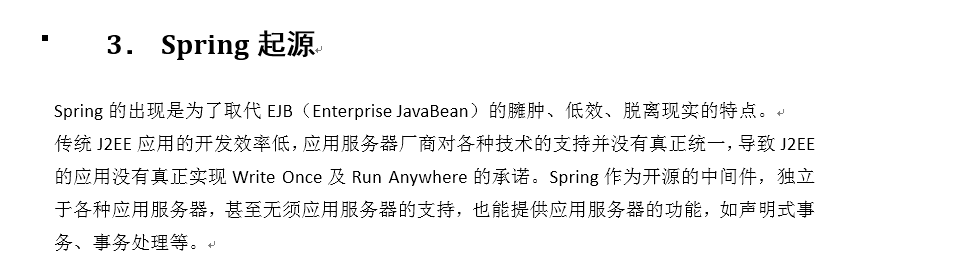
# Spring第一天

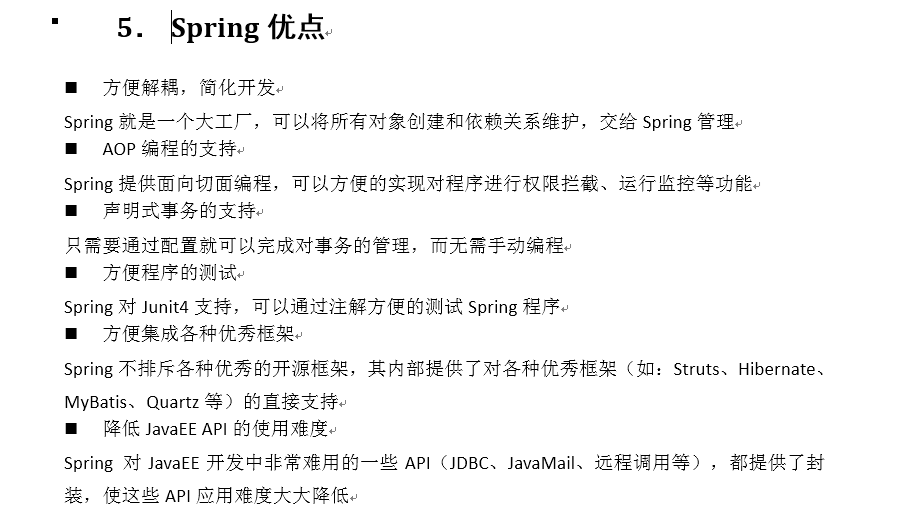
# 一：spring的介绍

Spring是什么：spring是一个一站式的解决框架。

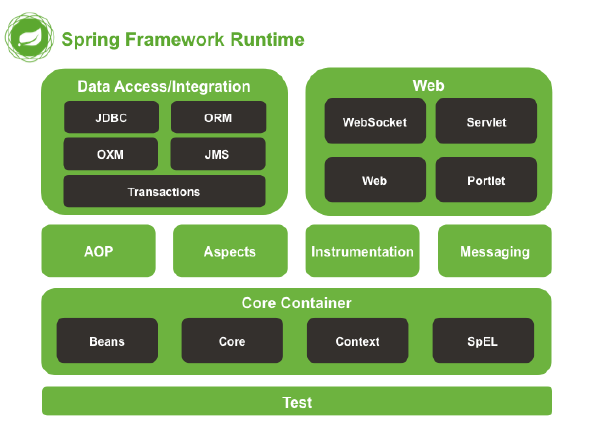
解决了什么问题：解决了我们企业开发中各个层级的问题。

一站式解决方案：说白了就是spring可以支持我们各个层级包括我们的数据展示层，我们的业务层，我们的持久层等都有对应的解决方案。





# 二：spring的架构体系



Test：用于测试使用

Core container：核心容器。

容器是什么？？？用于装Java bean

Data access：数据访问。用于访问操作我们的数据库。支持我们持久层的操作。

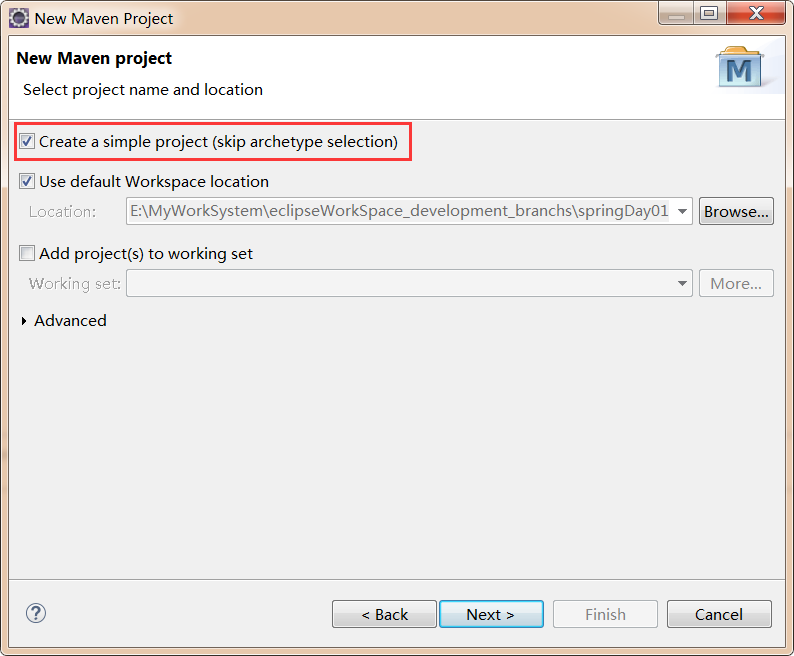
Web：集成我们的数据访问。用于支持我们的http请求。用于支持我们的数据展示层

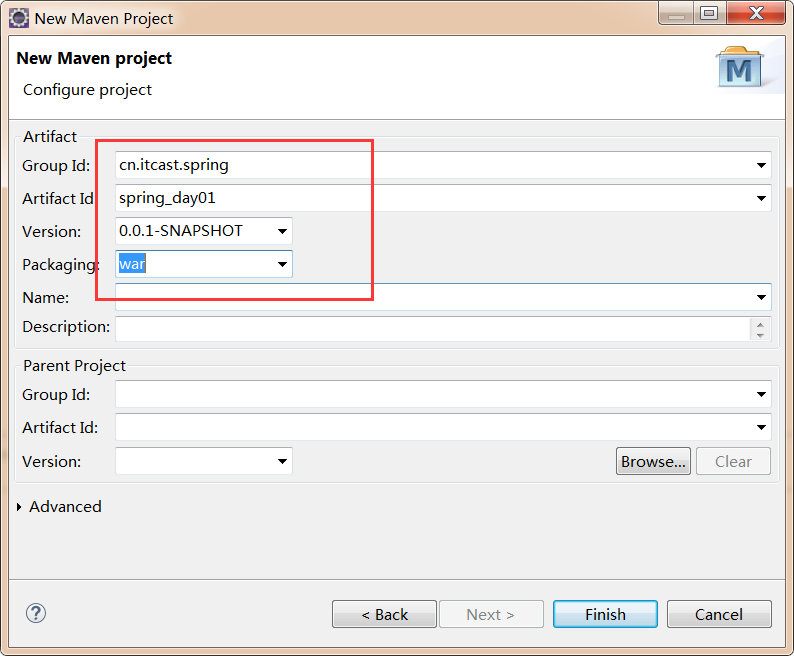
Transactions：用于支持我们的事物处理。用于解决我么业务层的事物处理问题。

# 三：spring中IOC与DI的快速入门

## 2：spring容器以及快速入门

### 2.1：创建maven web工程





### 2.2解决web工程的两个问题

第一个问题：jdk版本过低问题

解决办法：在pom.xml中添加一个jdk的编译插件即可。

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

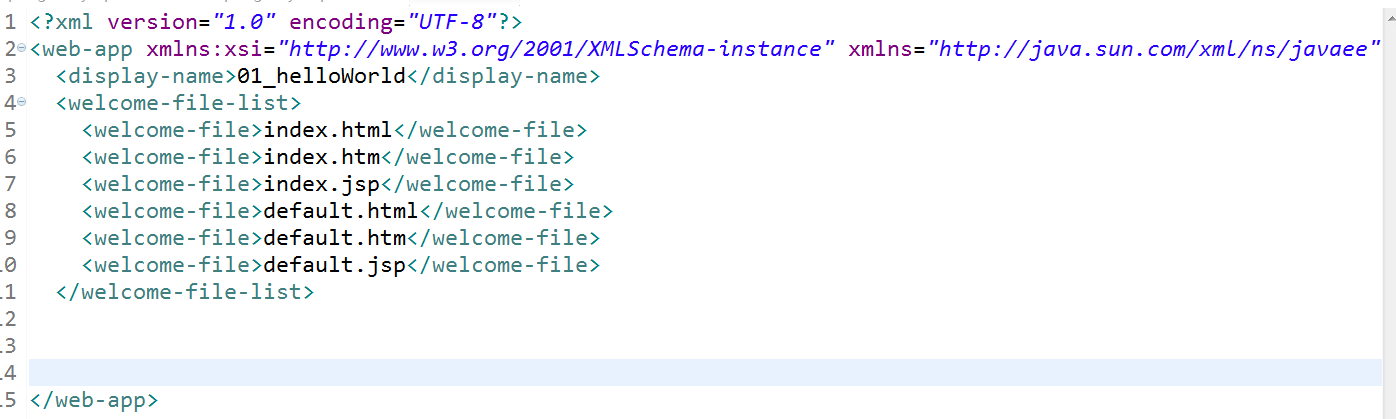
</plugin>

</plugins>

</build>

第二个问题：丢失web.xml的问题

解决方案：从其他项目中拷贝一个web.xml即可。



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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<display-name>01\_helloWorld</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

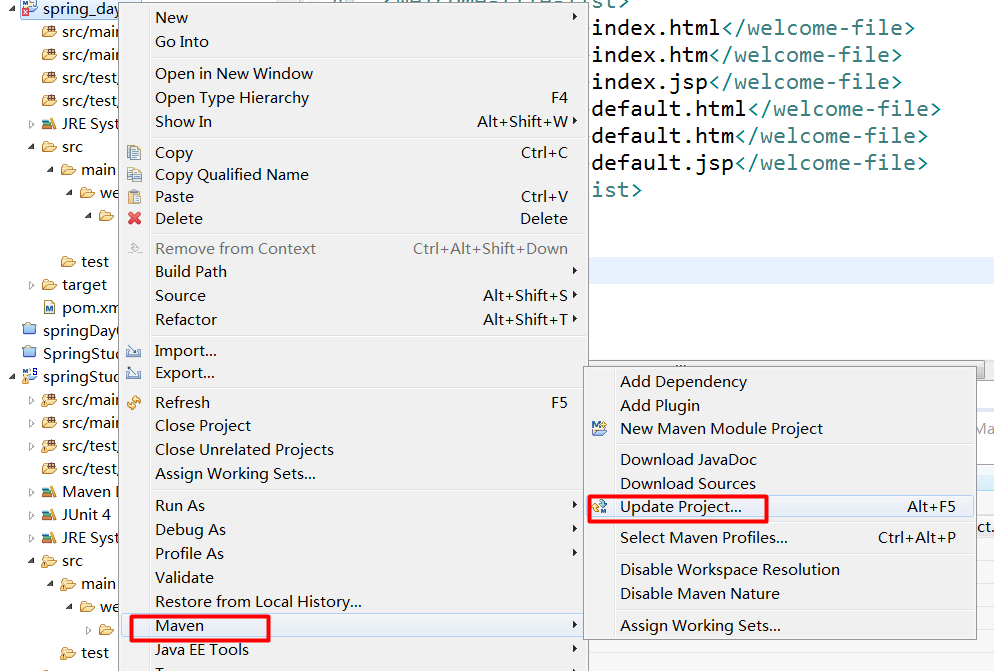
<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

将我们的web项目进行update



### 2.3导入我们工程必须的jar包

#### 第一步：导入必须的jar包

Jar包的查找网址：<http://mvnrepository.com/>

在pom.xml中添加我们的core container的核心依赖

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-expression -->

<dependency>

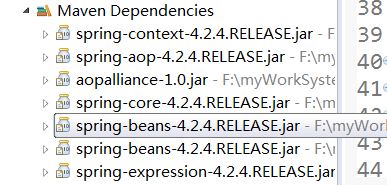
<groupId>org.springframework</groupId>

<artifactId>spring-expression</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

</dependencies>



今天所用到的所有jar包

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-expression -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-expression</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-test -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.2.4.RELEASE</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

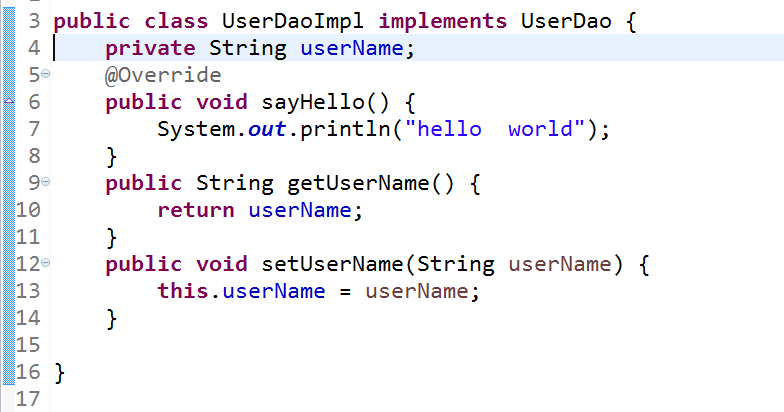
</plugins>

</build>

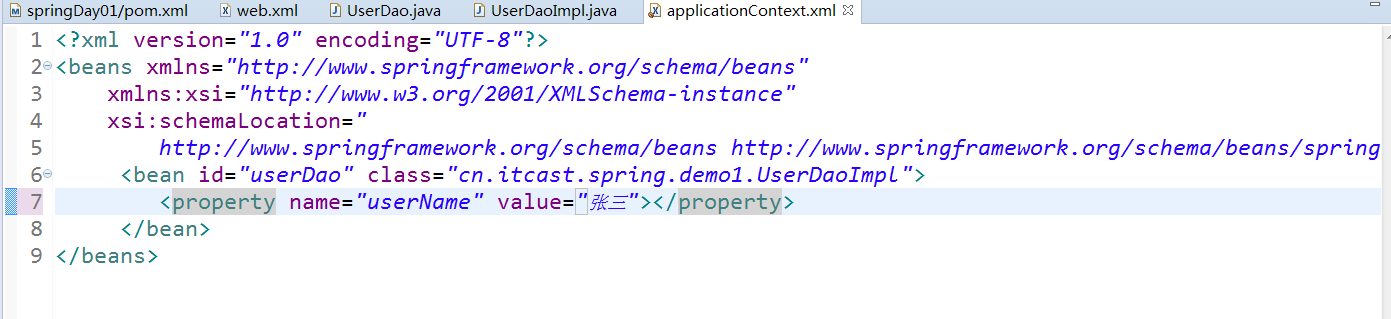
#### 第二步：IOC与DI的快速入门

##### 快速入门

创建接口和实现类



配置spring容器



<?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"*>

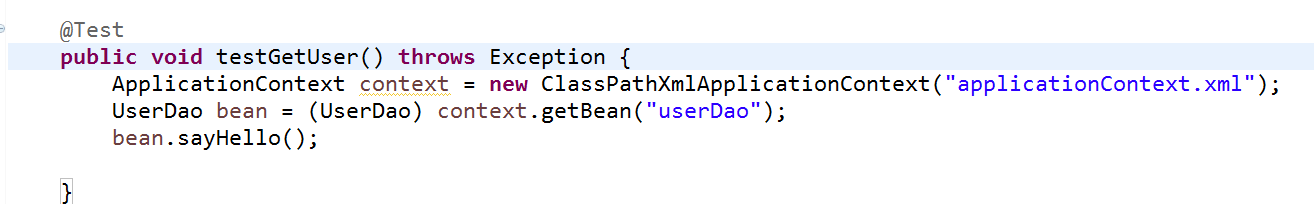
<bean id=*"userDao"* class=*"cn.itcast.spring.demo1.UserDaoImpl"*>

<property name=*"userName"* value=*"张三"*></property>

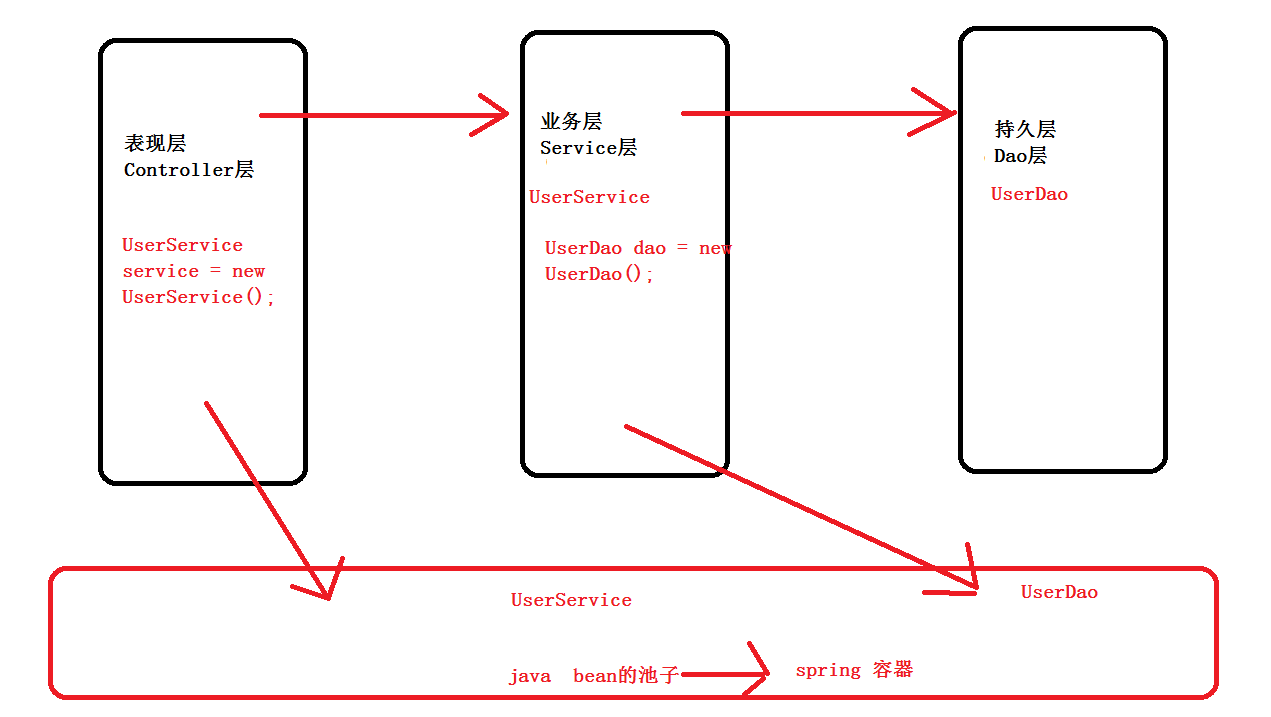
</bean>

</beans>

测试用例



##### 1：Spring容器的概念



Spring容器，就是用来装JavaBean的。

##### IOC与di的区别与联系

IOC是创建我们的对象

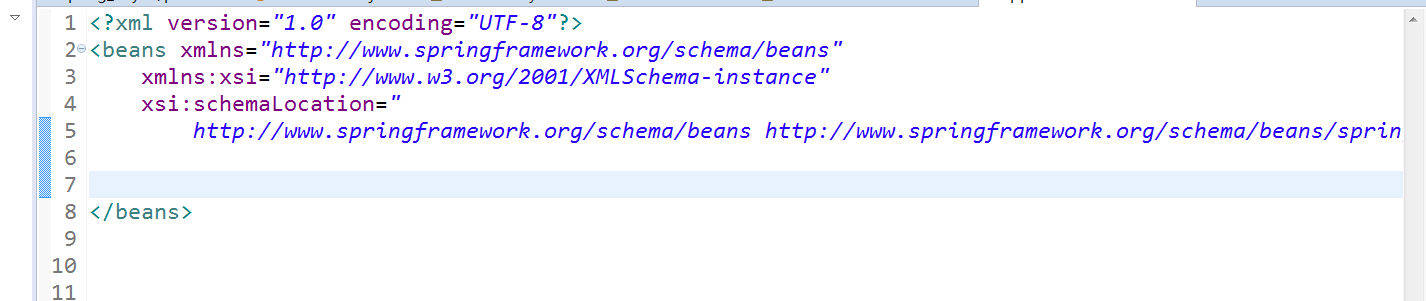
DI就是为我们对象中的属性赋值

#### 第三步：查找我们的容器类BeanFactory

##### 第一种方式：XmlBeanFactory来获取容器类

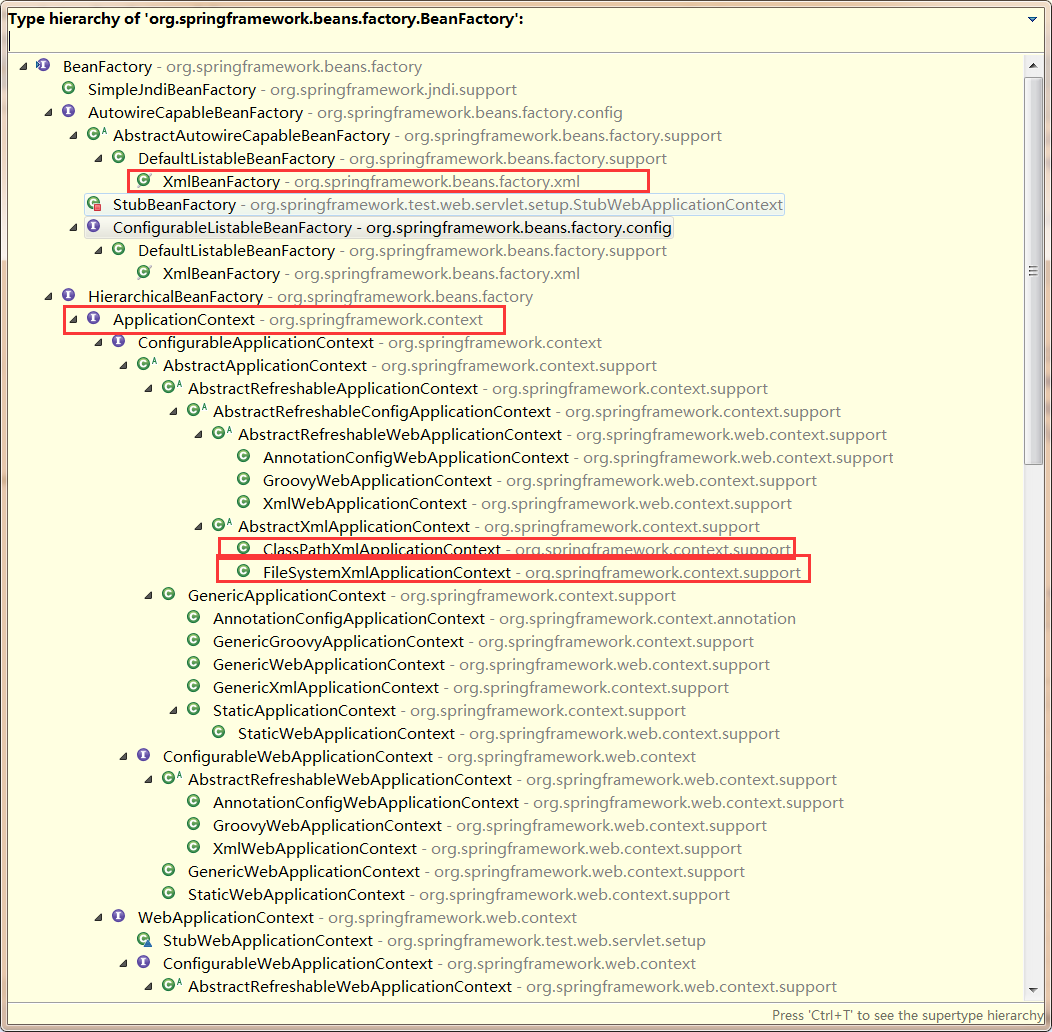
使用XmlBeanFactory来获取我们的beanFactory

创建applicationContext.xml

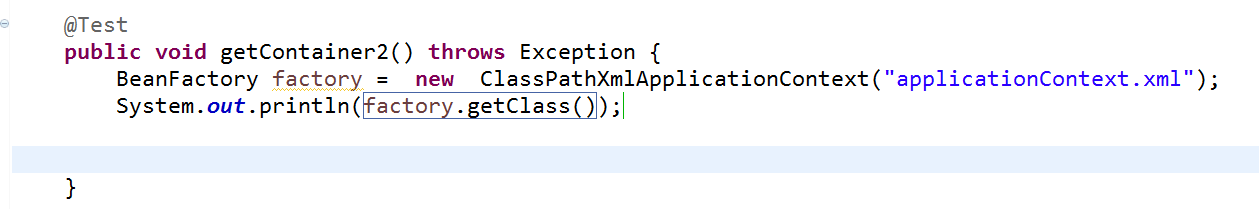


创建我们的BeanFactory工厂类





##### 第二种方式使用ClassPathXmlApplicationContext获取容器类



##### 第三种方式第三种方式，使用FileSystemXmlApplicationContext来获取容器



# 四：spring的第一大特性IOC

IOC：inversion of controller 控制反转

什么是控制反装：说白了就是将创建对象的过程或者说创建对象的权限交给了spring框架来帮我们处理，我们再不用通过new的方式来创建Javabean对象，这个过程就叫做控制反转。

DI：dependency injection 依赖注入。说白了就是使用spring框架为我们的JavaBean的属性赋值的过程

IOC与DI的区别：

1：ioc是为我们创建对象。Di是我们创建对象后的属性赋值

2：先有IOC，才能够使用DI

## 1：创建对象第一种方法，使用默认构造

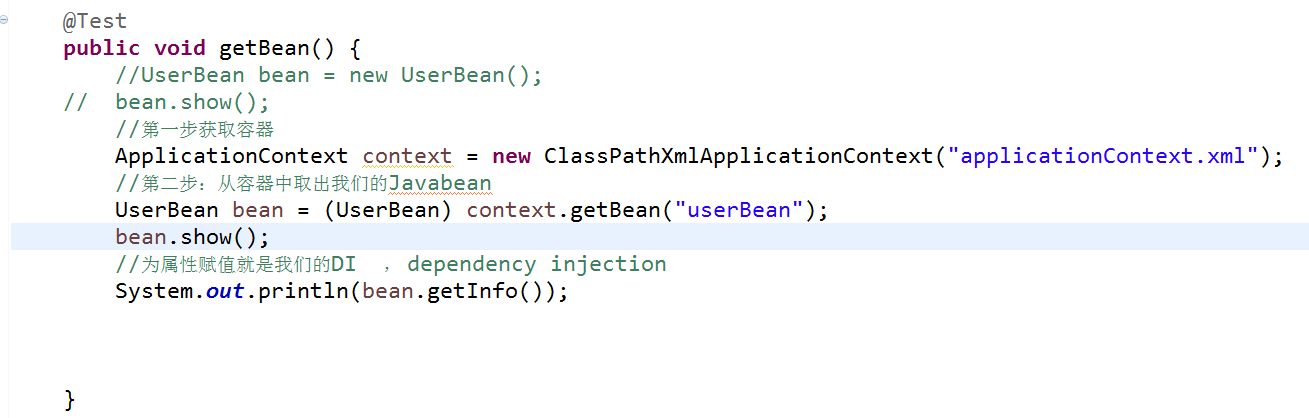
### 1.1：申明UserBean



### 1.2配置我们的UserBean

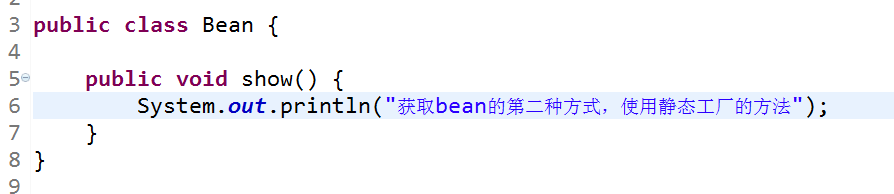


### 1.3从容器中获取我们的JavaBean对象



## 2：创建对象的第二种方式，使用静态工厂

### 2.1：创建JavaBean对象



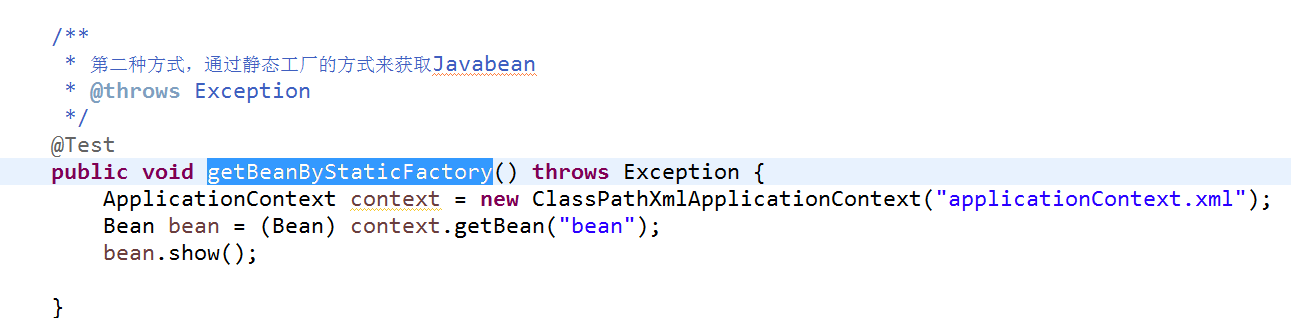
### 2.2：创建bean的工厂类



### 2.3配置我们的JavaBean

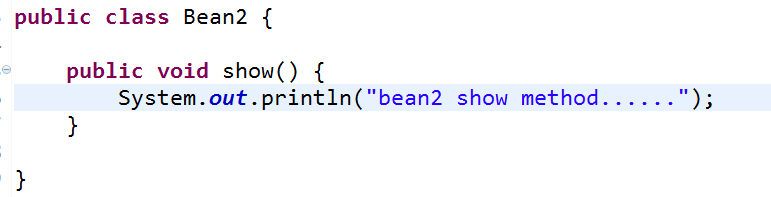


### 2.4获取我们的JavaBean

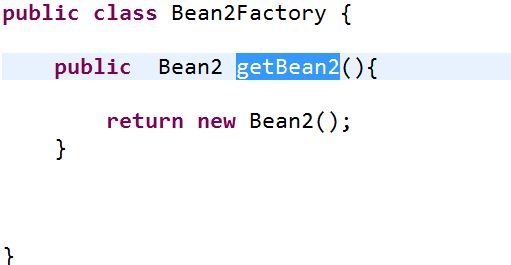


## 3：创建容器的第三种方式，使用实例工厂方式

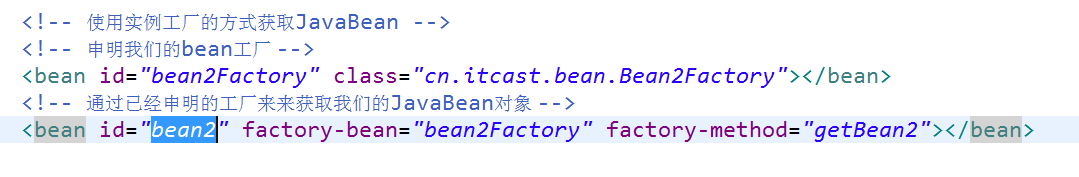
### 第一步：创建JavaBean对象



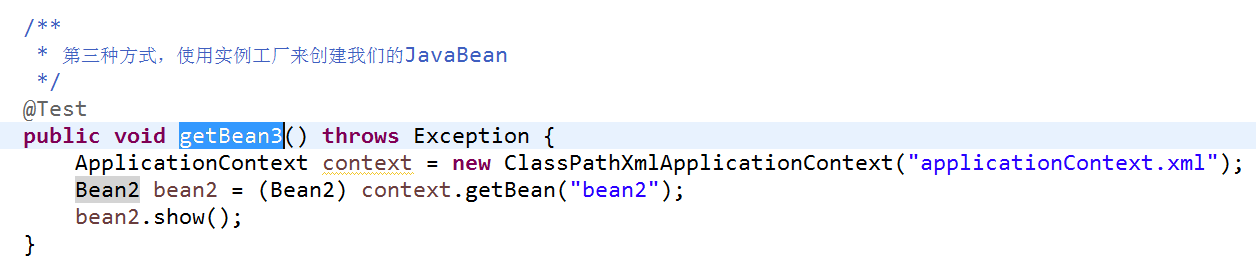
### 第二步：创建JavaBean对象的工厂类



### 第三步：配置我们的JavaBean



### 第四步：获取我们的JavaBean



## 4：Bean的作用域与作用范围

Bean的作用域范围有四种：

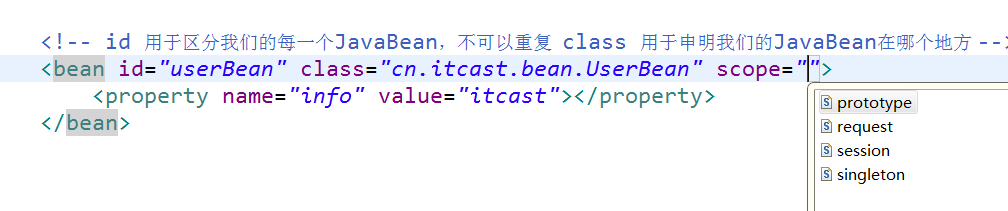
默认使用的是singleton这个范围。

Prototype：多例，每次调用都会创建一个新的实例

Singleton:单例，创建一个对象，每次调用，都使用这个对象

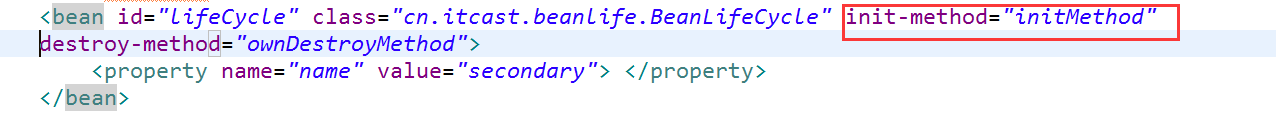
Request:适用于web开发当中，将我们的对象存储在request域中

Session: 适用于web开发当中，将我们的对象存储在session中



## 5：JavaBean的生命周期

可以配置init-method与destroy-method来实现bean的初始化和关闭的时候调用指定的方法

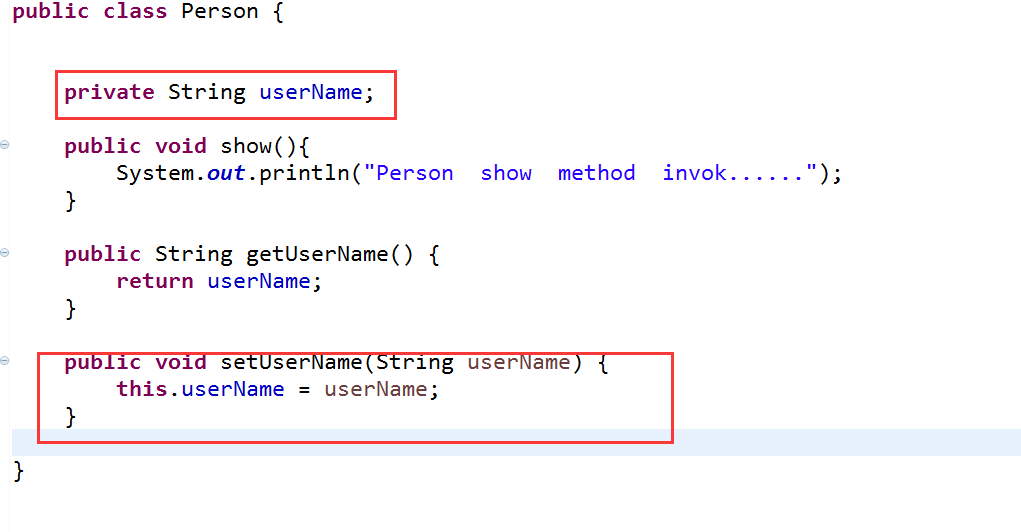


# 五：spring的属性注入之DI

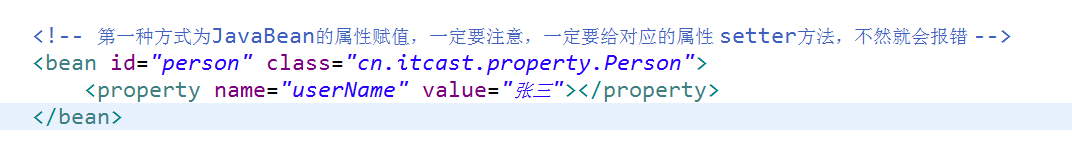
## 第一种方式：使用setter方法来进行赋值

### 第一步：创建JavaBean

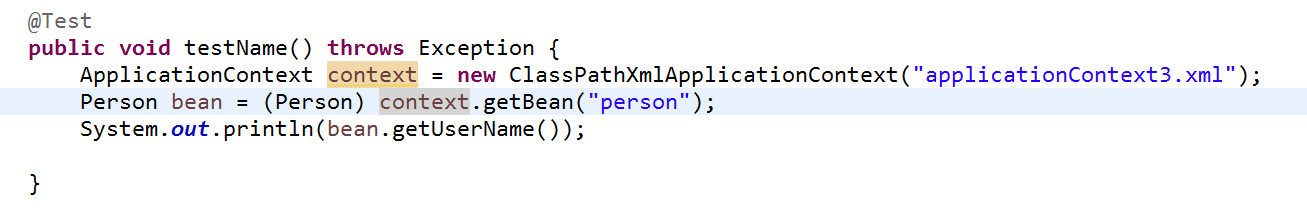
创建person类，并且给定属性userName，并且一定要为userName属性生成setter方法。



### 第二步：申明JavaBean



### 第三步：调用我们的JavaBean属性

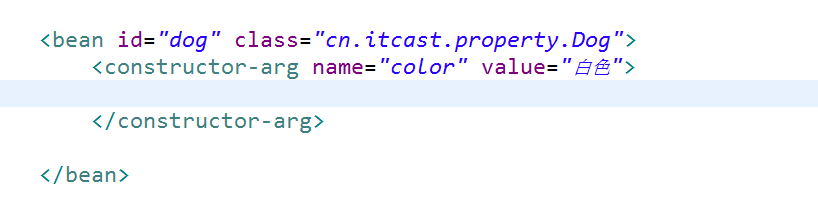


## 第二种方式：使用构造器来申明JavaBean

### 第一步：定义javaBean对象，并生成带所有属性的构造方法



### 第二步：申明JavaBean对象



Javabean申明的另外一种写法



### 第三步：调用JavaBean的属性



## P名称空间与C名称空间

P名称空间与C名称空间其实都是不存在的虚拟空间，主要是用于简化我们的spring为JavaBean属性赋值时的配置。

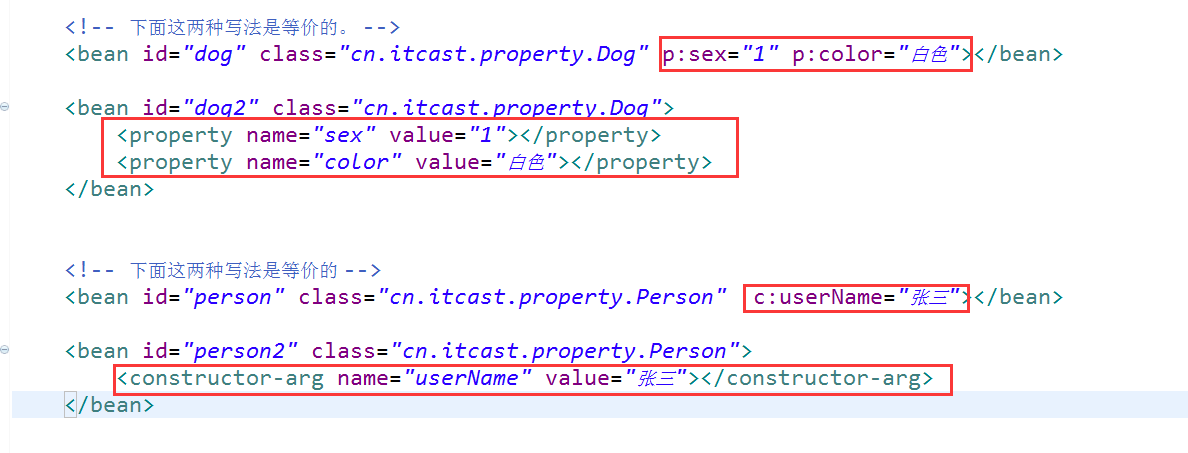
### 第一步添加P名称空间与C名称空间到schema

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:c=*"http://www.springframework.org/schema/c"*



### 第二步：使用P名称空间或者C名称空间来简化我们的开发



## SPEL表达式

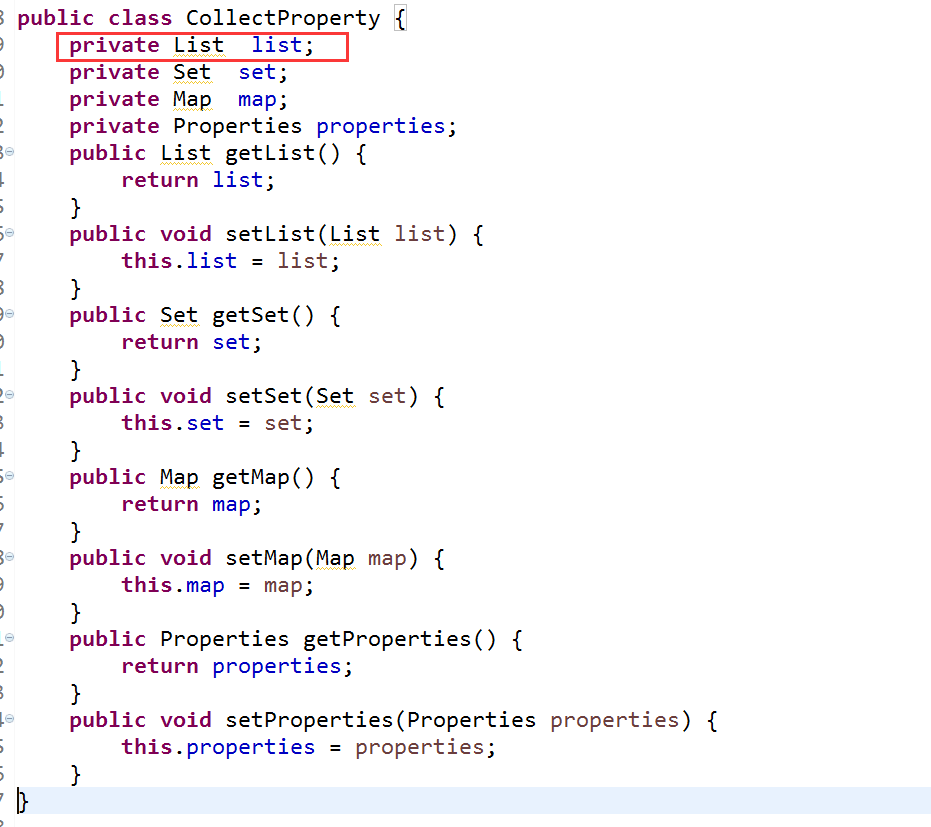
Spel：类似于jstl与el表达式的语言，spring可以支持我们在为属性赋值的时候，通过spel表达式来进行更改我们的属性值



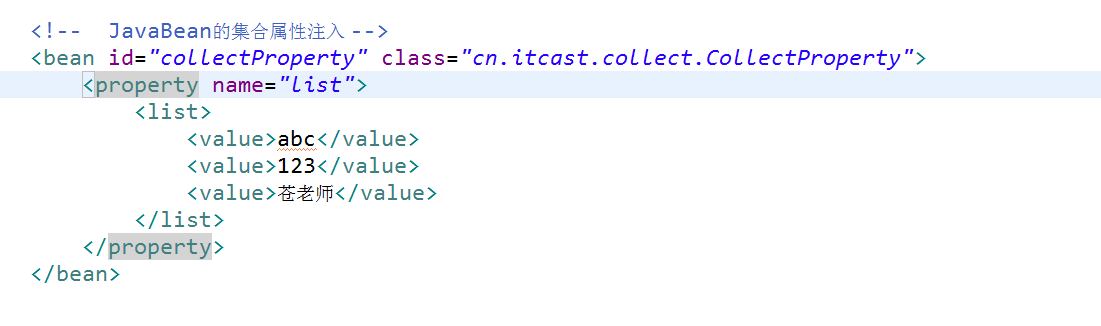
## 集合属性赋值

### 第一种：list集合属性赋值

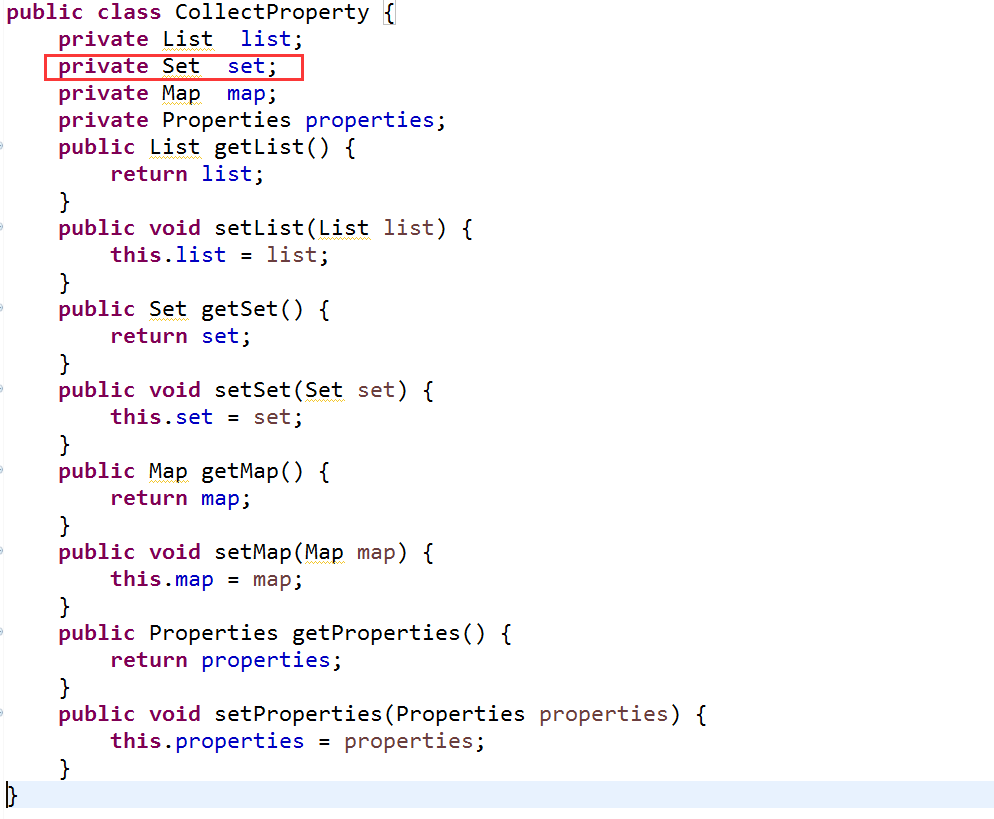
第一步：创建JavaBean对象，比设置list属性，而且list属性一定要给setter方法



第二步：申明JavaBean，并且为我们的list属性进行赋值



### 第二种：set集合属性赋值



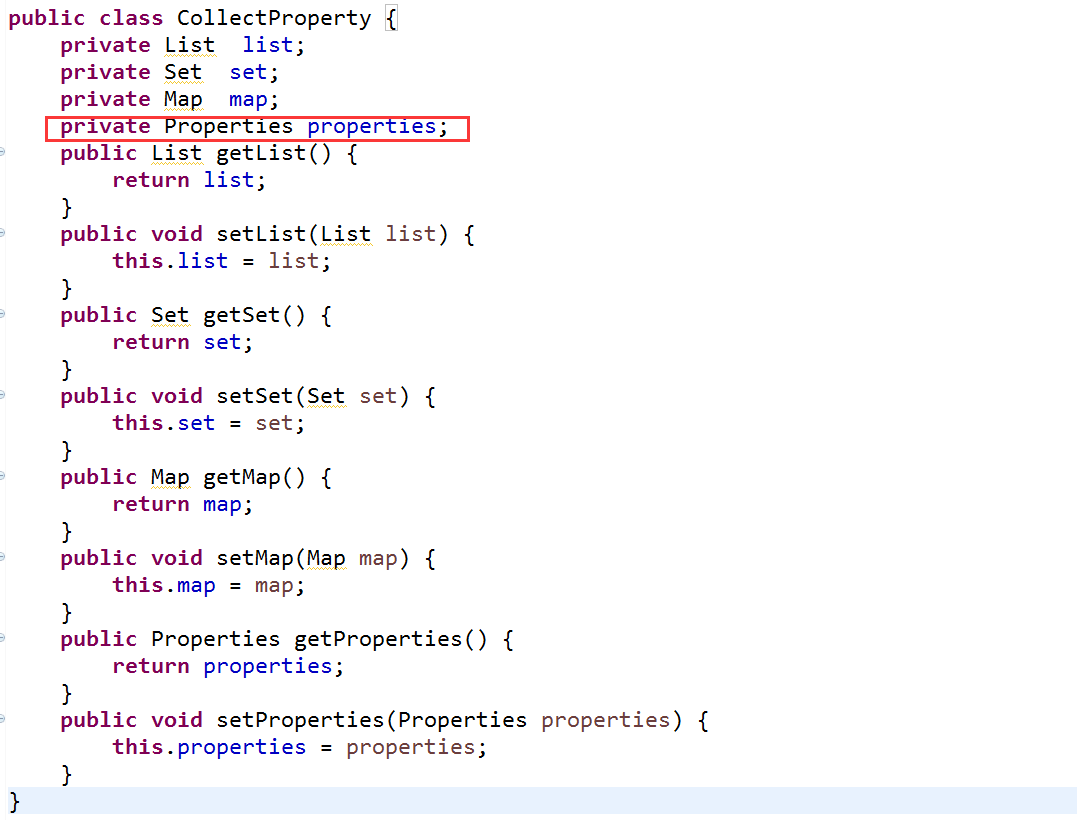


### 第三种：map集合属性赋值





### 第四种：properties属性赋值





# 六：spring与web整合

Spring为什么要与web项目进行整合：为了在我们项目一起动的时候，就加载我们的spring容器。

现在的需求，在我们的http请求中得到我们的spring容器。

第一步：导入spring与web整合的对应版本的jar包

<!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

第二步：配置监听器，监听我们的web项目的

在我们的web.xml中添加监听器

<!-- needed for ContextLoaderListener -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

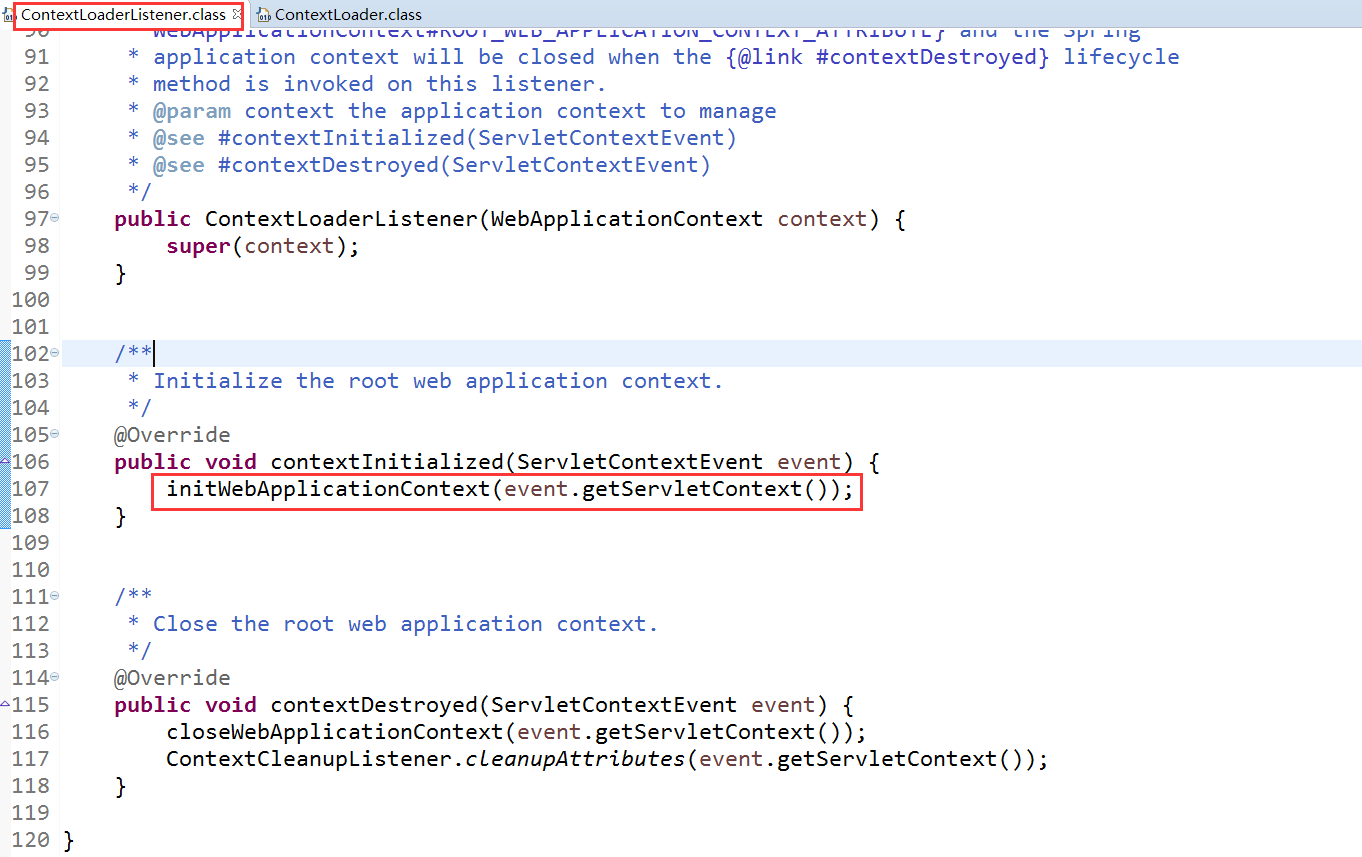
<!-- Bootstraps the root web application context before servlet initialization -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

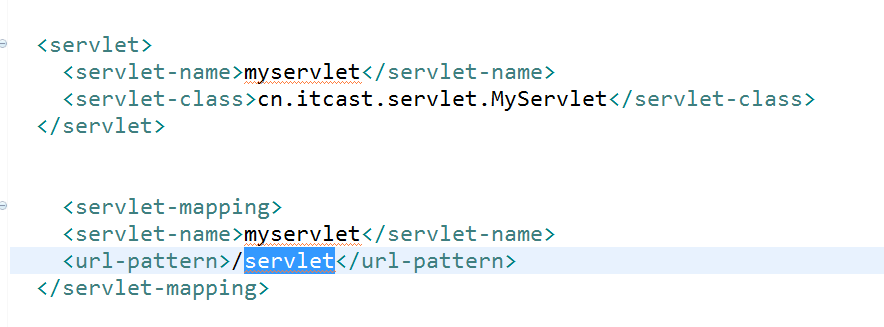
</listener>

第三步：定义自己的servlet，在servlet请求中获取spring容器





第四步：设置我们的servlet访问路径



第五步：页面访问，<http://localhost:8080/spring_day01/servlet>

断点可以看得到我们的spring容器已经加载了。

# 七：spring注解

如果需要使用spring的注解，需要spring-aop的jar包

### 第一步：导入jar包

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency>

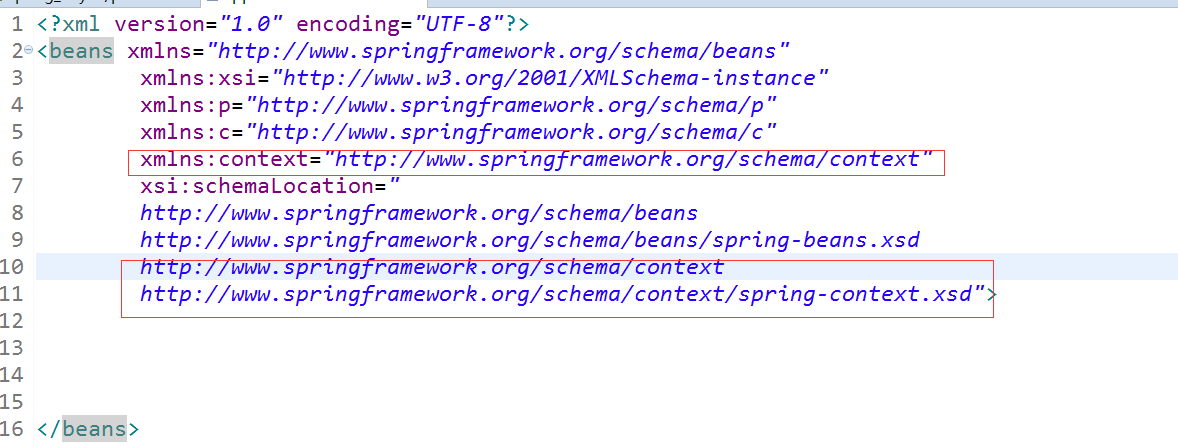
<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

### 第二步：配置schema



### 第三步：开启注解

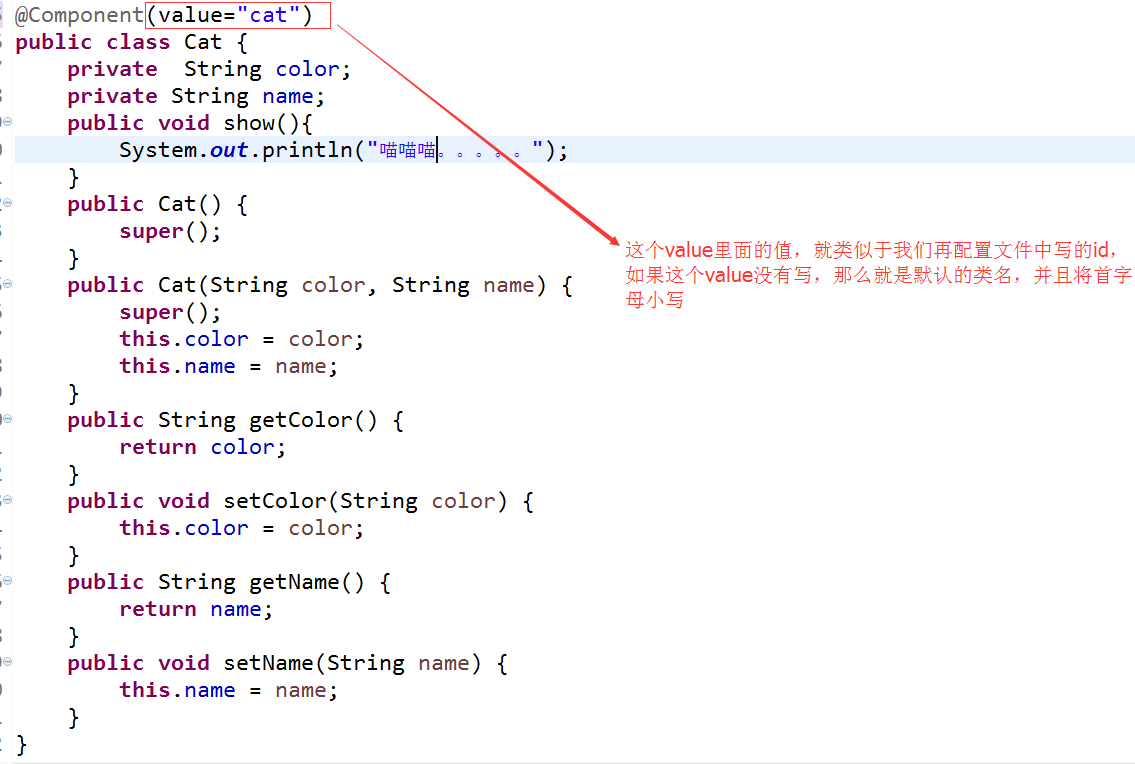
<!-- 配置通过注解的方式来使用我们的开发 -->

<context:annotation-config/>

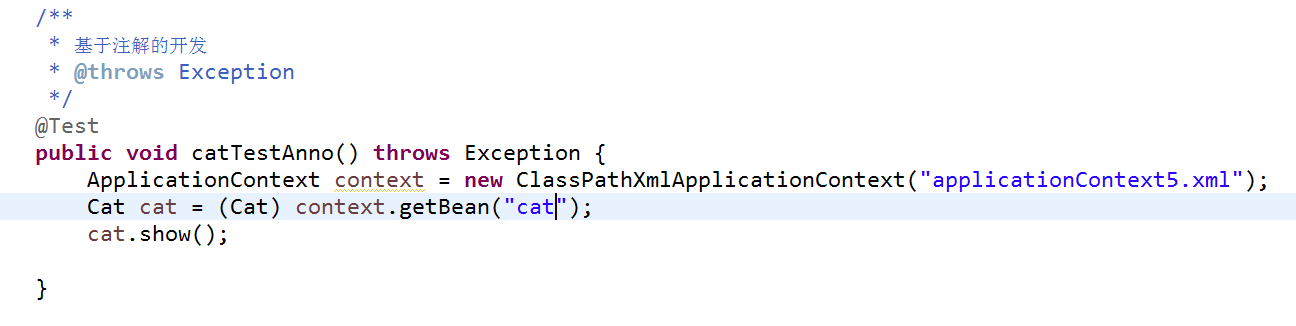
<!-- 配置注解扫描的包 -->

<context:component-scan base-package=*"cn.itcast.spring"*/>

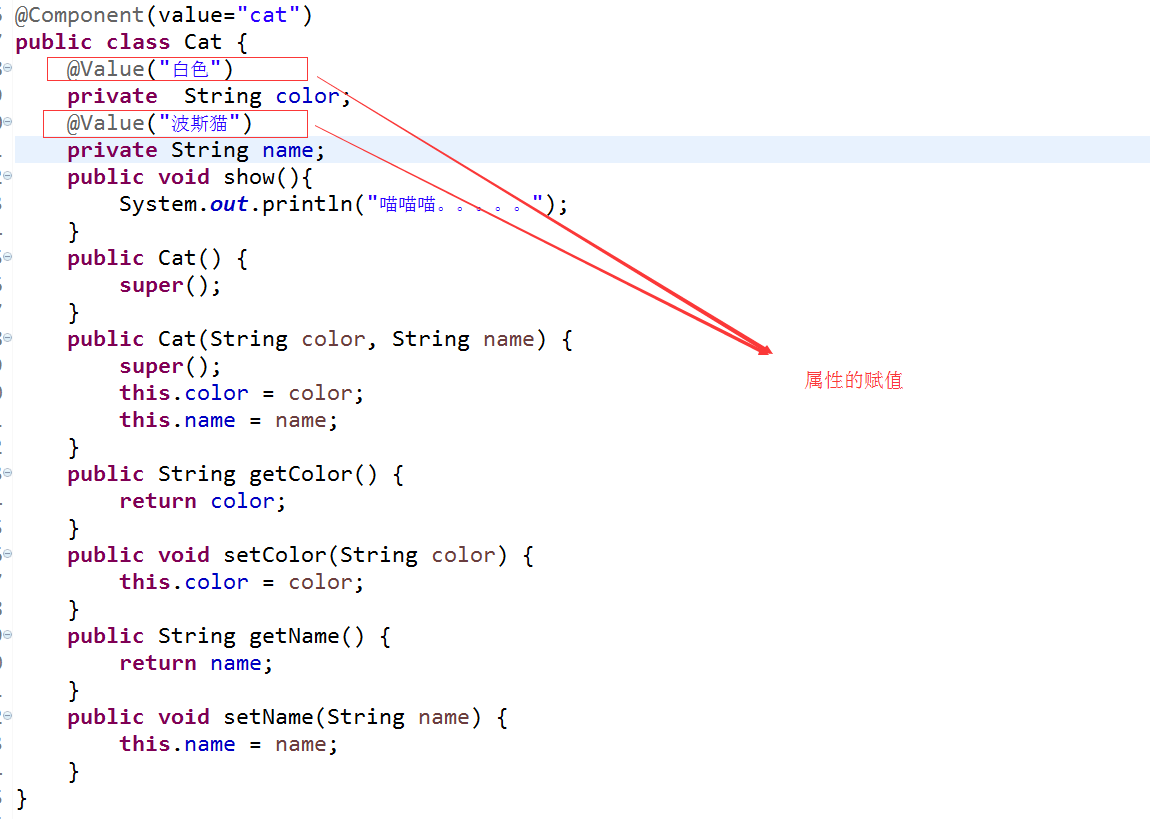
### 第四步：定义我们的JavaBean对象



### 第五步：调用我们的JavaBean对象

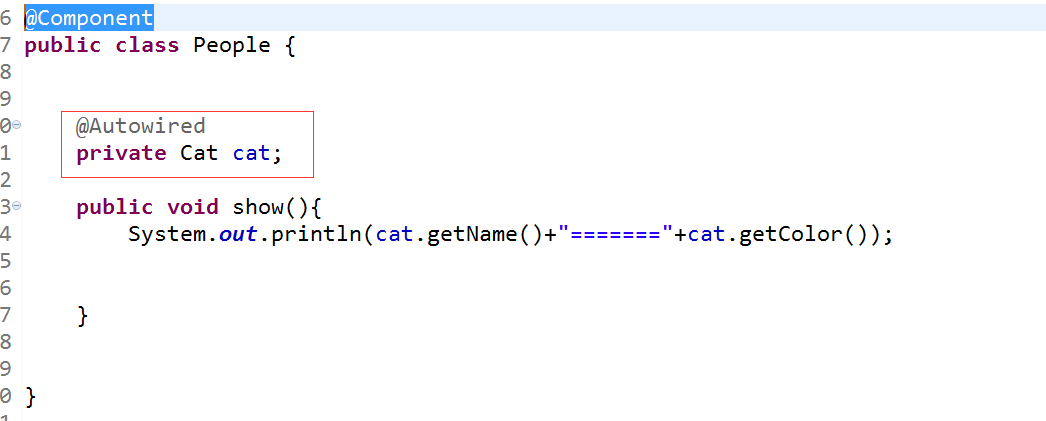


### JavaBean属性使用注解的方式来进行赋值

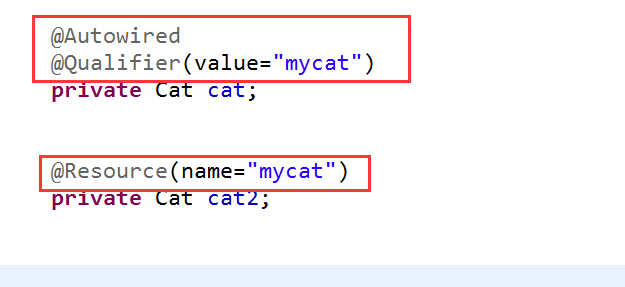


在注解开发中，如果是复杂类型的注入，一般是用

@Autowired



在实际开发中我们的注解 @Resource(name="mycat") 等价于@Autowired 加上 @Qualifier(value="mycat")



在我们的web开发当中，spring为了分层表示我们的开发架构，还衍生出了三个相同的注解，只不过用于不同层级的标识

@Controller :用于数据展现曾

@Service ：用于业务逻辑层

@Repository ：用于数据控制层

# 八：spring与junit的整合

## 第一步：导入相应的jar包

<!-- https://mvnrepository.com/artifact/org.springframework/spring-test -->

<dependency>

<groupId>org.springframework</groupId>

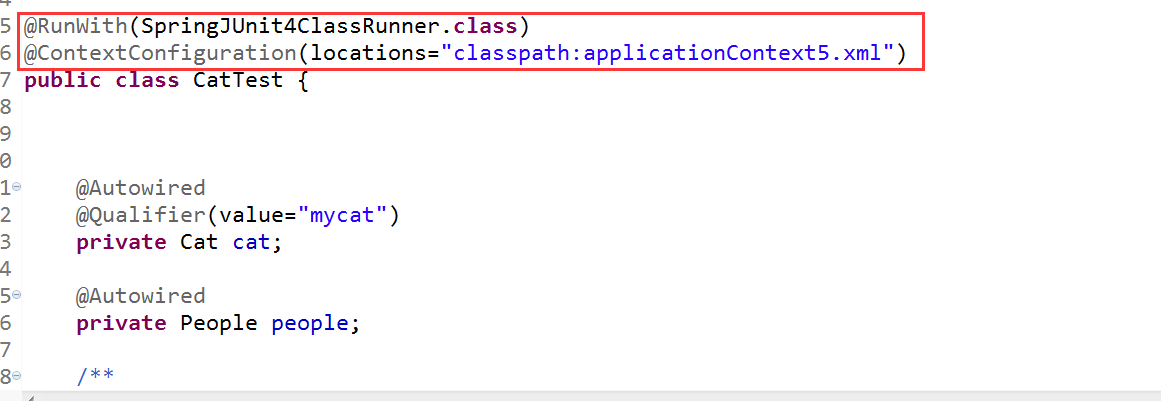
<artifactId>spring-test</artifactId>

<version>4.2.4.RELEASE</version>

<scope>test</scope>

</dependency>

第二步：在我们的测试类上添加两个注解，并且指定我们的配置文件所在的位置



# Spring第二天：aop

# Aop的概念介绍

Aop：aspect oriented programing 面向切面的编程

AOP的作用：

例如

权限校验

性能检测

日志记录

事物管理

AOP底层实现原理

使用代理的方式来实现

# Aop底层的两种实现方式

在spring底层使用了两种方式的代理来实现我们的增强功能

Jdk的代理方式：

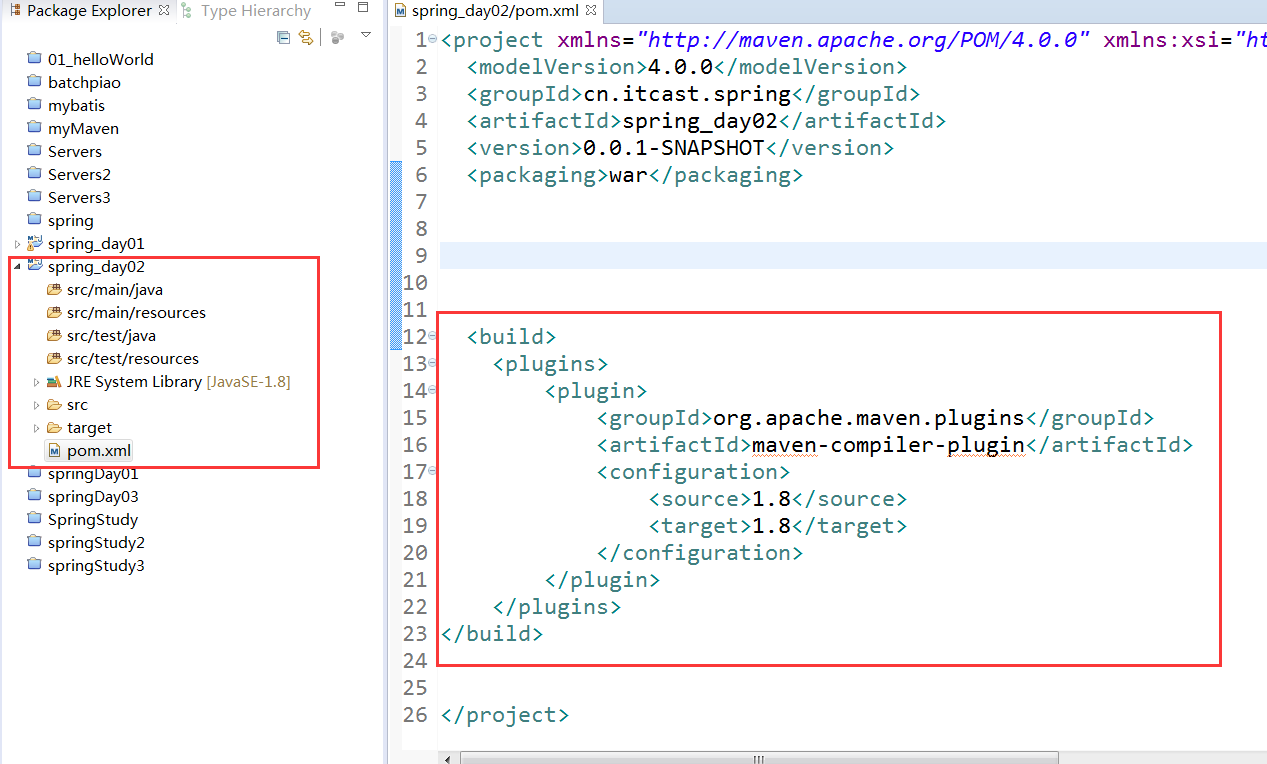
只能对接口进行增强

Cglib

可以对没有接口的类进行增强

代码实现

## 1、创建web maven工程



解决jdk版本问题

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>utf-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

解决web.xml丢失问题

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<display-name>01\_helloWorld</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

## 2、导入相关的jar包

除了spring第一天所有必须的依赖包之外还需要以下这些依赖jar包

<!-- https://mvnrepository.com/artifact/aopalliance/aopalliance -->

<dependency>

<groupId>aopalliance</groupId>

<artifactId>aopalliance</artifactId>

<version>1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.6.8</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aspects -->

<dependency>

<groupId>org.springframework</groupId>

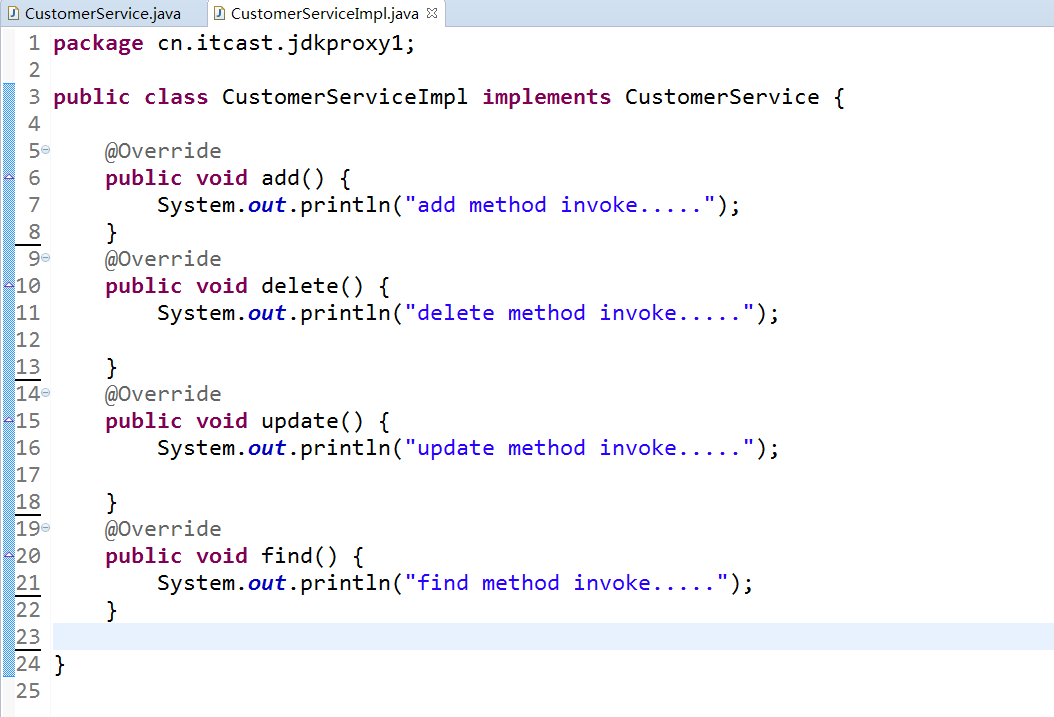
<artifactId>spring-aspects</artifactId>

<version>4.2.4.RELEASE</version>

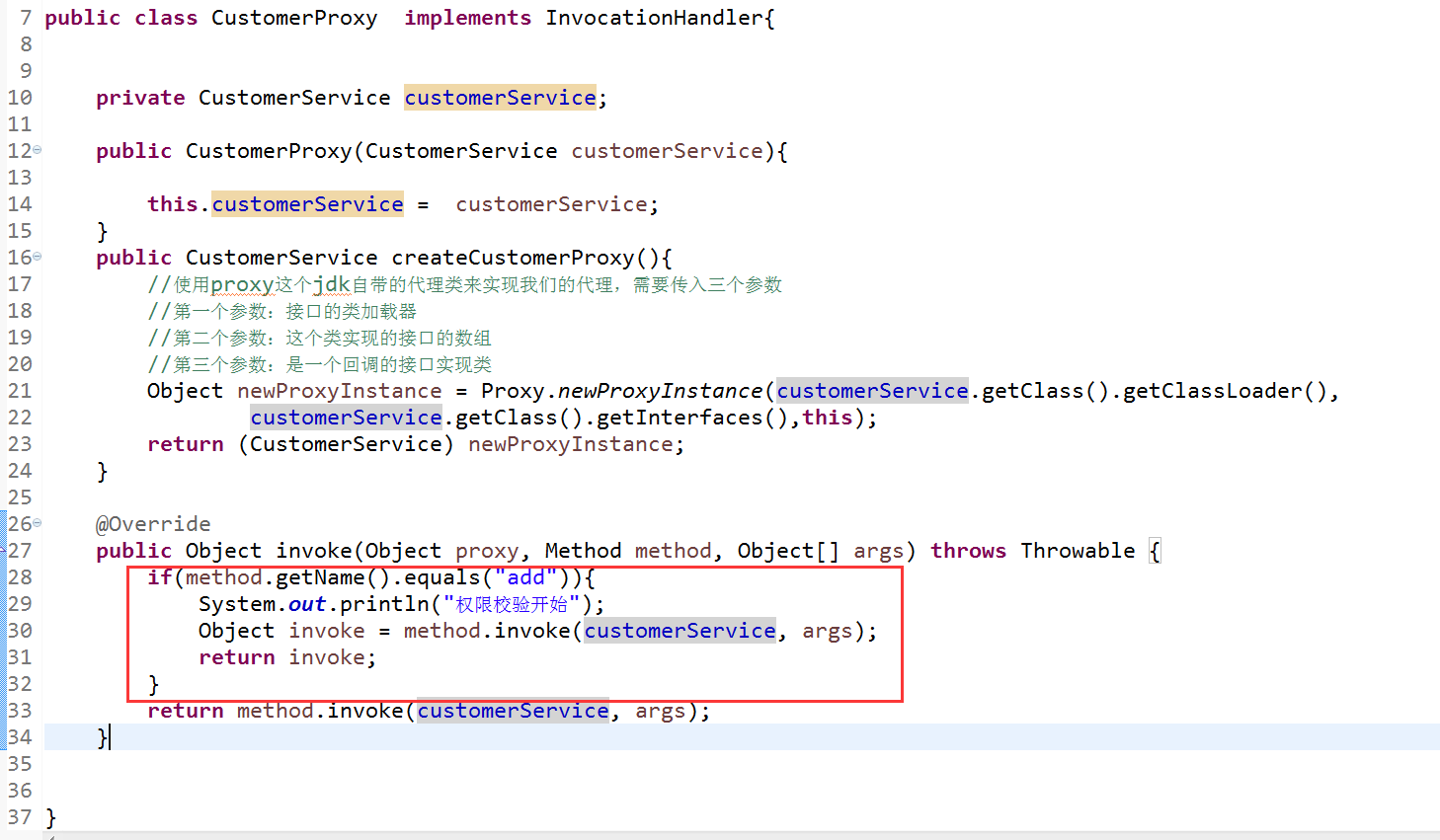
</dependency>

## 3、实现jdk的代理方式来进行增强

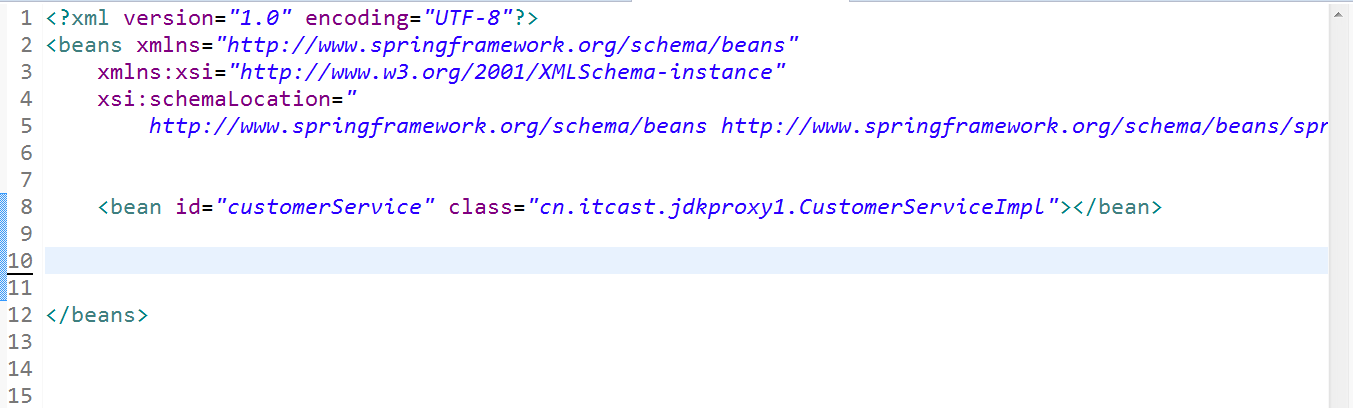
### 第一步：创建接口和实现类



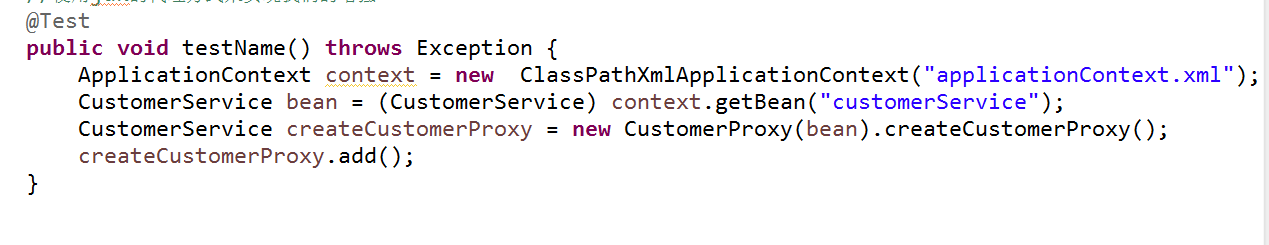
### 第二步：创建我们的代理类



### 第三步：配置spring容器

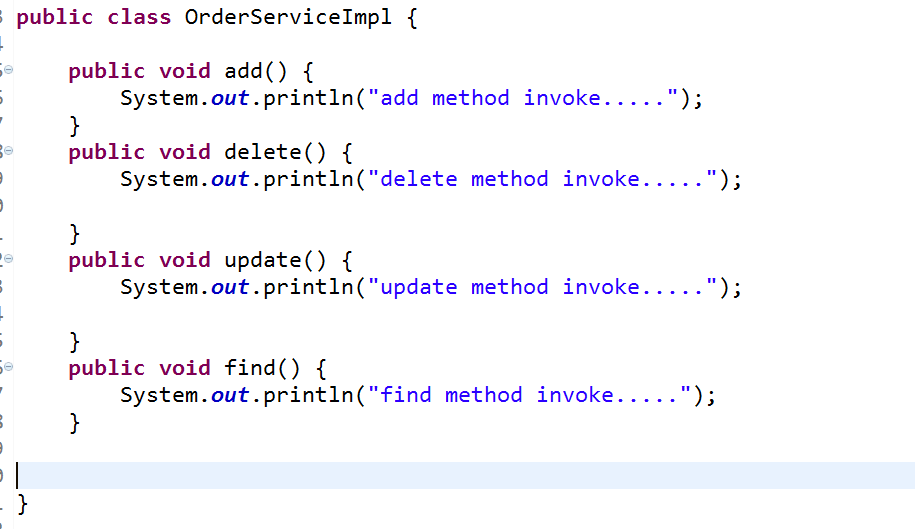


### 第四步：通过调用代理来进行增强的实现

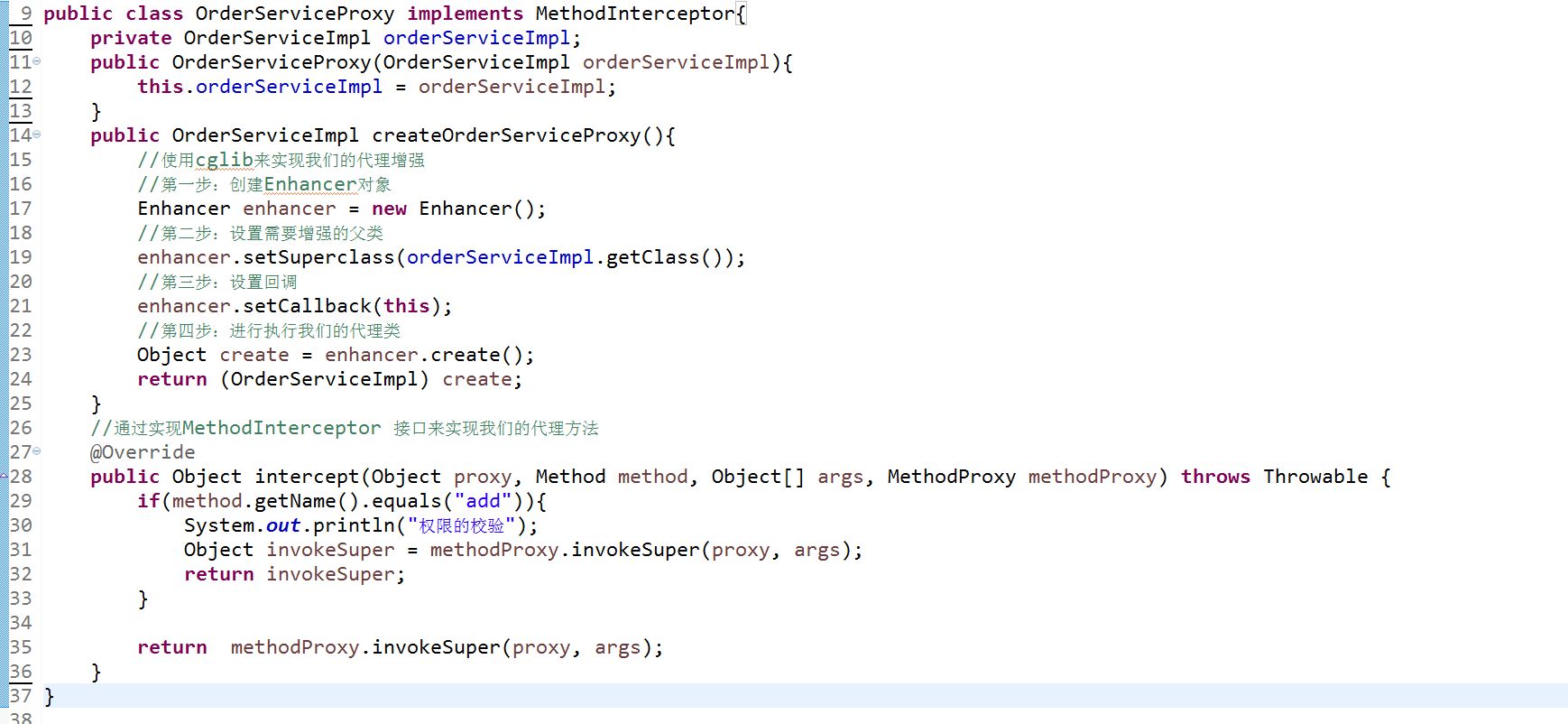


## 4、通过Cglib的方式来进行代理

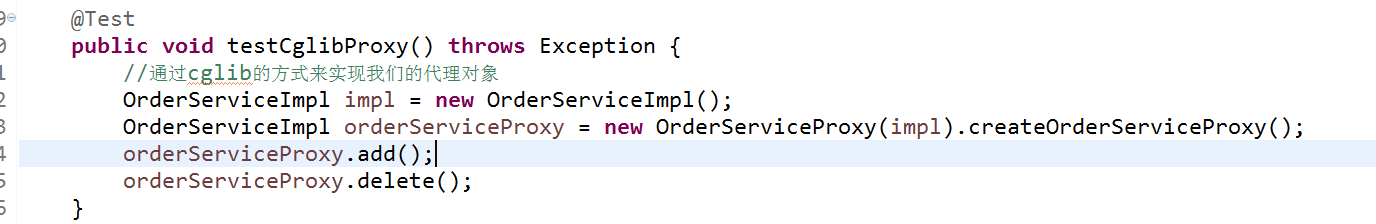
### 第一步：创建实现类，不需要接口



### 第二步：创建我们的代理类

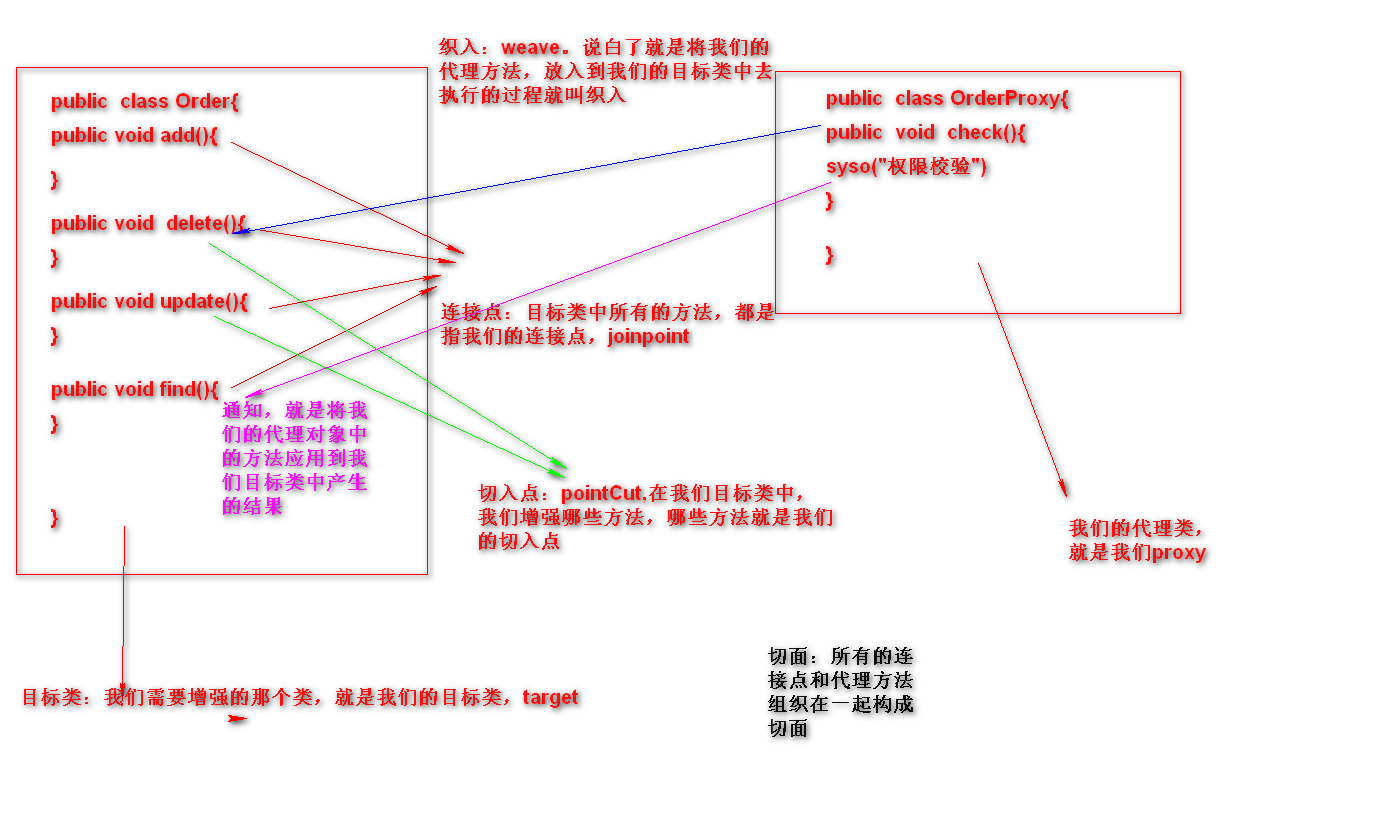


### 第三步：通过代理类来调用我们的目标类的方法进行增强



# aop当中的一些专业术语定义

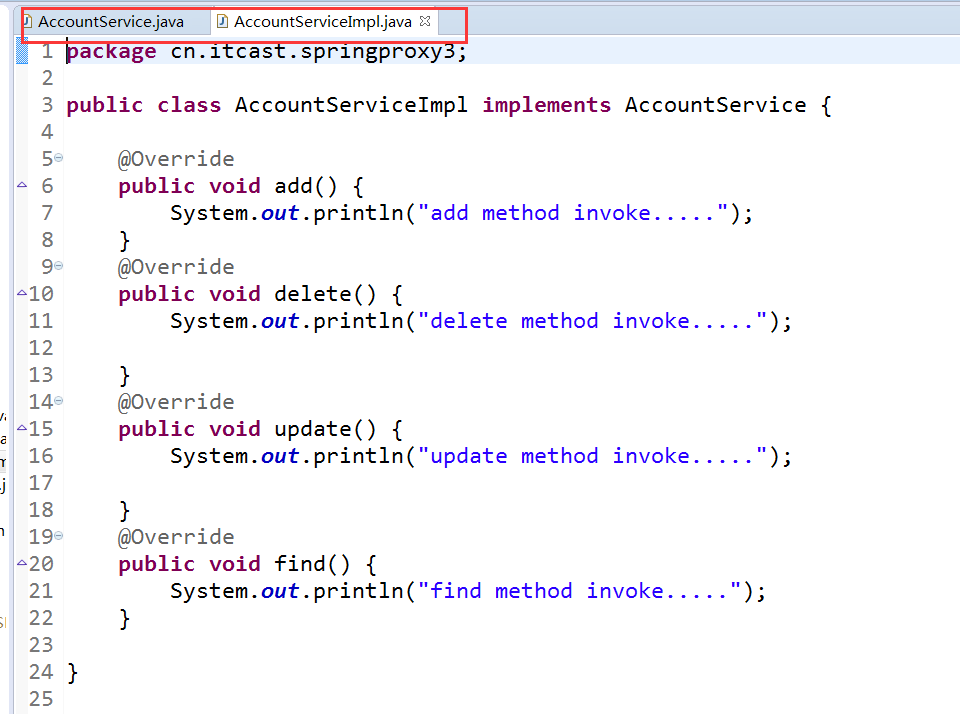
1. 目标类target：就是我们需要增强的那个累，target
2. 代理类proxy：就是我们自定义的那个代理的对象proxy
3. 连接点joinPoint：连接点说白了就是我们的目标类里面所有的方法。
4. 切入点pointCut：切入点说白了就是那些在目标类中我们实际上增强的方法。
5. 织入weave：说白了就是将我们的代理类中需要增强的方法放入到目标类中去执行的过程，就叫织入
6. 引介Introduction：是对我们的类中的方法或者属性进行一些创建的过程
7. 通知advice：说白了就是将我们代理对象中的方法应用到目标类的过程中产生的结果。
8. 切面aspect：说白了就是我们的所有的连接点和代理对象的方法组成在一起 构成了我们的切面



# Aop基于xml配置的方式实现

## Spring基于xml开发AOP

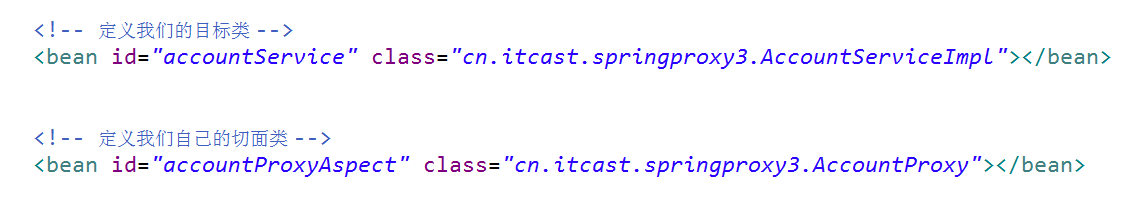
### 第一步：申明接口类和实现类。也就是我们的目标类



### 第二步：申明我们的切面类



### 第三步：在spirng容器中申明我们的目标类和切面类



### 第四步：配置spring的aop schema

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

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

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*

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

<bean id=*"customerService"* class=*"cn.itcast.jdkproxy1.CustomerServiceImpl"*></bean>

<!-- 定义我们的目标类 -->

<bean id=*"accountService"* class=*"cn.itcast.springproxy3.AccountServiceImpl"*></bean>

<!-- 定义我们自己的切面类 -->

<bean id=*"accountProxyAspect"* class=*"cn.itcast.springproxy3.AccountProxy"*></bean>

</beans>



### 第五步：通过applicationContext.xml来配置我们的切入点和织入的过程

<aop:config >

<!-- 这里的expression就是我们的切入点表示式，表示的是我们要对哪些类的哪些方法进行增强 -->

<aop:pointcut expression=*"execution(\* cn.itcast.\*.add\*(..))"* id=*"pointCut1"*/>

<!-- 通过我们的aop:aspect来定义我们的切面类 -->

<aop:aspect ref=*"accountProxyAspect"*>

<!-- 这里的method指的是我们代理对象中的哪个方法 -->

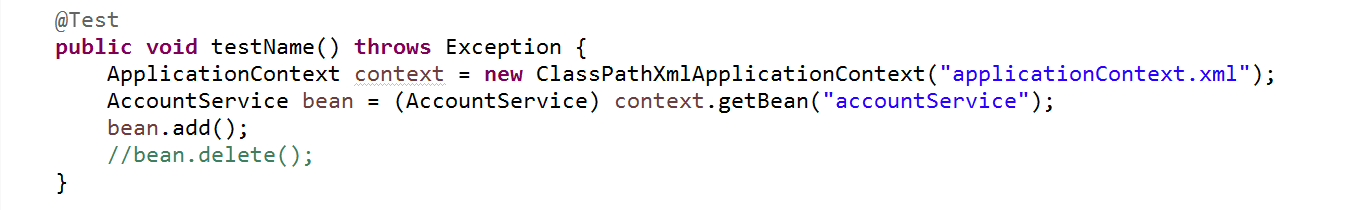
<aop:before method=*"checkPrivilege"*  pointcut-ref=*"pointCut1"*/>

</aop:aspect>

</aop:config>

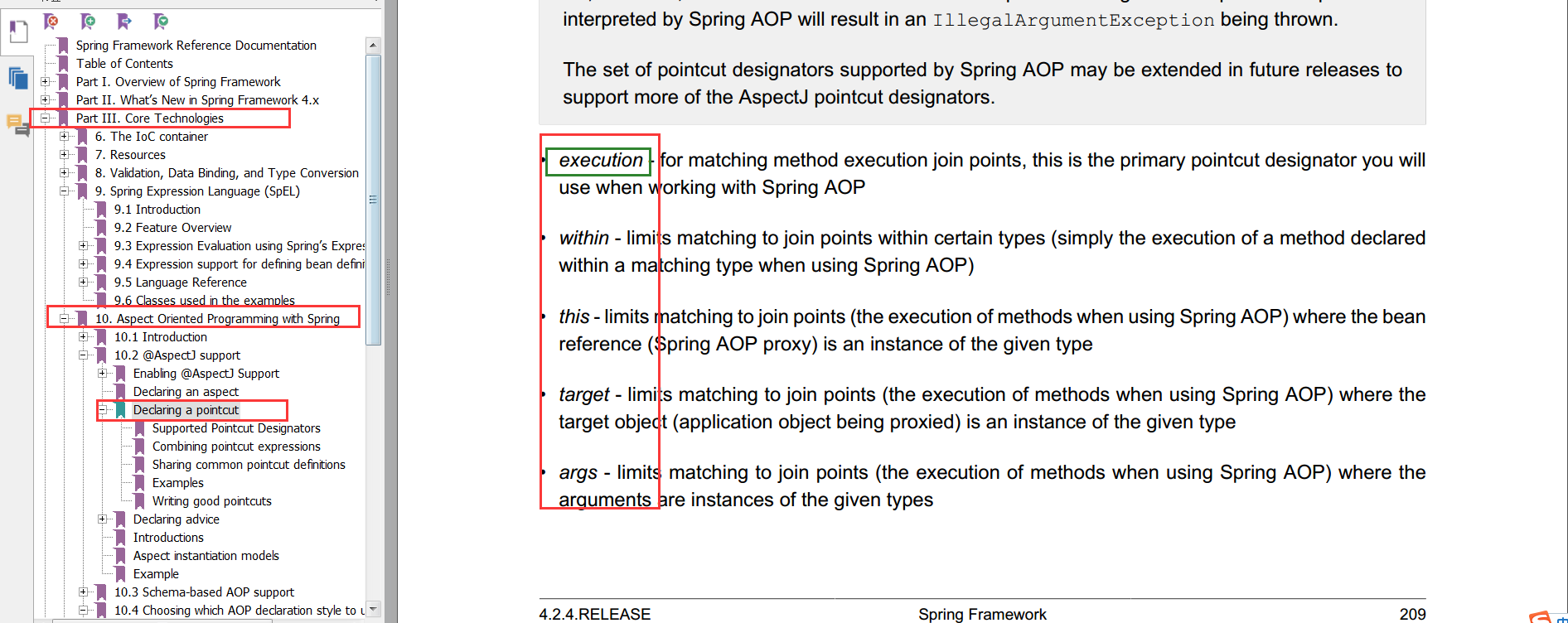


### 第六步：通过测试来实现我们的增强



## AOP的切入点表达式execution

关于切入点表达式，我们主要用的就是execution，pdf文档213页



Execution的写法

修饰关键词 返回值类型 包名.类名.方法名(..)

修饰关键词：protected , public , private ….一般都省略不写

返回值类型：一般返回值类型用\* 号代替，表示任意的返回值都可以

方法名中的两个点：表示所带的参数的个数。

1：任意public开头的任意方法都可以

execution(**public** \* \*(..))

2.任何以set开头的方法都可以

execution(\* set\*(..))

3.任何在AccountService 这个接口下面的方法都可以

execution(\* com.xyz.service.AccountService.\*(..))

1. 任何在com.xyz.service 这个包下面的方法都可以

execution(\* com.xyz.service.\*.\*(..))

6．任何在com.xyz.service 这个包或者这个包下一级的包路径下的方法都可以

execution(\* com.xyz.service..\*.\*(..))

## 3、spring当中的通知类型

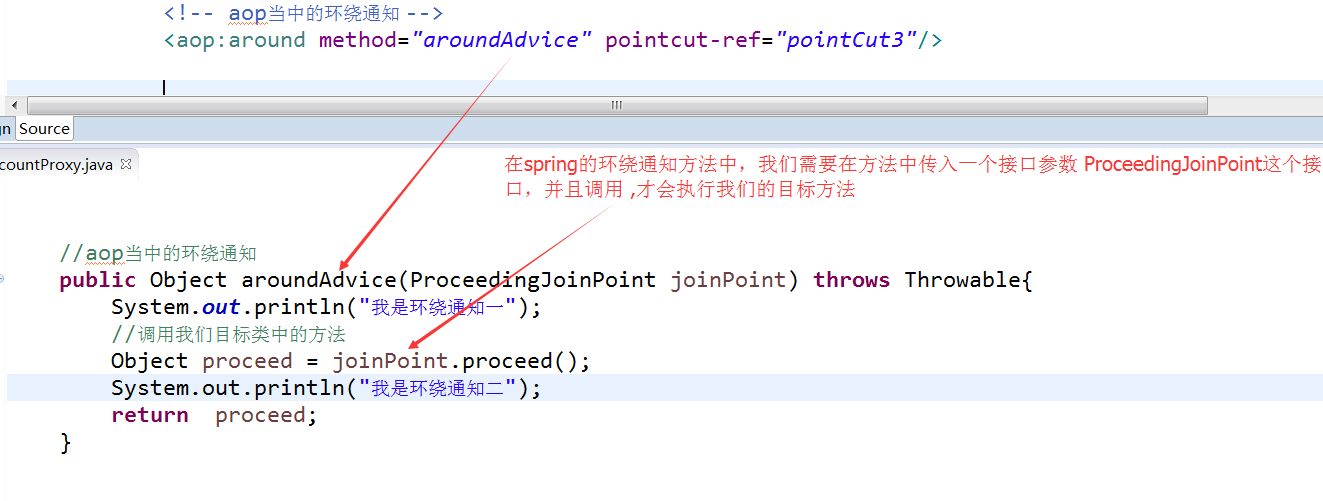
### 第一种：前置通知



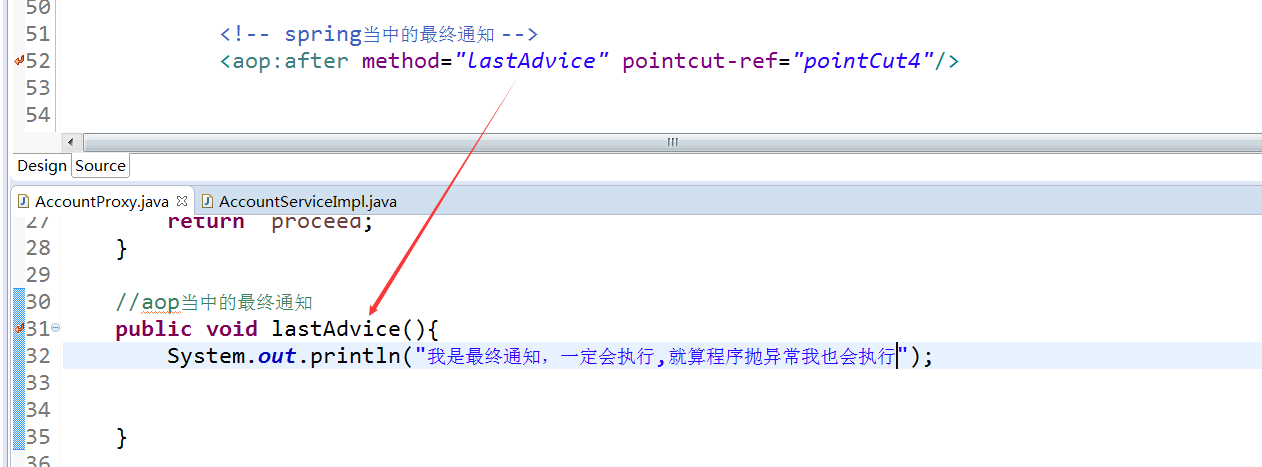
### 第二种：后置通知



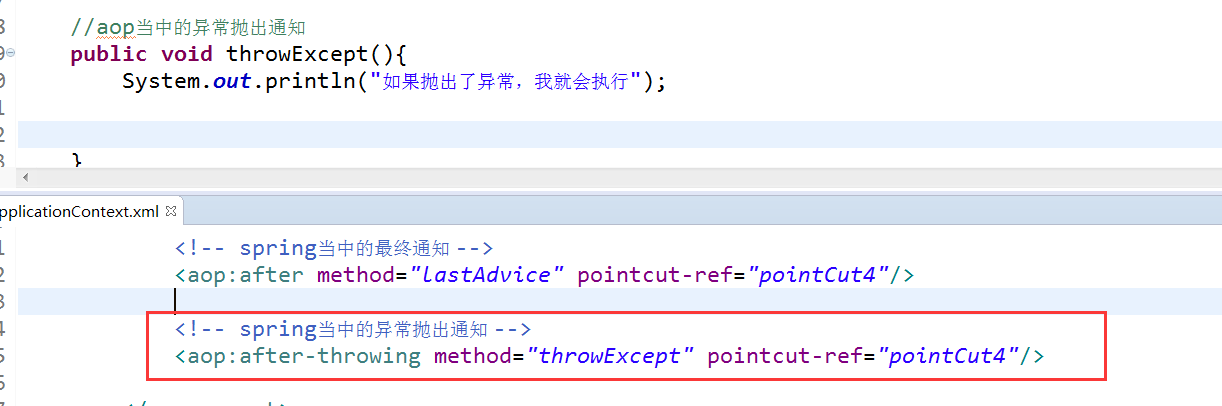
### 第三种：环绕通知



### 第四种：最终通知

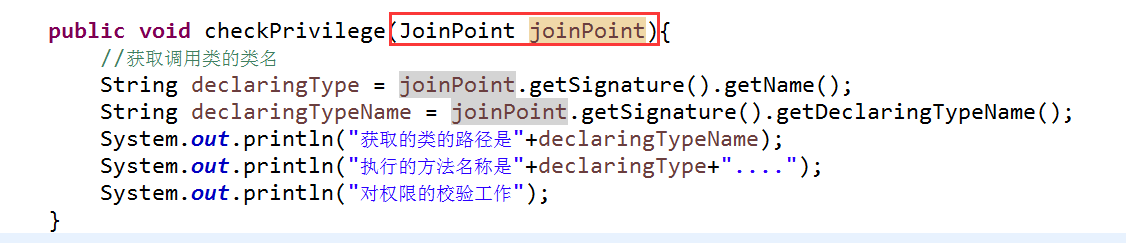


### 第五种：异常抛出通知

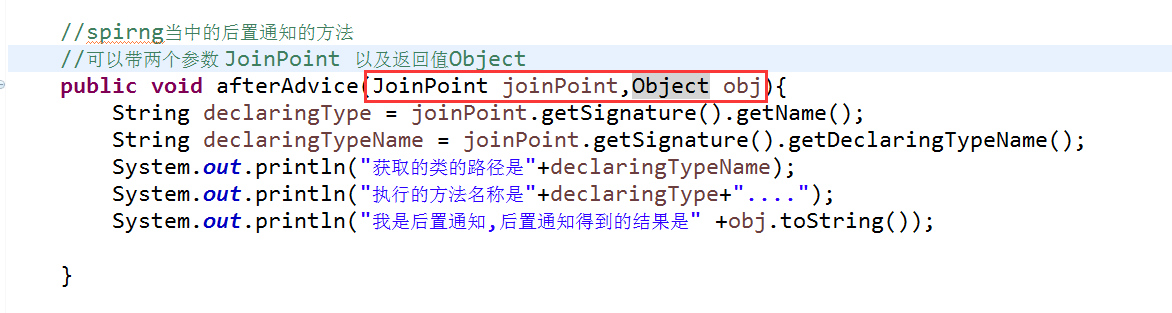


## 4、通知类型上面的参数定义

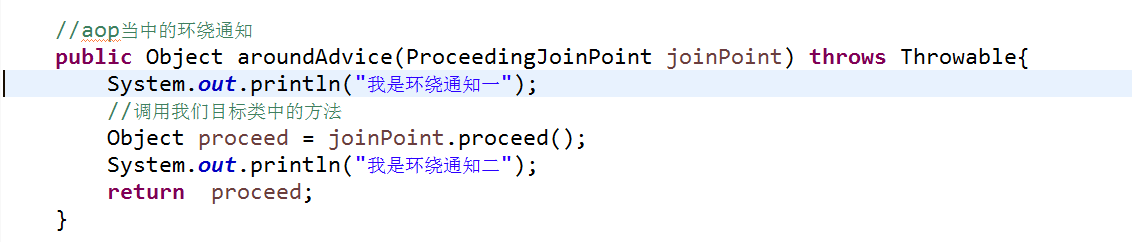
### 一：前置通知可以带的参数JoinPoint



### 二：后置通知可以带的参数JopiPoint和Object



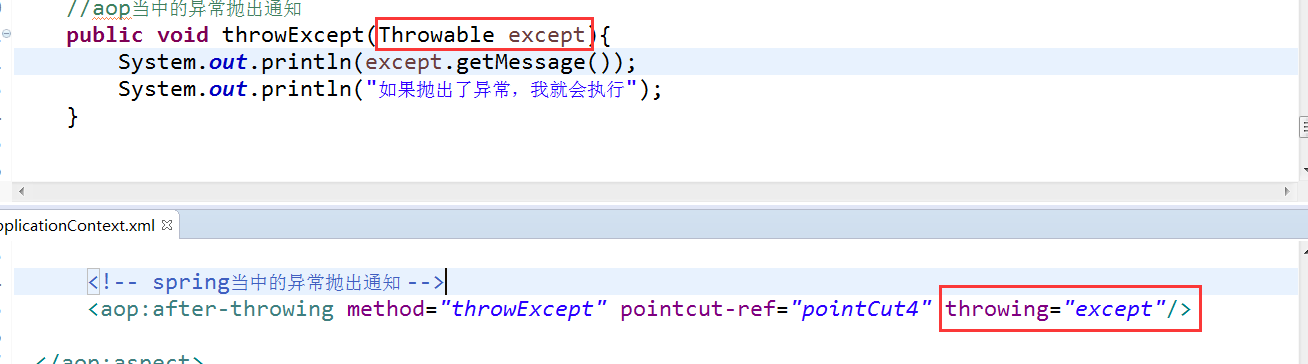
### 三：环绕通知可以带的参数ProceedingJoinPoint



### 四：最终通知可以带的参数JoinPoint

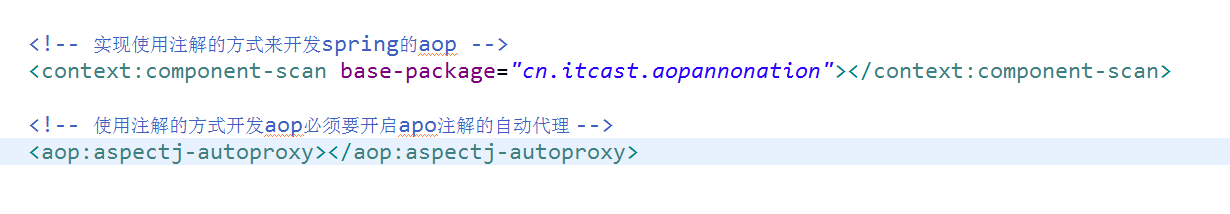


### 五：异常通知可带的参数Throwable

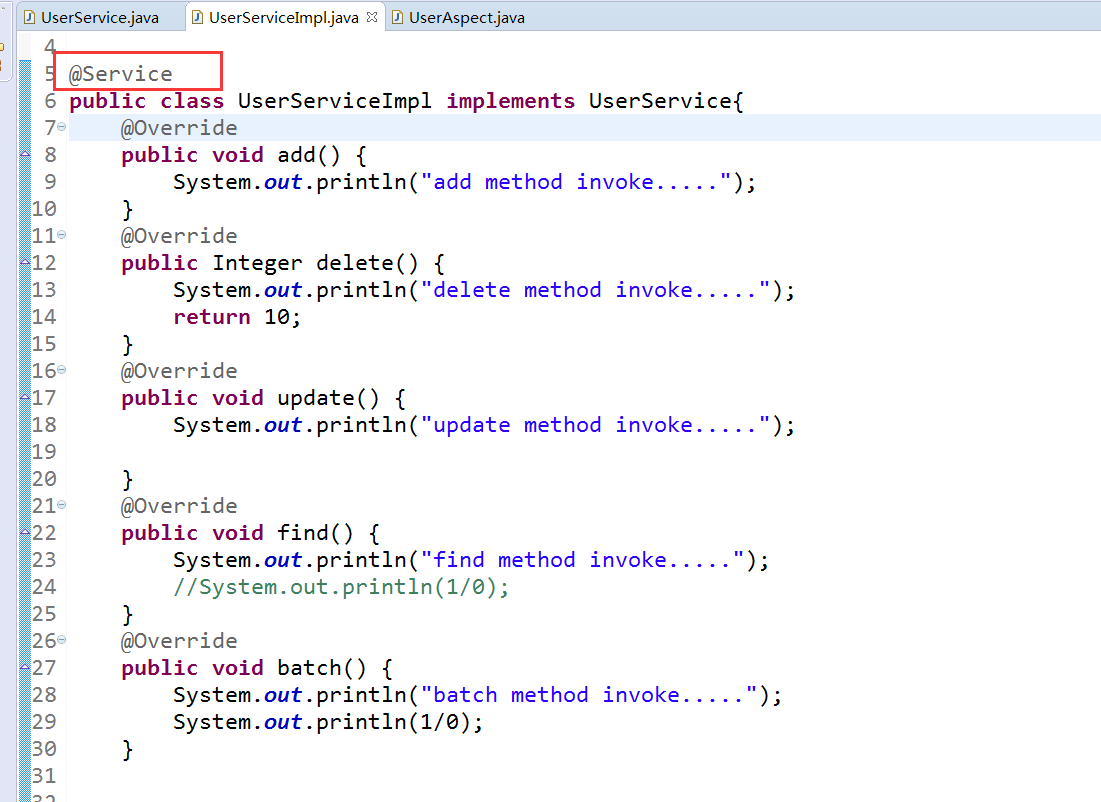


# 5、Aop基于注解的方式实现

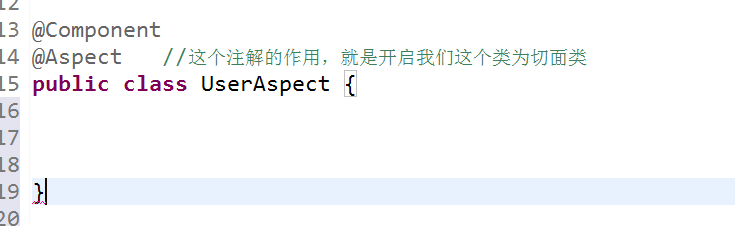
## 第一步：开启spring的注解功能以及开启使用代理的方式来进行aop的开发



## 第二步：定义我们的目标类



## 第三步：定义我们的切面类

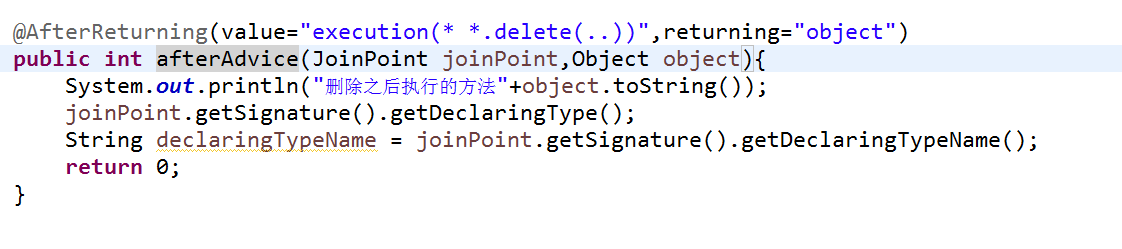


## 第四步：通过注解的方式来定义我们的五种通知

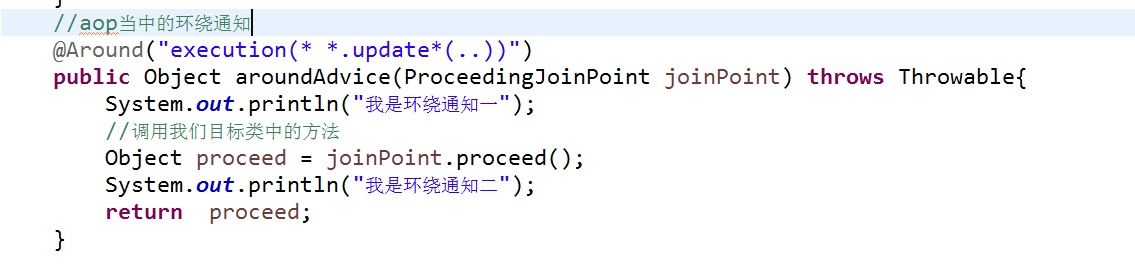
### 第一种通知：前置通知



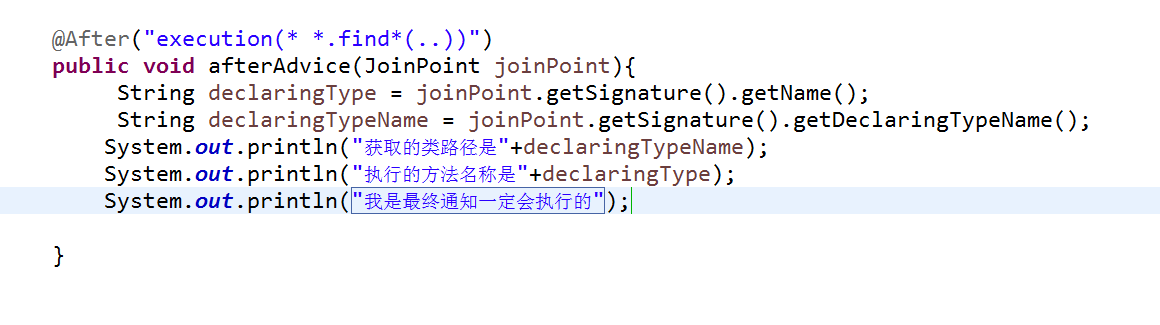
### 第二种通知：后置通知



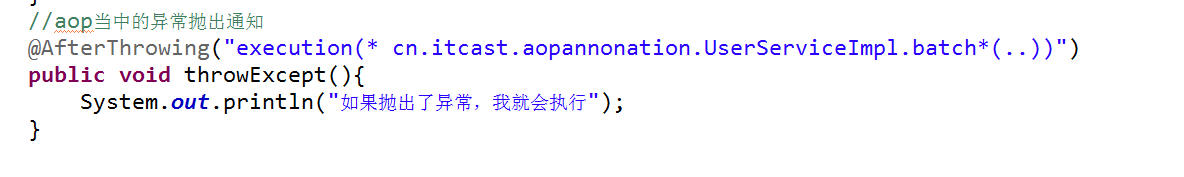
### 第三种通知：环绕通知



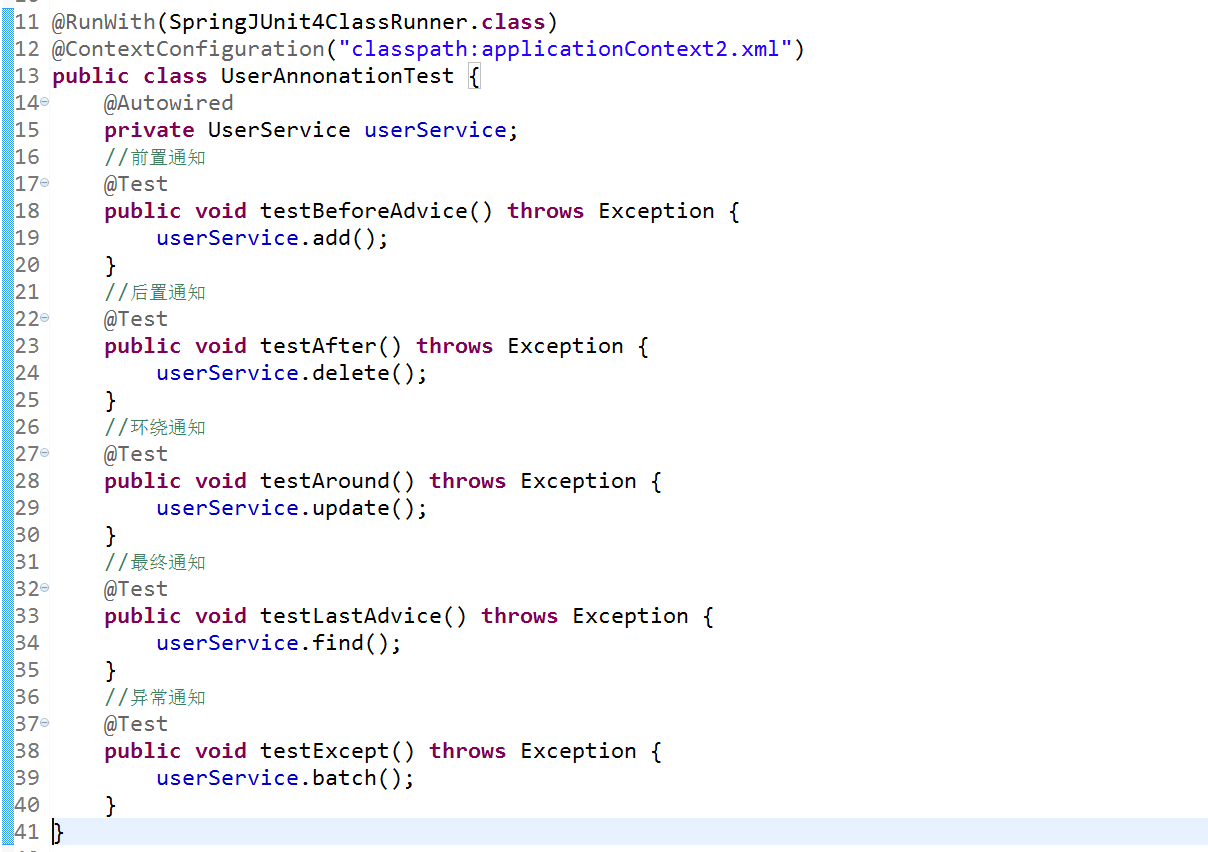
### 第四种通知：最终通知



### 第五种通知：异常通知



## 第五步：通过注解的方式来进行过调用我们的各种通知

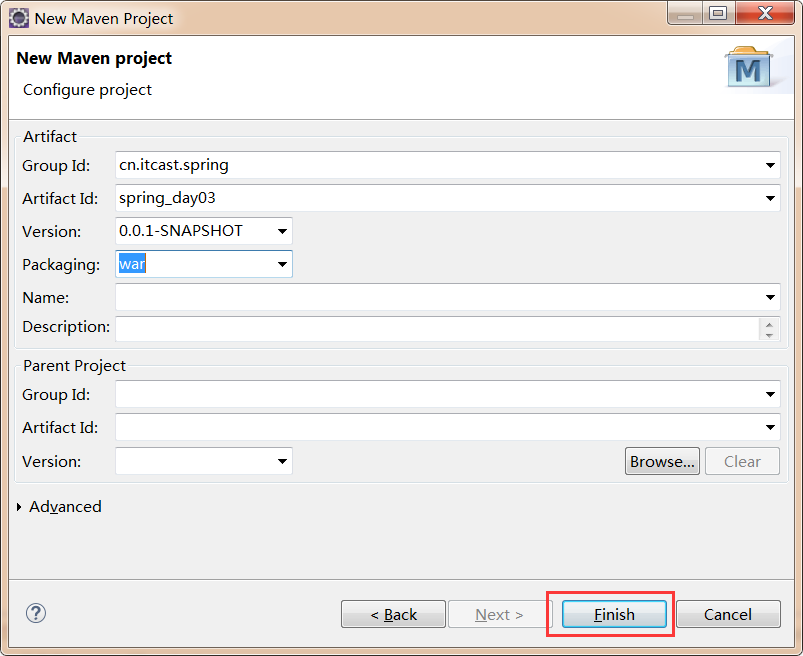


# Spring第三天

## 一：JdbcTemplate的入门程序

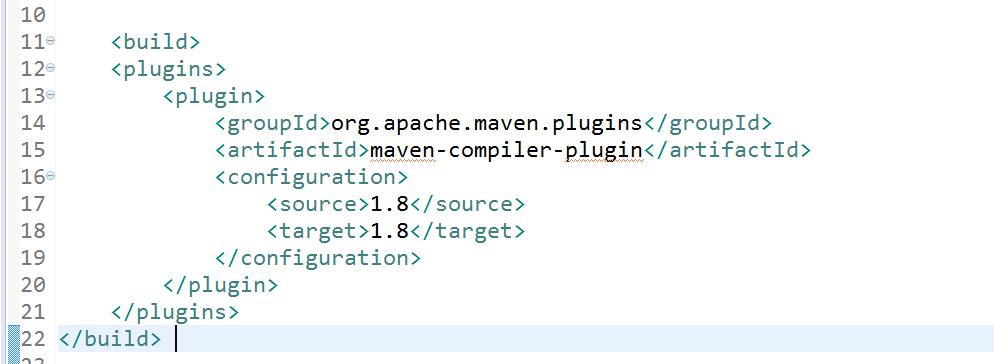
Spring是一个一站式的开发解决方案，对于web开发中的每层都有提供对应的解决方案

### 1、创建maven web工程



### 2、解决jdk版本过低与web.xml丢失的问题

解决jdk版本过低的问题：在pom.xml中添加以下配置



<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>utf-8</encoding>

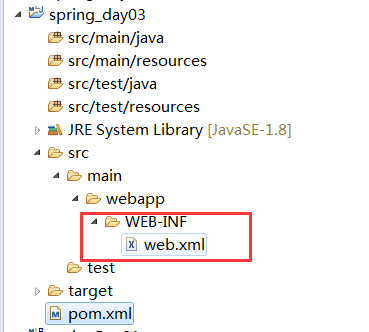
</configuration>

</plugin>

</plugins>

</build>

解决web.xml配置文件丢失的问题：



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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<display-name>01\_helloWorld</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

### 3、导入jdbc必须依赖的jar包

必须依赖jar包其坐标如下

<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-expression -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-expression</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/aopalliance/aopalliance -->

<dependency>

<groupId>aopalliance</groupId>

<artifactId>aopalliance</artifactId>

<version>1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.6.8</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aspects -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.38</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

### 4、创建数据库以及数据库表

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/`spring\_day03` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `spring\_day03`;

/\*Table structure for table `account` \*/

DROP TABLE IF EXISTS `account`;

CREATE TABLE `account` (

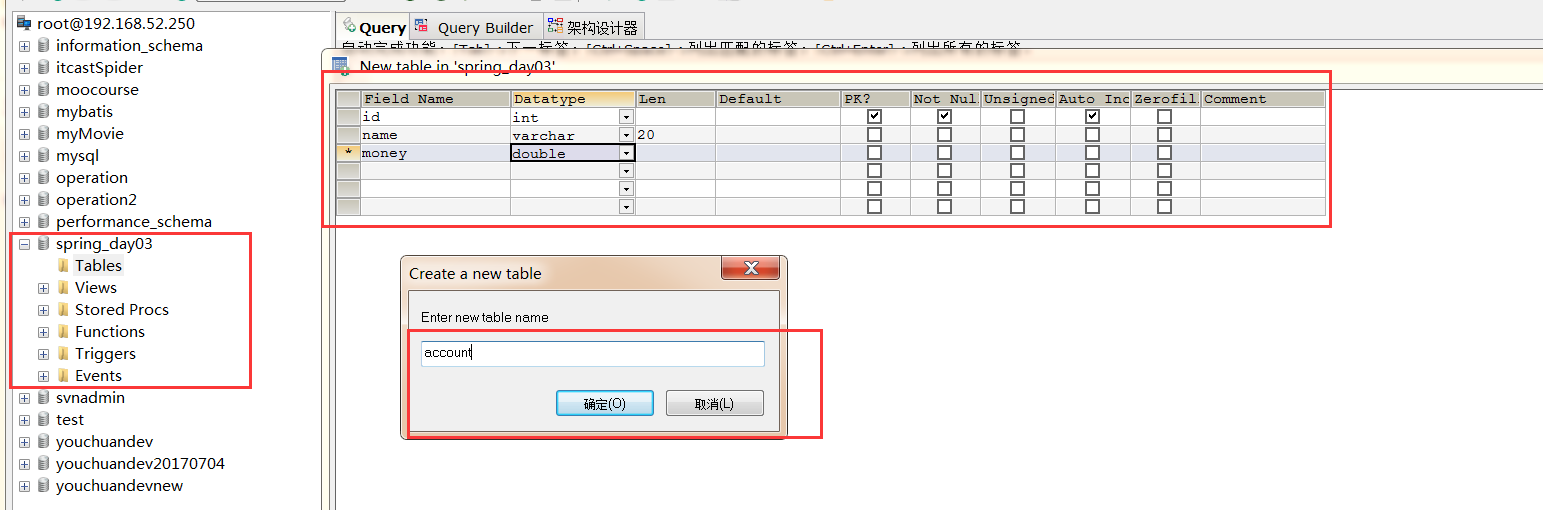
`id` int(8) NOT NULL AUTO\_INCREMENT,

`name` varchar(64) DEFAULT NULL,

`money` double DEFAULT NULL,

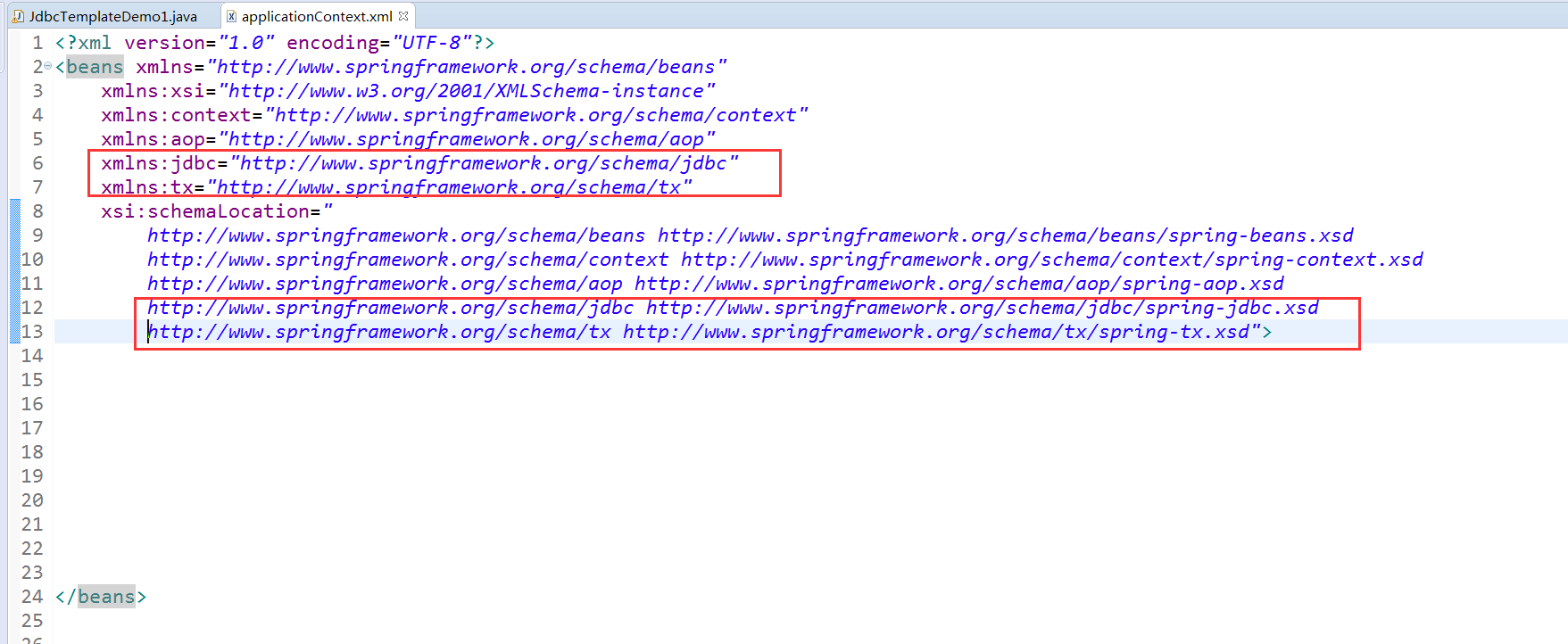
PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;



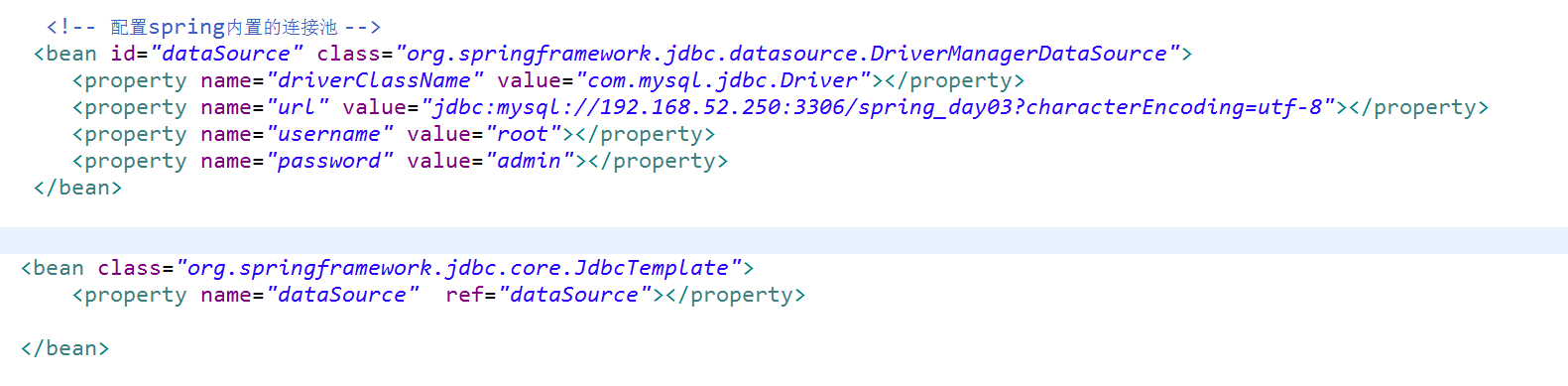
## 二：数据库连接池的配置

### 1、添加spring-jdbc与spring-tx的schema配置

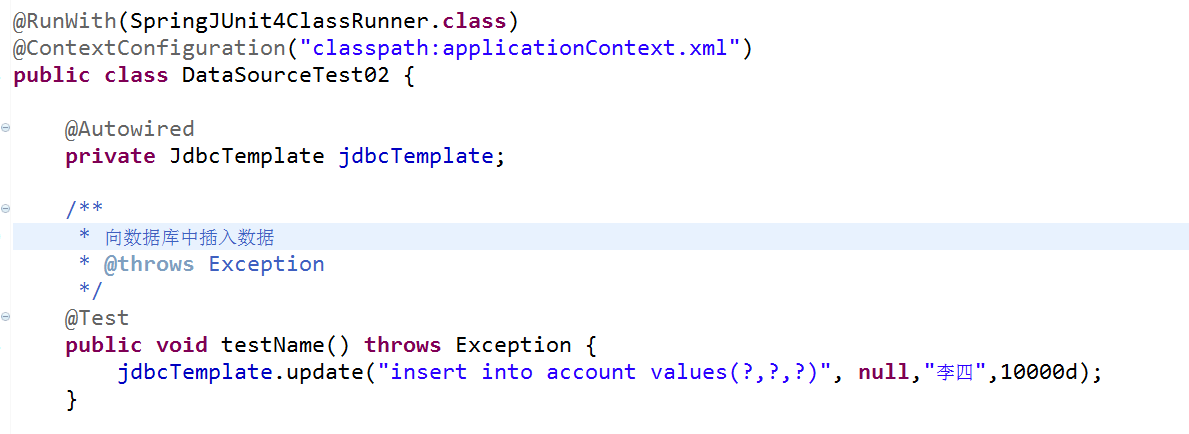


### 2、配置spring的数据库连接池

#### 第一种方式：使用spirng内置的数据库连接池



#### 第二步：注入jdbcTemplate来对数据库进行操作



### 3、配置dbcp数据库连接池

#### 第一步：导入dbcp的依赖包

<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.2.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-pool/commons-pool -->

<dependency>

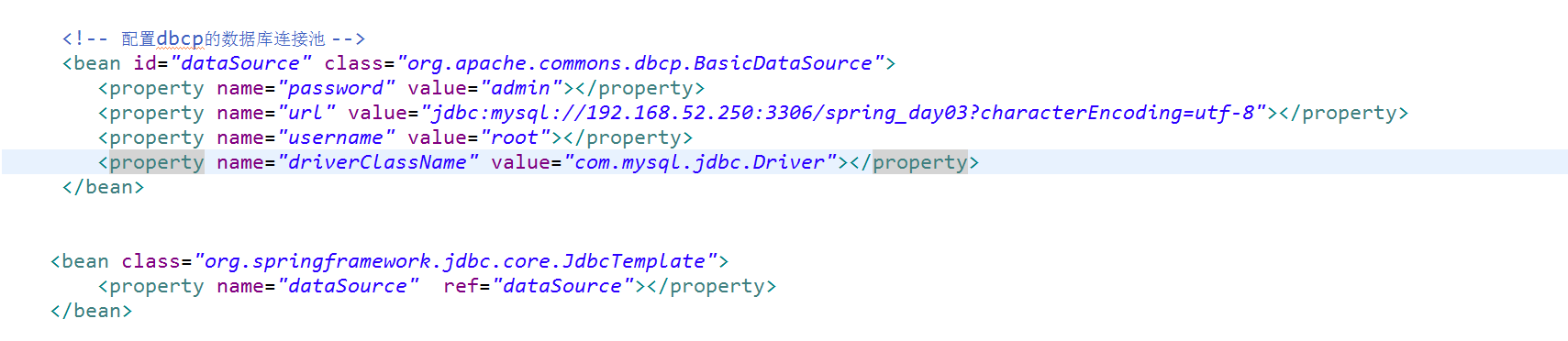
<groupId>commons-pool</groupId>

<artifactId>commons-pool</artifactId>

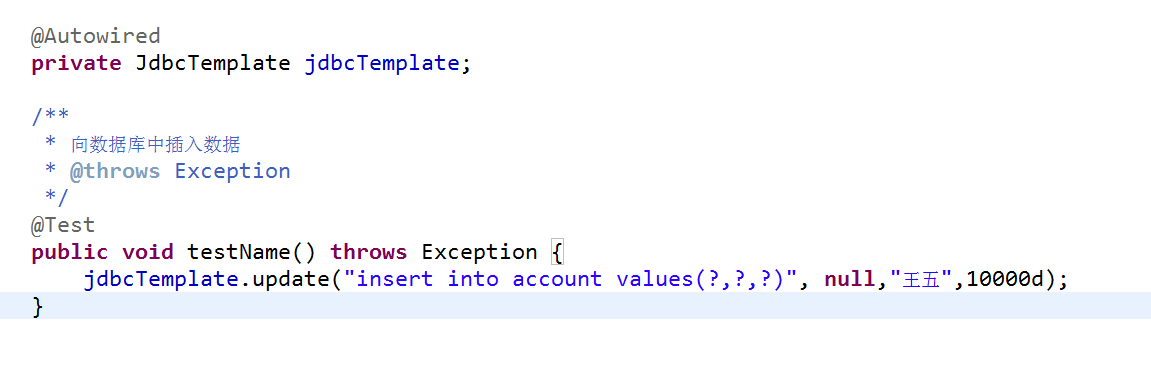
<version>1.5.3</version>

</dependency>

#### 第二步：配置dbcp的数据库连接池



#### 第三步：使用jdbcTemplate操作数据库



### 4、配置c3p0数据库连接池

#### 第一步：导入c3p0的相关jar包

<!-- https://mvnrepository.com/artifact/c3p0/c3p0 -->

<dependency>

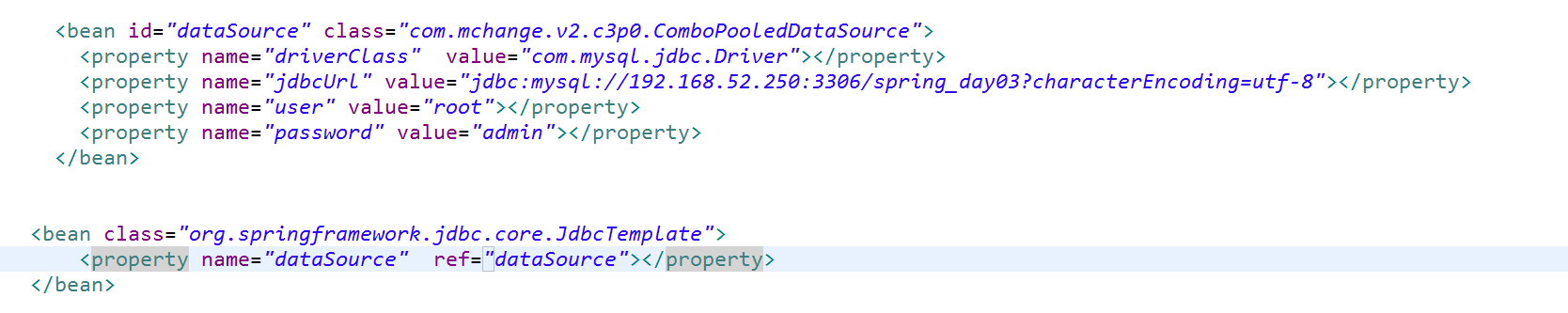
<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

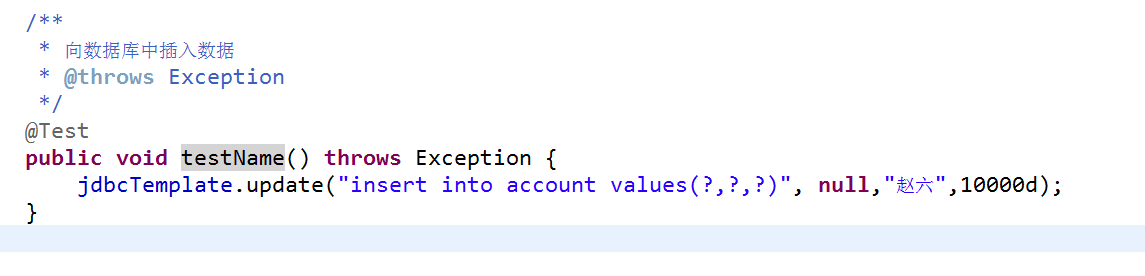
<version>0.9.1.2</version>

</dependency>

#### 第二步：配置c3p0数据库连接池



#### 第三步：使用jdbcTemplate来操作数据库



### 5、配置druid数据库连接池

#### 第一步：导入druid相关jar包

<!-- https://mvnrepository.com/artifact/com.alibaba/druid -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

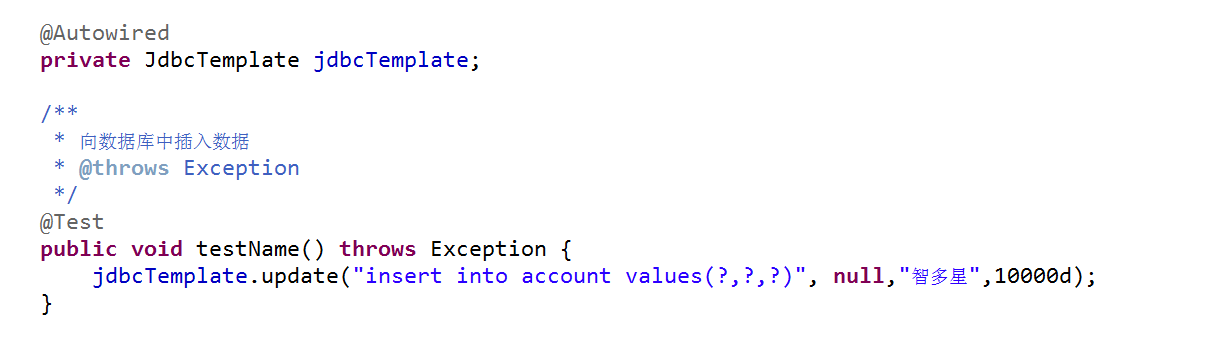
<version>1.0.18</version>

</dependency>

#### 第二步：配置druid数据库连接池



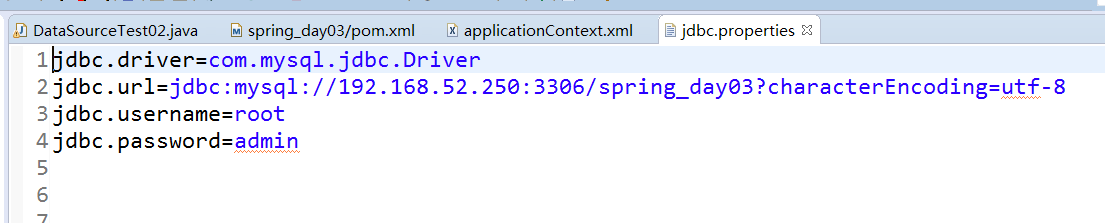
#### 第三步：使用jdbcTemplate操作数据库



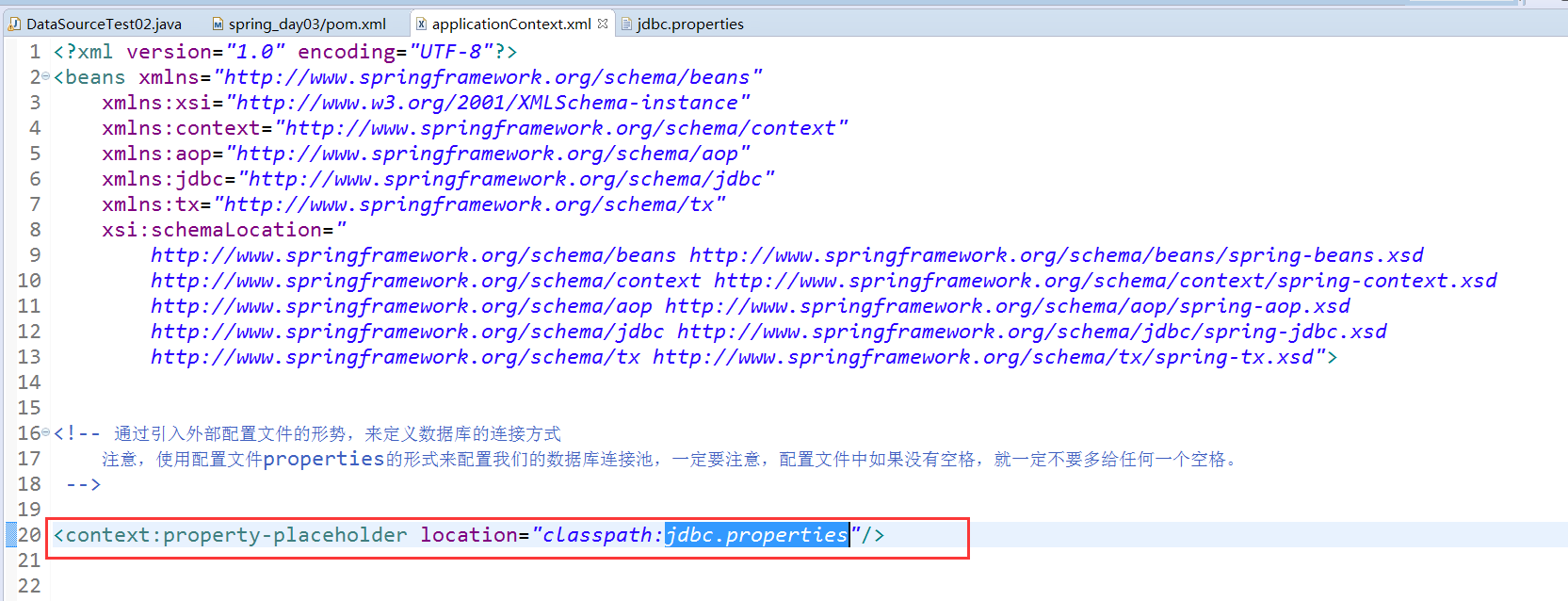
### 6、引用外部文件来配置数据库的连接信息

#### 第一步：在类路径下创建jdbc.properties文件

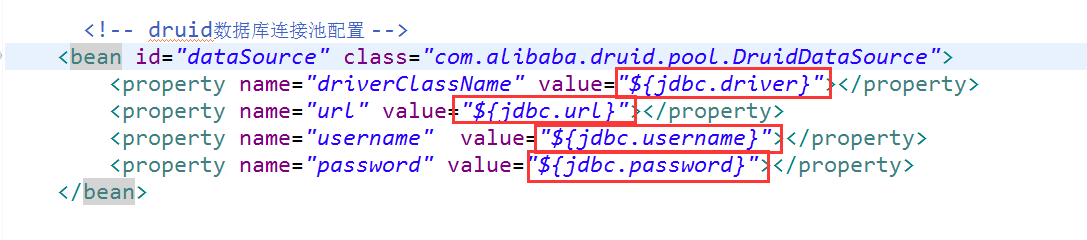
注意事项：如果配置文件中，有空格，一定要记得加上空格，如果配置文件中，没有空格，一定不要随意加入空格



#### 第二步：在applicationContext.xml中引入我们的jdbc.properties配置文件



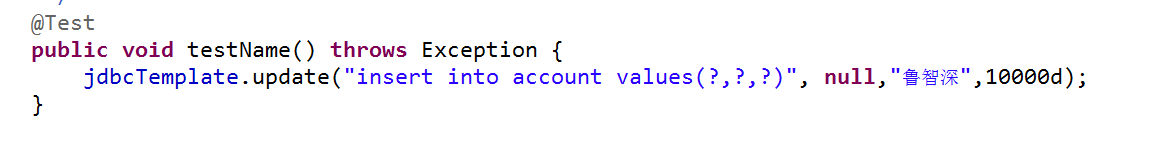
#### 第三步：改写配置文件，通过占位符的方式来使用我们的连接信息



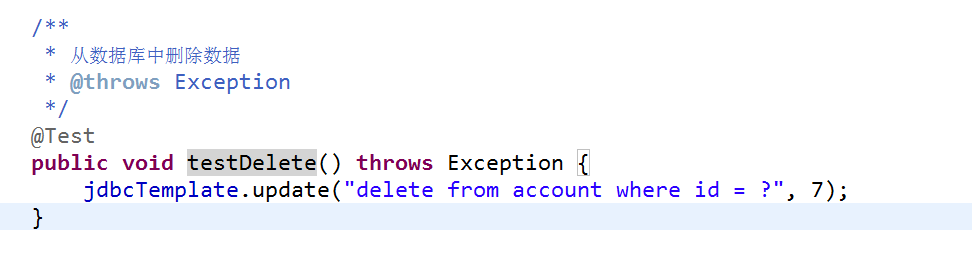
### 7、使用jdbcTemplate实现增删改查

增删改的方法都是一样的，唯一区别的就是查询的方法不一样

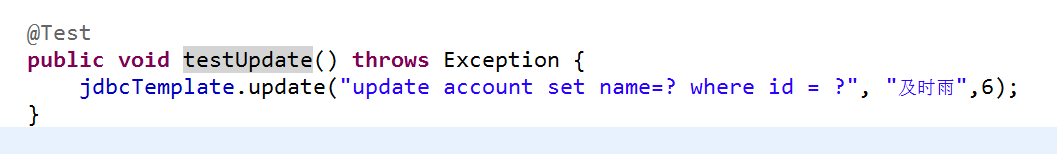
添加方法：



删除方法：



更新方法：



查询方法一：查询单个简单对象

QueryForObject(sql,class,Object…args)



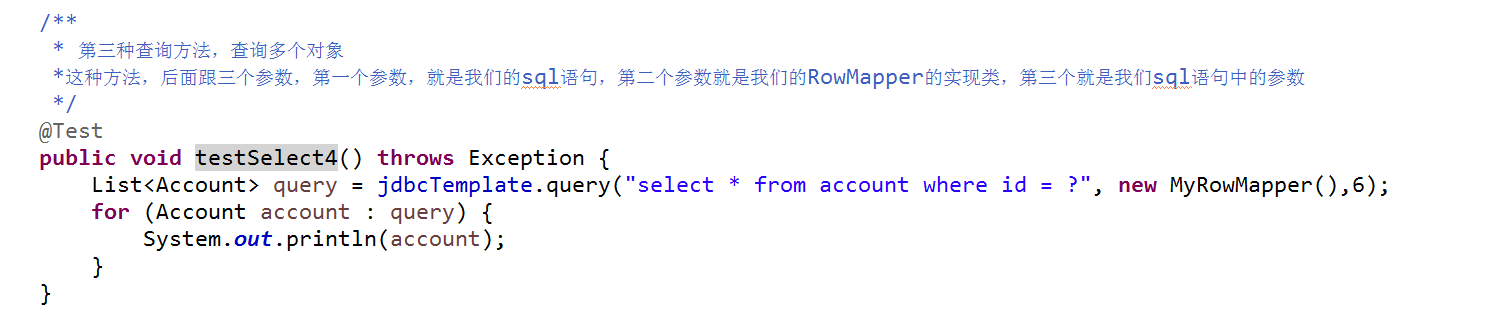
查询方法二：查询单个封装对象



查询方法三：查询多个对象集合



查询方法四：查询多个对象集合



## 三：数据库中的事物的概念

### 事物的概念：

事物是一组逻辑上的操作，组成这组操作的逻辑单元，要么全部成功，要么全部失败。

事物的特性：

原子性：每一个事物都是最小的不可再分割的单元

一致性：数据在事物执行前后不会发生变化

隔离性：各个事物之间相互不会影响

持久性：数据存储到数据库当中之后不会发生变化。

### 事物的问题

读问题：

脏读：一个事物读到另外一个事物未提交的数据

不可重复读：一个事物读到另外一个事物已经提交的update数据导致多次查询结果不一致

虚读：一个事物读到另外一个事物已经提交的insert数据导致多次查询结果不一致

写问题：

丢失更新

### 解决读问题：

设置事物的隔离级别

Read uncommited

Read commited：可以解决脏读的问题，Oracle默认使用

Repeatable read：可以解决脏读与不可重复读的问题，Mysql默认使用

Serializable：都可以解决

### Spring当中的事物解决：PlatformTransactionManager

PlatformTransactionManager：这个就是spring当中用于实现事物的类

[DataSourceTransactionManager](file:///G:\%E7%94%B5%E5%95%86%E5%A4%A7%E6%95%B0%E6%8D%AE%E5%A4%87%E8%AF%BE%E8%B5%84%E6%96%99\spring4\%E8%B5%84%E6%96%99\spring-framework-4.2.4.RELEASE-dist\spring-framework-4.2.4.RELEASE\docs\javadoc-api\org\springframework\jdbc\datasource\DataSourceTransactionManager.html)：这个就是PlatformTransactionManager的一个实现类，主要用于jdbc操作数据库来实现我们的事物控制

[HibernateTransactionManager](file:///G:\%E7%94%B5%E5%95%86%E5%A4%A7%E6%95%B0%E6%8D%AE%E5%A4%87%E8%AF%BE%E8%B5%84%E6%96%99\spring4\%E8%B5%84%E6%96%99\spring-framework-4.2.4.RELEASE-dist\spring-framework-4.2.4.RELEASE\docs\javadoc-api\org\springframework\orm\hibernate3\HibernateTransactionManager.html)：这个也是PlatformTransactionManager的一个实现类，主要用于hibernate这个框架来实现我们的事物控制

### Spring当中事物的定义：TransactionDefinition

事物的隔离级别：

事物的传播行为

事物的超时信息

事物是否只读

### Spring当中事物的状态：TransactionStatus

状态是否有保存点

事物是否完成

是不是一个新的事物

是否仅仅回滚

### Spring当中的事物执行过程

Spring当中通过TransactionDefinition来对事物进行定义，然后通过

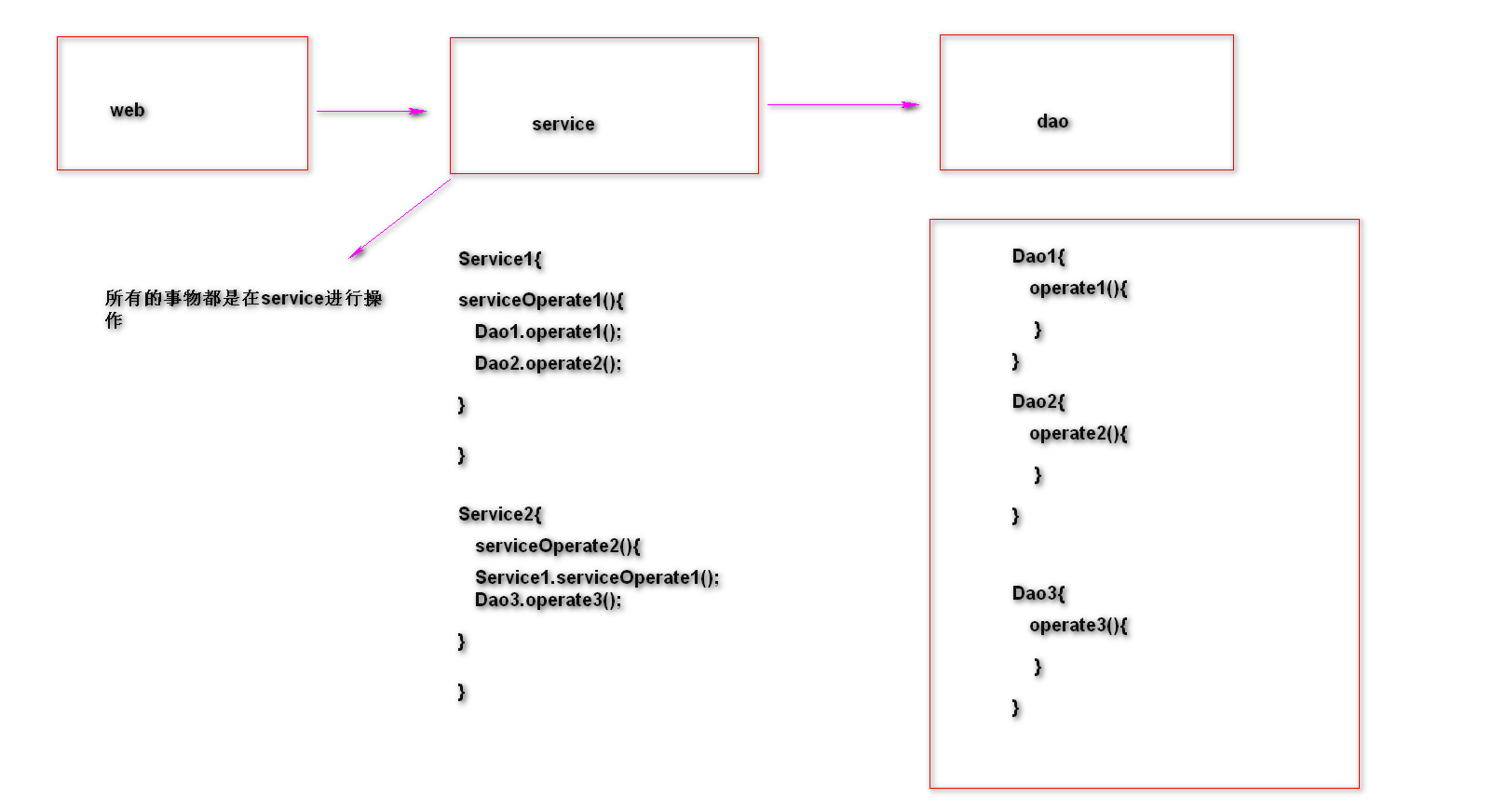
PlatformTransactionManager来对事物进行操作，并且将操作后的事物状态，保存到

TransactionStatus当中

### 事物的传播行为：

用于解决哪些问题：解决复杂的事物操作，尤其是业务层方法之间相互调用的问题





PROPARGATION\_REQUIRED：支持当前事物，如果不存在，就新建一个事物，保证后面的dao操作与前面的事物是在一个事物里面。通常是最常用的一种事物传播行为

PROPARGATION\_SUPPORT：支持前面调用的事物，如果前面调用的不存在事物，就不使用事物，不能保证事物的一致性，一般不用

PROPARGATION\_MANDATORY：支持当前事物，如果没有当前事物，就会抛异常。

PROPARGATION\_REQUIRES\_NEW：如果之前的操作事物存在，那么就刮起，创建一个新的事物，相当于废弃之前的事物，使用自己重新创建的事物，可以保证事物的一致性，但是比较消耗资源。

PROPARGATION\_NOT\_SUPPORTED：不以事物的形势运行，相当于没有任何事物的概念。绝对不可以使用得。

PROPARGATION\_NEVER：以非事物的方式运行，如果有事物存在，那么就会报异常。这个也是绝对不可以使用的。

PROPARGATION\_NESTED：如果当前事物存在，就进行事物的嵌套执行。有一个保存点的概念，相当于之前的事物正常执行完了之后，给一个保存点，后面的事物继续进行执行，如果后面的抛错，可以选择回滚到保存点或者所有的事物都回滚。这个比较复杂，一般用的也不多。

## 四：3种事物方式的配置开发

### Spring当中主要有两种事物的管理方式

第一种：编程式事物管理：手动编写代码实现事物管理

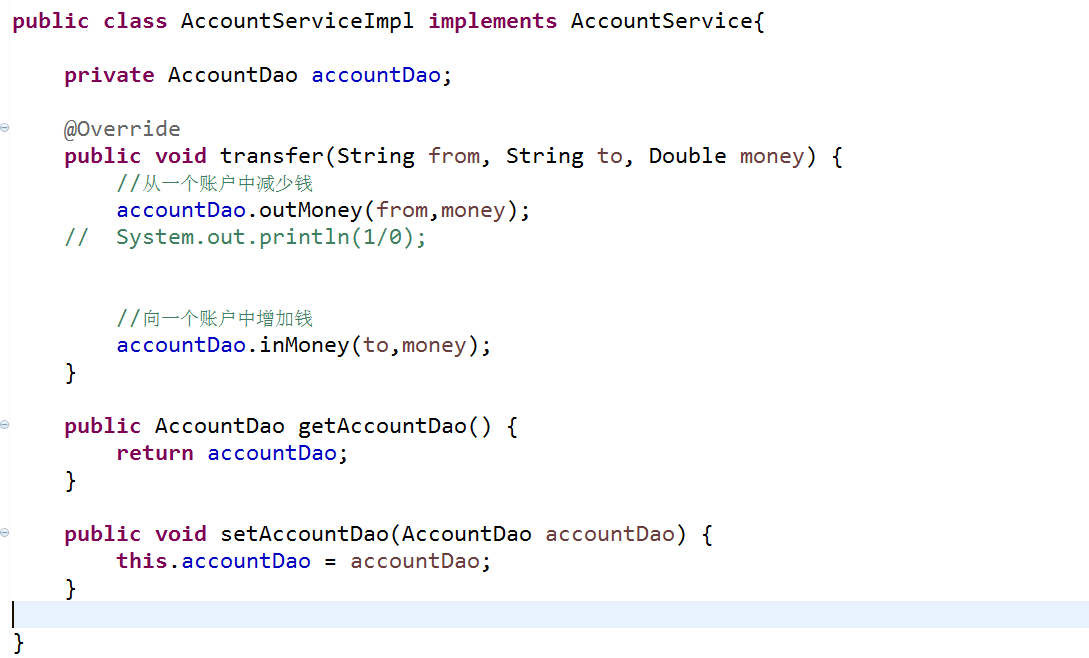
第二种：申明式事物管理

Xml方式申明事物管理

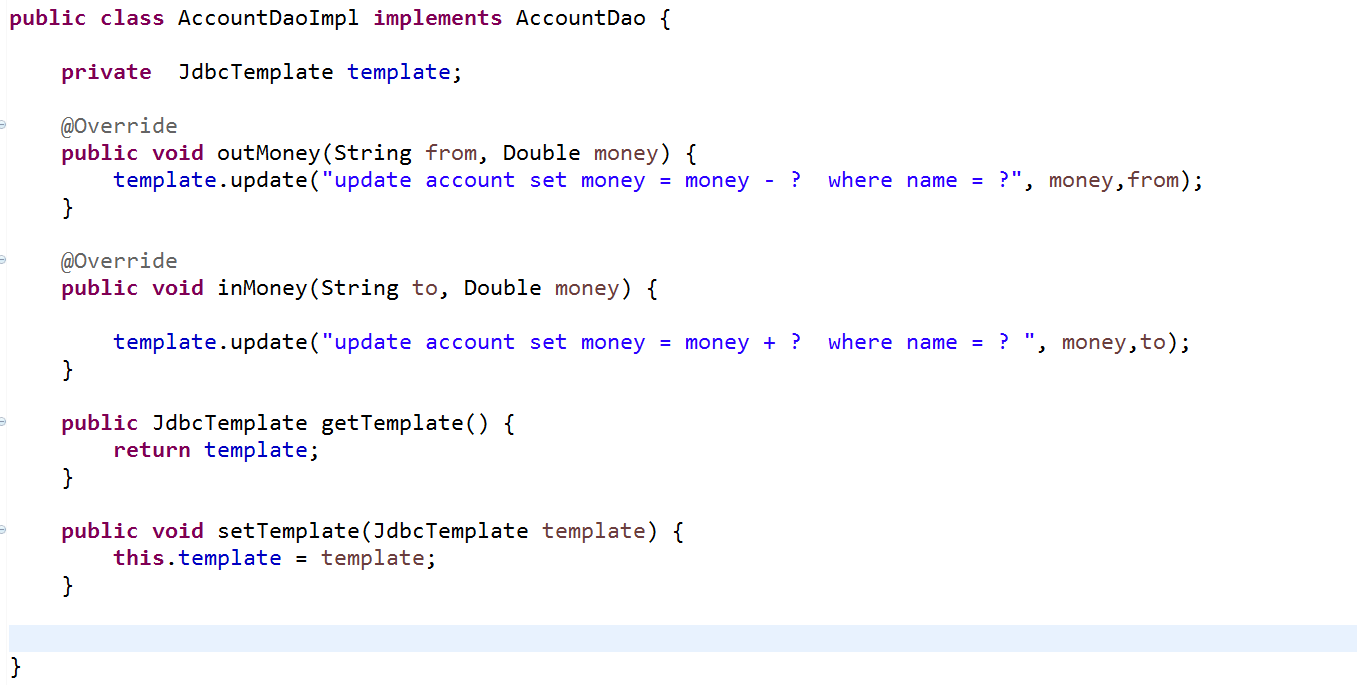
注解方式申明事物管理

### 搭建转账的环境

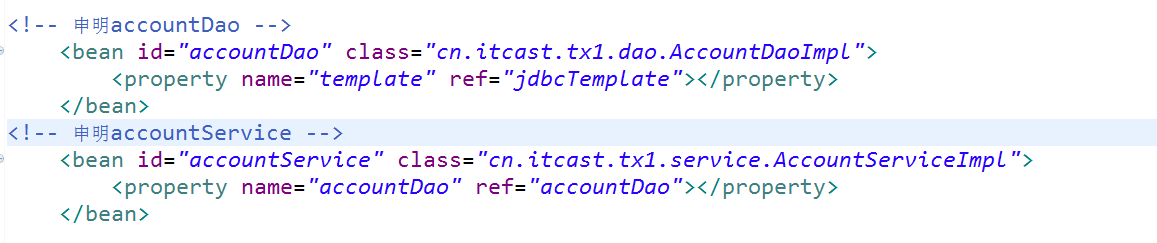
#### 第一步：创建service



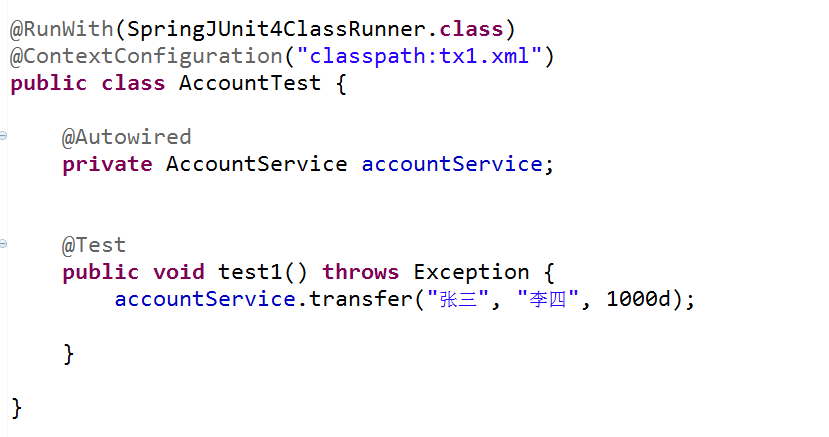
#### 第二步：创建转账的dao操作



#### 第三步：在配置文件中申明我们的dao与service的对象

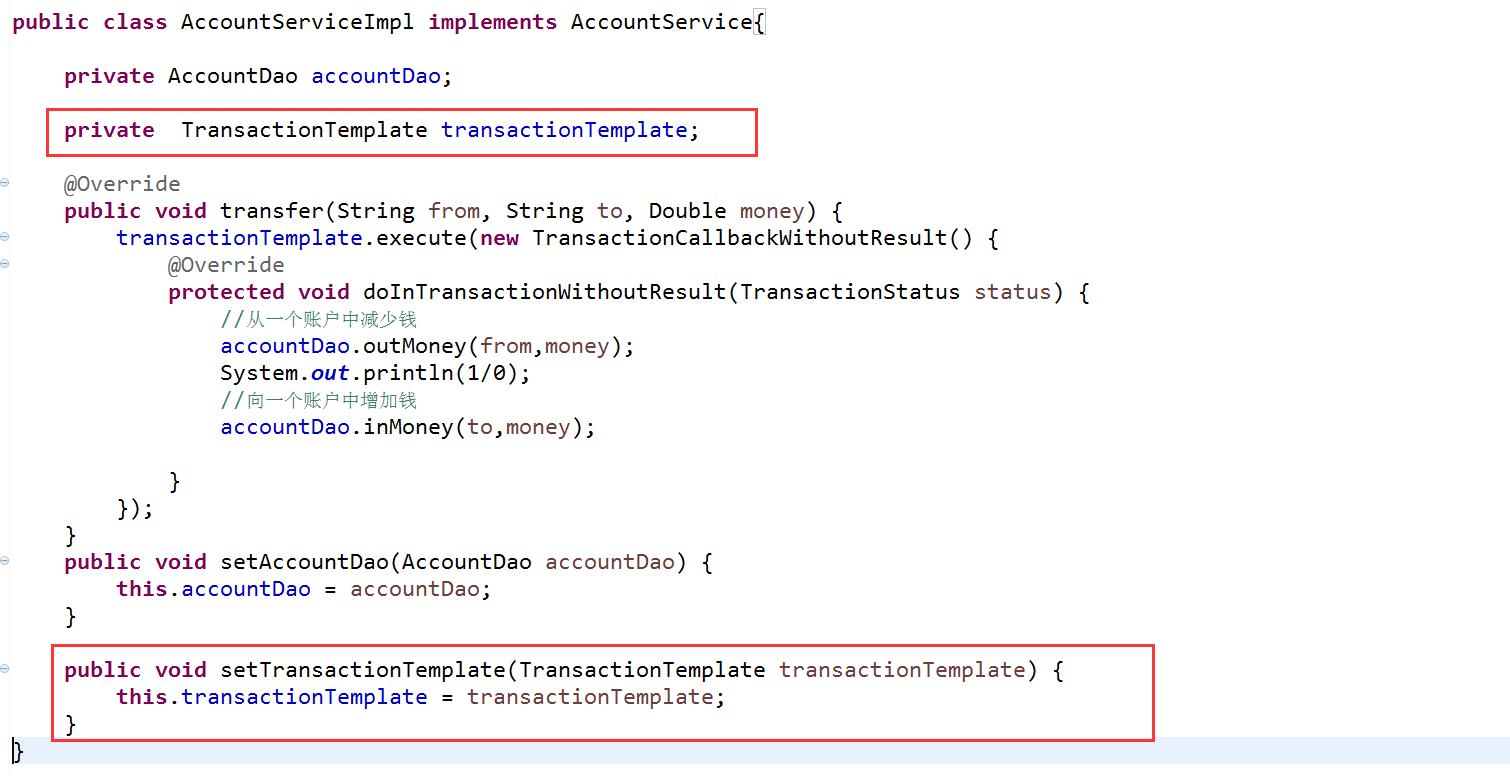


#### 第四步：测试我们的转账环境是否成功



### Spring当中编程式事物的申明

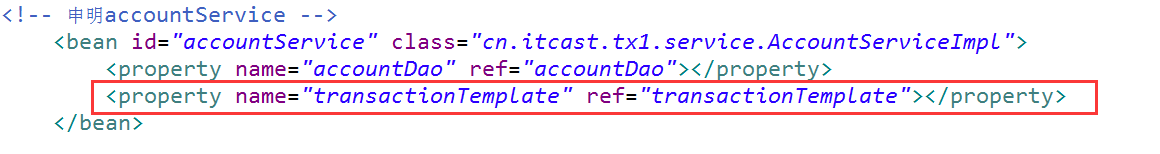
#### 第一步：在service层当中给出私有成员TransactionTemplate并同时给出set方法



#### 第二步：添加我们的事物管理类



#### 第三步：更改我们accountService，为其设置一个新的属性TransactionTemplate



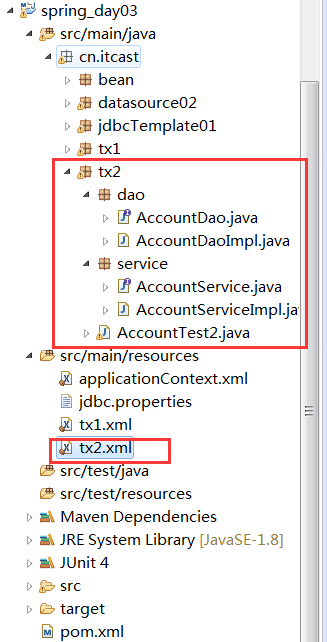
#### 第四步：更改转账时候的执行代码



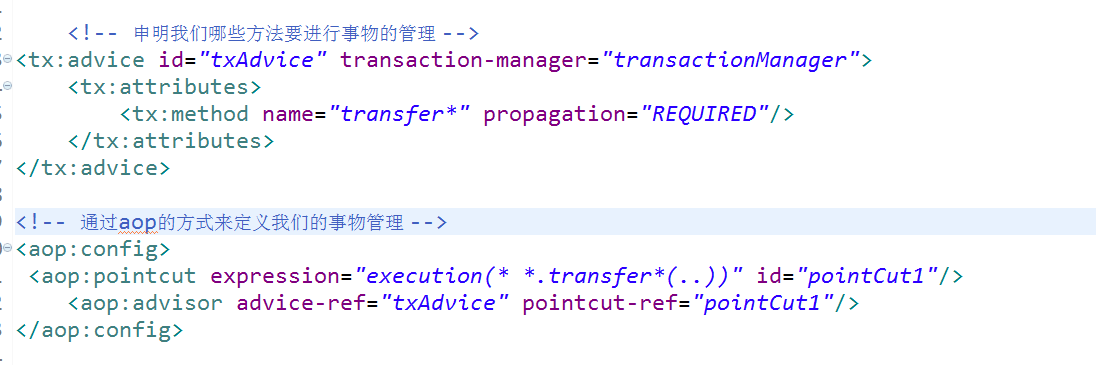
### Spring当中申明式事物的管理（通过xml配置的方式实现）

#### 第一步：恢复转账环境

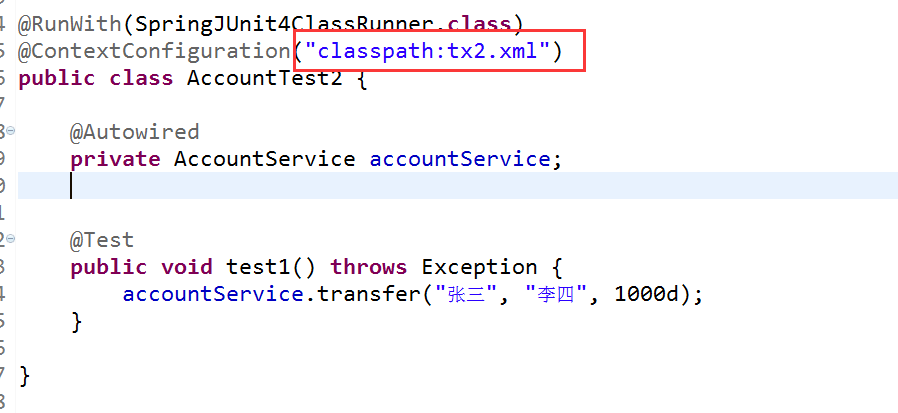
一定要注意导包的问题



#### 第二步：配置通过xml的方式来实现我们的事物管理

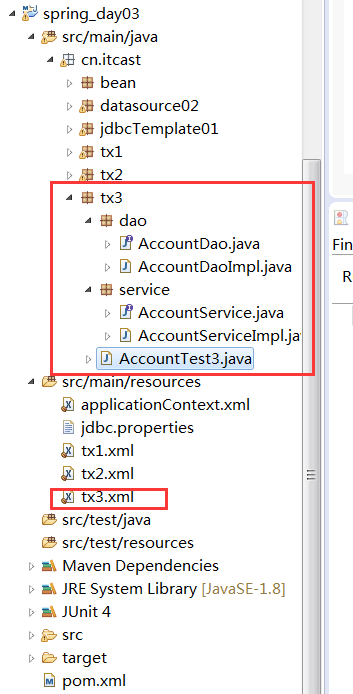


#### 第三步：测试我们的事物管理是否成功执行

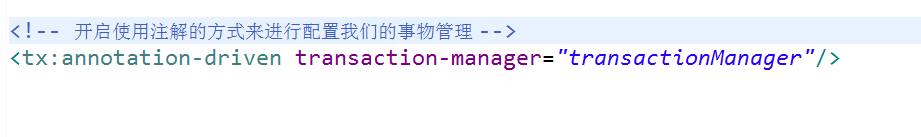


### Spring当中通过注解的方式来实现事物的控制

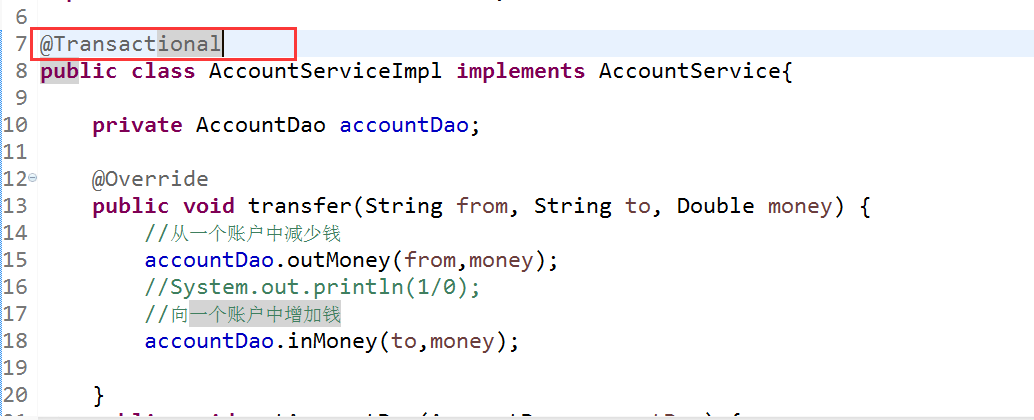
#### 第一步：转账环境的回复



#### 第二步：开启使用注解的方式来执行我们的事物



#### 第三步：在我们的service类中加上事物管理的注解



#### 第三步：测试我们的事物执行

