

互联网应用开发技术

*Web Application Development*

---

# 第7课

## WEB后端-SPRING BOOT

Episode Seven

**Spring Boot**

陈昊鹏

[chen-hp@sjtu.edu.cn](mailto:chen-hp@sjtu.edu.cn)

Web Application  
Development

- Spring
- Spring Boot
  - Develop Applications with Spring Boot
- Spring Initializr
- Spring Initializr in IntelliJ IDEA

- Spring makes building web applications fast and hassle-free.
  - By removing much of the **boilerplate code** and **configuration** associated with web development, you get a modern web programming model that streamlines the development of server-side HTML applications, REST APIs, and bidirectional, event-based systems.
  - Developer productivity
    - **Spring Boot** is the starting point of your developer experience, whatever you're building.
    - Spring Boot is designed to get you up and running as quickly as possible, with minimal upfront configuration. With its **embedded application servers**, you can be serving in seconds.
  - Battle-tested security
    - When it's time to secure your web application, **Spring Security** supports many industry-standard authentication protocols, including SAML, OAuth, and LDAP.
  - Data access made easy
    - Spring helps developers connect their web applications to a number of data stores. It supports **relational** and **non-relational databases**, **map-reduce frameworks**, and **cloud-based data services**.

- Spring Boot

- helps you to create stand-alone, production-grade Spring-based Applications that you can run.
- We take an opinionated view of the Spring platform and third-party libraries, so that you can get started with minimum fuss.
- Most Spring Boot applications need very little Spring configuration.

Build Tool	Version
Maven	3.3+
Gradle	6 (6.3 or later). 5.6.x is also supported but in a deprecated form

Name	Servlet Version
Tomcat 9.0	4.0
Jetty 9.4	3.1
Undertow 2.0	4.0

- Before we begin, open a terminal and run the following commands to ensure that you have valid versions of Java and Maven installed:

```
$ java -version java version "1.8.0_102" Java(TM) SE Runtime Environment (build 1.8.0_102-b14) Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

```
$ mvn -v Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T14:33:14-04:00) Maven home: /usr/local/Cellar/maven/3.3.9/libexec Java version: 1.8.0_102, vendor: Oracle Corporation
```

# Creating the **POM**

```
<?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 https://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>2.4.5</version>
  </parent>

  <description/>
  <developers>
    <developer/>
  </developers>
  <licenses>
    <license/>
  </licenses>
  <scm>
    <url/>
  </scm>
  <url/>

  <!-- Additional lines to be added here... -->
</project>
```

- To add the necessary dependencies,
  - edit your **pom.xml** and add the spring-boot-starter-web dependency immediately below the **parent** section:

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

```
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class Example {

    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }

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



- The `@RestController` and `@RequestMapping` Annotations
  - The first annotation on our Example class is `@RestController`. 最终会返回一个JSON对象
  - 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 considers it when handling incoming web requests.
  - The `@RequestMapping` annotation provides “routing” information.
    - It tells 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 **@EnableAutoConfiguration** Annotation
  - The second class-level annotation is **@EnableAutoConfiguration**.
  - This annotation tells Spring Boot to “guess” how you 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 assumes that **you are developing a web application and sets up 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.
  - Spring Boot still does its best to auto-configure your application.

- The “main” Method
  - The final part of our application is the **main** method.
  - This is 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** bootstraps our application, starting Spring, which, in turn, starts 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.

- Type `mvn spring-boot:run` from the root project directory to start the application.
  - You should see output similar to the following:

```
$ mvn spring-boot:run
```

```
  .
  /\ / ____' _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
( ( )\ ____| ' _ | ' _ | ' _ | ' _ | ' _ | ' _ | ' _ | ' _ | ' _ | ' _ | ' _ |
 \ \ / ____| | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ |
  ' | ____| . _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ | | _ |
=====|_|=====|_|/=/=/=/=/
:: Spring Boot :: (v2.4.5)
.....
..... (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!`

- Creating an Executable Jar

- We 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.
- To create an executable jar, we need to add the spring-boot-maven-plugin to our pom.xml.

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

- Save your pom.xml and run `mvn package` from the command line, as follows:

```
$ 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:2.4.5:repackage (default) @ myproject ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

```

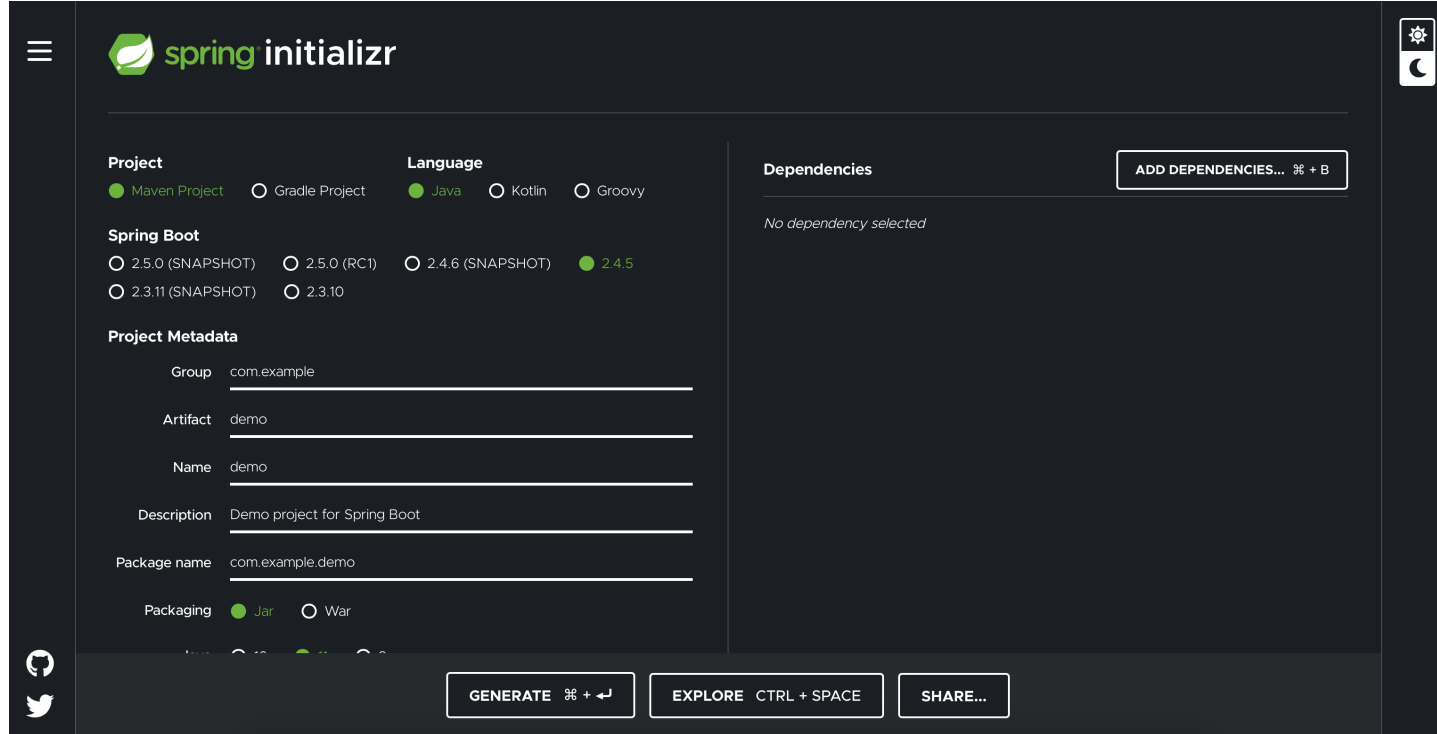
      .  _ _ _ _ _
     /\  / _ _ ' _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
    ( ( ) \ _ _ | ' _ | ' _ | | ' _ \ _ | \ \ \ \ \
     \ \ / _ _ ) | _ ) | | | | | | | ( _ | | ) ) ) )
      ' | _ _ | . _ | _ | _ | _ | _ \ _ , | / / / /
     ===== | _ | ===== | _ _ / = / _ / _ /
    :: Spring Boot :: (v2.4.5)

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

```

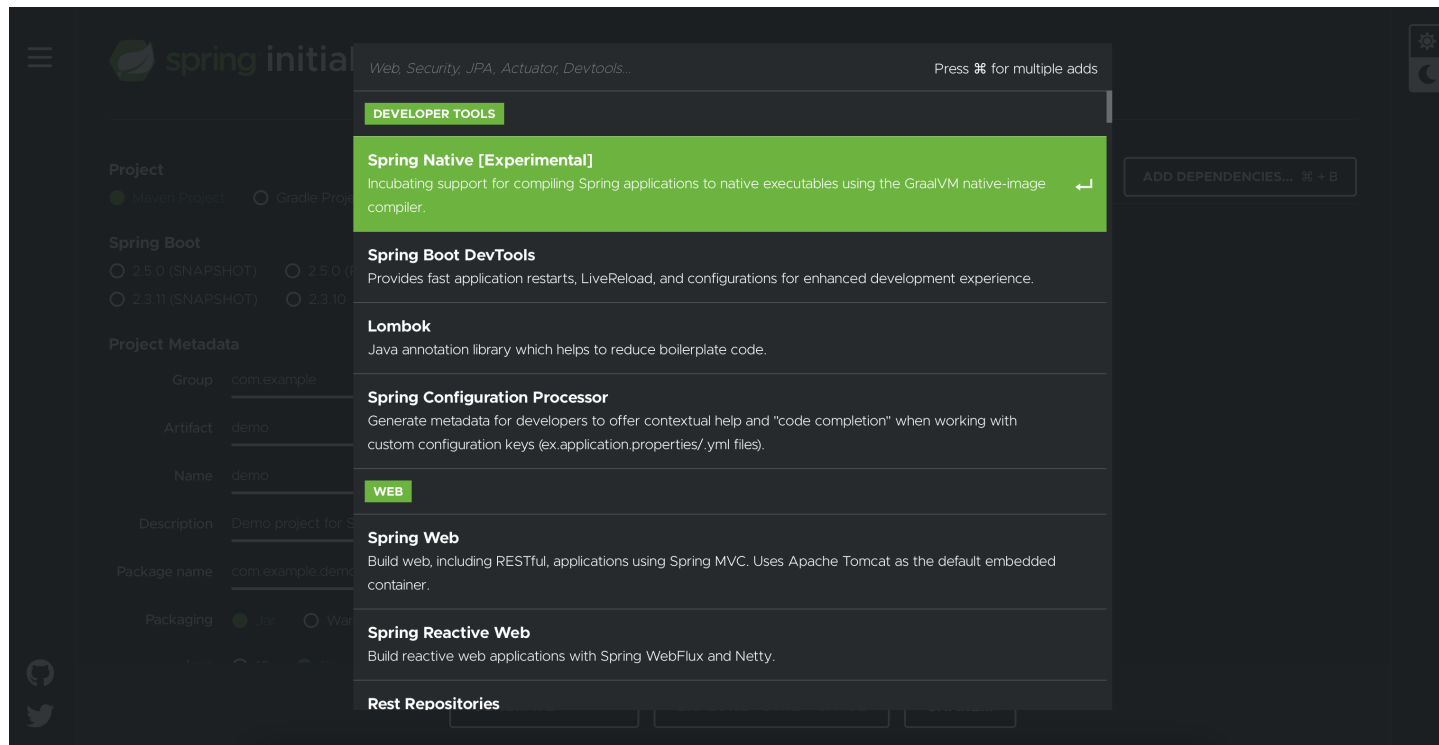
- 15

- <https://start.spring.io>

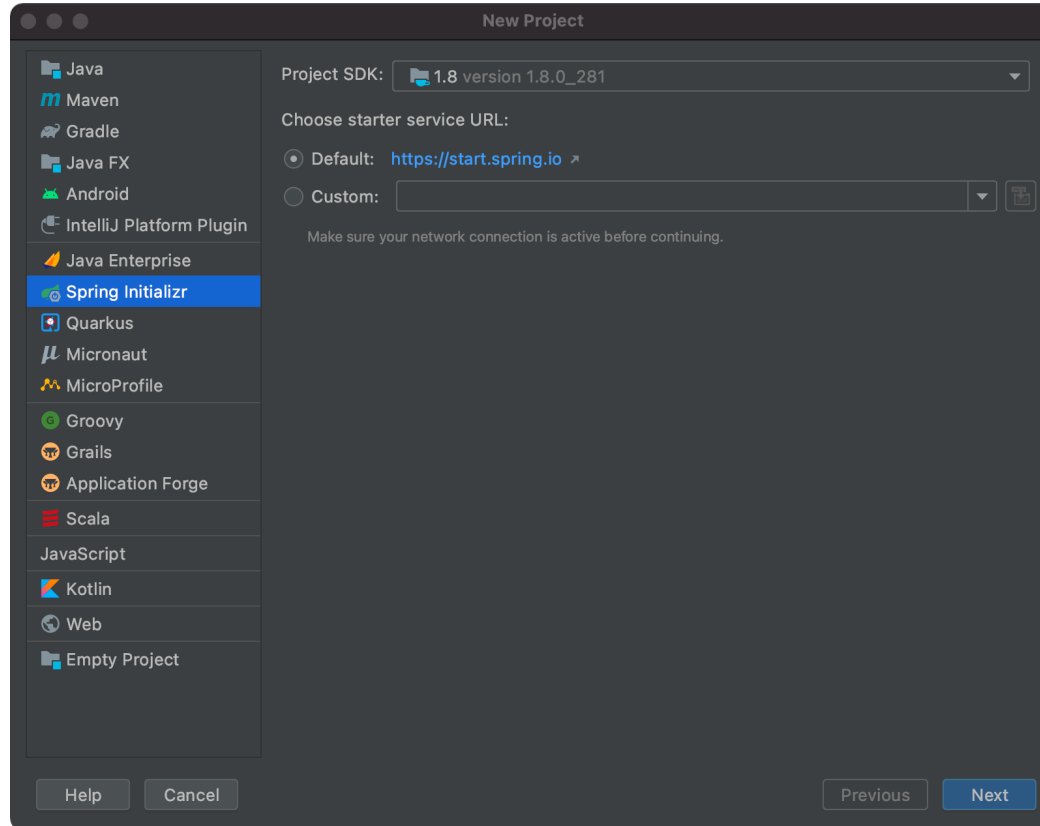
A screenshot of the Spring Initializr web application. The interface is dark-themed. At the top left is a hamburger menu icon. The main header shows the Spring Initializr logo. The form is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for versions 2.5.0 (SNAPSHOT), 2.5.0 (RC1), 2.4.6 (SNAPSHOT), 2.4.5 (selected), and 2.3.11 (SNAPSHOT); 'Project Metadata' with input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo); and 'Packaging' with radio buttons for 'Jar' (selected) and 'War'. On the right, there is a 'Dependencies' section with a button 'ADD DEPENDENCIES... ⌘ + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE ⌘ + ↵', 'EXPLORE CTRL + SPACE', and 'SHARE...'. On the far left, there are icons for GitHub and Twitter. On the far right, there are icons for settings and a dark mode toggle.



- Add Dependencies



# Spring Initializr in IntelliJ IDEA



# Spring Initializr in IntelliJ IDEA

New Project

### Spring Initializr Project Settings

Group:

Artifact:

Type: ☒ Maven ☐ Gradle

Language: ☒ Java ☐ Kotlin ☐ Groovy

Packaging: ☒ Jar ☐ War

Java version:

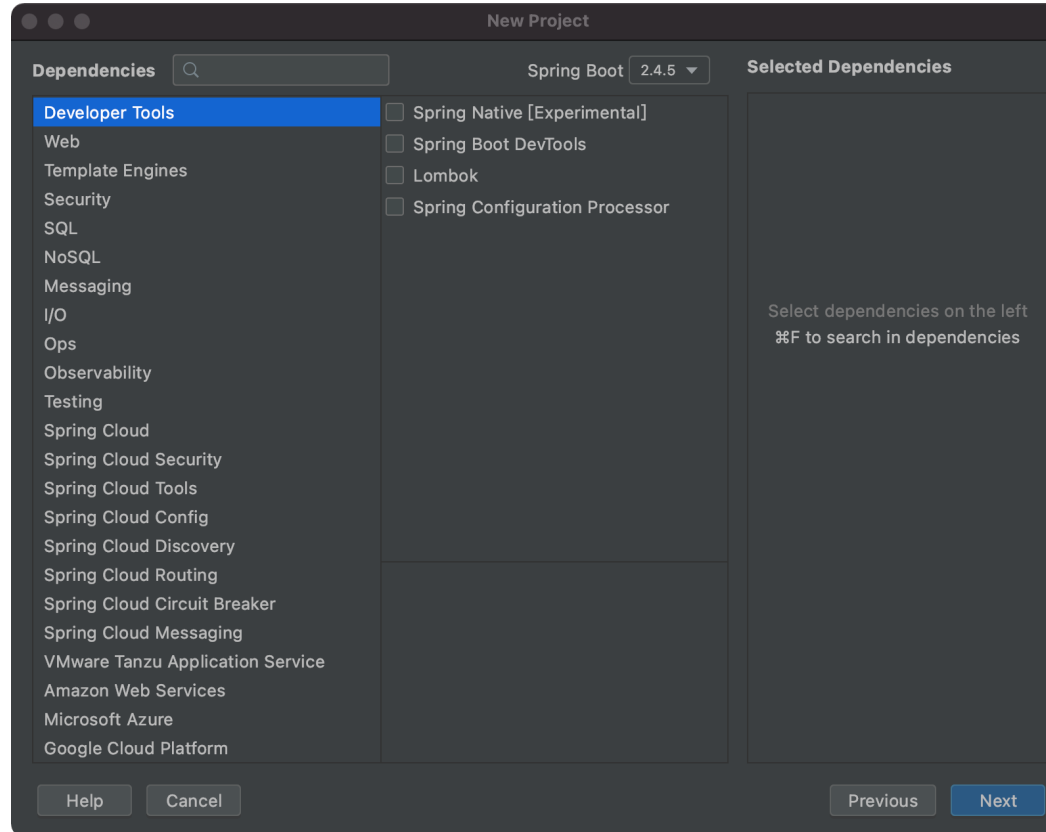
Version:

Name:

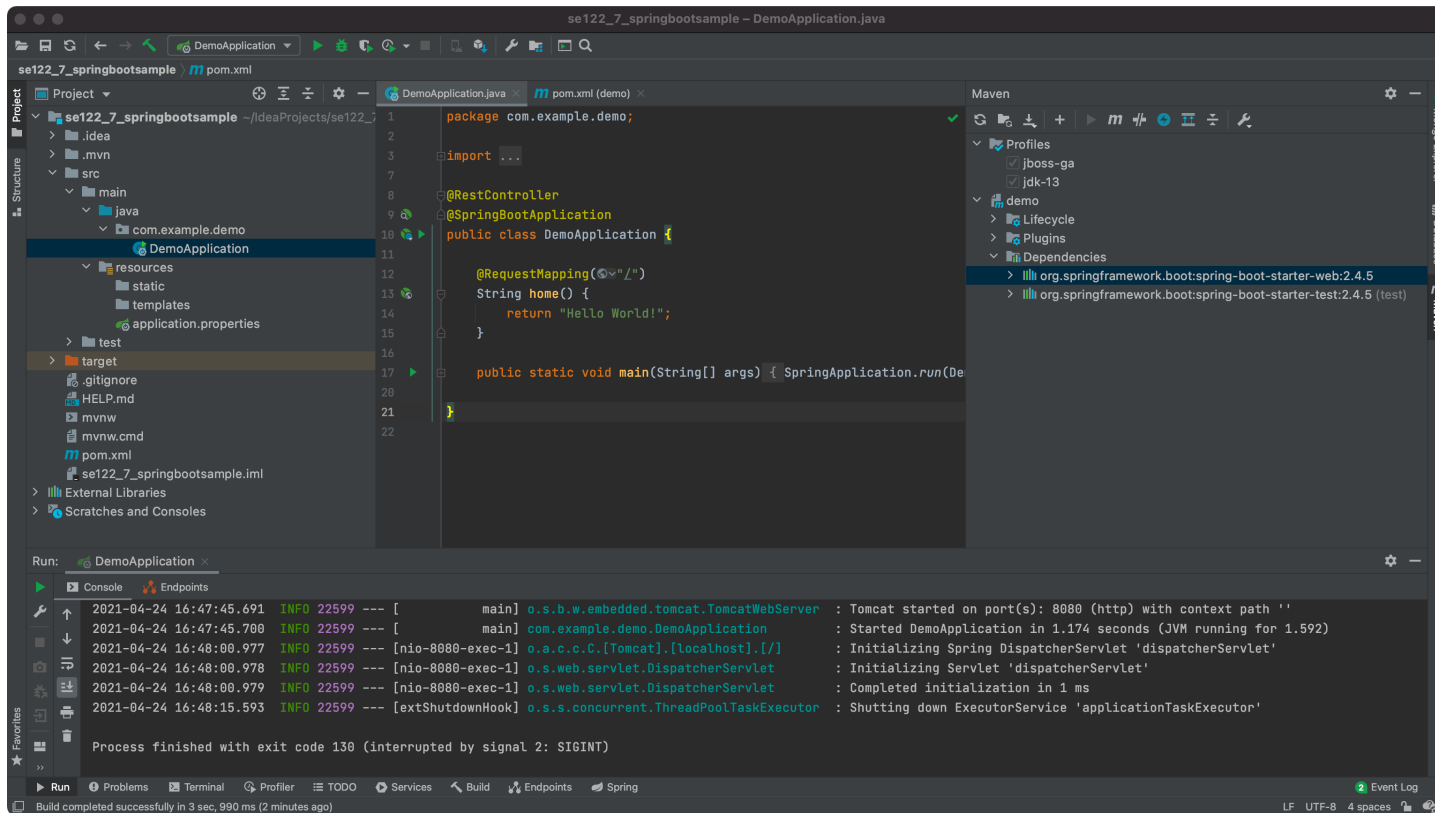
Description:

Package:

# Spring Initializr in IntelliJ IDEA



# Spring Initializr in IntelliJ IDEA



The screenshot displays the IntelliJ IDEA IDE with a Spring Boot application project named 'se122\_7\_springboot-sample'. The IDE interface includes a Project view on the left, a central code editor showing 'DemoApplication.java', a Maven view on the right, and a Run console at the bottom.

**Project View:** The project structure shows a 'main' directory containing 'java' and 'resources' subdirectories. The 'java' directory contains a package 'com.example.demo' with a class 'DemoApplication'. The 'resources' directory contains 'static', 'templates', and 'application.properties' files.

**Code Editor:** The 'DemoApplication.java' file is open, showing the following code:

```
package com.example.demo;

import ...

@RestController
@SpringBootApplication
public class DemoApplication {

    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }

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

**Maven View:** The Maven view on the right shows the project's dependencies, including 'org.springframework.boot:spring-boot-starter-web:2.4.5' and 'org.springframework.boot:spring-boot-starter-test:2.4.5 (test)'.

**Run Console:** The Run console at the bottom shows the application's execution log, indicating that the application started successfully on port 8080.

```
2021-04-24 16:47:45.691 INFO 22599 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-04-24 16:47:45.700 INFO 22599 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 1.174 seconds (JVM running for 1.592)
2021-04-24 16:48:00.977 INFO 22599 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-04-24 16:48:00.978 INFO 22599 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-04-24 16:48:00.979 INFO 22599 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2021-04-24 16:48:15.593 INFO 22599 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

- Spring MVC
  - provides an annotation-based programming model where **@Controller** and **@RestController** components use annotations to express request mappings, request input, exception handling, and more

模拟servlet

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

- Declaration

- To enable auto-detection of such **@Controller** beans, you can add component scanning to your Java configuration

```
@Configuration
@ComponentScan("org.example.web")
public class WebConfig {
    // ...
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation=" http://www.springframework.org/schema/beans
                           https://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">
  <context:component-scan base-package="org.example.web"/>
  <!-- ... -->
</beans>
```

- Request Mapping

requestmapping是对所有方法而言的

- You can use the `@RequestMapping` annotation to map requests to controllers methods.
- There are also HTTP method specific shortcut variants of `@RequestMapping`:

- `@GetMapping`
- `@PostMapping`
- `@PutMapping`
- `@DeleteMapping`
- `@PatchMapping`

```
@RestController
@RequestMapping("/persons")
class PersonController {

    @GetMapping("/{id}")
    public Person getPerson(@PathVariable Long id) {
        // ...
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void add(@RequestBody Person person) {
        // ...
    }
}
```



- Request Mapping
  - URI patterns
    - **?** matches one character
    - \* matches zero or more characters within a path segment
    - \*\* match zero or more path segments

```
@GetMapping("/owners/{ownerId}/pets/{petId}")  
public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {  
    // ...  
}
```

```
@Controller @RequestMapping("/owners/{ownerId}")  
public class OwnerController {  
    @GetMapping("/pets/{petId}")  
    public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {  
        // ...  
    }  
}
```

- **@RequestParam**

- You can use the **@RequestParam** annotation to bind Servlet request parameters (that is, query parameters or form data) to a method argument in a controller.

```
@Controller
@RequestMapping("/pets")
public class EditPetForm {
    // ...
    @GetMapping
    public String setupForm(@RequestParam("petId") int petId, Model model) {
        Pet pet = this.clinic.loadPet(petId);
        model.addAttribute("pet", pet);
        return "petForm";
    }
    // ...
}
```

- Spring Web Application
  - <https://spring.io/web-applications>
- Spring Boot Getting Started
  - <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started>
- Spring Initializr
  - <https://start.spring.io>
- Spring Annotated Controllers
  - <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#mvc-controller>



- *Web*开发技术
- *Web Application Development*

Thank You!