spring的@Transactional注解详细用法

标签：

概述

事务管理对于企业应用来说是至关重要的，即使出现异常情况，它也可以保证数据的一致性。

Spring Framew ork对事务管理提供了一致的抽象，其特点如下：

为不同的事务API提供一致的编程模型，比如JTA(Jav a Transaction API), JDBC, Hibernate, JPA(Jav a Persistence API和JDO(Jav a Data Objects)

支持声明式事务管理，特别是基于注解的声明式事务管理，简单易用

提供比其他事务API如JTA更简单的编程式事务管理API

与spring数据访问抽象的完美集成

事务管理方式

spring支持编程式事务管理和声明式事务管理两种方式。

编程式事务管理使用TransactionTemplate或者直接使用底层的PlatformTransactionManager。对于编程式事务管理，spring推荐使用TransactionTemplate。

声明式事务管理建立在AOP之上的。其本质是对方法前后进行拦截，然后在目标方法开始之前创建或者加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。声明式事务最大的优点就是不需要通过编程的方式管理事务，这样就不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明(或通过基于@Transactional注解的方式)，便可以将事务规则应用到业务逻辑中。

显然声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式。声明式事务管理使业务代码不受污染，一个普通的POJO对象，只要加上注解就可以获得完全的事务支持。和编程式事务相比，声明式事务唯一不足地方是，后者的最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。但是即便有这样的需求，也存在很多变通的方法，比如，可以将需要进行事务管理的代码块独立为方法等等。

声明式事务管理也有两种常用的方式，一种是基于tx和aop名字空间的xml配置文件，另一种就是基于@Transactional注解。显然基于注解的方式更简单易用，更清爽。

自动提交(AutoCom m it)与连接关闭时的是否自动提交

自动提交

默认情况下，数据库处于自动提交模式。每一条语句处于一个单独的事务中，在这条语句执行完毕时，如果执行成功则隐式的提交事务，如果
执行失败则隐式的回滚事务。

对于正常的事务管理，是一组相关的操作处于一个事务之中，因此必须关闭数据库的自动提交模式。不过，这个我们不用担心，spring会将底层连接的自动提交特性设置为false。

org/springframew ork/jdbc/datasource/DataSourceTransactionManager.java

```
1 // sw itch to manual commit if necessary. this is very expensive in some jdbc drivers,
```

```
2 // so w e don‘t w ant to do it unnecessarily (for example if w e‘ve explicitly
3 // configured the connection pool to set it already).
4 if (con.getautocommit()) {
5     txobject.setmustrestoreautocommit(true);
6     if (logger.isdebugenabled()) {
7         logger.debug("sw itching jdbc connection [" + con + "] to manual commit");
8     }
9     con.setautocommit(false);
10 }
```

有些数据连接池提供了关闭事务自动提交的设置，最好在设置连接池时就将其关闭。但C3P0没有提供这一特性，只能依靠spring来设置。

因为JDBC规范规定，当连接对象建立时应该处于自动提交模式，这是跨DBMS的缺省值，如果需要,必须显式的关闭自动提交。C3P0遵守这一规范，让客户代码来显式的设置需要的提交模式。

连接关闭时的是否自动提交

当一个连接关闭时，如果有未提交的事务应该如何处理？JDBC规范没有提及，C3P0默认的策略是回滚任何未提交的事务。这是一个正确的策略，但JDBC驱动提供商之间对此问题并没有达成一致。

C3P0的autoCommitOnClose属性默认是false,没有十分必要不要动它。或者可以显式的设置此属性为false，这样会更明确。

基于注解的声明式事务管理配置

spring-servlet.xml

```
1 <!-- transaction support-->
2 <!-- PlatformTransactionMnager -->
3 <bean id="txManager" class="org.springframew ork.jdbc.datasource.DataSourceTransactionManager">
4     <property name="dataSource" ref="dataSource" />
5 </bean>
6 <!-- enable transaction annotation support -->
7 <tx:annotation-driven transaction-manager="txManager" />
```

还要在spring-servlet.xml中添加tx名字空间

```
1 ...
2     xmlns:tx="http://w w w .springframew ork.org/schema/tx"
3     xmlns:aop="http://w w w .springframew ork.org/schema/aop"
4     xsi:schemaLocation="
5     ...
6
7 http://w w w .springframew ork.org/schema/tx
8
9
```

```
10 http://w w w .springframew ork.org/schema/tx/spring-tx.xsd
11
12     ...
```

MyBatis自动参与到spring事务管理中，无需额外配置，只要org.mybatis.spring.SqlSessionFactoryBean引用的数据源与DataSourceTransactionManager引用的数据源一致即可，否则事务管理会不起作用。

另外需要下载依赖包aopalliance.jar放置到WEB-INF/lib目录下。否则spring初始化时会报异常

java.lang.NoClassDefFoundError: org/aopalliance/intercept/MethodInterceptor

spring事务特性

spring所有的事务管理策略类都继承自org.springframew ork.transaction.PlatformTransactionManager接口

```
1 public interface PlatformTransactionManager {
2
3   TransactionStatus getTransaction(TransactionDefinition definition)
4     throw s TransactionException;
5
6   void commit(TransactionStatus status) throw s TransactionException;
7
8   void rollback(TransactionStatus status) throw s TransactionException;
9 }
```

其中TransactionDefinition接口定义以下特性：

事务隔离级别

隔离级别是指若干个并发的事务之间的隔离程度。TransactionDefinition 接口中定义了五个表示隔离级别的常量：

TransactionDef inition.ISOLATION_DEFAULT：这是默认值，表示使用底层数据库的默认隔离级别。对大部分数据库而言，通常这值就是Tran sactionDef inition.ISOLATION_READ_COMMITTED。

TransactionDef inition.ISOLATION_READ_UNCOMMITTED：该隔离级别表示一个事务可以读取另一个事务修改但还没有提交的数据。该级别不能防止脏读，不可重复读和幻读，因此很少使用该隔离级别。比如PostgreSQL实际上并没有此级别。

TransactionDef inition.ISOLATION_READ_COMMITTED：该隔离级别表示一个事务只能读取另一个事务已经提交的数据。该级别可以防止脏读，这也是大多数情况下的推荐值。

TransactionDef inition.ISOLATION_REPEATABLE_READ：该隔离级别表示一个事务在整个过程中可以多次重复执行某个查询，并且每次返回的记录都相同。该级别可以防止脏读和不可重复读。

TransactionDef inition.ISOLATION_SERIALIZABLE：所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

事务传播行为

所谓事务的传播行为是指，如果在开始当前事务之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务性方法的执行行为。在TransactionDefinition定义中包括了如下几个表示传播行为的常量：

TransactionDef inition.PROPAGATION_REQUIRED：如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。这是默

认值。

TransactionDefinition.PROPAGATION_REQUIRES_NEW：创建一个新的事务，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION_SUPPORTS：如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

TransactionDefinition.PROPAGATION_NOT_SUPPORTED：以非事务方式运行，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION_NEVER：以非事务方式运行，如果当前存在事务，则抛出异常。

TransactionDefinition.PROPAGATION_MANDATORY：如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。

TransactionDefinition.PROPAGATION_NESTED：如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于TransactionDefinition.PROPAGATION_REQUIRED。

事务超时

所谓事务超时，就是指一个事务所允许执行的最长时间，如果超过该时间限制但事务还没有完成，则自动回滚事务。在 TransactionDefinition 中以 int 的值来表示超时时间，其单位是秒。

默认设置为底层事务系统的超时值，如果底层数据库事务系统没有设置超时值，那么就是none，没有超时限制。

事务只读属性

只读事务用于客户代码只读但不修改数据的情形，只读事务用于特定情景下的优化，比如使用Hibernate的时候。
默认为读写事务。

spring事务回滚规则

指示spring事务管理器回滚一个事务的推荐方法是在当前事务的上下文内抛出异常。spring事务管理器会捕捉任何未处理的异常，然后依据规则决定是否回滚抛出异常的事务。

默认配置下，spring只有在抛出的异常为运行时unchecked异常时才回滚该事务，也就是抛出的异常为RuntimeException的子类(Errors也会导致事务回滚)，而抛出checked异常则不会导致事务回滚。
可以明确的配置在抛出那些异常时回滚事务，包括checked异常。也可以明确定义那些异常抛出时不回滚事务。

还可以编程性的通过setRollbackOnly()方法来指示一个事务必须回滚，在调用完setRollbackOnly()后你所能执行的唯一操作就是回滚。

@Transactional注解

@Transactional属性

属性	类型	描述
value	String	可选的限定描述符，指定使用的事务管理器
propagation	enum: Propagation	可选的事务传播行为设置
isolation	enum: Isolation	可选的事务隔离级别设置
readOnly	boolean	读写或只读事务，默认读写
timeout	int (in seconds granularity)	事务超时时间设置
rollbackFor	Class对象数组，必须继承自Throwable	导致事务回滚的异常类数组
rollbackForClassName	类名数组，必须继承自Throwable	导致事务回滚的异常类名字数组

属性	类型	描述
noRollbackFor	Class对象数组，必须继承自Throwable	不会导致事务回滚的异常类数组
noRollbackForClassName	类名数组，必须继承自Throwable	不会导致事务回滚的异常类名字数组

用法

@Transactional 可以作用于接口、接口方法、类以及类方法上。当作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。

虽然 @Transactional 注解可以作用于接口、接口方法、类以及类方法上，但是 Spring 建议不要在接口或者接口方法上使用该注解，因为这只有在使用基于接口的代理时它才会生效。另外， @Transactional 注解应该只被应用到 public 方法上，这是由 Spring AOP 的本质决定的。如果你在 protected、private 或者默认可见性的方法上使用 @Transactional 注解，这将被忽略，也不会抛出任何异常。

默认情况下，只有来自外部的方法调用才会被AOP代理捕获，也就是，类内部方法调用本类内部的其他方法并不会引起事务行为，即使被调用方法使用@Transactional注解进行修饰。

```
1 @Transactional(readOnly = true)
2 public class DefaultFooService implements FooService {
3
4   public Foo getFoo(String fooName) {
5     // do something
6   }
7
8   // these settings have precedence for this method
9   //方法上注解属性会覆盖类注解上的相同属性
10   @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
11   public void updateFoo(Foo foo) {
12     // do something
13   }
14 }
```

标签：