Spring Important Annotations

1. @Component Annotation:

`@Component` is a common annotation used to mark a Java class as a Spring-managed component. It indicates that the class should be considered as a candidate for auto-detection and automatic bean creation by Spring's component scanning mechanism.

Example:

```
@Component
public class MyComponent {
    // Class implementation
}
```

---

2. @Autowired Annotation:

`@Autowired` is used to automatically wire dependencies by Spring's dependency injection mechanism. It can be applied to fields, constructors, and setter methods, allowing Spring to automatically resolve and inject the required dependencies.

Example:

```
@Component
public class MyComponent {
    private MyDependency myDependency;

    @Autowired
    public MyComponent(MyDependency myDependency) {
        this.myDependency = myDependency;
    }
}
```

---

3. @Qualifier Annotation:

`@Qualifier` is used in conjunction with `@Autowired` to specify which bean should be autowired when multiple beans of the same type are available. It helps to resolve ambiguity when multiple beans match the required dependency.

Example:

```
@Component
public class MyComponent {
    @Autowired
    @Qualifier("myBean")
    private MyInterface myBean;
}
```

---

4. @Primary Annotation:

`@Primary` is used to indicate a primary bean when multiple beans of the same type are available. It specifies that a particular bean should be preferred when autowiring by type.

Example:

```
@Component
@Primary
public class MyPrimaryBean implements MyInterface {
    // Class implementation
}
```

---

5. @SpringBootApplication:

`@SpringBootApplication` is a convenience annotation that combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. It is typically placed on the main application class and enables the Spring Boot auto-configuration and component scanning features.

Example:

```
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

---

6. @Bean and @Configuration Annotations:

@Bean is used to explicitly declare a bean and is typically used within a class annotated with `@Configuration`. It allows manual configuration and definition of beans when auto-configuration is not sufficient.

Example:

```
@Configuration
public class MyConfiguration {
    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

---

7. @Controller, @Service, and @Repository Annotations:

These annotations are used to categorize Spring-managed components based on their roles in the application. `@Controller` is used to mark classes as controllers for handling web requests, `@Service` is used for business logic components, and `@Repository` is used for data access components.

Example:

```
@Controller
public class MyController {
    // Class implementation
}

@Service
public class MyService {
    // Class implementation
}

@Repository
public class MyRepository {
    // Class implementation
}
```

---

8. @Lazy Annotation:
@Lazy is used to indicate that a bean should be lazily initialized, i.e., the bean will be created only when it is first requested. This helps optimize the application's startup time.

Example:

```
@Component
@Lazy
public class MyLazyBean {
    // Class implementation
}
```

9. @Scope Annotation:
@Scope is used to define the scope of a Spring bean. It allows you to specify whether a bean should be singleton (default), prototype, request, session, or a custom scope.

Example:

```
@Component
@Scope("prototype")
public class MyPrototypeBean {
    // Class implementation
}
```

10. @Value Annotation:
@Value is used to inject values into fields, constructors, or setter methods. It allows you to specify property values from configuration files or provide literal values directly.

Example:

```
@Component
public class MyComponent {
    @Value("${my.property}")
    private String myProperty;
}
```

11. @PropertySource and @PropertySources Annotations:

`@PropertySource` is used to specify the location of a properties file to be used for configuring Spring beans. `@PropertySources` is used to specify multiple `@PropertySource` annotations.

Example:

```
@Configuration
@PropertySource("classpath:my.properties")
public class MyConfiguration {
    // Class implementation
}
```

12. @ConfigurationProperties Annotation:

`@ConfigurationProperties` is used to bind external properties to a Spring bean. It maps properties from a properties file or environment variables to the fields of the annotated class.

Example:

```
@Component
@ConfigurationProperties(prefix = "my")
public class MyConfiguration {
    private String property1;
    private String property2;
    // Getters and setters
}
```

---

13. @Controller and @ResponseBody Annotations:

`@Controller` is used to mark a class as a controller to handle web requests. `@ResponseBody` is used to indicate that the return value of a method should be serialized directly into the HTTP response body.

Example:

```
@Controller
public class MyController {
    @ResponseBody
    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }
}
```

---

14. @RestController Annotation:

`@RestController` is a combination of `@Controller` and `@ResponseBody`. It is used to simplify the creation of RESTful web services by automatically serializing the return value of methods into the response body.

Example:

```
@RestController
public class MyRestController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }
}
```

---

15. @RequestMapping Annotation:

`@RequestMapping` is used to map web requests to specific controller methods. It allows you to define the URL path and HTTP methods that should trigger the execution of a particular method.

Example:

```
@Controller
@RequestMapping("/api")
public class MyController {
    @GetMapping("/hello")
    public String hello() {
    return "Hello, World!";
    }
}
```

---

16. @GetMapping Annotation:

`@GetMapping` is a shortcut for `@RequestMapping(method = RequestMethod.GET)`. It is used to map GET HTTP requests to specific controller methods.

Example:

```
@Controller
@RequestMapping("/api")
public class MyController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }
}
```

---

17. @PostMapping and @RequestBody Annotations:

`@PostMapping` is a shortcut for `@RequestMapping(method = RequestMethod.POST)`. It is used to map POST HTTP requests to specific controller methods. `@RequestBody` is used to bind the request body to a method parameter.

Example:

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @PostMapping("/users")
    public User createUser(@RequestBody User user) {
        // Process the user data
        return user;
    }
}
```

---

18. @PutMapping Annotation:

`@PutMapping` is a shortcut for `@RequestMapping(method = RequestMethod.PUT)`. It is used to map PUT HTTP requests to specific controller methods.

Example:

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @PutMapping("/users/{id}")
    public User updateUser(@PathVariable("id") Long id, @RequestBody User user)
{
        // Update the user with the specified ID
        return user;
    }
}
```

---

19. @DeleteMapping Annotation:

`@DeleteMapping` is a shortcut for `@RequestMapping(method = RequestMethod.DELETE)`. It is used to map DELETE HTTP requests to specific controller methods.

Example:

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @DeleteMapping("/users/{id}")
    public void deleteUser(@PathVariable("id") Long id) {
        // Delete the user with the specified ID
    }
}
```

---

20. @PathVariable Annotation:

`@PathVariable` is used to bind a path variable from the request URL to a method parameter. It allows you to access dynamic parts of the URL.

Example:

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @GetMapping("/users/{id}")
    public User getUserById(@PathVariable("id") Long id) {
        // Retrieve and return the user with the specified ID
    }
}
```

---

21. @RequestParam Annotation:

`@RequestParam` is used to bind request parameters to method parameters in a controller method. It allows you to extract query parameters from the request URL.

Example:

```
@RestController
@RequestMapping("/api")
public class MyRestController {
    @GetMapping("/users")
    public List<User> getUsersByRole(@RequestParam("role") String role) {
        // Retrieve and return users based on the specified role
    }
}
```

---

22. @EnableAutoConfiguration:

`@EnableAutoConfiguration` is used to enable Spring Boot's auto-configuration feature. It automatically configures the Spring application based on the classpath and dependencies, reducing the need for manual configuration.

Example:

```
@SpringBootApplication
@EnableAutoConfiguration
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

---

23. @ComponentScan:

`@ComponentScan` is used to enable component scanning in Spring. It specifies the base package(s) to scan for Spring components such as `@Component`, `@Controller`, `@Service`, and `@Repository`.

Example:

```java
@SpringBootApplication
@ComponentScan(basePackages = "com.example")
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

These annotations play a crucial role in Spring applications, enabling various features such as component scanning, dependency injection, request mapping, and configuration management.