# Последовательность задач для создания Web-server на Java

1. Начать рекомендую с установки Java на компьютер, как ни странно. JDK8 можно скачать здесь: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
2. Поставить Maven: <https://maven.apache.org/install.html>
3. Создать первый Spring-boot проект: <http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html>.

|  |  |  |
| --- | --- | --- |
| **11. Developing your first Spring Boot application** | | |
| [Prev](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html) | **Part II. Getting started** | [Next](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-whats-next.html) |

## 11. Developing your first Spring Boot application

Let’s develop a simple “Hello World!” web application in Java that highlights some of Spring Boot’s key features. We’ll use Maven to build this project since most IDEs support it.

|  |
| --- |
|  |
| The [spring.io](https://spring.io) web site contains many “Getting Started” guides that use Spring Boot. If you’re looking to solve a specific problem; check there first.  You can shortcut the steps below by going to [start.spring.io](https://start.spring.io) and choosing the web starter from the dependencies searcher. This will automatically generate a new project structure so that you can [start coding right away](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html#getting-started-first-application-code). Check the [documentation for more details](https://github.com/spring-io/initializr). |

Before we begin, open a terminal to check that you have valid versions of Java and Maven installed.

$ java -version

java version "1.7.0\_51"

Java(TM) SE Runtime Environment (build 1.7.0\_51-b13)

Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)

$ mvn -v

Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4; 2014-08-11T13:58:10-07:00)

Maven home: /Users/user/tools/apache-maven-3.1.1

Java version: 1.7.0\_51, vendor: Oracle Corporation

|  |
| --- |
|  |
| This sample needs to be created in its own folder. Subsequent instructions assume that you have created a suitable folder and that it is your “current directory”. |

## 11.1 Creating the POM

We need to start by creating a Maven pom.xml file. The pom.xml is the recipe that will be used to build your project. Open your favorite text editor and add the following:

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

<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.example</groupId>

<artifactId>myproject</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>

<!-- Additional lines to be added here... -->

</project>

This should give you a working build, you can test it out by running mvn package (you can ignore the “jar will be empty - no content was marked for inclusion!” warning for now).

|  |
| --- |
|  |
| At this point you could import the project into an IDE (most modern Java IDE’s include built-in support for Maven). For simplicity, we will continue to use a plain text editor for this example. |

## 11.2 Adding classpath dependencies

Spring Boot provides a number of “Starters” that make easy to add jars to your classpath. Our sample application has already used spring-boot-starter-parent in the parent section of the POM. The spring-boot-starter-parent is a special starter that provides useful Maven defaults. It also provides a [dependency-management](http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-dependency-management) section so that you can omit version tags for “blessed” dependencies.

Other “Starters” simply provide dependencies that you are likely to need when developing a specific type of application. Since we are developing a web application, we will add a spring-boot-starter-web dependency — but before that, let’s look at what we currently have.

$ mvn dependency:tree

[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT

The mvn dependency:tree command prints a tree representation of your project dependencies. You can see that spring-boot-starter-parent provides no dependencies by itself. Let’s edit our pom.xml and add the spring-boot-starter-web dependency just below the parent section:

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

If you run mvn dependency:tree again, you will see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

## 11.3 Writing the code

To finish our application we need to create a single Java file. Maven will compile sources from src/main/java by default so you need to create that folder structure, then add a file named src/main/java/Example.java:

import org.springframework.boot.\*;

import org.springframework.boot.autoconfigure.\*;

import org.springframework.stereotype.\*;

import org.springframework.web.bind.annotation.\*;

*@RestController*

*@EnableAutoConfiguration*

public class Example {

*@RequestMapping("/")*

String home() {

return "Hello World!";

}

public static void main(String[] args) throws Exception {

SpringApplication.run(Example.class, args);

}

}

Although there isn’t much code here, quite a lot is going on. Let’s step through the important parts.

### 11.3.1 The @RestController and @RequestMapping annotations

The first annotation on our Example class is @RestController. This is known as a stereotype annotation. It provides hints for people reading the code, and for Spring, that the class plays a specific role. In this case, our class is a web @Controller so Spring will consider it when handling incoming web requests.

The @RequestMapping annotation provides “routing” information. It is telling Spring that any HTTP request with the path “/” should be mapped to the home method. The @RestController annotation tells Spring to render the resulting string directly back to the caller.

|  |
| --- |
|  |
| The @RestController and @RequestMapping annotations are Spring MVC annotations (they are not specific to Spring Boot). See the [MVC section](http://docs.spring.io/spring/docs/4.3.10.RELEASE/spring-framework-reference/htmlsingle#mvc) in the Spring Reference Documentation for more details. |

### 11.3.2 The @EnableAutoConfiguration annotation

The second class-level annotation is @EnableAutoConfiguration. This annotation tells Spring Boot to “guess” how you will want to configure Spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration will assume that you are developing a web application and setup Spring accordingly.

**Starters and Auto-Configuration**

Auto-configuration is designed to work well with “Starters”, but the two concepts are not directly tied. You are free to pick-and-choose jar dependencies outside of the starters and Spring Boot will still do its best to auto-configure your application.

### 11.3.3 The “main” method

The final part of our application is the main method. This is just a standard method that follows the Java convention for an application entry point. Our main method delegates to Spring Boot’s SpringApplication class by calling run. SpringApplication will bootstrap our application, starting Spring which will in turn start the auto-configured Tomcat web server. We need to pass Example.class as an argument to the run method to tell SpringApplication which is the primary Spring component. The args array is also passed through to expose any command-line arguments.

## 11.4 Running the example

At this point our application should work. Since we have used the spring-boot-starter-parent POM we have a useful run goal that we can use to start the application. Type mvn spring-boot:run from the root project directory to start the application:

$ mvn spring-boot:run

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v1.5.6.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.222 seconds (JVM running for 6.514)

If you open a web browser to [localhost:8080](http://localhost:8080) you should see the following output:

Hello World!

To gracefully exit the application hit ctrl-c.

## 11.5 Creating an executable jar

Let’s finish our example by creating a completely self-contained executable jar file that we could run in production. Executable jars (sometimes called “fat jars”) are archives containing your compiled classes along with all of the jar dependencies that your code needs to run.

**Executable jars and Java**

Java does not provide any standard way to load nested jar files (i.e. jar files that are themselves contained within a jar). This can be problematic if you are looking to distribute a self-contained application.

To solve this problem, many developers use “uber” jars. An uber jar simply packages all classes, from all jars, into a single archive. The problem with this approach is that it becomes hard to see which libraries you are actually using in your application. It can also be problematic if the same filename is used (but with different content) in multiple jars.

Spring Boot takes a [different approach](http://docs.spring.io/spring-boot/docs/current/reference/html/executable-jar.html) and allows you to actually nest jars directly.

To create an executable jar we need to add the spring-boot-maven-plugin to our pom.xml. Insert the following lines just below the dependencies section:

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

|  |
| --- |
|  |
| The spring-boot-starter-parent POM includes <executions> configuration to bind the repackage goal. If you are not using the parent POM you will need to declare this configuration yourself. See the [plugin documentation](http://docs.spring.io/spring-boot/docs/1.5.6.RELEASE/maven-plugin/usage.html) for details. |

Save your pom.xml and run mvn package from the command line:

$ mvn package

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building myproject 0.0.1-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO] .... ..

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myproject ---

[INFO] Building jar: /Users/developer/example/spring-boot-example/target/myproject-0.0.1-SNAPSHOT.jar

[INFO]

[INFO] --- spring-boot-maven-plugin:1.5.6.RELEASE:repackage (default) @ myproject ---

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

If you look in the target directory you should see myproject-0.0.1-SNAPSHOT.jar. The file should be around 10 MB in size. If you want to peek inside, you can use jar tvf:

$ jar tvf target/myproject-0.0.1-SNAPSHOT.jar

You should also see a much smaller file named myproject-0.0.1-SNAPSHOT.jar.original in the target directory. This is the original jar file that Maven created before it was repackaged by Spring Boot.

To run that application, use the java -jar command:

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v1.5.6.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.536 seconds (JVM running for 2.864)

As before, to gracefully exit the application hit ctrl-c.

|  |  |  |
| --- | --- | --- |
| [Prev](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html) | [Up](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html) | [Next](http://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-whats-next.html) |
| 10. Installing Spring Boot | [Home](http://docs.spring.io/spring-boot/docs/current/reference/html/index.html) | 12. What to read next |

# Adding Classes to the JAR File's Classpath

You may need to reference classes in other JAR files from within a JAR file.

For example, in a typical situation an applet is bundled in a JAR file whose manifest references a different JAR file (or several different JAR files) that serves as utilities for the purposes of that applet.

You specify classes to include in the Class-Path header field in the manifest file of an applet or application. The Class-Path header takes the following form:

Class-Path: jar1-name jar2-name directory-name/jar3-name

By using the Class-Path header in the manifest, you can avoid having to specify a long -classpath flag when invoking Java to run the your application.

**Note:** The Class-Path header points to classes or JAR files on the local network, not JAR files within the JAR file or classes accessible over Internet protocols. To load classes in JAR files within a JAR file into the class path, you must write custom code to load those classes. For example, if MyJar.jar contains another JAR file called MyUtils.jar, you cannot use the Class-Path header in MyJar.jar's manifest to load classes in MyUtils.jar into the class path.

## An Example

We want to load classes in MyUtils.jar into the class path for use in MyJar.jar. These two JAR files are in the same directory.

We first create a text file named Manifest.txt with the following contents:

Class-Path: MyUtils.jar

**Warning:** The text file must end with a new line or carriage return. The last line will not be parsed properly if it does not end with a new line or carriage return.

We then create a JAR file named MyJar.jar by entering the following command:

jar cfm MyJar.jar Manifest.txt MyPackage/\*.class

This creates the JAR file with a manifest with the following contents:

Manifest-Version: 1.0

Class-Path: MyUtils.jar

Created-By: 1.7.0\_06 (Oracle Corporation)

The classes in MyUtils.jar are now loaded into the class path when you run MyJar.jar.

1. Здесь описано лишь как создать “Hello, World!”.
2. Поставить БД (например, postgre, h2 или mysql)
3. Создать классы моделей. Точнее, один простейший класс.
4. Реализовать CRUD (create-read-update-delete, операции с данными)
5. Создать простейшую jsp-форму.
6. Связать форму с контроллером.

По поводу пунктов 5-8 можно почитать следующее:

* <https://www.toptal.com/spring/beginners-guide-to-mvc-with-spring-framework> – здесь есть все, но вместо добавления простейших сообщений автор предлагает добавлять сотрудников.
* <https://stormpath.com/blog/tutorial-crud-spring-boot-20-minutes> – эта статья больше ориентирована на создание API-приложения, так что все понимать в ней ни к чему. Но прочитать ее тоже будет весьма недурно.

# Необходимые знания и инструменты:

 **Collections Framework**. Вообще говоря, это Core Java, то есть не часть экосистемы как таковой. Ориентироваться в структурах данных – обязательно. Необходимо четко понимать разницу между [LinkedList](http://habrahabr.net/?habr/127864/) и [ArrayList](http://habrahabr.net/?habr/128269/). А также понимать, что такое Set и Map (а еще помнить, что Map – это не Collection). Хорошая статья про HashMap, например: <http://habrahabr.net/?habr/128017/>. Я бы даже рекомендовал еще посмотреть сюда: <http://habrahabr.net/?habr/162017/>.

 **Иерархия исключений**: <http://www.programcreek.com/2009/02/diagram-for-hierarchy-of-exception-classes/>.

 **Иерархию классов** кстати тоже неплохо представлять в общих чертах. Хотя бы помнить, что все классы неявно наследуют от Object.

 **Spring**. Это фреймворк, который сильно упрощает жизнь. Фреймворк этот слишком большой, поэтому для начала достаточно просто понимать, зачем он нужен.

 Знать, как расшифровывается **ORM** и для чего это нужно. Я бы дал такую краткую и сильно упрощенную формулировку: ORM (ладно, **Object-Relational Mapping**) – это такая штука, которая позволяет преобразовывать записи (строки) базы данных в объекты языка. С этими объектами затем можно работать так же, как и с любыми другими – проверять или менять их состояние, вызывая те или иные методы. Соответственно, все изменения объекта отразятся в БД.

 **JDBC**. Это то, что позволяет довольно сложно и непонятно работать с базой данных. Но знать полезно. Сюда же и отнесу базовое знание SQL.

 **Maven**. Определенно необходим, так как эта штука помогает собрать проект из кусочков в один исполняемый файл (грубо говоря). Есть аналоги: **Gradle**, **Ant**. Последний, на мой взгляд, довольно сложен для начала, да и популярность его на сегодня довольно мала.

 **HTML** на базовом уровне, то есть уметь красиво верстать ни к чему для начала.

 **JUnit / TestNG**, в общем – **юнит-тестирование**. Я не буду делать на этом акцент, а также не буду говорить про TDD или BDD – итак уже слишком много материалов для начала.