# Spring之事务篇

## 事务的概念:

**指一系列的操作组成的一个整体,要么全部成功,要么全部失败**

举例:

张三转账给李四1000

1. 张三账户扣1000
2. 李四账户多1000

某业务系统新建一个用户:

1. 新建用户
2. 为用户分配初始角色

在张三刚扣1000后,或者用户建立完之后进行下一步操作时产生了错误

这个时候如果没有事务机制,那么张三的账户平白无故少了1000元,

或者是新建的那个用户因为身份证信息的唯一性而无法再次新建本用户.

## 事务的特性:

**事务四大特性(原子性,一致性,隔离性,持久性)**

### 原子性:

事务中的操作的不可分割性,要么都发生,要么操作都不发生

### 一致性:

事务执行前后数据的完整性必须保持一致(事务的划分原子操作的原则)

### 隔离性:

多个用户的事务同时访问数据库时,事务访问的数据不能被其他用户所干扰,多个并发事务之间的数据要互相隔离

#### 如果不考虑隔离性,会引发”脏读,不可重复读,幻读”

脏读:一个事务读取到了另一个事务还为提交的修改,如果后者被回滚,那么前一个事务读到的数据是无效的.

不可重复读:在同一事务中多次读取同一个数据,中途被别的事务修改过,导致前后读取的数据不相同

幻读:一个事务执行过程中读取到了,另一个事务实时插入的数据导致结果有误

#### 事务的隔离级别

**DEFAULT:**使用数据库自身的默认隔离级别,随着数据库的改变而改变

**READ\_UNCOMMITED:**事务执行过程中允许读取,其他事务已经修改,但是还未提交的数据,

可导致脏读,不可重复读,幻读

**READ\_COMMITED:** 事务执行过程中允许读取,其他事务已经提交的数据,

可导致不可重复读,幻读

**REPEATABLE\_READ:**对相同字段的多次读取是一致性的,除非数据被事务本身修改,

可防止脏读,不可重复读,但是幻读仍然有可能发生

**SERIALIZABLE:**完全隔离,在所有隔离级别中是相对最慢的,

它典型的通过完全锁定在事务中涉及的数据表来完成

### 持久性:

一个事务一旦被提交,数据在数据库中的改动就是永久的(不理解,提交了,不就是确定修改么)

## 事务的传播行为

(目前还不理解,先记下来)

|  |  |
| --- | --- |
| **PROPAGATION\_REQUIRED** | 支持当前事务,如果不存在就新建一个 |
| PROPAGATION\_SUPPORTS | 支持当前事务,如果不存在就不使用事务 |
| PROPAGATION\_MANDATORY | 支持当前事务,如果不存在就抛出异常 |
| **PROPAGATION\_REQUIRES\_NEW** | 如果当前事务存在,挂起当前事务,新建一个事务 |
| PROPAGATION\_NOT\_SUPPORTED | 以非事务方式运行,如果当前事务存在,挂起当前事务 |
| PROPAGATION\_NEVER | 以非事务方式运行,如果当前事务存在,抛出异常 |
| **PROPAGATION\_NESTED** | 如果当前事务存在,则嵌套事务执行 |

## 事务管理涉及的接口:

**PlatformTransactionManager:事务管理器**

|  |  |
| --- | --- |
| **Org.springframework.jdbc.datasource.DataSourceTransactionManager** | 使用SpringJDBC或者ibatis持久层时使用 |
| **Org.springframework.orm.hibernate3.HibernateTransactionManager** | 使用hibernate3.0时使用 |
| Org.springframework.orm.jpa.JpaTransactionManager | 使用jpa持久化 |
| Org.springframework.jdo.JdoTransactionManager | 使用jdo持久化机制 |
| Org.springframework.tansaction.jta.JtaTransactionManager | 使用jta实现事务管理,在一个事务跨多个资源时必须使用 |

上述接口主要:初阶,主要熟悉前面2个

**TransactionStatus:事务状态(目前好像没什么用到,但是为了保险起见还是记一下吧)**

方法:

flush()

boolean hashSavepoint()

boolean isCompleted()

boolean isNewTransaction()

boolean isRollbackOnly()

setRollBackOnly()

## 基于编程方式的事务管理

步骤:

<!-- 配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<!-- 配置事务管理模版 -->

<bean id=*"transactionTemplate"* class=*"org.springframework.transaction.support.TransactionTemplate"*>

<property name=*"transactionManager"* ref=*"transactionManager"*></property>

</bean>

Service层注入trasactionTemplate 对象

@Resource(name = "transactionTemplate")

**private** TransactionTemplate transactionTemplate;

@Override

**public** **void** transfer(**final** String toAccount, **final** String fromAccount,

**final** **double** money) {

transactionTemplate.execute(**new** TransactionCallbackWithoutResult() {

@Override

**protected** **void** doInTransactionWithoutResult(TransactionStatus transactionStatus) {

**int** i = accountDao.descMoney(fromAccount, money);

**if**(**true**){

**throw** **new** RuntimeException("测试异常");

}

**int** j = accountDao.addMoney(toAccount, money)

}

});

}