# Spring

## 一、Spring的概述

### 1.基本概念

Spring是全栈(full-stack)轻量级开源框架，以**IOC(Inverse Of Control)控制反转**和**AOP(Aspect Oriented Programming)面向切面编程**为内核。

Mybatis是持久层框架，SpringMVC则是MVC框架。

**Spring的优势**：

**①**方便解耦，简化开发。通过Spring提供的IOC容器，可以将对象间的依赖关系交由Spring进行控制，避免类与类之间的过渡耦合。

**②**AOP编程的支持。

**③**声明式事务的支持。

**④**方便程序的测试

**⑤**方便集成各种优秀的框架

**⑥**降低JavaEE API的使用难度。

### 2.程序的耦合和解耦

#### ⑴耦合和解耦的概念

程序中的**耦合**：简单说就是程序间的依赖关系。包括类之间的依赖、方法间的依赖。程序间的耦合会使程序之间的独立性变差。

**解耦**：就是降低程序间的依赖关系。

Bean：在计算机英语中，有可重复使用的组件的含义。

JavaBean：是用java语言编写的可重用的组件。

#### ⑵Properties类的常用的方法

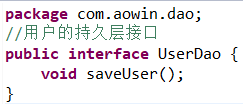
load(InputStream in)：从流中读取properties文件的内容。

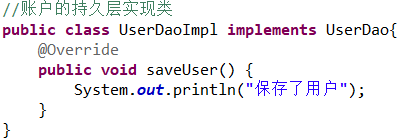
getProperty(String key)：根据key查找propeties文件中对应的value值，若没找到则返回null。返回值为String类型。

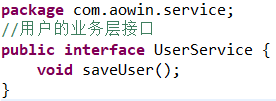
#### ⑶降低耦合的例子

当java中的对象都采用new关键字创建实例对象时，当删除某个类的java文件，程序在编译期就会报错，这说明程序在编译期就相互依赖，程序之间的耦合关系较强。

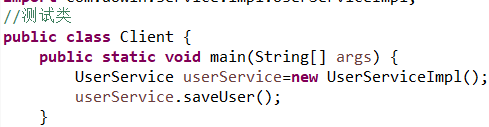
**类之间耦合的例子**











当删除上面的UserDaoImpl类或者UserServiceImpl类时，程序在编译期就会报错。

实际开发中应做到：编译期不依赖，运行时才依赖。

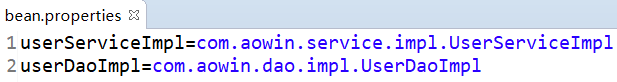
**解耦的思路**：

步骤一：使用反射来创建对象，而避免使用new关键字。采用Class.forName()创建对象，因此需要知道全限定类名。

步骤二：通过读取配置文件来获取要创建的对象全限定类名。

**解耦的例子**

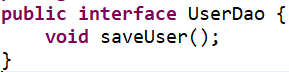
**①**创建一个配置文件(xml或者propeties类型)。这里采用properties类型做范例。Properties文件中的value值用来存放要使用Class.forName()创建对象的全限定类名。Key用来唯一标识这个value值。

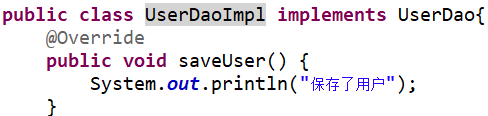


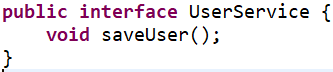
**②**采用工厂模式可以降低耦合。创建一个工厂类，用来实例化对象的。这个工厂类中需要读取properties文件，并解析properties文件中的内容，因此需要一个Properties的属性。

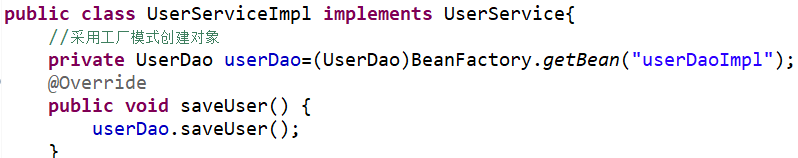


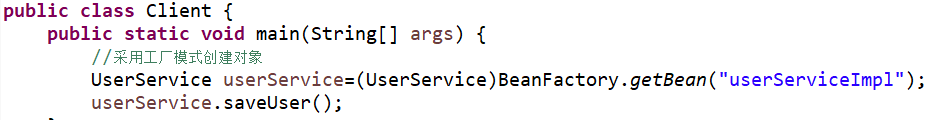
**③**将采用new创建的service和dao对象替换成采用工厂创建。







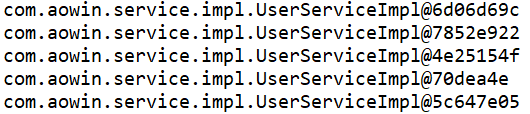




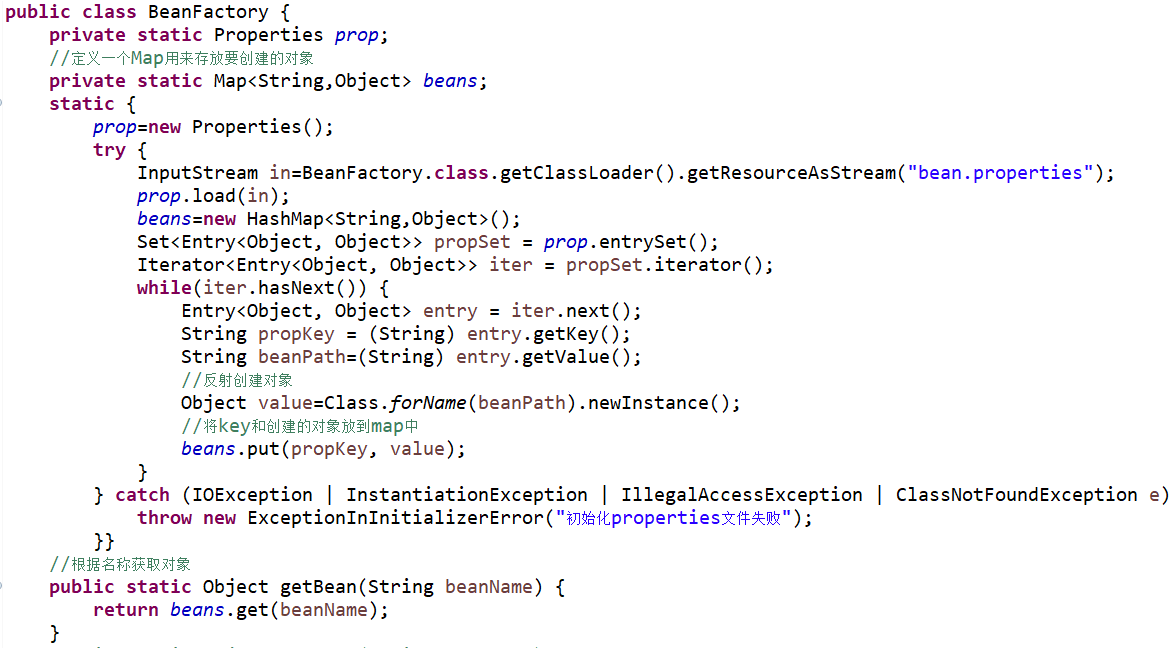
这样，即使直接将UserServiceImpl和UserDaoImpl的程序直接删除了，在编译的时候也不会报错，在运行时才会报错。

上述解耦的例子有个问题，就是每次采用上面的工厂模式创建的对象都是一个新对象。当对象被创建多次，则执行效率没有单例的高，同时也会浪费资源，但操作对象的属性时没有线程安全问题。当操作单例对象的属性时，则会有线程安全问题。但一般在service和dao中没有对属性进行修改的操作，因此可以不用考虑线程安全问题，因此为了执行效率，可以将上面的采用工厂模式创建对象的代码进一步成优化成单例的形式。

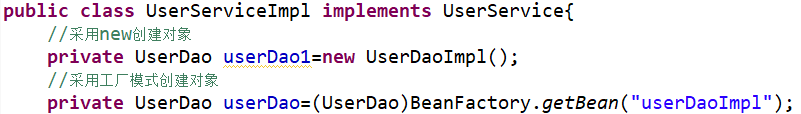




**将上面工厂模式的代码优化成如下的单例形式**：



## 二、IOC概念



上面UserServiceImpl类中创建userDao对象可以采用new的方式，也可采用反射的方式创建，UserServiceImpl中根据什么方式创建对象是由自身决定的。当它把创建对象的权力交给工厂来决定，这就是控制反转。

**控制反转(Inverse of Control)IOC**把创建对象的权力交给框架，是一种设计思想。它包括**依赖注入(Dependency Injection)DI**和**依赖查找(Dependency Lookup)**。即**DI是实现IOC的一种方式**。

在Spring中实现控制反转的是IOC容器，实现方法是DI。

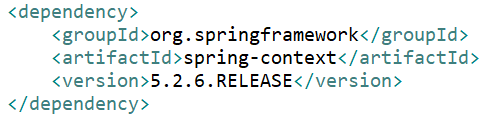
IOC的**作用**：削减程序间的耦合，即降低程序之间的依赖关系。Spring中的IOC就是降低程序间的依赖关系。

## 三、Spring入门和核心容器

Spring中核心容器需要使用的jar包：Spring-beans、Spring-core、Spring-context、Spring-SpEL。

### 1.使用Spring入门步骤：

**步骤一**：创建maven项目，并导入Spring的依赖如下：**任何Spring的运行都少不了这个依赖**。

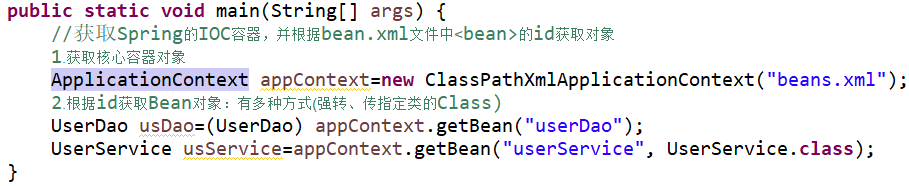


**步骤二**：配置bean的xml文件。该文件放在src/main/resources目录下，在该文件中添加spring-beans的约束，同时使用<bean>标签配置需要创建的对象。

<bean>标签的**id属性**用来唯一标识class中指定的内容。**class属性**的值为需要创建对象的类的全限定名称。



**步骤三**：获取Spring的IOC容器，并根据bean.xml文件中<bean>的id获取对象。



### 2.Spring的核心容器

**Spring容器是Spring框架的核心，是用来管理对象的。Spring容器管理对象的创建，并将它们关联起来，配置它们，并管理对象的生命周期**。从代码上说，一个Spring容器就是某个实现了ApplicationContext接口的类的实例。Spring提供了两种不同类型的容器：**ApplicationContext容器、BeanFactory容器**。Spring中的容器是Map结构的。

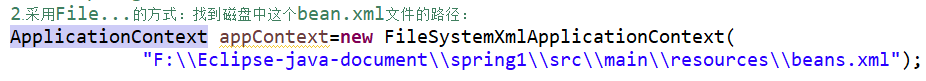
#### ⑴ApplicationContext接口的常用实现类

**ClassPathXmlApplicationContext**：它可以加载类路径下的配置文件，要求配置文件必须在类路径下。不在的话，加载不了。

**FileSystemXmlApplicationContext**：它可以加载磁盘任意路径下的配置文件(必须有访问权限的文件)。

**AnnotationConfigApplicationContext**：它是用于读取注解创建容器的。



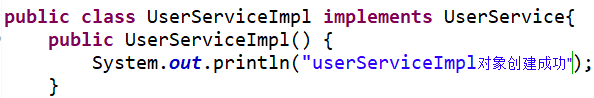
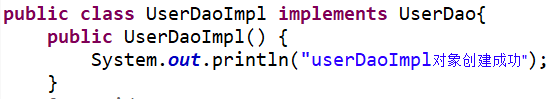


#### ⑵核心容器的两个接口

**ApplicationContext**：它在构建核心容器时，创建对象采取的策略是采用立即加载的方式。即一读取完配置文件马上就创建配置文件中配置的对象。**单例对象时使用**。常用此接口。

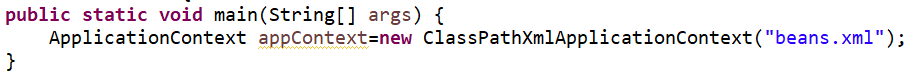
**BeanFactory**：它在构建核心容器时，创建对象采取的策略是采用延迟加载的方式。即需要创建配置文件中的对象时才创建对象，读取完配置文件不会立马创建配置文件中的对象。**多例对象时使用**。

**(ApplicationContext)例子**：





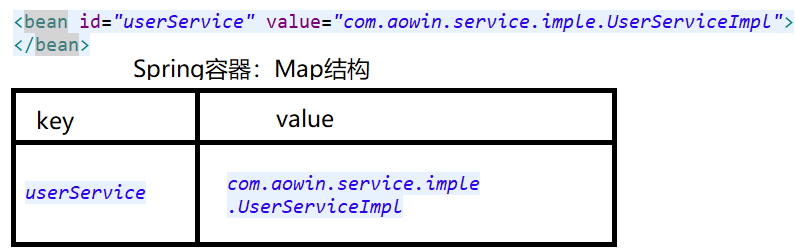
直接运行下面这段代码，则输出结果如下：



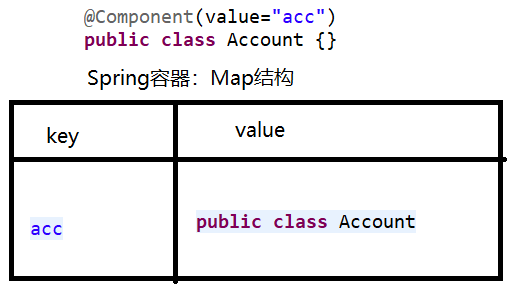


#### ⑶Spring容器是Map形式的

对于使用<bean>标签配置需要创建的对象，以Map的形式存在Spring容器中的形式如下：



对于使用注解配置需要创建的对象，以Map的形式存在Spring容器中的形式如下：若注解中没写value属性，则value的值默认为类名称，且首字母小写。



## 四、Spring中对bean的管理

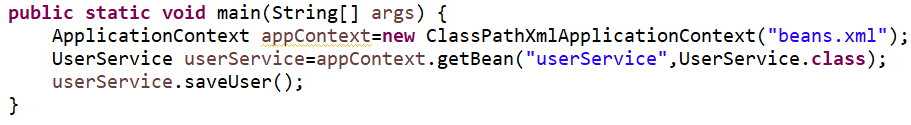
### 1.创建bean的三种方式

**①方式一**：使用默认构造函数创建

在Spring的配置文件中使用<bean>标签，配置id属性和class属性之后，且没有其他属性和标签时，采用的就是默认构造函数创建bean对象。此时若没有默认构造函数，则对象无法创建。若是有参的构造函数，则可以通过<constructor-arg>标签配置，然后进行创建。



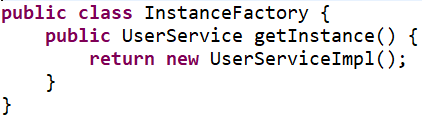
配置完之后，直接就可以采用getBean()方法创建UserService对象。

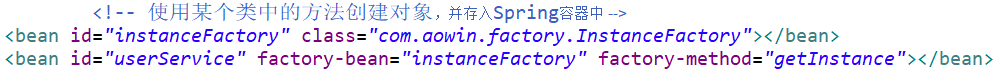


**通过默认的无参构造方式创建，其本质就是把类交给Spring自带的工厂（BeanFactory）管理，由Spring自带的工厂模式帮我们维护和创建这个类。**

**②方式二**：使用某个类中的普通方法创建对象，并存入Spring容器中(使用工厂中的普通方法创建对象)。需要配置factory-bean属性和factory-method属性。

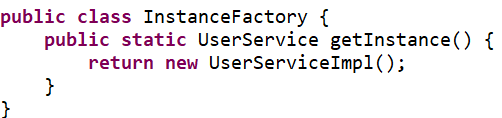
当需要使用InstanceFactory类中的getInstance()方法创建UserService对象时，在Spring的bean配置文件中进行如下配置：

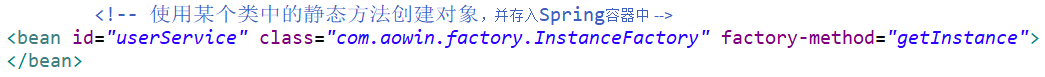


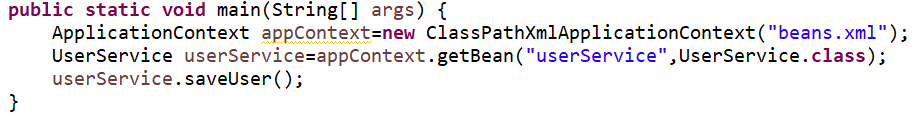


**通过工厂的普通方法创建，其本质就是把创建实例的工厂类交由Spring管理，同时把调用工厂类的方法创建实例的这个过程也交由Spring管理，而创建实例的这个过程也是由我们自己的实例工厂内部实现的。**

**③方式三**：使用某个类中的静态方法创建对象，并存入Spring容器(使用工厂中的静态方法创建对象)。需要配置factory-method属性。







注意这里class属性的值为这个类，factory-method属性的值为静态方法名，根据id的值创建的对象是静态方法中的对象。

**通过工厂中的静态方法创建，其本质就是把类交给我们自己的工厂管理，Spring只是帮我们调用了工厂类创建实例的方法，而创建实例的这个过程是由我们自己的工厂实现的**。在实际开发的过程中，很多时候我们需要使用到第三方jar包提供给我们的类，而这个类没有构造方法，而是通过第三方包提供的工厂创建的，这是时候，如果我们想把第三方jar里面的这个类交由spring来管理的话，就可以使用这种方式。

### 2.bean对象的作用范围

<bean>标签的scope属性用来指定bean的作用范围。默认bean的作用范围是**单例模式**。

**scope的取值**：

**①singleton**：单例的，默认值。表示在创建Spring的容器时立即创建该bean的对象，然后请求该bean时，只要方法中传入的id和容器中bean的id相匹配，则会返回同一个bean对象。

**②prototype**：多例的。在创建容器时并不会实例化bean对象，而是在请求该bean对象时(比如调用getBean()方法)才会创建bean对象，且每次创建的都是一个新对象。

每次调用getBean()方法时，都相当于new了一个xxxBean对象。

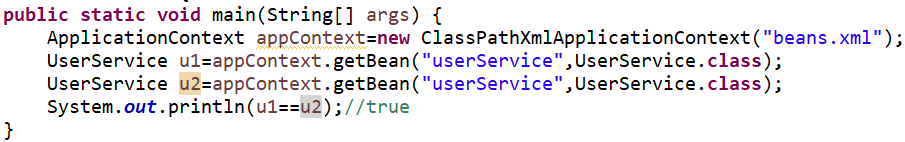
**③request**：作用于web应用的请求范围。每次发送一个Http请求都会创建一个bean对象。

**④session**：作用于web应用的会话范围。同一个session共享一个bean，不同session使用不同的bean。

**⑤global-session**：作用于集群环境的会话范围(全局会话范围)，当不是集群环境时，它就是session。

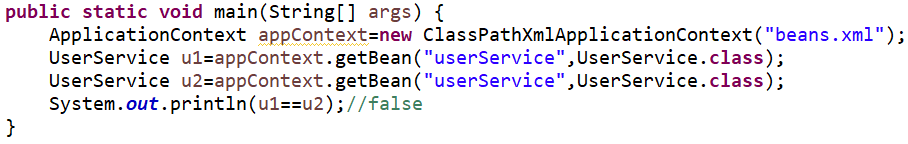
默认单例：





多例：



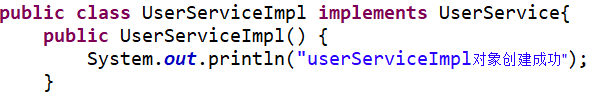


### 3.bean对象的生命周期

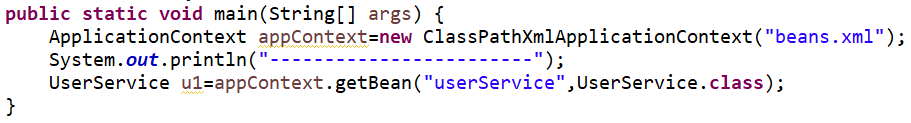
#### ⑴单例对象的生命周期

1. **创建**：默认情况下(scope=singleton)，在实例化容器时，单例对象就会创建，即立即加载对象。可以在<bean>标签中配置lazy-init属性值为true来延迟初始化bean，这时在第一次调用getBean()获取bean时才会初始化bean。

**默认实例化容器就创建bean对象**：



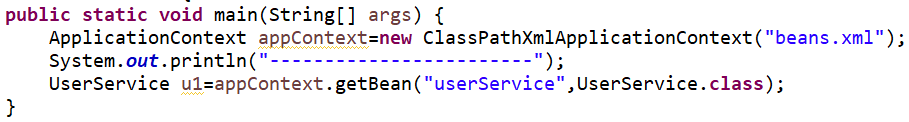






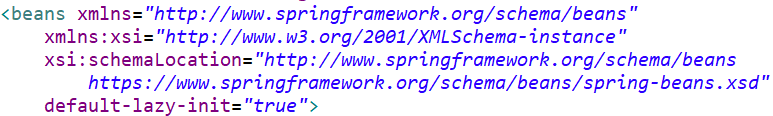
**配置lazy-init延迟初始化bean**：







若想对所有的默认单例bean都延迟初始化，可在根节点<beans>中设置default-lazy-init属性的值为true。

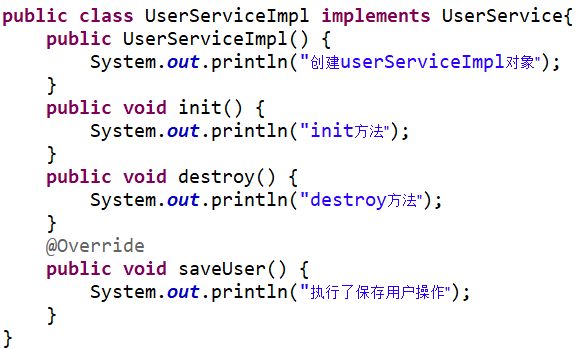


**②活着**：只要容器还在，bean对象就存在。

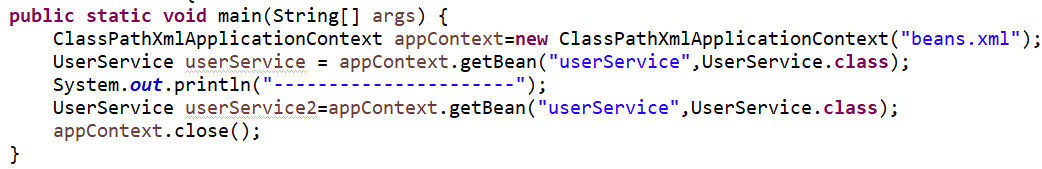
**③销毁**：容器销毁，bean对象就销毁。

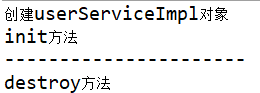
即**单例对象的生命周期和Spring中的IOC容器的生命周期相同**。

默认情况下，Spring在读取bean的xml文件实例化容器时，就会创建bean对象。在创建对象的时先调用构造器，然后调用init-method属性值中所指定的方法。对象在被销毁的时候，会调用destroy-method属性值中所指定的方法。**只要是同一个Spring容器，创建对象时调用的构造器、init-method指定的方法和destroy-method指定的方法都只执行一次**。









#### ⑵多例对象的生命周期

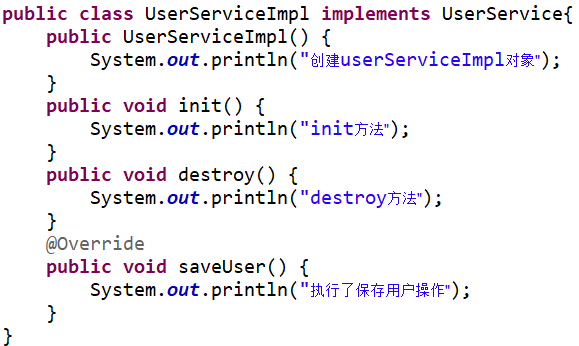
**①创建**：当scope=prototype时，Spring容器会延迟初始化bean对象，即初始化容器时并不会立即创建对象，而是在第一个请求该bean对象时(如调用getBean方法)才初始化。即延迟加载对象。

**②活着**：对象只要在使用过程中，就一直存在。

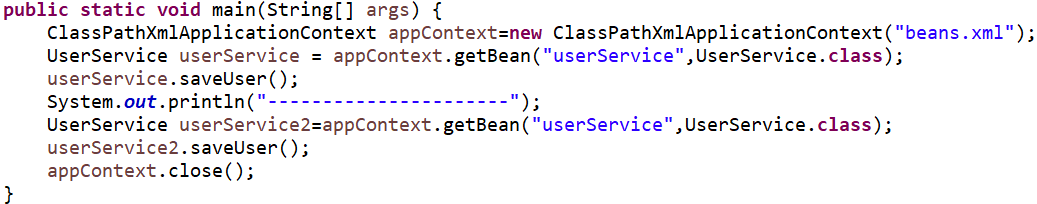
**③销毁**：当对象长时间不用，且没有别的对象引用时，由java的垃圾回收机制回收。

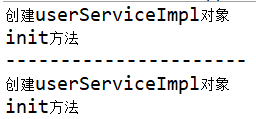
在scope=prototype时，每一次调用getBean()方法创建对象时，都会先调用构造器，然后调用init-method属性值中所指定的方法。但对象销毁时，不会调用destroy-method指定的方法，且Spring容器关闭也不会销毁该对象。该对象的销毁是由java的垃圾回收机制执行的。**构造器、init-method指定的方法每调用一个getBean()方法就执行一次**。

**例子：**









## 五、依赖注入(Dependency Injection)

### 1.依赖注入的数据类型

Spring中IOC的作用就是降低程序间的耦合(依赖关系)。对于依赖关系的管理，则由Spring来维护。在当前类中需要使用到其他类的对象时，由Spring为我们提供，我们只需在配置文件中说明。

依赖关系的维护就称为依赖注入。

**依赖注入中能注入的数据有三类**：

**①**基本数据类型及包装类和String。

**②**其他bean类型(在配置文件中或者注解配置过的bean)。

**③**复杂类型/集合类型。

### 2.依赖注入的方式

**(个人理解)依赖注入就是给属性赋值。**

**有三种**：

**①**使用构造函数注入：使用<constructor-arg>标签或c命名空间。

**②**使用set方法注入：使用<property>标签或p命名空间。

**③**使用注解注入：如@Autowired等。

#### ⑴构造函数注入：

**采用<constructor-arg>标签或者c命名空间**。

在指定创建类的对象使用哪个构造函数时，可在<bean>标签之中使用<constructor-arg>标签进行配置，**通过多个<constructor-arg>标签以及它的属性则可以指定是哪个构造函数**，并且给构造函数的参数赋值。**构造函数有几个参数就对应几个<constructor-arg>标签**。该标签的属性如下：

**type属性**：值为构造函数中某个或某些参数的数据类型。通过type属性指定的参数数据类型可以找到构造函数的参数列表中对应的参数，然后结合value属性给对应的参数赋值。缺陷是当构造函数的参数列表中的数据类型有相同的，则不知道type属性指的是哪个参数。**注意type属性的值要和构造器中参数的数据类型严格一致，如参数类型为int的type属性的值不能为java.lang.Integer**。

**index属性**：index从0开始，通过索引找到构造函数的参数列表中对应的参数，然后结合value属性给对应的参数赋值。

**name属性**：值为构造函数中的参数名称。通过参数名找到构造函数中对应的参数，然后结合value属性给对应的参数赋值。

type属性、index属性、name属性都是用来找到构造函数中对应的参数。其中name是常用的。

**value属性**：则是给参数进行赋值的，相当于实参。**value只能用于给基本数据类型、包装类、String类型的参数赋值**。

**ref属性**：则是用于给参数注入其他的bean类型的数据。**这个bean是在配置文件中或者使用注解的方式配置过的**。

c命名空间其实是构造函数注入的一种快捷方式，要想使用c命名空间必须注意以下几点：**①**在xml中添加c的xmlns:c="http://www.springframework.org/schema/c的约束； **②**实体类中必须存在有参构造器。

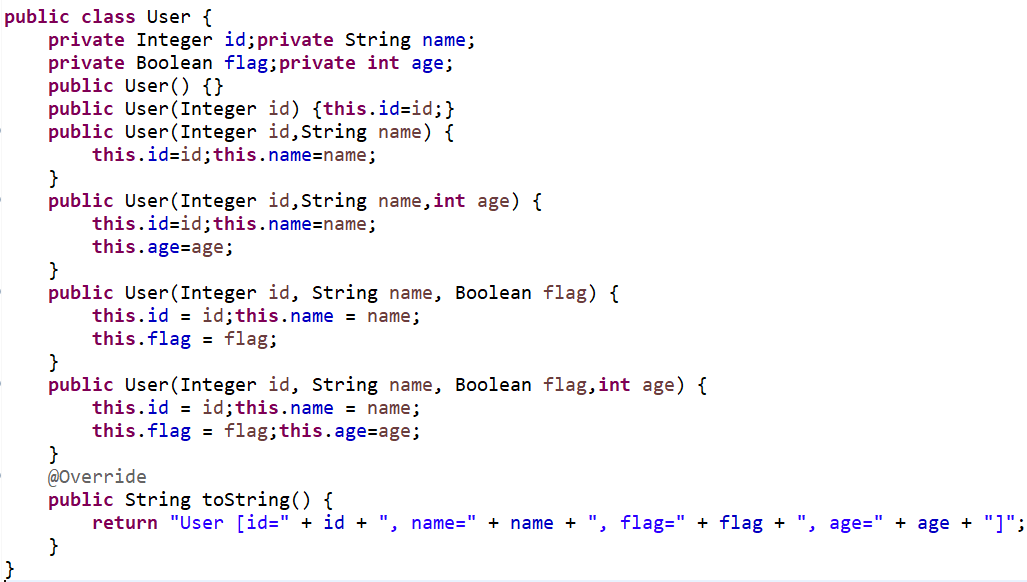


使用构造函数注入的**优势**：在获取bean对象时，注入数据是必须的操作，否则对象无法创建。

**弊端**：改变了bean对象的实例化方式，使创建对象时即使用不到要注入的数据，也必须提供这些数据。

**①注入的数据类型为基本数据类型、包装类和String类型的例子**：

**bean类：**

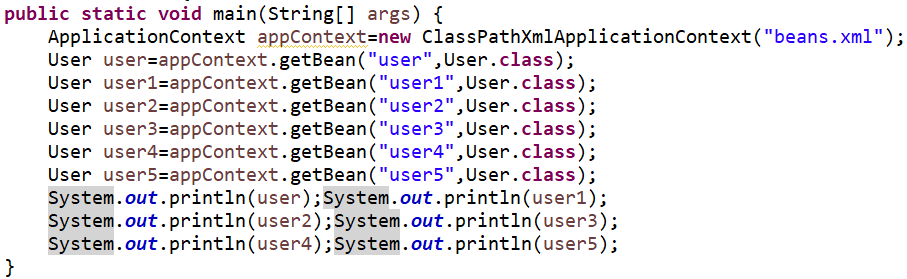


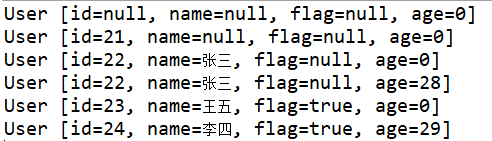
**bean.xml文件中的配置**：





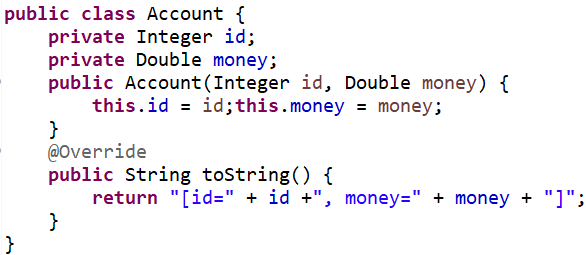
**测试：**

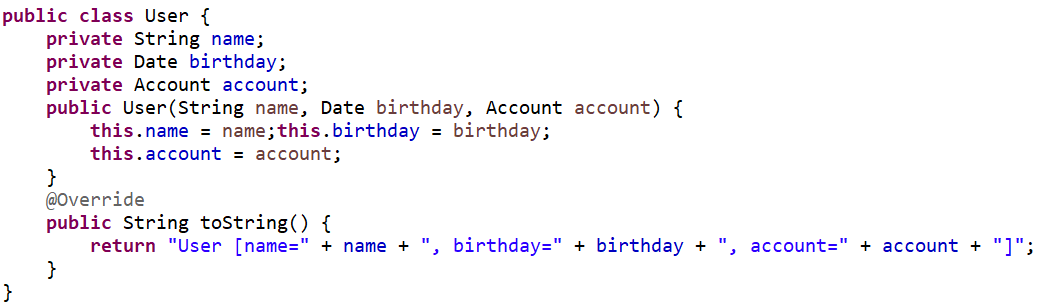




**②注入的数据类型是其他bean类型(在配置文件中或者注解配置过的bean)的例子：**

**bean类：**

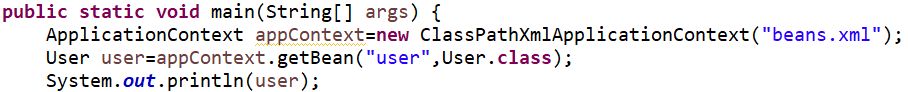




**bean.xml文件中的配置**



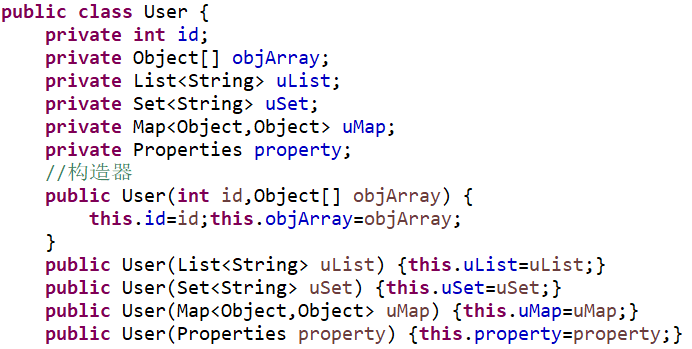
**测试：**





**③注入的数据类型是集合类型或者复杂类型的例子**：

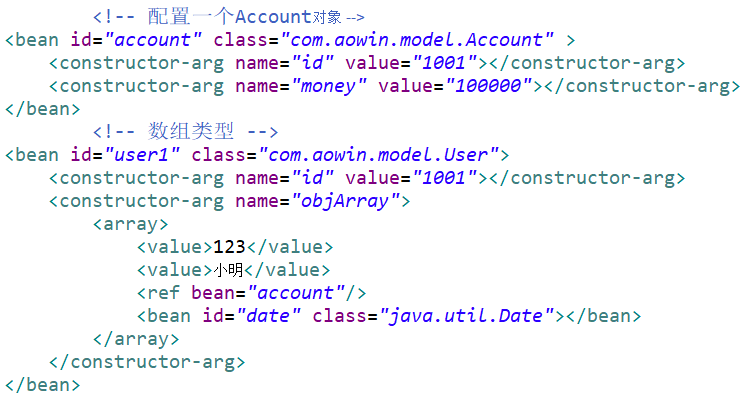
**实体类：**

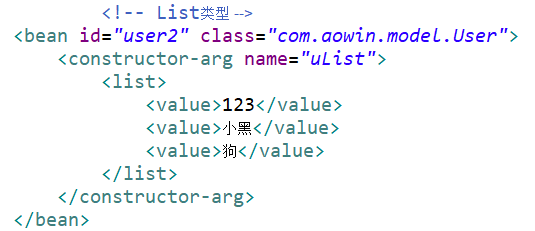


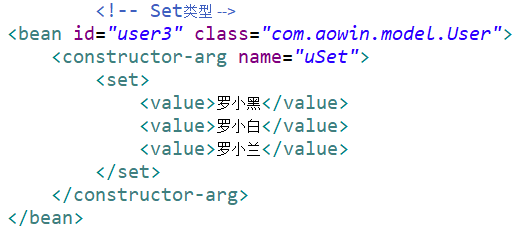
**bean.xml文件中的配置**：

对于注入的数据类型为**数组**、**List**、**Set**、**Map**、**Properties**，则在<constructor-arg>、<property>标签内分别使用**<array>**、**<list>**、**<set>**、**<map>**、**<props>**标签进行配置。

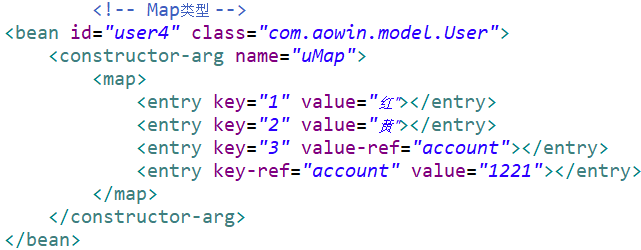
**若数组、List、Set中存放的是基本数据类型、包装类、String类型，则在<array>、<list>、<set>标签下使用<value>标签进行赋值；若存放的是引用数据类型(其他bean类型)，则可以使用<bean>标签或<ref>标签进行赋值**。





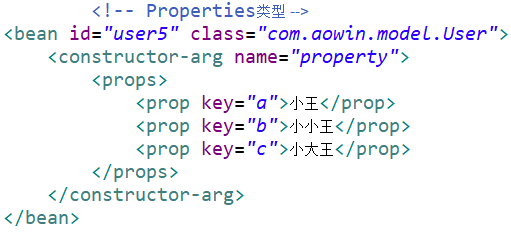


**对于注入的数据类型为Map，则在<constructor-arg>标签内<map>标签下使用<entry>标签进行赋值**。<entry>标签的属性有key属性、key-ref属性、value属性、value-ref属性。其中，**若Map中的键key或值value为基本数据类型、包装类、String类型，则使用<entry>标签的key属性、value属性进行赋值；若Map中的键key或值value为引用数据类型(其他bean类型)，则使用<entry>的key-ref属性、value-ref属性进行赋值**。**也可在<entry>标签下使用<key>、<value>、<ref>、<bean>、<list>等标签进行赋值**。





对于注入的数据类型为Properties，则**在<constructor-arg>标签内<props >标签下使用<prop >标签进行赋值。<prop>标签的key属性是给Properties的键赋值，Properties的值则在<prop></prop>标签之间进行赋值。**



#### ⑵set方法注入：

**采用<property>标签或者p命名空间**。

使用set方法注入时，需要在<bean>标签下使用<property>标签进行配置。若属性是boolean类型，没有set方法，则是is。<property>标签的属性有如下：

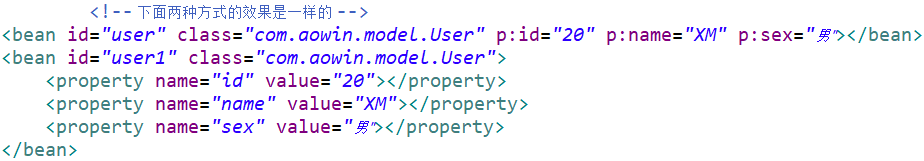
**name属性**：值为指定注入时所调用的set方法名称，如setAge，则name的值为age。name属性不关注类的属性的名称，只关注对应的set方法的名称是什么。

**value属性**：则是给参数进行赋值的，相当于实参。**value只能用于给基本数据类型、包装类、String类型的参数赋值**。

**ref属性**：则是用于给参数注入其他的bean类型的数据。**这个bean是在配置文件中或者使用注解的方式配置过的。**

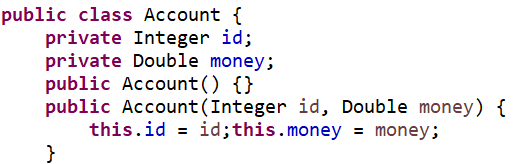
使用set方法注入的**优势**：**可以使用默认的无参构造函数创建对象**，然后再使用<property>标签给有set方法的属性赋值。若没有默认的无参构造函数，直接在<bean>下使用<propert>标签给属性赋值，运行时会报错。

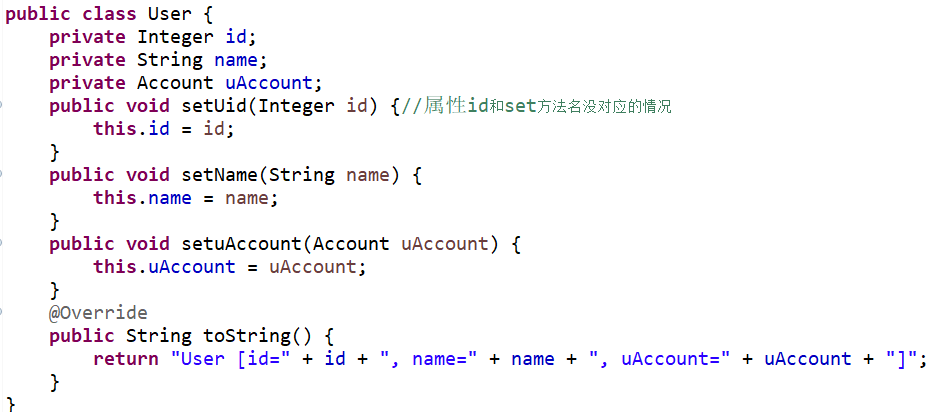
p命名空间是set注入的一种快捷实现方式，要想使用p命名空间的方式需要注意以下几点：**①**实体类中必须有set方法;**②**实体类中必须存在无参构造器；**③**xml中必须使用p命名空间的约束：xmlns:p="http://www.springframework.org/schema/p"；



**①采用set方法注入基本数据类型、包装类、String类型的数据以及其他bean类型的数据的例子**：

**实体类**：

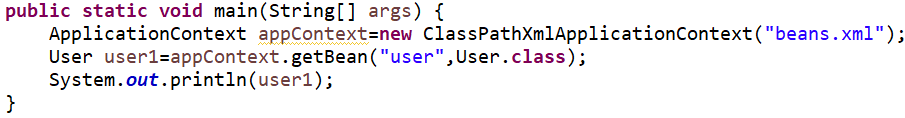




**bean.xml文件中的配置**：

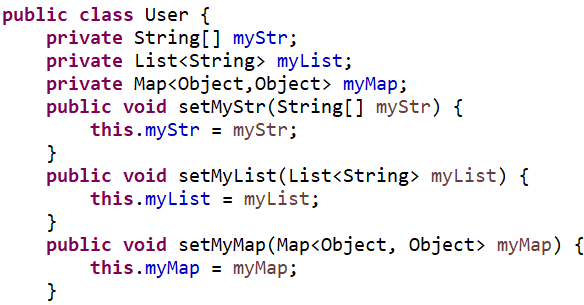


**测试**：

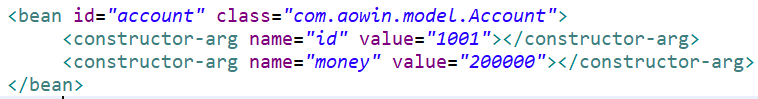




**②采用set方法注入集合数据类型或复杂数据类型的例子**：







用法和使用构造函数方式注入集合数据类型或复杂数据类型的相同。

#### ⑶使用注解注入

[注入数据的注解](#_⑵注入数据的注解)

## 六、基于注解的IOC及案例

### 1.Spring中IOC常用注解

#### ⑴创建对象的注解

用于创建对象的注解和在bean.xml文件中使用<bean>标签配置对象的作用一样。

**@Component注解**：当在类中使用了这个注解，则会使用反射的方式创建这个类的对象，然后存入Spring的容器中。它有**value属性**，这个属性和<bean>标签的id属性的作用是一样的。当不写value属性时，默认值是当前类的类名且首字母小写。

下面三个注解的作用效果和@Component注解的作用效果是一模一样的。下面三个只是Spring框架为我们提供的分别在表现层、业务层、持久层使用的注解，使这三个层的对象更加清晰。

**@Controller注解**：一般用在**表现层**。即对应以前的存放Servlet的Controller层。

**@Service注解**：一般用在**业务层**。即对应以前的service所在的层。

**@Repository注解**：一般用在**持久层**。即对应以前的dao接口所在的层。

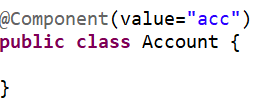
**特别注意**：当使用了上面的注解之后，需要在对应约束的xml文件中配置Spring在创建容器时要扫描的包，注意可以配置多个要扫描的包，每个要扫描的包之间用逗号隔开。

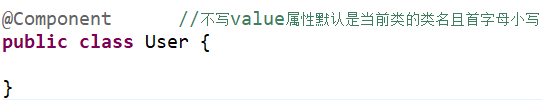


注意在xml中使用<bean>标签配置对象和在xml中使用注解配置扫描，这两者的xml约束是不同的，见笔记<4.Spring配置的范例文件>中的spring-beans.xml文件和spring-annotation.xml文件。

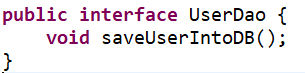
**例子**：

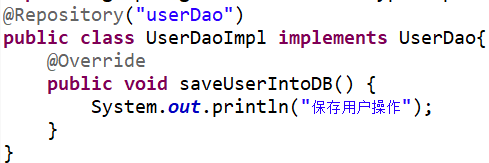
**model类：**



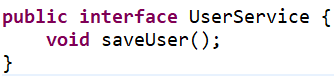


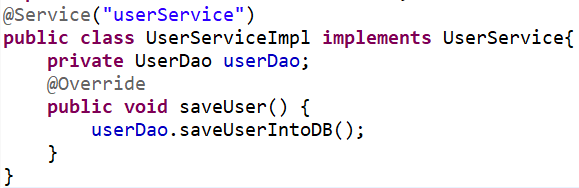
**dao包及其实现包下的类**：



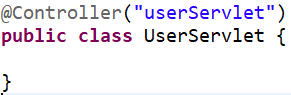


**service包及其实现包下的类**：

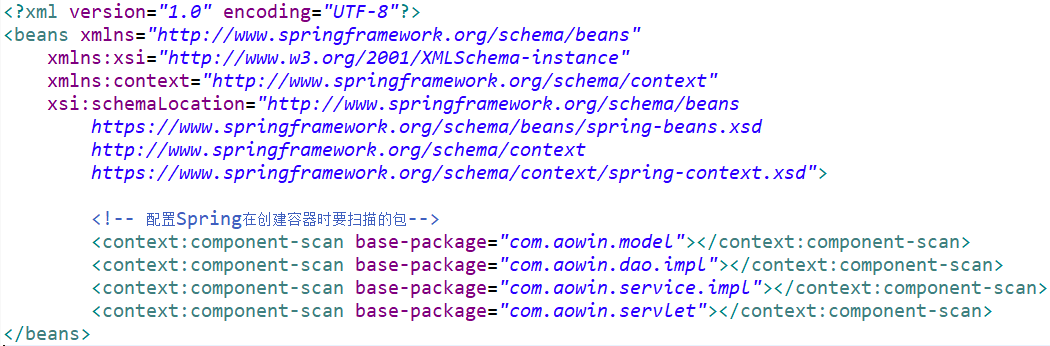




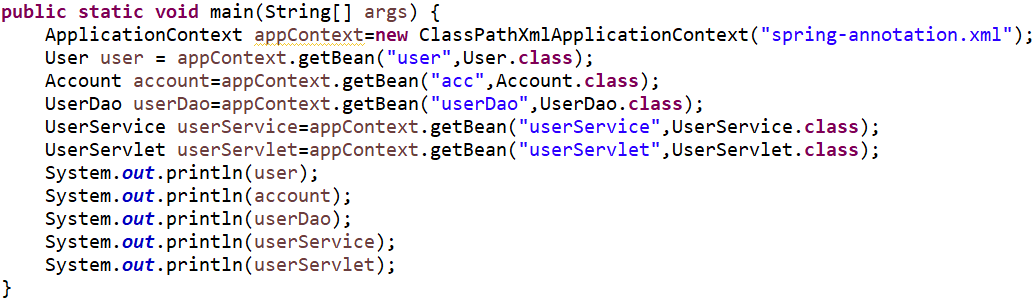
**servlet包或叫controller包下的类**：

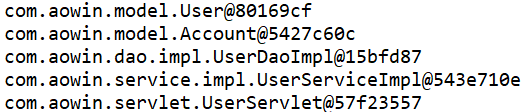


**对应的spring-annotation.xml文件中的配置**：



**测试**：





#### ⑵注入数据的注解

用于注入数据的注解和在<bean>标签下采用set方式(<property>)或者构造器方式(<constructor-arg)进行数据的注入效果一样。

**注意下面的@Autowired注解、@Qualifier注解、@Resource注解都只能注入bean类型的数据，对于基本数据类型、包装类、String类型则使用@Value注解**。对于**集合类型的注入只能通过xml文件实现注入。**

##### ①@Autowired注解

@Autowired注解：自动按照类注入数据。只要Spring容器中有**唯一的**一个bean对象类型和要注入的变量类型相匹配，就可以注入成功。若容器中没有任何bean的类型和要注入的变量类型相匹配，则报错。如下：要将作为属性的UserDao注入，首先直接查找容器中是否有UserDao对象，若容器中有唯一的一个UserDao对象已经存在，则下面作为属性的UserDao就能被注入成功，否则运行时会报错。

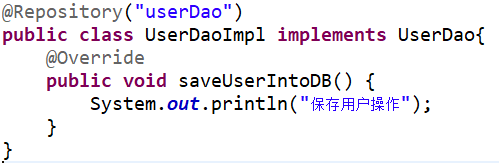


如果Spring容器中，有多个类型匹配时，则先按照类型找到匹配的对象，然后再使用变量名称作为id查找是否有跟这个id相同的内容，若有唯一一个，则可正常注入，若没有则报错。如：Spring容器中存放了两个UserDao的对象，它们分别对应一个userDao1和userDao2用来唯一标识UserDao对象。使用@Autowired标签注入UserDao时，先从Spring中查找到UserDao对象，发现有两个，则再根据userDao变量名查找userDao1和userDao2是否有相同的，发现没有则报错。若有则可以正常注入。

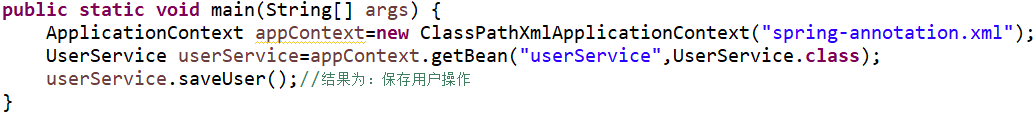
@Autowired注解可以出现在构造器上、方法上、参数上、变量上等。常用的是用在变量上和方法上。

**总结**：**单独使用@Autowired给bean类型的数据注入时，若容器中只有唯一一个这个bean类型的对象，则可以成功注入。若有多个则有可能注入不成功**。

**(容器中有唯一一个对应的bean类型的对象)例1**：



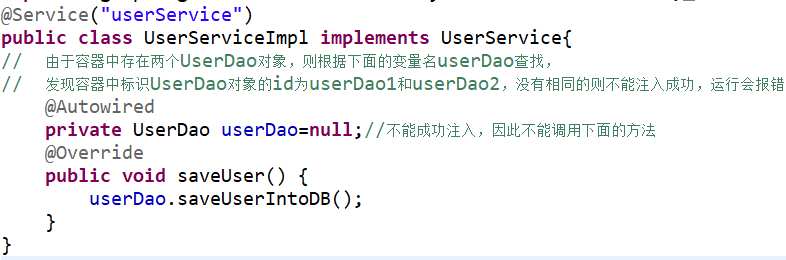


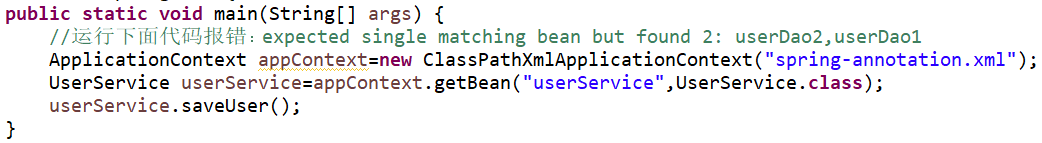


**(容器中有多个对应的bean类型的对象)例2**：





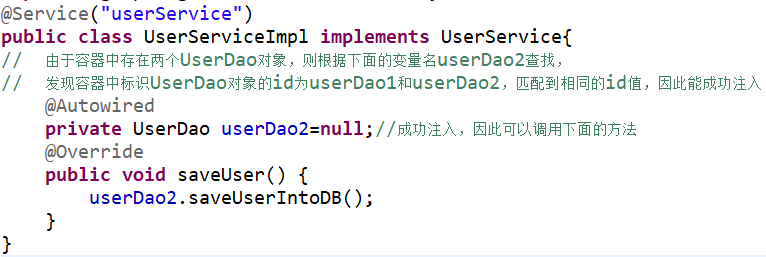




**(容器中有多个对应的bean类型的对象)例3**：









##### ②@Qualifier注解

**@Qualifier注解**：在按照类中注入的基础上再按照名称注入。**它在给类成员注入时不能单独使用，必须和@Autowired配合使用，但是给方法参数注入时可以单独使用**。其value属性：就是用于指定注入bean的id。

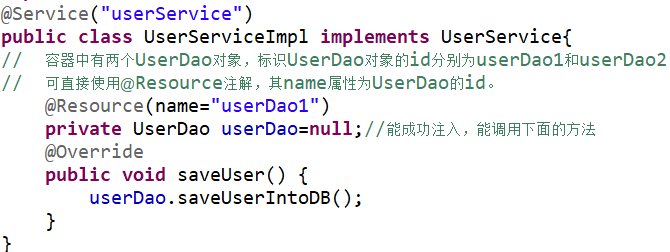
**示例**：



##### ③@Resource注解

**@Resource注解**：可以直接按照bean的id注入。可单独使用。其**name属性**用于指定bean的id。

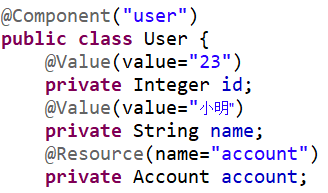
**示例**：

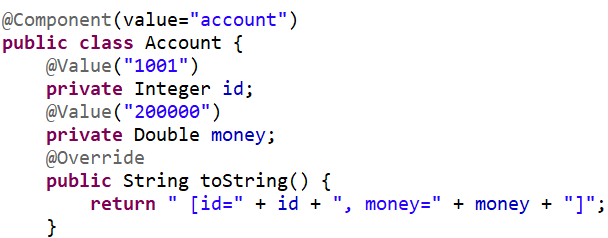


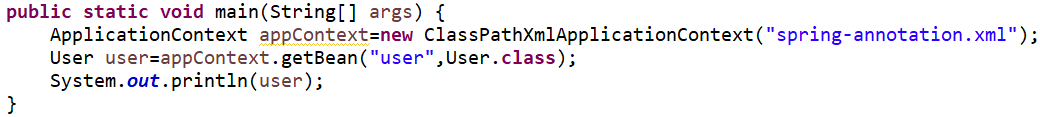
##### ④@Value注解

**@Value注解**：用来注入基本数据类型、包装类、String类型的数据。其value属性是用来指定数据的值，它可以使用Spring中的SpEL(即Spring中的EL表达式)。**SpEL的写法**：**${表达式}**，见后面。

示例：





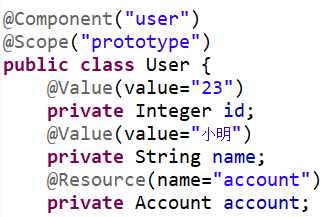


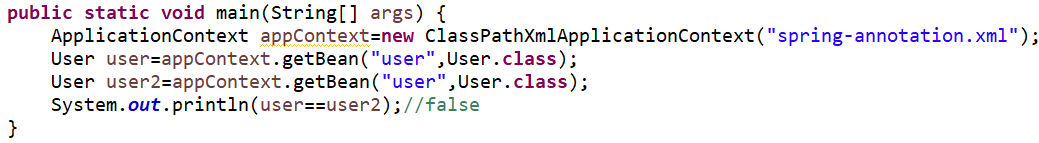


#### ⑶作用范围的注解

用于改变作用范围的注解同在<bean>标签中使用scope属性配置bean对象的作用范围的作用是一样的。

**@Scope注解**：用于指定bean的作用范围。其**value属性**是用来指定范围的取值，常用的取值为**singleton**和**prototype**。不使用@Scope注解默认使用的是singleton(单例)。





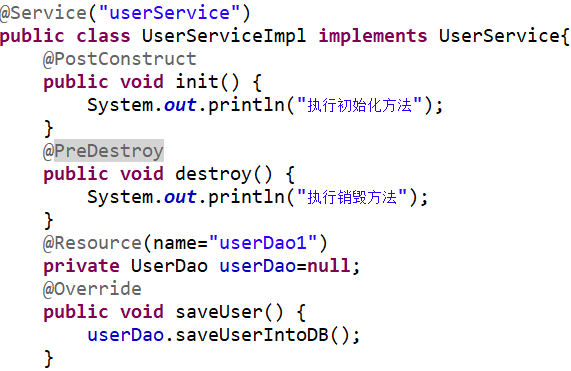
#### ⑷和生命周期有关的注解

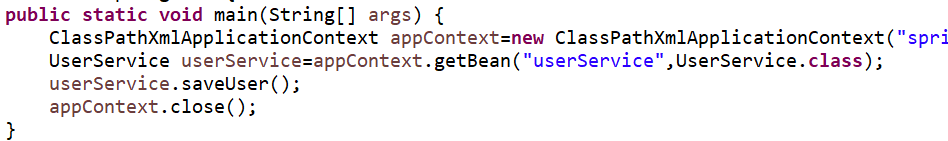
这个内容是属于了解。

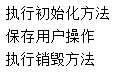
与生命周期有关的注解和在<bean>标签中使用init-method和destroy-method属性的作用是一样的。

**@PreDestroy注解**：用于指定销毁方法。同destroy-method。

**@PostConstruct注解**：用于指定初始化方法。同init-method。







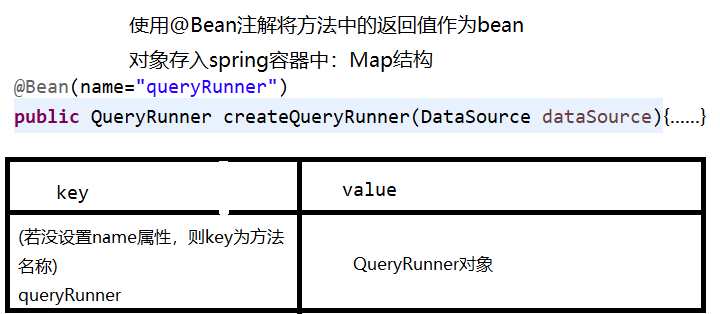
#### ⑸新注解

**@Configuration**注解：用来指定当前类是一个配置类。当配置类作为创建AnnotationConfigApplicationContext对象的参数时，该注解可以省略，或者配置类作为@Import注解的value属性的值时，该注解可以省略，否则的话该配置类不起作用，可能会报错。

**@ComponentScan**注解：该注解用于指定spring在创建容器时要扫描的包(使用@Repository、@Service、@Controller注解的包)，而不是这个@ComponentScan注解所在的包。它有**value属性**和**basePackages属性**，这两个属性的作用是一样的，都是用于指定创建容器时要扫描的包。这个注解的作用和xml文件中配置的<context:component-scan base-package=*"包 "*>的作用一样。value属性和basePackages属性的值可以具体到类。

**@Bean**注解：用于把当前方法的返回值作为bean对象存入spring的IoC容器中。其有**name属性**用于指定bean的id。**当不写name属性时，默认是当前方法的名称**。

**注意**：当使用@Bean注解配置方法时，若方法的参数是bean对象，spring框架会去容器中查找是否有可用的bean对象，查找的方式和@Autowired注解的作用是一样的，即若容器中有唯一的bean对象相匹配则可以成功注入，若有多个，则匹配容器中bean的id值(即下面Map中的key)，若匹配成功则可成功注入，否则注入失败。



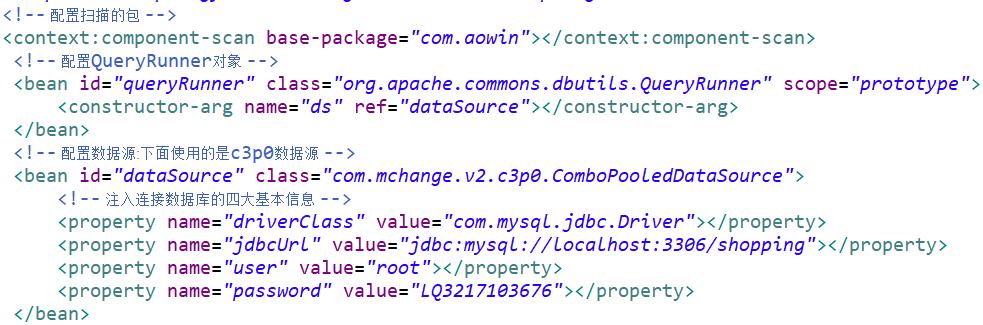
**@Import**注解：用于导入其他的配置类。其value属性用于指定其他的配置类，值为类.class。有@Import注解的类就是父(主)配置类，而导入的都是子配置类。

**@PropertySource**注解：用于指定properties文件的位置。其value属性用于指定文件的名称和路径。关键字classPath表示是在类路径下，若properties文件是直接在src目录下，则classPath:xxx.properties。若在properties文件在包下，则classPath:xxx/xxx/xxx.properties。注意这个路径可以写成绝对路径或者相对路径。

**@Qualifier**注解使用在参数上：见下面的例子。

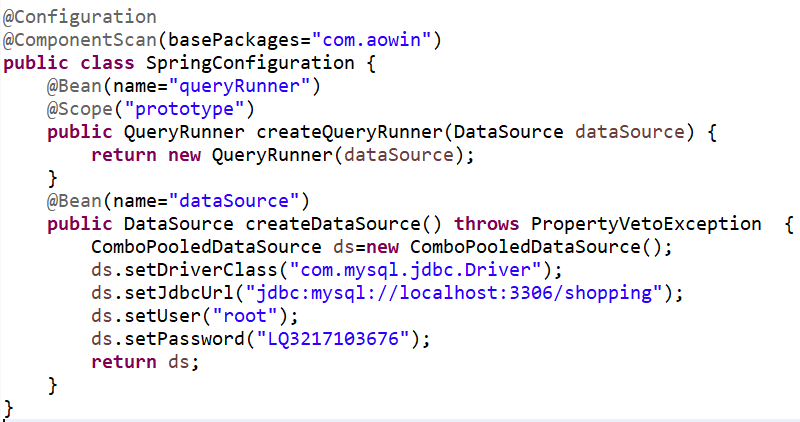
**①下面的SpringConfiguration类的作用效果和下面的xml文件的效果一样，可以等价**。





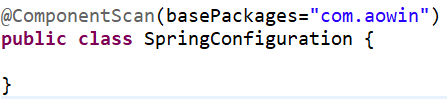
**②下面A、B、C的作用效果一样**：

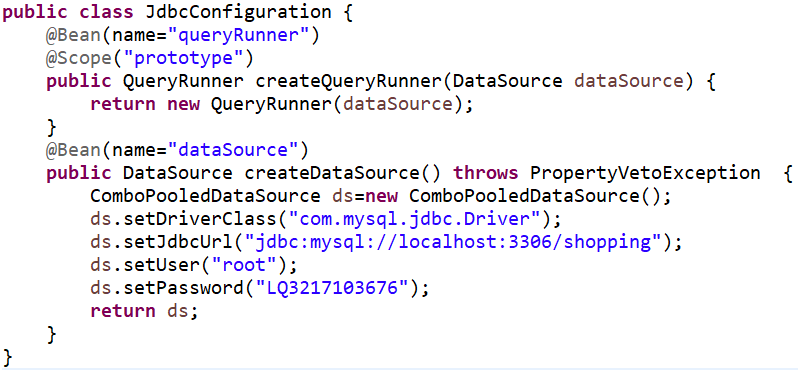
**A:**





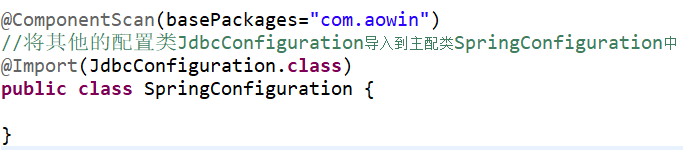
**B:**

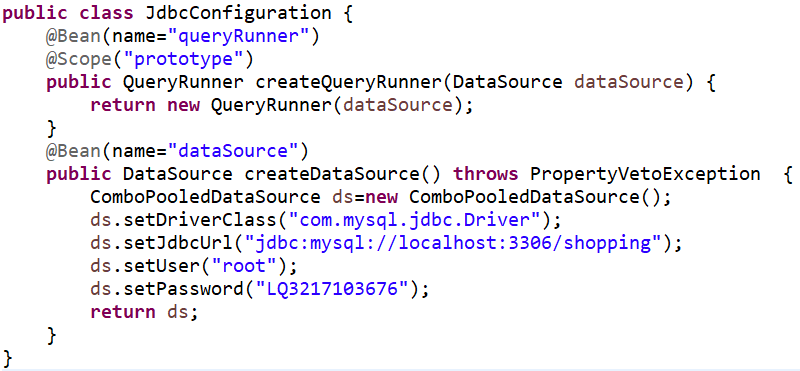






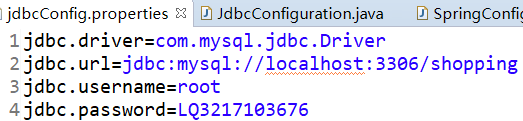
**C：**

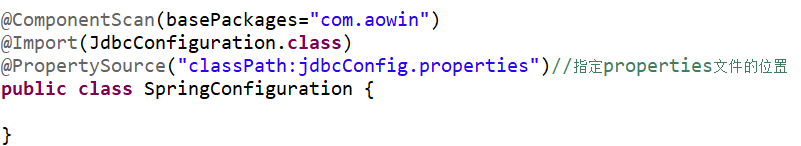


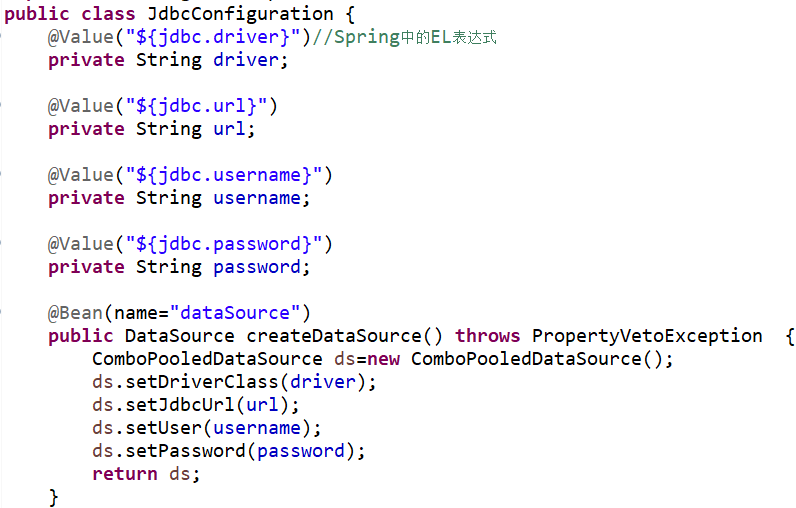




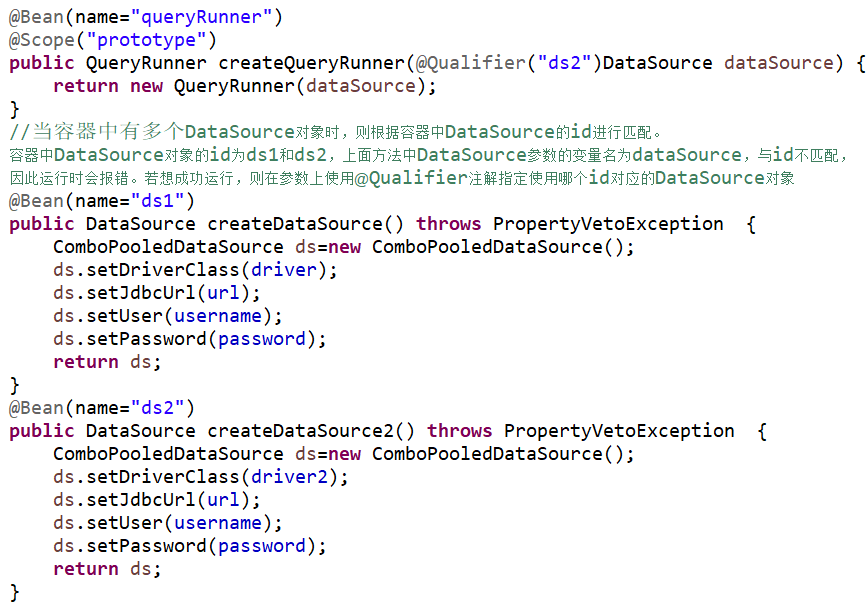
**使用@PropertySource标签的例子**：







**使用@Bean注解时，注入多个同类型的bean对象时，具体使用哪个bean对象可以使用@Qualifier注解指定**，例子如下：



### 2.使用xml和注解方式实现单表CRUD操作

#### ⑴额外内容：

DBUtils(JDBC Utility Component)是一种简化JDBC操作的工具。DBUtils的核心类和核心接口分别为**QueryRunner**类、**ResultSetHandler**接口。DBUtils中的创建QueryRunner对象可以使用带参数的，也可以使用不带参数的。区别是：是每条语句独立一个事务，还是所有的SQL语句都在同一个事务中。

**①创建QueryRunner对象常用的方法**如下：

QueryRunner queryRunner=**new** QueryRunner();

QueryRunner queryRunner=**new** QueryRunner(DataSource ds)：参数为数据源。

**②QueryRunner常用的方法**如下：

**执行select操作的方法**：

<T> T query(String sql, ResultSetHandler<T> rsh, Object... params)：参数sql为sql语句，参数rsh为指定处理结果的类型， params参数为实参，用来替换sql语句中用？占位的内容。

<T> T query(String sql, ResultSetHandler<T> rsh)：直接给定sql和处理结果的类型就能直接执行sql语句。

<T> T query(Connection con,String sql, ResultSetHandler<T> rsh, Object... params)：这是指定Connection对象来执行相应的操作。

**执行insert、update、delete操作的方法**：

int update(String sql)：直接给定sql语句就能执行insert、update、delete的操作。

**int** update(String sql, Object... params)：可变参数params用来替换sql语句中用?占位的内容。

**③ResultSetHandler接口常用的实现类**：

**ArrayHandler**：把结果集中的第一行数据转成对象数组。

**ArrayListHandler**：把结果集中的每一行数据都转成一个对象数组，再存放到List中。

**BeanHandler**：将结果集中的第一行数据封装到一个对应的JavaBean实例中。

**BeanListHandler**：将结果集中的每一行数据都封装到一个对应的JavaBean实例中，存放到List里。

**MapHandler**：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值。

**MapListHandler**：将结果集中的每一行数据都封装到一个Map里，然后再存放到List。

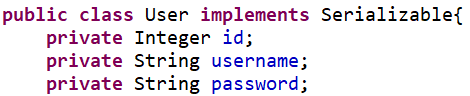
**ColumnListHandler**：将结果集中某一列的数据存放到List中。

#### ⑵例子1：

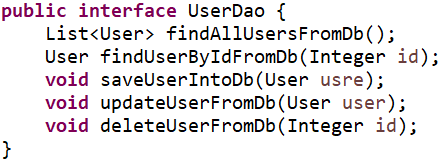
**pom.xml文件的依赖：**



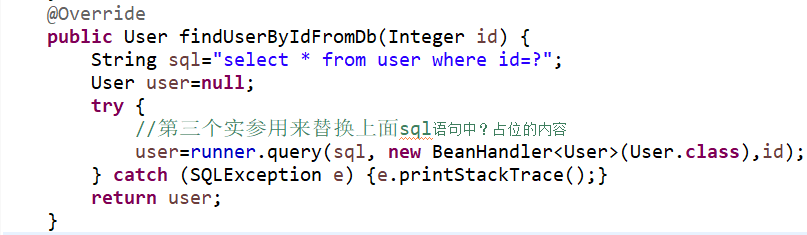
**model类**：

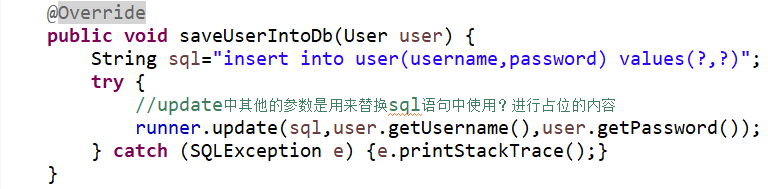


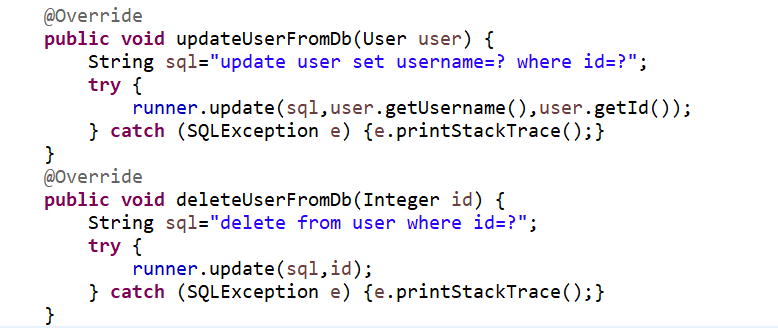
**dao接口和实现类**：



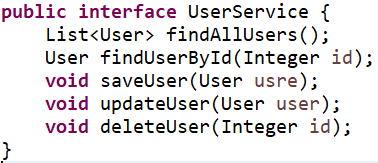


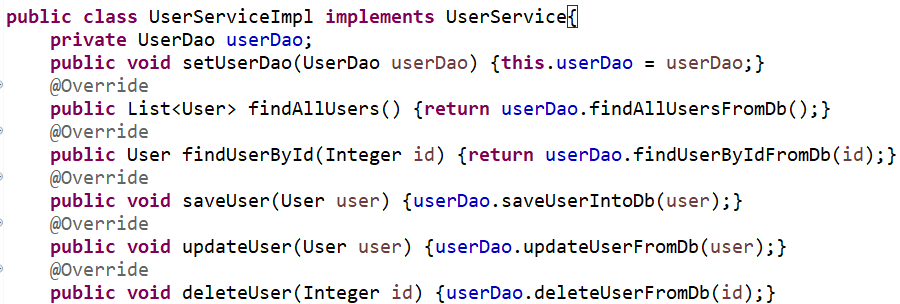






**service接口和实现类**：

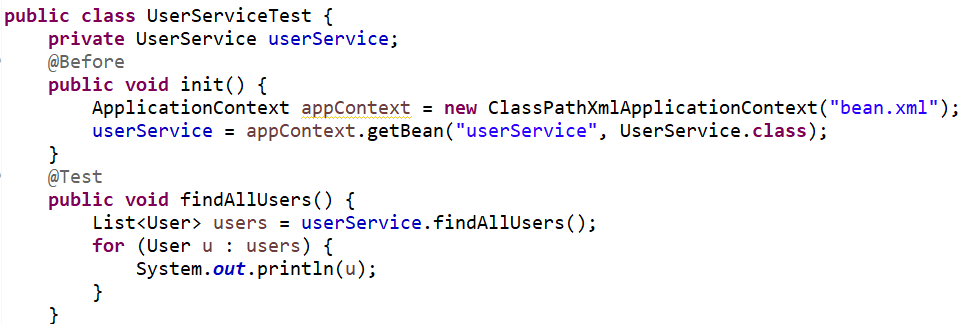




**bean的配置文件(使用spring-beans的约束)**：

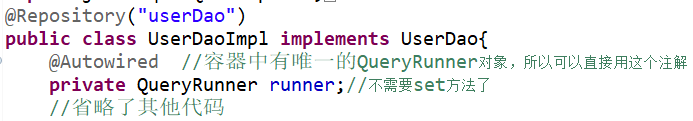


**测试方法**：

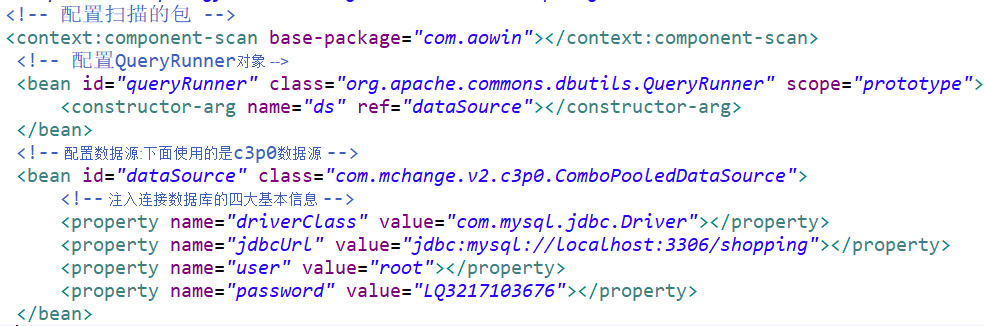


#### ⑶例子1的优化：

使用注解的方式配置spring容器要创建的对象。然后将bean.xml文件中使用<bean>标签配置的Service对象和Dao对象删除，同时将xml文件的约束修改为spring-context.xml的约束，并配置扫描包。同时使用注解的方式注入数据。







#### ⑷上面例子的继续优化：

dao接口及其实现类和service接口及其实现类同上面的一样。上面的xml文件优化称下面的配置类：





### 3.Spring和Junit整合

应用程序的入口是main方法。

在Junit单元测试中，即使没有main方法也能执行，这是因为Junit集成了一个main方法，该方法会判断当前测试类中有哪些方法有@Test注解，Junit就让有Test注解的方法执行。

在执行测试方法时，junit根本不知道我们是否使用了spring框架，因此不会自动帮我们读取配置文件或者配置类来创建Spring容器。因此，即使使用了@Autowired注解，也不能将bean对象注入到容器中。



**Spring整合Junit的使用步骤**：

**步骤一**：导入Spring整合Junit的jar包：spring-test的依赖。

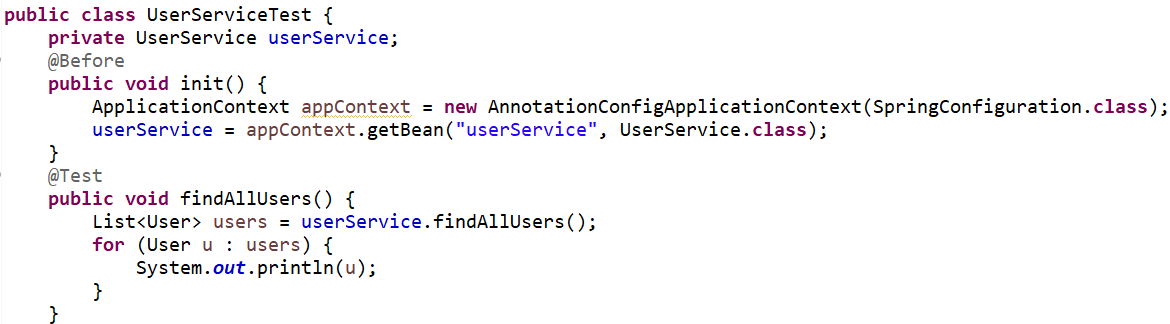
**步骤二**：使用Junit提供的一个注解把原来的main方法替换成Spring提供的：@RunWith()指定使用spring测试的运行器。如：@RunWith(SpringJUnit4ClassRunner.**class**)。

@RunWith()注解是用来类上面的，详细看Junit笔记：[Junit测试.docx](../3.javaSE基础/14Junit测试.docx)

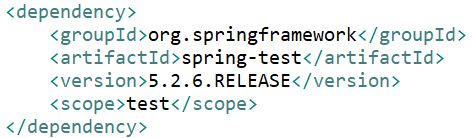
**步骤三**：使用注解告知Spring的运行器，Spring容器创建对象是基于xml文件还是注解，并指定位置：@ContextConfiguration()。该注解的**locations属性**：用来指定xml文件的位置，加上classPath属性表示类路径下；**classes属性**：指定注解类所在的位置。

**注意**：当使用Spring 5.x版本时，要求Junit的版本是4.12及以上，否则会报错。

**不将Spring和Junit整合的例子**：



**整合Spring和Junit的例子**：





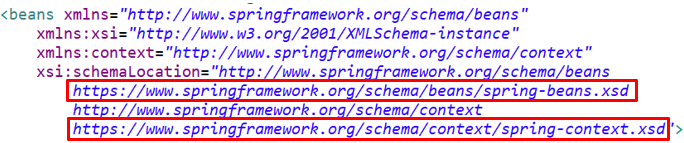
## 七、Eclipse中配置Spring的配置文件有提示功能

### 1.配置注解配置文件有提示

**步骤一**：找到注解配置文件中的xml的约束，并分别复制这几个约束：

<https://www.springframework.org/schema/beans/spring-beans.xsd>

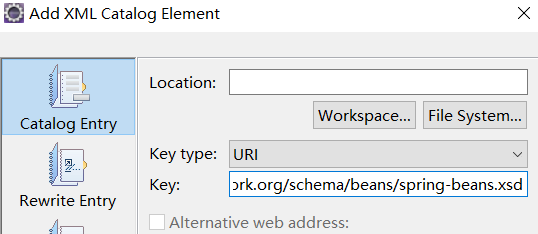
<https://www.springframework.org/schema/context/spring-context.xsd>





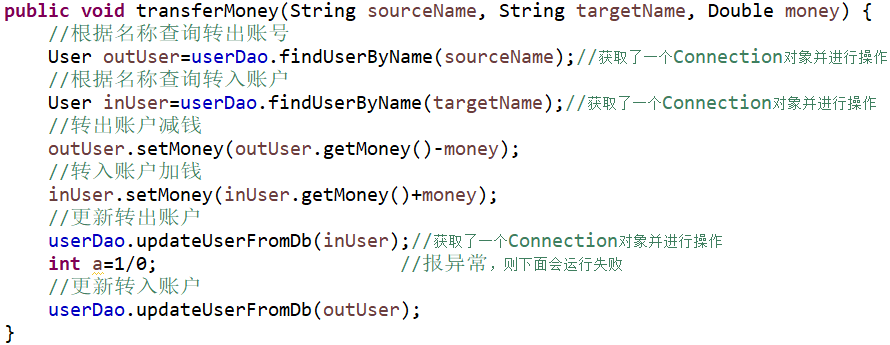
**步骤二**：点击Eclipse中的windows🡪preferences🡪XML🡪XML Catalog🡪点击Add🡪进入如下页面：Key type选择URI类型，同时将刚才复制的其中一个约束粘贴到Key中。然后再点击File System找到对应的spring-beans.xsd文件，则个文件放在E盘中的springFiles文件夹中，完成之后点击Apply即可。然后再粘贴第二个约束，同时找到spring-context.xsd文件，完成之后点击Apply即可。在粘贴第三个约束，找到spring-aop.xsd文件，完成之后点击Apply即可。

这几个文件的位置：[..\..\springFiles](../../springFiles)



## 八、spring和动态代理结合且手动事务控制

下面代码每向数据库进行操作一次都是获取的新Connection对象，每个Connection对象都能控制事务。当一个Connection对象操作事务成功之后，后面其他的Connection对象操作其他事务遇到异常，则可能操作失败，因此数据库中的数据就可能发生错误。因此需要让下面对数据库进行的各个操作都使用的是同一个Connection对象，这样当一个操作发生异常，则之前的操作也会回滚到之前的状态。



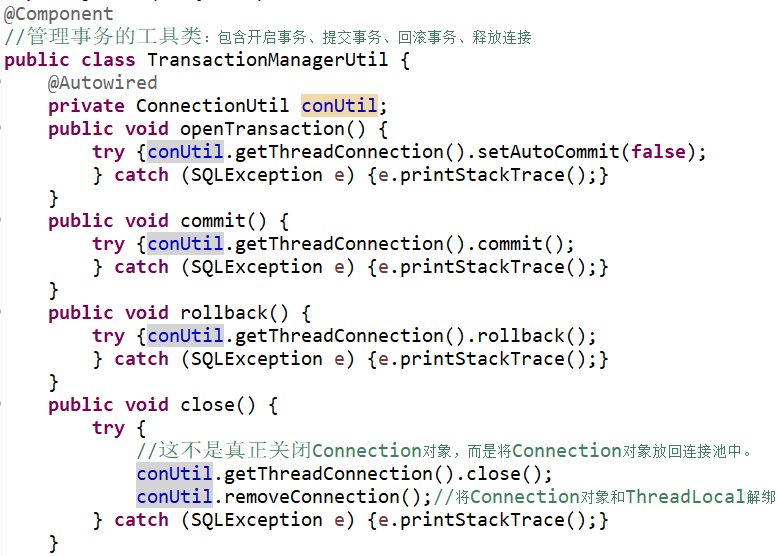
**解决办法**：**使用ThreadLocal对象把Connection对象和当前线程绑定，从而使一个线程中只有一个能控制事务的对象**。

编写两个工具类：一个连接工具类，用来实现将Connection对象和线程的绑定；一个管理事务的工具类，用来管理事务。

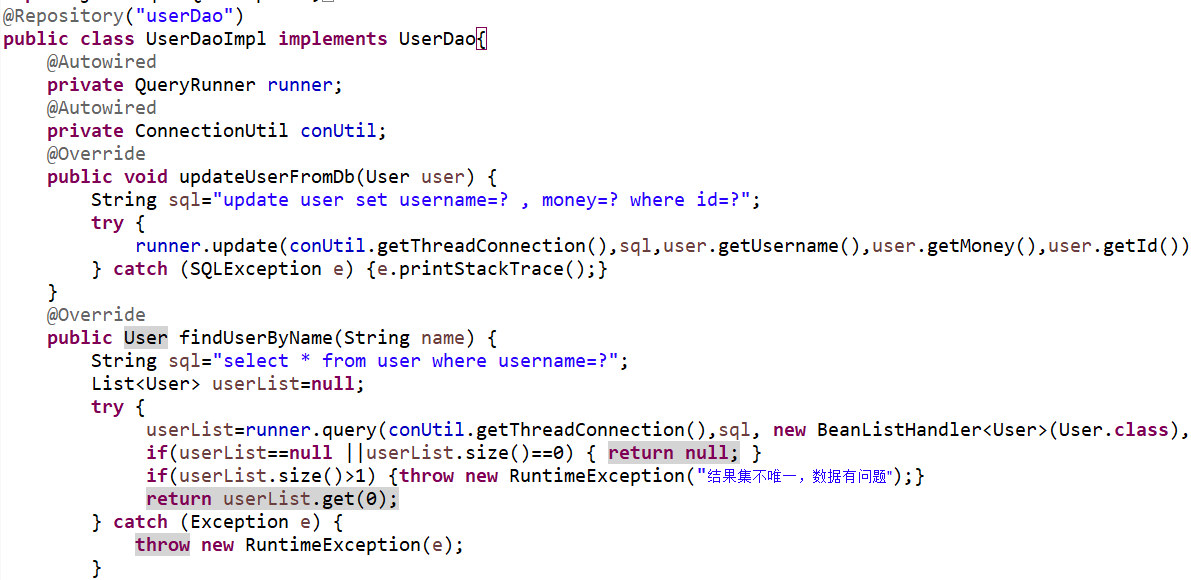
**例子**：

**两个工具类**：





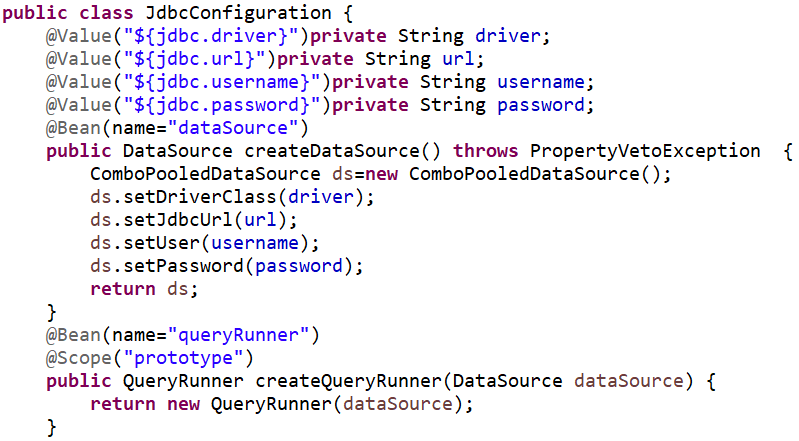
**dao接口的实现类**：

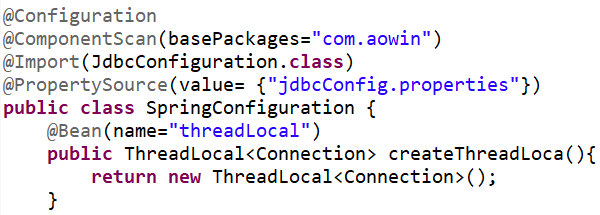


**service接口的实现类**：

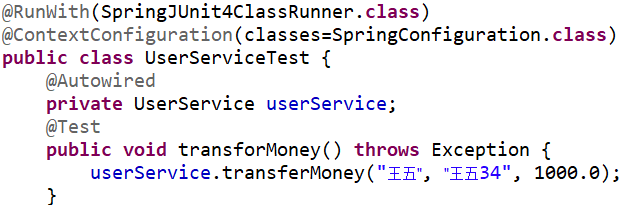


**配置类**：





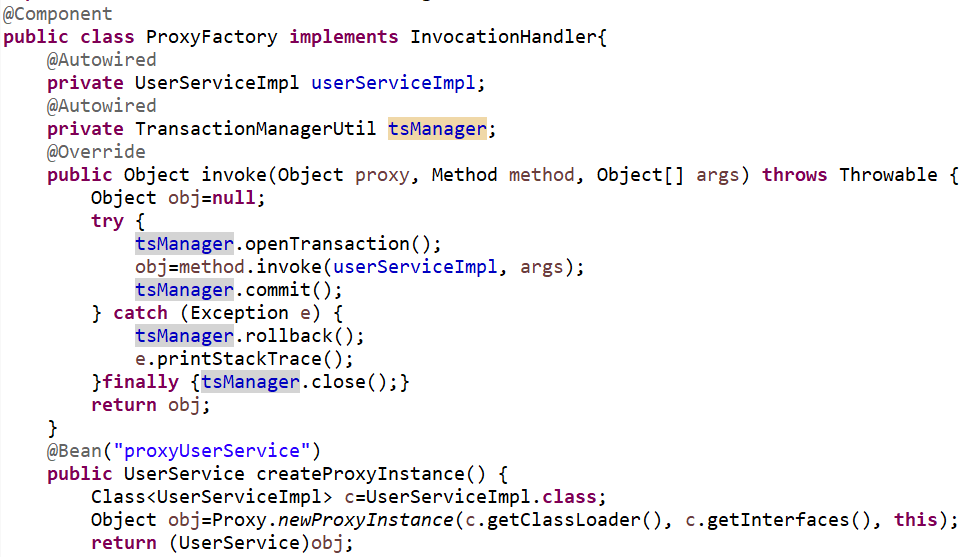
**测试类**：



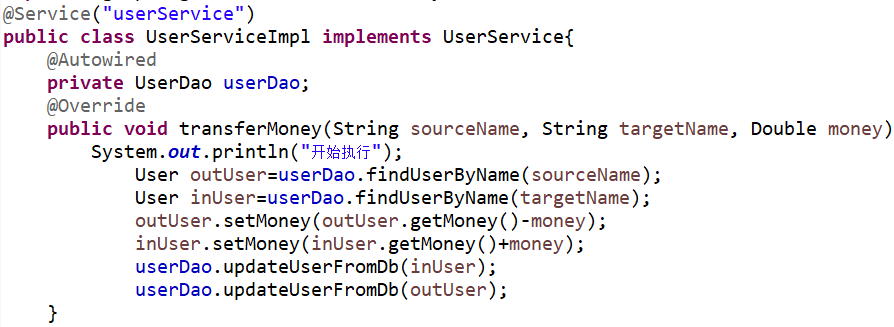
上面代码有点缺陷：若service接口中的方法都要使用到事务，则使用TransactionManagerUtil调用openTransaction()、commit()等方法就会重复很多遍，因此可以使用动态代理的方式解决这个问题。

**部分代码优化如下**：

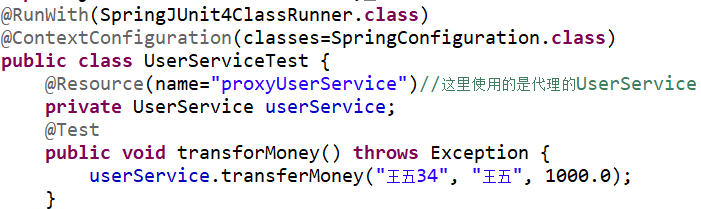
创建一个代理类，用来实现对Service实现类的代理。



Service接口的实现类的代码优化如下：



测试类的代码优化如下：



## 九、Spring中的AOP

### 1.AOP基本概念

**AOP(Aspect Oriented Programming)**：面向切面编程。面向切面编程是通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术，是OOP(面向对象编程)的延续。

AOP的**作用**：在程序运行期间，不修改源码的基础上实现对已有方法的增强。

AOP的**优势**：减少重复代码，提高开发效率，维护方便。

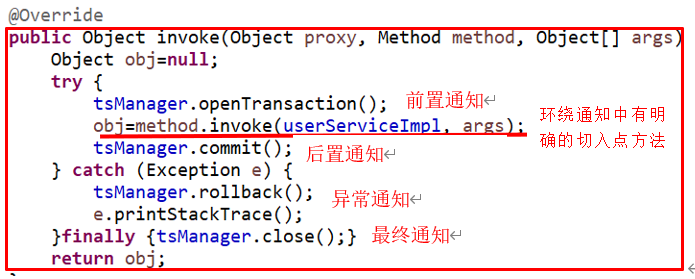
**AOP的实现方式就是使用动态代理技术**。

**①Joinpoint(连接点)**：是指那些被拦截到的点。在Spring中这些点指的就是方法，因为Spring中只支持方法类型的连接点。比如业务接口中所有的方法都可以称为连接点。

**②Pointcut(切入点)**：是指我们要对哪些Joinpoint进行拦截的定义。比如业务接口中那些被增强的方法则可以成为切入点。

**所有的切入点都是连接点，但连接点不一定是切入点**。

**③Advice(通知/增强)**：是指拦截到Joinpoint之后所要做的事情就是通知。通知的类型：前置通知、后置通知、异常通知、最终通知、环绕通知。下面在method.invoke()方法之前的就是前置通知，在之后的则是后置通知，在异常中的则是异常通知，在finally中的则是最终通知，整个invoke()方法在执行就是环绕通知。



⑥**Introduction(引入/引介)**：是一种特殊的通知，在不修改类代码的前提下，Introduction可以在运行期为类动态地添加一些方法或者Field。

**⑦Target(目标对象)**：被代理的对象。

**⑧Weaving(织入)**：是指把增强应用到目标对象来创建新的代理对象的过程。

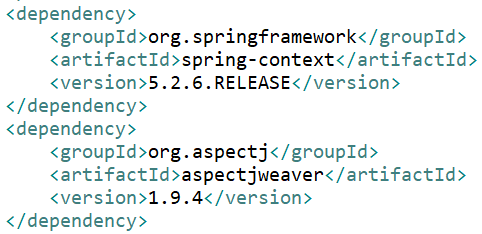
**⑨Proxy(代理)**：一个类被AOP织入增强后，就产生一个代理类。

**⑩Aspect(切面)**：是切入点和通知(引介)的结合。

### 2. Spring中基于xml配置的AOP

#### ⑴使用步骤

**步骤一**：在pom.xml文件中导入spring-context和aspectjweaver的依赖。



**步骤二**：创建一个xml文件用来配置AOP，这个xml文件的约束使用的是xmlns:aop中指定的约束，见：[Spring配置的范例文件\Spring-aop.xml](4.Spring配置的范例文件/Spring-aop.xml)

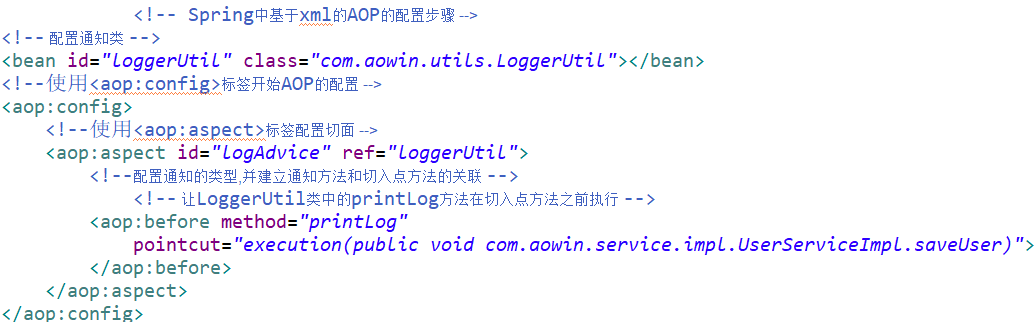
**步骤三：**在xml文件中配置AOP，详细步骤如下：

使用**<bean>**标签配置通知类，让Spring容器来管理通知类。

使用**<aop:config>**标签表明开始AOP的配置。

使用**<aop:aspect>**标签表明配置切面。该标签的**id属性**：用来给切面提供唯一的一个标识。**ref属性**：是指定通知类bean的id。

在<aop:aspect>标签的内部使用对应标签来配置通知的类型。



#### ⑵通知类型的标签

**①<aop:before>**标签表示**前置通知**，在切入点方法执行之前执行。

**②<aop:after-returning>**表**后置通知**，在切入点方法正常执行之后执行。

**③<aop:after-throwing>**表**异常通知**，在切入点方法执行产生异常之后执行。

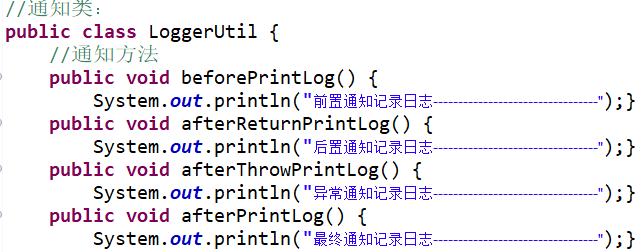
**后置通知和异常通知永远只能执行其中的一个，不可能同时执行。**

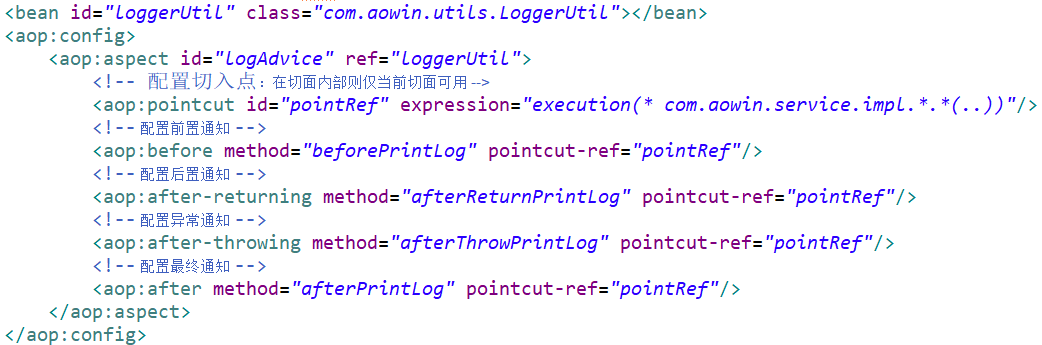
**④<aop:after>**表**最终通知**，无论切入点方法是否正常执行它都会在其后面执行。

这些通知类型的标签的**method属性**：用于指定通知类中的哪个方法是前置通知，**pointcut属性**：用于指定切入点表达式，该表达式的含义指的是对业务层中哪些方法增强。**pointcut-ref属性**：用于引用<aop:pointcut>标签配置的切入点表达式。

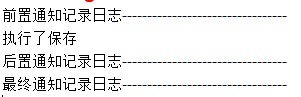
当切入点表达式的内容相同时，可以使用**<aop:pointcut>**标签配置切入点表达式，该标签的**id属性**用于指定表达式的唯一标识，**expression属性**用于指定表达式的内容。**注意**这个标签写在<aop:aspect>标签内部则只能在当前切面中使用；若写在<aop:aspect>标签外部，则变成所有切面可用。**注意**若写在<aop:aspect>标签的外部，一定要放在<aop:aspect>切面标签之前，因为这是xml的约束决定的，否则会报错。

**例子**：





运行测试结果如下：



**⑤<aop:around>**标签表**环绕通知**。在配置环绕通知的方法时，需要有明确的切入点方法调用，若没有，则不会执行切入点方法。Spring框架为我们提供了一个接口：**ProceedingJoinPoint**。该接口有一个方法proceed()，此方法就相当于明确调用切入点方法。该接口可以作为环绕通知的方法的参数，在程序执行时，Spring框架会为我们提供该接口的实现类供我们使用。

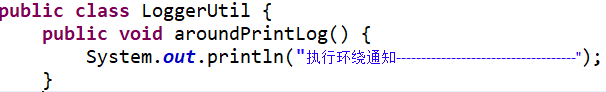
**ProceedingJoinPoint接口的常用方法如下**：

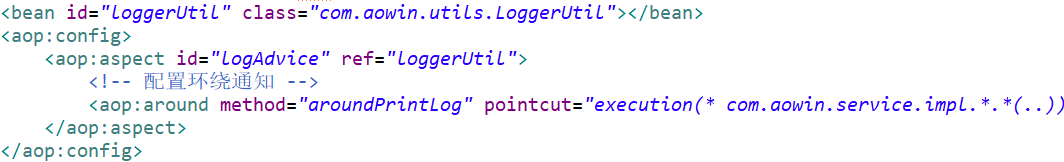
**getArgs()**：得到切入点方法的参数。返回值类型为Object[]。

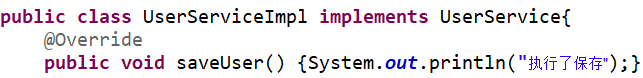
**proceed(args)**：明确调用切入点的方法，参数为方法的参数，返回值为切入点方法的返回值，返回类型为Object。

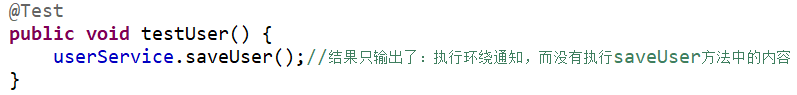
Spring中的环绕通知也可以理解是Spring框架为我们提供的一种可以在代码中手动控制增强方法何时执行的方式。

**环绕通知的例1**：



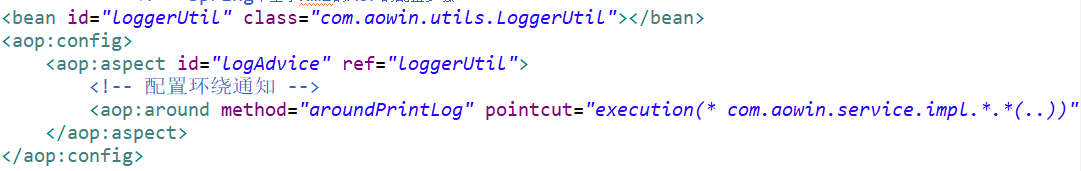


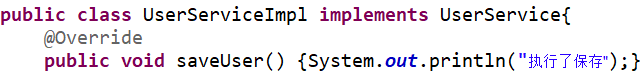


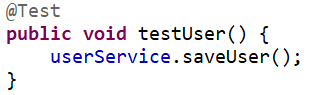


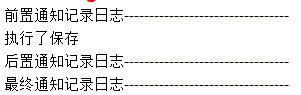
**环绕通知的例2**：











#### ⑶切入点表达式的写法

**关键字**：**execution(表达式)**

**表达式**：**访问修饰符 返回值 包名.类名.方法名(参数列表)**

标准表达式的写法示例如下：

**public com.aowin.service.impl.UserServiceImpl.saveUser(args)**

**①**表达式中访问权限修饰符可以省略**；**

**②返回值可以使用通配符(\*)表示任意返回值；**

**③包名可以使用通配符(\*)表示任意包，但是有几级包就需要写几个\*，如com.aowin.service.impl包写成\*.\*.\*.\*，同时包名可以是..表示当前包及其子包，如com.aowin.service.impl包可以写成\*..；**

**④类名和方法名也可以使用通配符\*；**

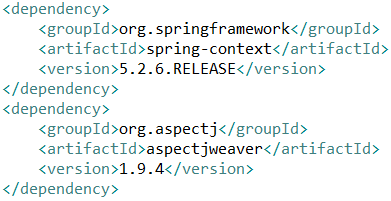
**⑤参数列表可以直接写对应的数据类型，对应基本数据类型直接写名称如int，对于引用类型写包名.类名的方式如java.lang.String，参数类型也可以使用通配符\*表示任意参数，但是必须有参数，也可使用..表示有无参数均可，若有参数则可以是任意类型。**

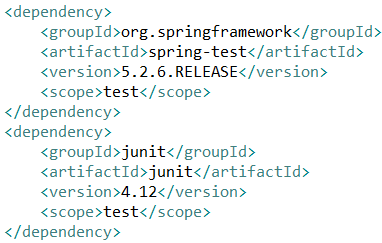
**实际开发中切入点表达式的通常写法：切入到业务实现类下的所有方法，即\* com.aowin.service.impl.\*.\*(..)。**

**pom.xml文件中导入的aspectjweaver依赖可以将上面的切入点表达式解析出来。**

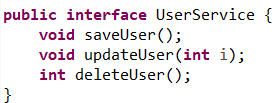
#### ⑷例子：

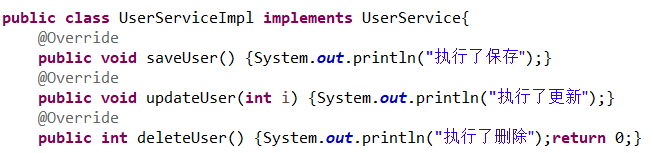
**pom.xml导入的依赖**



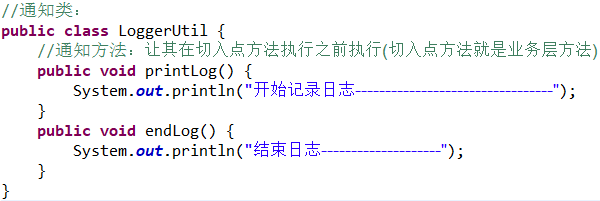


**Service接口及其实现类**





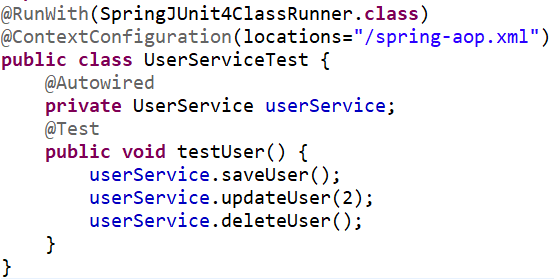
**通知类**



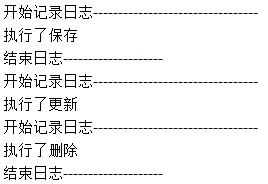
**配置文件**



**测试类**



**输出结果**



### 3.Spring中基于注解的AOP配置

#### ⑴常用注解：

**@Aspect注解**：表示当前类是一个切面类。

**@Pointcut()**注解：配置切入点表达式。定义一个方法，然后在这个方法上使用@Pointcut配置切入点表达式。

**@Before()**注解：前置通知。

**@AfterReturning()**注解：后置通知。

**@AfterThrowing()**注解：异常通知。

**@After()**注解：最终通知。

**@Around()**注解：环绕通知。

在实际开发中，不要使用@Before()、@AfterReturning()、@AfterThrowing()、@Around()注解配置通知，因为Spring使用这几个注解配置AOP时其执行顺序有问题，首先执行的是前置通知，然后是最终通知，之后才是后置通知或者异常通知。而是使用@Around()环绕通知，自己手动定义增强功能的执行顺序。

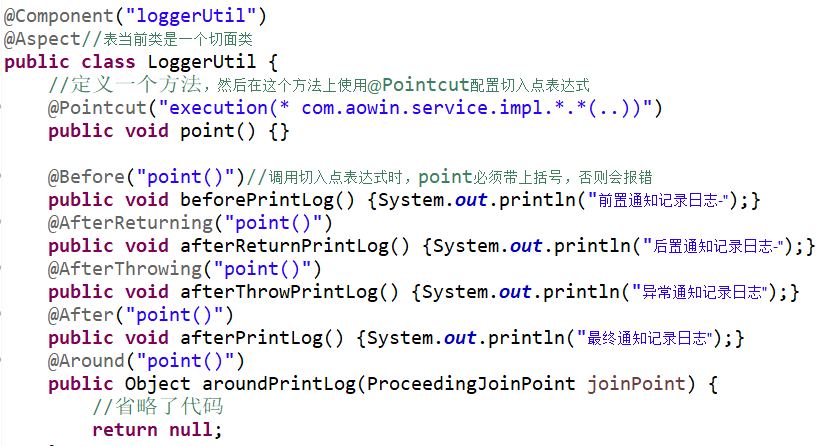
**注意**：使用注解的方式配置AOP，别忘了在xml文件中使用**<aop:aspectj-autoproxy>**标签开启注解AOP的支持，或者在配置类中使用**@EnableAspectJAutoProxy**的方式。

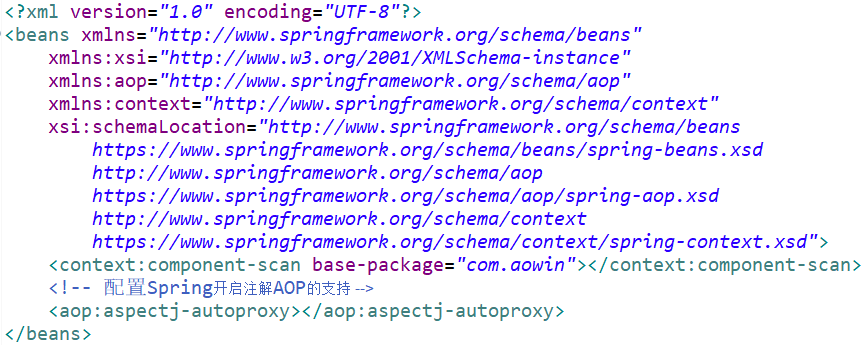
#### ⑵注解、纯注解配置AOP的例子：

**①使用xml配置AOP的方式**：

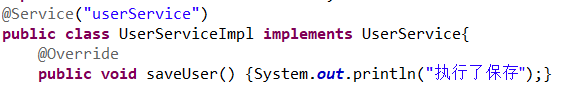


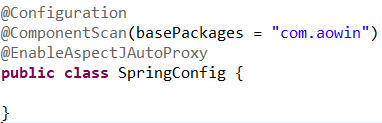
**②使用注解配置AOP的方式**：



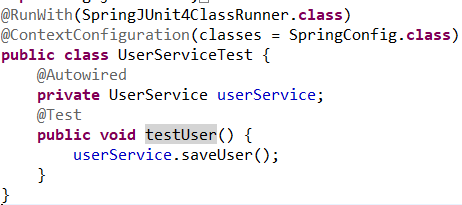


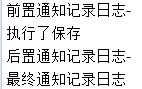
**③使用纯注解的方式**：











## 十、Spring中的JdbcTemplate

### 1.JdbcTemplate的基本用法：

JdbcTemplate的作用：它是用于和数据库交互的，实现对表的CRUD操作。

在Spring中使用JdbcTemplate操作数据库，需要在pom.xml中导入如下的依赖：spring-context的依赖(**任何Spring的运行都少不了这个**)，spring-jdbc的依赖，spring-tx的依赖(这个是事务处理)，这几个都是org. springframework中的。同时还需要导入mysql的依赖。

**JdbcTemplate的基本用法**：



### 2.使用JdbcTemplate实现单表CRUD操作

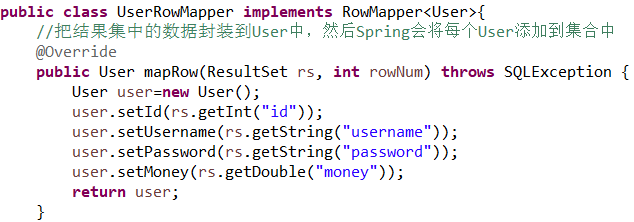
增删改操作使用的是JdbcTemplate的update()方法。

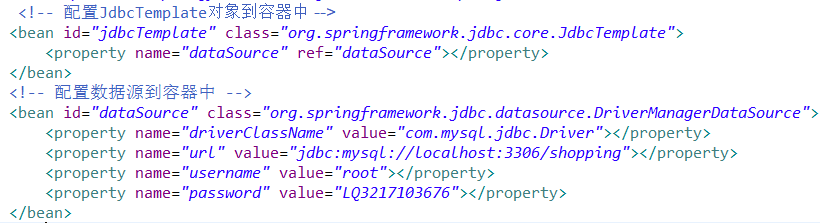
对于查询操作，需要使用query()方法，这个方法有一个RowMapper接口的参数。这个RowMapper是用来定义对查询结果的封装策略。这个RowMapper类型的参数可以自定义一个类实现这个接口，然后重写方法进行数据的封装，也可以使用Spring自己的BeanPropertyRowMapper类作为参数。

当查询结果返回的是单个结果，则使用queryForObject()方法，该方法有一个参数类型是Class类型的，这个参数用来指定查询的结果返回的数据类型是什么。

JdbcTemplate类中提供了一个setDataSource()的方法，因此在下面的配置中可以使用<property>标签注入到JdbcTemplate中。

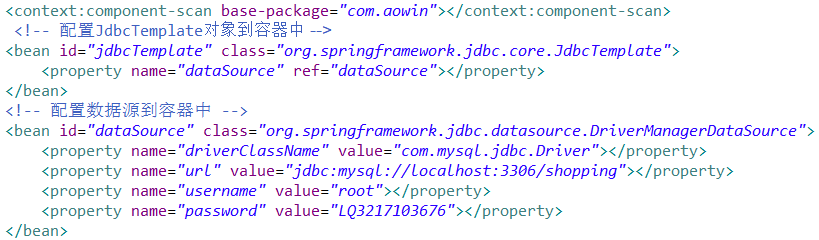
**例子1：**



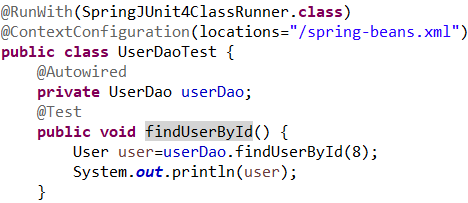




**JdbcTemplate使用在dao中的例子2**：

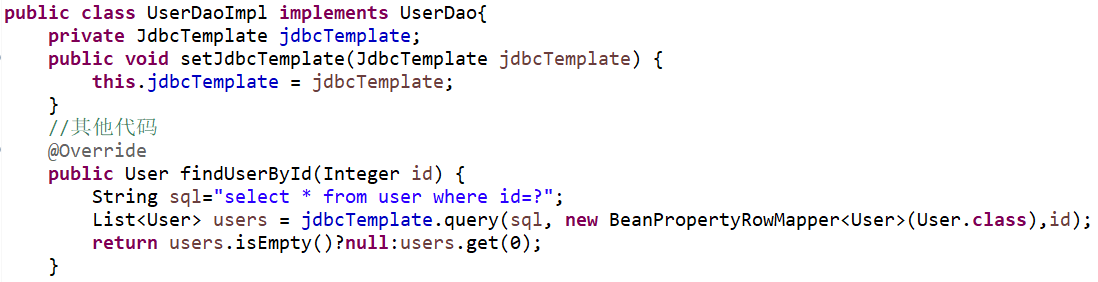




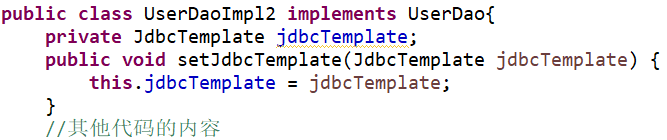


### 3.JdbcDaoSupport的使用

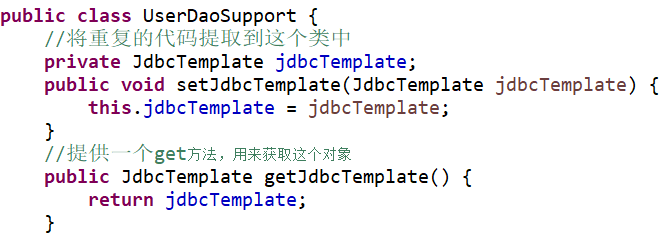
**思路**：比如下面这两个类中有重复的代码，解决办法是设计一个基类，用来存放这些重复的代码，然后让下面这些类继承这个新创建的类，父类中提供一个get()方法用来获取对象，子类中则采用super关键字调用父类的get方法获得需要的对象。

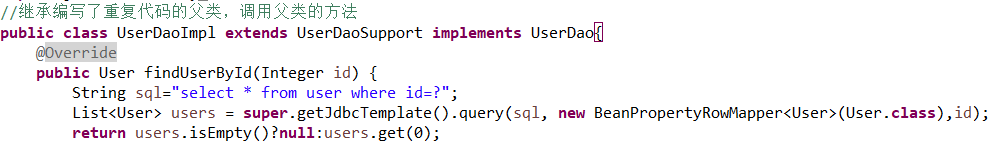


重复代码



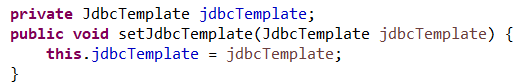
**上面代码可优化为如下：**



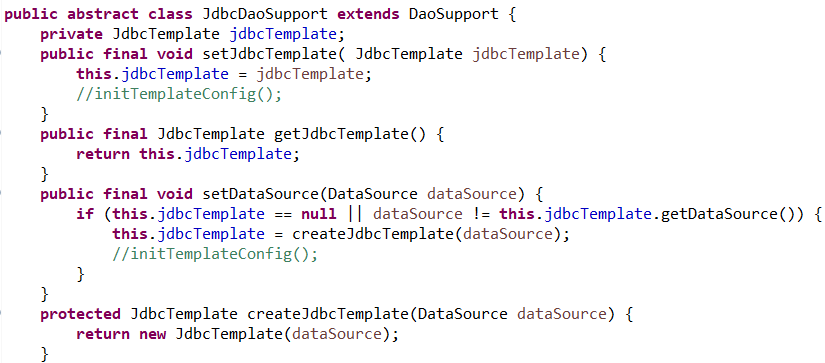




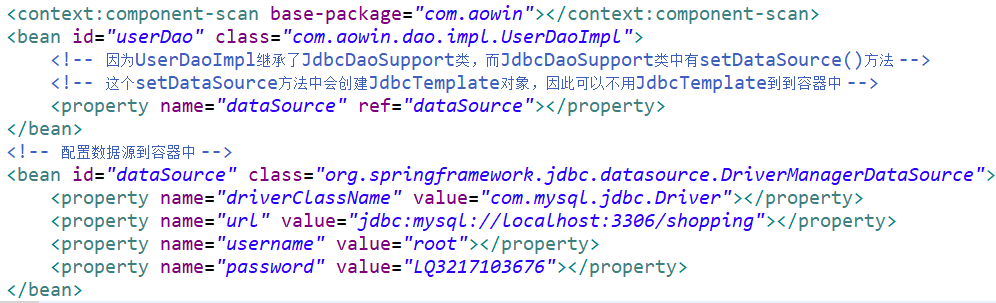
当有多个dao接口时，每个dao接口的实现类中都会用到下面这段代码，对于这段重复的代码，Spring中定义了一个JdbcDaoSupport类，用来存放下面这段重复代码，因此dao接口的实现类只要继承JdbcDaoSupport这个类就可以了，然后采用super关键字调用父类的getJdbcTemplate()方法获得JdbcTemplate对象。



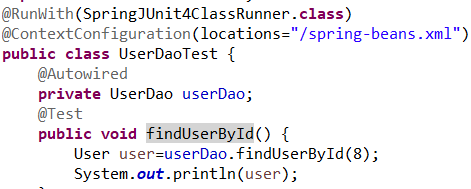
Spring中JdbcDaoSupport类的部分源码如下：



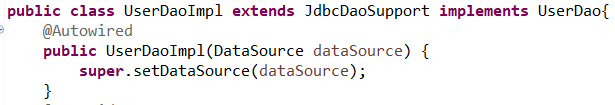
因此，**使用JdbcTemplate实现单表CRUD操作的例子2的代码可以优化为如下**：



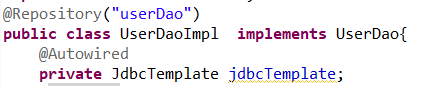


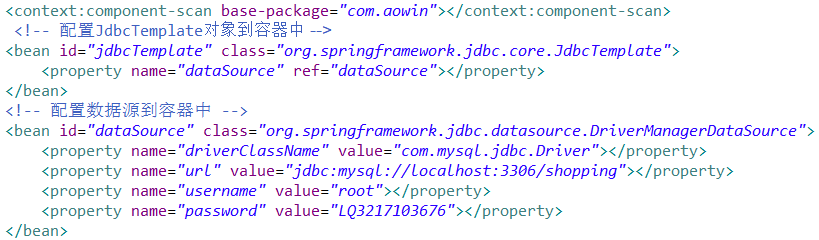


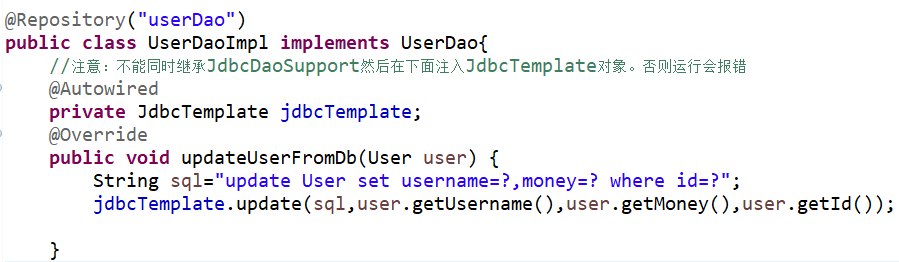
注意继承JdbcDaoSupport类之后，则直接使用@Repository注解UserDao的话，则不太好注入DataSource对象，除非在这个类中定义一个构造器，采用自动装配，如下所示，但这样又会导致多个dao实现类中出现重复代码。而使用<bean>标签注入则可以很好解决，如上例子所示。



因此，对于使用xml配置，则建议继承JdbcDaoSupport类，**对于使用注解注入的话，建议不要继承JdbcDaoSupport类**，而是直接使用注解注入JdbcTemplate，在xml中则配置成如下：







## 十一、Spring中采用AOP的事务控制(理解)

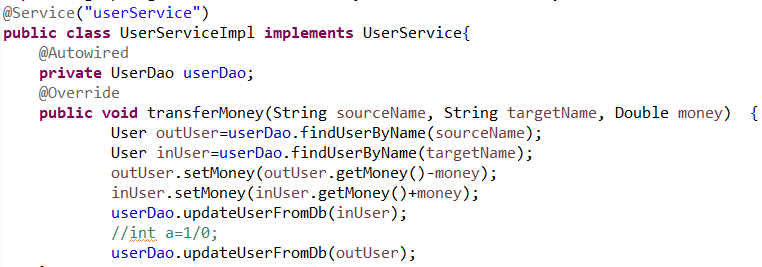
这部分内容是自己编写事务管理器，然后采用Spring的AOP实现事务的控制，而[八、spring和动态代理结合且手动事务控制](#_八、spring和动态代理结合且手动事务控制)则是自己编写事务管理器结合动态代理实现的事务控制。

### 1.xml的AOP事务控制

**dao实现类：**



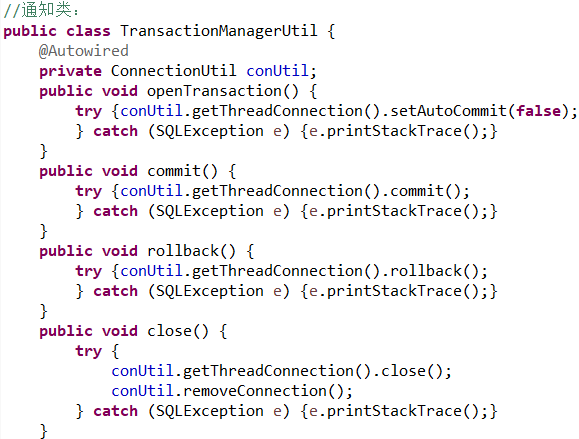
**service实现类：**



**工具类**：



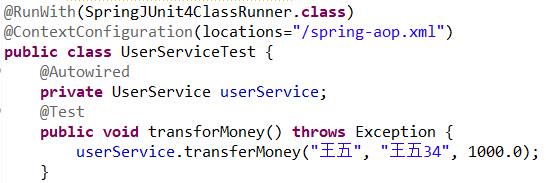
**通知类**：



**配置文件**：



**测试类**：

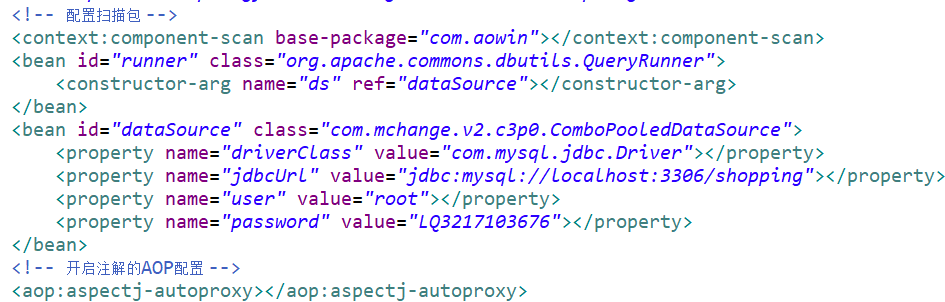


### 2.xml配合注解的AOP事务控制

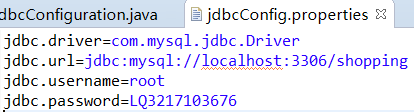
将xml版本的例子中的通知类修改成如下，即在通知类中使用注解配置。



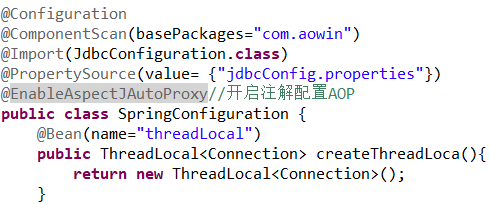
使用注解配置完通知类(切面类)之后，在xml文件中使用<aop:aspectj-autoproxy>开启AOP的注解配置。

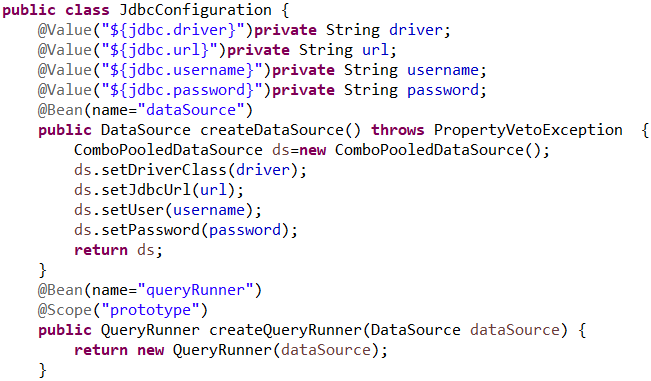


### 3.纯注解的AOP事务控制

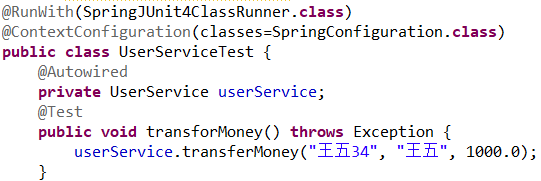


**配置类**：





**测试类**：



dao实现类、service实现类的代码同xml配置注解的AOP事务控制的例子一样。

## 十二、Spring中事务控制的API

Spring框架为我们提供了一组事务控制的接口，这组接口是存放在**spring-tx的jar包**中的。Spring的事务控制都是基于AOP的，因此不可缺少**aspectjweaver这个jar包**，可使用编程的方式实现，也可使用配置的方式实现。

### 1.PlatformTransactionManager接口

**PlatformTransactionManager**接口是Spring的事务管理器，它提供了操作事务的方法：

TransactionStatus **getTransaction**(TransactionDefinition definition)：该方法是获取事务状态信息。

void **commit**(TransactionStatus status)：提交事务

void **rollback**(TransactionStatus status)：回滚事务

在实际开发中，是使用PlatformTransactionManager的实现类来实现事务管理的，若使用SpringJDBC或者iBatis进行持久化数据时，使用**DataSourceTransactionManager**这个类进行事务管理。

### 2.TransactionDefinition接口

这个接口是事务的定义信息对象。它具有**如下方法**：

String **getName**()：获取事务对象的名称。

int **getIsolationLevel**()：获取事务的隔离级别。事务的隔离级别有四个，Spring默认使用的是数据库的隔离级别。

int **getPropagationBehavior**()：获取事务传播行为。即什么时候必须有事务，什么时候事务可有可无。一般增删改必须有事务，查询操作可有可无。

int **getTimeout**()：获取事务超时时间。

boolean **isReadOnly**()：获取事务是否只读。建议查询操作设置事务只读。

事务隔离级别反映事务提交并发访问时的处理态度。下面**四种事务的隔离级别**：

***ISOLATION\_READ\_UNCOMMITTED***：可以读取未提交的数据。

***ISOLATION\_READ\_COMMITTED***：只能读取已提交的数据，解决脏读问题(Oracle默认级别)

***ISOLATION\_REPEATABLE\_READ***：是否读取其他事务提交修改后的数据，解决不可重复读问题(MySQL默认级别)

***ISOLATION\_SERIALIZABLE***：是否读取其他事务提交添加后的数据，解决幻影读问题。

**事务的传播行为**：

**REQUIRED**:若当前没有事务，则新建一个事务，若已经存在一个事务，则加入到这个事务中。一般选择默认值。

**SUPPORTS**：支持当前事务，若当前没有事务，就以非事务方式执行。这个**只有查询操作才能用**。

**超时时间**：默认值是-1，表没有超时限制，若有，则以秒为单位进行设置。

### 3.TransactionStatus接口

此接口提供的是事务具体的运行状态。这个接口描述了某个时间点上事务对象的状态信息。常用方法：

void **flush**()：刷新事务。

boolean **hasSavepoint**()：获取是否存在存储点。

boolean **isCompleted**()：获取事务是否完成。

boolean **isNewTransaction**()：获取事务是否为新事务。

boolean **isRollbackOnly**()：获取事务是否回滚。

void **setRollbackOnly**()：设置事务回滚。

## 十三、Spring中的声明式事务控制(重点)

Spring中的事务控制是基于AOP的。这里使用的是Spring中自身的事务管理器**DataSourceTransactionManager**进行事务管理。注意这个类是在spring-jdbc依赖中的，而不是在spring-tx依赖中的。@Transactional注解默认是运行时异常事务才有效，若遇到编译期异常则无效，可以添加rollbackFor=Exception.class解决。

### 1.配置事务所使用的标签

**①<tx:advice>**标签：用来配置事务的通知。该标签的**id**属性是给事务通知起一个唯一标识。**transaction-manager**属性：给事务通知提供一个事务管理器的引用。

**②<tx:attributes>**标签：用来配置事务的属性。这个标签在<tx:advice>内部使用。

**③<tx:method>**标签：这个标签在<tx:attributes>标签中使用，用来给某个方法配置事务的属性。该标签有如下属性：

**name属性**：值可以为业务类中的某个需要使用事务的方法名；也可以使用统配符\*表示业务类中的所有方法都使用设置的事务属性，或者使用如“find\*”这样的方式，表示以find开头的所有方法都使用设置的事务属性。形如“find\*”的级别比“\*”的级别要高。建议查询操作对应的方法名都统一以find或者select等开头，这样就可以使用“find\*”或者“select\*”的方式给所有的查询操作配置事务的属性。

**isolation属性**：用于指定事务的隔离级别，**默认值是DEFAULT，表示使用数据库的默认隔离级别**；

**propagation属性**：用于指定事务的传播行为。**默认值是REQUIRED，表示一定会有事务，增删改操作要使用这个。查询操作的方法可以选择SUPPORTS**。

**read-only属性**：用于指定事务是否可读。**只有查询方法才能设置为true，默认值为false，表示可读写**。

**timeout**属性：用于指定事务的超时时间，**默认值是-1，表示永不超时**。若指定了数值，以秒为单位。

**rollback-for**属性：用于指定一个异常，当发生该异常时，事务回滚，发生其他异常时，事务不回滚。**没有默认值，若没设置，则表示任何异常都回滚**。

**no-rallback-for**属性：用于指定一个异常，当发生该异常时，事务不回滚，产生其他异常时事务回滚。**没有默认值，不设置时表任何异常都回滚**。



**④<tx:annotation-driven>**标签：当使用注解配置AOP的事务控制时，使用该标签用来开启Spring对注解的事务控制。其**transaction-manager**属性用来引用一个事务管理器。



**⑤@EnableTransactionManagement**注解：当使用纯注解的方式配置AOP事务控制时，需要使用这个注解开启事务配置。这个注解的功能和<tx:annotation-driven>标签的一样。

### 2.xml的AOP事务控制配置步骤

**步骤一**：配置事务管理器。这里需要使用的是**DataSourceTransactionManager**事务管理器。

**步骤二**：使用**<tx:advice>**标签配置事务的通知，同时需要导入事务的xml约束。

事务的约束需要使用tx的约束，也需要aop的约束，因为Spring中的事务是基于AOP的。

**步骤三**：在<tx:advice>标签内使用<tx:attributes>标签配置事务的属性。然后使用<tx:method>标签配置指定某个方法需要的事务属性。

**步骤四**：在AOP配置中配置通用切入点表达式，并使用<aop:advisor>标签建立切入点表达式和事务通知的对应关系。

**例子**：

**xml文件**：



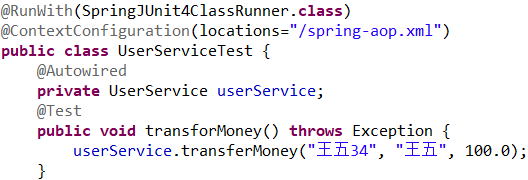
**dao实现类**：



**service实现类**：



**测试类**：



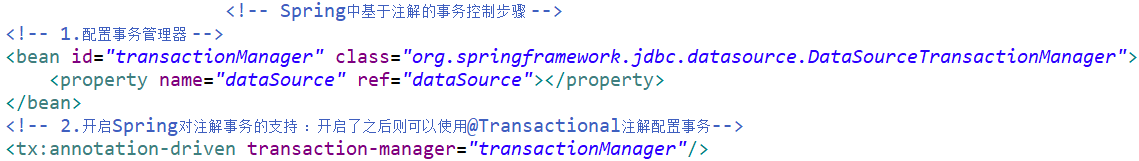
### 3.注解的AOP事务控制配置步骤

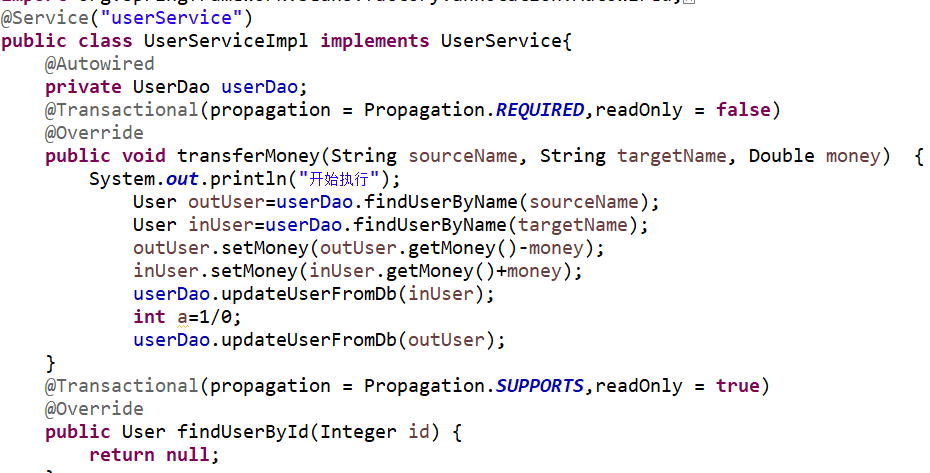
**步骤一**：配置事务管理器DataSourceTransactionManager。

**步骤二**：使用<tx:annotaion-driven>标签开启Spring对注解事务的支持。

**步骤三**：在需要使用到事务的业务层中使用**@Transactional**注解配置事务。

在@Transactional注解中可以配置事务的属性，不配置则使用的是默认的属性。该注解标注在类前，表示类中的所有方法都进行事务处理。标注在方法前，则表示该方法进行事务处理。





### 4.纯注解的AOP事务控制案例

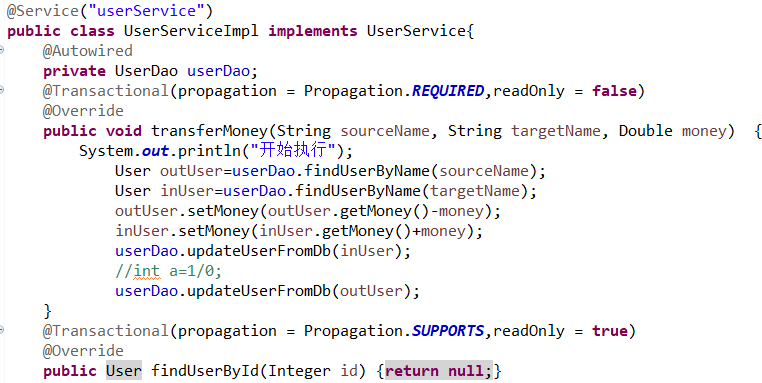
使用纯注解配置AOP事务控制时，需要在配置类上添加@EnableTransactionManagement注解开启AOP的事务控制。

**例子**：

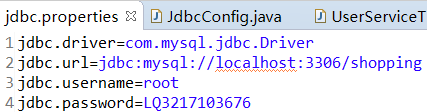
**dao接口实现类**：



**service接口实现类**：

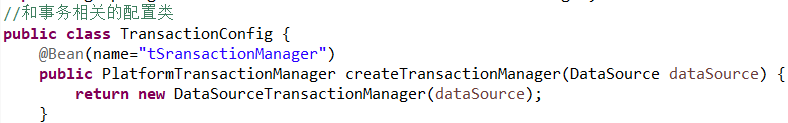


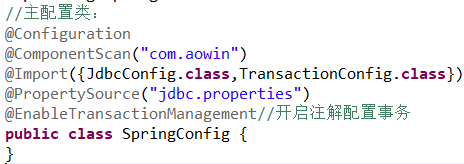
**jdbc.properties文件**：



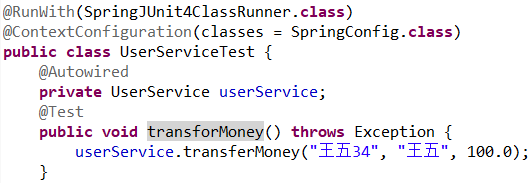
**配置类**：







**测试类**：



## 十四、Spring中基于编程式事务控制(了解)

实在有时间多再学习，基本不怎么使用这种方法。

## 十五、Spring5的新特性

Spring5.0是基于jdk1.8编写的。

有时间多再学习。

## 十六、总结

### 1.Spring中约束的使用

**spring-beans.xsd**的约束：使用xml的方式将bean对象配置到spring的容器中需要这个约束。

**spring-context.xsd**的约束：使用注解的方式进行Spring的配置需要使用这个约束。

**spring-aop.xsd**的约束：Spring中使用AOP时需要使用这个约束。

**spring-tx.xsd**的约束：使用Spring自己的事务管理器管理事务时，需要使用到这个约束，同时也必须使用spring-aop.xsd这个约束。因为Spring自身的事务管理是基于AOP的。

### 2.Spring中常用的核心容器

一个Spring容器可以理解为某个实现了ApplicationContext接口的类的实例。Spring提供两种类型的容器：**ApplicationContext接口**、**BeanFactory接口**。

**创建容器常用的方式**：

**①**使用ClassPathXmlApplicationContext这个类加载类路径下的xml文件创建Spring容器。

**②**使用AnnotationConfigApplicationContext这个类读取关于注解的配置类创建Spring容器。





**③**使用FileSystemXmlApplicationContext这个不常用。

在Junit中使用@Runwith和@ContextConfiguration注解一起使用时，可以不用手动创建出Spring的容器对象，使用@Autowired自动装配就可以使用容器中的bean对象。

### 3.Spring中创建bean的三种方式

**①**通过默认的无参构造函数创建bean对象。这种方式的本质是将bean这个类交由Spring自带的BeanFactory工厂管理，通过BeanFactory来管理和维护这个类。

**②**使用工厂中的普通方法创建bean对象。这种方式的本质是把创建实例的工厂类和调用工厂类的方法创建实例对象都交由Spring类管理，而创建实例对象的过程则在工厂类中实现。

**③**使用工厂中的静态方法创建bean对象。这种方式的本质是把bean这个类交给我们自己的这个工厂来管理，而Spring只是调用工厂类创建实例的方法。创建实例对象的过程也是在工厂类中实现。

### 4.Spring中的实现IOC的三种方式

**①**在xml文件中使用<bean>标签配置。

**②**使用注解@Component、@Service、@Repository、@Controller配置bean到Spring的容器中，然后在xml文件中使用<context:component-scan>配置扫描。

**③**创建一个配置类，代替xml文件的内容，使用@Configuration注解声明这是一个配置类。然后使用@Bean注解将当前方法的返回值作为bean对象存入spring的IoC容器中。

### 5.Spring中的DI的常用方式

**①**若是在xml文件中，可以在<bean>标签内使用<constructor-arg>标签完成注入，这种方式采用的是类中对应的构造函数完成的注入。

**②**若是在xml文件中，也可以在<bean>标签内使用<property >标签完成注入，这种方式采用的是类中对应的set方法完成的注入，这种方式要注意使用的是类中的无参构造器，若类中没有对应的无参构造器，则会注入失败。

**③**采用xml中的<bean>标签或者@Component等注解或者在配置类中使用@Bean注解将bean对象注入到容器中之后，Spring的容器中已存在所需的bean对象，则可以通过@Autowired、@Resource等注解完成bean对象的自动装配注入。

使用注解注入数据时，基本数据类型、包装类及String类型使用@Value注解注入，bean对象则采用@Autowired、@Resource等注解注入，对于集合类型的数据则只能在xml中完成注入。

## 十七、常用的数据源的配置

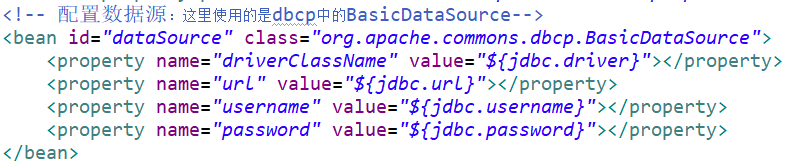
c3p0：c3p0是自动化操作，可以自动化加载配置文件，且可以自动设置到对象中。使用ComboPooledDatasource类。

dbcp：其中dbcp是半自动化操作，不能自动连接。使用BasicDataSource类。

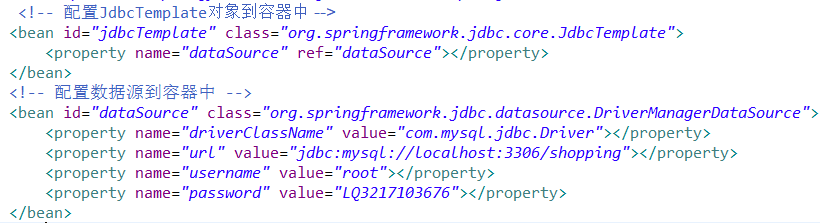
spring自带的DriverManagerDataSource也可以作为数据源。

(还有druid、hikari，这两个还没接触过)。

**dbcp中的BasicDataSource配置如下**：



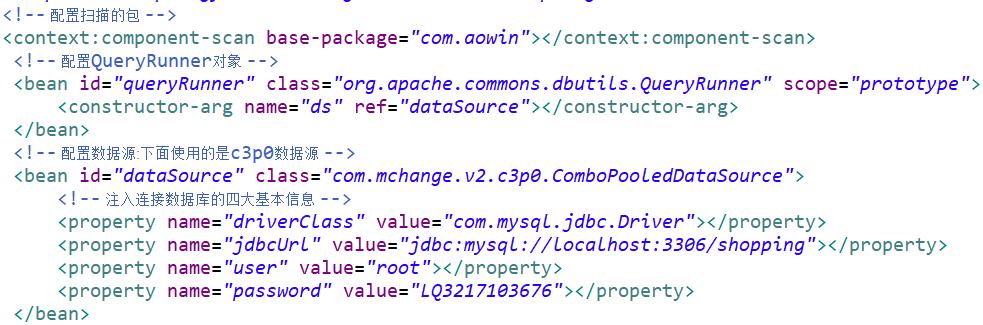
**spring自带的DriverManagerDataSource配置如下：**



**直接在mybatis的主配置文件中的配置如下**：



**c3p0中的ComboPooledDataSource中的配置如下**：



**上面主要要注意<property>标签中的name的不同值**。

注意：可变参数是jdk1.5之后才有的，因此对于参数类型是可变参数时，说明运行这个代码要在jdk1.5版本之后才能使用。

抽象类能实现接口，接口不能继承类，接口只能继承接口。