Spring注解驱动开发第20讲——使用@Autowired、@Qualifier、@Primary这三大注解自动装配组件，你会了吗？

你知道@Autowired、@Qualifier、@Primary这些注解吗？

@Autowired注解

@Autowired注解可以对类 成员变量 、方法和构造函数进行标注，完成自动装配的工作。@Autowired注解可以放在类、接口以及方法上。

在使用@Autowired注解之前，我们对一个bean配置属性时，是用如下XML配置文件的形式进行配置的。

```
<property name="属性名" value=" 属性值"/>
```
AI写代码xml

下面我们来看一下@Autowired注解的源码，如下所示。

Autowired.class

```
 * Copyright 2002-2016 the original author or authors.

package org.springframework.beans.factory.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

 * Marks a constructor, field, setter method or config method as to be
@Target({ElementType.CONSTRUCTOR, ElementType.METHOD, ElementType.PARAMETER, ElementType.FIELD, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Autowired {

    /**
     * Declares whether the annotated dependency is required.
     * <p>Defaults to {@code true}.
     */
    boolean required() default true;

}
```

这儿对@Autowired注解说明一下：

1. @Autowired注解默认是优先按照类型去容器中找对应的组件，相当于是调用了如下这个方法：

   ```
   applicationContext.getBean(类名.class);
   ```
   AI写代码java运行

   若找到则就赋值。

2. 如果找到多个相同类型的组件，那么是将属性名称作为组件的id，到IOC容器中进行查找，这时就相当于是调用了如下这个方法：

   ```
   applicationContext.getBean("组件的id");
   ```
   AI写代码java运行

@Qualifier注解

@Autowired是根据类型进行 自动装配 的，如果需要按名称进行装配，那么就需要配合@Qualifier注解来使用了。

下面我们来看一下@Qualifier注解的源码，如下所示。

Autowired.class Qualifier.class

```java
 * Copyright 2002-2011 the original author or authors.

package org.springframework.beans.factory.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * This annotation may be used on a field or parameter as a qualifier for
 * candidate beans when autowiring. It may also be used to annotate other
 * custom annotations that can then in turn be used as qualifiers.
 *
 * @author Mark Fisher
 * @author Juergen Hoeller
 * @since 2.5
 * @see Autowired
 */
@Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER, ElementType.TYPE, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface Qualifier {

    String value() default "";

}
```

@Primary注解

在Spring中使用注解时，常常会使用到@Autowired这个注解，它默认是根据类型Type来自动注入的。但有些特殊情况，对同一个接口而言，可能会有几种不同的实现类，而在默认只会采取其中一种实现的情况下，就可以使用@Primary注解来标注优先使用哪一个实现类。

下面我们来看一下@Primary注解的源码，如下所示。

Autowired.class Qualifier.class Primary.class

```java
 * Copyright 2002-2016 the original author or authors.

package org.springframework.context.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

 * Indicates that a bean should be given preference when multiple candidates
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface Primary {

}
```

自动装配

在进行项目实战之前，我们先来说说什么是Spring组件的自动装配。Spring组件的自动装配就是**Spring利用依赖注入，也就是我们通常所说的DI，完成对IOC容器中各个组件的依赖关系赋值。**

项目实战

测试@Autowired注解

这里，我们以之前项目中创建的BookDao、BookService和BookController为例进行说明。BookDao、BookService和BookController的初始代码分别如下所示。

- BookDao

```java
package com.meimeixia.dao;

import org.springframework.stereotype.Repository;

// 名字默认是类名首字母小写
@Repository
public class BookDao {

```

```
}
```

AI写代码java运行

- BookService

```java
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Autowired
    private BookDao bookDao;

    public void print() {
        System.out.println(bookDao);
    }

}
```

AI写代码java运行

- BookController

```java
package com.meimeixia.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import com.meimeixia.service.BookService;

@Controller
public class BookController {

    @Autowired
    private BookService bookService;

}
```

AI写代码java运行

可以看到，我们在BookService中使用@Autowired注解注入了BookDao，在BookController中使用@Autowired注解注入了BookService。为了方便测试，我们可以在BookService类中生成一个toString()方法，如下所示。

```java
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Autowired
    private BookDao bookDao;

    public void print() {
        System.out.println(bookDao);
    }

    @Override
    public String toString() {
        return "BookService [bookDao=" + bookDao + "]";
    }

```

```
}
```

AI写代码java运行

为了更好的看到演示效果，我们在项目的com.meimeixia.config包下创建一个配置类，例如MainConfigOfAutowired，如下所示。

```java
package com.meimeixia.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 *
 * @author liayun
 *
 */
@Configuration
@ComponentScan({"com.meimeixia.service", "com.meimeixia.dao", "com.meimeixia.controller"})
public class MainConfigOfAutowired {

}
```

AI写代码java运行

接下来，我们便来测试一下上面的程序。在项目的src/test/java目录下的com.meimeixia.test包中创建一个单元测试类，例如IOCTest_Autowired，如下所示。

```java
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.config.MainConfigOfAutowired;
import com.meimeixia.service.BookService;

public class IOCTest_Autowired {

    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfAutowired.class);

        BookService bookService = applicationContext.getBean(BookService.class);
        System.out.println(bookService);

        applicationContext.close();
    }

}
```

AI写代码java运行

测试方法比较简单，这里我就不做过多说明了。然后，我们运行一下IOCTest_Autowired类中的test01()方法，得出的输出结果信息如下所示。

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Problems  Console  Progress  Search  Maven Repositories  Synchronize  JUnit
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午7:26:56)
十二月 02, 2020 7:26:57 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prep
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: star
BookService [bookDao=com.meimeixia.dao.BookDao@4de5031f]
十二月 02, 2020 7:26:57 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doC
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup
```


可以看到，输出了BookDao信息。

那么问题来了，我们在BookService类中使用@Autowired注解注入的BookDao（最后输出了该BookDao的信息），和我们直接在Spring IOC容器中获取的BookDao是不是同一个对象呢？

为了说明这一点，我们可以在IOCTest_Autowired类的test01()方法中添加获取BookDao对象的方法，并输出获取到的BookDao对象，如下所示。

```java
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.config.MainConfigOfAutowired;
import com.meimeixia.dao.BookDao;
import com.meimeixia.service.BookService;

public class IOCTest_Autowired {

    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfAutowired.class);

        BookService bookService = applicationContext.getBean(BookService.class);
        System.out.println(bookService);

        BookDao bookDao = applicationContext.getBean(BookDao.class);
        System.out.println(bookDao);

        applicationContext.close();
    }

}
```

AI写代码java运行

我们再次运行以上test01()方法，输出的结果信息如下所示。

```
Markers Properties Servers Data Source Explorer Snippets Problems Console Progress Search Maven Repositories Synchronize JUnit
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午7:32:58)
十二月 02, 2020 7:32:58 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prep
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: star
BookService [bookDao=com.meimeixia.dao.BookDao@4de5031f]
com.meimeixia.dao.BookDao@4de5031f
十二月 02, 2020 7:32:58 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doC
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup
```

可以看到，我们在BookService类中使用@Autowired注解注入的BookDao对象和直接从IOC容器中获取的BookDao对象是同一个对象。

你可能会问了，**如果在Spring容器中存在对多个BookDao对象，那么这时又该如何处理呢？**

首先，为了更加直观的看到我们使用@Autowired注解装配的是哪个BookDao对象，我们得对BookDao类进行改造，为其加上一个lable字段，并为其赋一个默认值，如下所示。

```java
package com.meimeixia.dao;

import org.springframework.stereotype.Repository;

// 名字默认是类名首字母小写
@Repository
public class BookDao {

    private String lable = "1";

    public String getLable() {
        return lable;
    }

    public void setLable(String lable) {
        this.lable = lable;
    }

    @Override
    public String toString() {

```

```
        return "BookDao [lable=" + lable + "]";
    }

}
```

AI写代码java运行

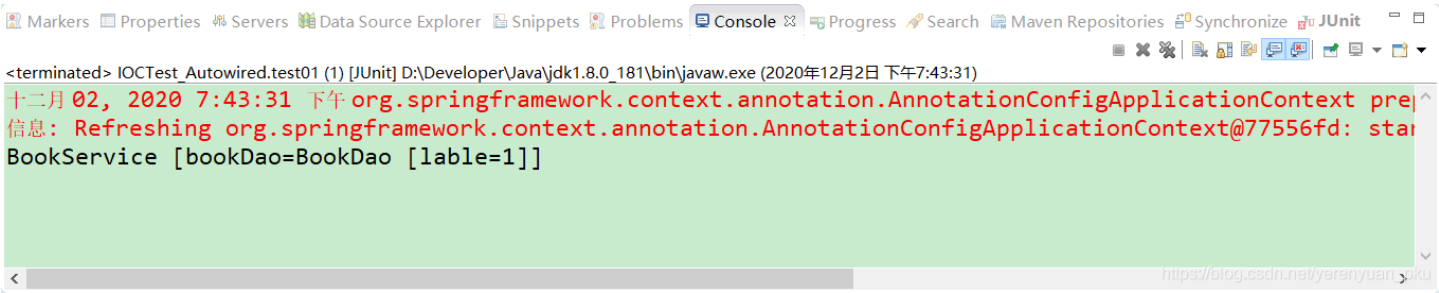
然后，我们就在MainConfigOfAutowired配置类中注入一个BookDao对象，并且显示指定该对象在IOC容器中的bean的名称为bookDao2，并还为该对象的lable字段赋值为2，如下所示。

```
package com.meimeixia.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.meimeixia.dao.BookDao;

/**
 *
 * @author liayun
 *
 */
@Configuration
@ComponentScan({"com.meimeixia.service", "com.meimeixia.dao", "com.meimeixia.controller"})
public class MainConfigOfAutowired {

    @Bean("bookDao2")
    public BookDao bookDao() {
        BookDao bookDao = new BookDao();
        bookDao.setLable("2");
        return bookDao;
    }

}
```

AI写代码java运行

目前，在我们的IOC容器中就会注入两个BookDao对象。那此时，**@Autowired注解到底装配的是哪个BookDao对象呢？**

接着，我们来运行一下IOCTest_Autowired类中的test01()方法，发现输出的结果信息如下所示。



可以看到，结果信息输出了 `lable=1` ，这说明，**@Autowired注解默认是优先按照类型去容器中找对应的组件，找到就赋值；如果找到多个相同类型的组件，那么再将属性的名称作为组件的id，到IOC容器中进行查找。**

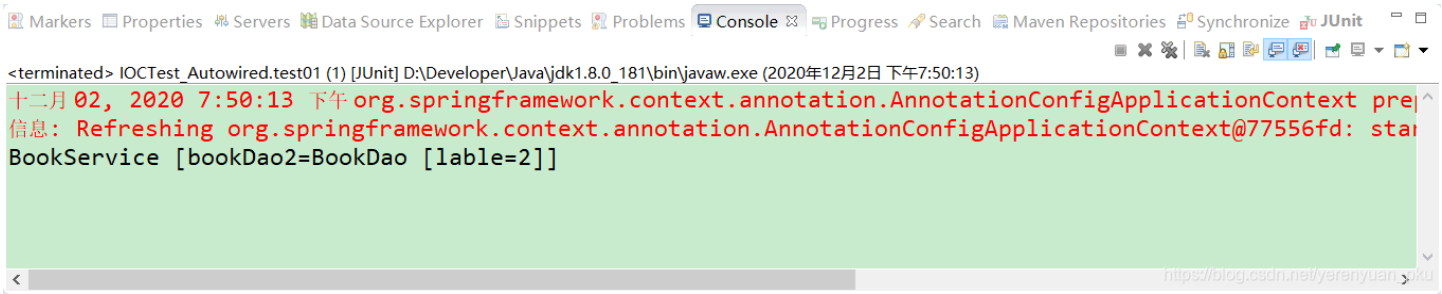
那我们如何让@Autowired注解装配bookDao2呢？ 这个问题问的好，其实很简单，我们只须将BookService类中的bookDao属性的名称全部修改为bookDao2即可，如下所示。

```
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Autowired
    private BookDao bookDao2;

```

```
    public void print() {
        System.out.println(bookDao2);
    }

    @Override
    public String toString() {
        return "BookService [bookDao2=" + bookDao2 + "]";
    }

}
```

AI写代码java运行

此时，我们再运行IOCTest_Autowired类中的test01()方法，输出的结果信息如下所示。



可以看到，此时Eclipse 控制台中输出了bookDao2的信息。

测试@Qualifier注解

从测试@Autowired注解的结果来看，**@Autowired注解默认是优先按照类型去容器中找对应的组件，找到就赋值；如果找到多个相同类型的组件，那么再将属性的名称作为组件的id，到IOC容器中进行查找。**

如果IOC容器中存在多个相同类型的组件时，那么我们可不可以显示指定@Autowired注解装配哪个组件呢？有些小伙伴肯定会说：废话！你都这么问了，那肯定可以啊！没错，确实是可以的！此时，@Qualifier注解就派上用场了！

在之前的测试案例中，Eclipse控制台中输出了 `BookDao [lable=2]` ，这说明@Autowired注解装配了bookDao2，那我们如何显示的让@Autowired注解装配bookDao呢？

比较简单，我们只需要在BookService类里面的bookDao2字段上添加@Qualifier注解，显示指定@Autowired注解装配bookDao即可，如下所示。

```
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Qualifier("bookDao")
    @Autowired
    private BookDao bookDao2;

    public void print() {
        System.out.println(bookDao2);
    }

    @Override
    public String toString() {
        return "BookService [bookDao2=" + bookDao2 + "]";
    }

}
```

AI写代码java运行

此时，我们再次运行IOCTest_Autowired类中的test01()方法，输出的结果信息如下所示。

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Problems  Console  Progress  Search  Maven Repositories  Synchronize  JUnit
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午8:01:35)
十二月 02, 2020 8:01:36 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRe
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup d
BookService [bookDao2=BookDao [lable=1]]
```

可以看到，此时尽管字段的名称为bookDao2，但是我们使用了@Qualifier注解显示指定了@Autowired注解装配bookDao对象，所以，最终的结果中输出了bookDao对象的信息。

测试容器中无组件的情况

如果IOC容器中无相应的组件，那么会发生什么情况呢？这时我们可以做这样一件事情，先注释掉BookDao类上的@Repository注解，

```java
package com.meimeixia.dao;

import org.springframework.stereotype.Repository;

// 名字默认是类名首字母小写
//@Repository
public class BookDao {

    private String lable = "1";

    public String getLable() {
        return lable;
    }

    public void setLable(String lable) {
        this.lable = lable;
    }

    @Override
    public String toString() {
        return "BookDao [lable=" + lable + "]";
    }

}
```

AI写代码java运行

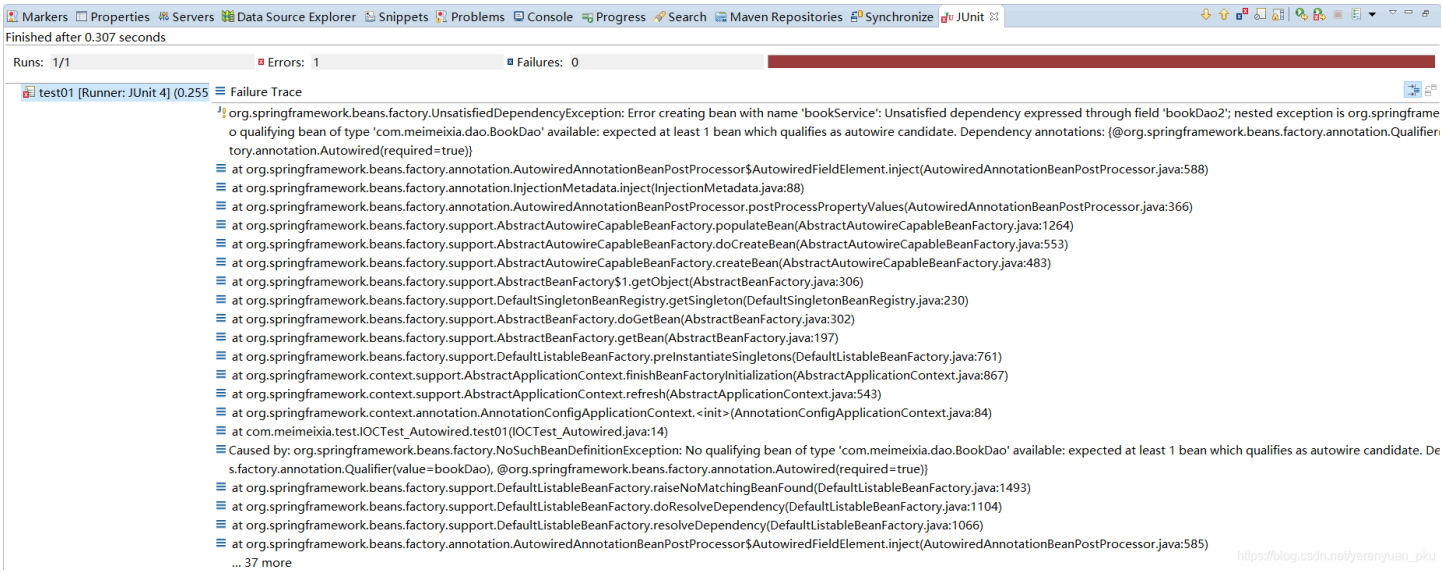
然后再注释掉MainConfigOfAutowired配置类中的bookDao()方法上的@Bean注解，如下所示。

```java
package com.meimeixia.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.meimeixia.dao.BookDao;

/**
 *
 * @author liayun
 *
 */
@Configuration
@ComponentScan({"com.meimeixia.service", "com.meimeixia.dao", "com.meimeixia.controller"})
public class MainConfigOfAutowired {

//  @Bean("bookDao2")
    public BookDao bookDao() {
        BookDao bookDao = new BookDao();
        bookDao.setLable("2");
        return bookDao;
    }

}
```

AI写代码java运行

此时IOC容器中不再有任何BookDao对象了。

接着，我们再次运行IOCTest_Autowired类中的test01()方法，发现Eclipse控制台报了一个错误，截图如下。



详细的错误信息如下：

```
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'bookService': Unsatisfied dependency
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotat
    at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:88)
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotation
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.ja
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.ja
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java
    at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:306)
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:230)
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:302)
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:197)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:7
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:543)
    at org.springframework.context.annotation.AnnotationConfigApplicationContext.<init>(AnnotationConfigApplicationContext.java:84)
    at com.meimeixia.test.IOCTest_Autowired.test01(IOCTest_Autowired.java:14)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
    at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
    at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:675)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'com.meimeixia.dao.BookDao' ava
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.raiseNoMatchingBeanFound(DefaultListableBeanFactory.java:1
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1104)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1066)
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotat
    ... 37 more
```

AI写代码java运行

此时，Spring抛出了异常，未找到相应的bean对象，那我们能不能让Spring不报错呢？那肯定可以啊！抛出的异常信息中都给出了相应的提示。

```
{@org.springframework.beans.factory.annotation.Qualifier(value=bookDao), @org.springframework.beans.factory.annotation.Autowired(requi
```
AI写代码java运行

解决方案就是在BookService类的@Autowired注解里面添加一个属性 `required=false` ，如下所示。

```java
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Qualifier("bookDao")
    @Autowired(required=false)
    private BookDao bookDao2;

    public void print() {
        System.out.println(bookDao2);
    }

    @Override
    public String toString() {
        return "BookService [bookDao2=" + bookDao2 + "]";
    }

}
```
AI写代码java运行

加上 `required=false` 这个玩意的意思就是说找到就装配，找不到就拉到，就别装配了。

此时，还需要将IOCTest_Autowired类的test01()方法中直接从IOC容器中获取BookDao对象的代码注释掉，如下所示。

```java
package com.meimeixia.test;

import org.junit.Test;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.meimeixia.config.MainConfigOfAutowired;
import com.meimeixia.dao.BookDao;
import com.meimeixia.service.BookService;

public class IOCTest_Autowired {

    @Test
    public void test01() {
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfAutowired.class);

        BookService bookService = applicationContext.getBean(BookService.class);
        System.out.println(bookService);

//      BookDao bookDao = applicationContext.getBean(BookDao.class);
//      System.out.println(bookDao);

        applicationContext.close();
    }

}
```
AI写代码java运行

紧接着，我们再次运行以上test01()方法，输出的结果信息如下所示。

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Problems  Console  Progress  Search  Maven Repositories  Synchronize  JUnit
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午8:23:03)
十二月 02, 2020 8:23:03 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRe
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup d
BookService [bookDao2=null]
十二月 02, 2020 8:23:03 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup date
```

可以看到，当为@Autowired注解添加属性 `required=false` 后，即使IOC容器中没有对应的对象，Spring也不会抛出异常了。不过，此时装配的对象就为null了。

测试完成后，我们还得恢复原样，即再次为BookDao类添加@Repository注解，并且在MainConfigOfAutowired配置类中的bookDao()方法上添加@Bean注解，好方便进一步的测试。

测试@Primary注解

在Spring中，对同一个接口而言，可能会有几种不同的实现类，而默认只会采取其中一种实现的情况下，就可以使用@Primary注解来标注优先使用哪一个实现类。

如果IOC容器中相同类型的组件有多个，那么我们不可避免地就要来回用@Qualifier注解来指定要装配哪个组件，这还是比较麻烦的，Spring正是帮我们考虑到了这样一种情况，就提供了这样一个比较强大的注解，即@Primary。我们可以利用这个注解让Spring进行自动装配的时候，默认使用首选的bean。

说了这么多，下面我们就用一个小例子来测试一下@Primary注解。

首先，我们在MainConfigOfAutowired配置类的bookDao()方法上添加上@Primary注解，如下所示。

```java
package com.meimeixia.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

import com.meimeixia.dao.BookDao;

/**
 *
 * @author liayun
 *
 */
@Configuration
@ComponentScan({"com.meimeixia.service", "com.meimeixia.dao", "com.meimeixia.controller"})
public class MainConfigOfAutowired {

    @Primary
    @Bean("bookDao2")
    public BookDao bookDao() {
        BookDao bookDao = new BookDao();
        bookDao.setLable("2");
        return bookDao;
    }

}
```

AI写代码java运行

注意：此时，我们需要注释掉BookService类中bookDao字段上的@Qualifier注解，这是因为@Qualifier注解为显示指定装配哪个组件，如果使用了@Qualifier注解，无论是否使用了@Primary注解，都会装配@Qualifier注解标注的对象。

```java
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

//  @Qualifier("bookDao") // 要让首选装配起效果，@Qualifier自然就不能用了
    @Autowired(required=false)

```

```
    private BookDao bookDao;

    public void print() {
        System.out.println(bookDao);
    }

    @Override
    public String toString() {
        return "BookService [bookDao=" + bookDao + "]";
    }

}
```

AI写代码java运行

设置完成后，我们再次运行IOCTest_Autowired类中的test01()方法，输出的结果信息如下所示。

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Problems  Console  Progress  Search  Maven Repositories  Synchronize  JUnit
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午8:43:10)
十二月 02, 2020 8:43:10 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRe
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup d
BookService [bookDao=BookDao [lable=2]]
十二月 02, 2020 8:43:10 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup date
```

可以看到，此时lable的值为2，这说明装配了MainConfigOfAutowired配置类中注入的bookDao2。

那我们非要装配bookDao，可不可以呢？当然可以了，我们只须使用 `@Qualifier("bookDao")` 来显示指定装配bookDao即可。也就是说如果是在没有明确指定的情况下，那么就装配优先级最高的首选的那个bean，如果是在明确指定了的情况下，那么自然就是装配指定的那个bean了。

因此，我们可以为BookService类中的bookDao字段再次添加@Qualifier注解，如下所示。

```
package com.meimeixia.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import com.meimeixia.dao.BookDao;

@Service
public class BookService {

    @Qualifier("bookDao")
    @Autowired(required=false)
    private BookDao bookDao;

    public void print() {
        System.out.println(bookDao);
    }

    @Override
    public String toString() {
        return "BookService [bookDao=" + bookDao + "]";
    }

}
```

AI写代码java运行

此时，我们再次运行IOCTest_Autowired类中的test01()方法，输出的结果信息如下所示。

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress Search Maven Repositories Synchronize JUnit

```
<terminated> IOCTest_Autowired.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午8:53:28)
十二月 02, 2020 8:53:29 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRe
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup d
BookService [bookDao=BookDao [lable=1]]
十二月 02, 2020 8:53:29 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: startup date
```


可以看到，此时Spring装配了使用@Qualifier显示指定的需要装配的bookDao。