# Spring Boot

Introduction

* Spring Boot is an open source Java-based framework used to create a Micro Service.
* Spring Boot contains a comprehensive infrastructure support for developing a micro service and enables you to develop enterprise-ready applications that you can **“just run”**.
* You can get started with minimum configurations without the need for an entire Spring configuration setup.

Micro Service - is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.

Advantages

* To avoid complex XML configuration in Spring
* To develop a production ready Spring applications in an easier way
* It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
* It provides a powerful batch processing and manages REST endpoints.
* In Spring Boot, everything is auto configured; no manual configurations are needed.
* It offers annotation-based spring application
* Eases dependency management
* It includes Embedded Servlet Container

Annotations

|  |  |
| --- | --- |
| **@EnableAutoConfiguration** | Spring Boot automatically configures your application based on the dependencies you have added to the project |
| **@SpringBootApplication** | he entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.  **@SpringBootApplication** annotation includes Auto- Configuration, Component Scan, and Spring Boot Configuration. |
| **@ComponentScan** | Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation. |
| **@RestController** | is used to define the RESTful web services |
| **@RequestMapping** | write a Request URI method |
| @Value | is used to read the environment or application property value in Java code |
| @RequestBody | is used to define the request body content type. |
| @PathVariable | is used to define the custom or dynamic request URI. |
| @RequestParam | is used to read the request parameters from the Request URL. |
| @ControllerAdvice | is an annotation, to handle the exceptions globally. |
| @ExceptionHandler | is an annotation used to handle the specific exceptions and sending the custom responses to the client. |
| @EnableScheduling | is used to enable the scheduler for your application. |
| @Scheduled | is used to trigger the scheduler for a specific time period. |

Dependencies

Spring Boot resolves this problem by providing a set of dependencies for developer’s convenience.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** \*, where \* indicates that it is a type of the application

|  |  |
| --- | --- |
| **spring-boot-starter-data-jpa** | if you want to use Spring and JPA for database access |
| **spring-boot-starter-actuator** | used to monitor and manage your application |
| **spring-boot-starter-security** | used for Spring Security |
| **spring-boot-starter-web** | used to write a Rest Endpoints. |
| **spring-boot-starter-thymeleaf** | used to create a web application |
| **spring-boot-starter-test** | used for writing Test cases |
| **spring-boot-starter-parent** | we should inherit the Spring Boot Starter parent project to manage the Spring Boot Starters dependencies |
|  |  |

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project.

Interfaces

**SpringBootServletInitializer**  - We need to extend the class **SpringBootServletInitializer** to support WAR file deployment.

For Maven, add the start class in **pom.xml** properties as shown below −

<start-class>com.tutorialspoint.demo.DemoApplication</start-class>

## Application Runner - is an interface used to execute the code after the Spring Boot application started.

ApplicationRunner has run() with argument ApplicationArgument. It has convenient methods like getOptionNames(), getOptionValues() and getSourceArgs()

## Command Line Runner - is an interface. It is used to execute the code after the Spring Boot application started.

the only difference between CommandLineRunner and ApplicationRunner is CommandLineRunner.run() accepts String array[] whereas ApplicationRunner.run() accepts ApplicationArguments as argument.

## Dependency Management

Spring Boot team provides a list of dependencies to support the Spring Boot version for its every release. You do not need to provide a version for dependencies in the build configuration file.

## Spring Boot Active Profile

By default, application. properties will be used to run the Spring Boot application. If you want to use profile based properties, we can keep separate properties file for each profile like

**application-dev.properties**

**application-prod.properties**

The command to set the spring active profile

--spring.profiles.active=dev

**Spring Boot - Logging**

Spring Boot uses Apache Commons logging for all internal logging.

**Interceptor** to perform operations under the following situations −

* Before sending the request to the controller
* Before sending the response to the client

To work with interceptor, you need to create **@Component** class that supports it and it should implement the **HandlerInterceptor** interface.

You will have to register this Interceptor with **InterceptorRegistry** by using **WebMvcConfigurerAdapter**

The following are the three methods you should know about while working on Interceptors −

* **preHandle()** method − This is used to perform operations before sending the request to the controller. This method should return true to return the response to the client.
* **postHandle()** method − This is used to perform operations before sending the response to the client.
* **afterCompletion()** method − This is used to perform operations after completing the request and response.

**Servlet Filter**

By using filter, we can perform two operations at two instances −

* Before sending the request to the controller
* Before sending a response to the client.

**Tomcat Port Number**

If the **server.port** number is 0 while starting the Spring Boot application, Tomcat uses the random port number.