

Spring注解驱动开发第18讲——如何使用@Value注解为bean的属性赋值呢？

写在前面

在之前的文章中，我们探讨了如何向Spring的 **IOC容器** 中注册bean组件，并且还讲解了有关bean组件的生命周期的知识。今天，我们就来一起聊聊@Value注解的用法。

@Value注解

Spring中的@Value注解可以为bean中的 **属性赋值** 。我们先来看看@Value注解的源码，如下所示。

```
Value.class
16
17 package org.springframework.beans.factory.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+ * Annotation at the field or method/constructor parameter level
52 @Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER, ElementType.ANNOTATION_TYPE})
53 @Retention(RetentionPolicy.RUNTIME)
54 @Documented
55 public @interface Value {
56
57     /**
58      * The actual value expression: e.g. "${systemProperties.myProp}".
59      */
60     String value();
61
62 }
63
```

从@Value注解的源码中我们可以看出，@Value注解可以标注在字段、方法、参数以及注解上，而且在程序运行期间生效。

@Value注解的用法

不通过配置文件注入属性的情况

通过@Value注解将外部的值动态注入到bean的属性中，一般有如下这几种情况：

- 注入普通字符串

```
1 | @Value("李阿韵")
2 | private String name; // 注入普通字符串
   AI写代码java运行
```

- 注入操作系统属性

```
1 | @Value("${systemProperties['os.name']}")
2 | private String systemPropertiesName; // 注入操作系统属性
   AI写代码java运行
```

- 注入SpEL表达式结果

```
1 | @Value("#{ T(java.lang.Math).random() * 100.0 }")
2 | private double randomNumber; // 注入SpEL表达式结果
   AI写代码java运行
```

- 注入其他bean中属性的值

```
1 | @Value("#{person.name}")
2 | private String username; // 注入其他bean中属性的值，即注入person对象的name属性中的值
   AI写代码java运行
```

- 注入文件资源

```
1 | @Value("classpath:/config.properties")
2 | private Resource resourceFile; // 注入文件资源
   AI写代码java运行
```

- 注入URL资源

```
1 | @Value("http://www.baidu.com")
2 | private Resource url; // 注入URL资源
   AI写代码java运行
```

通过配置文件注入属性的情况

首先，我们可以在项目的src/main/resources目录下新建一个属性文件，例如person.properties，其内容如下：

```
1 | person.nickName=美美侠
   AI写代码xml
```

然后，我们新建一个MainConfigOfPropertyValues配置类，并在该类上使用@PropertySource注解读取外部配置文件中的key/value并保存到运行的环境变量中。

```
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.PropertySource;
6 |
7 | import com.meimeixia.bean.Person;
8 |
9 | @PropertySource(value={"classpath:/person.properties"})
10 | @Configuration
11 | public class MainConfigOfPropertyValues {
12 |
13 |     @Bean
14 |     public Person person() {
15 |         return new Person();
16 |     }
17 |
18 | }
```

AI写代码java运行



加载完外部的配置文件以后，接着我们就可以使用 `${key}` 取出配置文件中key所对应的值，并将其注入到bean的属性中了。

```
1 | package com.meimeixia.bean;
2 |
3 | import org.springframework.beans.factory.annotation.Value;
4 |
5 | public class Person {
6 |
7 |     @Value("李阿昀")
8 |     private String name;
9 |     @Value("#{20-2}")
10 |     private Integer age;
11 |
12 |     @Value("${person.nickName}")
13 |     private String nickName; // 昵称
14 |
15 |     public String getNickName() {
16 |         return nickName;
17 |     }
18 |     public void setNickName(String nickName) {
19 |         this.nickName = nickName;
20 |     }
21 |     public String getName() {
22 |         return name;
23 |     }
24 |     public void setName(String name) {
25 |         this.name = name;
26 |     }
27 |     public Integer getAge() {
28 |         return age;
29 |     }
30 |     public void setAge(Integer age) {
31 |         this.age = age;
32 |     }
33 |     public Person(String name, Integer age) {
34 |         super();
35 |         this.name = name;
36 |     }
37 | }
```

```
37         this.age = age;
38     }
39     public Person() {
40         super();
41         // TODO Auto-generated constructor stub
42     }
43     @Override
44     public String toString() {
45         return "Person [name=" + name + ", age=" + age + ", nickName=" + nickName + "];"
46     }
47 }
```

AI写代码java运行



@Value中#{...}和\${...}的区别

我们在这里提供一个测试属性文件，例如advance_value_inject.properties，大致的内容如下所示。

```
1 server.name=server1,server2,server3
2 author.name=liayun
```

AI写代码xml

然后，新建一个AdvanceValueInject类，并在该类上使用@PropertySource注解读取外部属性文件中的key/value并保存到运行的环境变量中，即加载外部的advance_value_inject.properties属性文件。

```
1 package com.meimeixia.bean;
2
3 import org.springframework.context.annotation.PropertySource;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 @PropertySource(value={"classpath:/advance_value_inject.properties"})
8 public class AdvanceValueInject {
9
10     // ...
11
12 }
```

AI写代码java运行



以上准备工作做好之后，下面我们就来看看 \${...} 的用法。

\${...}的用法

{}里面的内容必须符合SpEL表达式，通过@Value("\${spelDefault.value}")我们可以获取属性文件中对应的值，但是如果属性文件中没有这个属性，那么就会报错。不过，我们可以通过赋予默认值来解决这个问题，如下所示。

```
1 @Value("${author.name:meimeixia}")
2 private String name;
```

AI写代码java运行

上述代码的含义是表示向bean的属性中注入属性文件中的author.name属性所对应的值，如果属性文件中没有author.name这个属性，那么便向bean的属性中注入默认值meimeixia。

#{...}的用法

{}里面的内容同样也是必须符合SpEL表达式。例如，

```
1 // SpEL: 调用字符串Hello World的concat方法
2 @Value("#{Hello World'.concat('!')}")
3 private String helloWorld;
4
5 // SpEL: 调用字符串的getBytes方法，然后再调用其length属性
6 @Value("#{Hello World'.bytes.length}")
7 private String helloWorldBytes;
```

AI写代码java运行

\${...}和#{...}的混合使用

`\${...}` 和 `#{...}` 可以混合使用，例如，

```
1 // SpEL: 传入一个字符串, 根据", " 切分后插入列表中, #{} 和 ${} 配合使用时, 注意不能反过来 ${} 在外面, 而 #{} 在里面
2 @Value("#{${server.name}'.split(',')}")
3 private List<String> servers;
AI写代码java运行
```

上面片段的代码的执行顺序：通过 `${server.name}` 从属性文件中获取值并进行替换，然后就变成了执行SpEL表达式 `'server1,server2,server3'.split(',')`。

在上文中 `#{}` 在外面，`${}` 在里面可以执行成功，那么反过来是否可以呢？也就是说能否让 `${}` 在外面，`#{}` 在里面，就像下面这样呢？

```
1 // SpEL: 注意不能反过来, ${} 在外面, 而 #{} 在里面, 因为这样会执行失败
2 @Value("${#{'HelloWorld'.concat('_')}}")
3 private List<String> servers2;
AI写代码java运行
```

答案是不能。因为Spring执行 `${}` 的时机要早于 `#{}` ，当Spring执行外层的 `${}` 时，内部的 `#{}` 为空，所以会执行失败！

小结

- `#{...}`：用于执行SpEL表达式，并将内容赋值给属性
- `${...}`：主要用于加载外部属性文件中的值
- `${...}` 和 `#{...}` 可以混合使用，但是必须 `#{}` 在外面，`${}` 在里面

@Value注解案例

这里，我们还是以一个小案例的形式来说明。

首先，我们创建一个Person类来作为测试用的bean组件，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
```



然后，创建一个新的配置类，例如MainConfigOfPropertyValues，用来配置Spring的bean组件。我们在该配置类中将Person类的对象注册到IOC容器中了，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 -
```

```
6 import com.meimeixia.bean.Person;
7
8 @Configuration
9 public class MainConfigOfPropertyValues {
10
11     @Bean
12     public Person person() {
13         return new Person();
14     }
15 }
16 }
```

AI写代码java运行



接着，我们再来创建一个测试类，例如IOCTest_PropertyValue，在该测试类中创建一个test01()测试方法，在该测试方法中我们所要做的事情就是通过MainConfigOfPropertyValues配置类来创建AnnotationConfigApplicationContext对象，并打印出目前IOC容器中存在的组件的名称，如下所示。

```
1 package com.meimeixia.test;
2
3 import org.junit.Test;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 import com.meimeixia.config.MainConfigOfPropertyValues;
7
8 public class IOCTest_PropertyValue {
9
10     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfPropertyValues.class);
11
12     @Test
13     public void test01() {
14         printBeans(applicationContext);
15         // 关闭容器
16         applicationContext.close();
17     }
18
19     private void printBeans(AnnotationConfigApplicationContext applicationContext) {
20         String[] definitionNames = applicationContext.getBeanDefinitionNames();
21         for (String name : definitionNames) {
22             System.out.println(name);
23         }
24     }
25 }
26 }
```

AI写代码java运行



紧接着，我们运行IOCTest_PropertyValue测试类中的test01()方法，输出的结果信息如下所示。

```
<terminated> IOCTest_PropertyValue.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午3:19:58)
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup date 2020年12月2日 下午3:19:58
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
mainConfigOfPropertyValues
person
十二月 02, 2020 3:19:58 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup date 2020年12月2日 下午3:19:58
```

从输出的结果信息中可以看出，IOC容器中除了Spring框架注册的bean之外，还包含我们自己向IOC容器中注册的bean组件，即mainConfigOfPropertyValues和person。

接下来，我们改造下IOCTest_PropertyValue测试类中的test01()方法，让其输出Person对象的信息，如下所示。

```
1 package com.meimeixia.test;
2
3 import org.junit.Test;
4
```

```

7 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
8
9 import com.meimeixia.bean.Person;
10 import com.meimeixia.config.MainConfigOfPropertyValues;
11
12 public class IOCTest_PropertyValue {
13
14     AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfPropertyValues.class);
15
16     @Test
17     public void test01() {
18         printBeans(applicationContext);
19         System.out.println("=====");
20
21         Person person = (Person) applicationContext.getBean("person");
22         System.out.println(person);
23
24         // 关闭容器
25         applicationContext.close();
26     }
27
28     private void printBeans(AnnotationConfigApplicationContext applicationContext) {
29         String[] definitionNames = applicationContext.getBeanDefinitionNames();
30         for (String name : definitionNames) {
31             System.out.println(name);
32         }
33     }
34 }

```

AI写代码java运行

此时，再次运行以上test01()方法，输出的结果信息如下所示。

```

<terminated> IOCTest_PropertyValue.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午3:23:20)
十二月 02, 2020 3:23:20 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRef
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup c
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
mainConfigOfPropertyValues
person
=====
Person [name=null, age=null]
十二月 02, 2020 3:23:20 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup date

```

可以看到，向IOC容器中注册的Person对象的名字属性为null，age属性也为null。那如何向Person对象中的name属性和age属性赋值呢？此时，Spring中的@Value注解就派上用场了。

如果我们通过XML配置文件为bean的属性赋值，那么可以通过如下配置的方式来实现。

```

1 <bean id="person" class="com.meimeixia.bean.Person">
2     <property name="age" value="18"></property>
3     <property name="name" value="liayun"></property>
4 </bean>

```

AI写代码xml

如果使用注解，那么该如何实现呢？别急，往下看！

我们可以在Person类的属性上使用@Value注解为属性赋值，如下所示。

```

1 package com.meimeixia.bean;
2
3 import org.springframework.beans.factory.annotation.Value;
4
5 public class Person {
6
7

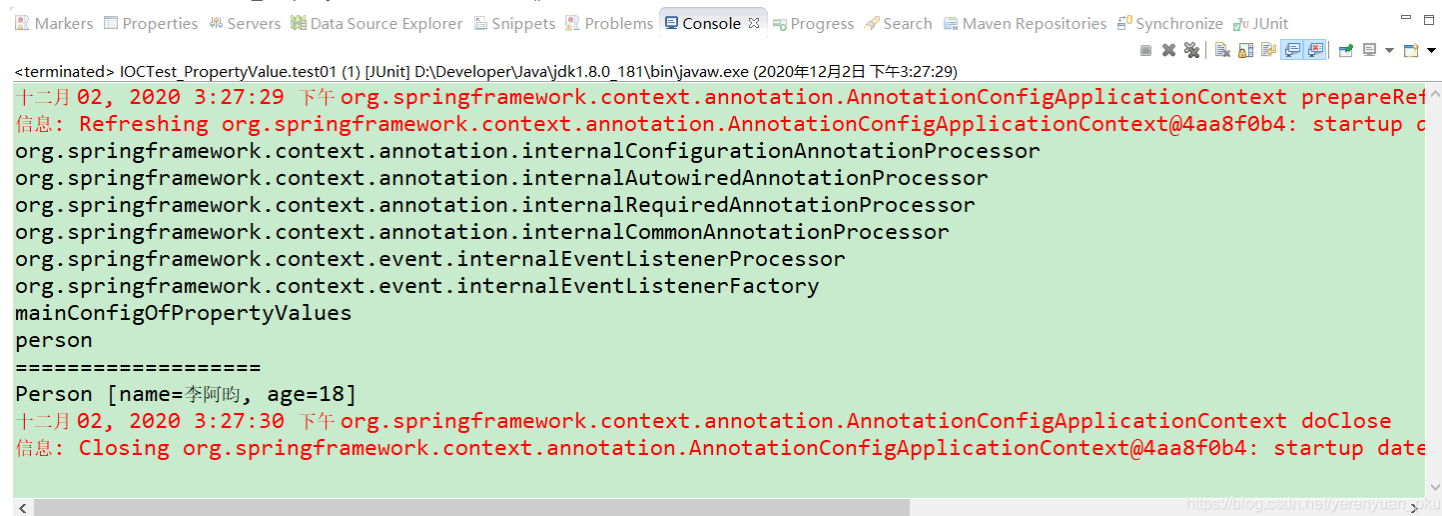
```

```
1  @Value("李阿昀")
2  private String name;
3  @Value("#{20-2}")
4  private Integer age;
5
6  public String getName() {
7      return name;
8  }
9
10 public void setName(String name) {
11     this.name = name;
12 }
13
14 public Integer getAge() {
15     return age;
16 }
17
18 public void setAge(Integer age) {
19     this.age = age;
20 }
21
22 public Person(String name, Integer age) {
23     super();
24     this.name = name;
25     this.age = age;
26 }
27
28 public Person() {
29     super();
30     // TODO Auto-generated constructor stub
31 }
32
33 @Override
34 public String toString() {
35     return "Person [name=" + name + ", age=" + age + "];";
36 }
37
38 }
```

AI写代码java运行



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



```
<terminated> IOCTest_PropertyValue.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月2日 下午3:27:29)
十二月 02, 2020 3:27:29 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRef
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup c
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
mainConfigOfPropertyValues
person
=====
Person [name=李阿昀, age=18]
十二月 02, 2020 3:27:30 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@4aa8f0b4: startup date
```

可以看到，使用@Value注解已经向Person对象的name属性中注入了 李阿昀，向age属性中注入了18。