# 一 事务

事务是一组操作的执行单元，针对数据库操作，事务管理的是一组SQL指令，事务内的操作要不全部成功，要不全部失败。比如执行过程中，如果有一条SQL语句没有执行成功，那么这一组操作都将全部回滚

事务特性（ACID）：

Atomic(原子性):要么都成功，要么都失败

Consistent(一致性):数据不应该被破坏

Isolate(隔离性):用户间操作不相混淆

Durable(持久性):永久保存

## 1 编程式事务

编写程序式的事务管理可以清楚的定义事务的边界，可以实现细粒度的事务控制，比如你可以通过程序代码来控制你的事务何时开始，何时结束等，与后面介绍的声明式事务管理相比，它可以实现细粒度的事务控制，例如jdbc、hibernate，但是spring中不提倡使用。

## 2 声明式事务

如果你并不需要细粒度的事务控制，你可以使用声明式事务，在Spring中，你只需要在Spring配置文件中做一些配置，即可将操作纳入到事务管理中，解除了和代码的耦合， 这是对应用代码影响最小的选择，这一点再次验证了Spring关于AOP的概念。当你不需要事务管理的时候，可以直接从Spring配置文件中移除该设置

Spring使用声明式事务，结合AOP一起使用

Spring中的事务针对方法，一般将事务用在service层

# 二 事务配置

## 1 引入jar包

需要导入AOP相关jar文件，

还需要导入spring-tx-5.1.5.RELEASE.jar

## 2 引入名称空间

<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"*

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

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*

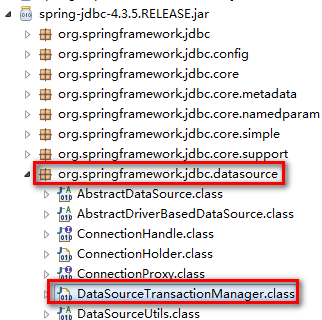
*http://www.springframework.org/schema/aop*

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

*http://www.springframework.org/schema/tx*

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

## 3 xml方式事务的配置



<!-- spring事务 -->

<!-- 1配置事务管理类 -->

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

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

</bean>

<!-- 2配置事务的增强 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"txManage"*>

<tx:attributes>

<!-- 针对使用事务的add开头的方法

read-only 是否只读，true 是，false 可读可写

如果有插入等操作，设为为true，运行程序会报异常-->

<tx:method name=*"getPersonDao"* propagation=*"NOT\_SUPPORTED"*/>

<tx:method name=*"add\*"* read-only=*"false"*/>

</tx:attributes>

</tx:advice>

<!-- 3AOP配置 -->

<aop:config>

<!-- 切入点 -->

<aop:pointcut expression=*"execution(\* com.rr.xml.PersonService.\*(..))"* id=*"pc"*/>

<!-- 通知 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pc"*/>

</aop:config>

## 4 注解方式

@Transactional 既可以修饰方法，也可以修饰类

//rollbackFor=Exception.class

//rollbackFor 表示哪些异常回滚

//noRollbackFor 表示某些异常下不回滚

//声明事务的注解

@Transactional(readOnly=**false**,//是否只读，如果只有查询，设置为true，效率高

propagation=Propagation.***REQUIRED***, //事务的传播特性

isolation=Isolation.***DEFAULT***, //事务的隔离特性

timeout=3,//事务的超时时间，单位秒，默认-1

noRollbackFor=ArithmeticException.**class** //回滚规则

)

**public** **void** addPerson(String name){}

<!-- 1配置事务管理类 -->

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

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

</bean>

<!-- 2 事务的注解扫描 -->

<tx:annotation-driven transaction-manager=*"txManage"*/>

事务的传播特性：多个事务方法相互调用时，事务如何在这些方法间传播

|  |  |
| --- | --- |
| **传播特性** | **意义** |
| REQUIRED | 业务方法需要在一个事务中运行。如果方法运行时，已经处在一个事务中，进行事务的合并 |
| NOT\_SUPPORTED | 声明方法不需要事务。如果方法没有关联到一个事务，容器不会为它开启事务。如果方法在一个事务中被调用，该事务会被挂起，在方法调用结束后，原先的事务便会恢复执行 |
| REQUIRES\_NEW | 属性表明不管是否存在事务，业务方法总会为自己发起一个新的事务。如果方法已经运行在一个事务中，则原有事务会被挂起，新的事务会被创建，直到方法执行结束，新事务才算结束，原先的事务才会恢复执行 |
| **MANDATORY** | 该属性指定业务方法只能在一个已经存在的事务中执行，业务方法不能发起自己的事务。如果业务方法在没有事务的环境下调用，容器就会抛出例外。 |
| **SUPPORTS** | 这一事务属性表明，如果业务方法在某个事务范围内被调用，则方法成为该事务的一部分。如果业务方法在事务范围外被调用，则方法在没有事务的环境下执行 |
| **Never** | 指定业务方法绝对不能在事务范围内执行。如果业务方法在某个事务中执行，容器会抛出例外，只有业务方法没有关联到任何事务，才能正常执行 |
| **NESTED** | 如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务, 则按REQUIRED属性执行.它使用了一个单独的事务， 这个事务拥有多个可以回滚的保存点。内部事务的回滚不会对外部事务造成影响。它只对DataSourceTransactionManager事务管理器起效 |

事务的隔离级别：用来限定事务内外的哪些改变是可见的，哪些是不可见的。低级别的隔离级一般支持更高的并发处理，并拥有更低的系统开销。

|  |  |
| --- | --- |
| 隔离级别 | 含义 |
| DEFAULT | 使用后端数据库默认的隔离级别(spring中的的选择项) |
| READ\_UNCOMMITED | 允许你读取还未提交的改变了的数据。可能导致脏、幻、不可重复读 |
| READ\_COMMITTED | 允许在并发事务已经提交后读取。可防止脏读，但幻读和 不可重复读仍可发生 |
| REPEATABLE\_READ | 对相同字段的多次读取是一致的，除非数据被事务本身改变。可防止脏、不可重复读，但幻读仍可能发生。 |
| SERIALIZABLE | 完全服从ACID的隔离级别，确保不发生脏、幻、不可重复读。这在所有的隔离级别中是最慢的，它是典型的通过完全锁定在事务中涉及的数据表来完成的。 |

什么是脏数据，脏读，不可重复读，幻觉读？

1）脏读: 指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

2）不可重复读: 指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

3）幻觉读: 指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

mysql默认的事务处理级别是'REPEATABLE-READ',也就是可重复读

查看当前会话隔离级别

select @@tx\_isolation;

查看系统当前隔离级别

select @@global.tx\_isolation;

# 三 Spring与Mybatis整合

将SqlSessionFactory的创建交给spring完成

## 1导入jar文件

导入spring相关的jar；

导入myabatis相关的jar

导入mybatis-spring-XXXX.jar

如果使用连接池，导入连接池的jar文件，如druid

使用mysql数据库，导入mysql的驱动jar文件

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

<!-- 事务会自动引入 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.1.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjrt</artifactId>

<version>1.8.2</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.2</version>

</dependency>

<dependency>

<groupId>aopalliance</groupId>

<artifactId>aopalliance</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.5</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.18</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.39</version>

</dependency>

</dependencies>

## 2 配置

<?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"*

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

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*

*http://www.springframework.org/schema/tx*

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

<!-- 扫描相关注解 -->

<context:component-scan base-package=*"com.qianfeng.sm"*></context:component-scan>

<!-- 1 druid数据源 -->

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*

init-method=*"init"* destroy-method=*"close"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/j1806"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*"root"* />

<!-- 配置初始化大小、最小、最大 -->

<property name=*"initialSize"* value=*"1"* />

<property name=*"minIdle"* value=*"1"* />

<property name=*"maxActive"* value=*"10"* />

<!-- 配置获取连接等待超时的时间 -->

<property name=*"maxWait"* value=*"10000"* />

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name=*"timeBetweenEvictionRunsMillis"* value=*"60000"* />

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name=*"minEvictableIdleTimeMillis"* value=*"300000"* />

<property name=*"testWhileIdle"* value=*"true"* />

<!-- 这里建议配置为TRUE，防止取到的连接不可用 -->

<property name=*"testOnBorrow"* value=*"true"* />

<property name=*"testOnReturn"* value=*"false"* />

</bean>

<!-- 2 创建Mybatis的工厂对象 -->

<bean id=*"sqlSessionFactory"*

class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!--设置数据库连接池 -->

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

<!-- 加载mybatis主配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis2.xml"*/>

<!-- 加载映射文件 -->

<property name=*"mapperLocations"* value=*"classpath:com/qianfeng/sm/\*.xml"*/>

</bean>

<!--3设置Mybatis的映射接口 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*/>

<!-- 设置映射文件所在包 -->

<property name=*"basePackage"* value=*"com.qianfeng.sm"*></property>

</bean>

<!-- 4配置事务管理类 -->

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

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

</bean>

<!-- 5配置事务的特性 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"txManage"*>

<tx:attributes>

<!-- 针对使用事务的add开头的方法

read-only 是否只读，true 是，false 可读可写

如果有插入等操作，设为为true，运行程序会报异常-->

<tx:method name=*"add\*"* read-only=*"false"* propagation=*"REQUIRED"*/>

<tx:method name=*"delete\*"* read-only=*"false"* propagation=*"REQUIRED"*/>

<tx:method name=*"update\*"* read-only=*"false"* propagation=*"REQUIRED"*/>

<tx:method name=*"find\*"* read-only=*"true"* propagation=*"REQUIRED"*/>

<!-- 针对其余的方法 -->

<tx:method name=*"\*"* propagation=*"NOT\_SUPPORTED"*/>

</tx:attributes>

</tx:advice>

<!-- 6 AOP配置 -->

<aop:config>

<!-- 切入点 -->

<aop:pointcut expression=*"execution(\* com.qianfeng.sm.UserService.\*(..))"* id=*"pc"*/>

<!-- 通知 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pc"*/>

</aop:config>

</beans>