# Spring事务管理

## Spring事务管理的主要接口关联关系

根据事务的定义信息，调用具体的事务管理器来真正执行相应的事务(提交/回滚/开启新事务…)，执行事务时，产生的事务状态 将保存在事务运行状态中

## PlatformTransactionManage 事务管理器

平台事务管理器 接口 根据不同的持久层框架来选择不同的的具体实现类，主要有以下几个实现类

### DataSourceTransactionManager JDBC/MyBatis

org.springframework.jdbc.datasource.DataSourceTransactionManager

### HibernateTransactionManager Hibernate

org.springframework.orm.hibernate4.HibernateTransactionManager

其他的事务管理器有

JdoTransactionManager 持久层使用Jdo

JpaTransactionManager 持久层使用JPA

JtaTransactionManager 持久层使用JTA

## TransactionDefinition 事务的定义信息

隔离 传播 超时 只读

### (隔离级别)事务的隔离性

常量TransactionDefinition.ISOLATION\_\*\*\*\* 定义事务的隔离级别

TransactionDefinition.ISOLATION\_DEFAULT使用底层数据默认的隔离级别

### 事务的传播行为

发生的情况，在定义事务级别的方法中，又调用其他的方法，第一个方法的事务传递至第二个方法，如Service中的 add1()方法后又执行add2()方法 就需要控制传播行为

常量TransactionDefinition.PROPAGATION\_\*\*\*\*定义事务的传播行为

主要的传播行为

1. TransactionDefinition.PROPAGATION\_REQUIRED 支持当前事务，如果不存在就新建一个事务
2. PROPAGATION\_REQUIRES\_NEW 如果有事务存在，挂起当前事务，创建一个新的事务，2个方法不再一个事务中
3. PROPAGATION\_NESTED 如果有事务存在，使用嵌套事务

### 事务的超时

### 事务的只读

## TransactionStatus 事务的状态

此接口来 获得/判断 事务的 执行状态

**boolean** isNewTransaction();

**boolean** hasSavepoint();

**void** setRollbackOnly();

**boolean** isRollbackOnly();

**void** flush();

**boolean** isCompleted();

## TransactionTemplate事务管理模板主要用于编程式事务

为了简化事务管理器的配置，可以用事务管理模板来进行事务管理

需要注入事务管理器

## Spring的声明式事务管理

### TransactionProxyFactoryBean基于代理对象的事务管理

在使用对象时需要使用代理对象,才能有事务，需要为每一个需要开启事务的对象配置代理

<!-- 配置代理对象 -->

<bean id=*"serviceProxy"*

class=*"org.springframework.transaction.interceptor.TransactionProxyFactoryBean"*>

<!-- 注入要代理的对象 -->

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

<!-- 注入事务管理器 -->

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

<!-- 注入事务的属性 -->

<property name=*"transactionAttributes"*>

<props>

<!-- key表示方法 prop值的格式:

PROPAGATION :事务的传播行为

ISOLATION :事务的隔离级别

readOnly :设置为只读,当 update/delete将抛出异常

-Exception :发生那些异常回滚事务

+Exception :发生哪些异常不回滚事务

-->

<prop key=*"insert\*"*>PROPAGATION\_REQUIRED,</prop>

</props>

</property>

</bean>

### 基于AspectJ的XML配置的事务管理

<!-- 事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

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

</bean>

<!-- 配置切面 -->

<aop:config>

<!-- 配置切入点 -->

<aop:pointcut expression=*"execution(\*)"* id=*"txPointCut"* />

<!-- 配置切面 -->

<aop:advisor advice-ref=*"Mytx"* pointcut-ref=*"txPointCut"* />

</aop:config>

<!-- 配置事务的通知(事务增强) -->

<tx:advice id=*"Mytx"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- propagation:事务的传播,isolation:事务的隔离级别,read-only:只读,rollback-for:发生哪些异常回滚,no-rollback-for:哪些异常不回滚,timeout:超时-->

<tx:method name=*"\*"* propagation=*"REQUIRED"* isolation=*"DEFAULT"* read-only=*"false"* rollback-for=*""* no-rollback-for=*""* timeout=*"-1"*/>

</tx:attributes>

</tx:advice>

### 基于注解的事务管理

<!-- 开启注解事务 ,配置事务管理器-->

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

<!-- 事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*><property name=*"dataSource"* ref=*"dataSource"* />

</bean>

在类或方法注解 @Transactional即可使用事务

注解可以设置属性值

(isolation=Isolation.*DEFAULT*,propagation=Propagation.*REQUIRED*,readOnly=**false**,rollbackForClassName={“”,””}) 等