Spring Framework Annotations – 2022

**@Required**: The @Required annotation indicates that the affected bean must be populated at configuration time with the required property. **Deprecated as of Spring 5.1, use @Autowired**.

**@Autowired**: This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.

**@Qualifier**: This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used.

@Component

public class BeanB1 implements BeanInterface {

//

}

@Component

public class BeanB2 implements BeanInterface {

//

}

**@Configuration**: This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file.

**@ComponentScan**: This annotation is used with @Configuration annotation to allow Spring to know the packages to scan for annotated components.

**@Bean**: This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans.

**@Lazy**: This annotation is used on component classes. By default all autowired dependencies are created and configured at startup. But if you want to initialize a bean lazily, you can use @Lazy annotation over the class. This means that the bean will be created and initialized only when it is first requested for. **By default, Spring creates all singleton beans eagerly at the startup/bootstrapping of the application context.**

**@Value**: This annotation is used at the field, constructor parameter, and method parameter level. The @Value annotation indicates a default value expression for the field or parameter to initialize the property with. As the @Autowired annotation tells Spring to inject object into another when it loads your application context, you can also use @Value annotation to inject values from a property file into a bean’s attribute.

**Spring Framework Stereotype Annotations**

**@Component**: This annotation is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or say component so that the component-scanning mechanism of Spring can add into the application context.

**@Controller**: The @Controller annotation is used to indicate the class is a Spring controller.

**@Service**: This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs.

**@Repository**: This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.

**Spring Boot Annotations**

**@EnableAutoConfiguration**: This annotation is usually placed on the main application class. The @EnableAutoConfiguration annotation implicitly defines a base “search package”. This annotation tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

**@SpringBootApplication**: This annotation is used on the application class while setting up a Spring Boot project. The class that is annotated with the @SpringBootApplication must be kept in the base package.

The @SpringBootApplication is a convenient annotation that adds all the following:

**@Configuration**

**@EnableAutoConfiguration**

**@ComponentScan**

**Spring MVC and REST Annotations**

**@Controller**: This annotation is used on Java classes that play the role of controller in your application. The @Controller annotation allows autodetection of component classes in the classpath and auto-registering bean definitions for them.

**@RequestMapping**: This annotation is used both at class and method level. The @RequestMapping annotation is used to map web requests onto specific handler classes and handler methods

**@CookieValue**: This annotation is used at method parameter level. @CookieValue is used as argument of request mapping method.

**@CrossOrigin**: This annotation is used both at class and method level to enable cross origin requests.

**Composed @RequestMapping Variants**

**@GetMapping**: This annotation is used for mapping HTTP GET requests onto specific handler methods.

**@PostMapping**: This annotation is used for mapping HTTP POST requests onto specific handler methods.

a shortcut for @RequestMapping(method = RequestMethod.POST)

**@PutMapping**: This annotation is used for mapping HTTP PUT requests onto specific handler methods.

**@PatchMapping**: This annotation is used for mapping HTTP PATCH requests onto specific handler methods.

**@DeleteMapping**: This annotation is used for mapping HTTP DELETE requests onto specific handler methods.

**@ExceptionHandler**: This annotation is used at method levels to handle exception at the controller level.

**@InitBinder**: This annotation is a method level annotation that plays the role of identifying the methods which initialize the WebDataBinder – a DataBinder that binds the request parameter to JavaBean objects. To customize request parameter data binding , you can use @InitBinder annotated methods within our controller. It is defined in the controller, helps in controlling and formatting each and every request that comes to it. This annotation is used with the methods which initializes WebDataBinder and works as a preprocessor for each request coming to the controller.

**@Mappings and @Mapping**

This annotation is used on fields. The @Mapping annotation is a meta annotation that indicates a web mapping annotation. When mapping different field names, you need to configure the source field to its target field and to do that you have to add the @Mappings annotation.

**@MatrixVariable**: This annotation is used to annotate request handler method arguments so that Spring can inject the relevant bits of matrix URI.

**@PathVariable**: This annotation is used to annotate request handler method arguments.

**@RequestAttribute**: This annotation is used to bind the request attribute to a handler method parameter.

**@RequestBody**: This annotation is used to annotate request handler method arguments. The @RequestBody annotation indicates that a method parameter should be bound to the value of the HTTP request body.

**@RequestHeader**: This annotation is used to annotate request handler method arguments. The @RequestHeader annotation is used to map controller parameter to request header value.

**@RequestParam**: This annotation is used to annotate request handler method arguments.

**@RequestPart**: This annotation is used to annotate request handler method arguments. The @RequestPart annotation can be used instead of @RequestParam to get the content of a specific multipart and bind to the method argument annotated with @RequestPart. This annotation takes into consideration the “Content-Type” header in the multipart(request part).

**@ResponseBody**: This annotation is used to annotate request handler methods. The @ResponseBody annotation is similar to the @RequestBody annotation. The @ResponseBody annotation indicates that the result type should be written straight in the response body in whatever format you specify like JSON or XML.

**@ResponseStatus**: This annotation is used on methods and exception classes. @ResponseStatus marks a method or exception class with a status code and a reason that must be returned.

**@ControllerAdvice**: This annotation is applied at the class level. As explained earlier, for each controller you can use @ExceptionHandler on a method that will be called when a given exception occurs. But this handles only those exception that occur within the controller in which it is defined. To overcome this problem you can now use the @ControllerAdvice annotation.

**@RestController**: This annotation is used at the class level. The @RestController annotation marks the class as a controller where every method returns a domain object instead of a view.

**@RestControllerAdvice**: This annotation is applied on Java classes. **@RestControllerAdvice** is a convenience annotation which combines **@ControllerAdvice** and **@ResponseBody**. This annotation is used along with the @ExceptionHandler annotation to handle exceptions that occur within the controller

**Difference between @Controller and @RestController**

* @Controller is used to mark classes as Spring MVC Controller.
* @RestController is a convenience annotation that does nothing more than adding the @Controller and @ResponseBody annotations.

So the following two controller definitions should do the same.

@Controller

@ResponseBody

public class MyController { }

@RestController

public class MyRestController { }

If you use @RestController you cannot return a view (By using Viewresolver in Spring/springboot) and yes @ResponseBody is not needed in this case. If you use @Controller you can return a view in Spring web MVC.

**@Lazy**

@Lazy  
@Component  
public class Address {  
 public Address() {  
 System.*out*.println("Address constructor invoked");  
 }  
 public String getStreetName() {  
 return "some street name";  
 }  
}

@Data  
@Component  
**public class** Emp {  
  
 @Autowired @Lazy  
 **private** Address **adrs**;  
 **public void** showName() {  
 System.***out***.println(**"Emp Name: John"**);  
 }  
 **public void** showStreetName() {  
 System.***out***.println(**"Street Name: "**+**this**.**adrs**.getStreetName());  
 }  
}

**Test class**

@Autowired  
**private** Emp **emp**;  
  
@EventListener(ApplicationReadyEvent.**class**)  
**public void** onReady() {  
 **emp**.showName();  
 **emp**.showStreetName();  
}

**OUTPUT**

Emp Name: John

Address constructor invoked

Street Name: some street name

Various Mappings and their usage

**@PostMapping**(path = "/user",

consumes = MediaType.APPLICATION\_JSON\_VALUE,

produces = MediaType.APPLICATION\_JSON\_VALUE)

**@PutMapping**(path = "/user",

consumes = MediaType.APPLICATION\_JSON\_VALUE,

produces = MediaType.APPLICATION\_JSON\_VALUE)

**@GetMapping**("/employees/{id}")

**@DeleteMapping**("/employees/{id}")

**@PatchMapping**("/employees/{id}/{firstName}")