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 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 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. Check the documentation for more details.

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
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/2
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org
    <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` 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 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 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 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 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/myprojec
```

```
[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	Up	Next
10. Installing Spring Boot	Home	12. What to read next