Spring Boot - Annotations

Last Updated : 19 Aug, 2025

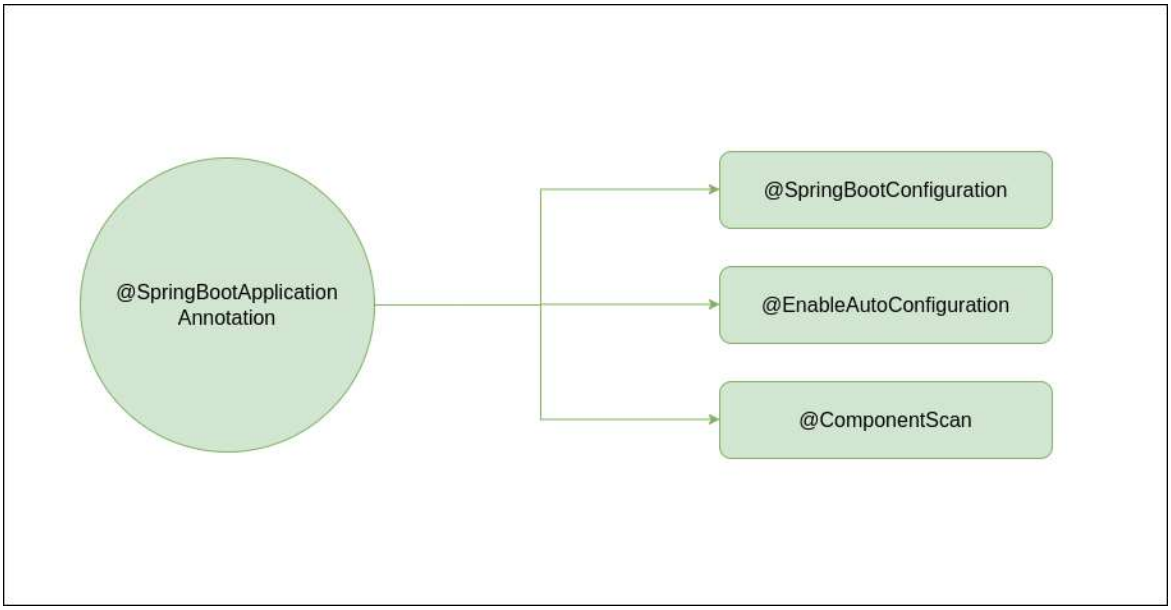
Annotations in Spring Boot are metadata that simplify configuration and development. Instead of XML, annotations are used to define beans,

code and make building applications faster and easier.

Core Spring Boot Annotations

1. @SpringBootApplication Annotation

- This annotation is used to mark the main class of a Spring Boot application.
- It encapsulates @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan annotations with their default attributes.



Example:

```
@SpringBootApplication

// Class
public class DemoApplication {

    // Main driver method
    public static void main(String[] args)
    {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

2. @SpringBootConfiguration Annotation

- Indicates that a class provides configuration for a Spring Boot application.
- Alternative to Spring's @Configuration.
- Included automatically with @SpringBootApplication.

Example:

```
@SpringBootConfiguration
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public StudentService studentService() {
        return new StudentServiceImpl();
    }
}
```

3. @EnableAutoConfiguration Annotation

- This annotation auto-configures the beans that are present in the classpath.
- Enables Spring Boot's auto-configuration mechanism.

Example:

```
@Configuration
@EnableAutoConfiguration
public class Application {

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

4. @ComponentScan Annotation

- Tells Spring where to search for components (@Controller, @Service, @Repository, etc.).
- Typically used with @Configuration.

Example:

```
@Configuration
@ComponentScan

// Main class
public class Application {

    // Main driver method
    public static void main(String[] args)
    {
        SpringApplication.run(Application.class, args);
    }
}
```

5. @ConditionalOnClass Annotation and @ConditionalOnMissingClass Annotation

@ConditionalOnClass Annotation used to mark auto-configuration bean if the class in the annotation's argument is present/absent.

Example:

```
@Configuration
@ConditionalOnClass(MongoDBService.class)

class MongoDBConfiguration {
```

```
    // Insert code here
}
```

6. @ConditionalOnBean Annotation and @ConditionalOnMissingBean Annotation

These annotations are used to let a bean be included based on the presence or absence of specific beans.

Example:

```
@Bean
@ConditionalOnMissingBean(type = "JpaTransactionManager")

JpaTransactionManager jpaTransactionManager(
    EntityManagerFactory entityManagerFactory)
{
    // Insert code here
}
```

7. @ConditionalOnProperty Annotation

These annotations are used to let configuration be included based on the presence and value of a Spring Environment property.

Example:

```
@Bean
@ConditionalOnProperty(name = "usemongodb",
                       havingValue = "local")

DataSource
dataSource()
{
    // Insert code here
}

@Bean
@ConditionalOnProperty(name = "usemongodb",
                       havingValue = "prod")

DataSource
dataSource()
{
```

```
    // Insert code here
}
```

8. @ConditionalOnResource Annotation

These annotations are used to let configuration be included only when a specific resource is present in the classpath.

Example:

```
@ConditionalOnResource(resources = "classpath:mongodb.properties")

Properties
additionalProperties()
{
    // Insert code here
}
```

9. @ConditionalOnExpression Annotation

These annotations are used to let configuration be included based on the result of a **SpEL (Spring Expression Language) expression.**

> ***SpEL (Spring Expression Language):*** *It is a expression language that supports querying and manipulating an object graph at runtime.*

Example:

```
@Bean
@ConditionalOnExpression("'${env}' == 'local'")
public DataSource dataSource() {
    return new HikariDataSource(); // or your DataSource impl
}
```

10. @ConditionalOnCloudPlatform Annotation

These annotations are used to let configuration be included when the specified cloud platform is active.

Example:

```
@Configuration
@ConditionalOnCloudPlatform(CloudPlatform.CLOUD_FOUNDRY)

public class CloudConfigurationExample
{
  // Insert code here
}
```

Request Handling and Controller annotations:

Some important annotations comes under this category are:

- @Controller
- @RestController
- @RequestMapping
- @RequestParam
- @PathVariable
- @RequestBody
- @ResponseBody
- @ModelAttribute

1. @Controller Annotation

- This annotation provides **Spring MVC** features.
- It is used to create Controller classes and simultaneously it handles the HTTP requests.
- Generally we use **@Controller** annotation with **@RequestMapping** annotation to map HTTP requests with methods inside a controller class.

Example:

```
@Controller
public class MyController{
  public String GFG(){
    //insert code here
  }
}
```

2. @RestController Annotation

- This annotation is used to handle REST APIs. and also used to create RESTful web services using Spring MVC.
- It encapsulates **@Controller** annotation and **@ResponseBody** annotation with their default attributes.

@RestController = @Controller + @ResponseBody

Example: Create a Java class and use @RestController annotation to make the class as a request handler

```
@RestController
public class HelloController{
  public String GFG(){
    //insert code here
  }
}
```

3. @RequestMapping Annotation

- Maps HTTP requests to handler methods.
- Supports GET, POST, PUT, DELETE, etc.

Example:

```
@RestController
public class Geeks{
    @RequestMapping(value = "/welcome", method = RequestMethod.GET)
    public String welcome() {
        return "Welcome to Spring Boot!";
    }
}
```

For handling specific HTTP requests we can use

- @GetMapping
- @PutMapping
- @PostMapping

- @PatchMapping
- @DeleteMapping

> ***NOTE:*** *We can manually use GET, POST, PUT and DELETE annotations along with the path as well as we can use @RequestMapping annotation along with the method for all the above handler requests*

4. @RequestParam Annotation

@RequestParam annotation is used to read the form data and binds the web request parameter to a specific controller method.

Example: Java code to demonstrate @RequestParam annotation

```
@RestController
public class MyController{

  @GetMapping("/authors")
  public String getAuthors(@RequestParam(name="authorName") String name){
    //insert code here
  }
}
```

5. @PathVariable Annotation

This annotation is used to extract the data from the URI path. It binds the URL template path variable with method variable.

Example:

```
@RestController
public class MyController{

    @GetMapping("/author/{authorName}")
    public String getAuthorName(@PathVariable(name = "authorName") String
name){
      //insert your code here
    }
}
```

6. @RequestBody Annotation

This annotation is used to convert HTTP requests from incoming JSON format to domain objects directly from request body. Here method parameter binds with the body of the HTTP request.

Example: Java Code to Demonstrate @RequestBody annotation

```
@RestController
public class MyController{

  @GetMapping("/author")
  public void printAuthor(@RequestBody Author author){
    //insert code here
  }
}
```

7. @ResponseBody Annotation

This annotation is used to convert the domain object into HTTP request in the form of JSON or any other text. Here, the return type of the method binds with the HTTP response body.

Example: Java code to demonstrate @ResponseBody annotation

```
@Controller
public class MyController{

  public @ResponseBody Author getAuthor(){
    Author author = new Author();
    author.setName("GFG");
    author.setAge(20);
    return author;
  }
}
```

8. @ModelAttribute Annotation

This annotation refers to model object in Spring MVC. It can be used on methods or method arguments as well.

Example: