#### 一、事务的基础知识

1. 数据库事务：复杂的事务要分步执行，要么整体生效、要么整体失效。
2. 必须满足：原子性、一致性、隔离性、持久性。
3. 数据并发问题：脏读：A读取了B未提交的更改数据。

         不可重复读：A两次读，第二次读到了B已经提交的数据。（行级锁）

         幻读(虚读)：A读取B新提交的新增数据。（需添加表级锁）

         第一类丢失更新：A撤销时恢复原数据把B提交的数据覆盖了。

         第二类丢失更新：A提交时覆盖了B已经提交的数据。

1. 数据库锁机制：一般分为表锁和行锁，按并发来分有共享锁和独占锁。数据库必须在更改的行上施加独占锁；行共享锁、行独占锁、表共享锁、表共享行独占锁、表独占锁。
2. 事务隔离级别：



#### 二、ThreadLocal

1. 概念：ThreadLocal是保存线程本地化的容器，为每个使用该变量的线程分配一个独立的变量副本。
2. 原理：通过Map来保存每个线程的变量副本，key为线程对象，值为线程的副本。

public class TestNum {    
    // ①通过匿名内部类覆盖ThreadLocal的initialValue()方法，指定初始值    
    private static ThreadLocal<Integer> seqNum = new ThreadLocal<Integer>() {    
        public Integer initialValue() {    
            return 0;    
        }    
    };

    // ②获取下一个序列值    
    public int getNextNum() {    
        seqNum.set(seqNum.get() + 1);    
        return seqNum.get();    
    }    
    
    public static void main(String[] args) {    
        TestNum sn = new TestNum();    
        // ③ 3个线程共享sn，各自产生序列号    
        TestClient t1 = new TestClient(sn);    
        TestClient t2 = new TestClient(sn);    
        TestClient t3 = new TestClient(sn);    
        t1.start();    
        t2.start();    
        t3.start();    
    }    
    
    private static class TestClient extends Thread {    
        private TestNum sn;    
    
        public TestClient(TestNum sn) {    
            this.sn = sn;    
        }    
    
        public void run() {    
            for (int i = 0; i < 3; i++) {    
                // ④每个线程打出3个序列值    
  System.out.println("thread[" + Thread.currentThread().getName() + "] --> sn["    
                + sn.getNextNum() + "]");    
            }    
        }    
    }    
}

在spring中大部分Bean都可以声明为singleton，故而spring要对这些非线程安全bean采用Threadlocal进行封装，这样有状态的bean就能够以sigleton的方式在多线程中正常工作。spring通过Threadlocal来实现事务管理

#### 三、spring对事务的支持

1. spring封装了事务模版类TranscationTemplate.
2. 事务管理主要有3个接口：PlatformTransactionManager,TransactionDefinition,TransactionStatus.

TransactionDefinition：用于描述事务的隔离级别、超时时间、等事务属性；

PlatformTransactionManager根据TransactionDefinition提供的事务属性来创建事务；就三个方法getTransaction、commit、rollback；

TransactionStatus描述激活事务的状态；

3.spring把事务管理委托给底层具体的持久化实现框架来完成，为不同的持久化框架提供了不同的PlatformTransactionManager接口的实现类。

<bean id="dataSource"      
      class="org.apache.commons.dbcp.BasicDataSource"  
      destroy-method="close"  
      p:driverClassName="${jdbc.driverClassName}"  
      p:url="${jdbc.url}"  
      p:username="${jdbc.username}"  
      p:password="${jdbc.password}"/>  
<bean id="transactionManager"              2）基于数据源的事务管理器  
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"  
      p:dataSource-ref="dataSource"/>      3）引用数据源

#### 四、声明事务

1.基于aop/tx命名空间的配置：spring在Schema的配置中添加了一个tx命名空间，在配置文件中定义事务属性。

<?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:p="http://www.springframework.org/schema/p" 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-4.0.xsd  
     http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">  
  
    <import resource="classpath:applicationContext-dao.xml"/>  
  
    <!--配置事务管理器-->  
    <bean id="transactionManager"  
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager"  
          p:dataSource-ref="dataSource"/>  
    <!--使用切点表达式定义目标方法-->  
    <aop:config>  
        <!--定义切面-->  
        <aop:pointcut id="serviceMethod"  
                      expression="execution(\* com.smart.service.\*Forum.\*(..))"/>  
        <!--引用事务增强-->  
        <aop:advisor pointcut-ref="serviceMethod"  
                     advice-ref="txAdvice"/>  
    </aop:config>  
    <!--定义事务增强-->  
    <tx:advice id="txAdvice" transaction-manager="txManager">  
        <tx:attributes>  
            <tx:method name="get\*" read-only="false"/>  
            <tx:method name="add\*" rollback-for="PessimisticLockingFailureException"/>  
            <tx:method name="update\*"/>  
        </tx:attributes>  
    </tx:advice>  
    <!--  
        1.引入tx命名空间的声明  
        2.配置事务管理器  
        3.定义事务增强:事务增强一定需要一个事务管理器的支持,<tx:advice>通过transaction-manager属性引用之前定义的事务管理器,默认查找名为transactionManager  
                       所以如果事务管理器命名为transactionManager,可以不指定transaction-manager属性.  
        4.使用AOP  
    -->  
    <!--tx:method的属性  
        name 就name是必须的属性 与事务属性关联的方法名 get\* handle\*等自己起的名字  
        isolation 事务隔离级别  
        timeout 默认为-1 超时时间  
        read-only 默认false 事务是否只读  
        rollback-for 默认所有运行期间异常都滚回   
        no-rollback-for 默认所有检查型异常都不滚回  
    -->

2.使用注解配置声明式事务

1）在xml中配置

<bean id="txManager"  
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"  
      p:dataSource-ref="dataSource"/>  
  
<tx:annotation-driven transaction-manager="txManager" proxy-target-class="true"/>  
<!--同上，如果名字是transactionManager 可以省略  
    proxy-target-class="true"表示spring会通过创建子类来代理业务类  
-->

2）在业务类上注解

@Service  
@Transactional  
public class BbtForum {  
    public ForumDao forumDao;  
  
    public TopicDao topicDao;  
  
    public PostDao postDao;  
    ...}

3）@Transactional的属性

默认属性：

事务传播行为：PROPAGATION\_REQUIRED

事务隔离级别：IOSLATION\_DEFAULT

读写事务属性：读/写事务

超时时间：-1

回滚设置：热河运行期异常引发回滚、任何检查型异常不会引发回滚

4）spring要在具体的业务类上使用@Transactional注解

5）在方法处使用注解会覆盖类定义的注解，如果方法需要使用特殊的事务属性，可以在方法上使用注解

@Transactional(readOnly = true)  
public Forum getForum(int forumId) {  
    return forumDao.getForum(forumId);  
}

五、事务的一些注意点

1.使用不同的事务管理器

@Transactional("name")使用名为name的事务管理器

2.事务管理的目的是保证数据操作的事务性(原子性、一致性、隔离性、持久性)脱离了事务，DAO一样可以进行数据操作。

3.事务的传播性

PROPAGATION\_REQUIERD:如果当前没有事务就创建一个，有就加进去

PROPAGATION\_SUPPORTS:支持当前事务，没有就以非事务方式执行

PROPAGATION\_MANDATORY:使用当前事务，没有就抛出异常

PROPAGATION\_REQURES\_NEW:新建事务，如果当前存在事务，就将其挂起

PROPAGATION\_NOT\_SUPPORTED:以非事务方式执行，如果当前有就挂起

PROPAGATION\_NEVER:以非事务方式执行，如果有就挂起

PROPAGATION\_NESTED:嵌套事务

相同线程中进行互相嵌套调用的事务方法工作在相同的事务中，如果在不同线程，则不同线程下事务方法工作在独立的事务中。