Spring - Required Annotation

Last Updated : 23 Jul, 2025

Consider a scenario where a developer wants to make some of the fields as mandatory fields. using the **Spring framework**, a developer can use the @Required annotation to those fields by pushing the responsibility for such checking onto the container. So container must check whether those fields are being set or not.

- The Spring Framework supports a number of custom Java 5+ annotations.
- From Spring 2.0, instead of using XML to describe a bean wiring, the developer can do configuration into the component class itself by using annotations on the relevant class, method, or field declaration.
- It introduced the possibility of enforcing required properties with the @Required annotation and can be registered as individual bean definitions.

@Required Annotation

1. The @Required annotation is available in the **org.springframework.beans.factory.annotation** package since 2.0.

> ***org.springframework.beans.factory.annotation***
>
> ***Annotation Type Required***
>
> *Deprecated as of 5.1, in favor of using constructor injection for required settings.*

2. It applies to bean property setter methods.

3. It provides a method-level annotation that is applied to the bean property setter methods for making the setter-injection mandatory. This means it can be used to mark a property as 'required-to-be-set'.

4. So that container will check the annotated (setter) method of a class if it is configured to be dependency injected with a value or not. If not, an Exception will be thrown by the container at runtime.

5. This annotation indicates that the affected bean property must be populated at configuration time: either through an explicit property value in a bean definition or through autowiring.

```
@Deprecated
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface Required
```

Simply annotating the 'setter' properties of the classes is not enough to get the required behavior, a developer needs to enable or activate the @Required annotation so that it can process appropriately. We can enable @Required annotation in **two ways** in Spring XML configuration:

- Either by registering **RequiredAnnotationBeanPostProcessor** in the bean definition.
- Or can be implicitly registered by including the **<context:annotation-config/>** tag in an XML-based Spring configuration.

Method 1: RequiredAnnotationBeanPostProcessor

1. Spring provides BeanPostProcessor interface and its implementation classes that enforce required JavaBean properties to have been configured.

2. If a developer wants to implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean, can use one or more BeanPostProcessor implementations.

3. This interface defines callback methods that developers can implement to provide their own instantiation logic, dependency-resolution logic, etc.

An example is Spring's **RequiredAnnotationBeanPostProcessor** - a special BeanPostProcessor implementation that is @Required-aware which ensures that JavaBean properties are marked with an @Required annotation must be dependency-injected with a value.

```
java.lang.Object
org.springframework.beans.factory.annotation.RequiredAnnotationBean
PostProcessor
Class RequiredAnnotationBeanPostProcessor
```

Illustration:

```
@Deprecated
public class RequiredAnnotationBeanPostProcessor
extends Object
implements SmartInstantiationAwareBeanPostProcessor,
MergedBeanDefinitionPostProcessor, PriorityOrdered, BeanFactoryAware
```

- Once the RequiredAnnotationBeanPostProcessor is defined in the Spring XML configuration file, after completing the instantiating, configuring, and initializing a bean, the container will check if the affected bean property has been populated or not, if not it will throw an exception.
- This allows for eager and explicit failure and avoiding any Exceptions later on.

We can use RequiredAnnotationBeanPostProcessor in the Spring XML configuration file as shown below.

Example:

```
<?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/
        http://www.springframework.org/schema/beans//spring-beans.xsd
        http://www.springframework.org/schema/context/
```

```
        http://www.springframework.org/schema/context//spring-context.xsd">

    <bean
class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPo
stProcessor" />

</beans>
```

Method 2: context:annotation-config

1. <context:annotation-config/> looks for the annotations on beans in the same application context it is defined in.

2. This is mainly used to activate the dependency injection annotations such as @Required, @Autowired, @PostConstruct, @PreDestroy, etc.

We can use <context:annotation-config/> in Spring XML file as shown below.

Example:

```
<?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/
        http://www.springframework.org/schema/beans//spring-beans.xsd
        http://www.springframework.org/schema/context/
        http://www.springframework.org/schema/context//spring-context.xsd">

    <context:annotation-config/>

</beans>
```

> ***Note:*** *A default RequiredAnnotationBeanPostProcessor will already be registered if you are using the "context:annotation-config" XML tag. You can remove or turn off the default annotation configuration if the intention is to specify a custom RequiredAnnotationBeanPostProcessor bean definition.*

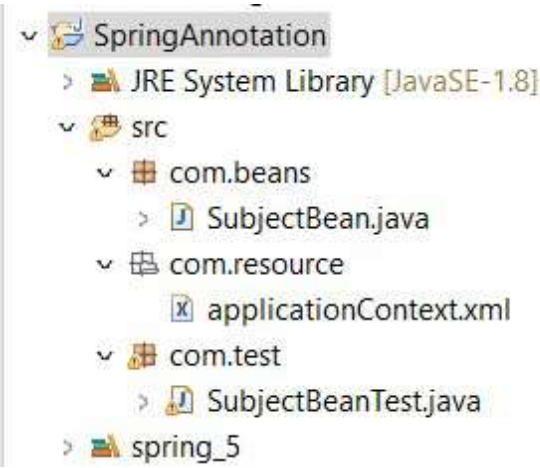
Implementation: We will create a basic Java application using Spring to display a title, subject name, and ID. The steps are as follows:

1. Create a Java application and add necessary Spring library files to it.

2. Create a bean class to define the properties, getter, and setter methods.
3. Create Spring XML configuration file.
4. Create a test class to run the application.

Step 1: Create Java application and add necessary Spring library files to it

- In this example, we are using Eclipse IDE. Create a Java application in Eclipse.
- Download the necessary Spring jar files from Maven Repository and add those to the Java project.
- Below will be the project structure after creating all the necessary classes.



Project_Structure

Step 2: Create a bean class to define the properties, getter, and setter methods. Create a Java bean class to define all the required properties and their getter/setter methods. (**SubjectBean.java)**

Example:

```
package com.beans;

import org.springframework.beans.factory.annotation.Required;

public class SubjectBean {

    private Integer subId;
    private String subName;

    @Required
```

```
    public void setSubName(String subName) {
        this.subName = subName;
    }

    public String getSubName() {
        return subName;
    }

    public Integer getSubId() {
        return subId;
    }

    public void setSubId(Integer subId) {
        this.subId = subId;
    }

}
```

- Created a JavaBean named "**SubjectBean**" with two properties **subName** and **subId.**
- Mark the setter method of **subName** as @Required, so that property must be set with a value - a mandatory field.

Step 3: Create a Spring XML configuration file. To configure the bean values, define those in the XML configuration file.

File: applicationContext.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/
        http://www.springframework.org/schema/beans//spring-beans.xsd
        http://www.springframework.org/schema/context/
        http://www.springframework.org/schema/context//spring-context.xsd">

    <!-- use any one of the below bean definition to activate/enable the
@Required annotation -->

    <!-- Method 1: -->
    <bean
class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPo
stProcessor" />

    <!-- or -->

    <!-- Method 2:
    <context:annotation-config/> -->
```

```
    <bean id="subjectBean" class="com.beans.SubjectBean">
        <property name="subName" value="Java Spring - Annotations" />
        <property name="subId" value="1002" />
    </bean>

</beans>
```

- As explained before, we can use any one of the RequiredAnnotationBeanPostProcessor or context:annotation-config to enable the annotation.
- Create a bean object for SubjectBean class and set the parameter values using the property tag as shown.

Step 4: Create a test class to run the application.

Create **SubjectBeanTest.java** to get the bean and print the property values.

File: SubjectBeanTest.java

```
package com.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.beans.SubjectBean;

public class SubjectBeanTest {

    public static void main(String[] args) throws Exception {

        ApplicationContext con = new
ClassPathXmlApplicationContext("com/resource/applicationContext.xml");
        SubjectBean subject = (SubjectBean) con.getBean("subjectBean");

        System.out.println("Welcome to GeeksforGeeks");
        System.out.println("Subject Name: "+ subject.getSubName());
        System.out.println("Subject ID: "+ subject.getSubId());
    }

}
```

- Create the ApplicationContext object and get the bean object.
- Print the property values to the console.

Execution/Output:

- After creating the required classes and the XML file, run the project as Java application.
- The output should be as below,

Output

- Now to check the @Required annotation, remove/comment the property value of **subName** in XML file like below,

```
<?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/
        http://www.springframework.org/schema/beans//spring-beans.xsd
        http://www.springframework.org/schema/context/
        http://www.springframework.org/schema/context//spring-context.xsd">

    <!-- use any one of the below bean definition to activate/enable the
@Required annotation -->

    <!-- Method 1: -->
    <bean
class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPo
stProcessor" />

    <!-- or -->

    <!-- Method 2:
    <context:annotation-config/> -->

    <bean id="subjectBean" class="com.beans.SubjectBean">
        <!-- <property name="subName" value="Java Spring - Annotations" /> --
>
        <property name="subId" value="1002" />
    </bean>

</beans>
```

- Here, we are only setting the value of **subId.**
- Now, again run the project as a Java application. We will get the below error.

WARNING: Exception encountered during context initialization - cancelling refresh attempt: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'subjectBean' defined in class path resource [com/resource/applicationContext.xml]: Initialization of bean failed; nested exception is org.springframework.beans.factory.BeanInitializationException: ***Property 'subName' is required for bean 'subjectBean'***

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'subjectBean' defined in class path resource [com/resource/applicationContext.xml]: Initialization of bean failed; nested exception is org.springframework.beans.factory.BeanInitializationException: Property 'subName' is required for bean 'subjectBean'

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:587)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:501)

at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:317)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:315)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:199)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:760)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:869)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:550)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:144)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:85)

at com.test.SubjectBeanTest.main(SubjectBeanTest.java:12)

Caused by: org.springframework.beans.factory.BeanInitializationException: ***Property 'subName' is required for bean 'subjectBean'***

at org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor.postProcessPropertyValues(RequiredAnnotationBeanPostProcessor.java:156)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1344)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:578)

... 11 more

The container will check if the property **subName** is initialized or not. Here we are not setting the value. So, the container is throwing the Exception saying **"Property 'subName' is required for bean 'subjectBean'"**.

Conclusion: This way we can use the @Required annotation to the setter methods in the spring beans so that by giving the responsibility to the spring container of checking whether the required property values are being set.