# Spring\_day03笔记

## 上次课内容回顾

* Spring的IOC的注解开发
  + 注解的入门
    - 引入aop的包
    - 引入context约束
      * <context:component-scan />
    - 使用注解开发
      * @Component ：定义Bean
        + **@Controller ：WEB层**
        + **@Service ：Service层**
        + **@Repository ：DAO层**
      * **属性注入：**
        + **普通属性 ：@Value**
        + **对象属性 ：@Resource**

**@Autowired ：按类型注入属性，按名称@Qulifier**

* + **XML方式和注解方式比较**
    - **XML方式 ：适用性更广，结构更加清晰。**
    - **注解方式 ：适用类是自己定义，开发更方便。**
  + **XML和注解的整合开发**
    - **XML定义类**
    - **注解属性注入**
* Spring的AOP的基于AspectJ的XML的开发
  + AOP的概述
    - AOP：面向切面编程，是OOP的扩展和延伸，是用来解决OOP遇到问题。
  + Spring的AOP
    - 底层的实现
      * JDK的动态代理
      * Cglib的动态代理
    - AOP的相关术语
      * 连接点：可以被拦截的点。
      * 切入点：真正被拦截的点。
      * 通知：增强方法
      * 引介：类的增强
      * 目标：被增强的对象
      * 织入：将增强应用到目标的过程。
      * 代理：织入增强后产生的对象
      * 切面：切入点和通知的组合
    - AOP的入门开发
      * 引入jar包
      * 编写目标类并配置
      * 编写切面类并配置
      * 进行aop的配置

<aop:config>

<aop:pointcut expression=”execution(表达式)” id=”pc1”/>

<aop:aspect >

<aop:before method=”” pointcut-ref=”pc1”/>

</aop:aspect>

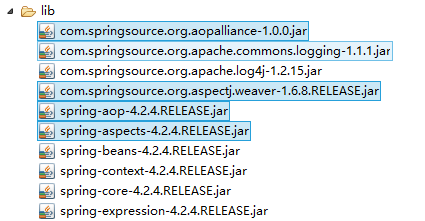
</aop:config>

* + - 通知类型
      * 前置通知
      * 后置通知
      * 环绕通知
      * 异常抛出通知
      * 最终通知
    - 切入点表达式写法

## Spring的AOP的基于AspectJ注解开发

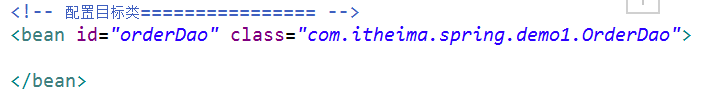
### Spring的基于ApsectJ的注解的AOP开发

#### 创建项目，引入jar包

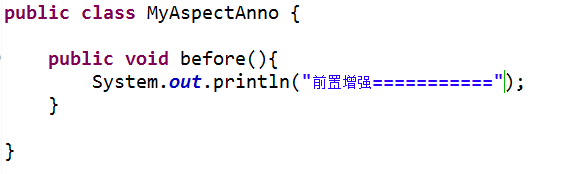


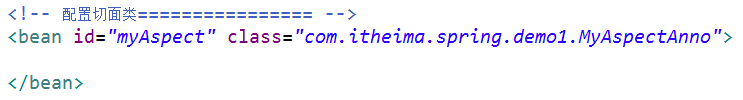
#### 引入配置文件

#### 编写目标类并配置



#### 编写切面类并配置





#### 使用注解的AOP对象目标类进行增强

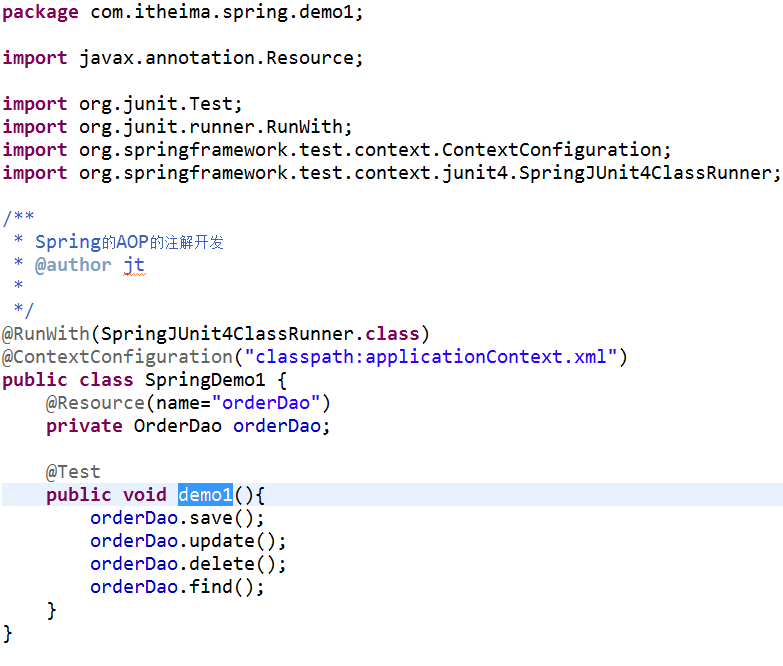
* 在配置文件中打开注解的AOP开发



* 在切面类上使用注解

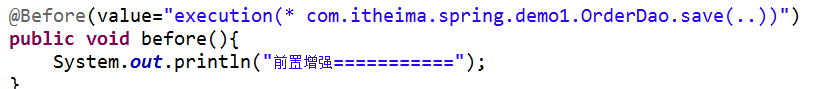


#### 编写测试类

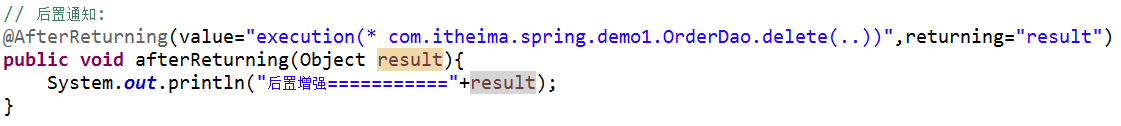


### Spring的注解的AOP的通知类型

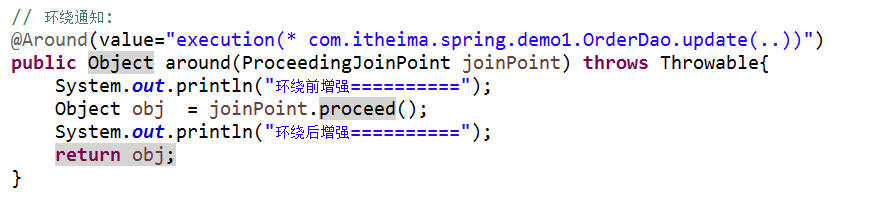
#### @Before ：前置通知



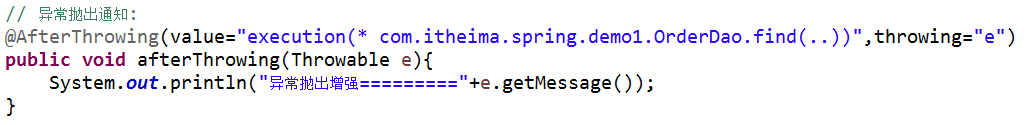
#### @AfterReturning ：后置通知



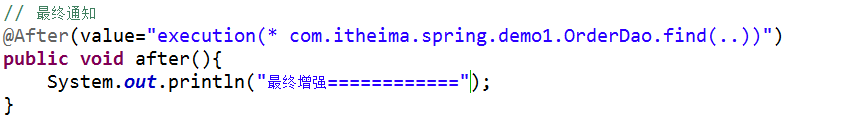
#### @Around ：环绕通知



#### @AfterThrowing ：异常抛出通知

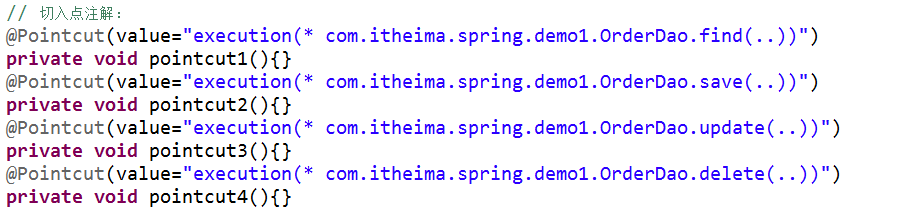


#### @After ：最终通知



### Spring的注解的AOP的切入点的配置

#### Spring的AOP的注解切入点的配置

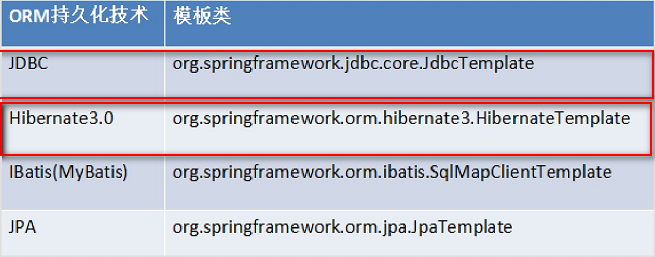


## Spring的JDBC的模板的使用

### Spring的JDBC的模板

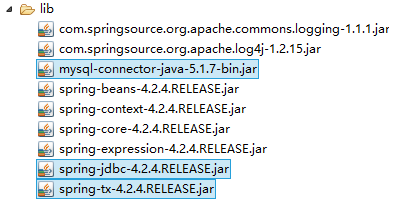
Spring是EE开发的一站式的框架，有EE开发的每层的解决方案。Spring对持久层也提供了解决方案：ORM模块和**JDBC的模板**。

Spring提供了很多的模板用于简化开发：



#### JDBC模板使用的入门

* 创建项目，引入jar包
  + 引入基本开发包：
  + 数据库驱动
  + Spring的JDBC模板的jar包



#### 创建数据库和表

create database spring4\_day03;

use spring4\_day03;

create table account(

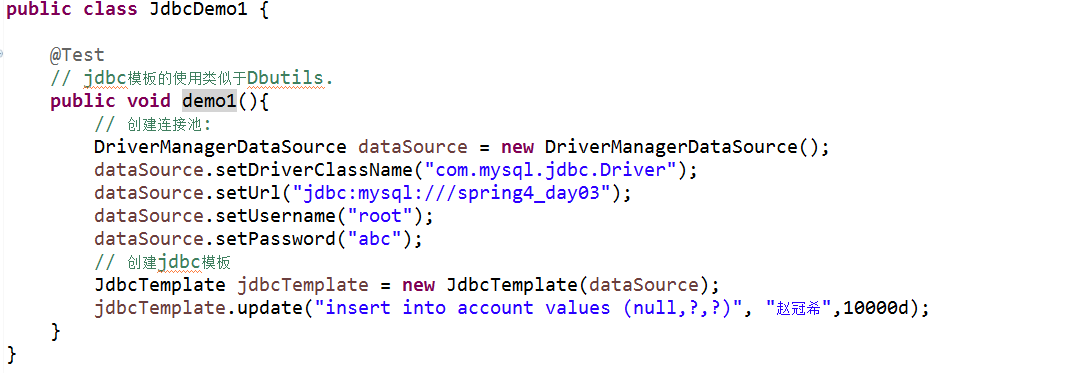
id int primary key auto\_increment,

name varchar(20),

money double

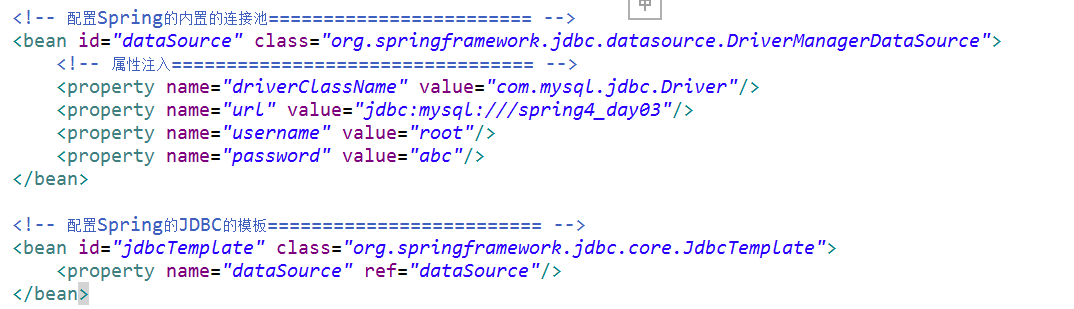
);

#### 使用JDBC的模板：保存数据



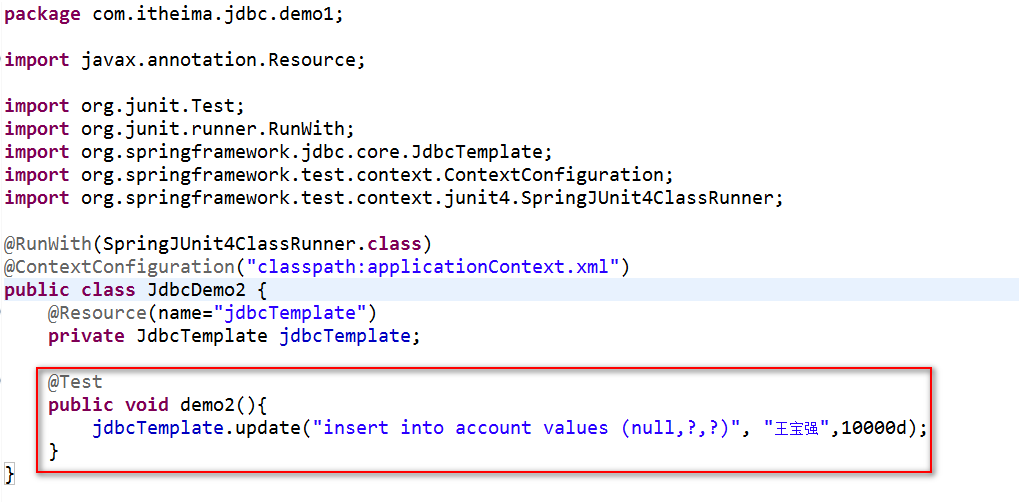
### 将连接池和模板交给Spring管理

#### 引入Spring的配置文件



#### 使用Jdbc的模板

* **引入spring\_aop的jar包**



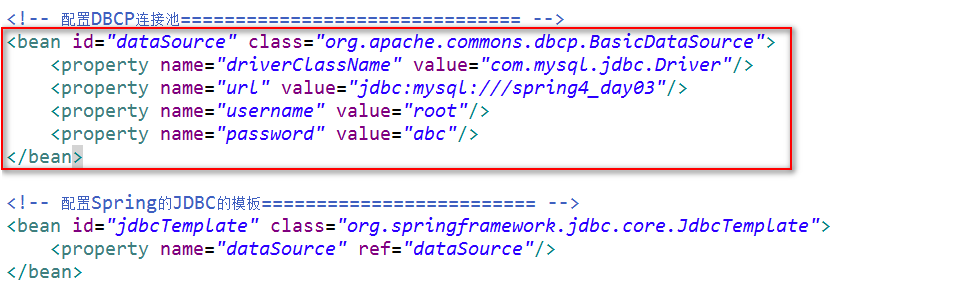
### 使用开源的数据库连接池：

#### DBCP的使用

* 引入jar包



* 配置DBCP连接池

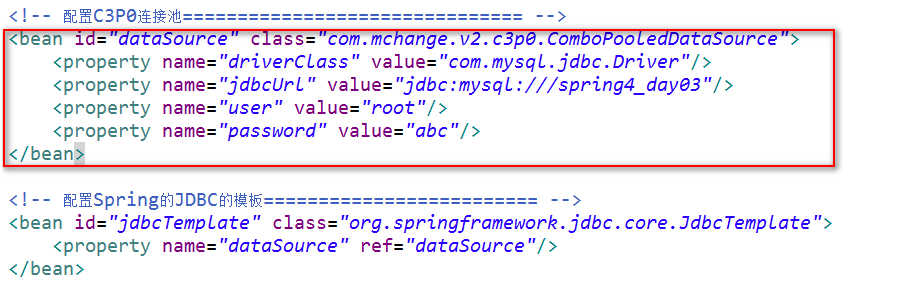


#### C3P0的使用

* 引入c3p0连接池jar包

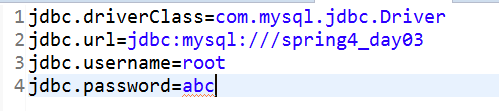


* 配置c3p0连接池



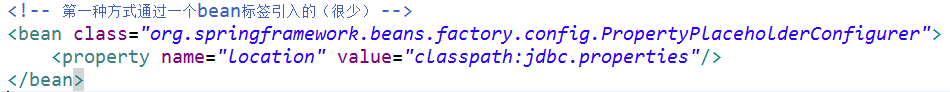
### 抽取配置到属性文件

#### 定义一个属性文件



#### 在Spring的配置文件中引入属性文件

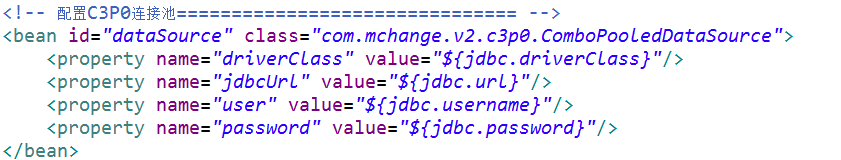
* 第一种：



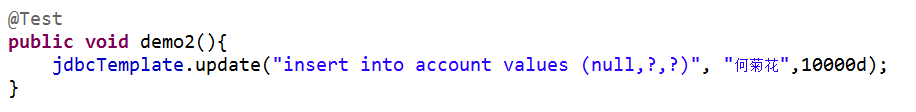
* 第二种：



#### 引入属性文件的值

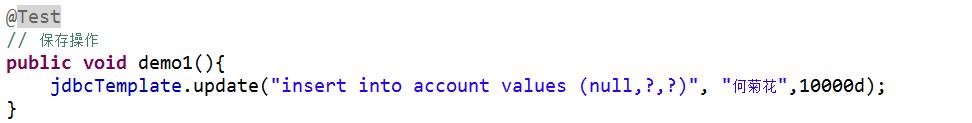


#### 测试

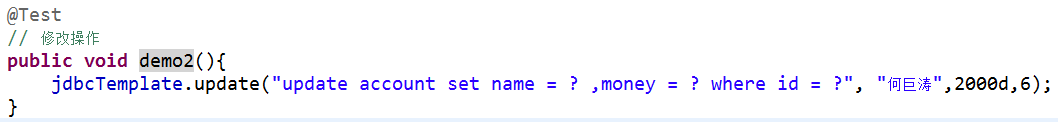


### 使用JDBC的模板完成CRUD的操作

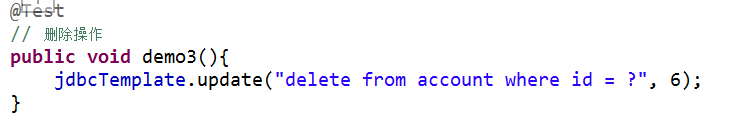
#### 保存操作



#### 修改操作

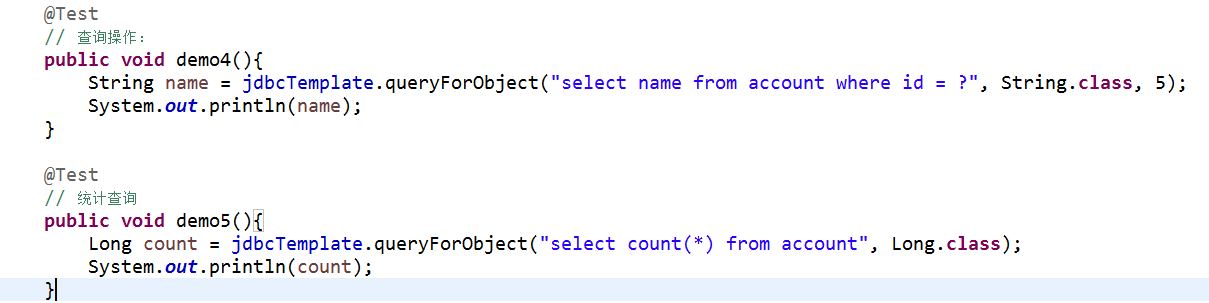


#### 删除操作

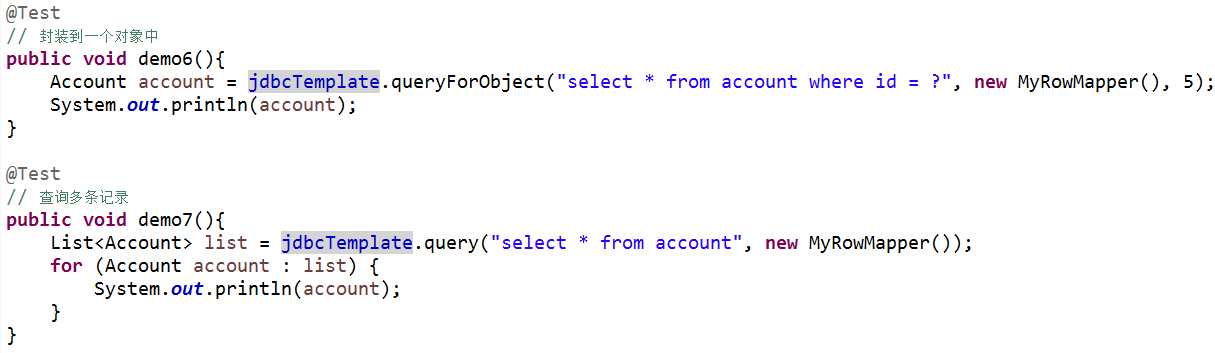


#### 查询操作

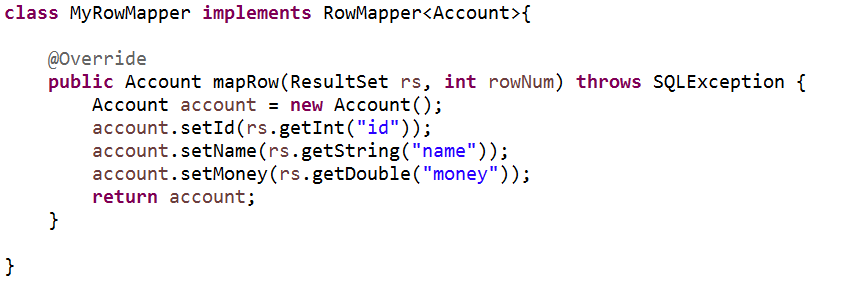
* 查询某个属性



* 查询返回对象或集合



* 数据封装



## Spring的事务管理

### 事务的回顾

#### 什么事务

* 事务：逻辑上的一组操作，组成这组操作的各个单元，要么全都成功，要么全都失败。

#### 事务的特性

* 原子性：事务不可分割
* 一致性：事务执行前后数据完整性保持一致
* 隔离性：一个事务的执行不应该受到其他事务的干扰
* 持久性：一旦事务结束，数据就持久化到数据库

#### 如果不考虑隔离性引发安全性问题

* 读问题
  + 脏读 ：一个事务读到另一个事务未提交的数据
  + 不可重复读 ：一个事务读到另一个事务已经提交的update的数据，导致一个事务中多次查询结果不一致
  + 虚读、幻读 ：一个事务读到另一个事务已经提交的insert的数据，导致一个事务中多次查询结果不一致。
* 写问题
  + 丢失更新

#### 解决读问题

* 设置事务的隔离级别
  + Read uncommitted ：未提交读，任何读问题解决不了。
  + Read committed ：已提交读，解决脏读，但是不可重复读和虚读有可能发生。
  + Repeatable read ：重复读，解决脏读和不可重复读，但是虚读有可能发生。
  + Serializable ：解决所有读问题。

### Spring的事务管理的API

#### PlatformTransactionManager：平台事务管理器

* 平台事务管理器：接口，是Spring用于管理事务的真正的对象。
  + **DataSourceTransactionManager ：底层使用JDBC管理事务**
  + HibernateTransactionManager ：底层使用Hibernate管理事务

#### TransactionDefinition ：事务定义信息

* 事务定义：用于定义事务的相关的信息，隔离级别、超时信息、**传播行为**、是否只读

#### TransactionStatus：事务的状态

* 事务状态：用于记录在事务管理过程中，事务的状态的对象。

#### 事务管理的API的关系：

Spring进行事务管理的时候，首先平台事务管理器根据事务定义信息进行事务的管理，在事务管理过程中，产生各种状态，将这些状态的信息记录到事务状态的对象中。

### Spring的事务的传播行为

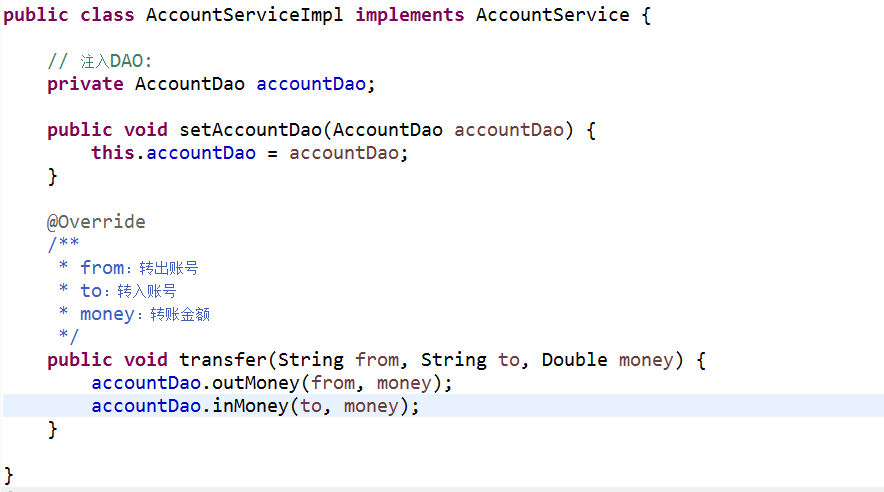
#### Spring的传播行为

* Spring中提供了七种事务的传播行为：
  + 保证多个操作在同一个事务中
    - [**PROPAGATION\_REQUIRED**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_REQUIRED) **：默认值，如果A中有事务，使用A中的事务，如果A没有，创建一个新的事务，将操作包含进来**
    - [**PROPAGATION\_SUPPORTS**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_SUPPORTS) **：支持事务，如果A中有事务，使用A中的事务。如果A没有事务，不使用事务。**
    - [**PROPAGATION\_MANDATORY**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_MANDATORY) **：如果A中有事务，使用A中的事务。如果A没有事务，抛出异常。**
  + 保证多个操作不在同一个事务中
    - [**PROPAGATION\_REQUIRES\_NEW**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_REQUIRES_NEW) **：如果A中有事务，将A的事务挂起（暂停），创建新事务，只包含自身操作。如果A中没有事务，创建一个新事务，包含自身操作。**
    - [**PROPAGATION\_NOT\_SUPPORTED**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NOT_SUPPORTED) **：如果A中有事务，将A的事务挂起。不使用事务管理。**
    - [**PROPAGATION\_NEVER**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NEVER) **：如果A中有事务，报异常。**
  + 嵌套式事务
    - [**PROPAGATION\_NESTED**](mk:@MSITStore:E:\手册\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NESTED) **：嵌套事务，如果A中有事务，按照A的事务执行，执行完成后，设置一个保存点，执行B中的操作，如果没有异常，执行通过，如果有异常，可以选择回滚到最初始位置，也可以回滚到保存点。**

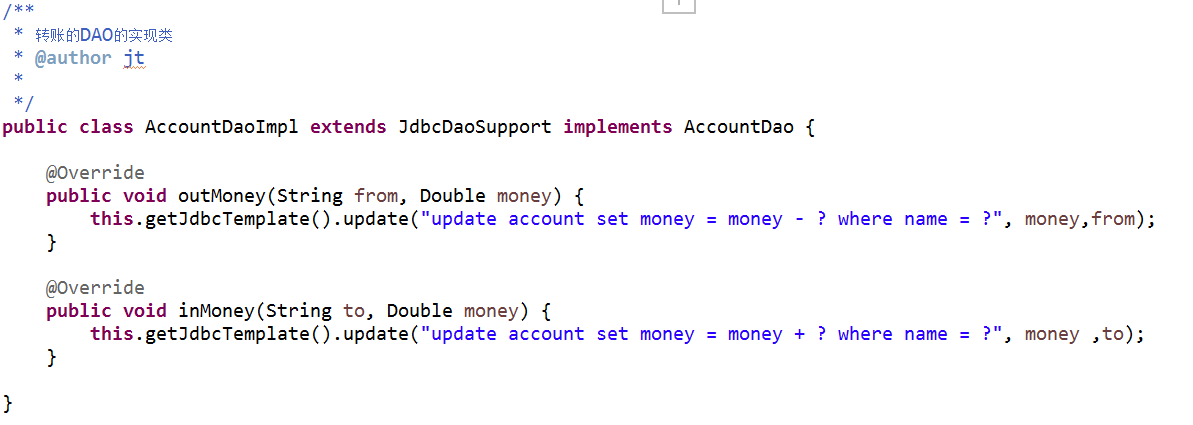
### Spring的事务管理

#### 搭建Spring的事务管理的环境

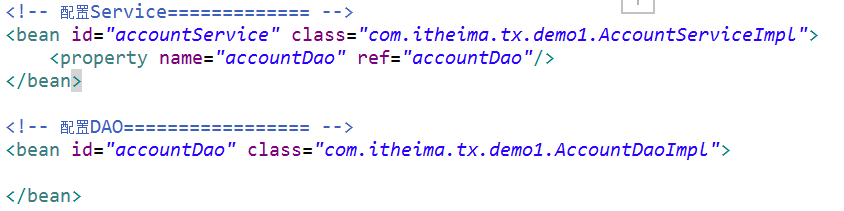
* 创建Service的接口和实现类



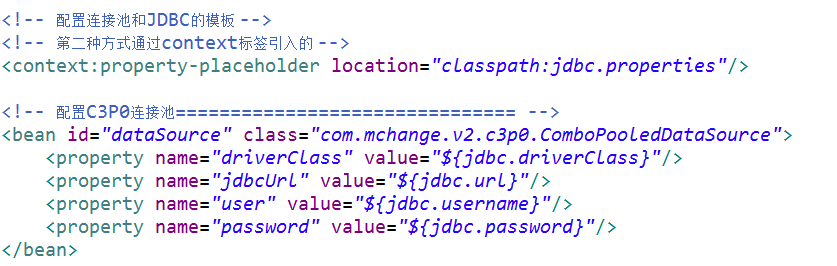
* 创建DAO的接口和实现类



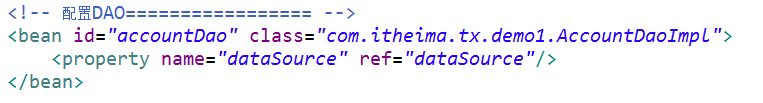
* 配置Service和DAO：交给Spring管理



* 在DAO中编写扣钱和加钱方法：
  + 配置连接池和JDBC的模板



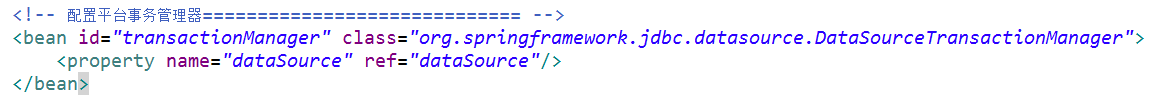
* + 在DAO注入Jdbc的模板：



* 测试

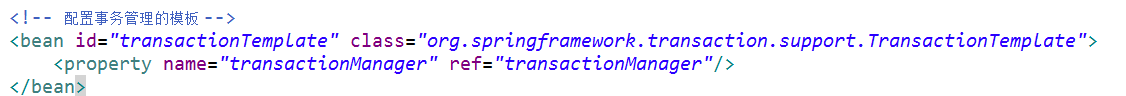
### Spring的事务管理：一类：编程式事务（需要手动编写代码）--了解

#### 第一步：配置平台事务管理器

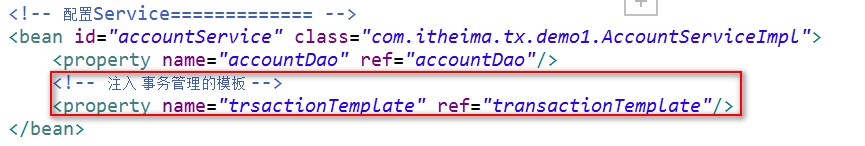


#### 第二步：Spring提供了事务管理的模板类

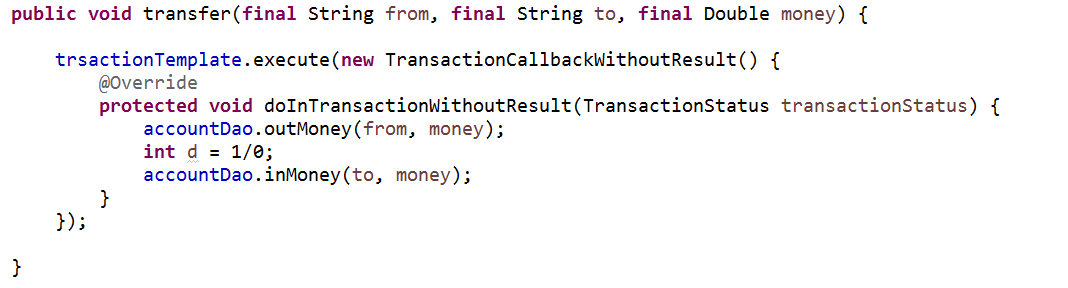
* 配置事务的管理的模板类



#### 第三步：在业务层注入事务管理的模板



#### 编写事务管理的代码

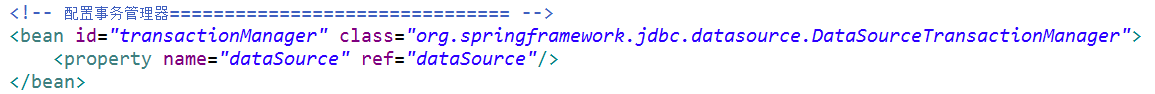


#### 测试：

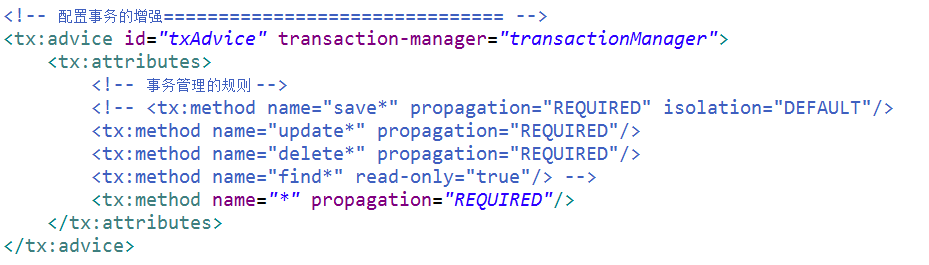
### Spring的事务管理：二类：声明式事务管理（通过配置实现）---AOP

#### XML方式的声明式事务管理

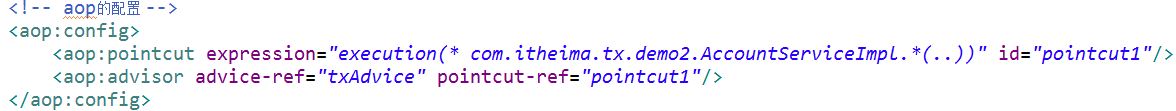
* 第一步：引入aop的开发包
* 第二步：恢复转账环境
* 第三步：配置事务管理器



* 第四步：配置增强



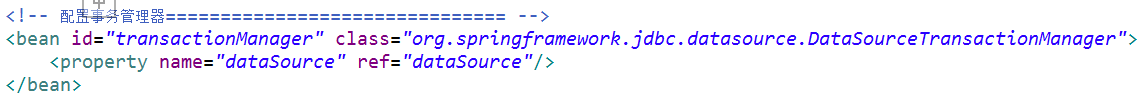
* 第五步：AOP的配置



* 测试

#### 注解方式的声明式事务管理

* 第一步：引入aop的开发包
* 第二步：恢复转账环境
* 第三步：配置事务管理器



* 第四步：开启注解事务



* 第五步：在业务层添加注解

