# 第4天: Spring

## 一、目标

1. 能够说出事务的ACID原则
2. 能够说出spring事务管理常用接口
3. 能够理解事务的隔离级别
4. 能够理解事务的传播行为
5. 能够通过xml实现声明式事务控制
6. 能够通过注解实现声明式事务控制
7. 了解spring编程式事务控制

## 二、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。

## 三、Spring声明式事务管理

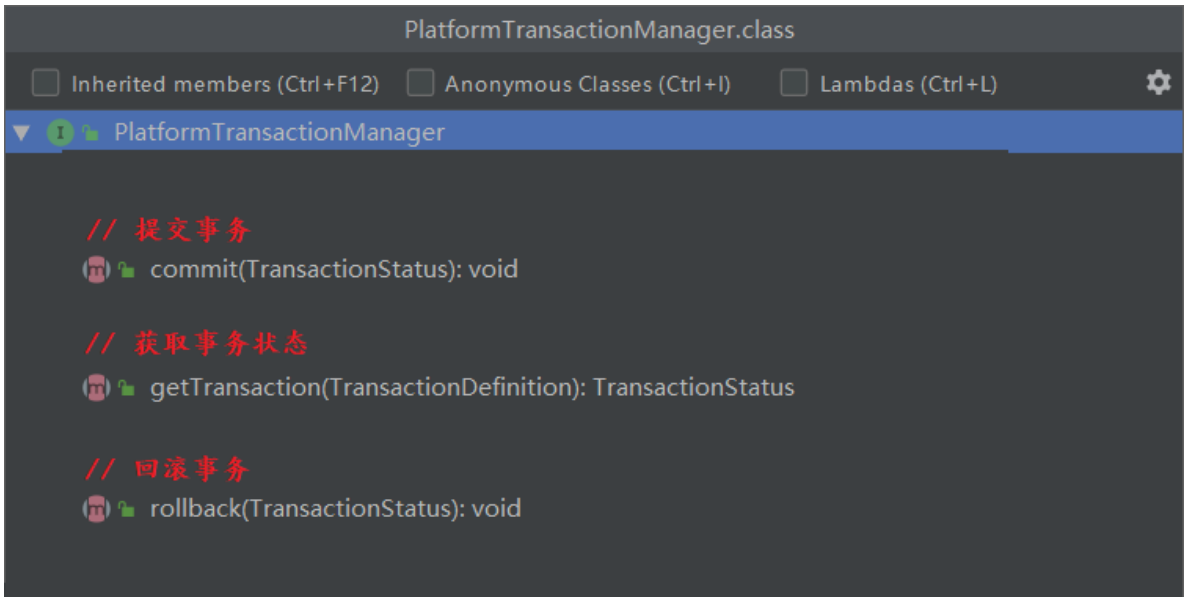
### 3.1 前言

- 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。

## 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。



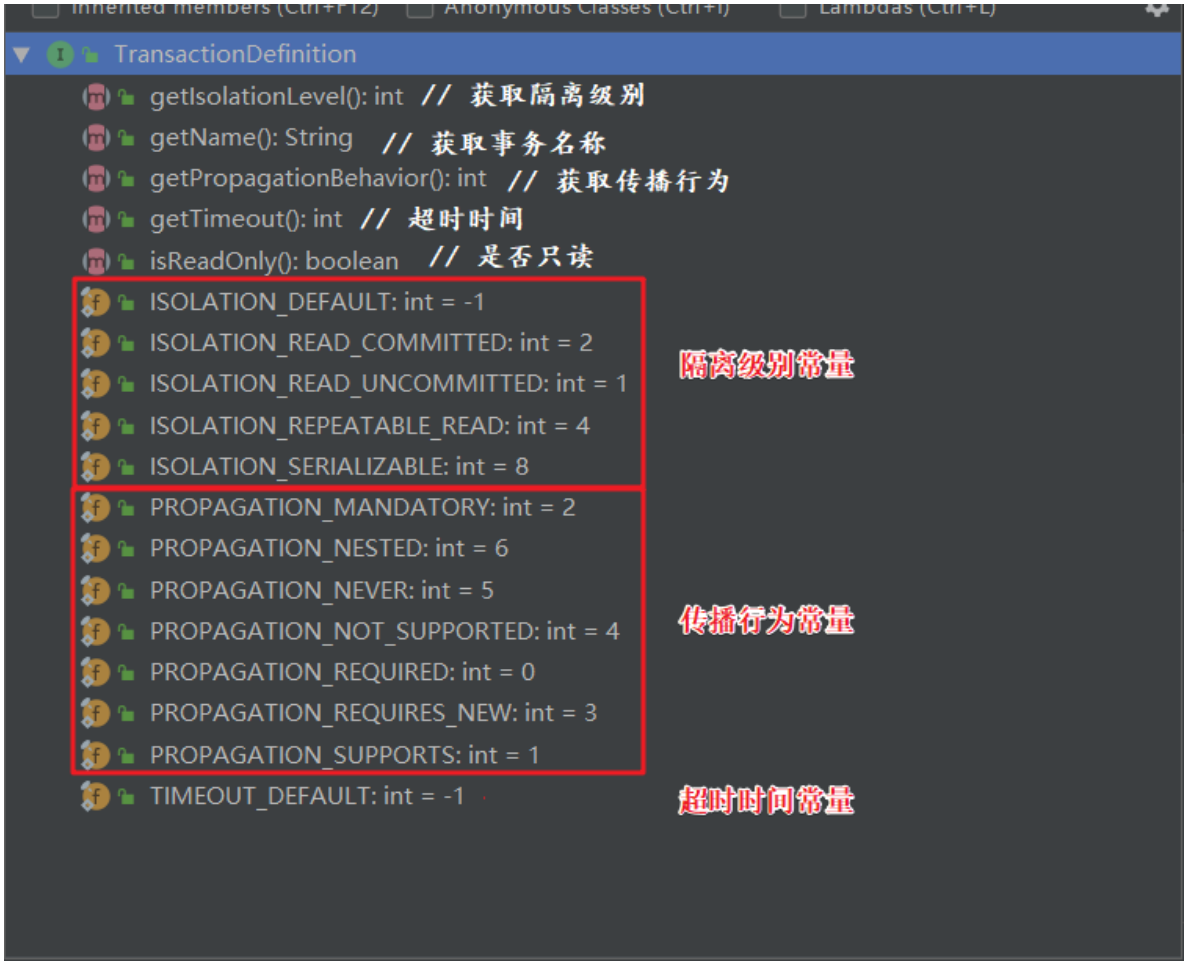

> 它是一个接口，是spring的事务管理器核心接口，spring并不实现具体的事务，而是负责包装底层事务，提供规范。应用底层支持什么样的事务策略，spring就支持什么样的事务策略。该接口提供了我们操作事务的常用方法。

- 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**

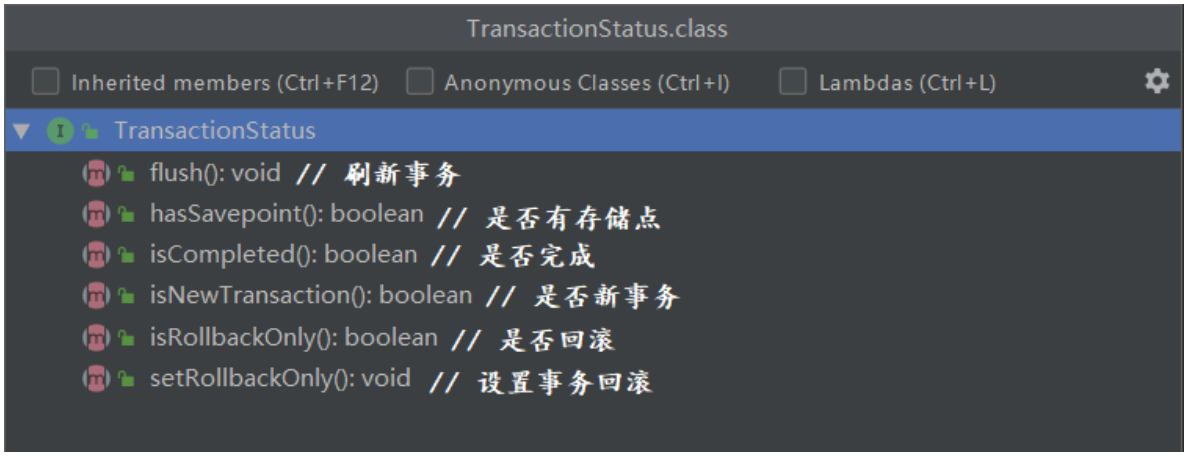
> 事务未响应的时间, 超过一定时间事务将回滚。

**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);
    }
}
```