Search...

Spring Core Annotations

Last Updated : 23 Jul, 2025

Spring Annotations are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

Spring Framework is one of the most popular frameworks for building Java applications. It provides a wide range of annotations that simplify configuration, dependency injection, and bean management. In this article, we will explore **Spring Core Annotations, their types, and how they are used in Spring applications.**

Types of Spring Framework Annotations

The below diagram demonstrates the types of Spring Framework Annotations

Spring Framework Annotations
01 Spring Core Annotations
02 Spring Web Annotations
03 Spring Boot Annotations
04 Spring Scheduling Annotations
05 Spring Data Annotations
06 Spring Bean Annotations

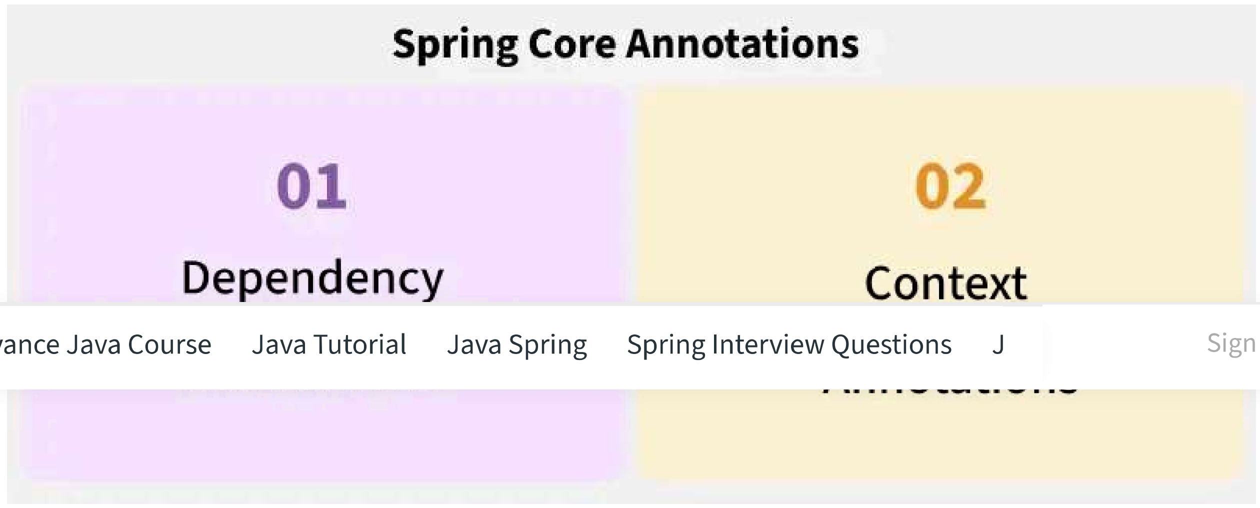
- **Spring Core Annotations:** Used for dependency injection, bean configuration, and context management.
- **Spring Web Annotations:** Used for building web applications and RESTful services.

- **Spring Boot Annotations:** Simplify Spring Boot application configuration.
- **Spring Scheduling Annotations:** Used for scheduling tasks.
- **Spring Data Annotations:** Used for data access and persistence.
- **Spring Bean Annotations:** Used for defining and managing beans.

Spring Core Annotations

Spring annotations present in the **org.springframework.beans.factory.annotation** and **org.springframework.context.annotation** packages are commonly known as Spring Core annotations. We can divide them into two categories:

- **DI-Related Annotations:** Used for dependency injection.
- **Context Configuration Annotations:** Used for configuring the Spring application context.



DI-Related Annotations

These annotations are used to manage dependency injection (DI) in Spring. The most commonly used DI-related annotations are listed below

1. @Autowired: This annotation is used to inject dependencies automatically. It can be applied to fields, setter methods, and constructors.

Example of Field Injection:

```
@Component
```

```
public class Student {
    @Autowired
    private Address address;
}
```

Example of Constructor Injection:

```
@Component
public class Student {
    private Address address;

    @Autowired
    public Student(Address address) {
        this.address = address;
    }
}
```

Example of Setter Injection:

```
@Component
public class Student {
    private Address address;

    @Autowired
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

2. @Qualifier: The @Qualifier annotation is used to resolve the autowiring conflict when there are multiple beans of the same type. The @Qualifier annotation can be used on any class annotated with @Component or on methods annotated with @Bean. This annotation can also be applied to constructor arguments or method parameters.

Example:

```
@Component
public class VehicleService {
    @Autowired
    @Qualifier("bike")
    private Vehicle vehicle;
}
```

3. @Primary: This indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency. If exactly one ‘primary’ bean exists among the candidates, it will be the autowired value. Let’s understand this statement with an example

Example: In some cases, we need to register more than one bean of the same type. In this example employee1() and employee2() beans of the Employee type:

```
@Configuration
public class Config {
    @Bean
    public Employee employee1() {
        return new Employee("Employee1");
    }

    @Bean
    @Primary
    public Employee employee2() {
        return new Employee("Employee2");
    }
}
```

In this case, if we try to run the application Spring will throw ***NoUniqueBeanDefinitionException***. To resolve this issue Spring offers the @Primary annotation.

```
@Configuration
public class Config {

    @Bean
    public Employee Employee1() {
        return new Employee("Employee1");
    }

    @Bean
```

```
        @Primary
    public Employee Employee2() {
        return new Employee("Employee2");
    }
}
```

4. @Bean: This annotation is used to define a bean in a configuration class. It is typically used in @Configuration classes.

Example:

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

5. @Lazy: This annotation delays the initialization of a bean until it is first requested.

Example:

```
@Component
@Lazy
public class MyService {
    // Bean will be initialized only when requested
}
```

6. @Value: This annotation is used to inject values from properties files or environment variables into fields.

Example:

```
@Component
public class MyService {
    @Value("${app.name}")
    private String appName;
}
```

7. @Scope: This annotation defines the scope of a bean, such as singleton, prototype, request, or session.

Example:

```
@Component
@Scope("prototype")
public class MyService {
    // Bean will be created each time it is requested
}
```

8. @Lookup: This annotation is used for method injection. It allows a method to return a new instance of a bean each time it is called.

Example:

```
@Component
public abstract class MyService {
    @Lookup
    public abstract MyBean getMyBean();
}
```

9. @Required: This annotation was used to enforce that a particular property must be injected.

> ***Note:*** *This annotation is deprecated as of Spring 5.1*

Context Configuration Annotations

These annotations are used to configure the Spring application context.

1. @Configuration: This annotation indicates that a class is a configuration class and can define beans using @Bean methods.

Example:

```java
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

2. @ComponentScan: This annotation is used to specify the packages to scan for Spring components like @Component, @Service, @Repository, etc.

Example:

```java
@Configuration
@ComponentScan("com.example")
public class AppConfig {
}
```

3. @Import: This annotation is used to import one or more @Configuration classes into another configuration class.

Example:

```java
@Configuration
@Import({Config1.class, Config2.class})
public class AppConfig {
}
```

4. @ImportResource: This annotation is used to import XML configuration files into a Java-based configuration class.

Example:

```java
@Configuration
@ImportResource("classpath:app-config.xml")
public class AppConfig {
}
```

5. @PropertySource: This annotation is used to load properties files into the Spring environment.

Example:

```
@Configuration
@PropertySource("classpath:app.properties")
public class AppConfig {
}
```

6. @Profile: This annotation is used to conditionally load beans based on active profiles.

Example:

```
@Component
@Profile("dev")
public class DevDataSource {
    // Bean will be loaded only if the "dev" profile is active
}
```

7. @Conditional: This annotation is used to conditionally register beans based on specific conditions.

Example:

```
@Bean
@Conditional(MyCondition.class)
public MyService myService() {
    return new MyService();
}
```

By mastering these annotations, we can build robust and maintainable Spring applications.

Comment More info Advertise with us