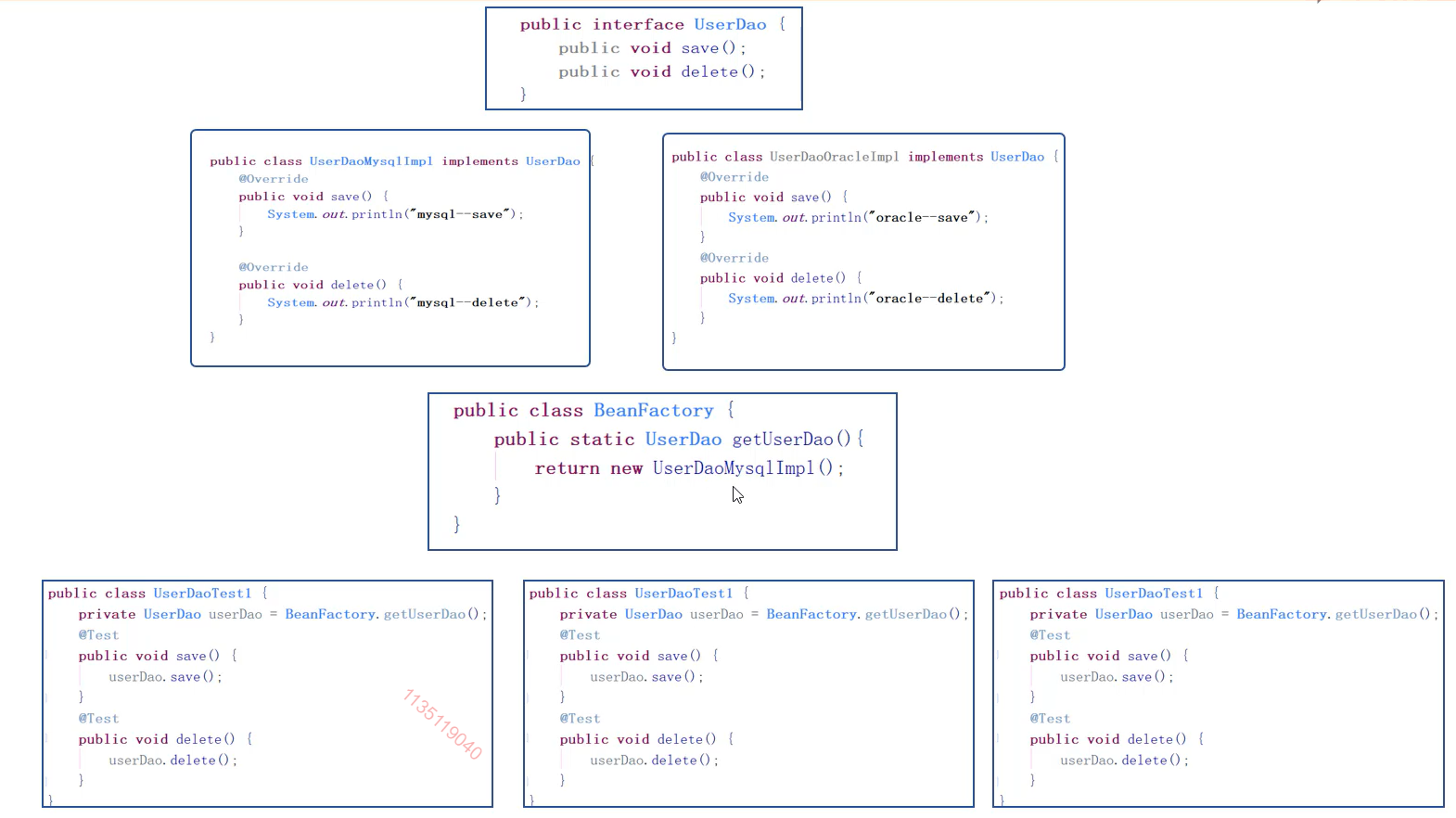
# 1 spring IOC

iOC简称控制反转，即将对象的创建权从用户反转给Spring。

## 1.1传统开发的弊端

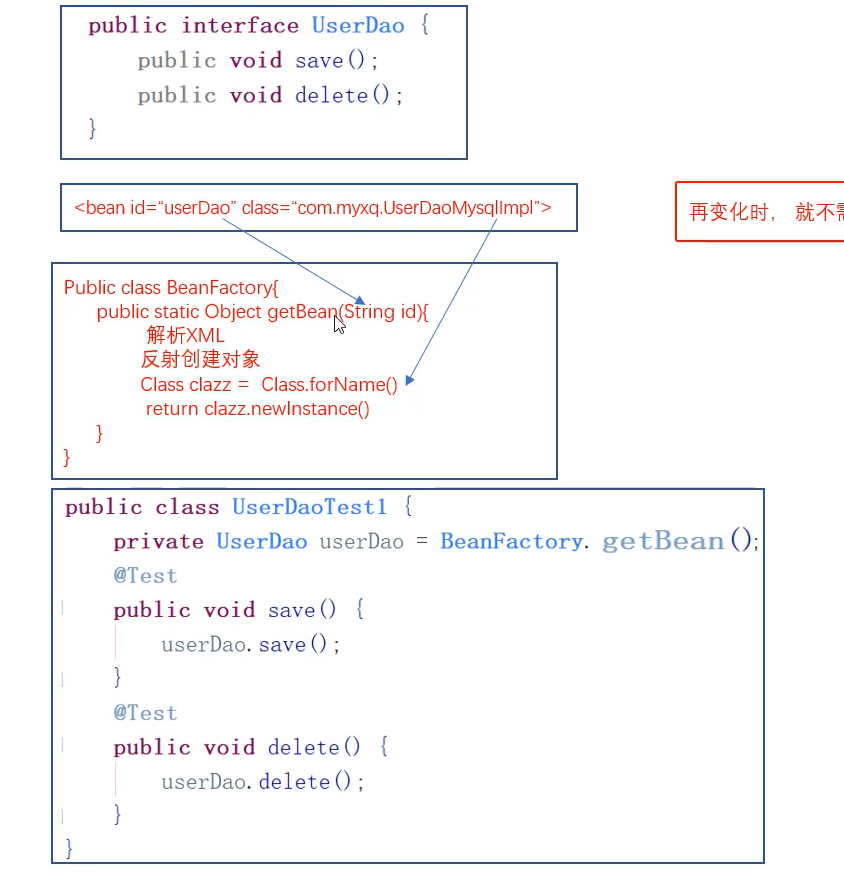
比如我们实现了两个同属于一个接口的类，整个项目在在业务层可能要new出来很多该类的对象，当我们从之前的类切换到另一个类的时候，我们便需要将文件中所有的类名更改为另一个类名，这样便不符合程序设计的原则。（业务与实现类过于紧密）。

上述问题的传统解决办法（找一个代理）：



每一个业务类，都是从中间代理的工厂类中获取相应的类，当我们需要切换类的时候，只需要修改工厂类中的文件即可。

而spring ioc不需要修改源码就可以切换类，只需对配置进行修改即可，spring解决方法：



## ­­­­1.2 工程样例（C:\项目学习\SpringPro1）

（1）添加jar包

既然使用spring框架，那么就应该将相应jar包导入的工程项目中去，这里使用的时蚂蚁小强教程中给的简化后的jar包。Jar存储位置位于：

C:\开发工具\spring相关Jar包\所需Jar包

在项目的web/WEB-INF/下新建lib目录，然后将jar包复制进该目录中，并全部右键单击，最后addLibary中。

(2) 创建配置文件，该配置文件放在工程项目中的src目录下：applicationContext.xml文件中应先导入下列语句：

<?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

http://www.springframework.org/schema/beans/spring-beans.xsd">

（3）添加配置配置项

<bean id="userDao" class="com.itlike.dom1.UserDaoOrcalImpl"></bean>

（4）通过ApplicationContext对象找到配置项中的类，从而调用该类中的方法：

@Test  
public void test(){  
 //1加载配置文件  
 ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");  
 //2根据id获取对象  
 UserDao userDao = (UserDao) applicationContext.getBean("userDao");  
 userDao.delete();  
}

## 1.3 DI(依赖注入)

DI（依赖注入）和IOC一起构成了spring的核心。DI的使用前提必须有IOC环境，spring管理这个类的时候才能将类的依赖的属性值注入（设置）进来；

依赖注入：给spring管理类当中依赖的属性，通过配置文件进行赋值的过程

类成员变量在配置文件中复制的书写格式：

（1）首先在源码中增加类成员变量和set方法

public class UserDaoMysqlImpl implements UserDao{

private String name;

public void setName(String name) {

this.name = name;

}

……}

（2）在配置文件中进行配置

<bean id="userDao" class="com.itlike.dom1.UserDaoMysqlImpl">

<property name="name" value="wangjiahui"/>

</bean>

（3）测试

//

@Test  
public void test2() {  
 //1加载配置文件  
 ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");  
 //2根据id获取对象  
 UserDaoMysqlImpl obj =(UserDaoMysqlImpl) applicationContext.getBean("userDao");  
 System.*out*.println(obj.name);  
}

## 1.4 spring工厂类

## 1.5通过ApplicationContex.getBean()获取的是单例类

//

@Test  
public void test2() {  
 //1加载配置文件  
 ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");  
 //2根据id获取对象  
 UserDaoMysqlImpl obj =(UserDaoMysqlImpl) applicationContext.getBean("userDao");  
 UserDaoMysqlImpl obj2 =(UserDaoMysqlImpl) applicationContext.getBean("userDao");  
 System.*out*.println(obj.name);  
 System.*out*.println(obj);  
 System.*out*.println(obj2);  
}

输出：

wangjiahui

com.itlike.dom1.UserDaoMysqlImpl@6fa51cd4

[com.itlike.dom1.UserDaoMysqlImpl@6fa51cd4](mailto:com.itlike.dom1.UserDaoMysqlImpl@6fa51cd4)

从输出可以看出，我们得到的两个对象是一个地址处的对象。

## 1.6 springBean对象生命周期

配置文件新增如下：

//

<bean id="userDao" class="com.itlike.dom1.UserDaoMysqlImpl" init-method="init">  
 <property name="name" value="wangjiahui"/>  
</bean>  
<bean id = "person" class="com.itlike.dom2.Person" init-method="init">  
 <property name="name" value="wangzeyu"/>

这时候，配置文件管理了两个类，当我们加载配置文件，即调用：

ApplicationContext obj = new ClassPathXmlApplicationContext("applicationContext.xml");的时候，配置文件配置的两个init-method方法都将运行。

这表明springBean在加载配置文件的时候，就创建了所有的管理类的相应对象，并输出所有配置的init方法。

## 1.7 关闭工厂

applicationContext.close();

close()关闭是将调用所有管理类的destroy方法。

//

<bean id="userDao" class="com.itlike.dom1.UserDaoMysqlImpl" init-method="init" destroy-method="destroy">  
 <property name="name" value="wangjiahui"/>  
</bean>  
<bean id = "person" class="com.itlike.dom2.Person" init-method="init" destroy-method="destroy">  
 <property name="name" value="wangzeyu"/>

## 1.8 bean的作用范围

Scope属性的默认选项便是Singleton（单例模式），此种模式下，当我们关闭工厂的时候，所有的对象就会销毁。而我们指定了目标管理类为scope多例模式，则不会销毁。

我们进行web开法的时候一般使用多例模式：prototype多例模式

## 1.9 分模块配置

我们在开发过程中大概率是多人共同开发，因此如果所有人都在一个配置文件中写配置则会出现紊乱的情况，这时候可以新建自己的配置文件，然后在主配置文件中引入自己的配置文件即可。如我们新建配置文件applicationContext2.xml:

则可以在主配置文件中写入<import resource=”applicationContext2.xml”>

这两个配置文件应该在同一目录内。

# 2 Spring属性注入

## 2.1 构造方式的属性注入

新建class文件：

//

public class User {  
 private String name;  
 private Integer age;  
 public User(String name, Integer age){  
 this.name = name;  
 this.age = age;  
 }  
 @Override  
 public String toString() {  
 return "User{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 '}';  
 }  
}

配置文件中使用构造方式的属性注入：

///

<bean id = "user" class="com.itlike.dom2.User">  
 <constructor-arg name="name" value="赵云"/>  
 <constructor-arg name ="age" value="26"/>  
</bean>

## 2.2 引用类型属性注入

//Dog类

public class Dog {  
 public String name;  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

//User类

public class User {  
 private String name;  
 private Integer age;  
 public Dog dog;  
  
 public void setDog(Dog dog) {  
 this.dog = dog;  
 }  
  
 public User(String name, Integer age){  
 this.name = name;  
 this.age = age;  
 }  
 @Override  
 public String toString() {  
 return "User{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 '}';  
 }  
}

// 先对dog类中的名字进行配置

<bean id="dog" class="com.itlike.dom2.Dog">  
 <property name="name" value="哮天犬"/> //这里用的set属性注入  
</bean>  
<bean id = "user" class="com.itlike.dom2.User">  
 <constructor-arg name="name" value="赵云"/>  
 <constructor-arg name ="age" value="26"/>

//然后引入dog类  
 <property name="dog" ref="dog"/>  
</bean>

## 2.3 名称空间属性注入

使用set属性注入的时候要写很多的property，我们可以采用简化的方式进行书写：

首先要在<beans /beans>中引入

xmlns:p=“<http://www.springframework.org/schema/p>”

然后就可以采用简写的方式：

//简写方式

<bean id="dog" class="com.itlike.dom2.Dog" p:name="哮天犬">  
<!-- <property name="name" value="哮天犬"/>-->

<bean id = "user" class="com.itlike.dom2.User" p:dog-ref="dog" >  
 <constructor-arg name="name" value="赵云"/>  
 <constructor-arg name ="age" value="26"/>  
<!-- <property name="dog" ref="dog"/>-->

## 2.4 集合类型属性注入

数组、链表、set、map四种注入方式：

//注入举例

<property name="attr">  
 <list>  
 <value>zs</value>  
 <value>ls</value>  
 <value>we</value>  
 </list>  
</property>  
<property name="myMap">  
 <map>  
 <entry key="name" value="张飞"></entry>  
 <entry key="name1" value="关羽"></entry>  
 </map>  
</property>

# 3 Spring注解开发