# [**摘要**]

本文介绍Spring声明式事务管理与配置，包括Spring声明式事务配置的五种方式、事务的传播属性（Propagation）、Spring事务的隔离级别（Isolation level）等内容。

摘自：http://java.9sssd.com/javafw/art/1215

# 、Spring声明式事务配置的五种方式

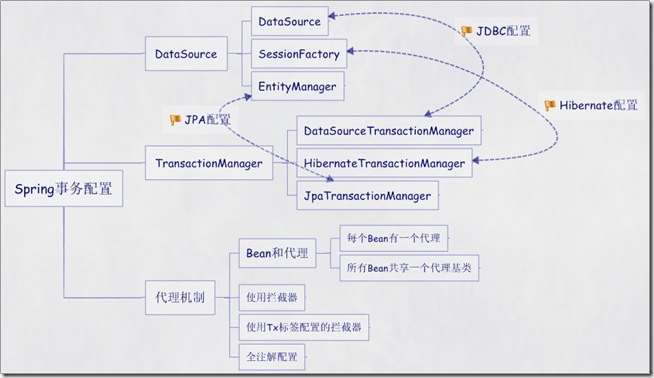
前段时间对Spring的事务配置做了比较深入的研究，在此之间对Spring的事务配置虽说也配置过，但是一直没有一个清楚的认识。通过这次的学习发觉Spring的事务配置只要把思路理清，还是比较好掌握的。

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：

## **第一种方式：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>

## **第二种方式：所有Bean共享一个代理基类**

|  |
| --- |
| <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>

## **第三种方式：使用拦截器**

|  |
| --- |
| ...... |
|  |

|  |
| --- |
| <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>

## **第四种方式：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> |
|  |

</bean>

## **第五种方式：全注解**

|  |
| --- |
| ...... |
|  |

|  |
| --- |
| <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" />

此时在DAO上需加上@Transactional注解，如下：

|  |
| --- |
| @Transactional |
|  |

|  |
| --- |
| @Component("userDao") |
|  |

|  |
| --- |
| public class UserDaoImpl extends HibernateDaoSupport implements UserDao { |
|  |

|  |
| --- |
| public List<User> listUsers() { |
|  |

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

|  |
| --- |
| } |
|  |

|  |
| --- |
| ...... |
|  |

}

# 、事务的传播属性（Propagation）

Propagation ：key属性确定代理应该给哪个方法增加事务行为。这样的属性最重要的部份是传播行为。有以下选项可供使用：PROPAGATION\_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

* PROPAGATION\_REQUIRED--加入当前正要执行的事务不在另外一个事务里，那么就起一个新的事务
* PROPAGATION\_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。
* PROPAGATION\_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。
* PROPAGATION\_REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起。
* PROPAGATION\_NOT\_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
* PROPAGATION\_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

**1： PROPAGATION\_REQUIRED**

加入当前正要执行的事务不在另外一个事务里，那么就起一个新的事务。

比 如说，ServiceB.methodB的事务级别定义为PROPAGATION\_REQUIRED, 那么由于执行ServiceA.methodA的时候，ServiceA.methodA已经起了事务，这时调用 ServiceB.methodB，ServiceB.methodB看到自己已经运行在ServiceA.methodA的事务内部，就不再起新的事 务。而假如ServiceA.methodA运行的时候发现自己没有在事务中，他就会为自己分配一个事务。这样，在ServiceA.methodA或者 在ServiceB.methodB内的任何地方出现异常，事务都会被回滚。即使ServiceB.methodB的事务已经被提交，但是 ServiceA.methodA在接下来fail要回滚，ServiceB.methodB也要回滚。

**2： PROPAGATION\_SUPPORTS**

如果当前在事务中，即以事务的形式运行，如果当前不再一个事务中，那么就以非事务的形式运行。

**3： PROPAGATION\_MANDATORY**

必须在一个事务中运行。也就是说，他只能被一个父事务调用。否则，他就要抛出异常。

**4： PROPAGATION\_REQUIRES\_NEW**

这 个就比较绕口了。 比如我们设计ServiceA.methodA的事务级别为PROPAGATION\_REQUIRED，ServiceB.methodB的事务级别为 PROPAGATION\_REQUIRES\_NEW，那么当执行到ServiceB.methodB的时候，ServiceA.methodA所在的事务 就会挂起，ServiceB.methodB会起一个新的事务，等待ServiceB.methodB的事务完成以后，他才继续执行。他与 PROPAGATION\_REQUIRED 的事务区别在于事务的回滚程度了。因为ServiceB.methodB是新起一个事务，那么就是存在两个不同的事务。如果 ServiceB.methodB已经提交，那么ServiceA.methodA失败回滚，ServiceB.methodB是不会回滚的。如果 ServiceB.methodB失败回滚，如果他抛出的异常被ServiceA.methodA捕获，ServiceA.methodA事务仍然可能提 交。

**5： PROPAGATION\_NOT\_SUPPORTED**

当前不支持事务。比如 ServiceA.methodA的事务级别是PROPAGATION\_REQUIRED ，而ServiceB.methodB的事务级别是PROPAGATION\_NOT\_SUPPORTED ，那么当执行到ServiceB.methodB时，ServiceA.methodA的事务挂起，而他以非事务的状态运行完，再继续 ServiceA.methodA的事务。

**6： PROPAGATION\_NEVER**

不能在事务中运行。假设ServiceA.methodA的事务级别是PROPAGATION\_REQUIRED， 而ServiceB.methodB的事务级别是PROPAGATION\_NEVER ，那么ServiceB.methodB就要抛出异常了。

**7： PROPAGATION\_NESTED**

理 解Nested的关键是savepoint。他与PROPAGATION\_REQUIRES\_NEW的区别 是，PROPAGATION\_REQUIRES\_NEW另起一个事务，将会与他的父事务相互独立，而Nested的事务和他的父事务是相依的，他的提交是 要等和他的父事务一块提交的。也就是说，如果父事务最后回滚，他也要回滚的。

而Nested事务的好处是他有一个savepoint。

[View Row Code](http://java.9sssd.com/javafw/art/1215)

|  |
| --- |
|  |
| ServiceA { | |
|  | |

|  |
| --- |
| //事务属性配置为 PROPAGATION\_REQUIRED |
|  |

|  |
| --- |
| void methodA() { |
|  |

|  |
| --- |
| try { |
|  |

|  |
| --- |
| //savepoint |
|  |

|  |
| --- |
| ServiceB.methodB(); //PROPAGATION\_NESTED 级别 |
|  |

|  |
| --- |
| } catch (SomeException) { |
|  |

|  |
| --- |
| // 执行其他业务, 如 ServiceC.methodC(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |

也就是说ServiceB.methodB失败回滚，那么ServiceA.methodA也会回滚到savepoint点上，ServiceA.methodA可以选择另外一个分支，比如：

ServiceC.methodC，继续执行，来尝试完成自己的事务。

但是这个事务并没有在EJB标准中定义。

# 、Spring事务的隔离级别（Isolation level）

1. ISOLATION\_DEFAULT： 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别。

另外四个与JDBC的隔离级别相对应。

2. ISOLATION\_READ\_UNCOMMITTED： 这是事务最低的隔离级别，它充许令外一个事务可以看到这个事务未提交的数据。

这种隔离级别会产生脏读，不可重复读和幻像读。

3. ISOLATION\_READ\_COMMITTED： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。

4. ISOLATION\_REPEATABLE\_READ： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。

它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

5. ISOLATION\_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。

除了防止脏读，不可重复读外，还避免了幻像读。

**什么是脏数据，脏读，不可重复读，幻觉读？**

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。

**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。

**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。