Chapter 6: Developing with Spring Boot

1. @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 considers it when handling incoming web requests.
2. @RequestMapping: The @RequestMapping annotation provides “routing” information.
3. @SpringBootApplication: The second class-level annotation is @SpringBootApplication. This annotation is known as a *meta-annotation*, it combines @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan.  
     
   Of those, the annotation we’re most interested in here is @EnableAutoConfiguration. @EnableAutoConfiguration 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**

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop shop for all the Spring and related technologies that you need without having to hunt through sample code and copy-paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, include the **spring-boot-starter-data-jpa** dependency in your project.

All **official** starters follow a similar naming pattern; spring-boot-starter-\*, where \* is a particular type of application.

Spring Boot Starter Examples

1. **spring-boot-starter-web:** Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container.
2. **spring-boot-starter-security**: Starter for using Spring Security
3. **spring-boot-starter-data-jpa**: Starter for using Spring Data JPA with Hibernate.
4. **spring-boot-starter-actuator**: Starter for using Spring Boot’s Actuator which provides production ready features to help you monitor and manage your application.

There are many other starters.

Locating the Main Application Class

We generally recommend that you locate your main application class in a root package above other classes. The @SpringBootApplication annotation is often placed on your main class, and it implicitly defines a base “search package” for certain items. For example, if you are writing a JPA application, the package of the @SpringBootApplication annotated class is used to search for @Entity items. Using a root package also allows component scan to apply only on your project.



@SpringBootApplication

public class MyApplication {

public static void main(String[] args) {

SpringApplication.run(MyApplication.class, args);

}

}

**Auto-Configuration in Spring Boot**

* The annotation **@EnableAutoConfiguration** is used to enable the auto-configuration feature.
* The @EnableAutoConfiguration annotation enables the auto-configuration of Spring ApplicationContext by scanning the classpath components and registering the beans.
* This annotation is wrapped inside the @SpringBootApplication annotation along with @ComponentScan and @SpringBootConfiguration annotations.
* When running main() method, this annotation initiates auto-configuration.

***Note****: You should use the ‘@EnableAutoConfiguration’ annotation only one time in your application.*

If you find that specific auto-configuration classes that you do not want are being applied, you can use the exclude attribute of @SpringBootApplication to disable them, as shown in the following example:

@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })

public class MyApplication {

}

**Spring Beans and Dependency Injection**

You are free to use any of the standard Spring Framework techniques to define your beans and their injected dependencies. We generally recommend using constructor injection to wire up dependencies and @ComponentScan to find beans.

If you structure your code as suggested above (locating your application class in a top package), you can add @ComponentScan without any arguments or use the @SpringBootApplication annotation which implicitly includes it. All of your application components (@Component, @Service, @Repository, @Controller, and others) are automatically registered as Spring Beans.

import org.springframework.stereotype.Service;

@Service

public class MyAccountService implements AccountService {

private final RiskAssessor riskAssessor;

public MyAccountService(RiskAssessor riskAssessor) {

this.riskAssessor = riskAssessor;

}

// NOTE: No need to use @Autowire annotation if you are having only one constructor injector

}

import java.io.PrintStream;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class MyAccountService implements AccountService {

private final RiskAssessor riskAssessor;

private final PrintStream out;

@Autowired

public MyAccountService(RiskAssessor riskAssessor) {

this.riskAssessor = riskAssessor;

this.out = System.out;

}

public MyAccountService(RiskAssessor riskAssessor, PrintStream out) {

this.riskAssessor = riskAssessor;

this.out = out;

}

// NOTE: If a bean has more than one constructor, you will need to mark the one you want Spring to use with @Autowired

}