事务Transaction的基本概念

<https://www.cnblogs.com/mxmbk/p/5341258.html>

# 什么是事务？

**事务（Transaction）**是并发控制的单位，是用户定义的一个操作序列。**这些操作要么都做，要么都不做，是一个不可分割的工作单位**。

事务提交与事务回滚

数据库向用户提供保存当前程序状态的方法，叫**事务提交（commit）**；当事务执行过程中，使数据库忽略当前的状态并回到前面保存的状态的方法叫**事务回滚（rollback）**。

# 事务特性(ACID)

## Atomicity原子性

**原子性（Atomicity）**：将事务中所做的操作捆绑成一个原子单元，即对于事务所进行的数据修改等操作，**要么全部执行，要么全部不执行**。

## Consistency一致性

**一致性（Consistency）**：事务在完成时，必须使所有的数据都保持一致状态，而且在相关数据中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构都应该是正确的。

## Isolation隔离性

**隔离性（Isolation）**：由并发事务所做的修改必须与任何其他事务所做的修改相隔离。事务查看数据时数据所处的状态，要么是被另一并发事务修改之前的状态，要么是被另一并发事务修改之后的状态，即事务不会查看由另一个并发事务正在修改的数据。这种隔离方式也叫可串行性。

## Durability持久性

**持久性（Durability）**：事务完成之后，它对系统的影响是**永久的**，即使出现系统故障也是如此。

# 事务隔离级别（Isolation Level）

事务隔离意味着对于某一个正在运行的事务来说，好像系统中只有这一个事务，其他并发的事务都不存在一样。

大部分情况下，很少使用完全隔离的事务。但不完全隔离的事务会带来如下一些问题。

* **更新丢失（Lost Update）**：两个事务都企图去更新一行数据，导致事务抛出异常退出，两个事务的更新都白费了。
* **脏读（Dirty Read）**：如果第二个应用程序使用了第一个应用程序修改过的数据，而这个数据处于未提交状态，这时就会发生脏读。第一个应用程序随后可能会请求回滚被修改的数据，从而导致第二个事务使用的数据被损坏，即所谓的“变脏”。
* **不可重读（Unrepeatable Read）**：一个事务两次读同一行数据，可是这两次读到的数据不一样，就叫**不可重读**。如果一个事务在提交数据之前，另一个事务可以修改和删除这些数据，就会发生不可重读。
* **幻读（Phantom Read）**：一个事务执行了两次查询，发现第二次查询结果比第一次查询多出了一行，这可能是因为另一个事务在这两次查询之间插入了新行。

针对由事务的不完全隔离所引起的上述问题，提出了**一些隔离级别**，用来防范这些问题。

**读操作未提交（Read Uncommitted）**：**读取未提交的数据是允许的**。说明一个事务在提交前，其变化对于其他事务来说是可见的。这样脏读、不可重读和幻读都是允许的。当一个事务已经写入一行数据但未提交，其他事务都不能再写入此行数据；但是，任何事务都可以读任何数据。这个隔离级别使用**排写锁**实现。

**读操作已提交（Read Committed）**：读取未提交的数据是不允许的，它使用**临时的共读锁和排写锁**实现。这种隔离级别不允许脏读，但不可重读和幻读是允许的。

**可重读（Repeatable Read）**：说明事务保证能够再次读取相同的数据而不会失败。此隔离级别不允许脏读和不可重读，但幻读会出现。

**可串行化（Serializable）**：提供最严格的事务隔离。这个隔离级别不允许事务并行执行，只允许串行执行。这样，脏读、不可重读或幻读都不可能发生。



# 事务的传播（Propagation）

## 事务传播行为类型：7种

|  |  |
| --- | --- |
| **事务传播行为类型** | **说明** |
| **PROPAGATION\_REQUIRED** | **如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是 最常见的选择。** |
| **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** | **使用当前的事务，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。** |
| **PROPAGATION\_NESTED** | **如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与 PROPAGATION\_REQUIRED 类似的操作。** |

当使用 PROPAGATION\_NESTED 时， 底层的数据源必须基于 JDBC 3.0 ，并且实现者需要支持**保存点**事务机制。

## readOnly

事务属性中的**readOnly标志**表示对应的事务应该被最优化为**只读事务**。这是一个最优化提示 。在一些情况下，一些事务策略能够起到显著的最优化效果，例如在使用Object/Relational映射工具 （如：Hibernate或TopLink）时避免dirty checking（试图“刷新”）。

## Timeout

在事务属性中还有定义“timeout”值的选项，**指定事务超时为几秒**。在JTA中，这将被简单地传递到J2EE服务器的事务协调程序，并据此得到相应的解释。

## 举例说明

1 ServiceA {

2

3 void methodA() {

4 try {

5 //savepoint

6 ServiceB.methodB();

7 }

8 catch (SomeException) {

9 // 执行其他业务, 如 ServiceC.methodC();

10 }

11 }

12 }

### **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。**

**例如：**

        ServiceB.methodB的事务级别定义为PROPAGATION\_REQUIRED； ServiceA.methodA已经起了事务，这时调用ServiceB.methodB，ServiceB.methodB就加入ServiceA.methodA的事务内部，就不再起新的事务。ServiceA.methodA没有在事务中，这时调用ServiceB.methodB，ServiceB.methodB就会为自己分配一个事务。

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。**

### PROPAGATION\_**SUPPORTS**

如果当前在事务中，即以事务的形式运行，如果当前不在一个事务中，那么就以非事务的形式运行。

### 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。

### PROPAGATION\_REQUIRES\_NEW

例如： ServiceA.methodA的事务级别为PROPAGATION\_**REQUIRED**，ServiceB.methodB的事务级别为PROPAGATION\_**REQUIRES\_NEW**，当调用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。

PROPAGATION\_**REQUIRES\_NEW**与PROPAGATION\_**REQUIRED** 的事务区别在于事务的回滚程度：因为ServiceB.methodB和ServiceA.methodA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。

### PROPAGATION\_**NOT\_SUPPORTED**

当前不支持事务。

例如:

　　ServiceA.methodA的事务级别是PROPAGATION\_**REQUIRED** ，而ServiceB.methodB的事务级别是PROPAGATION\_**NOT\_SUPPORTED** ，调用ServiceB.methodB时，ServiceA.methodA的事务挂起，而以**非事务的状态**运行完，再继续ServiceA.methodA的事务。

### 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。

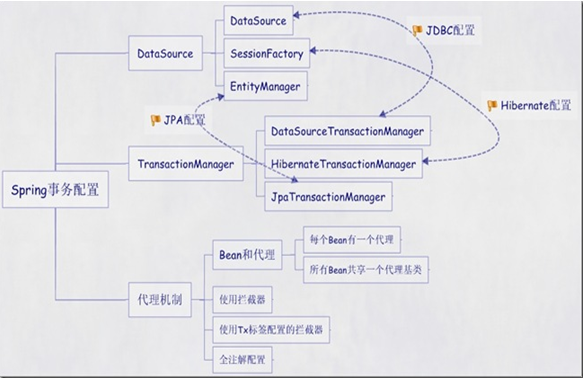
### PROPAGATION\_NESTED

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**。

# Spring事务处理

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**。



## **第一种方式：每个Bean都有一个代理**

<?xml version="1.0" encoding="UTF-8"?>

2. <beans xmlns="http://www.springframework.org/schema/beans"

3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4. xmlns:context="http://www.springframework.org/schema/context"

5. xmlns:aop="http://www.springframework.org/schema/aop"

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

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

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

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

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

11.

12. <bean id="sessionFactory"

13. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

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

16. </bean>

17.

18. <!-- 定义事务管理器（声明式的事务） -->

19. <bean id="transactionManager"

20. class="org.springframework.orm.hibernate3.HibernateTransactionManager">

21. <property name="**sessionFactory**" ref="sessionFactory" />

22. </bean>

23.

24. <!-- 配置DAO -->

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

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

27. </bean>

28.

29. <bean id="userDao"

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

31. <!-- 配置事务管理器 -->

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

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

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

35. <!-- 配置事务属性 -->

36. <property name="transactionAttributes">

37. <props>

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

39. </props>

40. </property>

41. </bean>

42. </beans>

## 第二种方式：所有Bean共享一个代理基类

<?xml version="1.0" encoding="UTF-8"?>

2 <beans xmlns="http://www.springframework.org/schema/beans"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xmlns:context="http://www.springframework.org/schema/context"

5 xmlns:aop="http://www.springframework.org/schema/aop"

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

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

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

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

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

11

12 <bean id="sessionFactory"

13 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

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

16 </bean>

17

18 <!-- 定义事务管理器（声明式的事务） -->

19 <bean id="transactionManager"

20 class="org.springframework.orm.hibernate3.HibernateTransactionManager">

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

22 </bean>

23

24 <bean id="transactionBase"

25 class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"

26 lazy-init="true" abstract="true">

27 <!-- 配置事务管理器 -->

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

29 <!-- 配置事务属性 -->

30 <property name="transactionAttributes">

31 <props>

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

33 </props>

34 </property>

35 </bean>

36

37 <!-- 配置DAO -->

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

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

40 </bean>

41

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

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

44 </bean>

45 </beans>

## 第三种方式：使用拦截器

1 <?xml version="1.0" encoding="UTF-8"?>

2 <beans xmlns="http://www.springframework.org/schema/beans"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xmlns:context="http://www.springframework.org/schema/context"

5 xmlns:aop="http://www.springframework.org/schema/aop"

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

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

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

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

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

11

12 <bean id="sessionFactory"

13 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

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

16 </bean>

17

18 <!-- 定义事务管理器（声明式的事务） -->

19 <bean id="transactionManager"

20 class="org.springframework.orm.hibernate3.HibernateTransactionManager">

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

22 </bean>

23

24 <bean id="transactionInterceptor"

25 class="org.springframework.transaction.interceptor.TransactionInterceptor">

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

27 <!-- 配置事务属性 -->

28 <property name="transactionAttributes">

29 <props>

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

31 </props>

32 </property>

33 </bean>

34

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

36 <property name="beanNames">

37 <list>

38 <value>\*Dao</value>

39 </list>

40 </property>

41 <property name="interceptorNames">

42 <list>

43 <value>transactionInterceptor</value>

44 </list>

45 </property>

46 </bean>

47

48 <!-- 配置DAO -->

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

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

51 </bean>

52 </beans>

## 第四种方式：使用tx标签配置的拦截器

1 <?xml version="1.0" encoding="UTF-8"?>

2 <beans xmlns="http://www.springframework.org/schema/beans"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xmlns:context="http://www.springframework.org/schema/context"

5 xmlns:aop="http://www.springframework.org/schema/aop"

6 xmlns:tx="http://www.springframework.org/schema/tx"

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

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

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

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

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

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

13

14 <context:annotation-config />

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

16

17 <bean id="sessionFactory"

18 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

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

21 </bean>

22

23 <!-- 定义事务管理器（声明式的事务） -->

24 <bean id="transactionManager"

25 class="org.springframework.orm.hibernate3.HibernateTransactionManager">

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

27 </bean>

28

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

30 <tx:attributes>

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

32 </tx:attributes>

33 </tx:advice>

34

35 <aop:config>

36 <aop:pointcut id="interceptorPointCuts"

37 expression="execution(\* com.bluesky.spring.dao.\*.\*(..))" />

38 <aop:advisor advice-ref="txAdvice"

39 pointcut-ref="interceptorPointCuts" />

40 </aop:config>

41 </beans>

## 第五种方式：全注解

1 <?xml version="1.0" encoding="UTF-8"?>

2 <beans xmlns="http://www.springframework.org/schema/beans"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xmlns:context="http://www.springframework.org/schema/context"

5 xmlns:aop="http://www.springframework.org/schema/aop"

6 xmlns:tx="http://www.springframework.org/schema/tx"

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

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

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

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

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

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

13

14 <context:annotation-config />

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

16

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

18

19 <bean id="sessionFactory"

20 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

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

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

23 </bean>

24

25 <!-- 定义事务管理器（声明式的事务） -->

26 <bean id="**transactionManager**"

27 class="org.springframework.orm.hibernate3.HibernateTransactionManager">

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

29 </bean>

30

31 </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 {

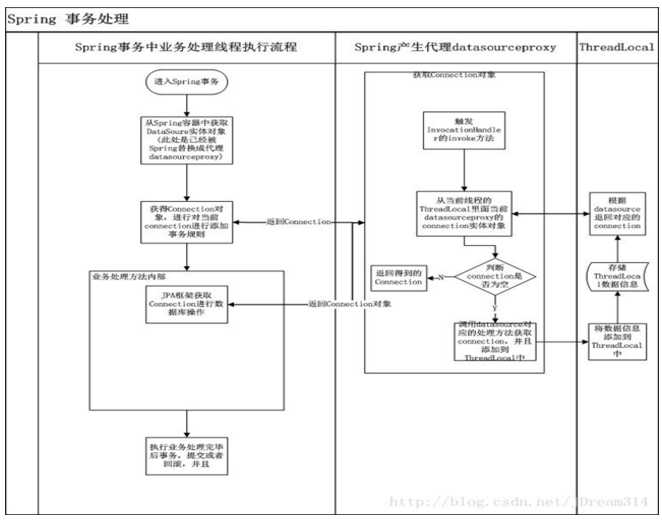
public List<User> listUsers() {

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

}

}

# Spring事务处理原理



问题：

1、当JPA框架对数据库进行操作的时候，是从哪里获取Connection？

2、jdbc对事务的配置，比如事务的开启，提交以及回滚是在哪里设置的？

3、Spring是通过**aop拦截切面的所有需要进行事务管理的业务处理方法**，那如何获取业务处理方法里面对数据库操作的事务呢？

解答：

1、既然在JPA的框架里面配置了datasource，那自然会从这个datasource里面去获得连接。

2、jdbc的事务配置是在Connection对象里面有对应的方法，比如**setAutoCommit,commit,rollback**这些方法就是对事务的操作。

3、Spring需要操作事务，那必须要对Connection来进行设置。Spring的AOP可以拦截业务处理方法，并且也知道业务处理方法里面的 DAO操作的JAP框架是从datasource里面获取Connection对象，那么Spring需要对当前拦截的业务处理方法进行事务控制，那 　　　　必然 需要得到他内部的Connection对象。整体的结构图如下：

