第3天: Spring

一、目标

1. 理解AOP相关概念和术语
2. 理解AspectJ表达式语言
3. 编写AOP的通知代码
4. 掌握AOP的注解实现
5. 能够说出事务的ACID原则
6. 能够说出spring的事务管理的方式和常用接口
7. 能够理解事务的隔离级别
8. 能够理解事务的传播行为
9. 能够应用声明式事务
10. 基于aop配置的事务控制案例
11. 基于aop注解的事务控制案例
12. 能够应用编程式事务

二、AOP概念与术语

3.1 AOP概念

- AOP (Aspect Oriented Programming) 即面向切面编程
- 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

3.2 AOP的作用

- 非侵入式编程: 在不修改源码的情况下对已有方法进行**增强**
- 提高代码复用: 增强的内容抽象成方法或者对象可**重复使用**
- 统一管理维护: 抽象成**独立**的方法或对象方便后期维护管理

3.3 AOP的原理

- Spring AOP 实现的原理是动态代理技术
- 底层支持两种动态代理
    - 当目标实现接口时采用JDK动态代理
    - 当目标没有实现接口采用Cglib动态代理 (可配置统一使用Cglib)

3.4 AOP相关术语

- 连接点（Joinpoint）

> 程序执行的某个特定位置：如类开始初始化前、类初始化后、类某个方法调用前、调用后、方法抛出异常后。一个类或一段程序代码拥有一些具有边界性质的特定点，这些点中的特定点就称为“连接点”。Spring仅支持方法的连接点，即仅能在方法调用前、方法调用后、方法抛出异常时以及方法调用前后这些程序执行点织入增强。连接点由两个信息确定：第一是用方法表示的程序执

- 切点（Pointcut）

> 每个程序类都拥有多个连接点，如一个拥有两个方法的类，这两个方法都是连接点，即连接点是程序类中客观存在的事物。AOP通过“切点”定位特定的连接点。连接点相当于数据库中的记录，而切点相当于查询条件。切点和连接点不是一对一的关系，一个切点可以匹配多个连接点。在Spring中，切点通过org.springframework.aop.Pointcut接口进行描述，它使用类和方法作为连接点的查询条件，Spring AOP的规则解析引擎负责切点所设定的查询条件，找到对应的连接点。其实确切地说，不能称之为查询连接点，因为连接点是方法执行前、执行后等包括方位信息的具体程序执行点，而切点只定位到某个方法上，所以如果希望定位到具体连接点上，还需要提供方位信息。

- 通知（Advice）

> 通知也叫增强，增强是织入到目标类连接点上的一段程序代码，在Spring中，增强除用于描述一段程序代码外，还拥有另一个和连接点相关的信息，这便是执行点的方位。结合执行点方位信息和切点信息，我们就可以找到特定的连接点。

- 目标对象（Target）

> 增强逻辑的织入目标类。如果没有AOP，目标业务类需要自己实现所有逻辑，而在AOP的帮助下，目标业务类只实现那些非横切逻辑的程序逻辑，而性能监视和事务管理等这些横切逻辑则可以使用AOP动态织入到特定的连接点上。

- 引介（Introduction）

> 引介是一种特殊的增强，它为类添加一些属性和方法。这样，即使一个业务类原本没有实现某个接口，通过AOP的引介功能，我们可以动态地为该业务类添加接口的实现逻辑，让业务类成为这个接口的实现类。

- 织入（Weaving）

> 织入是将增强添加对目标类具体连接点上的过程。AOP像一台织布机，将目标类、增强或引介通过AOP这台织布机天衣无缝地编织到一起。根据不同的实现技术，AOP有三种织入的方式：
>
> a、编译期织入，这要求使用特殊的Java编译器。
>
> b、类装载期织入，这要求使用特殊的类装载器。
>
> c、动态代理织入，在运行期为目标类添加增强生成子类的方式。
>
> Spring采用动态代理织入，而AspectJ采用编译期织入和类装载期织入。

- 代理（Proxy）

> 一个类被AOP织入增强后，就产出了一个结果类，它是融合了原类和增强逻辑的代理类。根据不同的代理方式，代理类既可能是和原类具有相同接口的类，也可能就是原类的子类，所以我们可以采用调用原类相同的方式调用代理类。

> 切面由切点和增强（引介）组成，它既包括了横切逻辑的定义，也包括了连接点的定义，Spring AOP就是负责实施切面的框架，它将切面所定义的横切逻辑织入到切面所指定的连接点中。

三、AOP配置案例

- 使用Spring AOP实现自动记录日志的功能

4.1 基于XML的AOP配置

4.1.1 环境搭建

- 工程名称: spring-day02-xml
- pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>spring3</artifactId>
        <groupId>com.itheima</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>spring-day03-xml</artifactId>

    <dependencies>
        <!-- Spring IOC 依赖以及Cglib支持 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <!-- AspectJ切面表达式支持 -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.8.13</version>
        </dependency>
        <!-- 环境测试 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
```

- AccountServiceImpl.java

```java
package com.itheima.xml.impl;

import com.itheima.xml.AccountService;

/**
 * 账户业务实现类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
public class AccountServiceImpl implements AccountService {
    @Override
    public void save() {
        System.out.println("保存账户");
    }
}
```

- logAdvice.java

```java
package com.itheima.xml.advice;

/**
 * 记录日志功能的通知/切面类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
public class LogAdvice {

    public void log(){
        System.out.println("记录操作日志..");
    }
}
```

- applicationContext.xml

```xml
<?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">

    <!-- 创建AccountServiceImpl对象并加入到IOC容器 -->
    <bean id="accountService"
class="com.itheima.xml.impl.AccountServiceImpl"/>

    <!-- 创建并加入IOC容器 -->
    <bean id="logAdvice" class="com.itheima.xml.advice.LogAdvice"/>
</beans>
```

```
// 打印对象的字节码
// class com.itheima.xml.impl.AccountServiceImpl
System.out.println(accountService.getClass());
```

4.1.2 AOP配置

- applicationContext.xml

```
<?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: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/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 创建AccountServiceImpl对象并加入到IOC容器 -->
    <bean id="accountService"
class="com.itheima.xml.impl.AccountServiceImpl"/>


    <!-- 创建并加入IOC容器 -->
    <bean id="logAdvice" class="com.itheima.xml.advice.LogAdvice"/>

    <!--
        1. 导入aop命名空间和约束文件:
            xmlns:aop="http://www.springframework.org/schema/aop"
            http://www.springframework.org/schema/aop
            https://www.springframework.org/schema/aop/spring-aop.xsd"
        2. 声明AOP配置: <aop:config>..
        3. 配置切面: <aop:aspect..
            id: 切面(唯一)标识; ref: 通知/切面的引用(bean的id);
        4. 配置切入点: <aop:pointcut..
            id: 切入点唯一标识; expression: 指定切入点表达式;
            表达式组成: [修饰符] 返回值类型 包名.类名.方法名(参数)
            execution(modifiers-pattern? ret-type-pattern declaring-type-
pattern?name-pattern(param-pattern) throws-pattern?)
        5. 配置通知与切入点关系: <aop:after/before/after-throwing/after-
returning..
            method: 通知方法名称; pointcut-ref: 切入点唯一标识;
    -->
    <aop:config>
        <!-- 切面 -->
        <aop:aspect id="logAspect" ref="logAdvice">
            <!-- 切入点 -->
            <aop:pointcut id="pt" expression="execution(public void
com.itheima.xml.impl.AccountServiceImpl.save())"/>
            <!-- 通知与切入点关系 -->
            <aop:after method="log" pointcut-ref="pt"/>
        </aop:aspect>
    </aop:config>
</beans>
```

```java
import com.itheima.xml.AccountService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * AOP XML测试类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:applicationContext.xml")
public class XmlTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testAop (){
        // 打印对象的字节码
        // class com.sun.proxy.$Proxy15
        System.out.println(accountService.getClass());
        accountService.save();
    }
}
```

4.2 切入点表达式

- 切入点表达式: 定位切入点的规则
- 通用符号说明: 表达式中的特殊符号
  - `*`: 通配符, 表示1个或多个
  - `.`: 路径符, 表示包以及子包
  - `..`: 参数符, 表示参数可以0个可以1个也可以多个

- 指示符: AspectJ pointcut designators (PCD)

指示符	作用
execution	匹配表达式指定的所有方法执行的连接点
within	匹配指定一个或多个类型的方法
bean	匹配指定类型的方法

4.2.1 execution

execution(void com.itheima..AccountServiceImpl.save())	匹配方法
execution(void com.itheima..AccountServiceImpl.save(Integer))	方法参数必须为整型
execution(* *(..))	万能配置

- execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?name-pattern(param-pattern) throws-pattern?)
- **modifiers-pattern**: 权限访问修饰符 (可填)
- **ret-type-pattern**: 返回值类型 (必填)
- **declaring-type-pattern**: 全限定类名 (可填)
- **name-pattern**: 方法名 (必填)
- **param-pattern**: 参数名 (必填)
- **throws-pattern**: 异常类型 (可填)

4.2.2 within

> within表达式也是应用于类级别,实现粗粒度的(对类中的所有方法进行)控制的，指定相关类时，需要指定具体包名。

within(com.itheima..AccountServiceImpl)	指定一个类
within(com.itheima.*)	只包含当前目录下的类
within(com.itheima..*)	包含当前目录及所有子目录下的类

4.2.3 bean

> 使用bean表达式针对类中所有方法(粗粒度的切入点)定义

bean(accountService)	匹配一个对象
bean(*Service)	匹配一类对象
bean(*)	包含当前目录及所有子目录下的类

4.3 常用标签说明

4.3.1 <aop:config

- 声明AOP配置

4.3.2 <aop:aspect

- 配置切面
  - id: 切面名称/唯一标识
  - ref: 通知对象的引用

4.3.3 <aop:pointcut

- id: 切入点表达式名称唯一标识
- expression: 切入点表达式

4.3.4 <aop:before

- 配置前置通知
  - method: 指定通知方法的名称
  - pointcut: 定义切入点表达式
  - pointcut-ref: 引用切入点表达式

4.3.5 <aop:after-returning

- 配置后置通知
  - method: 指定通知方法的名称
  - pointcut: 定义切入点表达式
  - pointcut-ref: 引用切入点表达式

4.3.6 <aop:after-throwing

- 配置异常通知
  - method: 指定通知方法的名称
  - pointcut: 定义切入点表达式
  - pointcut-ref: 引用切入点表达式

4.3.7 <aop:after

- 配置最终通知
  - method: 指定通知方法的名称
  - pointcut: 定义切入点表达式
  - pointcut-ref: 引用切入点表达式

4.3.8 <aop:around

- 配置环绕通知
  - method: 指定通知方法的名称
  - pointcut: 定义切入点表达式
  - pointcut-ref: 引用切入点表达式

4.4 通知

4.4.1 通知类型

- 伪代码

```
try{
    [前置通知]
    执行目标对象方法..
    [后置通知]
}catch(Exception e){
    [异常通知]
}finally{
    [最终通知]
}
```

- 最终通知: 无论目标方法正常执行还是发生异常都会执行
- 环绕通知: 在目标方法执行的任意时间点自由选择执行 (通用)

4.4.2 通知案例

- 改造: LogAdvice.java

```java
package com.itheima.xml.advice;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.Signature;

/**
 * 记录日志功能的通知/切面类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
public class LogAdvice {

    public void log(){
        System.out.println("记录操作日志..");
    }

    public void before(){
        System.out.println("【前置通知】..");
    }

    public void afterReturning(){
        System.out.println("【后置通知】..");
    }

    public void afterThrowing(){
        System.out.println("【异常通知】..");
    }

    public void after(){
        System.out.println("【最终通知】..");
    }

    /**
     * 环绕通知
     * @param pjp 正在执行的连接点对象 (目标方法)
     * @return 目标方法执行后的返回值
     * @throws Throwable 目标方法执行发生的异常
     * JoinPoint:
     *  getArgs(): 返回方法参数。
     *  getThis(): 返回代理对象。
     *  getTarget(): 返回目标对象。
     *  getSignature(): 返回正在执行的方法的描述。
     *  toString(): 打印通知方法的有用说明。
     */
    public Object around(ProceedingJoinPoint pjp) throws Throwable {
```

```
        try {
            // 前置通知
            before();
            Object ret = pjp.proceed(args);// 执行目标方法
            // 后置通知
            afterReturning();
            return ret;
        } catch (Exception e){
            // 异常通知
            afterThrowing();
            throw e;
        } finally {
            // 最终通知
            after();
        }
    }
}
```

- advice.xml

```
<?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: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/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 创建AccountServiceImpl对象并加入到IOC容器 -->
    <bean id="accountService"
class="com.itheima.xml.impl.AccountServiceImpl"/>


    <!-- 创建并加入IOC容器 -->
    <bean id="logAdvice" class="com.itheima.xml.advice.LogAdvice"/>

    <aop:config>
        <aop:aspect id="logAspect" ref="logAdvice">
            <aop:pointcut id="pt" expression="execution(* *())"/>
            <!-- 前置通知 -->
            <aop:before method="before" pointcut-ref="pt"/>
            <!-- 后置通知 -->
            <aop:after-returning method="afterReturning" pointcut-
ref="pt"/>
            <!-- 异常通知 -->
            <aop:after-throwing method="afterThrowing" pointcut-ref="pt"/>
            <!-- 最终通知 -->
            <aop:after method="after" pointcut-ref="pt"/>
            <!-- 环绕通知 -->
            <!--<aop:around method="around" pointcut-ref="pt"/>-->
        </aop:aspect>
    </aop:config>
</beans>
```

```java
import com.itheima.xml.AccountService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * AOP XML测试类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
//@ContextConfiguration(locations = "classpath:applicationContext.xml") // AOP案例配置
@ContextConfiguration(locations = "classpath:advice.xml") // AOP通知案例配置
public class XmlTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testAop (){
        // 打印对象的字节码
        System.out.println(accountService.getClass());
        accountService.save();
    }
}
```

四、AOP注解改造案例

5.1 使用注解改造案例

5.1.1 改造工程

- 工程名称: spring-day03-anno

5.2.2 改造通知类

- LogAdvice.java

```java
package com.itheima.xml.advice;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.Signature;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;

/**
 * @Component: 创建对象并添加到IOC容器
 * @Aspect: 定义对象为切面/通知类
 *  value: 名称
```

```java
@Aspect
public class LogAdvice {

    /**
     * @Pointcut: 定义切入点表达式
     */
    @Pointcut("execution(* *(..))")
    public void pt(){

    }

    /**
     * @Before: 定义前置通知和切入点的关系
     *  value: 切入点表达式或者对切入点表达式的引用
     */
    @Before("pt()")
    public void before(){
        System.out.println("【前置通知】..");
    }


    /**
     * @Before: 定义后置通知和切入点的关系
     *  value: 切入点表达式或者对切入点表达式的引用
     */
    @AfterReturning("pt()")
    public void afterReturning(){
        System.out.println("【后置通知】..");
    }


    /**
     * @Before: 定义异常通知和切入点的关系
     *  value: 切入点表达式或者对切入点表达式的引用
     */
    @AfterThrowing("pt()")
    public void afterThrowing(){
        System.out.println("【异常通知】..");
    }


    /**
     * @Before: 定义最终通知和切入点的关系
     *  value: 切入点表达式或者对切入点表达式的引用
     */
    @After("pt()")
    public void after(){
        System.out.println("【最终通知】..");
    }


    /**
     * @Before: 定义环绕通知和切入点的关系
     *  value: 切入点表达式或者对切入点表达式的引用
     */
```

```
        Object[] args = pjp.getArgs();
        // 实现环绕通知
        try {
            // 前置通知
            before();
            Object ret = pjp.proceed(args);// 执行目标方法
            // 后置通知
            afterReturning();
            return ret;
        } catch (Exception e){
            // 异常通知
            afterThrowing();
            throw e;
        } finally {
            // 最终通知
            after();
        }
    }
}
```

5.1.3 改造配置

- applicationContext.xml

```
<?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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/aop
       https://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.itheima.xml"/>

    <!-- 开启自动代理
            proxy-target-class: 是否通用Cglib代理
     -->
    <aop:aspectj-autoproxy proxy-target-class="true"/>
</beans>
```

5.1.4 单元测试

- AnnoTests.java

```
import com.itheima.xml.AccountService;
import org.junit.Test;
```

```java
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * AOP XML测试类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:applicationContext.xml")
public class AnnoTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testAop (){
        // 打印对象的字节码
        System.out.println(accountService.getClass());
        accountService.save();
    }
}
```

五、事务的概念

2.1 事务定义

> 事务（transaction），一般是指要做的或者所做的事情。在程序中，尤其是在操作数据库的程序中，指的是访问并且可能更新数据库中数据项的一个执行单元（unit），这个执行单元由事务开始（begin transaction）和事务结束（end transaction）之间执行的全部操作组成。

2.2 ACID原则

- 事务具有4个基本特性：原子性、一致性、隔离性、持久性。也就是我们常说的ACID原则。

2.2.1 原子性（Atomicity）

- 一个事务已经是一个不可再分割的工作单位。事务中的全部操作要么都做；要么都不做。

2.2.2 一致性（Consistency）

- 事务必须是使得数据库状态从一个一致性状态，转变到另外一个一致性状态。也就是说在事务前，和事务后，被操作的目标资源状态一致。比如银行转账案例中，转账前和转账后，总账不变。

2.2.3 隔离性（Isolation）

- 一个事务的执行不能被其他事务的影响。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，多个并发事务之间不能相互干扰。

2.2.4 持久性（Durability）

- 一个事务一旦提交，它对数据库中数据的改变会永久存储起来，其他操作不会对它产生影响

六、Spring声明式事务管理

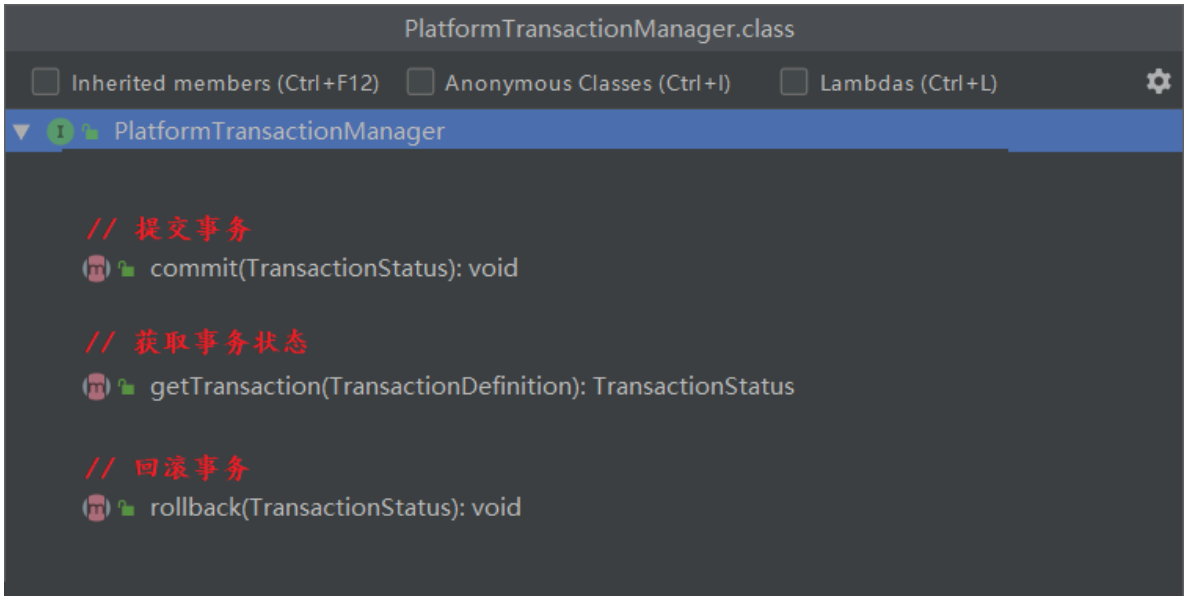
3.1 前言

- JavaEE 体系进行分层开发，事务处理位于**业务层**， Spring 提供了分层设计业务层的事务处理解决方案。
- spring 框架为我们提供了一组事务控制的接口(**API**)。这组接口是在spring-tx-5.x.RELEASE.jar 中。
- spring 的事务控制都是基于 AOP 的，它既可以使用编程的方式实现，也可以使用 **配置的方式**实现。

3.2 API

3.2.1 PlatformTransactionManager

- 事务的顶层接口，定义了：提交事务、获取事务、回滚事务的方法。



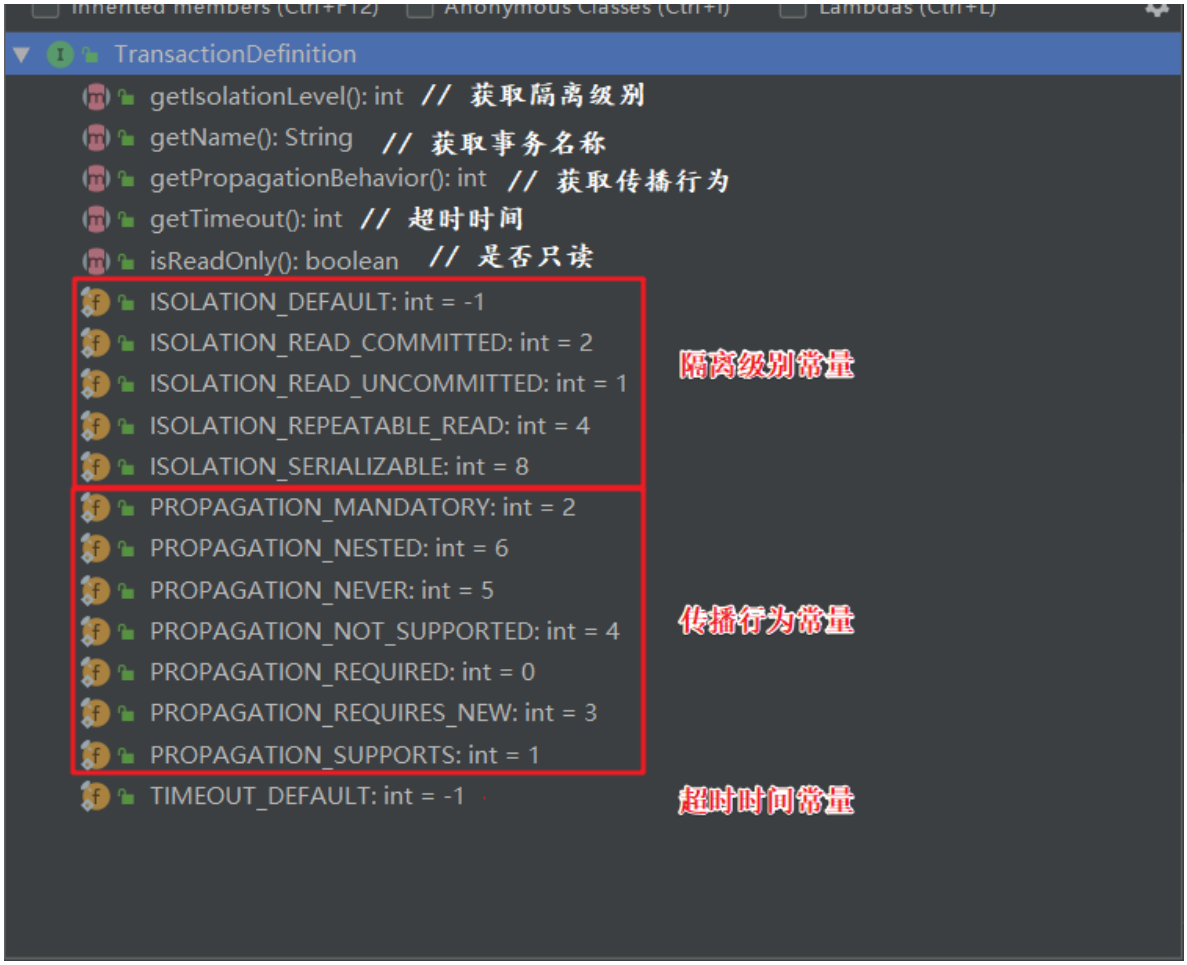
> 它是一个接口，是spring的事务管理器核心接口，spring并不实现具体的事务，而是负责包装底层事务，提供规范。应用底层支持什么样的事务策略，spring就支持什么样的事务策略。该接口提供了我们操作事务的常用方法。

- org.springframework.jdbc.datasource.DataSourceTransactionManager

> 支持使用spring jdbc 或者 Mybatis框架的事务管理器

3.2.2 TransactionDefinition

- 定义了事务的隔离级别、传播行为、超时时间等。



隔离级别

- ISOLATION_DEFAULT

  数据库默认级别，归属于下列某一种隔离级别。

- ISOLATION_READ_UNCOMMITTED

  可以读取其他事务未提交的数据（**脏读**）。

- ISOLATION_READ_COMMITTED

  只读取其他事务已经提交的数据，解决脏读的问题; **不可重复读**（oracle数据库默认级别）。

- ISOLATION_REPEATABLE_READ

  读取其他事务提交修改后的数据, **可重复读**（mysql数据库默认级别）。

- ISOLATION_SERIALIZABLE

  读取其他事务添加后的数据，**解决幻读** 的问题。

> 事务指定一个隔离级别，该隔离级别定义一个事务必须与由其他事务进行的资源或数据更改相隔离的程度。隔离级别从允许的并发副作用（例如，脏读 或 幻读）的角度进行描述。
>
> 1. 事务级别从低到高：脏读->不可重复读->可重复读->解决幻读
> 2. 事务级别越高，数据越安全，消耗的资源越多，数据库操作性能越低

传播行为

- REQUIRED

- SUPPORTS

如果已经有事务，支持当前事务的执行；如果没有事务，就以非事务的方式执行。对应查询操作

- MANDATORY

要求在事务环境下执行，如果当前没有事务，则抛出异常

- REQUIRES_NEW

新建事务，如果当前已经存在事务，则把当前事务挂起

- NOT_SUPPORTED

以非事务方式执行，如果当前有事务，则把当前事务挂起

- NEVER

以非事务方式执行，如果当前有事务，则抛出异常

- NESTED

嵌套事务方式执行，如果当前已经有事务，**嵌套** 在当前事务中执行；如果当前没有事务，则类似于REQUIRED操作; 其中嵌套事务与REQUIRED不同, REQUIRED与外部方法是同属一个事务, 而NESTED是外部方法的子事务: **NESTED方法抛出异常被回滚，不会影响到外围方法的事务。**

> 事务传播行为用来描述由某一个事务传播行为修饰的方法被嵌套进另一个方法的时事务如何传播。

超时时间

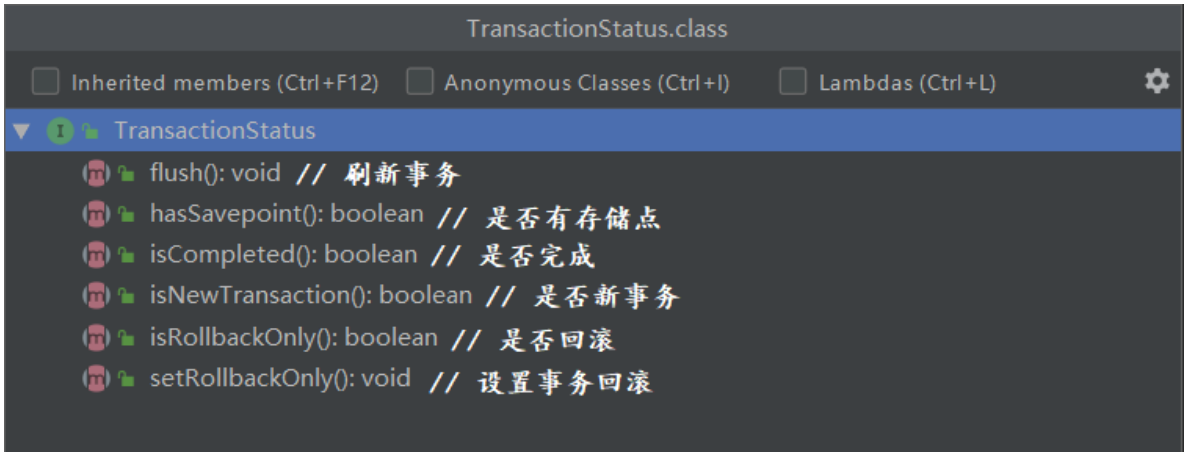
> 事务未响应的时间, 超过一定时间事务将回滚。

是否只读

> 只读事务比读写事务性能要高，所以查询类的只读操作一般设置为只读。

3.2.3 TransactionStatus

- 事务状态（Spring声明式事务内部实现规范的定义）



3.3 XML案例

3.3.1 搭建环境

- 工程名称: spring-day04-xml
- 添加依赖: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>spring4</artifactId>
        <groupId>com.itheima</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>spring-day04-xml</artifactId>

    <dependencies>
        <!-- Spring IOC 依赖 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <!-- 切面编程 依赖 -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.8.13</version>
        </dependency>
        <!-- Spring 事务 依赖 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <!-- Spring  Jdbc 依赖 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <!-- 连接池 依赖 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.12</version>
        </dependency>
        <!-- Mysql 依赖 -->
        <dependency>
            <groupId>mysql</groupId>
```

```
        <!-- Spring测试 依赖 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>5.1.7.RELEASE</version>
        </dependency>
        <!-- 单元测试 依赖 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>
</project>
```

3.3.2 业务代码

- com.itheima.xml.dao.impl.AccountDaoImpl.java

```
package com.itheima.xml.dao.impl;

import com.itheima.xml.dao.AccountDao;
import com.itheima.xml.domain.Account;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

/**
 * 账户数据中心.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Repository
public class AccountDaoImpl implements AccountDao {

    @Autowired
    JdbcTemplate jdbcTemplate;

    @Override
    public void save(Account account) {
        String sql = "update account set uid=?, money=? where id=?";
        jdbcTemplate.update(sql, account.getUid(), account.getMoney(),
account.getId());
    }
}
```

- com.itheima.xml.service.impl.AccountServiceImpl.java

```
package com.itheima.xml.service.impl;

import com.itheima.xml.dao.AccountDao;
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
```

```java
/**
 * 账户业务中心.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    AccountDao accountDao;

    @Override
    public void save(Account account) {
        accountDao.save(account);
        int i = 1/0;
    }
}
```

3.3.3 事务配置

- db.properties

```properties
db.driver=com.mysql.cj.jdbc.Driver
# 针对Mysql 8.x数据库的参数
#   serverTimezone: 指定时区(UTC)
#   useSSL: 指定是否使用加密安全连接(false)
#   allowPublicKeyRetrieval: 是否允许检索公钥(true)
db.url=jdbc:mysql:///mybatisdb?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true
db.username=root
db.password=root
```

- applicationContext.xml

```xml
<?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:tx="http://www.springframework.org/schema/tx"
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
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 加载配置文件 -->
    <context:property-placeholder location="classpath:db.properties"/>

    <!-- 使用注解需要指定扫描路包径 -->
```

```
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${db.driver}"/>
        <property name="url" value="${db.url}"/>
        <property name="username" value="${db.username}"/>
        <property name="password" value="${db.password}"/>
    </bean>

    <!-- 创建JdbcTemplate对象并添加至IOC容器 -->
    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 配置事务管理器 (切面类/通知类) -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 通知规则 (方法拦截到之后事务管理规则) -->
    <tx:advice id="txAdvice">
        <tx:attributes>
            <tx:method name="save*" propagation="REQUIRED" read-
only="false"/>
            <tx:method name="find*" propagation="SUPPORTS" read-
only="true"/>
        </tx:attributes>
    </tx:advice>

    <!-- 配置AOP (拦截方法添加通知规则)
            aop:advisor 通知与切入点的组合关系映射
     -->
    <aop:config>
        <aop:pointcut id="pt" expression="execution(* com.itheima..*.*
(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pt"/>
    </aop:config>
</beans>
```

3.3.4 单元测试

- XmlTests.java

```
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * 单元测试.
 *
 * @author : Jason.lee
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:applicationContext.xml")
public class XmlTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testTransaction (){
        // 查看代理对象字节码
        System.out.println(accountService.getClass());
        Account account = new Account();
        account.setUid(1);
        account.setMoney(10.0);
        accountService.save(account);
    }
}
```

3.4 注解案例

3.4.1 环境改造

- 工程名称: spring-day04-anno

3.4.2 业务代码

- com.itheima.xml.service.impl.AccountServiceImpl.java

```
package com.itheima.xml.service.impl;

import com.itheima.xml.dao.AccountDao;
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

/**
 * 账户业务中心.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    AccountDao accountDao;

    /**
     * @Transactional: 使用transactionManager管理事务
```

```
     *  isolation = Isolation.DEFAULT 默认值，默认级别
     *  propagation = Propagation.REQUIRED 默认值，表示当前执行的方法必须有事务环境。
     *  rollbackFor = {} 遇到指定的异常回滚
     *  noRollbackFor = {} 遇到指定的异常不回滚
     *      【与rollbackFor共同使用】：以rollbackFor为准
     *
     */
    @Override
    @Transactional(readOnly = false, timeout = -1,
            isolation = Isolation.DEFAULT,
            propagation = Propagation.REQUIRED,
            rollbackFor = { RuntimeException.class },
            noRollbackFor = { ArithmeticException.class }
            )
    public void save(Account account) {
        accountDao.save(account);
        int i = 1/0;
    }
}
```

3.4.3 配置支持

- applicationContext.xml

```
<?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:tx="http://www.springframework.org/schema/tx"
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
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 加载配置文件 -->
    <context:property-placeholder location="classpath:db.properties"/>

    <!-- 使用注解需要指定扫描路包径 -->
    <context:component-scan base-package="com.itheima.xml"/>

    <!-- 配置数据源（使用druid连接池作为数据源） -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${db.driver}"/>
        <property name="url" value="${db.url}"/>
        <property name="username" value="${db.username}"/>
        <property name="password" value="${db.password}"/>
    </bean>

    <!-- 创建JdbcTemplate对象并添加至IOC容器 -->
```

```
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 配置事务管理器 (切面类/通知类) -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 开启声明式事务的注解支持 (使用@Transactional代替AOP配置) -->
    <tx:annotation-driven/>
</beans>
```

3.4.4 单元测试

- AnnoTests.java

```
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * 单元测试.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:applicationContext.xml")
public class AnnoTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testTransaction (){
        // 查看代理对象字节码
        System.out.println(accountService.getClass());
        Account account = new Account();
        account.setUid(1);
        account.setMoney(10.0);
        accountService.save(account);
    }
}
```

3.5 纯注解改造

3.5.1 注解改造

- com.itheima.xml.config.SpringConfig.java

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import
org.springframework.transaction.annotation.EnableTransactionManagement;

/**
 * Spring配置类.
 * @EnableTransactionManagement:
 *  修饰类，开启声明式事务注解支持代替  <tx:annotation-driven/>
 */
@Configuration
@ComponentScan("com.itheima.xml")
@Import(JdbcConfig.class)
@EnableTransactionManagement
public class SpringConfig {
}
```

- com.itheima.xml.config.JdbcConfig.java

```
package com.itheima.xml.config;

import com.alibaba.druid.pool.DruidDataSource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.PropertySource;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import javax.sql.DataSource;

/**
 * Jdbc配置类.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@PropertySource("classpath:db.properties")
public class JdbcConfig {
    @Value("${db.driver}")
    private String driver;
    @Value("${db.url}")
    private String url;
    @Value("${db.username}")
    private String username;
    @Value("${db.password}")
    private String password;

    @Bean
    public DataSource dataSource(){
        DruidDataSource ds = new DruidDataSource();
        ds.setDriverClassName(driver);
        ds.setUrl(url);
        ds.setUsername(username);
```

```
38
39      @Bean
40      public JdbcTemplate jdbcTemplate(DataSource dataSource){
41          return new JdbcTemplate(dataSource);
42      }
43
44      @Bean
45      public DataSourceTransactionManager
dataSourceTransactionManager(DataSource dataSource){
46          return new DataSourceTransactionManager(dataSource);
47      }
48  }
```

3.5.2 单元测试

- AnnoTests.java

```
1   import com.itheima.xml.config.JdbcConfig;
2   import com.itheima.xml.config.SpringConfig;
3   import com.itheima.xml.domain.Account;
4   import com.itheima.xml.service.AccountService;
5   import org.junit.Test;
6   import org.junit.runner.RunWith;
7   import org.springframework.beans.factory.annotation.Autowired;
8   import org.springframework.test.context.ContextConfiguration;
9   import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10
11  /**
12   * 单元测试.
13   *
14   * @author : Jason.lee
15   * @version : 1.0
16   */
17  @RunWith(SpringJUnit4ClassRunner.class)
18  //@ContextConfiguration(locations = "classpath:applicationContext.xml")
19  @ContextConfiguration(classes = {SpringConfig.class})
20  public class AnnoTests {
21
22      @Autowired
23      AccountService accountService;
24
25      @Test
26      public void testTransaction (){
27          // 查看代理对象字节码
28          System.out.println(accountService.getClass());
29          Account account = new Account();
30          account.setUid(1);
31          account.setMoney(10.0);
32          accountService.save(account);
33      }
34  }
```

4.1 概念

- Spring提供了两种事务管理机制: 编程式事务指的是事务需要通过编码来实现的方式

4.2 API

4.2.1 TransactionTemplate

> 它是事务管理模版类，继承DefaultTransactionDefinition类。用于简化事务管理，事务管理由模板类定义，主要是通过TransactionCallback回调接口或TransactionCallbackWithoutResult回调接口指定，通过调用模板类的execute()方法来自动实现事务管理。

4.2.2 TransactionCallback

> 通过实现该接口的“T doInTransaction(TransactionStatus status) ”方法来定义需要事务管理的操作代码。适合于有返回值的目标方法。

4.2.3 TransactionCallbackWithoutResult

> 实现TransactionCallback接口，提供“void doInTransactionWithoutResult(TransactionStatus status)”。适合于不需要返回值的目标方法。

4.3 XML案例

4.3.1 环境改造

- 工程名称: spring-day04-code

4.3.2 业务代码

- com.itheima.xml.service.impl.AccountServiceImpl.java

```
package com.itheima.xml.service.impl;

import com.itheima.xml.dao.AccountDao;
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallback;
import org.springframework.transaction.support.TransactionTemplate;

/**
 * 账户业务中心.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    AccountDao accountDao;

    @Autowired
```

```
    public void save(Account account) {
        // 创建事务回调对象
        TransactionCallback callback = new TransactionCallback() {
            /**
             * 在事务中运行的代码
             * @param transactionStatus 当前事务状态
             * @return 回调返回值
             */
            @Override
            public Object doInTransaction(TransactionStatus
transactionStatus) {
                accountDao.save(account);
                int i = 1 / 0;
                return null;
            }
        };
        transactionTemplate.execute(callback);
    }
}
```

4.3.3 配置支持

- applicationContext.xml

```
<?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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 加载配置文件 -->
    <context:property-placeholder location="classpath:db.properties"/>

    <!-- 使用注解需要指定扫描路包径 -->
    <context:component-scan base-package="com.itheima.xml"/>

    <!-- 配置数据源（使用druid连接池作为数据源） -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${db.driver}"/>
        <property name="url" value="${db.url}"/>
        <property name="username" value="${db.username}"/>
        <property name="password" value="${db.password}"/>
    </bean>

    <!-- 创建JdbcTemplate对象并添加至IOC容器 -->
    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 配置事务管理器（切面类/通知类） -->
```

```
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 配置事务管理模板（代替切面编程） -->
    <bean id="transactionTemplate"
class="org.springframework.transaction.support.TransactionTemplate">
        <property name="transactionManager" ref="transactionManager"/>
    </bean>
</beans>
```

4.3.4 单元测试

- XmlTests.java

```
import com.itheima.xml.domain.Account;
import com.itheima.xml.service.AccountService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * 单元测试.
 *
 * @author : Jason.lee
 * @version : 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:applicationContext.xml")
public class XmlTests {

    @Autowired
    AccountService accountService;

    @Test
    public void testTransaction (){
        // 查看对象字节码（没有使用代理）
        // class com.itheima.xml.service.impl.AccountServiceImpl
        System.out.println(accountService.getClass());
        Account account = new Account();
        account.setUid(1);
        account.setMoney(10.0);
        accountService.save(account);
    }
}
```