Spring @Configuration Annotation with Example

Last Updated : 23 Jul, 2025

The **@Configuration annotation** in Spring is one of the most important annotations. It indicates that a class contains @Bean definition methods, which the Spring container can process to generate Spring Beans for use in the application. This annotation is part of the Spring Core framework.

Let's understand the @Configuration annotation in Spring with an example project.

Steps to Implement @Configuration Annotation

Suppose we already have a Java project, and all the Spring JAR files are imported into that project. No,w let’s create a simple class named College, and inside the class.

Step 1: Create the College Class

We start by creating a simple Java class named College. This class contains a method called test() that prints a message to the console.

College.java

```java
// Java Program to Illustrate College Class

package ComponentAnnotation;

// Class
public class College {

    // Method
    public void test()
    {
        // Print statement
        System.out.println("Test College Method");
    }
}
```

We can use the @Component annotation to create the bean for this class. So, we can modify our College.java file as follows:

```java
// Java Program to Illustrate College Class

package ComponentAnnotation;

// Class
public class College {
    // Method
    public void test() {
        // Print statement
        System.out.println("Test College Method");
    }
}
```

In this case, we have to write the following line inside the beans.xml file:

<context:component-scan base-package="ComponentAnnotation"/>

However, we don't want to use the complete beans.xml file in our project. So, what can we do to replace the beans.xml file? In general, beans.xml is a configuration file. So, we can create a configuration class in Java and make this class our configuration class by using the @Configuration annotation.

Step 2: Create CollegeConfig Class

```java
// Java Program to Illustrate CollegeConfig Class

package ComponentAnnotation;

// Importing required classes
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
// Class
@Configuration
public class CollegeConfig {

    // Using Bean annotation to create College class Bean
    @Bean
    // Here the method name is the bean id/bean name
    public College collegeBean() {
        // Return the College object
        return new College();
    }
}
```

Now, we don't want to use the @Component and @ComponentScan annotations to create the beans. Let's discuss another way of doing the same task. We are going to create the Spring beans using the @Bean annotation. But how? Where to write these methods? As we have discussed at the beginning, the @Configuration annotation indicates that the class has @Bean definition methods. So, let's explain this statement and create our beans inside the CollegeConfig.java file using the @Bean annotation.

Step 3: Define Beans Using @Bean Annotation

In the CollegeConfig class, we define beans using the @Bean annotation. Each method annotated with @Bean returns an instance of a class that will be managed by the Spring container.

For example, the collegeBean() method returns a College object, which is registered as a Spring bean with the name collegeBean.

```
@Bean
public College collegeBean() {
    return new College();
}
```

This approach allows us to define beans programmatically without relying on XML configuration files.

Step 4: Define Beans in the Configuration Class

Similarly, if we have another class named Student and we want to create the bean for this Student class, we can define it in the same configuration class using the @Bean annotation.

```
@Bean
public Student studentBean() {
    return new Student();
}
```

This method creates and returns a Student object, which is registered as a Spring bean with the name studentBean. By defining multiple @Bean methods in the configuration class, we can manage all our beans in one place without needing an XML file.

Step 5: Create the Main Class

Finally, we create a main class to test our configuration. In this class, we use the AnnotationConfigApplicationContext to load the configuration class and retrieve the beans.

```
// Java Program to Illustrate Application Class

package ComponentAnnotation;

// Importing required classes
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

// Application class
public class Main {

    // Main driver method
    public static void main(String[] args) {
        // Use AnnotationConfigApplicationContext instead of
ClassPathXmlApplicationContext
        // because we are not using XML Configuration
        ApplicationContext context = new
AnnotationConfigApplicationContext(CollegeConfig.class);

        // Getting the bean
        College college = context.getBean("collegeBean", College.class);

        // Invoking the method inside main() method
        college.test();
    }
}
```

We use AnnotationConfigApplicationContext to load the CollegeConfig class, which contains our bean definitions. We retrieve the College bean using the getBean() method and call its test() method.

Output:

> *Test College Method*

This output confirms that the College bean was successfully created and its test() method was invoked.

Comment More info Advertise with us