

# Spring注解驱动开发第5讲——使用@Scope注解设置组件的作用域

## 写在前面

Spring容器中的组件默认是单例的，在Spring启动时就会 **实例化** 并初始化这些对象，并将其放到Spring容器中，之后，每次获取对象时，直接从Spring容器中获取，而不再创建对象。如果每次从Spring容器中获取对象时，都要创建一个新的实例对象，那么该如何处理呢？此时就需要使用@Scope注解来设置组件的作用域了。

## 本文内容概览

咱先看看在本文中我们要学习哪些知识点，从而从全局角度把握好方向。本文要学习的知识点疏览如下：

- @Scope注解概述
- 单实例bean作用域
- 多实例bean作用域
- 单实例bean作用域如何创建对象？
- 多实例bean作用域如何创建对象？
- 单实例bean注意的事项
- 多实例bean注意的事项
- 自定义Scope的实现

## @Scope注解概述

@Scope注解能够设置组件的作用域，我们先来看看@Scope注解类的源码，如下所示。

```

* Copyright 2002-2015 the original author or authors.

package org.springframework.context.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;

import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.core.annotation.AliasFor;

/**
 * When used as a type-level annotation in conjunction with
 * {@link org.springframework.stereotype.Component @Component},
 * {@code @Scope} indicates the name of a scope to use for instances of
 * the annotated type.
 *
 * <p>When used as a method-level annotation in conjunction with
 * {@link Bean @Bean}, {@code @Scope} indicates the name of a scope to use
 * for the instance returned from the method.
 *
 * <p>In this context, <em>scope</em> means the lifecycle of an instance,
 * such as {@code singleton}, {@code prototype}, and so forth. Scopes
 * provided out of the box in Spring may be referred to using the
 * {@code SCOPE_*} constants available in the {@link ConfigurableBeanFactory}
 * and {@link WebApplicationContext} interfaces.
 *
 * <p>To register additional custom scopes, see
 * {@link org.springframework.beans.factory.config.CustomScopeConfigurer
 * CustomScopeConfigurer}.
 *
 * @author Mark Fisher
 * @author Chris Beams
 * @author Sam Brannen
 * @since 2.5
 * @see org.springframework.stereotype.Component
 * @see org.springframework.context.annotation.Bean
 */
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Scope {

    /**
     * Alias for {@link #scopeName}.
     * @see #scopeName
     */
    @AliasFor("scopeName")
    String value() default "";

    /**
     * Specifies the name of the scope to use for the annotated component/bean.
     * <p>Defaults to an empty string ({@code ""}) which implies
     * {@link ConfigurableBeanFactory#SCOPE_SINGLETON SCOPE_SINGLETON}.
     * @since 4.2
     * @see ConfigurableBeanFactory#SCOPE_PROTOTYPE
     * @see ConfigurableBeanFactory#SCOPE_SINGLETON
     * @see org.springframework.web.context.WebApplicationContext#SCOPE_REQUEST
     * @see org.springframework.web.context.WebApplicationContext#SCOPE_SESSION
     * @see #value
     */
    @AliasFor("value")
    String scopeName() default "";
}

```

```

/**
 * Specifies whether a component should be configured as a scoped proxy
 * and if so, whether the proxy should be interface-based or subclass-based.
 * <p>Defaults to {@link ScopedProxyMode#DEFAULT}, which typically indicates
 * that no scoped proxy should be created unless a different default
 * has been configured at the component-scan instruction level.
 * <p>Analogous to {@code <aop:scoped-proxy/>} support in Spring XML.
 * @see ScopedProxyMode
 */
ScopedProxyMode proxyMode() default ScopedProxyMode.DEFAULT;
}

```

[https://blog.csdn.net/yerenyuan\\_pku](https://blog.csdn.net/yerenyuan_pku)

从@Scope注解类的源码中可以看出，在@Scope注解中可以设置如下值：

1. ConfigurableBeanFactory#SCOPE\_PROTOTYPE
2. ConfigurableBeanFactory#SCOPE\_SINGLETON
3. org.springframework.web.context.WebApplicationContext#SCOPE\_REQUEST
4. org.springframework.web.context.WebApplicationContext#SCOPE\_SESSION

很明显，在@Scope注解中可以设置的值包括ConfigurableBeanFactory接口中的SCOPE\_PROTOTYPE和SCOPE\_SINGLETON，以及WebApplicationContext类中的SCOPE\_REQUEST和SCOPE\_SESSION。这些都是什么鬼啊？别急，我们来一个个查看。

首先，我们查看一下ConfigurableBeanFactory接口的源码，发现在该接口中存在两个常量的定义，如下所示。



```

48 * @see ConfigurableListableBeanFactory
49 */
50 public interface ConfigurableBeanFactory extends HierarchicalBeanFactory, SingletonBeanRegistry {
51
52     /**
53      * Scope identifier for the standard singleton scope: "singleton".
54      * Custom scopes can be added via {@code registerScope}.
55      * @see #registerScope
56      */
57     String SCOPE_SINGLETON = "singleton";
58
59     /**
60      * Scope identifier for the standard prototype scope: "prototype".
61      * Custom scopes can be added via {@code registerScope}.
62      * @see #registerScope
63      */
64     String SCOPE_PROTOTYPE = "prototype";
65
66
67     /**
68      * Set the parent of this bean factory.
69      * <p>Note that the parent cannot be changed: It should only be set outside
70      * a constructor if it isn't available at the time of factory instantiation.
71      * @param parentBeanFactory the parent BeanFactory
72      * @throws IllegalStateException if this factory is already associated with
73      * a parent BeanFactory
74      * @see #getParentBeanFactory()
75      */
76     void setParentBeanFactory(BeansFactory parentBeanFactory) throws IllegalStateException;
77
78     /**

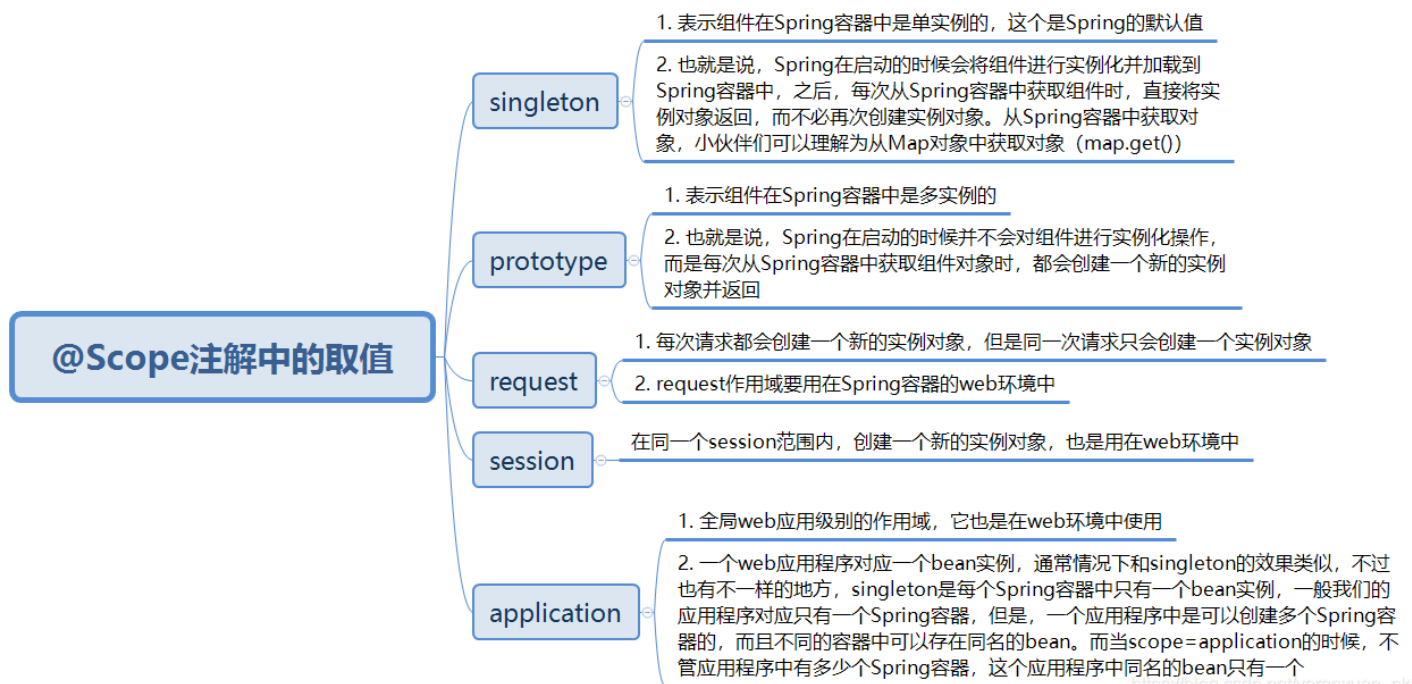
```

[https://blog.csdn.net/yerenyuan\\_pku](https://blog.csdn.net/yerenyuan_pku)

没错，SCOPE\_SINGLETON就是singleton，而SCOPE\_PROTOTYPE就是prototype。

那么，WebApplicationContext类中的SCOPE\_REQUEST和SCOPE\_SESSION又是什么鬼呢？我也不卖关子了，就直接说了，当我们使用Web容器来运行Spring应用时，在@Scope注解中可以设置WebApplicationContext类中的SCOPE\_REQUEST和SCOPE\_SESSION这俩的值，而SCOPE\_REQUEST的值就是request，SCOPE\_SESSION的值就是session。

综上，在@Scope注解中的取值如下所示。



其中，request和session 作用域是需要Web环境来支持的，这两个值基本上使用不到。当我们使用Web容器来运行Spring应用时，如果需要将组件的 **实例对象** 的作用域设置为request和session，那么我们通常会使用

```
1 request.setAttribute("key", object);
   AI写代码java运行
```

和

```
1 session.setAttribute("key", object);
   AI写代码java运行
```

这两种形式来将对象实例设置到request和session中，而不会使用@Scope注解来进行设置。

### 单实例bean作用域

首先，我们在com.meimeixia.config包下创建一个配置类，例如MainConfig2，然后在该配置类中实例化一个Person对象，并将其放置在Spring容器中，如下所示。

```

1 package com.meimeixia.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5
6 import com.meimeixia.bean.Person;
7
8 @Configuration
9 public class MainConfig2 {
10
11     @Bean("person")
12     public Person person() {
13         return new Person("美美侠", 25);
14     }
15 }
16
   AI写代码java运行

```

接着，在IOCTest类中创建一个test02()测试方法，在该测试方法中创建一个AnnotationConfigApplicationContext对象，创建完毕后，从Spring容器中按照id获取两个Person对象，并判断这两个对象是否是同一个对象，代码如下所示。

```

1 @SuppressWarnings("resource")
2 @Test
3 public void test02() {
4     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
5
6     // 获取到的这个Person对象默认是单实例的，因为在IOC容器中给我们加的这些组件默认都是单实例的，
7

```

```

8 // 所以说在这儿我们无论多少次获取，获取到的都是我们之前new的那个实例对象
9 Person person = (Person) applicationContext.getBean("person");
10 Person person2 = (Person) applicationContext.getBean("person");
11 System.out.println(person == person2);

```

AI写代码java运行



由于对象在Spring容器中默认是单实例的，所以，Spring容器在启动时就会将实例对象加载到Spring容器中，之后，每次从Spring容器中获取实例对象，都是直接将对象返回，而不必再创建新的实例对象了。很显然，此时运行test02()方法之后会输出true，如下所示。

这也正好验证了我们的结论：对象在Spring容器中默认是单实例的，Spring容器在启动时就会将实例对象加载到Spring容器中，之后，每次从Spring容器中获取实例对象，都是直接将对象返回，而不必再创建新的实例对象了。

## 多实例bean作用域

修改Spring容器中组件的作用域，我们需要借助于@Scope注解。此时，我们将MainConfig2配置类中Person对象的作用域修改成prototype，如下所示。

```

1 package com.meimeixia.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.context.annotation.Scope;
6
7 import com.meimeixia.bean.Person;
8
9 @Configuration
10 public class MainConfig2 {
11
12     @Scope("prototype") // 通过@Scope注解来指定该bean的作用范围，也可以说成是调整作用域
13     @Bean("person")
14     public Person person() {
15         return new Person("美美侠", 25);
16     }
17 }
18

```

AI写代码java运行



其实，通过@Scope注解来设置组件的作用域就等同于我们在XML配置文件中为 <bean> 标签设置scope属性，如下所示。

```

MainConfig2.java  IOCTest.java  beans.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
4     xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans
7         http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
8         http://www.springframework.org/schema/context
9         http://www.springframework.org/schema/context/spring-context-4.2.xsd">
10
11     <!-- 注册组件 -->
12     <bean id="person" class="com.meimeixia.bean.Person" scope="prototype">
13         <property name="age" value="18"></property>
14         <property name="name" value="Liayun"></property>
15     </bean>
16
17 </beans>

```

此时，我们再次运行IOCTest类中的test02()方法，你觉得从Spring容器中获取到的person对象和person2对象还是同一个对象吗？

```

Markers  Properties  Servers  Data Sour...  Snippets  Problems  Console  Progress  Search  Maven Rep...  Synchronize  JUnit
<terminated> IOCTest.test02 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午3:36:58)
十一月 28, 2020 3:36:58 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext@7755c
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@7755c
false

```

很显然不是，从以上输出结果中也可以看出，此时，输出的person对象和person2对象已经不是同一个对象了。

### 单实例bean作用域何时创建对象？

接下来，我们验证下在单实例作用域下，Spring是在什么时候创建对象的？

首先，我们将MainConfig2配置类中的Person对象的作用域修改成单实例，并在返回Person对象之前打印相关的信息，如下所示。

```

1 package com.meimeixia.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.context.annotation.Scope;
6
7 import com.meimeixia.bean.Person;
8
9 @Configuration
10 public class MainConfig2 {
11
12     @Scope
13     @Bean("person")
14     public Person person() {
15         System.out.println("给容器中添加咱们这个Person对象...");
16         return new Person("美美侠", 25);
17     }
18
19 }

```

AI写代码java运行



然后，我们在IOCTest类中再创建一个test03()方法，在该方法中我们只创建Spring容器，如下所示。

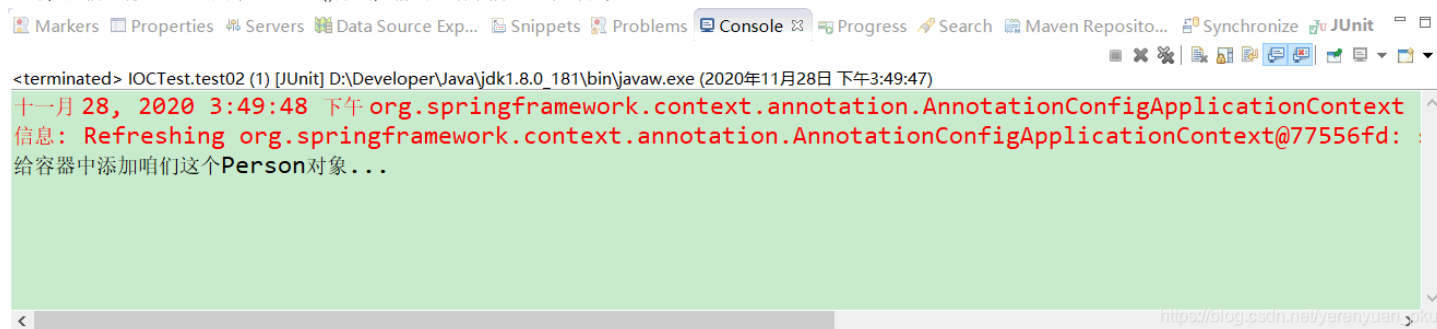
```

1 @Test
2 public void test03() {
3

```

```
4 | AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);  
    }  
    AI写代码java运行
```

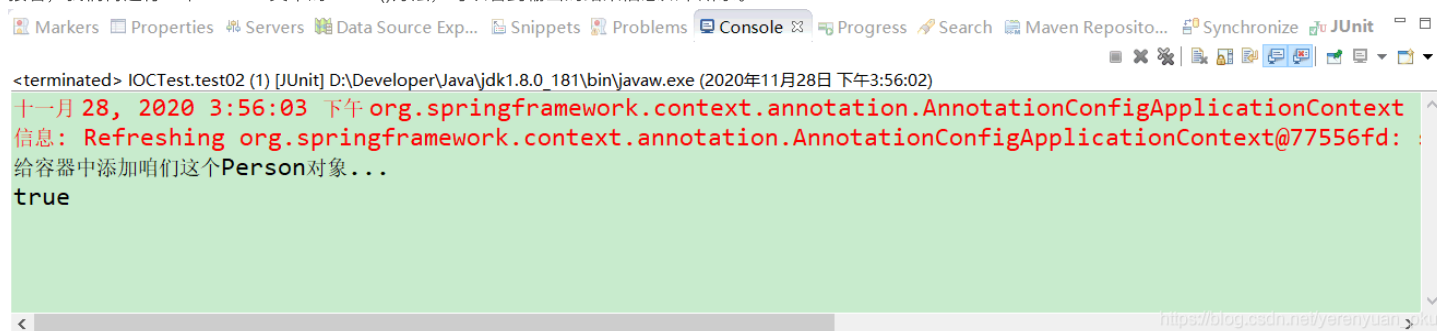
此时，我们运行IOCTest类中的test03()方法，输出的结果信息如下所示。



```
<terminated> IOCTest.test02 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午3:49:47)  
十一月 28, 2020 3:49:48 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext  
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd:  
给容器中添加咱们这个Person对象...
```

从以上输出的结果信息中可以看出，Spring容器在创建的时候，就将@Scope注解标注为singleton的组件进行了实例化，并加载到了Spring容器中。

接着，我们再运行一下IOCTest类中的test02()方法，可以看到输出的结果信息如下所示。



```
<terminated> IOCTest.test02 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午3:56:02)  
十一月 28, 2020 3:56:03 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext  
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd:  
给容器中添加咱们这个Person对象...  
true
```

这说明，Spring容器在启动时，将单实例组件实例化之后，会即刻加载到Spring容器中，以后每次从容器中获取组件实例对象时，都是直接返回相应的对象，而不必再创建新的对象了。

## 多实例bean作用域何时创建对象？

如果我们将对象的作用域修改成多实例，那么会什么时候创建对象呢？

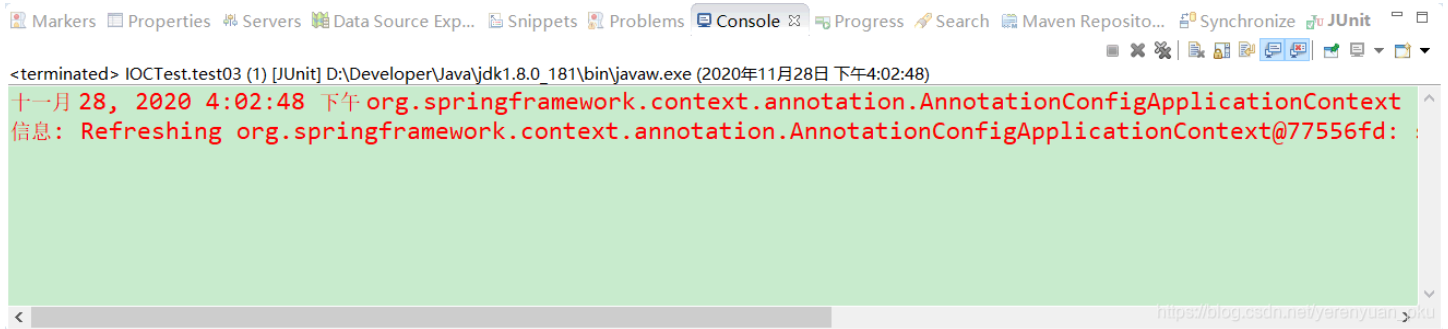
此时，我们将MainConfig2配置类中的Person对象的作用域修改成多实例，如下所示。

```
1 | package com.meimeixia.config;  
2 |  
3 | import org.springframework.context.annotation.Bean;  
4 | import org.springframework.context.annotation.Configuration;  
5 | import org.springframework.context.annotation.Scope;  
6 |  
7 | import com.meimeixia.bean.Person;  
8 |  
9 | @Configuration  
10 | public class MainConfig2 {  
11 |  
12 |     @Scope("prototype") // 通过@Scope注解来指定该bean的作用范围，也可以说成是调整作用域  
13 |     @Bean("person")  
14 |     public Person person() {  
15 |         System.out.println("给容器中添加咱们这个Person对象...");  
16 |         return new Person("美美侠", 25);  
17 |     }  
18 |  
19 | }
```

AI写代码java运行

我们再次运行IOCTest类中的test03()方法，发现没有输出任何结果信息。





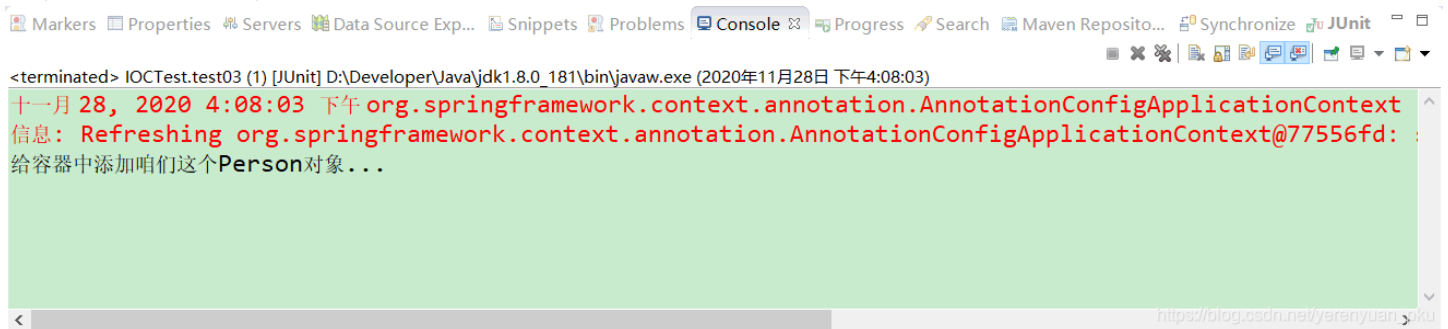
```
<terminated> IOCTest.test03 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午4:02:48)
十一月 28, 2020 4:02:48 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: ...
```

这说明在创建Spring容器时，并不会去实例化和加载多实例对象，那么多实例对象到底是什么时候实例化的呢？此时，我们可以在IOCTest类中的test03()方法中添加一行获取Person对象的代码，如下所示。

```
1 @Test
2 public void test03() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     Person person = (Person) applicationContext.getBean("person");
5 }
```

AI写代码java运行

然后，我们再次运行以上方法，输出的结果信息如下所示。



```
<terminated> IOCTest.test03 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午4:08:03)
十一月 28, 2020 4:08:03 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: ...
给容器中添加咱们这个Person对象...
```

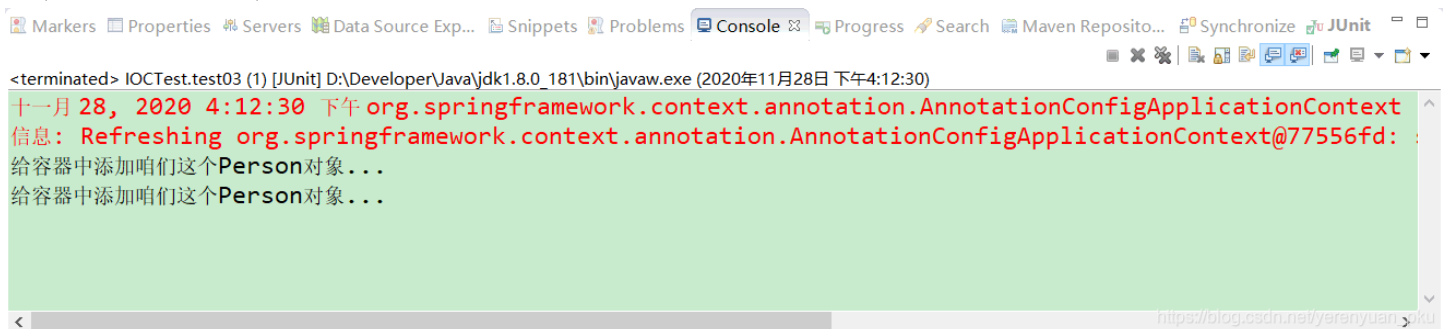
从以上输出的结果信息中可以看出，当向Spring容器中获取Person实例对象时，Spring容器才会实例化Person对象，再将其加载到Spring容器中去。

那问题来了，此时Spring容器是否只实例化了一个Person对象呢？我们在IOCTest类中的test03()方法中再添加一行获取Person对象的代码，如下所示。

```
1 @Test
2 public void test03() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     Person person = (Person) applicationContext.getBean("person");
5     Person person2 = (Person) applicationContext.getBean("person");
6 }
```

AI写代码java运行

此时，我们再次运行以上方法，输出的结果信息如下所示。



```
<terminated> IOCTest.test03 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午4:12:30)
十一月 28, 2020 4:12:30 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd: ...
给容器中添加咱们这个Person对象...
给容器中添加咱们这个Person对象...
```

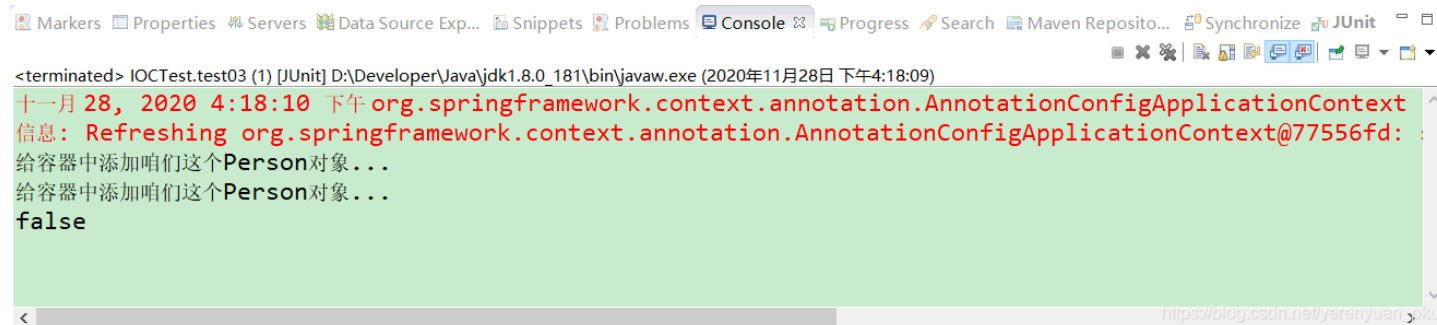
从以上输出的结果信息中可以看出，当对象的Scope作用域为多实例时，每次向Spring容器获取对象时，它都会创建一个新的对象并返回。很显然，以上获取到的person和person2就不是同一个对象了，这我们也可以打印结果信息来进行验证，即在IOCTest类中的test03()方法中判断两个对象是否相等，如下所示。

```
1 @Test
2 public void test03() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     Person person = (Person) applicationContext.getBean("person");
5     Person person2 = (Person) applicationContext.getBean("person");
6     System.out.println(person == person2);
7 }
```

AI写代码java运行



此时，我们再次运行以上方法，发现输出的结果信息如下所示。



```
<terminated> IOCTest.test03 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午4:18:09)
十一月 28, 2020 4:18:10 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@77556fd:
给容器中添加咱们这个Person对象...
给容器中添加咱们这个Person对象...
false
```

可以看到，当对象是多实例时，每次从Spring容器中获取对象时，都会创建新的实例对象，并且每个实例对象都不相等。

## 单实例bean注意的事项

单实例bean是整个应用所共享的，所以需要考虑到线程安全问题，之前在玩SpringMVC的时候，SpringMVC中的Controller默认是单例的，有些开发者在Controller中创建了一些变量，那么这些变量实际上就变成共享的了，Controller又可能会被很多线程同时访问，这些线程并发去修改Controller中的共享变量，此时很有可能会出现数据错乱的问题，所以使用的时候需要特别注意。

## 多实例bean注意的事项

多实例bean每次获取的时候都会重新创建，如果这个bean比较复杂，创建时间比较长，那么就会影响系统的性能，因此这个地方需要注意点。

## 自定义Scope

如果Spring内置的几种scope都无法满足我们的需求时，我们可以自定义bean的作用域。

### 如何实现自定义Scope呢？

自定义Scope主要分为三个步骤，如下所示。

第一步，实现Scope接口。我们先来看下Scope接口的源码，如下所示。

```
1 package org.springframework.beans.factory.config;
2
3 import org.springframework.beans.factory.ObjectFactory;
4
5 public interface Scope {
6
7     /**
8      * 返回当前作用域中name对应的bean对象
9      * @param name 需要检索的bean对象的名称
10     * @param objectFactory 如果name对应的bean对象在当前作用域中没有找到，那么可以调用这个objectFactory来创建这个对象
11     */
12     Object get(String name, ObjectFactory<?> objectFactory);
13
14     /**
15      * 将name对应的bean对象从当前作用域中移除
16      */
17     Object remove(String name);
18
19     /**
20      * 用于注册销毁回调，若想要销毁相应的对象，则由Spring容器注册相应的销毁回调，而由自定义作用域选择是不是要销毁相应的对象
21      */
22     void registerDestructionCallback(String name, Runnable callback);
23
24     /**
25      * 用于解析相应的上下文数据，比如request作用域将返回request中的属性
26      */
27     Object resolveContextualObject(String key);
28
29     /**
30      * 作用域的会话标识，比如session作用域的会话标识是sessionId
31      */
32     String getConversationId();
33
34 }
```

AI写代码java运行



第二步，将自定义Scope注册到容器中。此时，需要调用org.springframework.beans.factory.config.ConfigurableBeanFactory#registerScope这个方法，咱们看一下这个方法声明。

```
MainConfig2.java  IOCTest.java  Scope.class  ConfigurableBeanFactory.class
236  */
237  void addBeanPostProcessor(BeanPostProcessor beanPostProcessor);
238
239  /**
240   * Return the current number of registered BeanPostProcessors, if any.
241   */
242  int getBeanPostProcessorCount();
243
244  /**
245   * Register the given scope, backed by the given Scope implementation向容器中注册自定义的Scope
246   * @param scopeName the scope identifier scopeName: 作用域名称
247   * @param scope the backing Scope implementation scope: 作用域对象
248   */
249  void registerScope(String scopeName, Scope scope);
250
251  /**
252   * Return the names of all currently registered scopes.
253   * <p>This will only return the names of explicitly registered scopes.
254   * Built-in scopes such as "singleton" and "prototype" won't be exposed.
255   * @return the array of scope names, or an empty array if none
256   * @see #registerScope
257   */
258  String[] getRegisteredScopeNames();
259
260  /**
261   * Return the Scope implementation for the given scope name, if any.
262   * <p>This will only return explicitly registered scopes.
263   * Built-in scopes such as "singleton" and "prototype" won't be exposed.
264   * @param scopeName the name of the scope
265   * @return the registered Scope implementation, or {@code null} if none
266   * @see #registerScope
267   */
```

第三步，使用自定义的作用域。也就是在定义bean的时候，指定bean的scope属性为自定义的作用域名称。

### 一个自定义Scope实现案例

例如，我们来实现一个线程级别的bean作用域，同一个线程中同名的bean是同一个实例，不同的线程中的bean是不同的实例。

这里，要求bean在线程中是共享的，所以我们可以通过ThreadLocal来实现，ThreadLocal可以实现线程中数据的共享。

首先，我们在com.meimeixia.scope包下新建一个ThreadScope类，如下所示。

```
1 package com.meimeixia.scope;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import java.util.Objects;
6
7 import org.springframework.beans.factory.ObjectFactory;
8 import org.springframework.beans.factory.config.Scope;
9
10 /**
11  * 自定义本地线程级别的bean作用域，不同的线程中的bean是不同的实例，同一个线程中同名的bean是同一个实例
12  * @author liayun
13  */
14
15 public class ThreadScope implements Scope {
16
17     public static final String THREAD_SCOPE = "thread";
18
19     private ThreadLocal<Map<String, Object>> beanMap = new ThreadLocal() {
20
21         @Override
22         protected Object initialValue() {
23             return new HashMap<>();
24         }
25     };
26
27 }
28
```

```

29  /**
30   * 返回当前作用域中name对应的bean对象
31   * @param name: 需要检索的bean对象的名称
32   * @param objectFactory: 如果name对应的bean对象在当前作用域中没有找到, 那么可以调用这个objectFactory来创建这个bean对象
33   */
34  @Override
35  public Object get(String name, ObjectFactory<?> objectFactory) {
36      Object bean = beanMap.get().get(name);
37      if (Objects.isNull(bean)) {
38          bean = objectFactory.getObject();
39          beanMap.get().put(name, bean);
40      }
41      return bean;
42  }
43
44  /**
45   * 将name对应的bean对象从当前作用域中移除
46   */
47  @Override
48  public Object remove(String name) {
49      return this.beanMap.get().remove(name);
50  }
51
52  /**
53   * 用于注册销毁回调, 若想要销毁相应的对象, 则由Spring容器注册相应的销毁回调, 而由自定义作用域选择是不是要销毁相应的对象
54   */
55  // bean作用域范围结束的时候调用的方法, 用于bean的清理
56  @Override
57  public void registerDestructionCallback(String name, Runnable callback) {
58      System.out.println(name);
59  }
60
61  /**
62   * 用于解析相应的上下文数据, 比如request作用域将返回request中的属性
63   */
64  @Override
65  public Object resolveContextualObject(String key) {
66      return null;
67  }
68
69  /**
70   * 作用域的会话标识, 比如session作用域的会话标识是sessionId
71   */
72  @Override
73  public String getConversationId() {
74      return Thread.currentThread().getName();
75  }
76  }

```

AI写代码java运行



在ThreadScope类中, 我们定义了一个THREAD\_SCOPE常量, 该常量是在定义bean的时候给scope使用的。

然后, 我们在com.meimeixia.config包下创建一个配置类, 例如MainConfig3, 并使用@Scope("thread")注解标注Person对象的作用域为Thread范围, 如下所示。

```

1  package com.meimeixia.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.context.annotation.Scope;
6
7  import com.meimeixia.bean.Person;
8
9  /**
10   * 测试@Scope注解设置的作用域
11   * @author liayun
12   */
13  @Configuration
14  public class MainConfig3 {
15
16      @Scope("thread")
17      @Bean("person")
18      public Person person() {
19          System.out.println("给容器中添加咱们这个Person对象...");
20      }

```

```

21         return new Person("美美侠", 25);
22     }
23
24 }

```

AI写代码java运行



接着，我们在IOCTest类中创建一个test04()方法，我们所要做的事情就是在该方法中创建Spring容器，并向Spring容器中注册ThreadScope对象。最后，使用循环创建两个Thread线程，并分别在每个线程中获取两个Person对象，如下所示。

```

1  @Test
2  public void test04() {
3      AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig3.class);
4      // 向容器中注册自定义的Scope
5      applicationContext.getBeanFactory().registerScope(ThreadScope.THREAD_SCOPE, new ThreadScope());
6
7      // 使用容器获取bean
8      for (int i = 0; i < 2; i++) {
9          new Thread(() -> {
10              System.out.println(Thread.currentThread() + "," + applicationContext.getBean("person"));
11              System.out.println(Thread.currentThread() + "," + applicationContext.getBean("person"));
12          }).start();
13      }
14      try {
15          TimeUnit.SECONDS.sleep(1);
16      } catch (Exception e) {
17          e.printStackTrace();
18      }
19 }

```

AI写代码java运行



此时，我们运行以上方法，会看到输出的结果信息如下所示。

```

<terminated> IOCTest.test04 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月28日 下午5:38:47)
十一月 28, 2020 5:38:48 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@9e89d68:
给容器中添加咱们这个Person对象...
给容器中添加咱们这个Person对象...
Thread[Thread-2,5,main],com.meimeixia.bean.Person@293cbbda
Thread[Thread-2,5,main],com.meimeixia.bean.Person@293cbbda
Thread[Thread-1,5,main],com.meimeixia.bean.Person@7c8d74d7
Thread[Thread-1,5,main],com.meimeixia.bean.Person@7c8d74d7

```

从以上输出的结果信息中可以看到，bean在同样的线程中获取到的是同一个bean的实例，不同的线程中bean的实例是不同的。

注意：这里测试时，我将Person类进行了相应的调整，将toString()方法注释掉了，如下所示。

```

1  package com.meimeixia.bean;
2
3  public class Person {
4
5      private String name;
6      private Integer age;
7
8      public String getName() {
9          return name;
10     }
11     public void setName(String name) {
12         this.name = name;
13     }
14     public Integer getAge() {
15         return age;
16     }
17     public void setAge(Integer age) {
18         this.age = age;
19     }
20 }

```

```
19     }
20     public Person(String name, Integer age) {
21         super();
22         this.name = name;
23         this.age = age;
24     }
25     public Person() {
26         super();
27         // TODO Auto-generated constructor stub
28     }
29
30     // @Override
31     // public String toString() {
32     //     return "Person [name=" + name + ", age=" + age + " ]";
33     // }
34
35 }
```

AI写代码java运行

