

Spring注解驱动开发第7讲——如何按照条件向Spring容器中注册bean? 这次我懂了!!

写在前面

当bean是单实例，并且没有设置懒加载时，Spring容器启动时，就会 **实例化** bean，并将bean注册到IOC容器中，以后每次从IOC容器中获取bean时，直接返回IOC容器中的bean，而不用再创建新的bean了。

若bean是单实例，并且使用@Lazy注解设置了懒加载，则Spring容器启动时，不会立即实例化bean，自然就不会将 **bean注册** 到IOC容器中了，只有第一次获取bean的时候，才会实例化bean，并且将bean注册到IOC容器中。

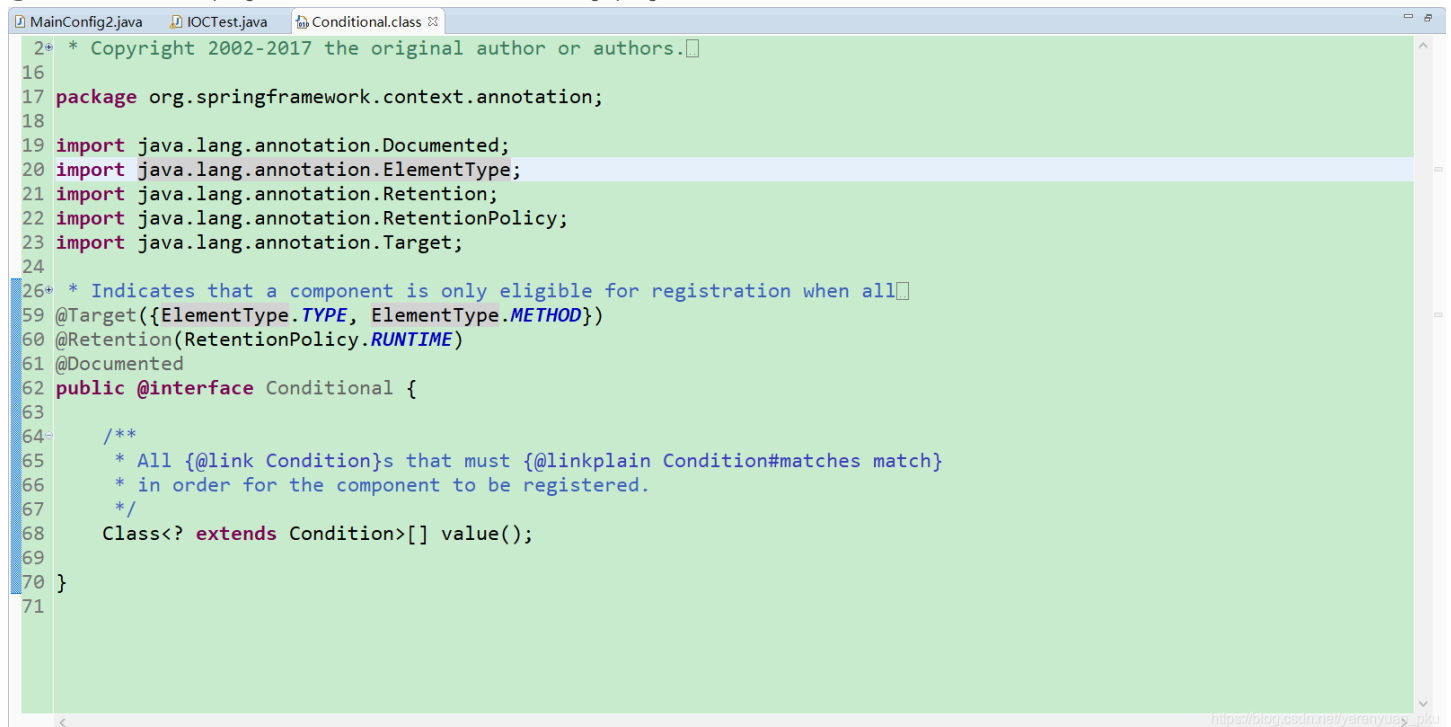
若bean是多实例，则Spring容器启动时，不会实例化bean，也不会将bean注册到IOC容器中，只是在以后每次从IOC容器中获取bean的时候，都会创建一个新的bean返回。

其实，Spring支持按照条件向IOC容器中注册bean，满足条件的bean就会被注册到IOC容器中，不满足条件的bean就不会被注册到IOC容器中。接下来，我们就一起来探讨一下Spring中是如何实现按照条件向IOC容器中注册bean的。

@Conditional注解概述

@Conditional注解可以按照一定的条件进行判断，满足条件向容器中注册bean，不满足条件就不向容器中注册bean。

@Conditional注解是由Spring Framework提供的一个注解，它位于 org.springframework.context.annotation包内，定义如下。



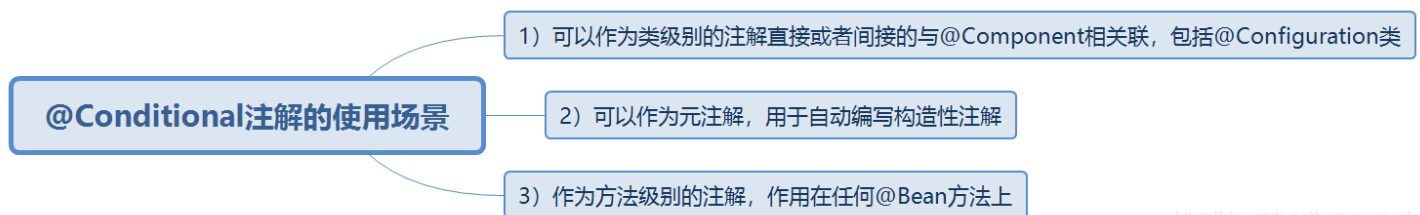
```
2* * Copyright 2002-2017 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import java.lang.annotation.Documented;
20 import java.lang.annotation.ElementType;
21 import java.lang.annotation.Retention;
22 import java.lang.annotation.RetentionPolicy;
23 import java.lang.annotation.Target;
24
26* * Indicates that a component is only eligible for registration when all
59 @Target({ElementType.TYPE, ElementType.METHOD})
60 @Retention(RetentionPolicy.RUNTIME)
61 @Documented
62 public @interface Conditional {
63
64     /**
65      * All {@link Condition}s that must {@linkplain Condition#matches match}
66      * in order for the component to be registered.
67      */
68     Class<? extends Condition>[] value();
69
70 }
71
```

从@Conditional注解的源码来看，@Conditional注解不仅可以添加到类上，也可以添加到方法上。在@Conditional注解中，还存在着一个Condition类型或者其子类型的Class对象数组，Condition是个啥呢？我们点进去看一下。

```
2* * Copyright 2002-2013 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import org.springframework.beans.factory.config.BeanFactoryPostProcessor;
20 import org.springframework.core.type.AnnotatedTypeMetadata;
21
22 * A single {@code Condition} that must be {@link Plain.matches} in order
41 public interface Condition {
42
43     /**
44      * Determine if the condition matches.
45      * @param context the condition context
46      * @param metadata metadata of the {@link org.springframework.core.type.AnnotationMetadata} class
47      * or {@link org.springframework.core.type.MethodMetadata} method} being checked.
48      * @return {@code true} if the condition matches and the component can be registered
49      * or {@code false} to veto registration.
50     */
51     boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);
52
53 }
54
```

可以看到，它是一个接口。所以，我们使用@Conditional注解时，需要写一个类来实现Spring提供的Condition接口，它会匹配@Conditional所符合的方法（这句话怎么说的那么不明白啊！），然后我们就可以使用我们在@Conditional注解中定义的方法来检查了。

我们可以在哪些场合使用@Conditional注解呢？@Conditional注解的使用场景如下图所示。



https://blog.csdn.net/yerenyuan_pku

向Spring容器注册bean

不带条件注册bean

我们在MainConfig2配置类中新增person01()方法和person02()方法，并为这两个方法添加@Bean注解，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.Lazy;
6
7 import com.meimeixia.bean.Person;
8
9 @Configuration
10 public class MainConfig2 {
11
12     @Lazy
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     @Bean("bill")
20     public Person person01() {
21         return new Person("Bill Gates", 62);
22     }
23
24     @Bean("linus")
25     public Person person02() {
26         return new Person("linus", 48);
27     }
28
29 }
```

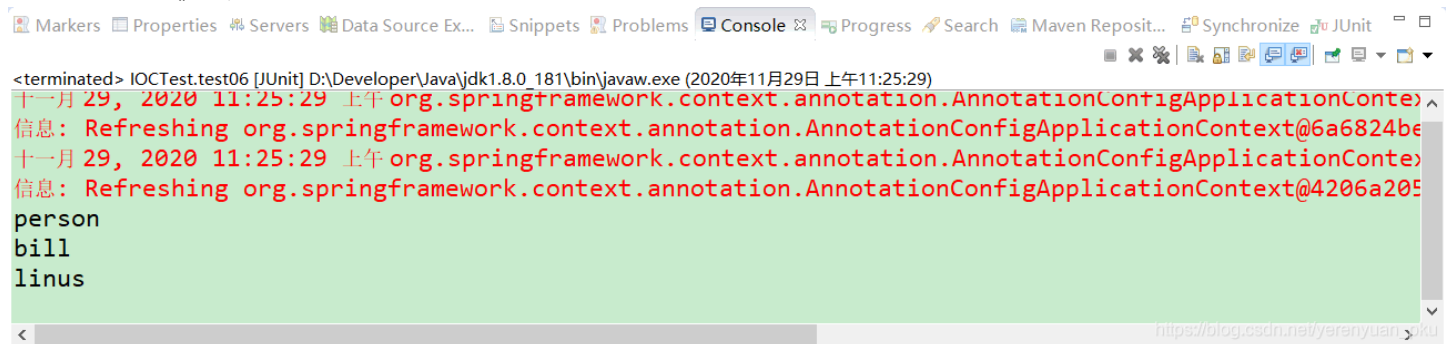
AI写代码java运行

那么，这两个bean默认是否会被注册到Spring容器中去呢？我们新建一个测试用例来测试一下，即在IOCTest类中新建一个test06()方法，如下所示。

```
1 @Test
2 public void test06() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     // 我们现在就来看一下IOC容器中Person这种类型的bean都有哪些
5     String[] namesForType = applicationContext.getBeanNamesForType(Person.class);
6
7     for (String name : namesForType) {
8         System.out.println(name);
9     }
10 }
11 }
```

AI写代码java运行

我们运行以上test06()方法，发现输出的结果信息如下所示。



```
<terminated> IOCTest.test06 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月29日 上午11:25:29)
十一月 29, 2020 11:25:29 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh:
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@6a6824be
十一月 29, 2020 11:25:29 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh:
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4206a205
person
bill
linus
```

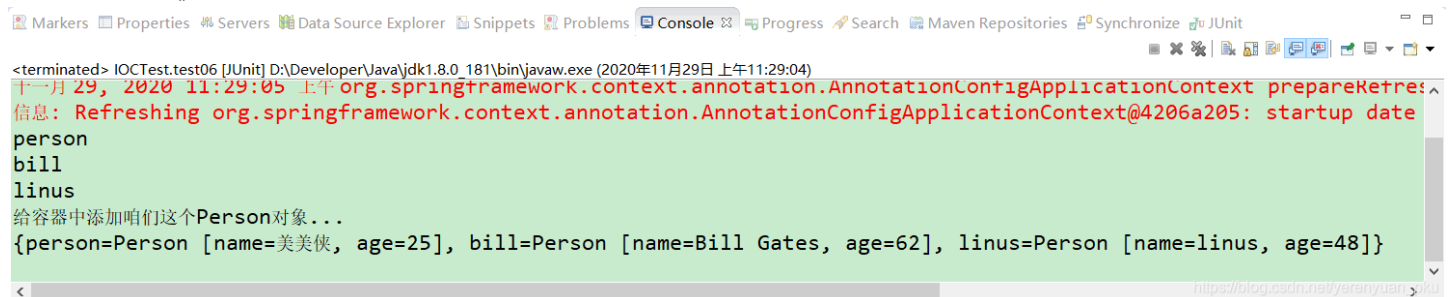
从输出结果中可以看出，同时输出了bill和linus。说明默认情况下，Spring容器会将单实例并且非懒加载的bean注册到IOC容器中。

接下来，我们再输出bean的名称和bean实例对象信息，此时我们只须在test06()方法中添加如下的代码片段即可。

```
1 @Test
2 public void test06() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     // 我们现在就来看一下IOC容器中Person这种类型的bean都有哪些
5     String[] namesForType = applicationContext.getBeanNamesForType(Person.class);
6
7     for (String name : namesForType) {
8         System.out.println(name);
9     }
10
11     Map<String, Person> persons = applicationContext.getBeansOfType(Person.class); // 找到这个Person类型的所有bean
12     System.out.println(persons);
13 }
```

AI写代码java运行

再次运行以上test06()方法，输出的结果如下所示。



```
<terminated> IOCTest.test06 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月29日 上午11:29:04)
十一月 29, 2020 11:29:05 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh:
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4206a205: startup date
person
bill
linus
给容器中添加咱们这个Person对象...
{person=Person [name=美美侠, age=25], bill=Person [name=Bill Gates, age=62], linus=Person [name=linus, age=48]}
```

可以看到，输出了注册到容器中的bean实例对象的详细信息。

注意：这里测试时，我将Person类进行了相应的调整，将toString()方法的注释放开了，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    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



带条件注册bean

现在，我们就要提出一个新的需求了，比如，如果当前操作系统是Windows 操作系统，那么就向Spring容器中注册名称为bill的Person对象；如果当前操作系统是Linux 操作系统，那么就向Spring容器中注册名称为linus的Person对象。要想实现这个需求，我们就得要使用@Conditional注解了。

这里，有小伙伴可能会问，如何获取操作系统的类型呢？别急，这个问题很简单，我们继续向下看。

使用Spring中的AnnotationConfigApplicationContext类就能够获取到当前操作系统的类型，如下所示。

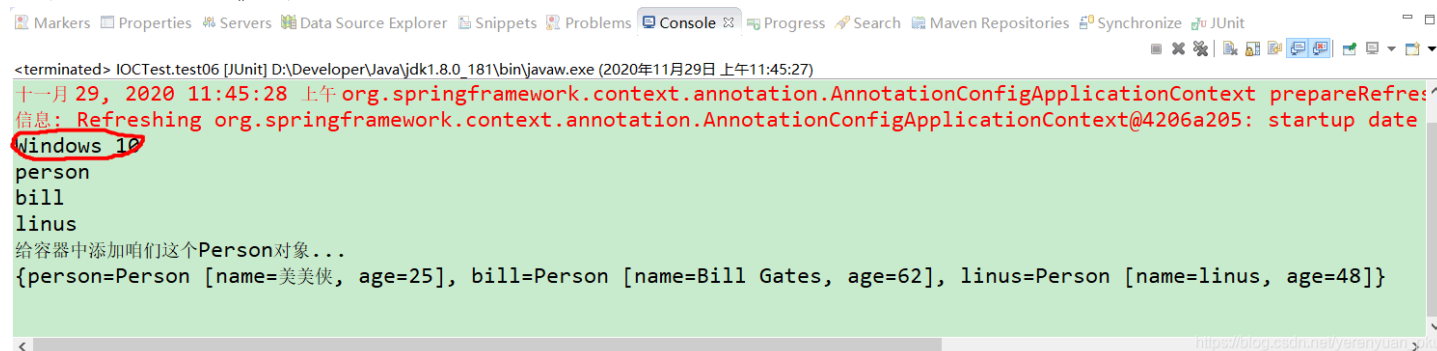
```
1 AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
2 ConfigurableEnvironment environment = applicationContext.getEnvironment(); // 拿到IOC运行环境
3 // 动态获取环境变量的值，例如操作系统的名字
4 String property = environment.getProperty("os.name"); // 获取操作系统的名字，例如Windows 10
5 System.out.println(property);
6 AI写代码java运行
```

我们将上述代码整合到IOCTest类中的test06()方法中，如下所示。

```
1 @Test
2 public void test06() {
3     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);
4     // 我们现在就来看一下IOC容器中Person这种类型的bean都有哪些
5     String[] namesForType = applicationContext.getBeanNamesForType(Person.class);
6     ConfigurableEnvironment environment = applicationContext.getEnvironment(); // 拿到IOC运行环境
7     // 动态获取环境变量的值，例如操作系统的名字
8     String property = environment.getProperty("os.name"); // 获取操作系统的名字，例如Windows 10
9     System.out.println(property);
10
11     for (String name : namesForType) {
12         System.out.println(name);
13     }
14
15     Map<String, Person> persons = applicationContext.getBeansOfType(Person.class); // 找到这个Person类型的所有bean
16     System.out.println(persons);
17 }
```



然后，我们运行以上test06()方法，会看到输出了如下图所示的结果信息。



```
<terminated> IOCtest.test06 [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年11月29日 上午11:45:27)
十一月 29, 2020 11:45:28 上午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4206a205: startup date
Windows 10
person
bill
linus
给容器中添加咱们这个Person对象...
{person=Person [name=美美侠, age=25], bill=Person [name=Bill Gates, age=62], linus=Person [name=linus, age=48]}
```

由于我使用的操作系统是Windows 10操作系统，所以在结果信息中输出了Windows 10。

到这里，我们成功获取到了操作系统的类型，接下来就要来实现上面那个需求了。此时，我们可以借助Spring中的@Conditional注解来实现。

要想使用@Conditional注解，我们需要实现Condition接口来为@Conditional注解设置条件，所以，这里我们创建了两个实现Condition接口的类，它们分别是LinuxCondition和WindowsCondition，如下所示。

- LinuxCondition

```
1 package com.meimeixia.condition;
2
3 import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
4 import org.springframework.beans.factory.support.BeanDefinitionRegistry;
5 import org.springframework.context.annotation.Condition;
6 import org.springframework.context.annotation.ConditionContext;
7 import org.springframework.core.env.Environment;
8 import org.springframework.core.type.AnnotatedTypeMetadata;
9
10 /**
11  * 判断操作系统是否是Linux系统
12  * @author liayun
13  *
14  */
15 public class LinuxCondition implements Condition {
16
17     /**
18      * ConditionContext: 判断条件能使用的上下文 (环境)
19      * AnnotatedTypeMetadata: 当前标注了@Conditional注解的注释信息
20      */
21     @Override
22     public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
23         // 判断操作系统是否是Linux系统
24
25         // 1. 获取到bean的创建工厂 (能获取到IOC容器使用到的BeanFactory, 它就是创建对象以及进行装配的工厂)
26         ConfigurableListableBeanFactory beanFactory = context.getBeanFactory();
27         // 2. 获取到类加载器
28         ClassLoader classLoader = context.getClassLoader();
29         // 3. 获取当前环境信息, 它里面就封装了我们这个当前运行时的一些信息, 包括环境变量, 以及包括虚拟机的一些变量
30         Environment environment = context.getEnvironment();
31         // 4. 获取到bean定义的注册类
32         BeanDefinitionRegistry registry = context.getRegistry();
33
34         String property = environment.getProperty("os.name");
35         if (property.contains("linux")) {
36             return true;
37         }
38
39         return false;
40     }
41 }
42 }
```

AI写代码java运行



这里我得好好说道说道通过context的getRegistry()方法获取到的bean定义的注册对象，即BeanDefinitionRegistry对象了。它到底是个啥呢？我们可以点进去看一下它的源码，如下所示，可以看到它是一个接口。

```
* Copyright 2002-2012 the original author or authors.

package org.springframework.beans.factory.support;

import org.springframework.beans.factory.BeanDefinitionStoreException;
import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.core.AliasRegistry;

* Interface for registries that hold bean definitions, for example RootBeanDefinition
public interface BeanDefinitionRegistry extends AliasRegistry {

    * Register a new bean definition with this registry.
    void registerBeanDefinition(String beanName, BeanDefinition beanDefinition)
        throws BeanDefinitionStoreException;

    /**
    * Remove the BeanDefinition for the given name.
    * @param beanName the name of the bean instance to register
    * @throws NoSuchBeanDefinitionException if there is no such bean definition
    */
    void removeBeanDefinition(String beanName) throws NoSuchBeanDefinitionException;

    * Return the BeanDefinition for the given bean name.
    BeanDefinition getBeanDefinition(String beanName) throws NoSuchBeanDefinitionException;

    * Check if this registry contains a bean definition with the given name.
    boolean containsBeanDefinition(String beanName);

    * Return the names of all beans defined in this registry.
    String[] getBeanDefinitionNames();

    * Return the number of beans defined in the registry.
    int getBeanDefinitionCount();

    * Determine whether the given bean name is already in use within this registry.
    boolean isBeanNameInUse(String beanName);
}
```

Spring容器中所有的bean都是通过BeanDefinitionRegistry对象来进行注册的。因此我们可以通过它来查看Spring容器中注册了哪些bean

该方法表明我们可以通过BeanDefinitionRegistry对象向Spring容器中注册一个bean

该方法表明我们可以通过BeanDefinitionRegistry对象在Spring容器中移除一个bean

定义信息

该方法表明我们可以通过BeanDefinitionRegistry对象查看Spring容器中是否包含有某一个bean的定义

https://blog.csdn.net/yerenyuan_pku

在上图中我对BeanDefinitionRegistry接口的源码作了一点简要的说明。知道了，Spring容器中所有的bean都可以通过BeanDefinitionRegistry对象来进行注册，因此我们可以通过它来查看Spring容器中到底注册了哪些bean。而且仔细查看一下BeanDefinitionRegistry接口中声明的各个方法，你就知道我们还可以通过BeanDefinitionRegistry对象向Spring容器中注册一个bean、移除一个bean、查询某一个bean的定义信息或者判断Spring容器中是否包含有某一个bean的定义。

因此，我们可以在这儿做更多的判断，比如说我可以判断一下Spring容器中是不是包含有某一个bean，就像下面这样，如果Spring容器中果真包含有名称为person的bean，那么就做些什么事情，如果没包含，那么我们还可以利用BeanDefinitionRegistry对象向Spring容器中注册一个bean。

```
1 package com.meimeixia.condition;
2
3 import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
4 import org.springframework.beans.factory.support.BeanDefinitionRegistry;
5 import org.springframework.context.annotation.Condition;
6 import org.springframework.context.annotation.ConditionContext;
7 import org.springframework.core.env.Environment;
8 import org.springframework.core.type.AnnotatedTypeMetadata;
9
10 /**
11  * 判断操作系统是否是Linux系统
12  * @author liayun
13  */
14
15 public class LinuxCondition implements Condition {
16
17     /**
18     * ConditionContext: 判断条件能使用的上下文 (环境)
19     * AnnotatedTypeMetadata: 当前标注了@Conditional注解的注释信息
20     */
21     @Override
22     public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
23         // 判断操作系统是否是Linux系统
24     }
25 }
```



```

24
25 // 1. 获取到bean的创建工厂 (能获取到IOC容器使用到的BeanFactory, 它就是创建对象以及进行装配的工厂)
26 ConfigurableListableBeanFactory beanFactory = context.getBeanFactory();
27 // 2. 获取到类加载器
28 ClassLoader classLoader = context.getClassLoader();
29 // 3. 获取当前环境信息, 它里面就封装了我们这个当前运行时的一些信息, 包括环境变量, 以及包括虚拟机的一些变量
30 Environment environment = context.getEnvironment();
31 // 4. 获取到bean定义的注册类
32 BeanDefinitionRegistry registry = context.getRegistry();
33
34 // 在这儿还可以做更多的判断, 比如说我判断一下Spring容器中是不是包含有某一个bean, 就像下面这样, 如果Spring容器中果真包含有名称为person的bean,
35 boolean definition = registry.containsBeanDefinition("person");
36
37 String property = environment.getProperty("os.name");
38 if (property.contains("linux")) {
39     return true;
40 }
41
42 return false;
43 }
44
45 }

```

AI写代码java运行



- WindowsCondition

```

1 package com.meimeixia.condition;
2
3 import org.springframework.context.annotation.Condition;
4 import org.springframework.context.annotation.ConditionContext;
5 import org.springframework.core.env.Environment;
6 import org.springframework.core.type.AnnotatedTypeMetadata;
7
8 /**
9  * 判断操作系统是否是Windows系统
10  * @author liayun
11  *
12  */
13 public class WindowsCondition implements Condition {
14
15     @Override
16     public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
17         Environment environment = context.getEnvironment();
18         String property = environment.getProperty("os.name");
19         if (property.contains("Windows")) {
20             return true;
21         }
22         return false;
23     }
24 }
25 }

```

AI写代码java运行



然后, 我们就需要在MainConfig2配置类中使用@Conditional注解添加条件了。添加该注解后的方法如下所示。

```

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.Lazy;
7
8 import com.meimeixia.bean.Person;
9 import com.meimeixia.condition.LinuxCondition;
10 import com.meimeixia.condition.WindowsCondition;
11
12 @Configuration
13 public class MainConfig2 {
14
15

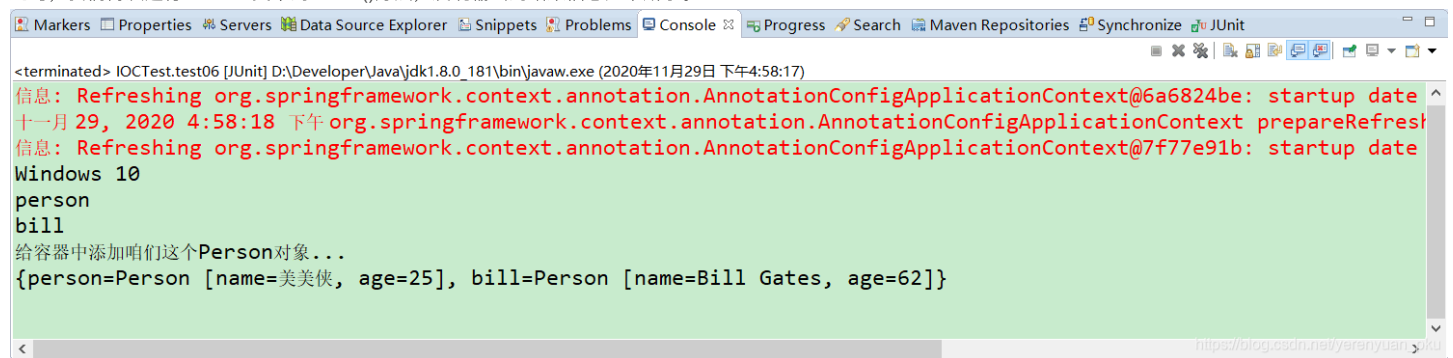
```

```
16     @Lazy
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         return new Person("美美侠", 25);
21     }
22
23     @Conditional({WindowsCondition.class})
24     @Bean("bill")
25     public Person person01() {
26         return new Person("Bill Gates", 62);
27     }
28
29     @Conditional({LinuxCondition.class})
30     @Bean("linus")
31     public Person person02() {
32         return new Person("linus", 48);
33     }
34 }
```

AI写代码java运行



此时，我们再次运行IOCTest类中的test06()方法，发现输出的结果信息如下所示。



```
<terminated> IOCTest.test06 [JUnit] D:\DeveloperJava\jdk1.8.0_181\bin\javaw.exe (2020年11月29日 下午4:58:17)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@6a6824be: startup date
十一月 29, 2020 4:58:18 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@7f77e91b: startup date
Windows 10
person
bill
给容器中添加咱们这个Person对象...
{person=Person [name=美美侠, age=25], bill=Person [name=Bill Gates, age=62]}
```

可以看到，输出结果中不再含有名称为linus的bean了，这说明程序中检测到当前操作系统为Windows 10之后，没有向Spring容器中注册名称为linus的bean。

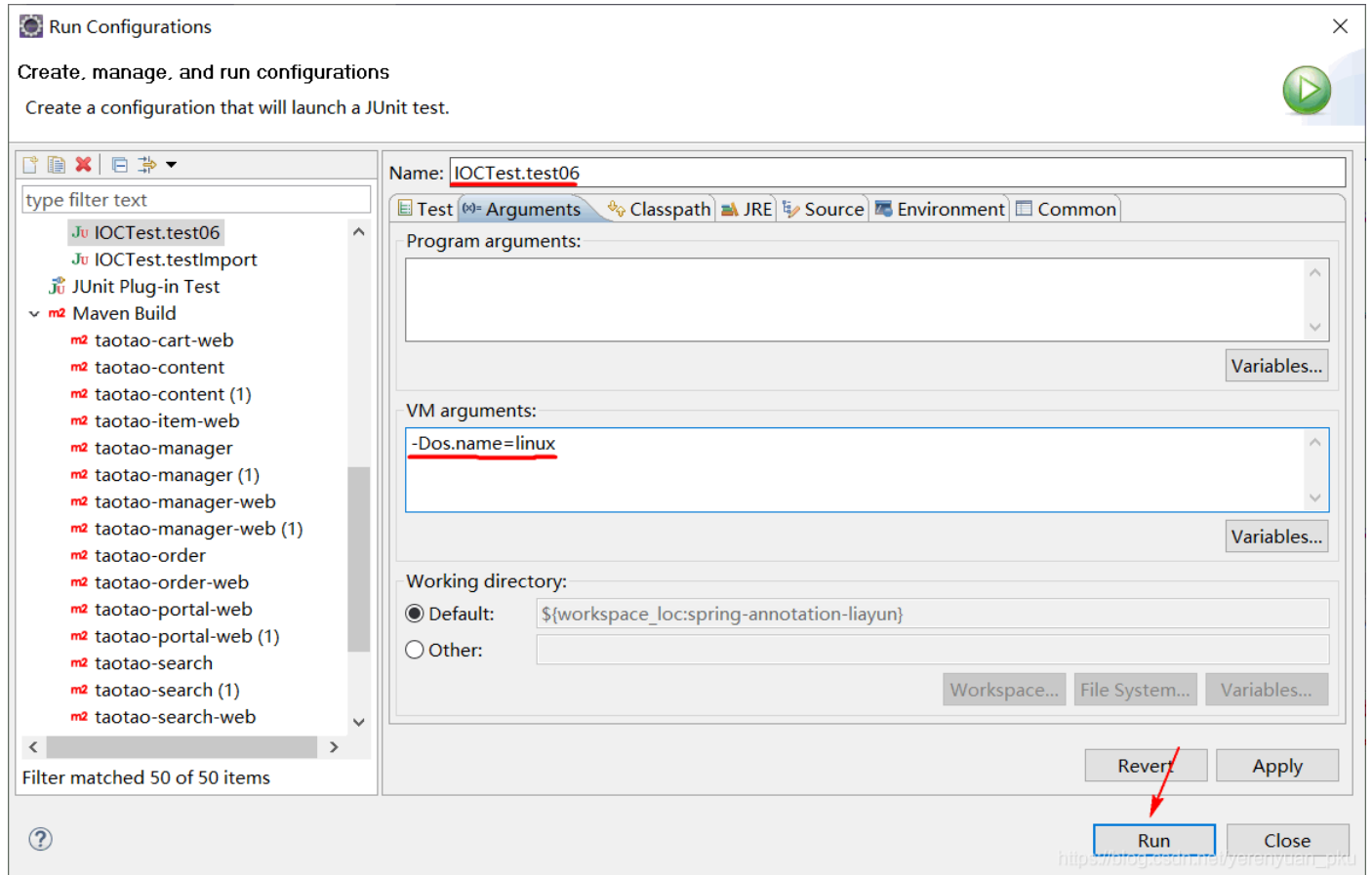
此外，@Conditional注解也可以标注在类上，标注在类上的含义是：只有满足了当前条件，这个配置类中配置的所有bean注册才能生效，也就是对配置类中的组件进行统一设置。

```
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.Lazy;
7
8 import com.meimeixia.bean.Person;
9 import com.meimeixia.condition.LinuxCondition;
10 import com.meimeixia.condition.WindowsCondition;
11
12 // 对配置类中的组件进行统一设置
13 @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
14 @Configuration
15 public class MainConfig2 {
16
17     @Lazy
18     @Bean("person")
19     public Person person() {
20         System.out.println("给容器中添加咱们这个Person对象...");
21         return new Person("美美侠", 25);
22     }
23
24     @Bean("bill")
25     public Person person01() {
26         return new Person("Bill Gates", 62);
27     }
28
29     @Conditional({LinuxCondition.class})
30     @Bean("linus")
31     public Person person02() {
32         return new Person("linus", 48);
33     }
34 }
```

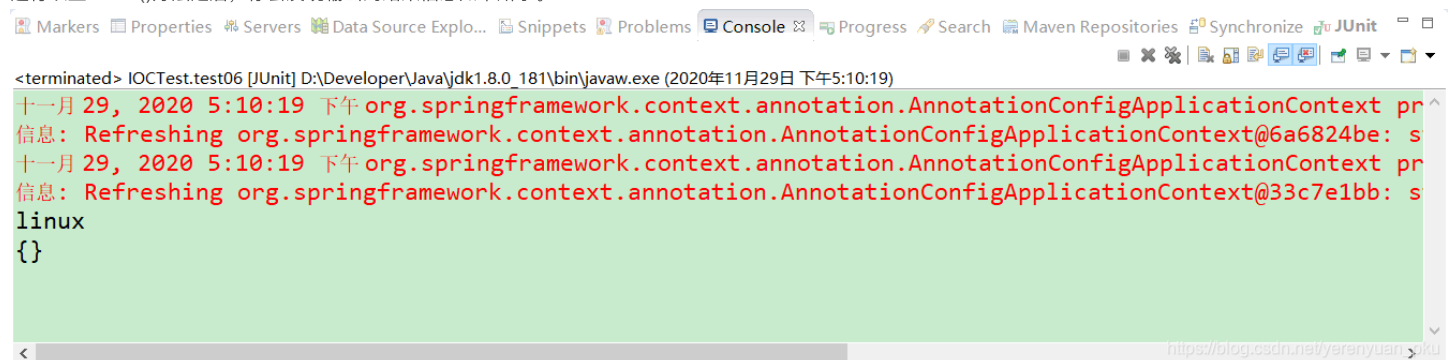


```
33 | }  
34 |  
35 | }  
AI写代码java运行
```

此时，我们在运行IOCTest类中的test06()方法时，设置一个 `-Dos.name=linux` 参数，就像下图所示的那样，这是我们将操作系统模拟为了linux 系统。



运行以上test06()方法之后，你会发现输出的结果信息如下所示。



可以看到，没有任何bean的定义信息输出，这是因为程序检测到了当前操作系统为linux，没有向Spring容器中注册任何bean的缘故导致的。

@Conditional的扩展注解



@Conditional与@Profile这俩注解的对比

Spring 3.0也有一些和@Conditional相似的注解，它们是**Spring SPEL表达式**和**Spring Profiles注解**，但是Spring 4.0之后的@Conditional注解要比@Profile注解更加高级。@Profile注解用来加载应用程序的环境，该注解仅限于根据预定义属性编写条件检查，而@Conditional注解则没有此限制。

Spring中的@Profile和@Conditional这俩注解都是用来检查 **If...then...else** 的语义。然而，Spring 4.0之后的@Conditional注解是@Profile注解的更新用法。

- Spring 3.0中的@Profile仅用于编写基于Environment变量的条件检查。配置文件可用于基于环境加载应用程序配置（这句话好绕口啊😭）。
- Spring 4.0之后的@Conditional注解允许开发人员为条件检查定义用户定义的策略。此外，@Conditional注解还可以用于条件bean注册。