Spring:

Spring vs Springboot

|  |  |  |
| --- | --- | --- |
| **Basis** | **Spring** | **Spring Boot** |
| Where it’s used? | Spring framework is a java EE framework that is used to build applications. | Spring Boot framework is mainly used to develop REST API’s |
| Key feature | The primary or most important feature of the Spring framework is dependency injection(Dependency Injection (DI) is a design technique that removes dependencies from computer code, making the application easier to maintain and test). | The main or primary feature of the Spring Boot is Autoconfiguration( Simply described, Spring Boot autoconfiguration is a method of automatically configuring a Spring application based on the dependencies found on the classpath.)  Autoconfiguration can speed up and simplify development by removing the need to define some beans that are part of the auto-configuration classes. |
| Why it’s used | Its goal is to make Java EE (Enterprise Edition) development easier, allowing developers to be more productive. | Spring Boot provides the RAD(Rapid Application Development) feature to the Spring framework for faster application development. |
| Type of Application Development | Spring framework helps to create a loosely coupled application. | Spring Boot helps to create a stand-alone application. |
| Servers dependency | In the Spring framework to test the Spring Project, we need to set up the servers explicitly. | Spring Boot offers built-in or embedded servers such as Tomcat and jetty. |
| Deployment descriptor | To run a Spring application a deployment descriptor is required. | In Spring Boot there is no need for the Deployment descriptor. |
| In-memory database support | Spring framework does not provide support for the in-memory database. | Spring Boot provides support for the in-memory database such as H2. |
| Boilerplate code | Spring framework requires too many lines of code (boilerplate code) even for minimal tasks. | You avoid boilerplate code which reduces time and increases productivity. |
| Configurations | In the Spring framework, you have to build configurations manually. | In Spring Boot there are default configurations that allow faster bootstrapping. |
| Dependencies | Spring Framework requires a number of dependencies to create a web app. | Spring Boot, on the other hand, can get an application working with just one dependency. There are several more dependencies required during build time that is added to the final archive by default. |
| HTTP Authentication | HTTP Basic Authentication is for enabling security confirmations, it indicates that several dependencies and configurations need to be enabled to enable security. Spring requires both the standard spring-security-web and spring-security-config dependencies to set up security in an application. Next, we need to add a class that extends the WebSecurityConfigurerAdapter and makes use of the @EnableWebSecurity annotation. | Spring Boot also requires these dependencies to make it work, but we only need to define the dependency of spring-boot-starter-security as this will automatically add all the relevant dependencies to the classpath. |
| Testing | Testing in Spring Boot is difficult in comparison to Spring Boot due to a large amount of source code. | Testing in Spring Boot is easier due to the reduced amount of source code. |
| XML Configuration | In the Spring framework, XML Configuration is required. | No need for XML configuration in Spring Boot. |
| CLI Tools | Spring framework does not provide any CLI tool for developing and testing applications. | Spring Boot provides a CLI tool for developing and testing Spring Boot applications. |
| Plugins | Spring framework does not provide any plugin for maven, Gradle, etc. like Spring Boot. | Spring Boot provides build tool plugins for Maven and Gradle. The Plugins offer a variety of features, including the packaging of executable jars. |

Springboot Feature:

1. Spring Application.:

The SpringApplication class provides a convenient way to bootstrap a Spring application that is started from a main() method. In many situations, you can delegate to the static SpringApplication.run method.

* Web Development
* SpringApplication
* Application events and listeners
* Admin features
* Externalized Configuration
* Properties Files
* YAML Support
* Type-safe Configuration
* Logging
* Security

@SpringBootApplication annotation can be used to enable those three features, that is:

* @EnableAutoConfiguration: enable [Spring Boot’s auto-configuration mechanism](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-auto-configuration.html)
* @ComponentScan: enable @Component scan on the package where the application is located (see [the best practices](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-structuring-your-code.html))
* @Configuration: allow to register extra beans in the context or import additional configuration classes

Disable Autoconfiguration Classes:

We can also disable the specific auto-configuration classes, if we do not want to be applied. We use the **exclude** attribute of the annotation @EnableAutoConfiguration to disable the auto-configuration classes.

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

 We can exclude any number of auto-configuration classes by using the property **spring.autoconfigure.exclude**.

Register a Custom Auto-Configuration:

@ConditionalOnClass(Logapi.class)

@ConditionalOnMissingBean

@ConditionalOnBean

Spring Profiles:

Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. Any @Component or @Configuration can be marked with @Profile to limit when it is loaded:

*@Configuration*

*@Profile("production")*

publicclassProductionConfiguration {

*// ...*}

In application.properties file

spring.profiles.active=dev,hsqldb

ResponseEntity:

ResponseEntity **represents the whole HTTP response: status code, headers, and body**. As a result, we can use it to fully configure the HTTP response. If we want to use it, we have to return it from the endpoint; Spring takes care of the rest. ResponseEntity is a generic type.

RequestParam vs PathVariable:

PathVariable:

**@PathVariable is used to extract data right from the URI**.

The URI Template "/owners/{ownerId}" specifies the variable name ownerId. When the controller handles this request, the value of ownerId is set to the value found in the appropriate part of the URI. For example, when a request comes in for /owners/fred, the value of ownerId is fred

RequestParam:

**The @RequestParam is used to extract query parameters**

For example, if the incoming HTTP request to retrieve a book on topic "Java" is <http://localhost:8080/shop/order/1001/receipts?date=12-05-2017>, then you can use the @RequestParam annotation to retrieve the query parameter date and you can use @PathVariable to extract the orderId i.e. "1001" as shown below:

@RequestMapping(value="/order/{orderId}/receipts", method = RequestMethod.GET) public List listUsersInvoices( @PathVariable("orderId") int order, @RequestParam(value = "date", required = false) Date dateOrNull) { ... }

What is DevTools:

With DevTools **when we make changes to Java code or properties file, the application gets updated with new changes**. It monitors for changes and automatically restarts the application. With the DevTools in the application's classpath, the application is loaded in two different class loaders in the Java Virtual Machine.

* With SpringBoot, the default embedded server is Tomcat. Other options available are **Jetty and UnderTow**.

* In pom.xml file we add depedencies for adding different server:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>*<!-- Exclude the Tomcat dependency -->*

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>*<!-- Use Jetty instead -->*

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

* Advantages of starter pom:

Starter POMs 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 technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors**

**increase pom manageability**. production-ready, tested & supported dependency configurations. decrease the overall configuration time for the project.

* Content Negotiation and how to use in Springboot:

A consumer can also send a POST request to <http://localhost:8080/students> with Content-Type header as 'application/json', and provide the JSON representation of the resource to be created. This dialog which happens between the Consumer and Service Provider is called Content Negotiation.

So content negotiation is actually a response in which we want to get response in XML or JSON format as per our requirement through content Negotiation.

we overwrite configureContentNegotiation method in our configuration class. We set the In defaultContentType(MediaType.APPLICATION\_JSON). default content type is set. It means that if we don't pass path expression then Spring will generate JSON as response. The configuration will be as follows depending on the content negotiation type

* Handle Exception in spring boot:

The **@ControllerAdvice** is an annotation, to handle the exceptions globally. Mainly used a class annotation.

Why do we need @ControllerAdvice?

@ControllerAdvice is a specialization of the @Component annotation which **allows to handle exceptions across the whole application in one global handling component**. It can be viewed as an interceptor of exceptions thrown by methods annotated with @RequestMapping and similar.

@ControllerAdvice is not specific to the exception handling , its also used for handling property, validation or formatter bindings at the global level. **@ControllerAdvice in the context of exception handling is just another way of doing exception handling at a global level using @Exceptionhandler annotation**.

The **@ExceptionHandler** is an annotation used to handle the specific exceptions and sending the custom responses to the client.

**@ExceptionHandler allows you to define a method that, as the name suggests, handle exceptions**. If you weren't using @ControllerAdvice the code for handling these exceptions would be in the controllers themselves, which could add quite a bit of duplication and clutter to the class and leading to it not being as “clean”.

* What is @qualifier used for in Spring?

By using the @Qualifier annotation, we can **eliminate the issue of which bean needs to be injected**. By including the @Qualifier annotation, together with the name of the specific implementation we want to use, in this example Foo, we can avoid ambiguity when Spring finds multiple beans of the same type

* How you document your REST APIs? => You suppose to explain about swagger

Swagger2 is an open source project used to generate the REST API documents for RESTful web services. It provides a user interface to access our RESTful web services via the web browser.

We have to add dependency:

springfox-swagger2

springfox-swagger-ui

In main method we have to add @EnableSwagger2 in class level and

@bean

public Docket productApi() {  
 return new Docket(DocumentationType.SWAGGER\_2).select()  
 .apis(RequestHandlerSelectors.basePackage("com.tutorialspoint.swaggerdemo")).build();  
 }

* What is spring boot actuator? Can we create custom endpoint and how?

Spring Boot Actuator is **a sub-project of the Spring Boot Framework**. It uses HTTP endpoints to expose operational information about any running application. The main benefit of using this library is that we get health and monitoring metrics from production-ready applications

We can create our own custom actuator endpoints using @Endpoint annotation on a class. Then we have to use @ReadOperation, @WriteOperation, or @DeleteOperation annotations on the methods to expose them as actuator endpoint bean.

@Endpoint(id="myendpoint")  
@Component  
public class MyCustomEndpoints {

@ReadOperation  
        @Bean  
        public String hi() {  
                return "Hi from custom endpoint";  
        }  
}

*We have to add property in application.properties file:*

management.endpoints.web.exposure.include=health,info,beans,env,myendpoint

* How to enable cors in spring boot?

Often, the host that serves the JS (e.g. example.com) is different from the host that serves the data (e.g. api.example.com). In such a case, CORS enables cross-domain communication.

**Enabling CORS is straightforward — just add the annotation *@CrossOrigin*.**

<https://www.baeldung.com/spring-cors> : for more details

* What is Spring Data JPA? How we can define our custom queries?

Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically

@Query is used to define our custom queries.

* What Are Possible Sources of External Configuration?

Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments. You can use **properties files, YAML files, environment variables, and command-line arguments** to externalize configuration.

* Which Is a Better Way to Configure a Spring Boot Project – Using Properties or YAML?

If you want to have multiple profiles in your Spring Boot project, by using the YAML file you just need to define one configuration file and put configuration data of specific profile into it, but if you use properties file you need to define individual file for each profile.

 You can define multiple profiles in a single YAML file by using a spring.profiles key