

依赖注入与自动装配

依赖注入

构造函数注入：

说明：

案例：

setter方法注入：

说明：

案例：

复杂数据类型注入：

短命名空间注入：

自动装配

XML：

byName：

byType：

注解：

@AutoWired：

@Resource：

@Component：

零配置：

案例：

@Value：

依赖注入

构造函数注入：

使用类中的构造函数，给成员变量赋值：

说明：

使用的标签:constructor-arg

标签出现的位置：bean标签的内部

标签中的属性：

1. type：用于指定要注入的数据的数据类型，该数据类型也是构造函数中某个或某些参数的类型。
2. index：用于指定要注入的数据给构造函数中指定索引位置的参数赋值。索引的位置是从0开始。
3. name：用于指定给构造函数中指定名称的参数赋值。
4. value：用于提供基本类型和String类型的数据
5. ref：用于指定其他的bean类型数据。它指的就是在spring的ioc核心容器中出现过的bean对象

优势：

在获取bean对象时，注入数据是必须的操作，否则对象无法创建成功。

弊端：

改变了bean对象的实例化方式，使我们在创建对象时，如果用不到这些数据，也必须提供

案例：

创建类Grade：

```
1  package org.iflytek.bean;
2
3  import org.iflytek.Tools.Logs;
4
5  public class Grade {
6      private Integer gradeId;
7      private String gradeName;
8      public Grade(Integer gradeId, String gradeName) {
9          System.out.println("正在执行 Grade 的有参构造方法, 参数分别为: gradeId="
+ gradeId + ",gradeName=" + gradeName);
10         this.gradeId = gradeId;
11         this.gradeName = gradeName;
12     }
13
14     public Grade(){
15
16     }
17
18     @Override
19     public String toString() {
20         return "Grade{" +
21             "gradeId=" + gradeId +
22             ", gradeName='" + gradeName + '\'' +
23             '}';
24     }
25
26
27 }
```

创建类Student:

```

1  package org.iflytek.bean;
2
3  import org.iflytek.Tools.Logs;
4
5  public class Student {
6      private int id;
7      private String name;
8      private Grade grade;
9      public Student(int id, String name, Grade grade) {
10         System.out.println("参数为: " + id + ", " + name + ", " + grade);
11         this.id = id;
12         this.name = name;
13         this.grade = grade;
14     }
15
16     public Student() {
17     }
18
19     @Override
20     public String toString() {
21         return "Student{" +
22             "id=" + id +
23             ", name='" + name + '\'' +
24             ", grade=" + grade +
25             '}';
26     }
27
28 }

```

spring配置文件中增加配置：

```

1  <bean id="student" class="org.iflytek.bean.Student">
2      <constructor-arg name="id" value="你的学号"></constructor-arg>
3      <constructor-arg name="name" value="你的姓名"></constructor-arg>
4      <constructor-arg name="grade" ref="grade"></constructor-arg>
5  </bean>
6  <bean id="grade" class="org.iflytek.bean.Grade">
7      <constructor-arg name="gradeId" value="6"></constructor-arg>
8      <constructor-arg name="gradeName" value="大三下"></constructor-arg>
9  </bean>

```

验证：

```

1  ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml"
   );
2  //获取名为 employee 的 Bean
3  Student student = context.getBean("student", Student.class);
4  //通过日志打印信息
5  Logs.info(student.toString());

```

控制台输出：

```

✓ MyBean      308 ms  正在执行 Grade  的有参构造方法，参数分别为： gradeId=6, gradeName=大三下
                    参数为： 1111, 你的姓名, Grade{gradeId=6, gradeName='大三下'}
08-03-2024 15:07:00: Student{id=1111, name='你的姓名', grade=Grade{gradeId=6, gradeName='大三下'}}
student
grade
2

```

通过打印的信息得出构造函数被调用。

setter方法注入：

类中提供需要注入成员的 set 方法

说明：

涉及的标签：property

出现的位置：bean标签的内部

标签的属性：

name：用于指定注入时所调用的set方法名称

value：用于提供基本类型和String类型的数据

ref：用于指定其他的bean类型数据。它指的就是在spring的ioc核心容器中出现过的bean对象。

优势：

创建对象时没有明确的限制，可以直接使用默认构造函数。

弊端：

如果有某个成员必须有值，则获取对象是有可能set方法没有执行

案例：

在之前Grade类的基础上增加set方法：

```
1      public void setGradeId(Integer gradeId) {
2          Logs.info("正在执行 Grade 类的 setGradeId() 方法..... ");
3          this.gradeId = gradeId;
4      }
5      public void setGradeName(String gradeName) {
6          Logs.info("正在执行 Grade 类的 setGradeName() 方法..... ");
7          this.gradeName = gradeName;
8      }
```

在之前Student类的基础上增加set方法：

```
1      public void setId(int id) {
2          Logs.info("正在执行 Student 类的 setId() 方法..... ");
3          this.id = id;
4      }
5      public void setName(String name) {
6          Logs.info("正在执行 Student 类的 setName() 方法..... ");
7          this.name = name;
8      }
9      public void setGrade(Grade grade) {
10         Logs.info("正在执行 Student 类的 setGrade() 方法..... ");
11         this.grade = grade;
12     }
```

修改spring配置文件：

```
1  <bean id="student" class="org.iflytek.bean.Student">
2      <property name="id" value="111"></property>
3      <property name="name" value="你的姓名"></property>
4      <property name="grade" ref="grade"></property>
5  </bean>
6
7  <bean id="grade" class="org.iflytek.bean.Grade">
8      <property name="gradeId" value="6"></property>
9      <property name="gradeName" value="大三下"></property>
10 </bean>
```

验证：

```
1  ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
2  //获取名为 employee 的 Bean
3  Student student = context.getBean("student", Student.class);
4  //通过日志打印信息
5  Logs.info(student.toString());
```

控制台输出：

✓ MyBean	412 ms	08-03-2024 15:18:32: 正在执行 Grade 类的 setGradeId() 方法.....
		08-03-2024 15:18:32: 正在执行 Grade 类的 setGradeName() 方法.....
		08-03-2024 15:18:32: 正在执行 Student 类的 setId() 方法.....
		08-03-2024 15:18:32: 正在执行 Student 类的 setName() 方法.....
		08-03-2024 15:18:32: 正在执行 Student 类的 setGrade() 方法.....
		08-03-2024 15:18:32: Student{id=111, name='你的姓名', grade=Grade{gradeId=6, gradeName='大三下'}}}

通过打印的信息得出set方法被调用。

复杂数据类型注入：

创建Person类：

```
1  package org.iflytek.bean;
2
3  import java.util.*;
4
5  public class Person {
6
7      private String name;
8
9      public void setName(String name) {
10         this.name = name;
11     }
12
13     public void setArr(String[] arr) {
14         this.arr = arr;
15     }
16
17     public void setMyList(List<String> myList) {
18         this.myList = myList;
19     }
20
21     public void setMyMap(Map<String, String> myMap) {
22         this.myMap = myMap;
23     }
24
25     public void setMySet(Set<String> mySet) {
26         this.mySet = mySet;
27     }
28
29     public void setMyPro(Properties myPro) {
30         this.myPro = myPro;
31     }
32
33     private String[] arr;
34     private List<String> myList;
35     private Map<String, String> myMap;
36     private Set<String> mySet;
37     private Properties myPro;
38
39
40     @Override
41     public String toString() {
42         return "Student{" +
43             "name='" + name + '\'' +
44             ", arr=" + Arrays.toString(arr) +
45             ", myList=" + myList +
```



```
46         ", myMap=" + myMap +  
47         ", mySet=" + mySet +  
48         ", myPro=" + myPro +  
49         '}'  
50     }  
51 }
```

注：可以借助IDEA快速生成构造函数、Get、Set方法

配置spring配置文件：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
5
6     <bean id="person" class="org.iflytek.bean.Person">
7         <!--普通值注入, value: 具体属性值-->
8         <property name="name" value="你的名字"/>
9
10        <!--数组注入-->
11        <property name="arr">
12            <array>
13                <value>AAA</value>
14                <value>BBB</value>
15                <value>CCC</value>
16            </array>
17        </property>
18
19        <!--List注入-->
20        <property name="myList">
21            <list>
22                <value>111</value>
23                <value>222</value>
24                <value>333</value>
25            </list>
26        </property>
27
28        <!--Map注入-->
29        <property name="myMap">
30            <map>
31                <entry key="aaa" value="aaaa"></entry>
32                <entry key="bbb" value="bbbb"></entry>
33                <entry key="ccc" value="cccc"></entry>
34            </map>
35        </property>
36
37        <!--Set注入-->
38        <property name="mySet">
39            <set>
40                <value>111</value>
41                <value>222</value>
42                <value>333</value>
43            </set>
44        </property>
```

```

45
46      <!--Properties注入-->
47      <property name="myPro">
48          <props>
49              <prop key="aaa">aaaa</prop>
50              <prop key="bbb">bbbb</prop>
51              <prop key="ccc">cccc</prop>
52          </props>
53      </property>
54  </bean>
55
56 </beans>

```

运行+验证:

✓ MyBean	720 ms	08-03-2024 15:29:49: Student{name='jerry', arr=[AAA, BBB, CCC], myList=[111, 222, 333], myMap={aaa=aaaa, bbb=bbbb, person 1
----------	--------	--

短命名空间注入:

基于Grade和Student类, 修改Spring配置:

p:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:p="http://www.springframework.org/schema/p"
5
6      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
7
8      <bean id="employee" class="org.iflytek.bean.Employee" p:empName="小李"
9          p:dept-ref="dept" p:empNo="22222"></bean>
10     <bean id="dept" class="org.iflytek.bean.Dept" p:deptNo="1111" p:deptName="技术部"></bean>
11 </beans>

```

c:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:c="http://www.springframework.org/schema/c"
5
6     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
7     <!-- c 构造函数方法注入 -->
8     <bean id="employee" class="org.iflytek.bean.Employee" c:empName="小胡"
9         c:dept-ref="dept" c:empNo="999"></bean>
10    <bean id="dept" class="org.iflytek.bean.Dept" c:deptNo="2222" c:deptName="测试部"></bean>
11
12 </beans>
```

自动装配

Spring会在上下文中自动寻找，并自动给bean装配属性

XML：

byName：

会自动在容器上下文中查找，和自己对象set方法后面的值对应的beanId

基于Grade类和Student类，修改配置文件：

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4
5
6       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
7
8     <bean id="student" class="org.iflytek.bean.Student" autowire="byName">
9       <property name="id" value="11"></property>
10      <property name="name" value="小郭"></property>
11    </bean>
12    <bean id="grade" class="org.iflytek.bean.Grade">
13      <property name="gradeId" value="6"></property>
14      <property name="gradeName" value="大三下"></property>
15    </bean>
16
17 </beans>

```

运行+验证:

```

✓ MyBean 427 ms 08-03-2024 16:57:21: 正在执行 Grade 类的 setGradeId() 方法.....
08-03-2024 16:57:21: 正在执行 Grade 类的 setGradeName() 方法.....
08-03-2024 16:57:21: 正在执行 Student 类的 setId() 方法.....
08-03-2024 16:57:21: 正在执行 Student 类的 setName() 方法..... |
08-03-2024 16:57:21: 正在执行 Student 类的 setGrade() 方法.....
08-03-2024 16:57:21: Student{id=11, name='小郭', grade=Grade{gradeId=6, gradeName='大三下'}}

```

修改grade对应的bean id为"grade1":

```

✓ MyBean 545 ms 08-03-2024 16:59:46: 正在执行 Student 类的 setId() 方法.....
08-03-2024 16:59:46: 正在执行 Student 类的 setName() 方法.....
08-03-2024 16:59:46: 正在执行 Grade 类的 setGradeId() 方法.....
08-03-2024 16:59:46: 正在执行 Grade 类的 setGradeName() 方法.....
08-03-2024 16:59:46: Student{id=11, name='小郭', grade=null}

```

可以发现并没有实现自动装备，因为在容器中没有找到name是grade的bean。

byType:

会自动在容器上下文中查找，和自己对象属性类型相同的bean

配置:

```

1 <bean id="student" class="org.iflytek.bean.Student" autowire="byType">
2     <property name="id" value="11"></property>
3     <property name="name" value="小郭"></property>
4 </bean>
5 <bean id="grade1" class="org.iflytek.bean.Grade">
6     <property name="gradeId" value="6"></property>
7     <property name="gradeName" value="大三下"></property>
8 </bean>

```

运行+验证：

```

✓ MyBean 454 ms 08-03-2024 17:10:44: 正在执行 Grade 类的 setGradeId() 方法.....
08-03-2024 17:10:44: 正在执行 Grade 类的 setGradeName() 方法.....
08-03-2024 17:10:44: 正在执行 Student 类的 setId() 方法.....
08-03-2024 17:10:44: 正在执行 Student 类的 setName() 方法..... |
08-03-2024 17:10:44: 正在执行 Student 类的 setGrade() 方法.....
08-03-2024 17:10:44: Student{id=11, name='小郭', grade=Grade{gradeId=6, gradeName='大三下'}}

```

尽管Grade对应的bean的id是grade1，但此时是通过类型进行装配，所以可以装配成功。

增加一个新的同类型的bean：

```

1 <bean id="student" class="org.iflytek.bean.Student" autowire="byType">
2     <property name="id" value="11"></property>
3     <property name="name" value="小郭"></property>
4 </bean>
5 <bean id="grade1" class="org.iflytek.bean.Grade">
6     <property name="gradeId" value="6"></property>
7     <property name="gradeName" value="大三下"></property>
8 </bean>
9 <bean id="grade2" class="org.iflytek.bean.Grade">
10     <property name="gradeId" value="6"></property>
11     <property name="gradeName" value="大三下"></property>
12 </bean>

```

运行：

```

! MyBean 377 ms .iflying bean of type 'org.iflytek.bean.Grade' available: expected single matching bean but found 2: grade1,grade2
out found 2: grade1,grade2

```

运行失败，因为找到了两个同类型的bean，不知道选择哪一个完成装配操作。

注解：

@Autowired:

作用：按照类型注入。只要容器中唯一的一个bean对象类型和要注入的变量类型匹配，就可以注入成功。如果ioc容器中没有任何bean的类型和要注入的变量类型匹配，则报错

位置：可以是变量上，也可以是方法上

案例：

创建类Animal：

```
1 public class Animal {
2
3     /////如果显示定义了Autowired的required属性为false, 说明这个对象可以为null, 否则
    不允许为空
4     @Autowired(required = false)
5     private Cat cat;
6     @Autowired(required = false)
7     private Dog dog;
8     @Value("your name")
9     private String name;
10
11     @Override
12     public String toString() {
13         return "Person{" +
14             "cat=" + cat +
15             ", dog=" + dog +
16             ", name='" + name + '\'' +
17             '}';
18     }
19
20 }
```

在其两个属性上添加@Autowired注解；

创建类Cat和Dog：

```
1 public class Dog {
2     private Integer id;
3     private String name;
4
5     public void setId(Integer id) {
6         this.id = id;
7     }
8
9     public void setName(String name) {
10        this.name = name;
11    }
12
13
14    public void shout() {
15        System.out.println("旺旺");
16    }
17
18    @Override
19    public String toString() {
20        return "Dog{" +
21            "id=" + id +
22            ", name='" + name + '\'' +
23            '}';
24    }
25 }
26
27 public class Cat {
28
29     private Integer id;
30     private String name;
31
32     public void setId(Integer id) {
33         this.id = id;
34     }
35
36     public void setName(String name) {
37         this.name = name;
38     }
39
40
41    public void shout() {
42        System.out.println("喵喵");
43    }
44
45    @Override
```



```

46     public String toString() {
47         return "Cat{" +
48             "id=" + id +
49             ", name='" + name + '\'' +
50             '}';
51     }
52 }

```

配置文件：

XML

```

1     <context:annotation-config/>
2
3     <bean id="cat1" class="org.iflytek.bean.Cat" >
4         <property name="id" value="111"/>
5         <property name="name" value="jerry"/>
6     </bean>
7
8     <bean id="dog1" class="org.iflytek.bean.Dog">
9         <property name="id" value="222"/>
10        <property name="name" value="tom"/>
11    </bean>
12
13
14    <bean id="animal" class="org.iflytek.bean.Animal"/>

```

运行+验证：

✓ App (org.iflytek)	387 ms	D:\wangchuang\development\Java\jdk-1.8\bin\java.exe ...
✓ MyBean	387 ms	08-03-2024 19:26:30: Animal{cat=Cat{id=111, name='jerry'}, dog=Dog{id=222, name='tom'}, name='your name'}

cat1和dog1被注入到animal当中；

修改配置文件：

```

1 <bean id="cat1" class="org.iflytek.bean.Cat" >
2     <property name="id" value="111"/>
3     <property name="name" value="jerry"/>
4 </bean>
5 <bean id="dog1" class="org.iflytek.bean.Dog">
6     <property name="id" value="222"/>
7     <property name="name" value="tom"/>
8 </bean>
9 <bean id="cat2" class="org.iflytek.bean.Cat" >
10    <property name="id" value="333"/>
11    <property name="name" value="jerry"/>
12 </bean>
13 <bean id="dog2" class="org.iflytek.bean.Dog">
14    <property name="id" value="444"/>
15    <property name="name" value="tom"/>
16 </bean>
17
18 <bean id="animal" class="org.iflytek.bean.Animal"/>

```

此时会因为同类型的bean不是一个，导致注入失败；

修改注解：

```

1 @Autowired(required = false)
2 @Qualifier(value="cat2")
3 private Cat cat;
4 @Autowired(required = false)
5 @Qualifier(value="dog2")
6 private Dog dog;
7 @Value("your name")
8 private String name;

```

增加@Qualifier，指定唯一的一个bean进行注入：

运行+验证：

✓ App (org.iflytek)	413 ms	D:\wangchuang\development\Java\jdk-1.8\bin\java.exe ...
✓ MyBean	413 ms	08-03-2024 19:31:13: Animal{cat=Cat{id=333, name='jerry'}, dog=Dog{id=444, name='tom'}, name='your name'}

@Resource：

默认通过byName实现注入，如果找不到名字，则通过byType实现。如果两个都找不到的情况下，就报错

案例：

采用上个案例中最后的配置，修改注解：

▼ Java

```
1    @Resource(name = "cat2")
2    private Cat cat;
3    @Resource(name = "dog2")
4    private Dog dog;
5    @Value("your name")
6    private String name;
```

运行+验证：

✓ App (org.iflytek)	362 ms	D:\wangchuang\development\Java\jdk-1.8\bin\java.exe ...
✓ MyBean	362 ms	08-03-2024 19:37:57: Animal{cat=Cat{id=333, name='jerry'}, dog=Dog{id=444, name='tom'}, name='your name'}

将配置中的cat2和dog2删除，只保留cat1和dog1，此时通过byName的方法找不到对应的bean，会接着采用byType查找bean，找到后完成注入：

▼ Java

```
1    @Resource
2    private Cat cat;
3    @Resource
4    private Dog dog;
5    @Value("your name")
6    private String name;
```

@Component：

@Component注解用于把当前类对象存入Spring容器中，把资源让spring来管理，相当于在xml中配置了一个bean；@Controller、@Service、@Repository这三个注解的作用和属性与Component注解是一模一样的。主要是Spring框架提供明确的三层架构使用的注解，使三层对象更加清晰

修改Animal类代码：类上增加Component注解

```
1  @Component
2  public class Animal {
3
4      /////如果显示定义了Autowired的required属性为false, 说明这个对象可以为null, 否则
      不允许为空
5      @Resource
6      private Cat cat;
7      @Resource
8      private Dog dog;
9      @Value("your name")
10     private String name;
11
12     @Override
13     public String toString() {
14         return "Animal{" +
15             "cat=" + cat +
16             ", dog=" + dog +
17             ", name='" + name + '\'' +
18             '}';
19     }
20 }
```

修改配置:

增加component-scan标签:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
6
7     <context:component-scan base-package="org.iflytek.bean"></context:component-scan>
8
9     <bean id="cat1" class="org.iflytek.bean.Cat" >
10         <property name="id" value="111"/>
11         <property name="name" value="jerry"/>
12     </bean>
13     <bean id="dog1" class="org.iflytek.bean.Dog">
14         <property name="id" value="222"/>
15         <property name="name" value="tom"/>
16     </bean>
17
18 </beans>

```

运行+验证:

✓ App (org.iflytek)	585 ms	D:\wangchuang\development\Java\jdk-1.8\bin\java.exe ...
✓ MyBean	585 ms	08-03-2024 19:47:48: Animal{cat=Cat{id=111, name='jerry'}, dog=Dog{id=222, name='tom'}, name='your name'} animal

零配置:

案例:

Animal:

```
1  @Component
2  public class Animal {
3
4      /////如果显示定义了Autowired的required属性为false, 说明这个对象可以为null, 否则
      不允许为空
5      @Resource
6      private Cat cat;
7      @Resource
8      private Dog dog;
9      @Value("your name")
10     private String name;
11
12     @Override
13     public String toString() {
14         return "Animal{" +
15             "cat=" + cat +
16             ", dog=" + dog +
17             ", name='" + name + '\'' +
18             '}';
19     }
20 }
```

Cat:

```
1  @Component
2  public class Cat {
3
4      @Value("222")
5      private Integer id;
6      @Value("jerry")
7      private String name;
8
9      public void setId(Integer id) {
10         this.id = id;
11     }
12
13     public void setName(String name) {
14         this.name = name;
15     }
16
17
18     public void shout() {
19         System.out.println("喵喵");
20     }
21
22     @Override
23     public String toString() {
24         return "Cat{" +
25             "id=" + id +
26             ", name='" + name + '\'' +
27             '}';
28     }
29 }
```

Dog:

```
1  @Component
2  public class Dog {
3      @Value("111")
4      private Integer id;
5      @Value("tom")
6      private String name;
7
8      public void setId(Integer id) {
9          this.id = id;
10     }
11
12     public void setName(String name) {
13         this.name = name;
14     }
15
16
17     public void shout() {
18         System.out.println("旺旺");
19     }
20
21     @Override
22     public String toString() {
23         return "Dog{" +
24             "id=" + id +
25             ", name='" + name + '\'' +
26             '}';
27     }
28 }
```

创建一个配置类：

```
1  @Configuration
2  @ComponentScan(basePackages = "org.iflytek.bean")
3  public class SpringConfig {
4  }
```

启动程序：


```
Java |
1      ApplicationContext context = new AnnotationConfigApplicationContext
    (SpringConfig.class);
2      //获取名为 employee 的 Bean
3      Animal animal = context.getBean("animal", Animal.class);
4  //      //通过日志打印信息
5      Logs.info(animal.toString());
```

运行：

```
✓ App (org.iflytek) 355 ms D:\wangchuang\development\Java\jdk-1.8\bin\java.exe ...
✓ MyBean 355 ms 08-03-2024 20:19:25: Animal{cat=Cat{id=222, name='jerry'}, dog=Dog{id=111, name='tom'}, name='your name'}
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
```

@Value：

主要用于赋值，该值可以是取值配置文件中的，也可以直接赋值，也可以使用SpEl表达式进行计算的结果，抑或直接从环境变量中获取。该注解不能处理日期类赋值

@Value和@PropertySource组合读取配置文件的值：

修改配置类：

```
Java |
1  @Configuration
2  @PropertySource("classpath:/application.properties")
3  @ComponentScan(basePackages = "org.iflytek.bean")
4  public class SpringConfig {
5
6  }
```

在Cat和Dog类中修改@Value注解：

```
Java |
1      @Value("222")
2      private Integer id;
3      @Value("${exec.cat}")
4      private String name;
5
6      @Value("111")
7      private Integer id;
8      @Value("${exec.dog}")
9      private String name;
```

