

Spring注解驱动开发第11讲——面试官让我说说：如何使用FactoryBean向Spring容器中注册bean?

写在前面

经过前面的学习，我们知道可以通过多种方式向Spring容器中注册bean。可以使用@Configuration注解结合@Bean注解向Spring容器中注册bean；可以按照条件向Spring容器中注册bean；可以使用@Import注解向容器中快速导入bean对象；可以在@Import注解中使用ImportBeanDefinitionRegistrar向容器中注册bean。

而在本文中，我就来讲讲如何使用FactoryBean向Spring容器中注册bean。

FactoryBean概述

一般情况下，Spring是通过 **反射机制** 利用bean的class属性指定实现类来实例化bean的。在某些情况下，实例化bean过程比较复杂，如果按照传统的方式，那么则需要 在标签中提供大量的配置信息，配置方式的灵活性是受限的，这时采用编码的方式可以得到一个更加简单的方案。Spring为此提供了一个 org.springframework.beans.factory.FactoryBean 的工厂类接口，用户可以通过实现该接口定制 **实例化** bean的逻辑。

FactoryBean接口对于Spring框架来说占有非常重要的地位，Spring自身就提供了70多个FactoryBean接口的实现。它们隐藏了实例化一些复杂bean的细节，给上层应用带来了便利。从Spring 3.0开始，FactoryBean开始支持泛型，即接口声明改为 **FactoryBean<T>** 的形式。

在 **Spring 4.3.12.RELEASE** 这个版本中，FactoryBean接口的定义如下所示。

```
2+ * Copyright 2002-2016 the original author or authors.
16
17 package org.springframework.beans.factory;
18
20+ * Interface to be implemented by objects used within a {@link BeanFactory} which
56 public interface FactoryBean<T> {
57
59+     * Return an instance (possibly shared or independent) of the object
75     T getObject() throws Exception;
76
78+     * Return the type of object that this FactoryBean creates,
96     Class<?> getObjectType();
97
99+     * Is the object managed by this factory a singleton? That is,
121     boolean isSingleton();
122
123 }
124
```

- T getObject(): 返回由FactoryBean创建的bean实例，如果isSingleton()返回true，那么该实例会放到Spring容器中单实例缓存池中
- boolean isSingleton(): 返回由FactoryBean创建的bean实例的作用域是singleton还是prototype
- Class getObjectType(): 返回FactoryBean创建的bean实例的类型

这里，需要注意的是：当配置文件中标签的class属性配置的实现类是FactoryBean时，通过 getBean()方法返回的不是FactoryBean本身，而是 **FactoryBean#getObject()**方法所返回的对象，相当于FactoryBean#getObject()代理了getBean()方法。

FactoryBean案例

首先，创建一个ColorFactoryBean类，它得实现FactoryBean接口，如下所示。

```
1 package com.meimeixia.bean;
2
3 import org.springframework.beans.factory.FactoryBean;
4
5 /**
6  * 创建一个Spring定义的FactoryBean
7  * T (泛型): 指定我们要创建什么类型的对象
8  * @author Liayun
9  *
10 */
11 public class ColorFactoryBean implements FactoryBean<Color> {
12
13     // 返回一个Color对象，这个对象会添加到容器中
14     @Override
15     public Color getObject() throws Exception {
16         // TODO Auto-generated method stub
17         System.out.println("ColorFactoryBean...getObject...");
18         return new Color();
19     }
20 }
```

```
20
21
22     @Override
23     public Class<?> getObjectType() {
24         // TODO Auto-generated method stub
25         return Color.class; // 返回这个对象的类型
26     }
27
28     // 是单例吗?
29     // 如果返回true, 那么代表这个bean是单实例, 在容器中只会保存一份;
30     // 如果返回false, 那么代表这个bean是多实例, 每次获取都会创建一个新的bean
31     @Override
32     public boolean isSingleton() {
33         // TODO Auto-generated method stub
34         return false;
35     }
36 }
```

AI写代码java运行



然后，我们在MainConfig2配置类中加入ColorFactoryBean的声明，如下所示。

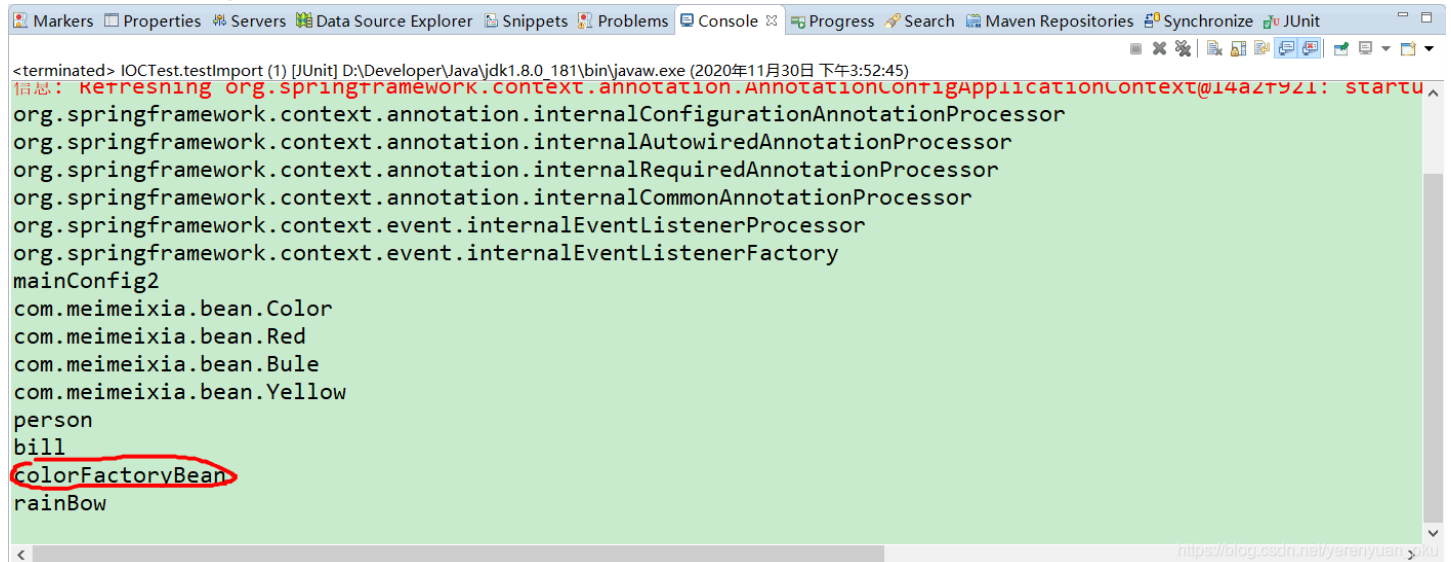
```
1  package com.meimeixia.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Conditional;
5  import org.springframework.context.annotation.Configuration;
6  import org.springframework.context.annotation.Import;
7  import org.springframework.context.annotation.Lazy;
8
9  import com.meimeixia.bean.Color;
10 import com.meimeixia.bean.ColorFactoryBean;
11 import com.meimeixia.bean.Person;
12 import com.meimeixia.bean.Red;
13 import com.meimeixia.condition.LinuxCondition;
14 import com.meimeixia.condition.MyImportBeanDefinitionRegistrar;
15 import com.meimeixia.condition.MyImportSelector;
16 import com.meimeixia.condition.WindowsCondition;
17
18 // 对配置类中的组件进行统一设置
19 @Conditional({WindowsCondition.class}) // 满足当前条件, 这个类中配置的所有bean注册才能生效
20 @Configuration
21 @Import({Color.class, Red.class, MyImportSelector.class, MyImportBeanDefinitionRegistrar.class}) // @Import快速地导入组件, id默认是组件的全
22 public class MainConfig2 {
23
24     @Lazy
25     @Bean("person")
26     public Person person() {
27         System.out.println("给容器中添加咱们这个Person对象...");
28         return new Person("美美侠", 25);
29     }
30
31     @Bean("bill")
32     public Person person01() {
33         return new Person("Bill Gates", 62);
34     }
35
36     @Conditional({LinuxCondition.class})
37     @Bean("linus")
38     public Person person02() {
39         return new Person("linus", 48);
40     }
41
42     @Bean
43     public ColorFactoryBean colorFactoryBean() {
44         return new ColorFactoryBean();
45     }
46
47 }
```

AI写代码java运行



这里需要小伙伴们注意的是：我在这里使用@Bean注解向Spring容器中注册的是ColorFactoryBean对象。

那现在我们就来看看Spring容器中到底都有哪些bean。我们所要做的事情就是，运行IOCTest类中的testImport()方法，此时，输出的结果信息如下所示。



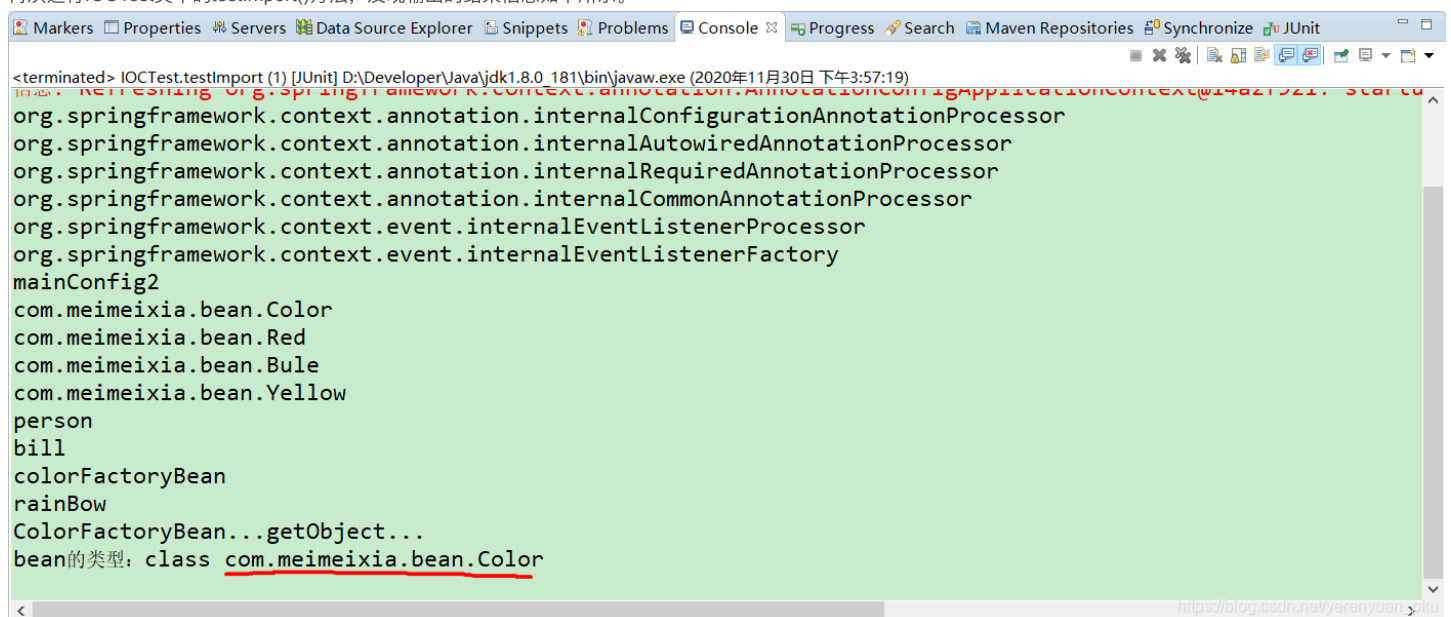
```
<terminated> IOCTest.testImport (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月30日 下午3:52:45)
org.springframework.context.annotation.AnnotationConfigApplicationContext@14a2f921: startup
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig2
com.meimeixia.bean.Color
com.meimeixia.bean.Red
com.meimeixia.bean.Bule
com.meimeixia.bean.Yellow
person
bill
colorFactoryBean
rainBow
```

可以看到，结果信息中输出了一个colorFactoryBean，我们看下这个colorFactoryBean到底是个什么鬼！此时，我们对IOCTest类中的testImport()方法稍加改动，添加获取colorFactoryBean的代码，并输出colorFactoryBean实例的类型，如下所示。

```
1 @Test
2 public void testImport() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     String[] definitionNames = applicationContext.getBeanDefinitionNames();
5     for (String name : definitionNames) {
6         System.out.println(name);
7     }
8
9     // 工厂bean获取的是调用getObject方法创建的对象
10    Object bean2 = applicationContext.getBean("colorFactoryBean");
11    System.out.println("bean的类型: " + bean2.getClass());
12 }
13 }
```

AI写代码java运行

再次运行IOCTest类中的testImport()方法，发现输出的结果信息如下所示。



```
<terminated> IOCTest.testImport (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月30日 下午3:57:19)
org.springframework.context.annotation.AnnotationConfigApplicationContext@14a2f921: startup
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig2
com.meimeixia.bean.Color
com.meimeixia.bean.Red
com.meimeixia.bean.Bule
com.meimeixia.bean.Yellow
person
bill
colorFactoryBean
rainBow
ColorFactoryBean...getObject...
bean的类型: class com.meimeixia.bean.Color
```

可以看到，虽然我在代码中使用@Bean注解注入的是ColorFactoryBean对象，但是实际上从Spring容器中获取到的bean对象却是调用ColorFactoryBean类中的getObject()方法获取到的Color对象。

看到这里，是不是有种豁然开朗的感觉！！

在ColorFactoryBean类中，我们将Color对象设置为单实例bean，即让isSingleton()方法返回true。接下来，我们在IOCTest类中的testImport()方法里面多次获取Color对象，并判断一下多次获取的对象是否为同一对象，如下所示。

```

1  @Test
2  public void testImport() {
3      AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4      String[] definitionNames = applicationContext.getBeanDefinitionNames();
5      for (String name : definitionNames) {
6          System.out.println(name);
7      }
8
9      // 工厂bean获取的是调用getObject方法创建的对象
10     Object bean2 = applicationContext.getBean("colorFactoryBean");
11     Object bean3 = applicationContext.getBean("colorFactoryBean");
12     System.out.println("bean的类型: " + bean2.getClass());
13     System.out.println(bean2 == bean3);
14 }

```

AI写代码java运行

然后，运行IOCTest类中的testImport()方法，此时输出的结果信息如下所示。

```

<terminated> IOCTest.testImport (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月30日 下午4:04:15)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@14a2f921: startup
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig2
com.meimeixia.bean.Color
com.meimeixia.bean.Red
com.meimeixia.bean.Bule
com.meimeixia.bean.Yellow
person
bill
colorFactoryBean
rainBow
ColorFactoryBean...getObject...
bean的类型: class com.meimeixia.bean.Color
true

```

可以看到，在ColorFactoryBean类中的isSingleton()方法里面返回true时，每次获取到的Color对象都是同一个对象，说明Color对象是单实例bean。

这里，可能就会有小伙伴要问了，如果将Color对象修改成多实例bean呢？别急，这里我们只需要在ColorFactoryBean类的isSingleton()方法中返回false即可，这样就会将Color对象设置为多实例bean，如下所示。

```

1  package com.meimeixia.bean;
2
3  import org.springframework.beans.factory.FactoryBean;
4
5  /**
6   * 创建一个Spring定义的FactoryBean
7   * T (泛型): 指定我们要创建什么类型的对象
8   * @author liayun
9   *
10  */
11  public class ColorFactoryBean implements FactoryBean<Color> {
12
13      // 返回一个Color对象，这个对象会添加到容器中
14      @Override
15      public Color getObject() throws Exception {
16          // TODO Auto-generated method stub
17          System.out.println("ColorFactoryBean...getObject...");
18          return new Color();
19      }
20
21      @Override
22      public Class<?> getObjectType() {
23          // TODO Auto-generated method stub

```

```

24         return Color.class; // 返回这个对象的类型
25     }
26
27     // 是单例吗?
28     // 如果返回true, 那么代表这个bean是单实例, 在容器中只会保存一份;
29     // 如果返回false, 那么代表这个bean是多实例, 每次获取都会创建一个新的bean
30     @Override
31     public boolean isSingleton() {
32         // TODO Auto-generated method stub
33         return false;
34     }
35
36 }

```

AI写代码java运行



接着，再次运行IOCTest类中的testImport()方法，会发现输出的结果信息如下所示。

```

<terminated> IOCTest.testImport(1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月30日 下午4:08:41)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@14a2f921: startup
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig2
com.meimeixia.bean.Color
com.meimeixia.bean.Red
com.meimeixia.bean.Bule
com.meimeixia.bean.Yellow
person
bill
colorFactoryBean
rainBow
ColorFactoryBean...getObject...
ColorFactoryBean...getObject...
bean的类型: class com.meimeixia.bean.Color
false

```

可以看到，最终结果返回了false，说明此时Color对象是多实例bean。

如何在Spring容器中获取到FactoryBean对象本身呢？

之前，我们使用@Bean注解向Spring容器中注册的是ColorFactoryBean，获取出来的却是Color对象。那么，小伙伴们可能会问了，我就想获取ColorFactoryBean实例，那该怎么办呢？

其实，这也很简单，只需要在获取工厂Bean本身时，在id前面加上&符号即可，例如&colorFactoryBean。

打开我们的IOCTest测试类，在testImport()方法中添加获取ColorFactoryBean实例的代码，如下所示。

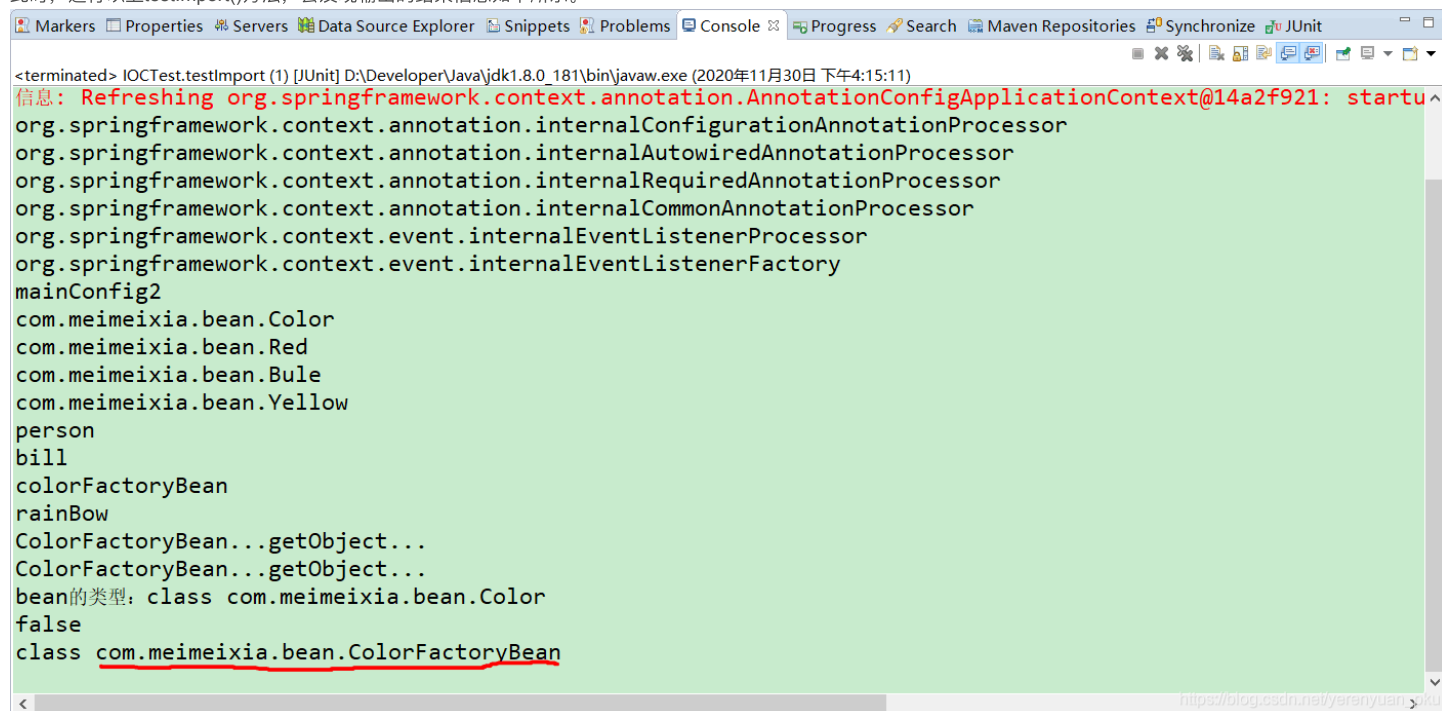
```

1 public void testImport() {
2     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
3     String[] definitionNames = applicationContext.getBeanDefinitionNames();
4     for (String name : definitionNames) {
5         System.out.println(name);
6     }
7
8     // 工厂bean获取的是调用getObject方法创建的对象
9     Object bean2 = applicationContext.getBean("colorFactoryBean");
10    Object bean3 = applicationContext.getBean("colorFactoryBean");
11    System.out.println("bean的类型: " + bean2.getClass());
12    System.out.println(bean2 == bean3);
13
14    Object bean4 = applicationContext.getBean("&colorFactoryBean");
15    System.out.println(bean4.getClass());
16 }

```

AI写代码java运行

此时，运行以上testImport()方法，会发现输出的结果信息如下所示。

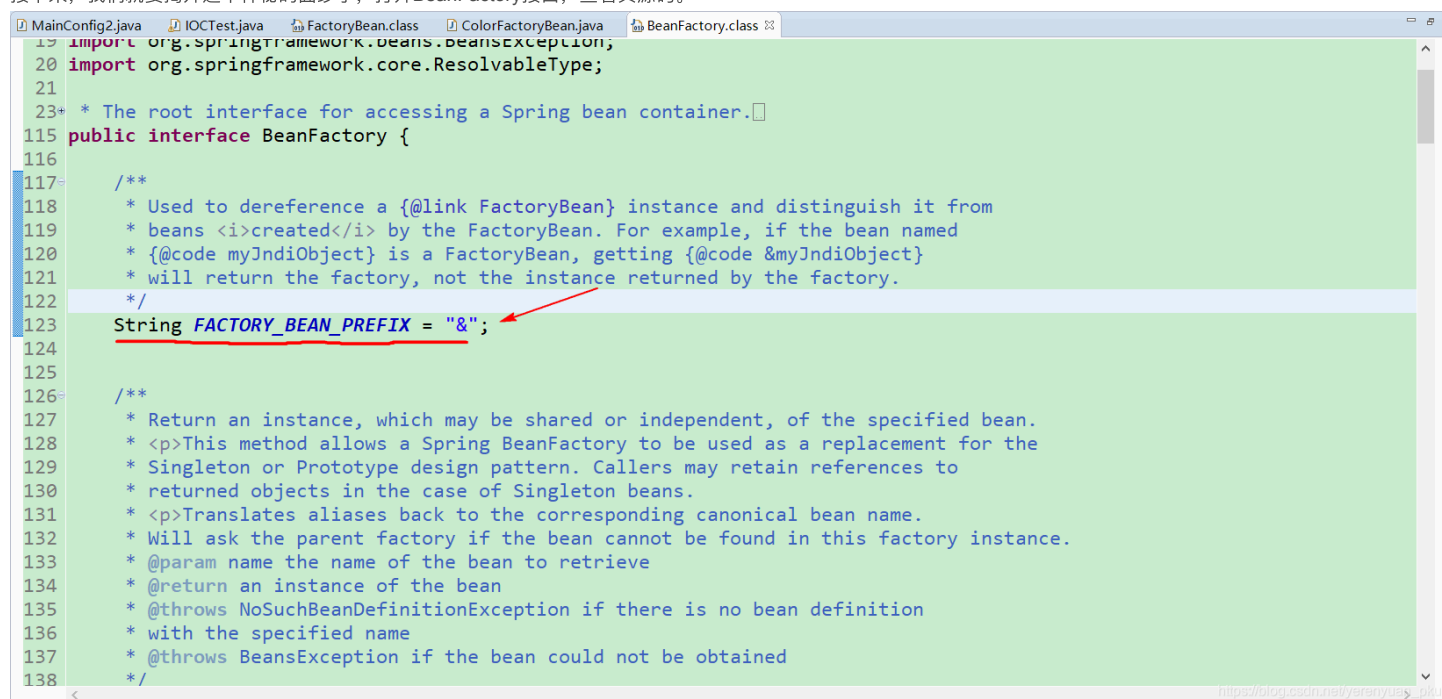


```
<terminated> IOCTest.testImport (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月30日 下午4:15:11)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@14a2f921: startup
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.event.internalEventListenerProcessor
org.springframework.context.event.internalEventListenerFactory
mainConfig2
com.meimeixia.bean.Color
com.meimeixia.bean.Red
com.meimeixia.bean.Bule
com.meimeixia.bean.Yellow
person
bill
colorFactoryBean
rainBow
ColorFactoryBean...getObject...
ColorFactoryBean...getObject...
bean的类型: class com.meimeixia.bean.Color
false
class com.meimeixia.bean.ColorFactoryBean
```

可以看到，在获取bean时，在id前面加上&符号就会获取到ColorFactoryBean实例对象。

那问题又来了！！为什么在id前面加上&符号就会获取到ColorFactoryBean实例对象呢？

接下来，我们就要揭开这个神秘的面纱了，打开BeanFactory接口，查看其源码。



```
19 import org.springframework.beans.BeanException;
20 import org.springframework.core.ResolvableType;
21
22 * The root interface for accessing a Spring bean container.
23
115 public interface BeanFactory {
116
117     /**
118      * Used to dereference a {@link FactoryBean} instance and distinguish it from
119      * beans <i>created</i> by the FactoryBean. For example, if the bean named
120      * {@code myJndiObject} is a FactoryBean, getting {@code &myJndiObject}
121      * will return the factory, not the instance returned by the factory.
122      */
123     String FACTORY_BEAN_PREFIX = "&";
124
125
126     /**
127      * Return an instance, which may be shared or independent, of the specified bean.
128      * <p>This method allows a Spring BeanFactory to be used as a replacement for the
129      * Singleton or Prototype design pattern. Callers may retain references to
130      * returned objects in the case of Singleton beans.
131      * <p>Translates aliases back to the corresponding canonical bean name.
132      * Will ask the parent factory if the bean cannot be found in this factory instance.
133      * @param name the name of the bean to retrieve
134      * @return an instance of the bean
135      * @throws NoSuchBeanDefinitionException if there is no bean definition
136      * with the specified name
137      * @throws BeansException if the bean could not be obtained
138      */
```

看到这里，是不是明白了呢？没错，在BeanFactory接口中定义了一个&前缀，只要我们使用bean的id来从Spring容器中获取bean时，Spring就会知道我们是在获取FactoryBean本身。