Spring @Value Annotation with Example

Last Updated : 23 Jul, 2025

The @Value annotation in Spring is one of the most important annotations. It is used to assign default values to variables and method arguments. It allows us to inject values from spring environment variables, system variables, and properties files. It also supports **Spring Expression Language (SpEL).**

Key Points about @Value:

- The @Value annotation is used to inject values into fields, methods, or constructor parameters in Spring beans, typically from property files, environment variables, or expressions.
- It also supports SpEL, allowing dynamic values or expressions to be evaluated and injected.
- The @Value annotation also binds values from **application.properties** or **application.yml** to Spring beans
- The @Value annotation also converts the injected value to the required data type.

Note: @Value annotation is commonly used to inject configuration values into spring beans.

In this article, we will demonstrate how to use the **@Value annotation** with a step-by-step example.

Steps to Use the @Value Annotation in Spring

Step 1: Create a Simple Spring Application

First, let's create a simple Spring Application and inject the literal values by setter injection. So, create a simple class Student having three attributes rollNo, name, and age. Create setter methods for these two attributes and a simple method to print the details of the student.

Student.java:

```
// Java Program to Illustrate Student Class
public class Student {

    // Class data members
    private int rollNo;
    private String name;
    private int age;

    // Setter for rollNo
    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }

    // Setter for name
    public void setName(String name) {
        this.name = name;
    }

    // Setter for age
    public void setAge(int age) {
        this.age = age;
    }

    // Method to display student details
    public void display() {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Step 2: Create a Properties File

Let's create a properties file in your classpath and name the file as **student-info.properties** (for this example we name it like this, you can name it according to your need). And in this file, we are going to write something like as follows:

Student-info.properties:

// properties

student.rollNo=101

student.name=Sagar

student.age=20

Step 3: Configure the Spring Bean in XML

Now let's create a Student Bean in the beans.xml file and inside the bean, you have to add your property's name and its corresponding values inside the <property> tag. For example, for this project, we can write something like below:

<context:property-placeholder location="classpath:student-info.properties"/>

<bean id="student" class="Student">

<property name="rollNo" value="${student.rollNo}"/>

<property name="name" value="${student.name}"/>

<property name="age" value="${student.age}"/>

</bean>

beans.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans/"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context/"
    xsi:schemaLocation="http://www.springframework.org/schema/beans/
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context/
        https://www.springframework.org/schema/context/spring-context.xsd">

    <context:property-placeholder location="classpath:student-
info.properties"/>
```

```
    <bean id="student" class="Student">
        <property name="rollNo" value="${student.rollNo}"/>
        <property name="name" value="${student.name}"/>
        <property name="age" value="${student.age}"/>
    </bean>

</beans>
```

Step 4: Create the Main Class

So now our bean is ready. Now let's create a class and define the main() method inside that class. Suppose we have created a class named Main and we have defined the main() method inside this class.

Main.java:

```
// Java Program to Illustrate Application Class
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

    public static void main(String[] args) {
        // Load the Spring context
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");

        // Retrieve the Student bean
        Student student = context.getBean("student", Student.class);

        // Display student details
        student.display();
    }
}
```

Step 5: Run the Application

When we run the Main class, we will get the output:

Roll No: 101

Name: Sagar

Age: 20

Explanation: So the application is working fine. Now come to the beans.xml file again. And in this file, we don't want to set value like this as we have done. We want to use some annotation for doing the same thing. And here @Value Annotation comes into the picture.

Using @ Value Annotation

Instead of using XML configuration, we can use the @Value annotation to inject values directly into the Student class. Let's modify the Student class to use @Value. So we can modify our Student.java file something like below:

Step 6: Update Student.java

```
// Java Program to Illustrate Student Class

// Importing required classes
import org.springframework.beans.factory.annotation.Value;

// Class
public class Student {

    // Class data members
    private int rollNo;
    private String name;
    private int age;

    @Value("101")
    // Setter
    public void setRollNo(int rollNo)
    {
        this.rollNo = rollNo;
    }

    @Value("Anshul")
    // Setter
    public void setName(String name)
    {
        this.name = name;
    }

    @Value("25")
    // Setter
    public void setAge(int age)
    {
```

```
        this.age = age;
    }

    // Method
    public void display()
    {
        // Print statement
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Step 7: Configure the Spring Bean in XML

Add the following line inside the beans.xml file.

<context:annotation-config/>

beans.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans/"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context/"
    xsi:schemaLocation="http://www.springframework.org/schema/beans/
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context/
        https://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config/>

    <bean id="student" class="Student">

    </bean>

</beans>
```

Step 8: Run the Application

Run the main() method and the output will be like this.

Roll No: 101

Name: Anshul

Age: 25

Note: We can also set the values dynamically from the properties files.

Student.java:

We can Modify the Student.java file something like this:

```
// Java Program to Illustrate Student Class

// Importing required classes
import org.springframework.beans.factory.annotation.Value;

// Class
public class Student {

    // Class data members
    private int rollNo;
    private String name;
    private int age;

    @Value("${student.rollNo}")
    // Setter
    public void setRollNo(int rollNo)
    {

        // this keyword refers to current instance itself
        this.rollNo = rollNo;
    }

    @Value("${student.name}")
    // Setter
    public void setName(String name)
    {
        this.name = name;
    }

    @Value("${student.age}")
    // Setter
    public void setAge(int age)
    {
        this.age = age;
    }

    // Method
    public void display()
    {
        // Printing attributes corresponding to student
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
```

```
    }
}
```

Note: Don't forget to add the below line inside your **beans.xml** file

<context:property-placeholder location="classpath:student-info.properties"/>

beans.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans/"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context/"
    xsi:schemaLocation="http://www.springframework.org/schema/beans/
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context/
        https://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config/>
    <context:property-placeholder location="classpath:student-
info.properties"/>

    <bean id="student" class="Student">

    </bean>

</beans>
```

Step 9: Again Run the Application

Run the main() method and the output will be like this.

Roll No: 101

Name: Sagar

Age: 20

Use @Value Annotation on Fields

Now let's discuss one more interesting concept on Spring @Value Annotation. We can also use the **@Value annotation before the fields**.

There is no need to create the setter method.

Student.java:

We can modify our Student.java file something like this.

```
// Java Program to Illustrate Student Class

// Importing required classes
import org.springframework.beans.factory.annotation.Value;

// Class
public class Student {

    // Class data members
    @Value("${student.rollNo}") private int rollNo;
    @Value("${student.name}") private String name;
    @Value("${student.age}") private int age;

    // Method
    public void display()
    {
        // Printing attributes corresponding to student
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

All other things should be unchanged. Run the main() method again and the output will be like this.

Roll No: 101

Name: Sagar

Age: 20