# [Spring 使用注解方式进行事务管理](http://www.cnblogs.com/younggun/p/3193800.html)

**使用步骤：**

**步骤一、在spring配置文件中引入<tx:>命名空间**  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 **xmlns:tx="http://www.springframework.org/schema/tx"**  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-2.0.xsd **http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">  
  
步骤二、具有@Transactional 注解的bean自动配置为声明式事务支持**

[复制代码](javascript:void(0);)

<!-- 事务管理器配置, Hibernate单数据源事务 -->

<bean id="defaultTransactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 使用annotation定义事务 -->

<tx:annotation-driven transaction-manager="defaultTransactionManager" proxy-target-class="true" />

[复制代码](javascript:void(0);)

**步骤三、在接口或类的声明处 ,写一个@Transactional.**  
要是只在接口上写, 接口的实现类就会继承下来、接口的实现类的具体方法,可以覆盖类声明处的设置  
@Transactional   //类级的注解、适用于类中所有的public的方法

**事务的传播行为和隔离级别**

大家在使用spring的注解式事务管理时，对事务的传播行为和隔离级别可能有点不知所措，下边就详细的介绍下以备方便查阅。

**事物注解方式: @Transactional**

当标于类前时, 标示类中所有方法都进行事物处理 , 例子:

@Transactional

public class TestServiceBean implements TestService {}

**当类中某些方法不需要事物时:**

[复制代码](javascript:void(0);)

@Transactional

public class TestServiceBean implements TestService {

private TestDao dao;

public void setDao(TestDao dao) {

this.dao = dao;

}

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

public List<Object> getAll() {

return null;

}

}

[复制代码](javascript:void(0);)

**事物传播行为介绍:**   
@Transactional(propagation=Propagation.REQUIRED)   
如果有事务, 那么加入事务, 没有的话新建一个(默认情况下)  
@Transactional(propagation=Propagation.NOT\_SUPPORTED)   
容器不为这个方法开启事务  
@Transactional(propagation=Propagation.REQUIRES\_NEW)   
不管是否存在事务,都创建一个新的事务,原来的挂起,新的执行完毕,继续执行老的事务  
@Transactional(propagation=Propagation.MANDATORY)   
必须在一个已有的事务中执行,否则抛出异常  
@Transactional(propagation=Propagation.NEVER)   
必须在一个没有的事务中执行,否则抛出异常(与Propagation.MANDATORY相反)  
@Transactional(propagation=Propagation.SUPPORTS)   
如果其他bean调用这个方法,在其他bean中声明事务,那就用事务.如果其他bean没有声明事务,那就不用事务.

**事物超时设置:**@Transactional(timeout=30) //默认是30秒

**事务隔离级别:**@Transactional(isolation = Isolation.READ\_UNCOMMITTED)  
读取未提交数据(会出现脏读, 不可重复读) 基本不使用  
@Transactional(isolation = Isolation.READ\_COMMITTED)  
读取已提交数据(会出现不可重复读和幻读)  
@Transactional(isolation = Isolation.REPEATABLE\_READ)  
可重复读(会出现幻读)  
@Transactional(isolation = Isolation.SERIALIZABLE)  
串行化

MYSQL: 默认为REPEATABLE\_READ级别  
SQLSERVER: 默认为READ\_COMMITTED

**脏读** : 一个事务读取到另一事务未提交的更新数据  
**不可重复读** : 在同一事务中, 多次读取同一数据返回的结果有所不同, 换句话说,   
后续读取可以读到另一事务已提交的更新数据. 相反, "可重复读"在同一事务中多次  
读取数据时, 能够保证所读数据一样, 也就是后续读取不能读到另一事务已提交的更新数据  
**幻读** : 一个事务读到另一个事务已提交的insert数据

**@Transactional注解中常用参数说明**

|  |  |
| --- | --- |
| 参 数 名 称 | 功 能 描 述 |
| readOnly | 该属性用于设置当前事务是否为只读事务，设置为true表示只读，false则表示可读写，默认值为false。例如：@Transactional(readOnly=true) |
| rollbackFor | 该属性用于设置需要进行回滚的异常类数组，当方法中抛出指定异常数组中的异常时，则进行事务回滚。例如：  指定单一异常类：@Transactional(rollbackFor=RuntimeException.class)  指定多个异常类：@Transactional(rollbackFor={RuntimeException.class, Exception.class}) |

 续表）

|  |  |
| --- | --- |
| 参 数 名 称 | 功 能 描 述 |
| rollbackForClassName | 该属性用于设置需要进行回滚的异常类名称数组，当方法中抛出指定异常名称数组中的异常时，则进行事务回滚。例如：  指定单一异常类名称：@Transactional(rollbackForClassName="RuntimeException")  指定多个异常类名称：@Transactional(rollbackForClassName={"RuntimeException","Exception"}) |
| noRollbackFor | 该属性用于设置不需要进行回滚的异常类数组，当方法中抛出指定异常数组中的异常时，不进行事务回滚。例如：  指定单一异常类：@Transactional(noRollbackFor=RuntimeException.class)  指定多个异常类：@Transactional(noRollbackFor={RuntimeException.class, Exception.class}) |
| noRollbackForClassName | 该属性用于设置不需要进行回滚的异常类名称数组，当方法中抛出指定异常名称数组中的异常时，不进行事务回滚。例如：  指定单一异常类名称：@Transactional(noRollbackForClassName="RuntimeException")  指定多个异常类名称：  @Transactional(noRollbackForClassName={"RuntimeException","Exception"}) |
| propagation | 该属性用于设置事务的传播行为，具体取值可参考表6-7。  例如：@Transactional(propagation=Propagation.NOT\_SUPPORTED,readOnly=true) |
| isolation | 该属性用于设置底层数据库的事务隔离级别，事务隔离级别用于处理多事务并发的情况，通常使用数据库的默认隔离级别即可，基本不需要进行设置 |
| timeout | 该属性用于设置事务的超时秒数，默认值为-1表示永不超时 |

**注意的几点:**  
1 @Transactional 只能被应用到public方法上, 对于其它非public的方法,如果标记了@Transactional也不会报错,但方法没有事务功能.  
  
2用 spring 事务管理器,由spring来负责数据库的打开,提交,回滚.默认遇到运行期例外(throw new RuntimeException("注释");)会回滚，即遇到不受检查（unchecked）的例外时回滚；而遇到需要捕获的例外(throw new Exception("注释");)不会回滚,即遇到受检查的例外（就是非运行时抛出的异常，编译器会检查到的异常叫受检查例外或说受检查异常）时，需我们指定方式来让事务回滚 要想所有异常都回滚,要加上 @Transactional( rollbackFor={Exception.class,其它异常}) .如果让unchecked例外不回滚： @Transactional(notRollbackFor=RunTimeException.class)  
如下:  
@Transactional(rollbackFor=Exception.class) //指定回滚,遇到异常Exception时回滚  
public void methodName() {  
throw new Exception("注释");  
  
}  
@Transactional(noRollbackFor=Exception.class)//指定不回滚,遇到运行期例外(throw new RuntimeException("注释");)会回滚  
public ItimDaoImpl getItemDaoImpl() {  
throw new RuntimeException("注释");  
}

3、@Transactional 注解应该只被应用到 public 可见度的方法上。 如果你在 protected、private 或者 package-visible 的方法上使用 @Transactional 注解，它也不会报错， 但是这个被注解的方法将不会展示已配置的事务设置。

4、@Transactional 注解可以被应用于接口定义和接口方法、类定义和类的 public 方法上。然而，请注意仅仅 @Transactional 注解的出现不足于开启事务行为，它仅仅 是一种元数据，能够被可以识别 @Transactional 注解和上述的配置适当的具有事务行为的beans所使用。上面的例子中，其实正是 <tx:annotation-driven/>元素的出现 开启 了事务行为。

5、Spring团队的建议是你在具体的类（或类的方法）上使用 @Transactional 注解，而不要使用在类所要实现的任何接口上。你当然可以在接口上使用 @Transactional 注解，但是这将只能当你设置了基于接口的代理时它才生效。因为注解是 不能继承 的，这就意味着如果你正在使用基于类的代理时，那么事务的设置将不能被基于类的代理所识别，而且对象也将不会被事务代理所包装（将被确认为严重的）。因 此，请接受Spring团队的建议并且在具体的类上使用 @Transactional 注解。