Edition 2020

Spring and Spring Boot Annotations

Tech Bulletin Prime

By Rajanikanta Pradhan

Spring Annotations FAQ

1. @SpringBootApplication

Spring Doc for details

@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiguration

The @SpringBootApplication annotation is a combination of following three Spring annotations and provides the functionality of all three with just one line of code.

@Configuration

This annotation marks a class as a Configuration class for Java-based configuration. This is particularly important if you favour Java-based configuration over XML configuration.

@ComponentScan

This annotation enables component-scanning so that the web controller classes and other components you create will be automatically discovered and registered as beans in Spring's Application Context. All the Controller classes you write are discovered by this annotation.

@EnableAutoConfiguration

This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for you.

1. @RestController:

Spring Doc Link For details

The ***@RestController*** annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. **It's a convenience annotation that combines *@Controller* and *@ResponseBody*** – which eliminates the need to annotate every request handling method of the controller class with the ***@ResponseBody*** annotation.

2. @RestController:

@Controller Spring Doc

This is simply a specialization of the ***@Component*** class and allows implementation classes to be autodetected through the classpath scanning.

@Controller is typically used in combination with a ***@RequestMapping*** annotation used on request handling methods.

3. @RequetMapping

@RequestMapping Spring Doc

@RequestMapping is one of the most common annotation used in Spring Web applications. This annotation maps HTTP requests to handler methods of MVC and REST controllers

4. @GetMapping, @PostMapping

@GetMapping Spring Doc

@GetMapping annotation maps HTTP GET requests onto specific handler methods. It is a composed annotation that acts as a shortcut for **@RequestMapping** (method = RequestMethod.GET). In the same way it applies for rest of the annotations.

5. @PathVariable:

@Pathvariable Spring Doc

@PathVariable is a Spring annotation which indicates that a method parameter should be bound to a URI template variable. If the method parameter is Map<String, String> then the map is populated with all path variable names and values.

6. @RequestParam:

@RequestParam Spring Doc

Annotation which indicates that a method parameter should be bound to a web request parameter.

7. @RequestBody:

@RequestBody Spring Doc

Annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request.

8. @RequestHeader

@RequestHeader Spring Doc

Annotation which indicates that a method parameter should be bound to a web request header.Supported for annotated handler methods in Spring MVC and Spring WebFlux.If the method parameter is Map<String, String>, MultiValueMap<String, String>, or HttpHeaders then the map is populated with all header names and values.

9. @ResponseBody

@ResponseBody Spring Doc

@ResponseBody is a Spring annotation which binds a method return value to the web response body. It is not interpreted as a view name. It uses HTTP Message converters to convert the return value to HTTP response body, based on the content-type in the request HTTP header.

10. @Component, @Repository, @Service:

@Component Spring Doc

Indicates that an annotated class is a "component". Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

11. @Autowired:

@Autowired Spring Doc

Marks a constructor, field, setter method, or config method as to be autowired by Spring's dependency injection facilities. This is an alternative to the JSR-330 Inject annotation, adding required-vs-optional semantics.

12. @Transactional:

@Transactional Spring Doc

Describes a transaction attribute on an individual method or on a class.

At the class level, this annotation applies as a default to all methods of the declaring class and its subclasses. Note that it does not apply to ancestor classes up the class hierarchy; methods need to be locally redeclared in order to participate in a subclass-level annotation.

13. @Bean:

@Bean Spring Doc

Indicates that a method produces a bean to be managed by the Spring container.

14. @Qualifier:

@Qualifier Spring Doc

There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the **@Qualifier** annotation along with **@Autowired** to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of **@Qualifier** annotation.

15. @Primary:

@Promary Spring Doc

Indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency. If exactly one 'primary' bean exists among the candidates, it will be the autowired value.

@Primary and @Autowired Difference:

If a bean has **@Autowired** *without* any **@Qualifier**, and multiple beans of the type exist, the candidate bean marked **@Primary** will be chosen,

i.e. it is the default selection when no other information is available, i.e. when **@Qualifier** is missing.

16. @PropertySource:

@PropertySource Spring Doc

Annotation providing a convenient and declarative mechanism for adding a PropertySource to Spring's Environment. To be used in conjunction with **@Configuration** classes.

```
@PropertySource("classpath:/com/myco/app.properties")
```

17. @Value:

@Value Spring Doc

Annotation used at the field or method/constructor parameter level that indicates a default value expression for the annotated element.

18. @CrossOrigin:

@CrossOrigin Spring Doc

Annotation for permitting cross-origin requests on specific handler classes and/or handler methods. Processed if an appropriate HandlerMapping is configured.

19. @ActiveProfiles:

@ActiveProfiles Spring Doc

@ActiveProfiles is a class-level annotation that is used to declare which *active bean definition profiles* should be used when loading an ApplicationContext for test classes.