

## 事务有四个特性:

ACID

## 事务有几个核心接口:

public interface TransactionDefinition {

int getPropagationBehavior(); // 返回事务的传播行为

int getIsolationLevel(); // 返回事务的隔离级别，事务管理器根据它来控制另外一个事务可以看到本事务内的哪些数据

int getTimeout(); // 返回事务必须在多少秒内完成

boolean isReadOnly(); // 事务是否只读，事务管理器能够根据这个返回值进行优化，确保事务是只读的

}

Public interface PlatformTransactionManager()...{

TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionException; // 由TransactionDefinition得到TransactionStatus对象

Void commit(TransactionStatus status) throws TransactionException; // 提交

Void rollback(TransactionStatus status) throws TransactionException; // 回滚

}

public interface TransactionStatus{

boolean isNewTransaction(); // 是否是新的事物

boolean hasSavepoint(); // 是否有恢复点

void setRollbackOnly(); // 设置为只回滚

boolean isRollbackOnly(); // 是否为只回滚

boolean isCompleted; // 是否已完成

}

## 事务属性包含5个方面:

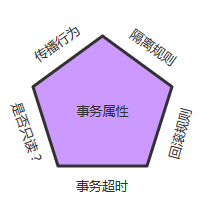
传播行为

隔离规则

回滚规则

事务超时

是否只读



public interface TransactionDefinition {

int getPropagationBehavior(); // 返回事务的传播行为

int getIsolationLevel(); // 返回事务的隔离级别，事务管理器根据它来控制另外一个事务可以看到本事务内的哪些数据

int getTimeout(); // 返回事务必须在多少秒内完成

boolean isReadOnly(); // 事务是否只读，事务管理器能够根据这个返回值进行优化，确保事务是只读的

}

我们可以发现TransactionDefinition正好用来定义事务属性，下面详细介绍一下各个事务属性。

传播行为:

| 传播行为 | 含义 |
| --- | --- |
| PROPAGATION\_REQUIRED | 表示当前方法必须运行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务 |
| PROPAGATION\_SUPPORTS | 表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行 |
| PROPAGATION\_MANDATORY | 表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常 |
| PROPAGATION\_REQUIRED\_NEW | 表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NOT\_SUPPORTED | 表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager |
| PROPAGATION\_NEVER | 表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常 |
| PROPAGATION\_NESTED | 表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION\_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务 |

隔离级别:

事务的第二个维度就是隔离级别（isolation level）。隔离级别定义了一个事务可能受其他并发事务影响的程度。

（1）并发事务引起的问题

* 脏读（Dirty reads）——脏读发生在一个事务读取了另一个事务改写但尚未提交的数据时。如果改写在稍后被回滚了，那么第一个事务获取的数据就是无效的。
* 不可重复读（Nonrepeatable read）——不可重复读发生在一个事务执行相同的查询两次或两次以上，但是每次都得到不同的数据时。这通常是因为另一个并发事务在两次查询期间进行了更新。
* 幻读（Phantom read）——幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录。

（2）隔离级别

| 隔离级别 | 含义 |
| --- | --- |
| ISOLATION\_DEFAULT | 使用后端数据库默认的隔离级别 |
| ISOLATION\_READ\_UNCOMMITTED | 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读 |
| ISOLATION\_READ\_COMMITTED | 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生 |
| ISOLATION\_REPEATABLE\_READ | 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生 |
| ISOLATION\_SERIALIZABLE | 最高的隔离级别，完全服从ACID的隔离级别，确保阻止脏读、不可重复读以及幻读，也是最慢的事务隔离级别，因为它通常是通过完全锁定事务相关的数据库表来实现的 |

## 3.1 编程式和声明式事务的区别

Spring提供了对编程式事务和声明式事务的支持，编程式事务允许用户在代码中精确定义事务的边界，而声明式事务（基于AOP）有助于用户将操作与事务规则进行解耦。   
简单地说，编程式事务侵入到了业务代码里面，但是提供了更加详细的事务管理；而声明式事务由于基于AOP，所以既能起到事务管理的作用，又可以不影响业务代码的具体实现。

## 3.2 如何实现编程式事务？

Spring提供两种方式的编程式事务管理，分别是：

使用TransactionTemplate

直接使用PlatformTransactionManager

TransactionTemplate tt = new TransactionTemplate(); // 新建一个TransactionTemplate

Object result = tt.execute(

new TransactionCallback(){

public Object doTransaction(TransactionStatus status){

updateOperation();

return resultOfUpdateOperation();

}

}); // 执行execute方法进行事务管理

---------------------------------------------------------------------------------------------------------------------------------------

DataSourceTransactionManager dataSourceTransactionManager = new DataSourceTransactionManager(); //定义一个某个框架平台的TransactionManager，如JDBC、Hibernate

dataSourceTransactionManager.setDataSource(this.getJdbcTemplate().getDataSource()); // 设置数据源

DefaultTransactionDefinition transDef = new DefaultTransactionDefinition(); // 定义事务属性

transDef.setPropagationBehavior(DefaultTransactionDefinition.PROPAGATION\_REQUIRED); // 设置传播行为属性

TransactionStatus status = dataSourceTransactionManager.getTransaction(transDef); // 获得事务状态

try {

// 数据库操作

dataSourceTransactionManager.commit(status);// 提交

} catch (Exception e) {

dataSourceTransactionManager.rollback(status);// 回滚

}

# 4 声明式事务

## 4.1 配置方式

注：以下配置代码参考自[*Spring事务配置的五种方式*](http://www.blogjava.net/robbie/archive/2009/04/05/264003.html)

根据代理机制的不同，总结了五种Spring事务的配置方式，配置文件如下：

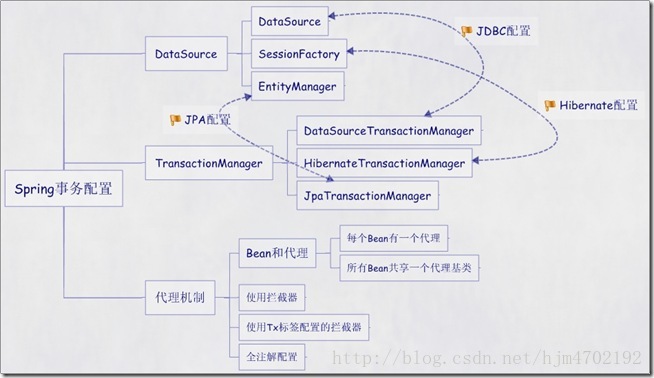
# Spring事务配置的五种方法

前段时间对spring的事务配置做了比较深入的研究，在此之间对Spring的事务配置虽说也配置过，但是一直没有一个清楚的认识。通过这次的学习发觉Spring的事务配置只要把思路理清，还是比较好掌握的。

总结如下：   
**Spring配置文件中关于事务配置总是由三个组成部分，分别是DataSource、TransactionManager和代理机制这三部分，无论哪种配置方式，一般变化的只是代理机制这部分。**

DataSource、TransactionManager这两部分只是会根据数据访问方式有所变化，比如使用hibernate进行数据访问时，DataSource实际为SessionFactory，TransactionManager的实现为HibernateTransactionManager。

具体如下图：



根据代理机制的不同，总结了五种Spring事务的配置方式，配置文件如下：

### （1）每个Bean都有一个代理

<?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"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

<property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />

</bean>

<!-- 定义事务管理器（声明式的事务） -->

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

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

</bean>

<!-- 配置DAO -->

<bean id="userDaoTarget" class="com.bluesky.spring.dao.UserDaoImpl">

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

</bean>

<bean id="userDao" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<!-- 配置事务管理器 -->

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

<property name="target" ref="userDaoTarget" />

<property name="proxyInterfaces" value="com.bluesky.spring.dao.GeneratorDao" />

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

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

</props>

</property>

</bean>

</beans>

### （2）所有Bean共享一个代理基类

<?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"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

<property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />

</bean>

<!-- 定义事务管理器（声明式的事务） -->

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

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

</bean>

<bean id="transactionBase" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean" lazy-init="true" abstract="true">

<!-- 配置事务管理器 -->

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

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

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

</props>

</property>

</bean>

<!-- 配置DAO -->

<bean id="userDaoTarget" class="com.bluesky.spring.dao.UserDaoImpl">

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

</bean>

<bean id="userDao" parent="transactionBase" >

<property name="target" ref="userDaoTarget" />

</bean>

</beans>

### （3）使用拦截器

<?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"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

<property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />

</bean>

<!-- 定义事务管理器（声明式的事务） -->

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

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

</bean>

<bean id="transactionInterceptor" class="org.springframework.transaction.interceptor.TransactionInterceptor">

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

<!-- 配置事务属性 -->

<property name="transactionAttributes">

<props>

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

</props>

</property>

</bean>

<bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">

<property name="beanNames">

<list>

<value>\*Dao</value>

</list>

</property>

<property name="interceptorNames">

<list>

<value>transactionInterceptor</value>

</list>

</property>

</bean>

<!-- 配置DAO -->

<bean id="userDao" class="com.bluesky.spring.dao.UserDaoImpl">

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

</bean>

</beans>

### （4）使用tx标签配置的拦截器

<?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-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:annotation-config />

<context:component-scan base-package="com.bluesky" />

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

<property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />

</bean>

<!-- 定义事务管理器（声明式的事务） -->

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

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

</bean>

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

<tx:attributes>

<tx:method name="\*" propagation="REQUIRED" />

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id="interceptorPointCuts" expression="execution(\* com.bluesky.spring.dao.\*.\*(..))" />

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

</aop:config>

</beans>

### （5）全注解

<?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-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:annotation-config />

<context:component-scan base-package="com.bluesky" />

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

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

<property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />

</bean>

<!-- 定义事务管理器（声明式的事务） -->

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

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

</bean>

</beans>

此时在DAO上需加上@Transactional注解，如下：

package com.bluesky.spring.dao;

import java.util.List;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import org.springframework.stereotype.Component;

import com.bluesky.spring.domain.User;

@Transactional

@Component("userDao")

public class UserDaoImpl extends HibernateDaoSupport implements UserDao {

public List<User> listUsers() {

return this.getSession().createQuery("from User").list();

}

}

配置方式

事务属性的配置方式通过以下关键字来指定：

| **关键字** | **含义** |
| --- | --- |
| isolation | 指定事务的隔离级别 |
| propagation | 定义事务的传播规则 |
| read-only | 指定事务为只读 |
| rollback-for | rollback-for指定事务对哪些检查型异常应当回滚而不提交 |
| no-rollback-for | no-rollback-for指定事务对哪些异常应当继续执行而不回滚 |
| timeout | 对于长时间运行的事务定义超时时间 |

下面是一个完整的常用的配置方便的事务配置例子

<!-- 配置dataSource -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<property name="driverClass" value="${driverClass}"></property>

<property name="jdbcUrl" value="${jdbcUrl}"></property>

<property name="user" value="${user}"></property>

<property name="password" value="${password}"></property>

</bean>

<!-- 配置transactionManager -->

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

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

</bean>

<!-- 定义事务处理规则 -->

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

<tx:attributes>

<!-- 事务传播属性、隔离级别、最优化为只读事务、事务超时 -->

<tx:method name="\*" propagation="REQUIRED" isolation="DEFAULT" read-only="true" timeout="-1"/>

</tx:attributes>

</tx:advice>

<aop:config>

<!-- 定义切面 -->

<aop:pointcut id="pointcutId" expression="execution(\* com.google.code..\*.\*Service.\*(..))"/>

<!-- 定义织入点规则 -->

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

</aop:config>

-------------------------------------------------------------------------------------------------------------------------------------------------------

# 使用spring的DataSourceTransactionManager接口，有什么好处

<!-- 定义事务管理器 -->  
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
 <property name="dataSource" ref="dataSource" />  
</bean>  
  
这是管理事务的。

你一个service可能写多条操作[sql语句](https://www.baidu.com/s?wd=sql%E8%AF%AD%E5%8F%A5&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。当第一条成功了。第二条报错了。如果你没有事务。那么第一条[sql语句](https://www.baidu.com/s?wd=sql%E8%AF%AD%E5%8F%A5&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)就会生效。有了事务就可以回滚。保证同时成功或失败。最典型的就是银行的取钱。