – Hello World Example Step by Step**

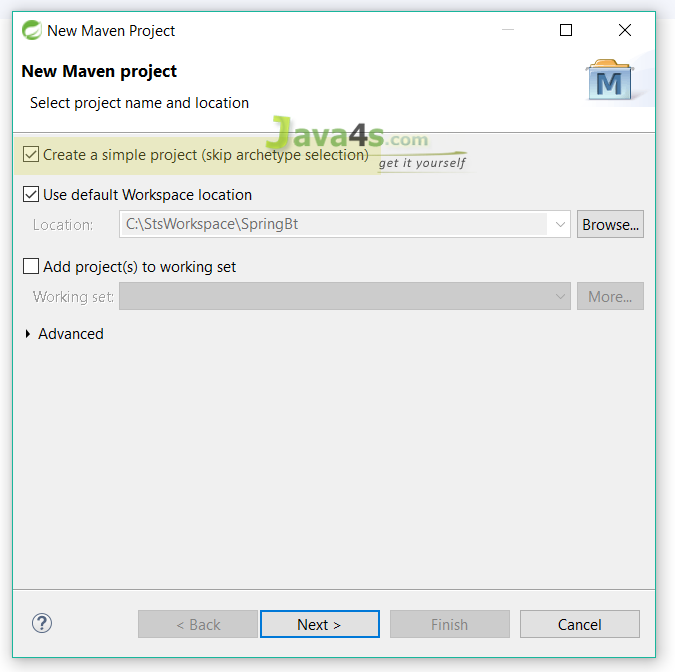
[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Aug 27, 2017 [**{ 24 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-maven-hello-world-example-step-by-step/#comments) By Sivateja

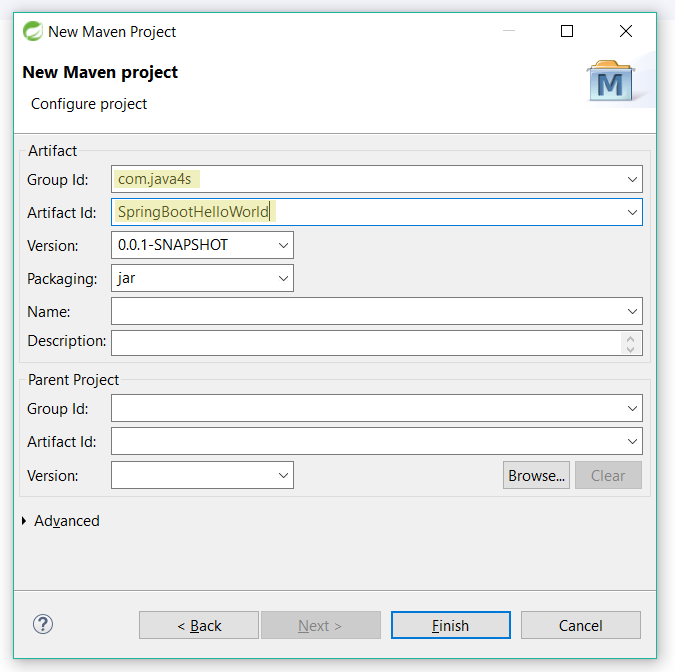
In this article, I am going to explain the steps to create a Spring Boot hello world application using Spring Tool Suite(STS) and Maven. Friends follow this article carefully, as this is the first spring boot application I am going to explain each and every step with screenshot, from the next tutorial on words, I will directly start with directory structure.

1. Open STS (Spring Tool Suite) > File > New > Maven Project

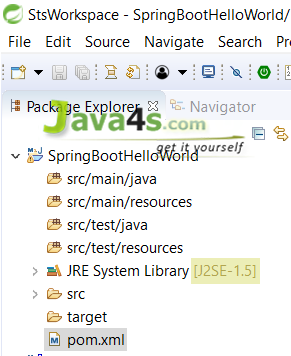
[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Maven-Project.png)

2. Tick ‘Create a simple project (skip archetype selection)‘ check box > click Next

[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Maven-Project-step-1.png)

3. Provide Group Id (its your package), Artifact Id (project name) and click Finish  
[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Maven-Project-step-2.png)

4. Now you will see a Maven project in your work space, something like..

[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Maven-Directory-Structure.png)

**Note**: If you observe, its showing ‘JRE System Library [J2SE-***1.5***]’ as default java version, lets keep it a side for now.

5. So, Maven project is created with default setup, lets add Spring Boot related stuff in the pom.xml ( Guys, hope you have a basic idea about Maven, in pom.xml we will include all dependencies ), open pom.xml

By default pom.xml contains

123456<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

</project>

Lets add Spring Boot related stuff in it

## pom.xml

123456789101112131415161718192021222324<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<properties>

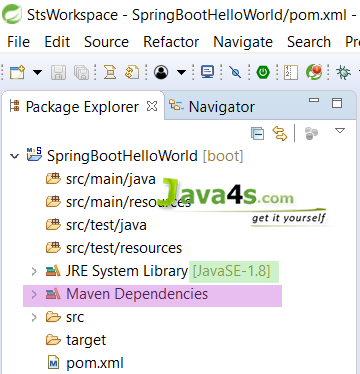
<java.version>1.8</java.version>

</properties>

</project>

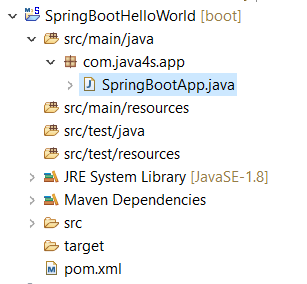
## Explanation

* I have added spring-boot-starter-parent, spring-boot-starter-web and I want to show Spring Boot tutorials in Java 8, so I have added java version at line number 21
* What is spring-boot-starter-parent? actually this is an existing project given by spring team which contains Spring Boot supporting configuration data (remember just configuration data, it wont download any jars), we have added this in a <parent> tag means, we are instructing Maven to consider our SpringBootHelloWorld project as a child to it, wait for a second, I will show you practically why we have to add spring-boot-starter-parent as parent  🙂
* In the dependencies, I have added spring-boot-starter-web for web module

6. Now right click on the application > Maven > Update Project, If you now observe the directory structure of the project, it will create a new folder named “Maven Dependencies” which contains all supporting .jars to run the Spring Boot application and the Java version also changed to **1.8  
  
[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Directory-Structure-After-Maven.png)**

**Note**: Did you observe lines 13-18 of pom.xml? I haven’t included version number for spring-boot-starter-web 🙂 but maven downloaded some jar files with some version(s) related to spring-boot-starter-web, how its possible? that’s because of Maven’s parent child relation. While adding spring boot parent project, I have included version as 1.5.6.RELEASE, so again we no need to add version numbers for the dependencies.  As I told you earlier, spring-boot-starter-parent contains configuration meta data, this means, it knows which version of dependency need to be downloaded.  So we no need to worry about dependencies versions., which will save lot of our time 😉

7.  Now create a java class in src/main/java > I have created one with name SpringBootApp.java in com.java4s.app package. I mean the final directory structure looks like…

[](https://www.java4s.com/wp-content/uploads/2017/08/spring-boot-directory-structure.PNG)

**Note**: put your java class in some package, it is mandatory.\*  If you haven’t created a package it gives the following error while running your Spring Boot application.

Your ApplicationContext is unlikely to start due to a @ComponentScan of the default package.

## SpringBootApp.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp

{

public static void main(String[] args)

{

SpringApplication.run(SpringBootApp.class, args);

}

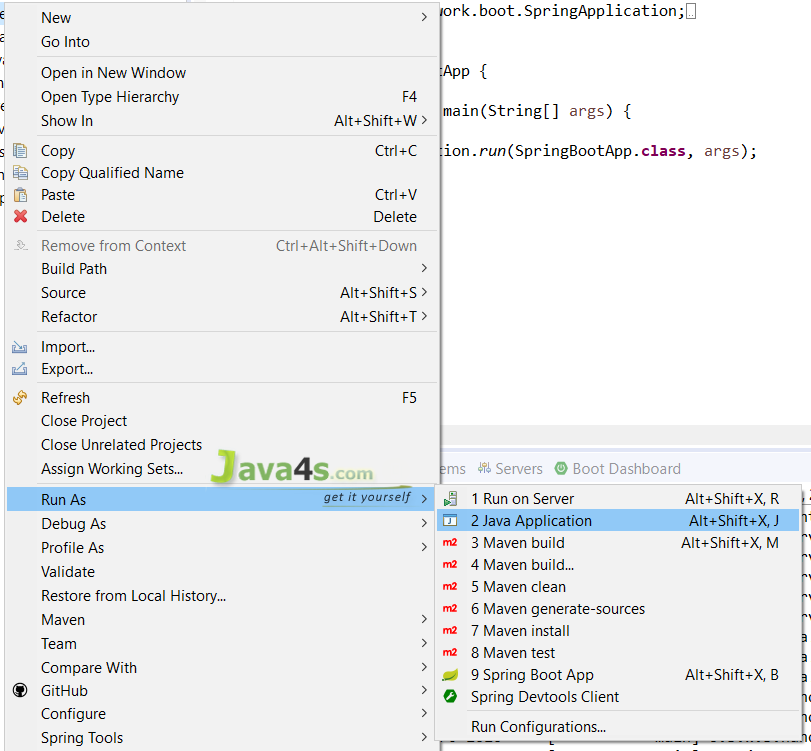
}

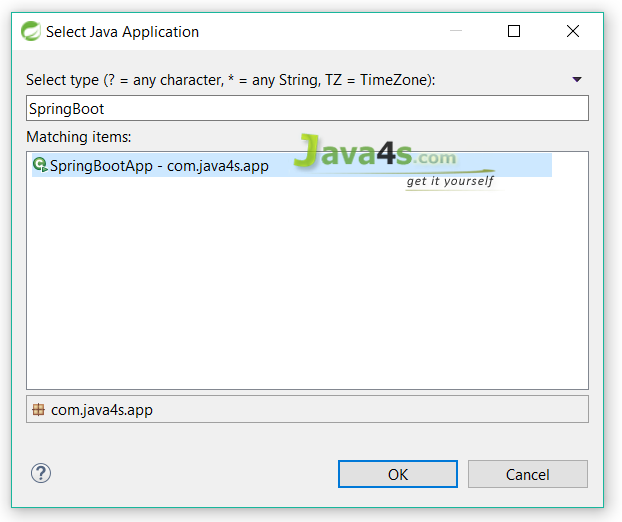
## Explanation

* In line number 6, I have added @SpringBootApplication annotation, means this is the starting point for our Spring Boot application
* In line number 11, I am bootstrapping the application

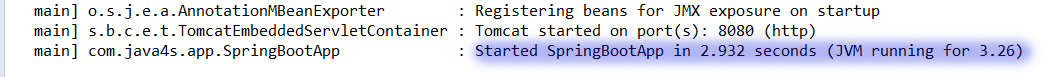
For now just remember, for every spring boot application we have to create a main class and that need to be annotate with @SpringBootApplication and bootstrap it 🙂

8. Finally, right click on the application > Run As > Java Application

[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Run-As-Java-Application.png)

9. Select our java class > ok  
[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Select-Java-Application.png)

10. Open console and check the output

[](https://www.java4s.com/wp-content/uploads/2017/08/Spring-Boot-Hello-World-Maven-Output.png)

You can check the above output, its saying Started SpringBootApp on 8080, in sometime. That’s it, we have successfully executed a spring boot application 🙂

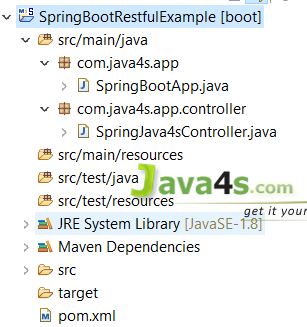
# **Spring Boot – Creating a RESTful Web Service Example**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Aug 27, 2017 [**{ 15 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/#comments) By Sivateja

In the previous article we have just created a simple hello world spring boot application, in this tutorial I am going to show you how to create a Restful web service using Spring Boot, believe me its very simple 🙂

Lets start with the directory structure of the project. (if you want to know how to create a simple spring boot project, you can go back to the previous article [Spring Boot + Maven – Hello World Example Step by Step](https://www.java4s.com/spring-boot-tutorials/spring-boot-maven-hello-world-example-step-by-step/))

## Directory Structure

[](https://www.java4s.com/wp-content/uploads/2017/08/spring-boot-restful-example.png)

## Required files

* SpringBootApp.java
* SpringJava4sController.java
* pom.xml

## pom.xml

123456789101112131415161718192021222324<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

123456789101112package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args)

{

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringJava4sController.java

123456789101112131415161718package com.java4s.app.controller;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

*@RestController*

public class SpringJava4sController {

*@RequestMapping("/")*

public String welcome() {

return "Welcome to Spring Boot Tutorials";

}

*@RequestMapping("/hello")*

public String myData() {

return "Hello Spring Boot";

}

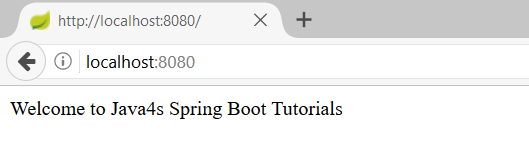
}

## Explanation

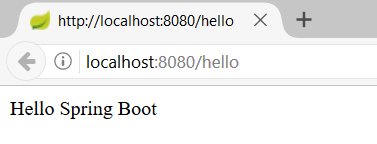
Friends, I took previous spring boot hello world application and just added the SpringJava4sController class and written RESTful web service related logic. I haven’t added any new dependencies nor written any XML’s, rather added a simple java class. My REST class is annotated with @RestController, which tells Spring Boot to consider this class as REST controller and register @RequestMapping paths inside it to respond to the HTTP requests.

**Note**: Spring Boot will use 8080 as default tomcat port

Run the application and hit http://localhost:8080  [ This is for @RequestMapping(“/”) ]

[](https://www.java4s.com/wp-content/uploads/2017/08/spring-boot-restful-web-service-example-output.PNG)

Now try hitting  http://localhost:8080/hello   [ This is for @RequestMapping(“/hello”) ]

[](https://www.java4s.com/wp-content/uploads/2017/08/spring-boot-restful-web-service-example-output-two.png)

***Note***:  Have you clearly observed the above directory structure? I have created Spring Boot main application class in com.java4s.app and controller class in com.java4s.app.controller, and in my controller class I have written my RESTful service logic and was able to execute the application successfully. How spring boot knows to scan our controller? As we have created our main class in com.java4s.app package, while starting our application, it will scan all the components under that package.  As we have created our controller class in com.java4s.app.controller which is inside com.java4s.app, our controller was registered by spring boot.

If you create the controller class outside of the main package, lets say com.java4s.controller, If you run the application it gives 404 error, just give a try and see 🙂

What’s the solution for this?  we have to add @ComponentScan annotation in our Spring Boot main class, something like this..

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

*@SpringBootApplication*

*@ComponentScan(basePackages="com.java4s.controller")*

public class SpringBootApp {

public static void main(String[] args){

---

}

}

That’s it friends, hope you enjoyed this article 🙂

# **Spring Boot – Common Application Properties (application.properties)**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Sep 3, 2017 [**{ 2 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-common-application-properties-application-properties/#comments) By Sivateja

In this article I am going to explain about Spring Boot’s application.properties.  Generally we will create property files for writing static values related to our application. If you consider some real time java applications, we will use these .property files for writing  environmental (server) related stuff and for even for some other reasons, I strongly believe you might aware of the general usage of .property files in the java applications 🙂

By default, Spring Boot provides a .properties file with name application.properties with some predefined key’s in it. I see more than 100 keys in the spring docs,  you can have a look at here..

**Spring Docs** **::.**  [Common application properties](https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html)

In the next article, I will show you how to use Spring Boot application.properties in our application with example 😉

# **Spring Boot – How to Change Default Tomcat Server Port**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Sep 3, 2017 [**{ 4 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-change-default-tomcat-server-port/#comments) By Sivateja

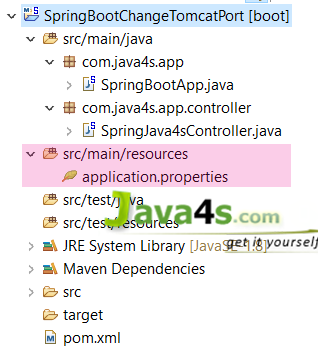
In our previous [RESTful](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/) example, when we start the application Spring Boot’s inbuilt tomcatserver by default will take 8080 as its port number, did you observe that 🙂 go back and have a look once. In this article, I am going to show you how to change that default tomcat’s port number 8080 to something else.

In Spring Boot, we can change tomcat’s port number in 2 ways…

* Using application.properties
* Using Java code change

Firstly, let me show you using application.properties. Consider the previous ‘[Creating a RESTful Web Service Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/)‘, here I just created application.properties file in src/main/resources no other changes.

## Structure looks

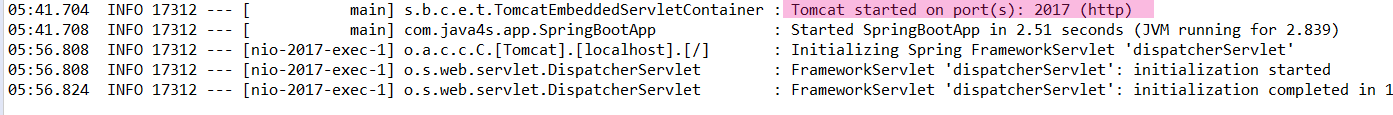
[](https://www.java4s.com/wp-content/uploads/2017/09/spring-boot-change-default-tomcat-port-using-properties.PNG)

## application.properties

server.port = 2017

A single line server.port will change the Spring Boot tomcat’s port number, if you run the application the server will takes 2017 as its port number, you can check the port in the console and can execute the application.

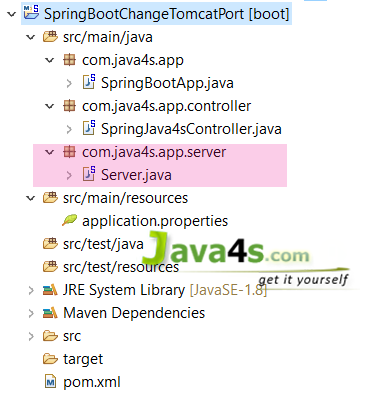
## ****Console****

[](https://www.java4s.com/wp-content/uploads/2017/09/spring-boot-change-default-tomcat-port-using-properties-console.png)

## Using Java code change

In this approach, we will create a simple java class which implements EmbeddedServletContainerCustomizer interface of Spring Boot, this is a strategy interface for customizing auto-configured embedded servlet containers, and we need to override customize() method of that interface that’s it, let me show you an example.

## Directory Structure

[](https://www.java4s.com/wp-content/uploads/2017/09/spring-boot-change-default-tomcat-port-using-java-directory-structure.png)

I have created a java class in com.java4s.app.server package with name Server.java

## Server.java

123456789101112131415package com.java4s.app.server;

import org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;

import org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;

import org.springframework.stereotype.Component;

*@Component*

public class Server implements EmbeddedServletContainerCustomizer {

*@Override*

public void customize(ConfigurableEmbeddedServletContainer container)

{

container.setPort(2018);

}

}

If you run the application, now the server will consider 2018 as its port number.

## ****Console****

[https://www.java4s.com/wp-content/uploads/2017/09/spring-boot-change-default-tomcat-port-using-java-console.png](https://www.java4s.com/wp-content/uploads/2017/09/spring-boot-change-default-tomcat-port-using-java-console.png)

**Note**: If you use both application.properties and Java config, Spring Boot will give preference for Java only, I mean it takes 2018 as its tomcat port number, you can download this example and give a try 😉

# **Spring Boot – How to Change Default Context Path**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Nov 26, 2017 [**{ 3 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-change-default-context-path/#comments) By Sivateja

Firstly what is this context path? simply its our application name. Generally while we are hittingany application in the browser, we will write the URL with the application name(context) right?

I mean…

http://localhost:<port>/***<***application\_name or context\_path***>***/operation\_name

But if you check [*Spring Boot RESTful Web Service Example*](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/) we haven’t included any context path, we directly ran the application with the path we have given in @RequestMapping, ( go back and have a look once ).

Spring Boot by default consider the context path as ‘**/**‘ so we no need to give our application name or context path, but in real-time we should use some context path for the applications. In this article I will show you how to change default spring boot application context path ‘/‘ to your application name.

In Spring Boot, we can change application default context path in two ways…

* Using applications.properties
* Using Java code changes

Its very simple just like changing tomcat port number in the previous article 🙂

## Using application.properties

Create application.properties in your application src/main/resources and write this line..

server.contextPath=/yourApplicationName

## Using Java Code Changes

123456789101112131415package com.java4s.app.server;

import org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;

import org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;

import org.springframework.stereotype.Component;

*@Component*

public class Server implements EmbeddedServletContainerCustomizer {

*@Override*

public void customize(ConfigurableEmbeddedServletContainer container)

{

container.setContextPath("/yourApplicationName");

}

}

Now we have to run the application by hitting http://localhost:8080/***yourApplicationName***/, you can download this example and give a try 😉

**Note**: If we use both java and properties file approaches, spring boot will consider java only.

# **Spring Boot – How to Reload Changes Without Restarting the Server**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Dec 8, 2017 [**{ 5 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-how-to-reload-changes-without-restarting-the-server/#comments) By Sivateja

One of the main challenge for the java developers is to deploy the apps and restart server when ever there is a code change. In this article, I am going to show you how to reload the code changes without having to restart the server. In Spring Boot this can be achieved by adding a DevTools module, just add the following dependency in your Spring Boots pom.xml and build it.

12345<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

Spring Boot DevTools module does exactly what developers needed, this eliminates the process of manually deploying the changes. DevTools will auto restart the server when we have changes. Spring team they haven’t included this feature in Spring Boot’s initial version, upon several request they added this feature later.

## Final pom.xml looks like

1234567891011121314151617181920212223242526272829<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootDevTools</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

You can start the server and run the application. Now if you do some code changes to your project files, you no need to restart the server again to view the changes rather just refresh your browser 😉 that’s it.

Hope your enjoy this article, you can download the complete example by clicking on downloadbutton.

# **Spring Boot JDBC + MySQL – How to Create/Configure a DataSource**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Jan 14, 2018 [**{ 13 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-jdbc-mysql-how-to-createconfigure-a-datasource/#comments) By Sivateja

In this article, I am going to explain you how to create/configure a datasource in Spring boot with an example. We are all aware that the process of creating a traditional Spring JDBC application is little tedious because of its XML configurations and we need to follow few steps to configure any datasource. But believe me with Spring Boot creating a JDBC application is as easy as counting 1,2,3.. 🙂

## Steps to Create DataSource in Spring Boot Application

* Add Spring Boot JDBC dependency in POM.xml
* Add datasource information in application.properties
* Get JDBCTemplate object in your DAO with @Autowired annotation and use it

…there is no point 4, that’t it 😉

## 1. Add Spring Boot JDBC Dependency

In order to work with Spring Boot JDBC, first we need to add the following dependency in your applications POM.xml

1234567891011<!-- Spring boot jdbc dependency -->

<dependency>

     <groupId>org.springframework.boot</groupId>

     <artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<!-- MySql dependency -->

<dependency>

     <groupId>mysql</groupId>

     <artifactId>mysql-connector-java</artifactId>

</dependency>

## 2. Add Datasource Information in application.properties

In real time applications, we will write the datasources information generally in the XML’s, I hope you all aware of that. In Spring Boot, rather then writing in XML’s just open your application.properties and add your datasource information to the Spring Boot’s predefined keys.

## application.properties

12345678# Applicationn context name

server.contextPath=/springbootds

# Here 'test' is the database name

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=java4s

spring.datasource.password=java4s

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

With this we have configured the datasource in our spring boot application.

## 3. Get JDBCTemplate object in your DAO with @Autowired annotation

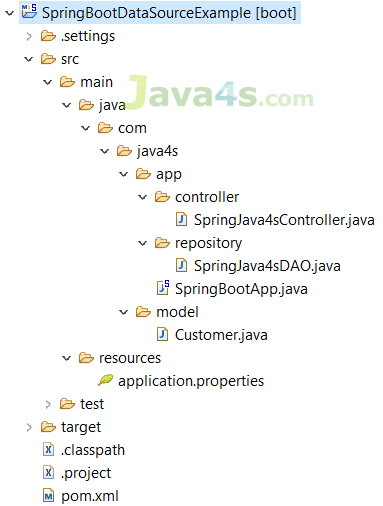
Go to your DAO class and get the object of JdbcTemplate by @Autowired annotation and use it. Spring Boot will automatically get the datasource details from application.propeties and injects to jdbcTemplate object while auto wiring.

12*@Autowired*

private JdbcTemplate jdbcTemplate;

## Spring Boot JDBC + MySQL Example – How to Create/Configure a DataSource

## Directory Structure



## Required files

* pom.xml
* SpringBootApp.java
* SpringJava4sController.java
* SpringJava4sDAO.java
* Customer.java
* application.properties

## pom.xml

1234567891011121314151617181920212223242526272829303132<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootDataSourceExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

1234567891011package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringJava4sController.java

1234567891011121314151617181920212223package com.java4s.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.app.repository.SpringJava4sDAO;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@Autowired*

public SpringJava4sDAO dao;

*@RequestMapping("/getcustInfo")*

public List<Customer> customerInformation() {

List<Customer> customers = dao.isData();

return customers;

}

}

## SpringJava4sDAO.java

1234567891011121314151617181920212223242526272829303132333435363738package com.java4s.app.repository;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

import com.java4s.model.Customer;

*@Repository*

public class SpringJava4sDAO {

*@Autowired*

private JdbcTemplate jdbcTemplate;

private static final String SQL = "select \* from customers";

public List<Customer> isData() {

List<Customer> customers = new ArrayList<Customer>();

List<Map<String, Object>> rows = jdbcTemplate.queryForList(SQL);

for (Map<String, Object> row : rows)

{

Customer customer = new Customer();

customer.setCustNo((int)row.get("Cust\_id"));

customer.setCustName((String)row.get("Cust\_name"));

customer.setCountry((String)row.get("Country"));

customers.add(customer);

}

return customers;

}

}

## Customer.java

1234567891011121314151617181920212223242526272829303132333435363738394041package com.java4s.model;

public class Customer {

private int custNo;

private String custName;

private String country;

public Customer() {

}

public Customer(int custNumber, String custName, String country) {

this.custNo = custNumber;

this.custName = custName;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

}

## application.properties

12345678# Applicationn context name

server.contextPath=/springbootds

# Here 'test' is the database name

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=java4s

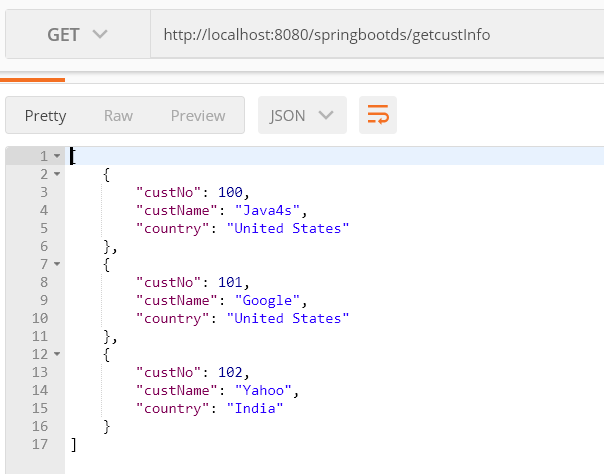
spring.datasource.password=java4s

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

## Output

In the properties file I have mentioned contextPath for the application, so the application URL will be…

http://localhost:8080/springbootds/getcustInfo

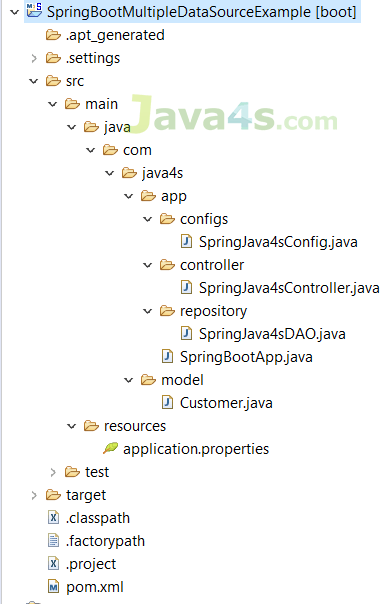


# **Spring Boot JDBC + MySQL – How to Configure Multiple DataSource**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Apr 5, 2018 [**{ 5 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-jdbc-mysql-how-to-configure-multiple-datasource/#comments) By Sivateja

In the [previous](https://www.java4s.com/spring-boot-tutorials/spring-boot-jdbc-mysql-how-to-createconfigure-a-datasource/) article we saw how to configure a datasource in a spring boot application,  that’s very straight forward.  In this article I will show you how to configure multiple datasources in spring boot application. Unlike single datasource, in order to create multiple datasources we may need to write little configuration, I will show you how.

## Directory Structure



## Add Spring Boot JDBC Dependency

123456789101112131415161718<!-- Spring boot jdbc -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<!-- MySql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!-- Configuration -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

## Final pom.xml

1234567891011121314151617181920212223242526272829303132333435<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootMultipleDataSourceExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## Add Datasources Information in application.properties

123456789101112131415# Applicationn context name

server.contextPath=/springbootds

# Here 'test' is the database name

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=java4s

spring.datasource.password=java4s

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

# Here 'test2' is the database name

spring.secondDatasource.url=jdbc:mysql://localhost/test2

spring.secondDatasource.username=java4s2

spring.secondDatasource.password=java4s2

spring.secondDatasource.driver-class-name=com.mysql.jdbc.Driver

## SpringBootApp.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringJava4sConfig.java

123456789101112131415161718192021222324252627282930313233343536373839package com.java4s.app.configs;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.jdbc.core.JdbcTemplate;

*@Configuration*

public class SpringJava4sConfig {

*@Bean*

*@Primary*

*@ConfigurationProperties(prefix = "spring.datasource")*

public DataSource firstDataSource() {

return DataSourceBuilder.create().build();

}

*@Bean*

*@ConfigurationProperties(prefix = "spring.secondDatasource")*

public DataSource secondDataSource() {

return DataSourceBuilder.create().build();

}

*@Bean*

public JdbcTemplate jdbcTemplateOne(*@Qualifier("firstDataSource") DataSource ds) {*

return new JdbcTemplate(ds);

}

*@Bean*

public JdbcTemplate jdbcTemplateTwo(*@Qualifier("secondDataSource") DataSource ds) {*

return new JdbcTemplate(ds);

}

}

## SpringJava4sController.java

12345678910111213141516171819202122232425262728package com.java4s.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.app.repository.SpringJava4sDAO;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@Autowired*

public SpringJava4sDAO dao;

*@RequestMapping("/getcustInfo")*

public List < Customer > customerInformation() {

List < Customer > customers = dao.isData();

return customers;

}

*@RequestMapping("/testSecondDatasource")*

public String dSverify() {

return dao.dsVerification();

}

}

## SpringJava4sDAO.java

12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758package com.java4s.app.repository;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

import com.java4s.model.Customer;

*@Repository*

public class SpringJava4sDAO {

*@Autowired*

private JdbcTemplate jdbcTemplateOne;

*@Autowired*

private JdbcTemplate jdbcTemplateTwo;

public List < Customer > isData() {

List < Customer > customers = new ArrayList < Customer > ();

List < Map < String, Object >> rows = jdbcTemplateOne.queryForList("SELECT \* FROM CUSTOMERS");

for (Map < String, Object > row: rows) {

Customer customer = new Customer();

customer.setCustNo((int) row.get("Cust\_id"));

customer.setCustName((String) row.get("Cust\_name"));

customer.setCountry((String) row.get("Country"));

customers.add(customer);

}

return customers;

}

public String dsVerification() {

String status = "";

String query = jdbcTemplateTwo.queryForObject("SELECT COUNT(\*) FROM DUAL;", String.class);

if (query.equals("1")) {

status = "Datasource connection successful..!";

} else {

status = "Datasource connection failed..!";

}

return status;

}

}

## Customer.java

12345678910111213141516171819202122232425262728293031323334353637383940414243package com.java4s.model;

public class Customer {

private int custNo;

private String custName;

private String country;

public Customer() {

}

public Customer(int custNumber, String custName, String country) {

this.custNo = custNumber;

this.custName = custName;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

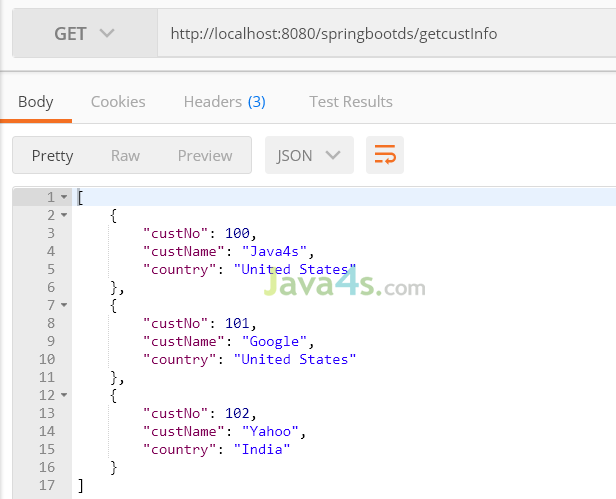
this.country = country;

}

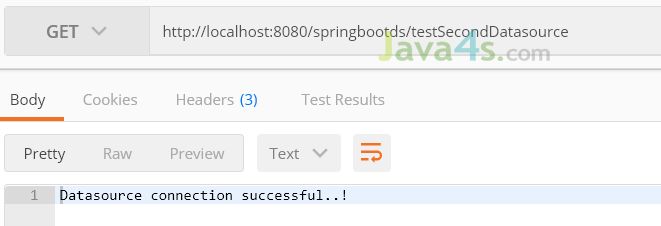
}

## Output:

If you run the app and hit the URL (this will use test database) : http://localhost:8080/springbootds/getcustInfo



And if you hit this URL (this will use test2 database) : http://localhost:8080/springbootds/testSecondDatasource



# **How to Deploy Spring Boot Applications on External Tomcat Server**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Apr 28, 2018 [**{ 12 Comments }**](https://www.java4s.com/spring-boot-tutorials/how-to-deploy-spring-boot-applications-on-external-tomcat-server/#comments) By Sivateja

So far, in previous examples we used to deploy and run the applications using embedded tomcat server provided by the spring boot. Generally in the real-time projects we wont use inbuilt servers provided by the frameworks because of many reasons like security, maintenance and control. So in this article I will show you how can we deploy the spring boot applications on external servers (in this tutorial I am going to consider the external server as Tomcat).

Just do these changes to your spring boot application which you want to deploy it on to external tomcat server.

* pom.xml, add dependency and packaging to war
* Extend your main class with SpringBootServletInitializer and override its configure method
* Generate WAR and deploy into the external server

If want you can change the context path also in the application.properties. (Optional)

## pom.xml

123456789<!-- to export as WAR -->

<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

<!-- packaging to WAR -->

<packaging>war</packaging>

## Final pom.xml

12345678910111213141516171819202122232425262728293031323334<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootAppInExternalTomcat</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- packaging to WAR -->

<packaging>war</packaging>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Instruct spring boot not to use the inbuilt Tomcat server -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## Extend main Class with SpringBootServletInitializer

12345678910111213141516171819package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.support.SpringBootServletInitializer;

*@SpringBootApplication*

public class SpringBootApp extends SpringBootServletInitializer {

*@Override*

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(SpringBootApp.class);

}

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## application.properties

1server.contextPath=/springbootapp

## SpringJava4sController.java

1234567891011121314package com.java4s.app.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

*@RestController*

public class SpringJava4sController {

*@RequestMapping("/java4s-spring-boot-ex-tomcat")*

public String customerInformation() {

return "Hey, I am from external tomcat";

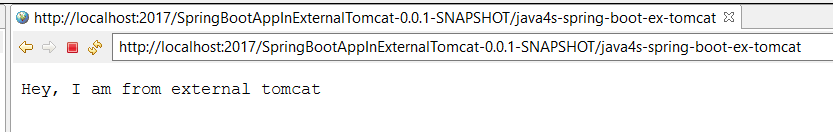
}

}

## Generate a WAR file

Right click on pom.xml -> Run as -> Maven install, this will generate a WAR file in your target folder.  Just copy that WAR file into your tomcat’s webapps folder and start the server ( or ) you can deploy and test from your IDE (eclipse/sts) by importing that WAR into your work space and run that in the external tomcat [I tried importing into eclipse].

## Output

  
**Note**:  
If you observe the context path in the URL, its showing SpringBootAppInExternalTomcat-0.0.1-SNAPSHOT. Its not taking the context path we have given in the application.properties, rather its considering the context path as Artifact Id + Version from pom.xml. So if you want required context path, the only possibility is change pom.xml accordingly (if you know other ways please comment).

You can download the application and generate the WAR file with the above step and verify.

# **Spring Boot + Spring Security – RESTful Web Service with basic Authentication**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 8, 2018 [**{ 5 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-security-restful-web-service-with-basic-authentication/#comments) By Sivateja

In this article, I am going to explain you how to implement basic authentication for RESTful web services using Spring Boot and Spring Security. We will need to create a java file with spring security configurations in it, that’s it 🙂

## Required Dependency

1234<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

## Final pom.xml

1234567891011121314151617181920212223242526<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootSpringSecurityBasicAuth</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

1234567891011package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringSecurityConfig.java

12345678910111213141516171819202122232425262728package com.java4s.app.configs;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

*@Configuration*

public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

// Authentication : set user/password details and mention the role

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().passwordEncoder(org.springframework.security.crypto.password.NoOpPasswordEncoder.getInstance())

.withUser("user").password("pass").roles("USER")

.and()

.withUser("admin").password("pass").roles("USER", "ADMIN");

}

// Authorization : mention which role can access which URL

protected void configure(HttpSecurity http) throws Exception {

http.httpBasic().and().authorizeRequests()

.antMatchers("/userlogin").hasRole("USER")

.antMatchers("/adminlogin").hasRole("ADMIN")

.and()

.csrf().disable().headers().frameOptions().disable();

}

}

## SpringJava4sController.java

123456789101112131415161718192021package com.java4s.app.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

*@RestController*

public class SpringJava4sController {

*@RequestMapping("/userlogin")*

public String userValidation() {

return "User: Successfully logged in!";

}

*@RequestMapping("/adminlogin")*

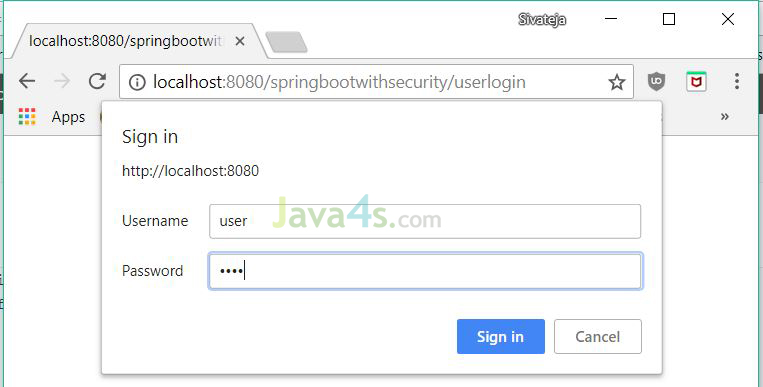
public String adminValidation() {

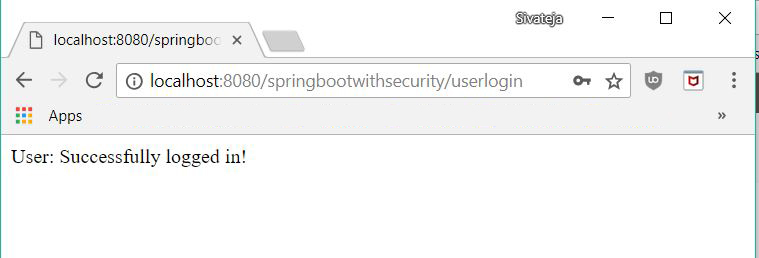
return "Admin: Successfully logged in!";

}

}

Now run the application, if you hit /springbootwithsecurity/userlogin you should provide user/passas credentials and for /springbootwithsecurity/adminlogin admin/pass, give a try 😉 I am not going to explain the SpringSecurityConfig.java as its easily understandable.

Output  




In the next article, I will explain how to implement the Authentication using database.

# **Spring Boot + Spring Security – RESTful Web Service with Database Authentication**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 17, 2018 [**{ 3 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-security-restful-web-service-with-database-authentication/#comments) By Sivateja

This article describes how to implement database authentication for your RESTful web services using Spring Boot and Spring Security. Let me start with the required dependencies..

## Dependencies

1234567891011121314151617<!-- Related to Database -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!-- Related to Database End-->

<!-- Related to Spring security -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- Related to Spring security End -->

## Final pom.xml

12345678910111213141516171819202122232425262728293031323334<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootSpringSecurityBasicAuthWithDB</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## application.properties

12345678# Applicationn context name

server.contextPath=/springbootauth

# Here 'test' is the database name

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=java4s

spring.datasource.password=java4s

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

## SpringSecurityConfig.java

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162package com.java4s.app.configs;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

*@Configuration*

public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

*@Autowired*

private DataSource ds;

*@Bean*

*@ConfigurationProperties("spring.datasource")*

public DataSource ds() {

return DataSourceBuilder.create().build();

}

/\* Spring Security Configurations Start \*/

*@Autowired*

public void configureAMBuilder(AuthenticationManagerBuilder auth) throws Exception {

auth.jdbcAuthentication().dataSource(ds)

.authoritiesByUsernameQuery("select email, role FROM USERS where email=?")

.usersByUsernameQuery("select email,userPassword, 1 FROM USERS where email=?");

}

*@Override*

protected void configure(HttpSecurity http) throws Exception {

http

.httpBasic()

.and()

.authorizeRequests()

.anyRequest().authenticated();

http.csrf().disable();

}

/\* Spring Security Configurations End \*/

}

/\*

FYI.

CREATE TABLE Users (

id int,

userName varchar(255),

email varchar(255),

userPassword varchar(255),

role varchar(10),

created timestamp

);

insert into users (id, userName, email, userPassword, role, created) values(1,'java4s', 'java4s*@java4s.com', 'java4spassword','ADMIN', CURRENT\_TIMESTAMP)*

insert into users (id, userName, email, userPassword, role, created) values(1,'siva', 'siva*@java4s.com', 'sivapassword','USER', CURRENT\_TIMESTAMP)*

\*/

## SpringJava4sController.java

123456789101112131415package com.java4s.app.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

*@RestController*

public class SpringJava4sController {

*@RequestMapping("/login")*

public String userValidation() {

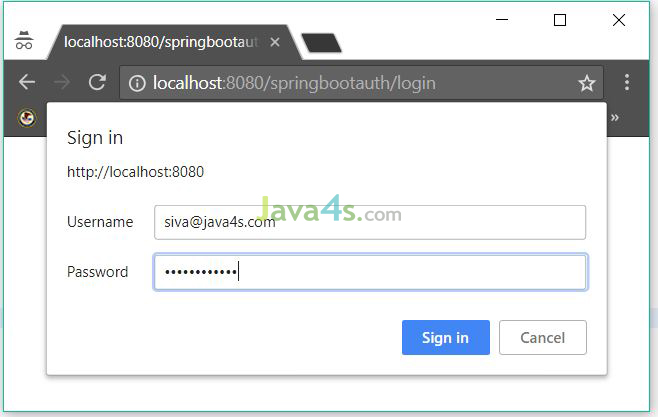
return "User: Successfully logged in!";

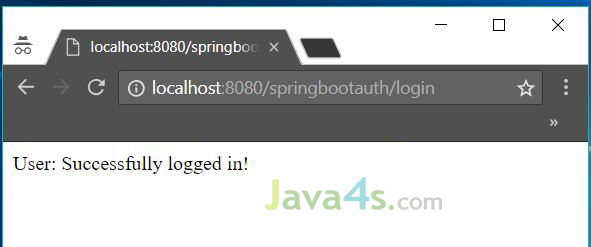
}

}

// URL: http://localhost:8080/springbootauth/login

Just start the application and hit http://localhost:8080/springbootauth/login





Friends, I haven’t explained any thing as its pretty straight forward 🙂 feel free to comment if you have any questions, I will get back to you as soon as I can.

**Note**: We can control the login using the user roles as well which I haven’t shown in this tutorial. But I will definitely cover that area in the coming articles.

# **Spring Boot + Spring MVC + JSP Hello World Example**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 17, 2018 [**{ 4 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-spring-mvc-jsp-hello-world-example/#comments) By Sivateja

This article describes how to create a Spring MVC application using Spring Boot. As this is an MVC application unlike previous examples, we have to create a webapp folder under /src/main(src > main > webapp) where we will place all our .jsp files.

## Dependencies

12345<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

## Final pom.xml

123456789101112131415161718192021222324252627<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootSpringMVC</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## application.properties

123456# Applicationn context name

server.contextPath=/springbootmvc

# Spring MVC related

spring.mvc.view.prefix=/WEB-INF/jsp/

spring.mvc.view.suffix=.jsp

In the current example, I am going to put all my .jsp file(s) under /WEB-INF/jsp/ folder, no need to mention ‘webapp’ in the properties file as spring boot by default consider that folder and search under webapp.

## SpringBootApp.java

1234567891011package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringJava4sController.java

1234567891011121314151617package com.java4s.app.controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.ModelAndView;

*@RestController*

public class SpringJava4sController {

*@GetMapping("/")*

public ModelAndView showLoginPage(Model model) {

model.addAttribute("message", "Welcome to Java4s Spring Boot Tutorials");

return new ModelAndView("welcomePage");

}

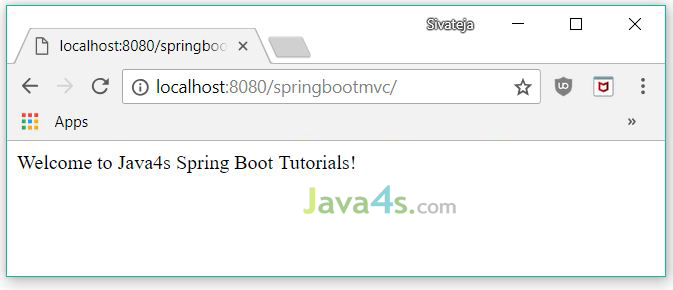
}

## welcomePage.jsp

1${message}!

## Output

Run the application and hit http://localhost:8080/springbootmvc/



# **Spring Boot – Example of RESTful Web Service with XML Response**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 23, 2018 [**{ 2 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-example-of-restful-web-service-with-xml-response/#comments) By Sivateja

Spring boot services by default gives the response in JSON format, but we can reverse this functionality in such a way that the default response will be in XML.  In order to do that we have to add a new dependency called jackson-dataformat-xml.  With this dependency services by default gives the response in XML format and if you want to see the response in JSON, just append .json to the URL that’s it 🙂 I will show you with an example.

## Dependency

1234<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

## Final pom.xml

12345678910111213141516171819202122232425262728<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootRestfulXML</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- xml dependency-->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

<!-- xml dependency end-->

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## application.properties

1server.contextPath=/springbootrestxml

## SpringJava4sController.java

123456789101112131415161718192021package com.java4s.app.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@GetMapping(path = "/get-cust-info")*

public Customer customerInformation() {

Customer cust = new Customer();

cust.setCustNo(100);

cust.setName("Bank of America");

cust.setCountry("United States");

return cust;

}

}

## Customer.java

12345678910111213141516171819202122232425262728293031323334353637383940414243package com.java4s.model;

public class Customer {

private int custNo;

private String name;

private String country;

public Customer() {

}

public Customer(int custNumber, String name, String country) {

this.custNo = custNumber;

this.name = name;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

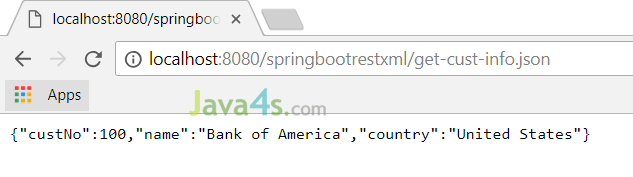
}

## Output

Run the application and hit http://localhost:8080/springbootrestxml/get-cust-info



If we want to see the response in JSON, append .json to the URL.. http://localhost:8080/springbootrestxml/get-cust-info.json



# **Spring Boot – RESTful Web Service with POST Request in JSON Example**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 24, 2018 [**{ 7 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-restful-web-service-with-post-request-in-json-format/#comments) By Sivateja

In the previous articles I didn’t get a chance to use the POST request in the examples, but this is very important. In this article I am going to show you how to create a Spring Boot REST service with POST request in JSON format with a simple example.

As Spring Boot by default supports JSON request and responses, we no need to add any dependencies. A simple annotation called @RequestBody will do the trick for us 😉

## pom.xml

12345678910111213141516171819202122<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootRestfulPostJSON</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## Customer.java

123456789101112131415161718192021222324252627282930313233343536373839404142package com.java4s.model;

public class Customer {

private int custNo;

private String name;

private String country;

public Customer() {

}

public Customer(int custNumber, String name, String country) {

this.custNo = custNumber;

this.name = name;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

}

## SpringJava4sController.java

123456789101112131415161718192021package com.java4s.app.controller;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@PostMapping(path = "/save-cust-info")*

public String customerInformation(*@RequestBody Customer cust) {*

/\* You can write your DAO logic here.

\* For time being I am printing the customer data just to show the POST call is working.

\*/

return "Customer information saved successfully ::." + cust.getCustNo() + " " + cust.getName() + " " + cust.getCountry();

}

}

## application.properties

1server.contextPath=/spring-boot-restful-post-json

## Output

Run the application and open the URL in Postman  
http://localhost:8080/spring-boot-restful-post-json/save-cust-info

Request:

12345{

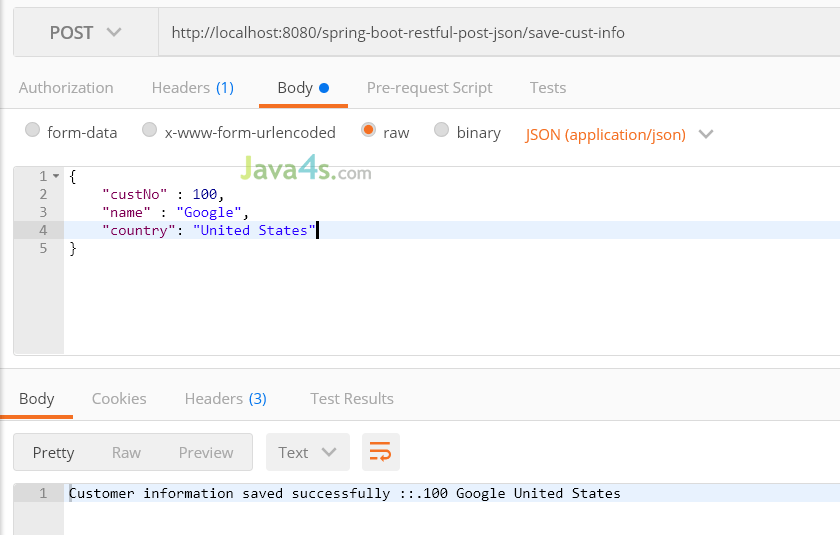
"custNo" : 100,

"name" : "Google",

"country": "United States"

}

Response:



# **Spring Boot – RESTful Web Service with POST Request in XML Example**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on May 28, 2018 [**{ 2 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-restful-web-service-with-post-request-in-xml-example/#comments) By Sivateja

In this article I will am going to show you how to read XML data from REST request using Spring Boot. As I told you in the previous articles, spring boot by default support reading and producing the JSON data. But for any XML support we have to include jackson-dataformat-xmldependency.

## xml dependency

1234<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

## Final pom.xml

12345678910111213141516171819202122232425262728<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootRestfulPostXML</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- xml dependency-->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

<!-- xml dependency end-->

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootRestfulPostXML.java

12345678910111213package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## Customer.java

12345678910111213141516171819202122232425262728293031323334353637383940414243package com.java4s.model;

public class Customer {

private int custNo;

private String name;

private String country;

public Customer() {

}

public Customer(int custNumber, String name, String country) {

this.custNo = custNumber;

this.name = name;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

}

## SpringJava4sController.java

123456789101112131415161718192021package com.java4s.app.controller;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@PostMapping(path = "/save-cust-info")*

public String customerInformation(*@RequestBody Customer cust) {*

/\* You can call your DAO logic here.

\* For time being I am printing the customer data just to show the POST call is working.

\*/

return "Customer information saved successfully ::." + cust.getCustNo() + " " + cust.getName() + " " + cust.getCountry();

}

}

## application.properties

1server.contextPath=/spring-boot-restful-post-xml

## Output

Run the application and open http://localhost:8080/spring-boot-restful-post-xml/save-cust-info in Postman.

Request:

123456<?xml version="1.0" encoding="UTF-8"?>

<customer>

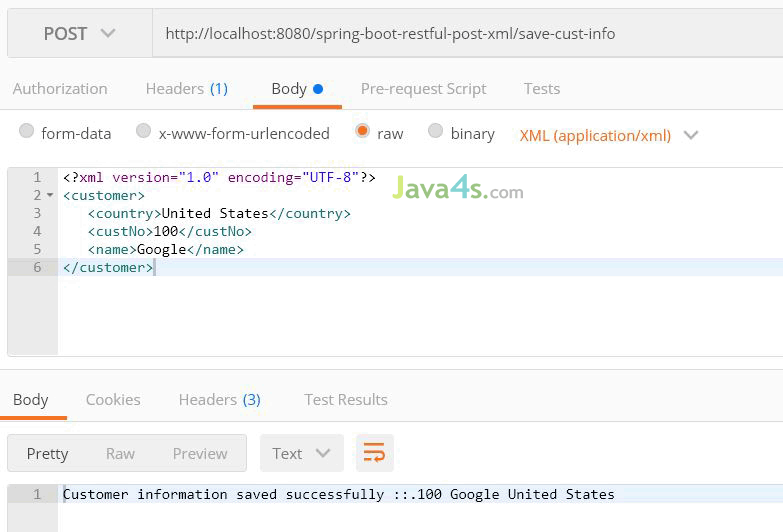
<country>United States</country>

<custNo>100</custNo>

<name>Google</name>

</customer>

**Response**:



# **Spring Boot – Display All Beans Available in ApplicationContext**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Jul 2, 2018 [**{ 2 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-display-all-beans-available-in-applicationcontext/#comments) By Sivateja

In this article, I am going to show you how to see the beans that are loaded by the Spring Boot from the ApplicationContext.  What we have to do is implement main class with CommandLineRunner/ApplicationRunner interface and override its run method.

CommandLineRunner/ApplicationRunner’s run() method will get execute right after ApplicationContext is created and before Spring Boot application initialized. This run() method will execute only once in an application’s life cycle.  So what’s the difference between CommandLineRunner/ApplicationRunner? Basically, both will do the same trick, the only difference is CommandLineRunner’s run() method will accept String array and ApplicationRunner’s run() will accept ApplicationArguments as arguments.

## SpringBootApp.java

123456789101112131415161718192021222324252627package com.java4s.app;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

*@SpringBootApplication*

public class SpringBootApp implements CommandLineRunner {

*@Autowired*

private ApplicationContext context;

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

*@Override*

public void run(String...args) throws Exception {

String[] beans = context.getBeanDefinitionNames();

for (String bean: beans) {

System.out.println(bean);

}

}

}

## Output

Run the application and you will be able to see all the beans available in the context, here I am just adding few bean names from my console 😉

12345678910111213141516171819202122...

org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$EnableWebMvcConfiguration

org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$DefaultErrorViewResolverConfiguration

requestMappingHandlerAdapter

requestMappingHandlerMapping

mvcValidator

mvcContentNegotiationManager

mvcPathMatcher

mvcUrlPathHelper

viewControllerHandlerMapping

beanNameHandlerMapping

resourceHandlerMapping

mvcResourceUrlProvider

defaultServletHandlerMapping

mvcConversionService

mvcUriComponentsContributor

httpRequestHandlerAdapter

simpleControllerHandlerAdapter

handlerExceptionResolver

mvcViewResolver

...

....

Give a try by downloading and executing this application in your local 🙂

# **How to Configure Cache in Spring Boot Applications**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Jul 29, 2018 [**{ 2 Comments }**](https://www.java4s.com/spring-boot-tutorials/how-to-configure-cache-in-spring-boot-applications/#comments) By Sivateja

In this article, I will explain step by step how to configure cache in spring boot applications.  Caching helps to increase the performance of the application by reducing number of round tripsbetween the database or any expensive resources. In real time we will face the scenarios like we have to execute heavy database query and lets say the data in the database will change very rarely, for this kind of scenarios its not a good idea to hit the database for every call, rather just cache the result at the first time when it hits the database and return the same data again for the other calls.

Steps to configure cache in spring boot applications..

* In pom.xml add spring cache dependency spring-boot-starter-cache module
* Enable cache in spring boot application by writing the @EnableCaching annotation to the main class
* Add @Cacheable annotation to the method which you would like to cache the result

## pom.xml

12345678910111213141516171819202122232425262728<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<artifactId>SpringBootCache</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

123456789101112131415package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cache.annotation.EnableCaching;

*@SpringBootApplication*

*@EnableCaching*

public class SpringBootApp {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

}

## SpringJava4sController.java

12345678910111213141516171819202122232425262728293031323334package com.java4s.app.controller;

import java.util.Arrays;

import java.util.List;

import org.springframework.cache.annotation.Cacheable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@RequestMapping("/get-cust-info")*

*@Cacheable(value="cacheCustInfo")*

public List customerInformation() {

System.out.println("I am from customerInformation");

List custDetails = Arrays.asList(

/\*

\* Here you can add your database logic/flow to get the customer details

\* For time being I am hard coding 2 values

\*/

new Customer(100,"Bank Of America","USA"),

new Customer(101,"Bank Of India","India")

);

return custDetails;

}

}

## Customer.java

12345678910111213141516171819202122232425262728293031323334353637383940414243package com.java4s.model;

public class Customer {

private int custNo;

private String name;

private String country;

public Customer() {

}

public Customer(int custNumber, String name, String country) {

this.custNo = custNumber;

this.name = name;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

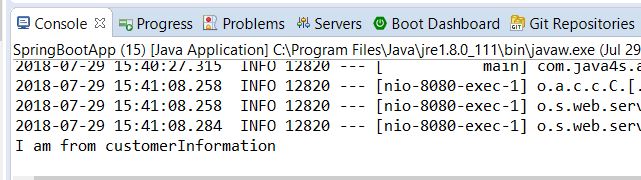
}

}

## Output

Run the application at..

http://localhost:8080/springbootcache/get-cust-info



I hit/refreshed the above URL several times and still its showing one system out message 🙂 means the cached method executing once and storing its data in the cache with key name cacheCustInfo, and giving the same data back when we make the same service call again.

**Note**: Cached data will be available until you clear it off manually, spring will not clear the cache automatically after some time. You can integrate some cache providers like EhCache, Redis..or something, so that you can get better control on the cached data.

Ref.

[*https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-caching.html*](https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-caching.html)

## Clear the Cache

Generally from our side we will clean or flush the cache after any update or delete operations, for that we use @CacheEvict annotation on the required method.

123456*@CacheEvict(value = "cacheCustInfo", allEntries=true) // @CacheEvict will clear the cache when delete any customer info from the database.*

public void removeEmployee(Id customerId) {

//Database logic will go here to remove the particular customer from the DB.

}

## How to Disable cache

If we would like to disable the cache, no need to remove all the annotations 🙂 just add the below line in the application.properties file, it takes care everything for you.

spring.cache.type=none

That’s all for now friends 🙂 hope you enjoy the article, you can download and play with the code.

# **Spring Boot Configure DataSource Using JNDI with Example**

[Spring-Boot-Tutorials](https://www.java4s.com/spring-boot-tutorials/) » on Aug 14, 2018 [**{ 5 Comments }**](https://www.java4s.com/spring-boot-tutorials/spring-boot-configure-datasource-using-jndi-with-example/#comments) By Sivateja

We already saw the default approach to configure datasource, in this article I am going to explain you how to configure datasources using JNDI lookup in spring boot applications.  Before you read this article, I highly recommend to read [How to Deploy Spring Boot Applications on External Tomcat Server](https://www.java4s.com/spring-boot-tutorials/how-to-deploy-spring-boot-applications-on-external-tomcat-server/).

## Spring Boot : Steps to Configure JNDI DataSource with External Tomcat

* Add a dependency to pom.xml to give support to our Spring Boot application to run on external servers and also add packaging war (I will explain this later 🙂 )
* Extend main class with SpringBootServletInitializer and override its configure method
* Add a property spring.datasource.jndi-name in application.properties
* Create new folder webapp/META-INF under main and add context.xml (will see this later)
* Generate a WAR and deploy into the external Tomcat Server, that’s it you are good to go 😉

Before all these, make sure you have datasource information in your external server’s server.xml 🙂 this is very very important. Add the below line in between <GlobalNamingResources/> tag.

123456789<Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"

maxActive="20"

maxIdle="0"

maxWait="10000"

name="jdbc/j4s"

password="java4s"

username="java4s"

type="javax.sql.DataSource"

url="jdbc:mysql://localhost/test"/>

I have given my local details, just change accordingly, mainly you need to change username, password and url.

## pom.xml

1234567891011121314151617181920212223242526272829303132333435363738<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.java4s</groupId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<artifactId>SpringBootDataSourceConfigJNDILookUp</artifactId>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.6.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

## SpringBootApp.java

12345678910111213141516171819package com.java4s.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.support.SpringBootServletInitializer;

*@SpringBootApplication*

public class SpringBootApp extends SpringBootServletInitializer {

public static void main(String[] args) {

SpringApplication.run(SpringBootApp.class, args);

}

*@Override*

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(SpringBootApp.class);

}

}

## application.properties

1spring.datasource.jndi-name=java:comp/env/jdbc/j4s

## context.xml

1234<?xml version="1.0" encoding="UTF-8"?>

<context>

<ResourceLink auth="Container" name="jdbc/j4s" global="jdbc/j4s" type="javax.sql.DataSource" />

</context>

Make sure value in global attribute should match with the name attribute’s value in server.xml’s Resource tag (read again if you didn’t understand 🙂 ), I hope you know the basic JNDI concept.

lets see other supporting files that I have used in this example.

## SpringJava4sController.java

12345678910111213141516171819202122232425package com.java4s.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.java4s.app.repository.SpringJava4sDAO;

import com.java4s.model.Customer;

*@RestController*

public class SpringJava4sController {

*@Autowired*

public SpringJava4sDAO dao;

*@RequestMapping("/get-cust-info")*

public List<Customer> customerInformation() {

List<Customer> customers = dao.isData();

return customers;

}

}

## SpringJava4sDAO.java

12345678910111213141516171819202122232425262728293031323334353637383940package com.java4s.app.repository;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

import com.java4s.model.Customer;

*@Repository*

public class SpringJava4sDAO {

*@Autowired*

private JdbcTemplate jdbcTemplate;

private static final String SQL = "select \* from customers";

public List<Customer> isData() {

List<Customer> customers = new ArrayList<Customer>();

List<Map<String, Object>> rows = jdbcTemplate.queryForList(SQL);

for (Map<String, Object> row : rows) {

Customer customer = new Customer();

customer.setCustNo((int)row.get("Cust\_id"));

customer.setCustName((String)row.get("Cust\_name"));

customer.setCountry((String)row.get("Country"));

customers.add(customer);

}

return customers;

}

}

## Customer.java

1234567891011121314151617181920212223242526272829303132333435363738394041package com.java4s.model;

public class Customer {

private int custNo;

private String custName;

private String country;

public Customer() {

}

public Customer(int custNumber, String custName, String country) {

this.custNo = custNumber;

this.custName = custName;

this.country = country;

}

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

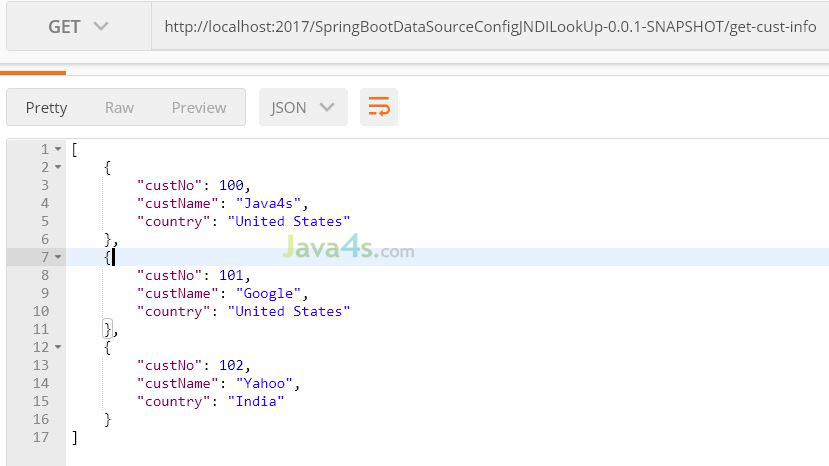
this.country = country;

}

}

Now just right click on the application > Run As > Maven install  ::. this will generate a WAR file in your applications target folder.  Now you can import that WAR and verify the changes.

## Output



You can download and play with the code 🙂

[[https://www.java4s.com/wp-content/uploads/2011/05/download.png](https://www.java4s.com/wp-content/spring-boot-source/SpringBootDataSourceConfigJNDILookUp.rar)](https://www.java4s.com/wp-content/spring-boot-source/SpringBootDataSourceConfigJNDILookUp.rar)

## Spring Boot : Steps to Configure JNDI DataSource with Embedded Tomcat

Lets compare the steps we have followed for the configuration done above for external tomcat server, for Embedded tomcat…

* No need to add any external dependency
* No need to extend main class with SpringBootServletInitializer
* No need to add spring.datasource.jndi-name in application.properties
* No need to create webapp/META-INF and context.xml
* No need to generate WAR

Rather just add one more java class which takes care of datasource and JNDI configuration 🙂 that’s it. Very easy with embedded tomcat.

## TomcatConfigs.java

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859package com.java4s.app.configs;

import javax.naming.NamingException;

import javax.sql.DataSource;

import org.apache.catalina.Context;

import org.apache.catalina.startup.Tomcat;

import org.apache.tomcat.util.descriptor.web.ContextResource;

import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainer;

import org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jndi.JndiObjectFactoryBean;

*@Configuration*

public class TomcatConfigs {

*@Bean*

public TomcatEmbeddedServletContainerFactory tomcatFactory() {

return new TomcatEmbeddedServletContainerFactory() {

*@Override*

protected TomcatEmbeddedServletContainer getTomcatEmbeddedServletContainer(Tomcat tomcat)

{

tomcat.enableNaming();

return super.getTomcatEmbeddedServletContainer(tomcat);

}

*@Override*

protected void postProcessContext(Context context)

{

ContextResource resource = new ContextResource();

resource.setType(DataSource.class.getName());

resource.setName("j4s");

resource.setProperty("factory", "org.apache.tomcat.jdbc.pool.DataSourceFactory");

resource.setProperty("driverClassName", "com.mysql.jdbc.Driver");

resource.setProperty("url", "jdbc:mysql://localhost/test");

resource.setProperty("username", "java4s");

resource.setProperty("password", "java4s");

context.getNamingResources().addResource(resource);

}

};

}

*@Bean*

public DataSource jndiDataSource() throws IllegalArgumentException, NamingException

{

JndiObjectFactoryBean bean = new JndiObjectFactoryBean();

bean.setJndiName("java:/comp/env/j4s");

bean.setProxyInterface(DataSource.class);

bean.setLookupOnStartup(false);

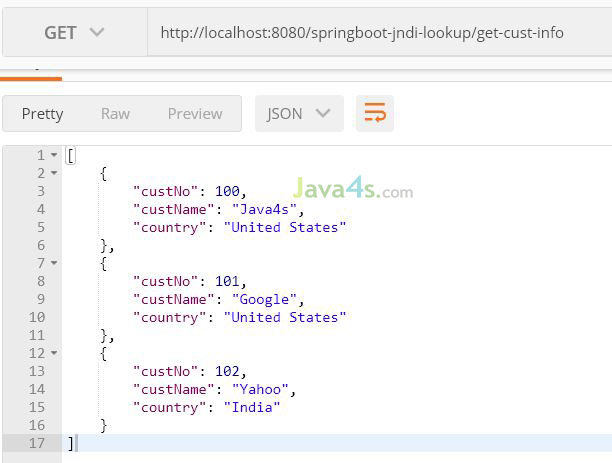
bean.afterPropertiesSet();

return (DataSource) bean.getObject();

}

}

## Output



Hope you enjoy the article and consider sharing this with your friends 🙂 BTW download the code and play with it.

[[https://www.java4s.com/wp-content/uploads/2011/05/download.png](https://www.java4s.com/wp-content/spring-boot-source/SpringBootDataSourceConfigJNDILookUpEmbTomcat.rar)](https://www.java4s.com/wp-content/spring-boot-source/SpringBootDataSourceConfigJNDILookUpEmbTomcat.rar)