**Annotation**:

1. @SpringBootApplication

The main class in the Spring Boot application is annotated with @SpringBootApplication. Spring boot application is about the autoconfiguring of various resources. It does it by component scanning. By scanning classes with @Component and @Configuration annotations. @SpringBootApplication enables all of these in a single step. Under the hood it enables,

* @ComponentScan – This annotation tells the Spring Boot framework to identify all the components under the same package or all of its sub-packages. Optionally we can even specify the packages to be scanned.
* @EnableAutoConfiguration – This annotation auto-configures all the beans in the classpath. It readies the beans by initializing all the required dependencies.
* @SpringBootConfiguration – This is a class-level annotation and indicates that the class is an application configuration class. Generally, this class has the bean definitions.

2. Conditional Annotations

Conditional annotations can be used on the components. This allows us to specify whether the annotated configuration/bean/method is eligible to be registered in the container or not based on some conditions. Spring Boot takes @Conditional annotation to the next level by providing several pre-defined @Conditional\* annotations under the package org.springframework.boot.autoconfigure.conditional.

ADVERTISEMENT

* @ConditionalOnClass and @ConditionalOnMissingClass – If some class to be loaded only if some other class is available then use @ConditionalOnClass. If a class to be loaded only if another class isn’t available in ApplicationContext then use @ConditionalOnMissingClass.
* @ConditionalOnBean and @ConditionalOnMissingBean – Load the bean only if the certain bean is there in the application context or if the certain bean is missing in the application context.
* @ConditionalOnProperty – This probably the most used conditional annotation. It enables us to load certain beans only when a specific property is set in the configuration.
* @ConditionalOnResource – Load some bean only if a certain resource is present in the classpath. A useful use case is to load/enable Logging only when logback.xml is present in the classpath.
* @ConditionalOnWebApplication and @ConditionalOnNotWebApplication – Load the bean if we are running a web application or load when not a web application.
* @ConditionalExpression – This can be used to build complex rules involving multiple configurations.
* @Conditional – More generic annotation enables us to specify the conditions on Classes, Beans, Methods, and configuration.

3. Context Configuration Annotations

Context Configuration Annotations are used by the beans to set the application context during run time. @Configuration annotation is used with @ComponentScan annotation for component scanning configuration. The default configuration looks at the current folder or package and sub-packages of components. The other annotations used for scanning the components are @Component, @Controller, @Service, @BeanFactory, and @Repository . Hibernate Configuration is shown below as an example below:*@ComponentScan Annotation*

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Configuration  @ComponentScan(basePackages = {org.javacdoegeeks.orm.hibernate})  **public** **class** HibernateConfig {    } |

Configuration classes have bean definition methods that have Bean annotations.  @Configuration annotation is used by the beans for declaring the Class with bean definitions using @Bean annotation. @ComponentScan annotation is used by the bean for generating bean definitions. DataConfig class is shown as an example for @Configuration annotation usage.*@Configuration annotation*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Configuration  **public** **class** DataConfig {        @Bean  **public** DataUtils dataUtils()      {  **return** **new** DataUnits();      }  } |

@Profile annotation is used by the bean for registration. Registration can happen when there are multiple profiles. These profiles can be for dev, prod, test, and other groups. dataUtils implementation is shown as an example for handling multiple profiles – dev and prod groups.*@Profile Annotation*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | @Bean  @Profile("dev")  **public** DataUtils dataUtils()  {  **return** **new** DevDataUnits();  }    @Bean  @Profile("prod")  **public** DataUtils dataUtils()  {  **return** **new** ProdDataUnits();  } |

@Import annotation is used for importing component classes such as @Configuration and @Bean definitions. The example shown below demonstrates the usage of Import annotation.*@Import Annotation*

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Configuration  @Import({ HibernateConfig.**class**, DataConfig.**class** })  **public** **class** SystemConfig {    } |

@ImportResource annotation is used by the bean for importing resources with bean definitions. Bean definitions can be in XML. ConfigClass example shows below the usage of @ImportResource annotation.*@ImportResource Annotation*

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Configuration  @ImportResource( { "spring-context.xml" } )  **public** **class** ConfigClass {    } |

4. Basic Spring Boot Framework Annotations

Spring Boot has support for all the basic Spring annotations. Below are some of the core annotations supported in Spring/Spring Boot.

* @Component – It is a generic annotation to indicate the class is managed by Spring container
* @Bean – This is a method level annotation and indicates that the method produces a container-managed bean, this is a replacement for <bean/> tag in XML based configuration
* @Service – This annotation is a class-level annotation and indicates that the class holds the business logic
* @Configuration – The classes marked with this annotation are processed by Spring container to generate bean definitions
* @Controller – This is a specialization of @Component annotation and typically used in combination with @RequestMapping annotation. @RestController annotation simplifies the creation of the REST service.
* @RequestMapping – This annotation maps the Web/Rest API HTTP requests to the handler methods. @GetMapping, @PostMapping, @PutMapping are the special implementations of @RequestMapping.
* @Repository – Spring repository is very close to the DAO pattern and it simplifies the CRUD operation implementation
* @Autowired – This annotation allows Spring Framework to resolve and inject the dependencies. It can be used on properties, setter methods or with constructor
* @Qualifier -Used o resolve the name conflicts among the beans of the same type

5. Test Annotations

In this section, I am going to briefly discuss some of the Spring Boot test annotations. Spring boot test requires dependency spring-boot-starter-test in the maven file. This enables us to run Unit and Integration tests.

@RunWith(SpringRunner.class) provides the bridge between Spring Boot and JUnit. This is required for all Spring Boot tests. Below are some of the Spring Boot Test annotations,

* @DataJpaTest – This provides a standard environment to run persistence layer tests. It also needs a test database setup. This sets up ORM, SpringData and Datasource. It is responsible for running the Entity Scan. TestEntityManager provided by Spring Boot can be used to set up the database and the data required for the tests to run.
* @MockBean – It provides the required mocking feature to handle the dependencies.
* @TestConfiguration – Indicates that the beans created using @Bean annotation in test classes shouldn’t be picked up while scanning
* @WebMvcTest – To test Controller classes we use this annotation. It auto-configures the Spring MVC infrastructure for our tests.
* @SpringBootTest – This enables us to write integration tests. This bootstraps the entire Spring container and creates the ApplicationContext to be used in the tests.

The example provided in this article doesn’t cover the test annotations. You can write your own tests to learn more about Spring Boot Test annotations.

6. Spring Cloud Annotations

@EnableConfigServer annotation is a spring cloud annotation used by the application for accessing the configuration. You can have a java service for server configuration. HelloWorldApplication example is shown below to demonstrate @EnableConfigServer annotation.*@EnableConfigServer Annotation*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **import** org.springframework.context.annotation.\*;    @SpringBootApplication  @EnableConfigServer  **public** **class** HelloWorldApplication {  **public** **static** **void** main(String[] args) {             SpringApplication.run(HelloWorldApplication.**class**, args);     }  } |

@EnableEurekaServer annotation is used by the application for Eureka discovery. Eureka’s discovery service can be used for service location on the Spring Cloud. HelloWorldApplication the example below shows the usage of EnableEurekaServer annotation.*@EnableEurekaServer Annotation*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | **import** org.spingframework.boot.SpringApplication;  **import** org.spingframework.boot.autoconfigure.SpringBootApplication;  **import** org.spingframework.cloud.netflix.eureka.server.EnableEurekaServer;    @SpringBootApplication  @EnableEurekaServer  **public** **class** HelloWorldApplication {  **public** **static** **void** main(String[] args) {             SpringApplication.run(HelloWorldApplication.**class**, args);     }  } |

@EnableDiscoveryClient annotation was used by the application for service discovery and other microservices. @EnableCircuitBreaker annotation is another Spring Cloud annotation used for Hystrix circuit breaker protocols. This is based on the Circuit Breaker pattern. This pattern helps in functional degradation during a call failure. When one service failure, cascading failure can be detected and a recovery process can be executed. @HystrixCommand annotation is used with method name which is a fallback for handling failure.

7. Spring Boot Framework Annotations Example

This example demonstrates some of the Spring Boot annotations. Additional annotations can be tried as an example.*SpringBootDemoApplication*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @SpringBootApplication  @ComponentScan(basePackages = {"com.jcg.example.controllers", "com.jcg.example.services"})  **public** **class** SpringBootDemoApplication {  **public** **static** **void** main(String[] args) {          SpringApplication.run(SpringBootDemoApplication.**class**, args);      }  } |

@SpringBootApplication is added on the main class and it does the initial setup for Sring Boot application. @ComponentScan enables the auto-scanning of annotated classes.*HelloWorldController*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | @RestController  **public** **class** HelloWorldController {  **private** **final** HelloWorldService service;    **public** HelloWorldController(HelloWorldService service) {  **this**.service = service;      }        @GetMapping(value="/hello", produces = MediaType.TEXT\_PLAIN\_VALUE)  **public** String sayHelloWorld() {  **return** service.sayMessage();      }  } |

The above class is marked with the annotation @RestController . As we have enabled component scan to the package com.jcg.example.controllers, classes marked with Spring Boot annotations in this package are automatically detected and prepared by the container.

Here I am using constructor injection. If you want to use setter based DI then you can make use of annotation @Autowired on the bean HelloWorldService.

Below is the snippet to use one of the conditional annotations @ConditionalOnResource. MySQLDatabaseService class is dependent on mysql.properties resource. If it doesn’t find it in the classpath then the container will throw an error. To test this behavior delete mysql.properties from the resources folder in the example project and rerun.*MySQLDatabaseService*

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @ConditionalOnResource(          resources = "classpath:mysql.properties")  @Service  **public** **class** MySQLDatabaseService {      //This class is available only if mysql.properties is present in the classpath  } |

All different flavors of conditional annotations can be tried as an exercise.

8. Download the source code

That was a Spring Boot Framework Annotations Tutorial