Berner
Fachhochschule

Spring Boot

Stephan Fischli

Introduction

- Spring Boot's primary goals are:
  - faster getting-started experience for Spring development
  - opinionated view of the Spring platform and third-party libraries
  - non-functional features that are common to most projects
  - no code generation and no XML configuration
- Spring Boot can be used to create stand-alone applications or traditional WAR deployments

> *Spring Boot makes it easy to create stand-alone, production-grade Spring-based applications.*

Spring Applications

Bootstrapping

- The static `run` method of the **`SpringApplication`** class provides a convenient way to bootstrap a Spring application
  - defines an environment by loading properties and activating profiles
  - creates an application context and loads all singleton beans into it
  - triggers any commandline runners

```
public class BookstoreApplication {

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

Bootstrapping (cont.)

- The meta-annotation `@SpringBootApplication` consists of the following annotations:
  - `@Configuration` allows programmatic registration of beans
  - `@ComponentScan` enables annotation-based scanning of components
  - `@EnableAutoConfiguration` enables Spring Boot's auto-configuration

```
@SpringBootApplication
public class BookstoreApplication {

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

Auto-Configuration

- Spring Boot's auto-configuration configures an application based on the JAR dependencies
- For example, if `H2` is on the classpath, an in-memory database is configured
- Specific auto-configuration classes can be disabled using the `exclude` attribute of the `@EnableAutoConfiguration` annotation

```
@SpringBootApplication
@EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)
public class BookstoreApplication {

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

The CommandLineRunner

- If code needs to be run once, a bean can implement the `CommandLineRunner` interface
- The interface defines a `run` method, which is called before the `run` method of `SpringApplication` completes
- The command line arguments are passed as parameters

```
@Component
public class BookstoreRunner implements CommandLineRunner {
    public void run(String[] args) {
        ...
    }
}
```

Application Arguments

- If any bean needs access to the application arguments, an `ApplicationArguments` bean can be injected
- The `ApplicationArguments` interface provides access to parsed options and non-option arguments

```
@Component
public class CustomerService {

    public CustomerService(ApplicationArguments args) {
        boolean debug = args.containsOption("debug");
        List<String> files = args.getNonOptionArgs();
        ...
    }
}
```

Spring Development

Project Setup

- For the development of Spring applications, a build system such as Maven or Gradle is recommended
- Each release of Spring Boot provides a list of supported dependencies (see **Reference Guide**)
- A convenient way to create a Spring project is to use the **Spring Initializr** web tool

Maven Projects

- Maven projects can inherit from the `spring-boot-starter-parent` project which provides dependency management and some sensible configurations
  - Java 21 compiler level
  - UTF-8 source encoding
  - execution of the repackage goal
  - plugin configuration
  - resource filtering

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.7</version>
</parent>
```

Starters

- **Starters** are dependency descriptors for particular types of Spring Boot applications
- As a general purpose starter, the `spring-boot-starter` dependency can be included
- Spring Boot also provides starters to exclude or exchange specific technical facets (e.g. logging)

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
</dependencies>
```

Maven Plugin

- The Spring Boot **Maven plugin** is used to run and to package an application

```
<plugins>
    <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
</plugins>
```

Running an Application

- A Spring Boot Maven application can be run in the exploded form using the `run` goal of the Maven plugin

```
> mvn spring-boot:run
```

- The plugin also allows a Spring Boot application to be packaged as a single JAR that can be run using the `java` command

```
> java -jar target/bookstore-1.0.jar
```

- If the application starts successfully, the Spring Boot banner is displayed

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
```

```
=========|_|==============|___/=/_/_/_/
:: Spring Boot ::  (v3.5.7)
```

Startup Failure

- If an application fails to start, failure analyzers try to provide a dedicated error message and a concrete action to fix the problem

```
***************************
APPLICATION FAILED TO START
***************************

Description:
Embedded servlet container failed to start. Port 8080 was already in
use.

Action:
Identify and stop the process that's listening on port 8080 or
configure
this application to listen on another port.s
```

- The full evaluation conditions report can be displayed by enabling the debug property

```
> java -jar target/bookstore-1.0.jar --debug
```

Customizing the Banner

- By adding a `banner.txt` file to the classpath, the startup banner can be changed
- Inside the file, the following placeholders can be used:
  - `${application.title}` for the application title as declared in the manifest file
  - `${application.version}` for the application version as declared in the manifest file
  - `${spring-boot.version}` for the Spring Boot version

```
              _\|/_
              (o o)
----------oOO-(_)-OOo----------
${application.title} ${application.version}
Spring Boot ${spring-boot.version}
------------------------------
```

Developer Tools

- Spring Boot provides tools that make the application development more efficient by
  - defining property defaults (e.g. disable caching, enable debug logging)
  - automatically restarting the application whenever files on the classpath change
  - including an embedded LiveReload server that can be used to trigger a browser refresh
- The developer tools can be added by including the `spring-boot-devtools` module

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
```

```
    </dependency>
</dependencies>
```

Code Structure

- The following code stucture is recommended:
  - the main application class is located in a root package
  - a different subpackage is provided for each application domain

```
org
 └─example
    └─bookstore
       ├─BookstoreApplication.java
       ├─customer
       │   ├─Customer.java
       │   ├─CustomerController.java
       │   ├─CustomerService.java
       │   └─CustomerRepository.java
       └─order
           ├─Order.java
           ├─OrderController.java
           ├─OrderService.java
           └─OrderRepository.java
```