Basic project structure

Theory

7 minutes reading

In this topic, you will consider the basic structure of any Spring Boot application. If you don't have such an application, just visit the Spring Initializr site and generate it with Gradle and Java or Kotlin.

§1. Gradle as a skeleton

The basic structure of a Spring Boot application depends on a build tool that we use for the project. Since we use Gradle, our project has the `build.gradle` file that describes how to build and manage the dependencies of the project.

▼ Java

```
.
├── build.gradle
├── gradle
│   └── ...
├── gradlew
├── gradlew.bat
├── HELP.md
├── settings.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── org
    │   │       └── hyperskill
    │   │           └── demo
    │   │               └── DemoApplication.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── org
                └── hyperskill
                    └── demo
                        └── DemoApplicationTests.java
```

▼ Kotlin

```
.
├── build.gradle
```

4 required topics
Dependency management
External resources
Getting started with Spring Boot
YAML
6 dependent topics
IoC Container
Getting data from REST
H2 database
Spring Boot with Kotlin
Using FreeMarker with Spring Boot
Introduction to JPA

```
├── gradle
│   └── ...
├── gradlew
├── gradlew.bat
├── HELP.md
├── settings.gradle
└── src
    ├── main
    │   ├── kotlin
    │   │   └── org
    │   │       └── hyperskill
    │   │           └── demo
    │   │               └── DemoApplication.kt
    │   └── resources
    │       └── application.properties
    └── test
        └── kotlin
            └── org
                └── hyperskill
                    └── demo
                        └── DemoApplicationTests.kt
```

There are also other Gradle-related files that you probably already know.

The initially generated Spring Boot application has several dependencies specified in the `build.gradle` file.

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

The first dependency adds the Spring Boot framework to this project, and the second one brings test libraries integrated with Spring Boot. It is enough for the simplest Spring Boot application. As you can see, no boring configurations or excessive amount of dependencies!

> These dependencies are called **starters** and there are more of them. Each such starter dependency is a group of related dependencies. Instead of specifying a lot of related dependencies and their versions (Spring approach), we can add a Spring Boot starter that contains a group of tested-for-compatibility dependencies. All the starters follow a similar naming pattern: `spring-boot-starter-*`, where * denotes a particular type of application. For example, if we want to use Spring Web for creating web apps, we can include the `spring-boot-starter-web` dependency that contains web-related dependencies, or if we want to secure our app we can add `spring-boot-starter-security` instead of a bunch of security-related dependencies. This approach greatly simplifies dependency management, is less error-prone, and speeds up the development process.

The source code is placed in the `src` directory in `main` and `test` subdirectories.

§2. The application properties

Spring Boot uses **Convention Over Configuration** approach. It means that a developer only needs to specify unconventional aspects of the application, while all other aspects work by default.

To configure some aspects of a Spring Boot application, there is an `application.properties` file located in `src/main/resources`. In a newly generated project, this file is empty, but the application still works thanks to default implicit configurations.

We will modify the properties in the next topics.

> The properties can also be stored in the YAML format within the `application.yml` file. We intend to add their examples in the future.

§3. The Application class

The entry point of our application is `DemoApplication` class located in `org.hyperskill.demo` package. This class contains the `main` method that is commonly-known among developers.

▼ Java

```java
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

▼ Kotlin

```kotlin
@SpringBootApplication
class DemoSpringApplication

fun main(args: Array<String>) {
    runApplication<DemoSpringApplication>(*args)
}
```

The presented program looks very simple, almost like a typical *hello-world* program. Inside `main`, the `SpringApplication.run()` or `runApplication()` method is invoked to launch the application with given arguments.

There is also an important annotation `@SpringBootApplication` that does all the Spring Boot magic! It makes your application work with incredible abilities: autoconfiguration, managing lifecycle of application components and a lot of other useful things that we will consider later. This annotation shows the convenient approach in Spring Boot: annotating classes and their members to get all features provided by Spring Boot.

That is all about the basic structure of Spring Boot applications. In the next topics, you will create new classes and set up configurations to develop an application that behaves as you like.

§4. Changing the default logo

As you know, when starting a Spring Boot application, you can see the default Spring logo. Let's take a look at one simple but amazing feature as a bonus: you can change the logo to any other, e.g. the logo of your company or the project. To

Table of contents:

change the logo, first, you need to create a file named `banner.txt` that contains your custom logo and then put it in the `/src/main/resources` directory (next to the `application.properties` file).

Here is our result after running an application with a custom logo:

```
 __    __   __          __    __       _______. __
|  |  |  | |  |        |  |  |  |     /       ||  |
|  |__|  | |  |        |  |__|  |    |   (----`|  |
|   __   | |  |        |   __   |     \   \    |  |
|  |  |  | |  |  __    |  |  |  | .----)   |   |__|
|__|  |__| |__| (_ )   |__|  |__| |_______/    (__)
                  |/
```

To create a stunning logo, you can use this Spring Boot Banner Generator. We hope this small bonus will help you have more fun and remember the structure of a Spring Boot project!

Report a typo

345 users liked this piece of theory. 9 didn't like it. **What about you?**

Start practicing

Unskip this topic

Comments (4) Useful links (6) Show discussion