

# Spring Boot Annotations: Top 30+ most Used Spring Annotations

Spring Boot Annotations is a type of metadata that provides data about a program. The annotations in Spring Boot is not a part of the program itself and do not have any direct effect on the annotated code's operation.

Spring Boot Annotations do not use XML configuration, instead, they use the convention over configuration.



In this section of the learning [Spring Boot](#) Series, we will discuss different types of annotations with their examples.

In this section, we have included:

- [What are the Spring Boot Annotations](#)

## Core Spring Framework Annotations

1. [@Required](#)
2. [@Autowired](#)

3. [@Configuration](#)
4. [@ComponentScans](#)
5. [@Bean](#)
6. [@Qualifier](#)
7. [@Lazy](#)
8. [@Value](#)

### **Spring Framework StereoType Annotations**

9. [@Component](#)
10. [@Controller](#)
11. [@Service](#)
12. [@Repository](#)

### **Spring Boot Annotations**

13. [@EnableAutoConfiguration](#)
14. [@SpringBootApplication](#)

### **Spring MVC and Rest Annotations**

15. [@Controller](#)
16. [@RequestMapping](#)
17. [@GetMapping](#)
18. [@PostMapping](#)
19. [@PutMapping](#)
20. [@DeleteMapping](#)
21. [@PatchMapping](#)
22. [@RequestBody](#)
23. [@ResponseBody](#)
24. [@PathVariable](#)
25. [@RequestParam](#)
26. [@RequestHeader](#)
27. [@RestController](#)
28. [@ModelAttribute](#)
29. [@CookieValue](#)
30. [@CrossOrigin](#)

## **What are the Spring Boot Annotations**

The Support for annotations in Java is provided since Java 5. leading Java frameworks quickly adopt the annotation's concept. Spring Boot has provided the support for the annotations since the release of Spring 2.5.

Due to their extensive structure and internal concepts, the annotations provide a lot of context in their declaration.

Before the annotations, the Spring Boot project's behavior was controlled and managed by the XML configurations. Now, the Annotations provide us several capabilities to manage and configure the behavior of the Spring Framework.

The annotations in Spring Boot takes place in **org.springframework.boot.autoconfigure** and **org.springframework.boot.autoconfigure.condition** packages.

Let's discuss some frequently used spring boot annotations with examples:

### **@Required**

The @Required annotation is applied on the setter method of the bean file. Use of this annotation indicates that the annotated bean must be populated at configuration time with the required property. Otherwise, it will throw a **BeanInitializationException**.

Consider the below example:

```
package com.javasterling.sb.SpringBootDemo;
import org.springframework.beans.factory.annotation.Required;
public class Employee
{
    private Integer id;

    public Integer getId() {
        return id;
    }
    @Required
    public void setId(Integer id) {
        this.id = id;
    }
}
```

### **@Autowired**

The @Autowired annotation is applied on fields, instance variables, setter methods, and constructors. It provides auto wiring to bean properties. When we apply the @autowire to a field, Spring contain will automatically assign the fields with assigned values. We can also use this annotation on private fields. Consider the below code:

```
package com.javasterling.sb.SpringBootDemo;
import org.springframework.beans.factory.annotation.Autowired;

public class Employee
{
    @Autowired
    private Person p;
```

```
private int sal;
```

```
}
```

When we use `@Autowired` on setter methods, Spring will try to perform the by Type autowiring on the method. It means we are instructing Spring that it should initiate this property using the setter method where you can add your custom code, such as initializing any other property with this property. Consider the below code:

```
package com.javasterling.sb.SpringBootDemo;
import org.springframework.beans.factory.annotation.Autowired;
public class Employee {
    private Person person;
    @Autowired
    public void setPerson (Person person) {
        this.person=person;
    }
}
```

### @Configuration

The `@Configuration` annotation is used on classes which defines bean as the source of the bean file. It is an analog for the XML configuration file. It is configured using the Java class. A Java class that is annotated with this annotation is a configuration itself and will import the methods to instantiate and configure the dependencies. Consider the below example:

```
package com.javasterling.sb.SpringBootDemo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
@Configuration
public class Employee {
    private Person person;
    @Autowired
    public void setPerson (Person person) {
        this.person=person;
    }
}
```

### @ComponentScan

The `@ComponentScan` annotation is used to scan a package for beans. It is used with `@Configuration` annotation. We can also define the base package for scanning. If we do not specify the base package, it will scan from the package of the class that declares this annotation.

Consider the below example:

```
package com.javasterling.sb.SpringBootDemo;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@ComponentScan(basePackages = "com.javasterling.sb.SpringBootDemo")
```

```
@Configuration
public class Employee
{
    // ...
}
```

### **@Bean**

The @Bean annotation is an alternative of XML <bean> tag. If you ever have gone through the <bean> tag, you must be familiar with it's working. It is a method level annotation, which tells the method to produce a bean to be managed by the Spring Container. It also works with @Configuration annotation to create Spring beans. As the @Configuration holds methods to instantiate and configure dependencies. The methods, which are annotated with @Bean annotation acts as bean ID and it creates and returns the actual bean.

Consider the below example:

```
@Bean
public BeanDemo beanDemo()
{
    return new BeanDemo ();
}
```

### **@Qualifier**

The @Qualifier annotation is used along with @Autowired annotation. It provides more control over the dependency injection process. It can be specified on individual method parameters or constructors' arguments. It is useful for reducing the duplicity while creating more bean files of the same type and we want to use only one of them with a property.

Consider the below example, which is implementing BeanInterface by two beans B1 and B2:

```
@Component
public class B1 implements BeanInterface {
    //
}
@Component
public class B2 implements BeanInterface {
    //
}
```

If B1 autowires the BeanInterface, it is hard for Spring to decide which one of the given implementation to inject.

The solution to this problem is that we can use the @Qualifier annotation. Consider the below example:

```
@Component
```

```

public class B1 {
    @Autowired
    @Qualifier("beanB2")
    private BeanInterface dependency;
    ...
}

```

Now, the Spring will easily decide which bean to autowire.

## **@Lazy**

The @Lazy annotation is used on component classes. By default, Spring boot initializes all the autowired dependencies and configure the project at startup. This annotation allows us to initialize the project components lazily, which means, the bean will be created and initialized when it is requested.

When the @Lazy annotation is used on a @Configuration class; it will initialize all the @Bean methods lazily.

Consider the below example:

```

package com.javasterling
@Lazy
@Configuration
@ComponentScan(basePackages = "com.javasterling")
public class AppConfig {
    @Bean
    public EmpID getEmpID(){
        return new EmpID();
    }
    @Bean
    public Sal getSal(){
        return new Sal();
    }
}

```

## **@Value**

The @value annotation is used at field, constructor parameter, and method parameter level. It indicates a default value for the field or parameter to initialize the property with. Its use is quite similar to @Autowired annotation; it also injects values from a property file into a bean's attribute. It supports both \${.....} and #{....} placeholders.

## **Spring Framework Stereotype Annotations**

### **@Component**

The `@Component` annotation is used at the class level to mark a Java class as a bean. When we mark a class with `@Component` annotation, it will be found during the classpath. The Spring framework will configure the annotated class in the application context as a Spring Bean.

Consider the below example:

```
package com.javasterling;
@Component
public class Employee
{
.....
}
```

### **@Controller**

The `@Controller` annotation is also used at the class-level; it is a specialized version of `@Component`. It is used to annotate a class as a web request handler; most commonly used to serve web pages. It returns a string having a redirect route. Generally, it is used with `@RequestMapping` annotation.

### **@Service**

The `@Service` annotation is also used at the class level to mark a service implementation including business logic, calculations, call external APIs, etc. Generally, it holds the business logic of the application.

Consider the below example:

```
package com.javasterling;
@Service
public class ServiceDemo
{
public void service1()
{
//business code
}
}
```

### **@Repository**

The `@Repository` annotation is used on java classes which directly access the database. It acts as a marker for any class that is used as a repository or DAO ( Data Access Object).

This annotation has a built-in translation feature means when the class found an exception, it will automatically handle with that; there is no need to add a try-catch block manually.

Consider the below example:

```
package com.javasterling;  
@Repository  
public class Test  
{  
    public void delete()  
{  
    //  
}  
}
```

## Spring Boot Annotations

### **@EnableAutoConfiguration**

The **@EnableAutoConfiguration** annotation is used to implicitly defines a base “search package”. Usually, it is placed on the main application class. It will automatically configure the projects by adding beans based on classpath settings, beans, and other property settings.

The use of this annotation is degraded in Spring Boot 1.2.0 release because there is a great alternative of the annotation **@SpringBootApplication** is available from.

### **@SpringBootApplication**

The **@SpringBootApplication** is the combination of three annotations

**@EnableAutoConfiguration**, **@ComponentScan**, and **@Configuration**.

The **@SpringBootApplication** annotation is used on the application class while setting up a new Spring Boot project. The class that is annotated with this annotation must be placed in the base package. The main task of this annotation is to scan the projects; it scans its sub-packages only. For example, if we annotate a class that is available in a sub-package then it will not scan the main package.

## Spring MVC and Rest Annotations

### **@Controller**

The **@Controller** annotation can be used with Spring MVC and WebFlux. It is used at the class level as a controller. It useful for detecting the component classes from the classpath and autoregister the bean definition. The class annotated with this annotated can handle multiple request mappings.

### **@RequestMapping**



The `@RequestMapping` annotation is used to map the web requests. It can hold several optional components such as consumes, header, method, name, params, path, value, etc. This annotation can be used with classes as well as methods. Consider the below example:

```
@Controller
public class SellController
{
    @RequestMapping("/computer-science/books")
    public String getAllItems(Model model)
    {
        //application code
        return "itemList";
    }
}
```

### **@GetMapping**

The `@GetMapping` annotation is used to map the HTTP GET request on the specific handler method. It is an alternative of **`@RequestMapping(method = RequestMethod.GET)`**.

### **@PostMapping**

The `@PostMapping` annotation is used for mapping HTTP POST requests onto specific handler methods. It is an alternative of **`@RequestMapping(method = RequestMethod.POST)`**.

### **@PutMapping**

The `@PutMapping` annotation is used to map the HTTP PUT requests on the specific handler method. It is useful for creating a web service endpoint that creates or updates. It is an alternative of **`@RequestMapping(method = RequestMethod.PUT)`**.

### **@DeleteMapping**

The `@DeleteMapping` is used to map the HTTP DELETE requests on the specific handler method. It is useful for creating a web service endpoint that deletes a resource. It is an alternative of **`@RequestMapping(method = RequestMethod.DELETE)`**.

### **@PatchMapping**

The `@PatchMapping` is used to map the HTTP PATCH requests on the specific handler method. It is an alternative of **`@RequestMapping(method = RequestMethod.PATCH)`**.

### **@RequestBody**

The `@RequestBody` annotations is used to bind HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the

request's body. When a method parameter is annotated with **@RequestBody** annotation, the Spring framework wraps the incoming HTTP request body to that parameter.

### **@ResponseBody**

The **@ResponseBody** annotation is used to bind the method return value to the response body. It acknowledges the Spring Boot Framework to serialize a return object into JSON and XML format.

### **@PathVariable**

The **@PathVariable** annotation is used to extract the values from the URI. It is the most suitable annotation for building the RESTful web service, where the URL contains a path variable. The multiple **@PathVariable** can be defined in a method.

### **@RequestParam**

The **@RequestParam** annotation aka query parameter is used to extract the query parameters from the URL. It is most suitable for developing web applications. It can define the default values if the query parameter is not present in the URL.

### **@RequestHeader**

The **@RequestHeader** annotation is used to get the details about the HTTP request headers. It can be used as a method parameter. The optional elements of this annotation are name, value, required, defaultValue. We must specify a separate annotation for each detail in the header. It can be used repeatedly in a method.

### **@RestController**

The **@RestController** annotation is a combination of **@Controller** and **@ResponseBody** annotations. It is itself annotated with the **@ResponseBody** annotation. If the **@RestController** is used, then there is no need for annotating each method with **@ResponseBody**.

### **@RequestAttribute**

The **@RequestAttribute** annotation is used to wrap a method parameter to the request attribute. It gives convenient access to the request attributes from a controller method. Using this annotation, we can access objects that are populated on the server-side.

### **@CookieValue**

The **@CookieValue** annotation is used at the method parameter level. Usually, it is used as request mapping method's argument. The HTTP cookie is bound for a given cookie name to the **@CookieValue** parameter. It is used with the method that is annotated with **@RequestMapping**.

### **@CrossOrigin**

The `@CrossOrigin` annotation is used both at the class & method level. It is used to enable cross-origin requests. It is useful for the cases, if a host serving JavaScript is different from the data serving host. In such cases, the CORS (Cross-Origin Resource Sharing) enables cross-domain communication. In order to enable this communication, we just need to add the `@CrossOrigin` annotation. The default behavior of this annotation allows all origins, all headers, the HTTP methods specified in the `@RequestMapping` annotation and maxAge of 30 min. However, we can control and customize the behavior by specifying the corresponding attribute values.

### **Summary:**

We have discussed some most frequently used annotations in Spring Boot. We will use these annotations through our [Spring Boot](#) Course with some real-time examples.