# 依赖注入(DI)

# 1 构造器注入

详见 04 IOC创建对象的方式

# 2 Set方式注入 (重要)

### 2.1 概念

• 依赖: bean对象的创建依赖于容器

• 注入: bean对象中的所有属性由容器来注入

# 2.2 搭建环境(使用最简单的value值注入)

POJO

```
1 package com.kuang.pojo;
2 import java.io.Serializable;
4 public class Address implements Serializable {
6 private String address;
7 /*....getter和setter....
9 .....toString方法....*/
11 }
```

```
package com.kuang.pojo;
 2
    import java.io.Serializable;
 4
    import java.util.*;
 5
 6
    public class Student implements Serializable {
 7
        private String name;
 8
 9
        //引用类型
10
        private Address address;
11
12
13
        private String[] books;
14
        //集合
15
16
        private List<String> hobbys;
17
        private Map<String, String> cards;
18
19
20
        private Set<String> games;
21
22
        private Properties info;
23
        private String wife;
24
```

• xml

```
<?xml version="1.0" encoding="UTF-8"?>
 1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xsi:schemaLocation="http://www.springframework.org/schema/beans"
 5
            https://www.springframework.org/schema/beans/spring-beans.xsd">
 6
 7
        <bean id="student" class="com.kuang.pojo.Student">
 8
            <!--第一种注入。普通注入, value-->
            roperty name="name" value="Geekst"/>
 9
10
        </bean>
11
12
13
    </beans>
```

• 测试运行

```
1  @Test
2  public void test(){
3     ApplicationContext context = new
    ClassPathXmlApplicationContext("beans.xml");
4     Student student = (Student)context.getBean("student");
5     System.out.println(student.getName());
6
7  }
```

Geekst

# 3 拓展方式

### 3.1 注入Bean

```
1 <!--第二种注入: 注入Bean-->
2 <property name="address" ref="address"/>
```

### 3.2 数组

```
1 <!--数组注入
2
   array
3
   value-->
4
   property name="books">
5
     <array>
6
        <value>红楼梦</value>
7
          <value>水浒传</value>
8
          <value>三国演义</value>
9
          <value>西游记</value>
     </array>
10
11 </property>
```

### 3.3 List

### 3.4 Map

#### 3.5 Set

# 3.6 空值NULL

# 3.7 Properties

```
<!--Properties
2
   key = value
3
4
   cproperty name="info">
5
      ops>
6
          prop key="学号">201401010203
7
           key="姓名">Geekst>
8
           prop key="性别">男</prop>
9
       </props>
10
   </property>
```

#### 测试运行:

```
1
     @Test
  2
         public void test(){
  3
             ApplicationContext context = new
     ClassPathXmlApplicationContext("beans.xml");
  4
             Student student = (Student)context.getBean("student");
  5
             System.out.println(student.toString());
  6
  7
         }
//
//
     Student{
      name=Geekst,
      address=Address{address='哈尔滨'},
```

```
// name=Geekst,
// address=Address{address='哈尔滨'},
// books=[红楼梦, 水浒传, 三国演义, 西游记],
// hobbys=[听歌, 打代码, 看电影],
// cards={身份证=232213131, 银行卡=1231252475},
// games=[LOL, 王者荣耀, 和平精英],
// info={学号=201401010203, 性别=男, 姓名=Geekst},
// wife=null
// }
```

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
Student{name=Geekst, address=Address(address= 哈尔滨'}, books=[紅楼梦,水浒传,三周演义,西游记], hobbys=[所歌,打代码,春电影], cards={身份证=232213131,
银行卡=1231252475}, games=[LOL,王者荣耀,和平精英], info={学号=201401010203, 性别=男,姓名=Geekst}, wife=null}
Process finished with exit code 0
```

## 3.8 P命名空间

```
package com.kuang.pojo;

import java.io.Serializable;

public class User implements Serializable {
   private String name;
   private int age;
```

```
9
       public User() {
10
       }
11
       public User(String name, int age) {
12
13
           this.name = name;
14
           this.age = age;
15
16
       /*.....getter和setter.....
17
18
       ......*/
19
20 }
```

- 先导入约束: xmlns:p="http://www.springframework.org/schema/p"
- xml:

```
1 <!--p命名空间注入
2 可以直接注入属性的值: property-->
3 <bean id="user" class="com.kuang.pojo.User" p:age="18" p:name="lili"/>
```

• 测试运行:

```
1  @Test
2  public void testUser() {
3    ApplicationContext context = new
    ClassPathXmlApplicationContext("userbeans.xml");
4    User user = context.getBean("user", User.class);
5    System.out.println(user);
7  }
```

```
User{name='lili', age=18}
Process finished with exit code 0
```

### 3.9 C命名空间

- 先导入约束: xmlns:c="http://www.springframework.org/schema/c"
- xml:

```
1 <!--c命名空间注入
2 通过构造器参数注入: construct-args-->
3 <bean id="user2" class="com.kuang.pojo.User" c:age="21" c:name="jack"/>
```

• 运行测试:

```
1    @Test
2    public void testC(){
3         ApplicationContext context = new
         ClassPathXmlApplicationContext("userbeans.xml");
4         User user = context.getBean("user2", User.class);
5         System.out.println(user);
7    }
```

```
User{name='jack', age=21}
Process finished with exit code 0
```

# 4 bean的作用域

范围	描述
singleton	(默认)为每个 Spring IoC 容器的单个 object 实例定义单个 bean 定义。
原型	为任意数量的 object 实例定义单个 bean 定义。
请求	将单个 bean 定义范围限定为单个 HTTP 请求的生命周期。也就是说,每个 HTTP 请求都有自己的 bean 实例,该实例是在单个 bean 定义的后面创建的。仅在 web-aware Spring ApplicationContext 的 context 中有效。
session	将单个 bean 定义范围限定为 HTTP Session 的生命周期。仅在 web-aware Spring ApplicationContext 的 context 中有效。
应用	将单个 bean 定义范围限定为 ServletContext 的生命周期。仅在 web-aware Spring ApplicationContext 的 context 中有效。
WebSocket	将单个 bean 定义范围限定为 WebSocket 的生命周期。仅在 web-aware Spring ApplicationContext 的 context 中有效。

```
scope="">

#prototype

#request

#session

#singleton

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot after
```

# 4.1 单例模式 (Spring默认机制)

默认作用域是: 单例模式singleton

```
1 | <bean id="student" class="com.kuang.pojo.Student" scope="singleton"/>
```

```
@Test
2
  public void testC(){
3
       ApplicationContext context = new
   ClassPathXmlApplicationContext("beans.xml");
4
       Student student = context.getBean("student", Student.class);
       Student student2 = context.getBean("student", Student.class);
5
6
7
       System.out.println(student == student2);
8
9
  }
```

```
true
Process finished with exit code 0
```

## 4.2 原型模式

每次从容器中get的时候,都会产生一个新的对象。

```
1 | <bean id="student" class="com.kuang.pojo.Student" scope="prototype"/>
```

```
false

Process finished with exit code 0
```

【其余的模式,只能在web开发中使用。】