**一、主要内容**

**1.底层原理**

1.什么是IOC：

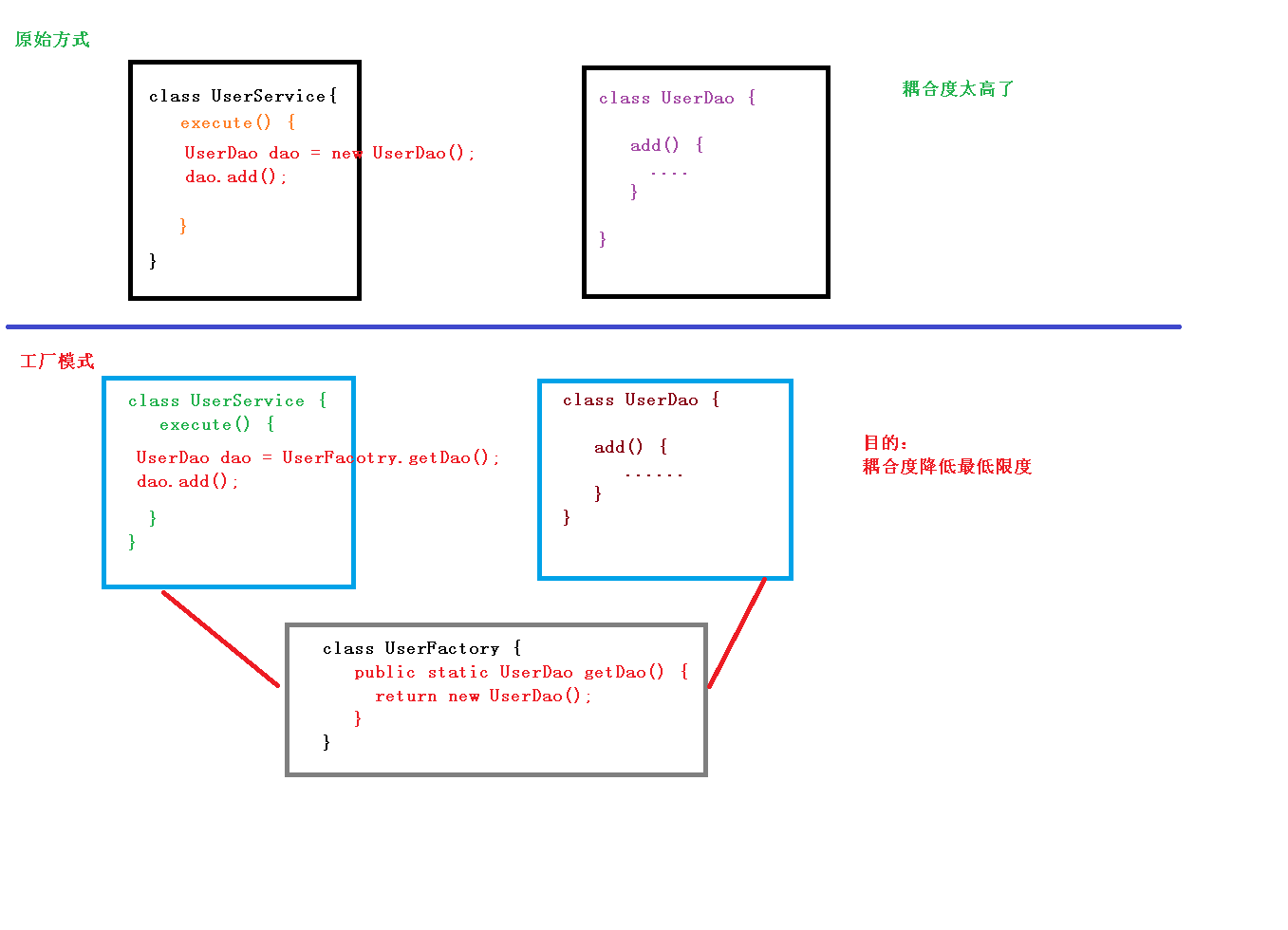
（1）控制反转，把对象创建和对象之间的调用过程，交给Spring管理

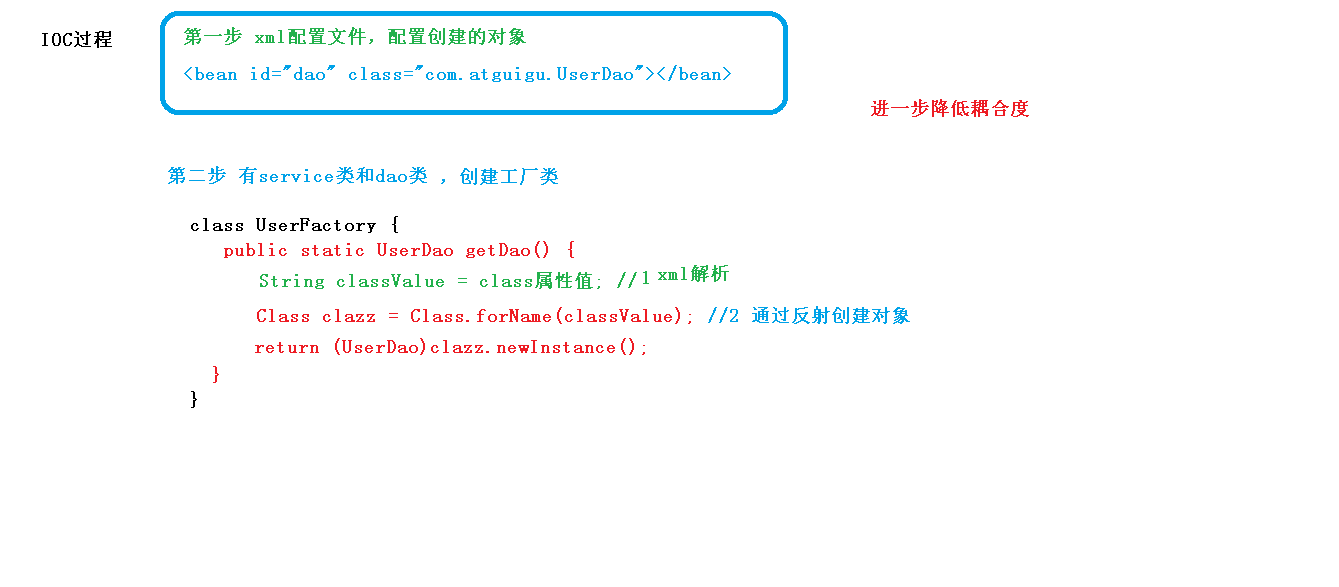
（2）目的：解耦，降低耦合度

（3）入门案例-》简单实现

2.原理：

工厂模式， xml文件，注解





**2.IOC接口(BeanFactory)**

1.IOC思想基于IOC容器完成，底层就是对象工厂

2.Spring提供实现IOC容器的两个接口：

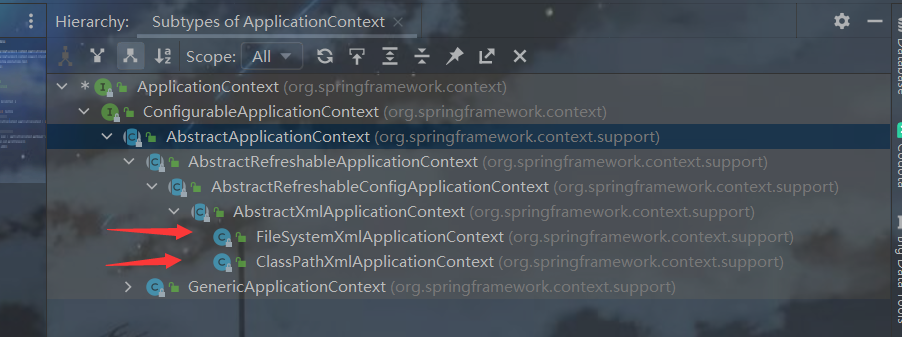
(1)BeanFactory：IOC容器基本实现，是Spring内部的使用接口，不提供开发人员进行使用。

\*加载配置文件时候不会创建对象，在获取对象（使用）才去创建对象

(2)ApplicationContext:BeanFactory接口的子接口,提供更多更强大的功能，一般由开发人员进行使用

\*在加载配置文件时就会在配置文件中加载对象(一般情况下选择此接口，可节约资源)

3.ApplicationContext接口有实现类:



**3.IOC操作Bean管理(基于配置文件)**

**1.什么是bean管理**

（1）bean管理指的是两个操作

（2）Spring创建对象

（3）Spring注入属性

**2.基于两种方式**

**（1）XML配置文件**

**DI: 依赖注入，就是注入属性**

1）注入方式一：使用set方法进行注入

@1 创建类，定义属性和set方法

package xh;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.testng.annotations.Test;

/\*\*

\* @author xh

\* @date 2022/9/29

\* @apiNote

\*/

public class Book {

private String bName;

private String bAuthor;

public void setbName(String bName) {

this.bName = bName;

}

public void setbAuthor(String bAuthor) {

this.bAuthor = bAuthor;

}

public void testDemo() {

System.out.println(bName);

System.out.println(bAuthor);

}

@Test

public void test(){

//读取配置文件

ApplicationContext context = new ClassPathXmlApplicationContext("bean1.xml");

//获取对象及属性

Book book = context.getBean("book",Book.class);

book.testDemo();

}

}

@2 编写配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="aaa" class="xh.User"></bean>

<bean id="book" class="xh.Book">

//property添加属性，name，value

<property name="bAuthor" value="余华"></p[roperty>](http://www.springframework.org/schema/beans)

<property name="bName" value="《活着》"></p[roperty>](http://www.springframework.org/schema/beans)

</bean>

</beans>

**2）注入方式二：使用有参构造方法方法进行注入(在xml配置文件bean对象中使用'constructor-arg'属性)**

@1 创建school类，使用有参构造传入参数

package xh;

/\*\*

\* @author xh

\* @date 2022/10/8

\* @apiNote

\*/

public class School {

public String student;

public String teacher;

//有参构造传入参数

public School(String student, String teacher) {

this.student = student;

this.teacher = teacher;

}

public void update(){

System.out.println("update...");

System.out.println(student);

System.out.println(teacher);

}

}

@2 编写配置文件，使用<constructor-arg/>标签

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<!--> <bean id="aaa" class="xh.User"></bean>

<bean id="book" class="xh.Book">

<property name="bAuthor" value="余华"></p[roperty>](http://www.springframework.org/schema/beans)

<property name="bName" value="《活着》"></p[roperty>](http://www.springframework.org/schema/beans)

</bean>

</!-->

<bean id="school" class="xh.School">

<constructor-arg name="student" value="[小明"></constructor-arg>](http://www.springframework.org/schema/beans)

<constructor-arg name="teacher" value="[小华"></constructor-arg>](http://www.springframework.org/schema/beans)

</bean>

</beans>

**3.xml注入其他属性**

1.字面量

（1）null值

在<property>标签中添加<null/>标签

<bean id="book" class="xh.Book" p:bName="《丰乳肥臀》" p:bAuthor="莫言">

<property name="address">

<null/>

</property>

</bean>

（2）属性值包含特殊符号

@1使用转义字符 &lt; &gt;

@2把带特殊符号的内容写到CDATA

<bean id="book" class="xh.Book" p:bName="《丰乳肥臀》" p:bAuthor="莫言">

<property name="address">

<value><![CDATA[<<南京>>]]></value>

</property>

</bean>

**4.xml注入外部bean**

(1)创建两个类：service类和dao类

public class UserService {

public UserDaoImpl userDao;

public void setUserDao(UserDaoImpl userDao) {

this.userDao = userDao;

}

public void start() {

System.out.println("service start...");

userDao.update();

}

@Test

public void testDao(){

//读取配置文件

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean2.xml");

//获取配置文件中的对象

UserService userService = (UserService) applicationContext.getBean("service");

userService.start();

}

}

package DAO;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class UserDaoImpl implements UserDao{

@Override

public void update() {

System.out.println("UserDao start...");

}

}

(2)在service中调用dao方法

(3)在Spring配置文件中配置

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="service" class="Service.UserService[">](http://www.springframework.org/schema/beans)

<!--注入userDao对象

name属性:类中属性名称

ref属性：外部bean对象标签ID值

-->

<property name="userDao" ref="userDao"></[property>](http://www.springframework.org/schema/beans)

</bean>

<bean id="userDao" class="DAO.UserDaoImpl"></[bean>](http://www.springframework.org/schema/beans)

</beans>

**5.注入属性-内部bean**

（1）一对多关系：部门和员工

（2）在实体类中表示一对多关系,员工表示所属部门，使用对象类型属性进行表示

package DeptAndEmp;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

//部门类

public class Dept {

private String dname;

public void setDname(String dname) {

this.dname = dname;

}

}

//员工类

public class Emp {

private String ename;

private String gender;

//使用对象表示员工所属的部门

private Dept dept;

//使用get方法获得dept对象，防止配置文件中使用级联报错

public Dept getDept() {

return dept;

}

public void setEname(String ename) {

this.ename = ename;

}

public void setGender(String gender) {

this.gender = gender;

}

public void setDept(Dept dept) {

this.dept = dept;

}

@Override

public String toString() {

return "Emp{" +

"ename='" + ename + '\'' +

", gender='" + gender + '\'' +

", dept=" + dept +

'}';

}

}

编写配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="emp" class="DeptAndEmp.Emp">

<property name="ename" value="Hans"></propert[y>](http://www.springframework.org/schema/beans)

<property name="gender" value="男"></property>

<!-- 使用内部bean，在property内部设置对象属性-->

<property name="dept" >

<bean id="dept" class="DeptAndEmp.Dept">

<property name="dname" value="人力资源部">[</property>](http://www.springframework.org/schema/beans)

</bean>

</property>

</bean>

</beans>

**6.注入属性-级联赋值**

方式一

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="emp" class="DeptAndEmp.Emp">

<property name="ename" value="刘邦"></prope[rty>](http://www.springframework.org/schema/beans)

<property name="gender" value= "男"></prop[erty>](http://www.springframework.org/schema/beans)

<!--级联赋值-->

<property name="dept" ref="dept"></proper[ty>](http://www.springframework.org/schema/beans)

</bean>

<bean id="dept" class="DeptAndEmp.Dept">

<property name="dname" value="技术支持部"></pr[operty>](http://www.springframework.org/schema/beans)

</bean>

</beans>

方式二（此方式要求必须在员工类中使用getDept方法，只有这样才能获得dept对象及其属性）

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="emp" class="DeptAndEmp.Emp">

<property name="ename" value="刘邦"></prope[rty>](http://www.springframework.org/schema/beans)

<property name="gender" value= "男"></prop[erty>](http://www.springframework.org/schema/beans)

<!--级联赋值-->

<property name='dept' ref='dept'></proper[ty>](http://www.springframework.org/schema/beans)

<property name="dept.dname" value="财务部"><[/property>](http://www.springframework.org/schema/beans)

</bean>

<bean id="dept" class="DeptAndEmp.Dept"></bea[n>](http://www.springframework.org/schema/beans)

</beans>

**6.注入集合属性**

**6.1注入集合四种基本属性**

(1）编写stu类，其中有array,list,map,set属性的集合

package Stu;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class Stu {

//String数组属性

private String[] stuName;

//List集合属性

private List<String> course;

//Map集合属性

private Map<String, String> course2;

//set集合属性

private Set<String> teacher;

public void setStuName(String[] stuName) {

this.stuName = stuName;

}

public void setCourse(List<String> course) {

this.course = course;

}

public void setCourse2(Map<String,String> course2) {

this.course2 = course2;

}

public void setTeacher(Set<String> teacher) {

this.teacher = teacher;

}

public void add(){

//使用Arrays工具类的toString方法才能输出数组的值

System.out.println(Arrays.toString(stuName));

System.out.println(course);

System.out.println(course2);

System.out.println(teacher);

}

}

（2）编写配置文件（主要掌握数组，list，set，map这四种属性的写法）

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xmlns:p="http://www.springframework.org/schema[/p"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<bean id="stu" class="Stu.Stu">

<property name="StuName">

<array>

<value>James</value>

<value>Curry</value>

<value>Harden</value>

</array>

</property>

<property name="course">

<list>

<value>数学</value>

<value>英语</value>

<value>语文</value>

</list>

</property>

<property name="course2">

<map>

<entry key="数学" value="Math"></[entry>](http://www.springframework.org/schema/beans)

<entry key="英语" value="English"[></entry>](http://www.springframework.org/schema/beans)

<entry key="语文" value="Chinese"[></entry>](http://www.springframework.org/schema/beans)

</map>

</property>

<property name="teacher">

<set>

<value>Jordan</value>

<value>Kobe</value>

</set>

</property>

</bean>

</beans>

**6.2在集合里面设置对象类型值**

@1创建Grade类

package Stu;

import java.util.List;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class Grade {

public List<String> grade;

public void setGrade(List<String> grade) {

this.grade = grade;

}

@Override

public String toString() {

return "Grade{" +

"grade=" + grade +

'}';

}

}

@2修改配置文件

<bean id="grade" class="Stu.Grade">

<property name="grade">

<list>

<value>四年级</value>

<value>五年级</value>

<value>六年级</value>

</list>

</property>

</bean>

<bean id="stu" class="Stu.Stu">

<property name="grades" ref="grade"></property>

</bean>

**6.3提取公共部分-----集合注入部分**

@1创建Book类

package Stu;

import java.util.List;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class Book{

private List<String> bname;

public void setBname(List<String> bname) {

this.bname = bname;

}

@Override

public String toString() {

return "Book{" +

"bname=" + bname +

'}';

}

public void add(){

System.out.println(bname);

}

}

@2在配置文件中使用util方法来实现list集合注入

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xmlns:util="http://www.springframework.org/sch[ema/util"](http://www.springframework.org/schema/beans)

xsi:schemaLocation= "http://www.springframewo[rk.org/schema/beans http://www.springframew](http://www.springframework.org/schema/beans)ork.org/schema/beans[/spring-beans.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframewo[rk.org/schema/util http://www.springframewo](http://www.springframework.org/schema/beans)rk.org/schema/util/s[pring-util.xsd](http://www.w3.org/2001/XMLSchema-instance)

">

<util:list id="bookList">

<value><![CDATA[《夏洛的网》]]></value>

<value><![CDATA[《假如给我三天光明》]]></value>

<value><![CDATA[《活着》]]></value>

</util:list>

<bean name="book" class="Stu.Book">

<property name="bname" ref="bookList"></prope[rty>](http://www.springframework.org/schema/beans)

</bean>

</beans>

**7.FactoryBean**

Spring中bean分为两类：普通bean和工厂bean

普通bean：返回的对象实例类型和配置文件中定义的类型一样

工厂bean：返回的类型和配置文件中定义的类型不一样

（1）第一步：创建一个类，让这个类作为工厂bean，实现接口FactoryBean

package Stu;

import org.springframework.beans.factory.FactoryBean;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class Fruit implements FactoryBean<Apple> {

}

（2）第二步：实现接口中的方法，在方法中定义返回的类的类型

package Stu;

import org.springframework.beans.factory.FactoryBean;

/\*\*

\* @author xh

\* @date 2022/10/9

\* @apiNote

\*/

public class Fruit implements FactoryBean<Apple> {

//确定返回的对象类型

@Override

public Apple getObject() throws Exception {

Apple apple = new Apple();

apple.setAppleName("abc");

return apple;

}

@Override

public Class<?> getObjectType() {

return null;

}

@Override

public boolean isSingleton() {

return FactoryBean.super.isSingleton();

}

}

**8.Bean作用域**

**8.1 在Spring中可以设置创建单实例还是多实例**

**8.2 默认情况下是单实例对象**

测试案例

@Test

public void test(){

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml");

Stu stu = applicationContext.getBean("stu",Stu.class);

Stu stu2 = applicationContext.getBean("stu",Stu.class);

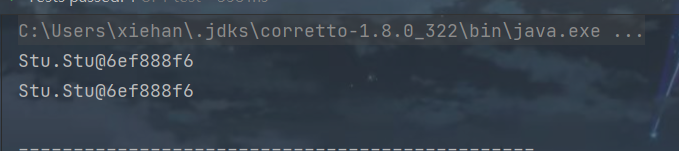
System.out.println(stu);

System.out.println(stu2);

//stu.add();

}

结果显示地址相同



**8.3 如何设置多实例**

（1）Spring配置文件bean标签有scope属性设置单实例还是多实例

（2）Scope属性值

默认值为singleton，表示单实例对象

prototype表示多实例对象

（3）Singleton和scope的区别

singleton:在加载配置文件时创建bean对象

prototype:在调用getBean方法时创建bean对象

**9.bean生命周期**

**9.1生命周期：**

从对象创建到销毁的过程

**9.2bean生命周期**

（1）通过构造器创建bean实例(无参数构造)

（2）为bean的属性设置值和对其他bean的引用（调用set方法）

（3）调用bean的初始化方法（需要进行配置初始化方法 ）

（4）使用bean（已经获取到对象）

（5）当关闭容器时，调用bean的销毁方法（需要进行配置销毁的方法）

**9.3演示bean的生命周期（注意：只有调用context的close方法才会执行bean实例的destroy-method方法，且只有ClassPathXmlApplicationContext类才有close方法）**

@1创建Mybeanr类

package Bean;

/\*\*

\* @author xh

\* @date 2022/10/10

\* @apiNote

\*/

public class MyBean {

private String Bname;

public MyBean() {

System.out.println("第一步:无参构造bean实例");

}

public void setBname(String bname) {

Bname = bname;

System.out.println("第二步:设置属性值");

}

public void initMethod(){

System.out.println("第三步:执行bean的初始化方法");

}

public void destroyMethod(){

System.out.println("第五步:bean的销毁方法");

}

}

@2编写spring配置文件

<bean id="mybean" class="Bean.MyBean" init-method="initMethod" destroy-method="destroyMethod">

<property name="bname" value="aaa"></property>

</bean>

**@3测试**

public class BeanTest {

@Test

public void beanTest(){

ApplicationContext context = new ClassPathXmlApplicationContext("bean4.xml");

MyBean bean = context.getBean("mybean", MyBean.class);

System.out.println("第四步:获取bean实例对象");

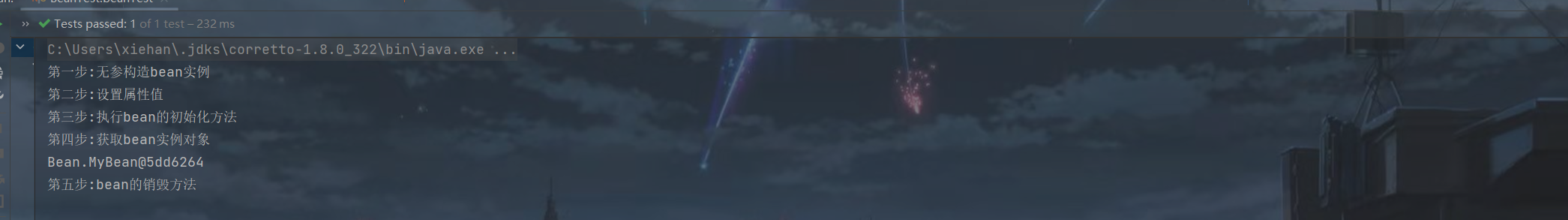
System.out.println(bean);

//销毁bean实例

((ClassPathXmlApplicationContext)context).close();

}

}



**9.4bean的后置处理器（生命周期有7步）**

（1）通过构造器创建bean实例(无参数构造)

（2）为bean的属性设置值和对其他bean的引用（调用set方法）

（3）把bean实例传递给后置处理器的方法postProcessBeforeInitialization

（4）调用bean的初始化方法（需要进行配置初始化方法 ）

（5）把bean实例传递给后置处理器的方法postProcessAfterInitialization

（6）使用bean（已经获取到对象）

（7）当关闭容器时，调用bean的销毁方法（需要进行配置销毁的方法）

**9.5演示添加后置处理器的效果**

@1创建MyBeanPost类，实现BeanPostProcessor接口并重写其中的两个方法

public class MyBeanPost implements BeanPostProcessor {

@Override

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

System.out.println("后置处理器:在初始化之前执行的方法");

return bean;

}

@Override

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {

System.out.println("后置处理器:在初始化之后执行的方法");

return bean;

}

}

**@2在配置文件中添加后置处理器**

<bean id="mybean" class="Bean.MyBean" init-method="initMethod" destroy-method="destroyMethod">

<property name="bname" value="aaa"></property>

</bean>

<!--配置后置处理器

-->

<bean id="mybeanpost" class="Bean.MyBeanPost"></bean>

注意：配置后置处理器后，Spring配置文件中任意一个bean实例对象都可以使用它

**10.xml自动装配**

**10.1 什么是自动装配**

**根据指定装配规则(属性名称或者属性类型)，Spring自动将匹配的属性值进行注入**

**10.2自动装配演示**

**（1）创建Emp类和Dept类**

**package AutoWire;**

**/\*\***

**\* @author xh**

**\* @date 2022/10/**10

**\* @apiNote**

**\*/**

**public class Emp** {

**private Dept** de**pt;**

**public void s**et**Dep**t**(Dept dept) {**

**this.dept** = **de**p**t;**

**}**

**@Override**

**public String** t**oSt**r**ing() {**

**return "E**mp**{"** +

**"**de**pt=**" **+ dept +**

**'**}'**;**

**}**

**}**

**public class Dept** {

**@Override**

**public String** t**oSt**r**ing() {**

**return "D**ep**t{}**"**;**

**}**

**}**

**(2)在配置文件中设置自动装配**

**<?xml version="1.0" encoding="UTF-8"?>**

**<beans xmlns="http://www.springframewo**r**k.org/schema/b**[**eans"**](http://www.springframework.org/schema/beans)

**xmlns:xsi="http://www.w3.org/20**0**1/XMLSchema-in**[**stance"**](http://www.springframework.org/schema/beans)

**xsi:schemaLocation="http://www.**s**pringframework**[**.org/schema/beans http://www.springframewor**](http://www.springframework.org/schema/beans)**k**.**org/schema/beans/s**[**pring-beans.xsd">**](http://www.w3.org/2001/XMLSchema-instance)

**<!--自动装配**

**bean中的属性autowire:**

**autowire常用的有两个值：byName和byType**

**byName:使用另一个bean实例对象中的id值注入,注意**i**d值要和类中定义的属性名称一**[**致**](http://www.springframework.org/schema/beans)

**byType:使用另一个bean中的类名注入，当一个类有多个**b**ean对象时，使用此值会报错**

**-->**

**<bean id="emp" class="AutoWire.Emp**" **autowire="byN**[**am">**](http://www.springframework.org/schema/beans)

**<!--自动装配**

**<property name="dept" ref="dep**t**"></property>**

**-->**

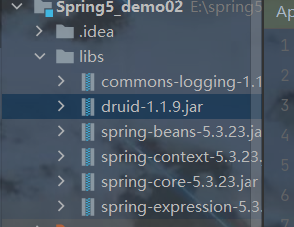
**</bean>**

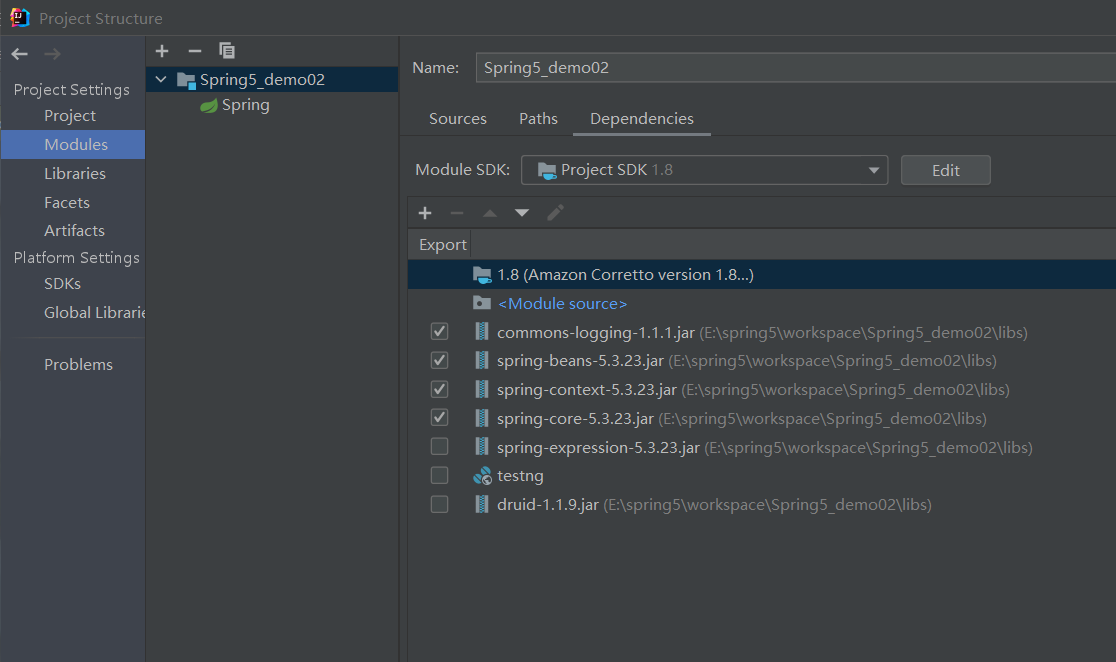
**<bean id="dept" class="AutoWire.De**p**t"></bean>**

**</beans>**

**11.外部属性文件(以配置德鲁伊数据库连接池为例)**

**11.1引入德鲁伊的jar包并添加依赖**





**11.2编写properties配置文件，添加数据库连接池的相关属性**

prop.driverClass=com.mysql.jdbc.Driver

prop.url=jdbc:mysql://localhost:3306/userDb

prop.userName=root

prop.password=root

**11.3在Spring配置文件中引入外部属性文件**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

<!--类似于p标签引入-->

xmlns:context="http://www.springframework.org/[schema/context"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframework[.org/schema/context http://www.springframew](http://www.springframework.org/schema/beans)ork.org/schema/conte[xt/spring-context.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<!--引入外部属性文件-->

<context:property-placeholder location="classpath[\*:jdbc.properties"></context:property-place](http://www.springframework.org/schema/beans)holder>

<bean id="datasource" class="com.alibaba.drui[d.pool.DruidDataSource">](http://www.springframework.org/schema/beans)

<!-- 获取properties文件内容，根据key获取，使用spring表达式[获取 -->](http://www.springframework.org/schema/beans)

<property name="driverClassName" value="$[{prop.driverClass}"></property>](http://www.springframework.org/schema/beans)

<property name="url" value="${prop.url}">[</property>](http://www.springframework.org/schema/beans)

<property name="username" value="${prop.u[serName}"></property>](http://www.springframework.org/schema/beans)

<property name="password" value="${prop.p[assword}"></property>](http://www.springframework.org/schema/beans)

</bean>

</beans>

**4.IOC操作Bean管理（基于注解）**

**1.什么是注解**

（1）注解是代码的特殊标记,格式：@注解名称（属性名=属性值，属性名称=属性值，...）

（2）注解作用范围：类上面，方法上面，属性上面

（3）使用目的：简化xml的配置

**2.Spring针对Bean管理中创建对象提供注解**

（1）@Component

（2）@Service

（3）@Controller

（4）@Repository

\*以上四个注解功能相同，都可用来创建bean实例，但是在实际开发中一般会作用于不同层

**3.基于注解方式实现对象创建**

**3.1引入相关jar包依赖**

截图.png

**3.2在配置文件中开启组件扫描**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xmlns:context="http://www.springframework.org/[schema/context"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframework[.org/schema/context http://www.springframew](http://www.springframework.org/schema/beans)ork.org/schema/conte[xt/spring-context.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<!--开启组件扫描

1.如果扫描多个包，使用逗号隔开

2.扫描包的上层目录

-->

<context:component-scan base-package="Spring5.DAO["></context:component-scan>](http://www.springframework.org/schema/beans)

</beans>

**3.3编写测试类及测试**

package Spring5.DAO;

import org.springframework.stereotype.Component;

/\*\*

\* @author xh

\* @date 2022/10/10

\* @apiNote

\*/

//在注解中value属性值可以不写

//默认是类名称，首字母小写->(value = "demo01")

@Component //此处其他三个注解效果相同

public class Demo01 {

public void add(){

System.out.println("add...");

}

}

//测试注解

@Test

public void testDemo01(){

ApplicationContext context = new ClassPathXmlApplicationContext("bean01.xml");

Demo01 demo01 = context.getBean("demo01", Demo01.class);

System.out.println(demo01);

demo01.add();

}

**4.开启组件扫描细节配置**

<!--

示例1：

use-default-filters="false":表示现在不使用默认filter，自己设置

context: include-filter: 设置扫描哪些内容

-->

<context:component-scan base-package="Spring5.DAO" use-default-filters="false">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Component"/>

</context:component-scan>

<!--

示例2：

use-default-filters="false":表示现在不使用默认filter，自己设置

context: exclude-filter: 设置不扫描哪些内容

-->

<context:component-scan base-package="Spring5.DAO" use-default-filters="false">

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Component"/>

</context:component-scan>

**5.基于注解方式实现属性注入**

（1）@AutoWired：根据属性类型自动装配

第一步：创建UserDaoImpl类和UserService类，在类上方添加@Component获取类的实例对象

@Component

public class UserDaoImpl implements UserDao{

@Override

public void add() {

System.out.println("UserDao add...");

}

}

@Component

public class UserService {

//添加类属性

private UserDao userDao;

public void add(){

System.out.println("service add...");

userDao.add();

}

}

第二步：在service中注入UserDaoImpl对象（通过设置属性的方式），在属性上方添加注解

@Component

public class UserService {

//添加类属性

//此处不需要使用set方法，注解自动封装

@Autowired

private UserDao userDao;

public void add(){

System.out.println("service add...");

userDao.add();

}

}

（2）@Qualifier:根据属性名称注入

与@Autowired一起使用，适用于一个接口下有多个实现类的情况

@Component

public class UserService {

//添加类属性

//此处不需要使用set方法，注解自动封装

@Autowired

@Qualifier(value = "userDaoImpl") //value默认值为首字母小写的类名称

private UserDao userDao;

public void add(){

System.out.println("service add...");

userDao.add();

}

}

（3）@Resource：可以根据类型注入，也可根据名称注入

//@Resource

//注意：此包位于javax包下，不是spring自带的，建议使用以上两种注解

@Resource(name = "userDaoImpl")

private UserDao userDao;

public void add(){

System.out.println("service add...");

userDao.add();

（4）@Value：注入普通类型属性

@Value("abc")

private String name;

**6.完全注解开发**

（1）编写配置类，替代配置文件

package Spring5.Service;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

/\*\*

\* @author xh

\* @date 2022/10/11

\* @apiNote

\*/

@Configuration

@ComponentScan(basePackages={"Spring5"})

public class SpringConfig {

}

（2）编写测试类

@Test

public void userTest(){

AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(SpringConfig.class);

UserService userService = applicationContext.getBean("userService", UserService.class);

userService.add();

}