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

```
package com.kuang.pojo;

import java.io.Serializable;

public class Address implements Serializable {
    private String address;

    /*......getter和setter......

    ......toString方法......*/
}
```

```
package com.kuang.pojo;

import java.io.Serializable;
import java.util.*;

public class Student implements Serializable {
    private String name;

    //引用类型
    private Address address;

    //数组
    private String[] books;

    //集合
    private List<String> hobbys;

    private Map<String, String> cards;

    private Set<String> games;

    private Properties info;
    private String wife;

```

```
    /*......getter和setter......

    ......toString方法......*/
}
```

- xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="student" class="com.kuang.pojo.Student">
        <!--第一种注入。普通注入，value-->
        <property name="name" value="Geekst"/>
    </bean>



</beans>
```

- 测试运行

```
@Test
public void test(){
    ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
    Student student = (Student)context.getBean("student");
    System.out.println(student.getName());

}
```

```
Geekst
```

3 拓展方式

```
<bean id="address" class="com.kuang.pojo.Address">
    <property name="address" value="哈尔滨"/>
</bean>
```

3.1 注入Bean

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

3.2 数组

```
<!--数组注入
array
value-->
<property name="books">
    <array>
        <value>红楼梦</value>
        <value>水浒传</value>
        <value>三国演义</value>
        <value>西游记</value>
    </array>
</property>
```

3.3 List

```
<!--List-->
<property name="hobbys">
    <list>
        <value>听歌</value>
        <value>打代码</value>
        <value>看电影</value>
    </list>
</property>
```

3.4 Map

```
<!--Map-->
<property name="cards">
    <map>
        <entry key="身份证" value="232213131"/>
        <entry key="银行卡" value="1231252475"/>
    </map>
</property>
```

3.5 Set

```
<!--Set-->
<property name="games">
    <set>
        <value>LOL</value>
        <value>王者荣耀</value>
        <value>和平精英</value>
    </set>
</property>
```

3.6 空值NULL

```
<!--NULL值注入（注意：不是空字符串）
空字符串用===》value=""
-->
<property name="wife">
    <null/>
</property>
```

3.7 Properties

```xml
<!--Properties
key = value
-->
<property name="info">
    <props>
        <prop key="学号">201401010203</prop>
        <prop key="姓名">Geekst</prop>
        <prop key="性别">男</prop>
    </props>
</property>
```

测试运行：

```
@Test
    public void test(){
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
        Student student = (Student)context.getBean("student");
        System.out.println(student.toString());

    }
```

```
//
//    Student{
//     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=[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}

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;

```

```
    public User() {
    }

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    /*......getter和setter......

    ......toString方法......*/

}
```

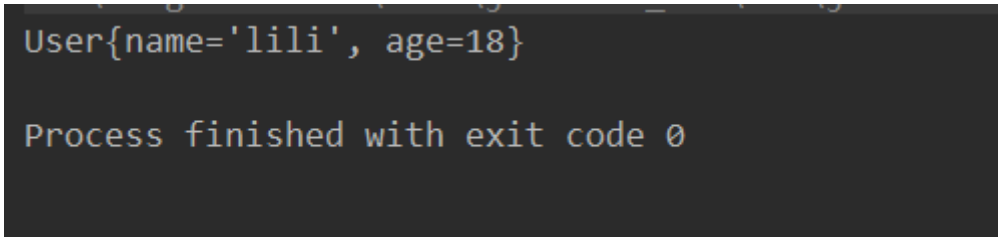
- 先导入约束： xmlns:p="http://www.springframework.org/schema/p"
- xml：

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

- 测试运行：

```
@Test
public void testUser() {
    ApplicationContext context = new
ClassPathXmlApplicationContext("userbeans.xml");
    User user = context.getBean("user", User.class);

    System.out.println(user);
}
```



3.9 C命名空间

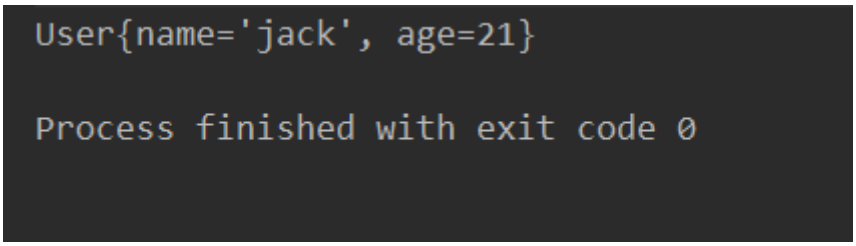
- 先导入约束：xmlns:c="http://www.springframework.org/schema/c"
- xml：

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

- 运行测试：

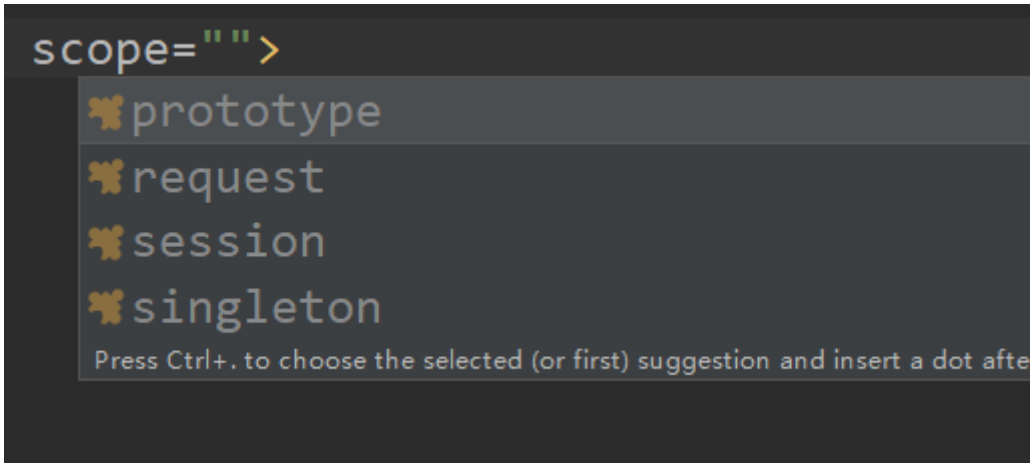
```java
@Test
public void testC(){
    ApplicationContext context = new
ClassPathXmlApplicationContext("userbeans.xml");
    User user = context.getBean("user2", User.class);

    System.out.println(user);
}
```



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。



4.1 单例模式（Spring默认机制）

默认作用域是：单例模式singleton

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

```java
@Test
public void testC(){
    ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
    Student student = context.getBean("student", Student.class);
    Student student2 = context.getBean("student", Student.class);

    System.out.println(student == student2);

}
```

```
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。

```xml
<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。】