**一、什么是ThreadLocal**

早在JDK 1.2的版本中就提供java.lang.ThreadLocal，ThreadLocal为解决多线程程序的并发问题提供了一种新的思路。使用这个工具类可以很简洁地编写出优美的多线程程序。

ThreadLocal很容易让人望文生义，想当然地认为是一个“本地线程”。其实，ThreadLocal并不是一个Thread，而是Thread的局部变量，也许把它命名为ThreadLocalVariable更容易让人理解一些。

当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。

从线程的角度看，目标变量就象是线程的本地变量，这也是类名中“Local”所要表达的意思。

线程局部变量并不是Java的新发明，很多语言（如IBM IBM XL FORTRAN）在语法层面就提供线程局部变量。在Java中没有提供在语言级支持，而是变相地通过ThreadLocal的类提供支持。

所以，在Java中编写线程局部变量的代码相对来说要笨拙一些，因此造成线程局部变量没有在Java开发者中得到很好的普及。

**ThreadLocal的接口方法**

ThreadLocal类接口很简单，只有4个方法，我们先来了解一下：

²  **void set(Object value)**

设置当前线程的线程局部变量的值。

²  **public Object get()**

该方法返回当前线程所对应的线程局部变量。

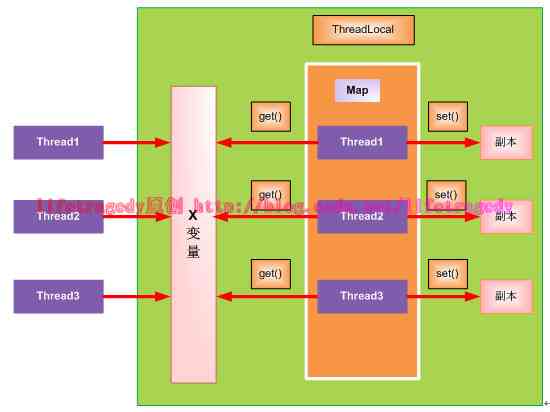
²  **public void remove()**

将当前线程局部变量的值删除，目的是为了减少内存的占用，该方法是JDK 5.0新增的方法。需要指出的是，当线程结束后，对应该线程的局部变量将自动被垃圾回收，所以显式调用该方法清除线程的局部变量并不是必须的操作，但它可以加快内存回收的速度。

²  **protected ObjectinitialValue()**

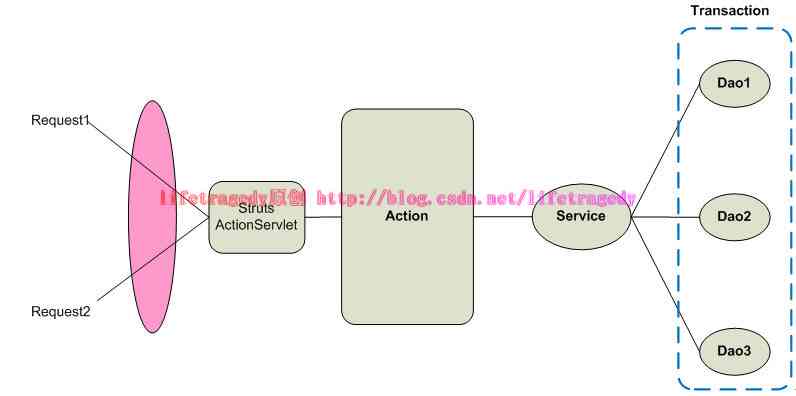
返回该线程局部变量的初始值，该方法是一个protected的方法，显然是为了让子类覆盖而设计的。这个方法是一个延迟调用方法，在线程第1次调用get()或set(Object)时才执行，并且仅执行1次。ThreadLocal中的缺省实现直接返回一个null。

值得一提的是，在JDK5.0中，ThreadLocal已经支持泛型，该类的类名已经变为ThreadLocal<T>。API方法也相应进行了调整，新版本的API方法分别是void set(T value)、T get()以及T initialValue()。



**一、来看一个实际案例**

**2.1 同一Service方法中调用多个Dao方法**



可以看到，我们有一个Service方法，在该Service方法中调用多个Dao方法，所有在该Service方法中的的Dao都处于同一事务中。

该Service方法结束后，提交事务；

该Service方法中有任何错，回滚事务；

**2.2 传统的做法**

来看下面这段伪代码

**Service层代码：**

|  |
| --- |
| **public void serviceMethod(){**  **Connection conn=null;**  **try{**  **Connection conn=getConnection();**  **conn.setAutoCommit(false);**  **Dao1 dao1=new Dao1(conn);**  **dao1.doSomething();**  **Dao2 dao2=new Dao2(conn);**  **dao2.doSomething();**  **Dao3 dao3=new Dao3(conn);**  **dao3.doSomething();         conn.commit();**  **}catch(Exception e){**  **try{**  **conn.rollback();**  **}catch(Exception ex){}**  **}finally{**  **try{**  **conn.setAutoCommit(true);**  **}catch(Exception e){}**  **try{**  **if(conn!=null){**  **conn.close();**  **conn=null;**  **}**  **}catch(Exception e){}**  **}**  **}** |

**每个Dao层的代码：**

|  |
| --- |
| **Class Dao1{**  **private Connection conn=null;**  **public Dao1(Connection conn){**  **this.conn=conn;**  **}**  **public void doSomething(){**  **PreparedStatement pstmt=null;**  **try{**  **pstmt=conn.preparedStatement(sql);**  **pstmt.execute…**  **…**  **}catch(Exception e){**  **log.error(e,”Exeception occurred in Dao1.doSomething():”+e.getMessage,e);**  **}finally{**  **try{**  **if(pstmt!=null){**  **pstmt.close();**  **pstmt=null;**  **}**  **}catch(Exception e){}**  **}**  **}**  **}** |

如果我一个Service方法有调用一堆dao方法，先不说这样写首先破坏了OOP的封装性原则，如果有一个dao多关了一个conn，那就会导致其它的dao得到的conn为null，这种事在这样的写法下由其当你还有业务逻辑混合在一起时很容易发生。

笔者曾经遇见过2个项目，出现out of memory或者是connection pool has been leakage，经查代码就是在每个dao中多关或者在service层中漏关，或者是每个dao有自己的conntionconn=getConnection()，然后还跑到Service层里去关这个connection（那关什么，关个P关！）。

当然，如果你说你在写法上绝对promise绝对注意这样的问题不会发生，但是我们来看看下面的这种做法，是否会比上面这个写法更好呢？

**2.3 Spring中的做法**

先来看Spring中的写法。

大家应该都很熟悉Spring中的写法了，来看一下它是怎么解决的。

**Service层**

|  |
| --- |
| public void serviceMethod(){  try{      //aop 自动加入connection，并且将conn.setAutoCommit(false);  dao1.doSomething();  dao2.doSomething();  dao3.doSomething();  }catch(Exception e){      //aop 自动加入rollback  }finally{      //aop自动加入conn.setAutoCommit(true)      //aop 自动加入conn.close();  } |

这边我们不讲AOP，因为用类反射结合xml很容易将aop 自动。。。这些东西加入我们的代码中去是不是？我们只管写dao方法，service方法，不需要关心在哪边commit哪边rollback何时connection，spring的声明式事务会帮我们负责，这种风格我们称为“优雅”，各层间耦合度极大程度上的降低，封装性好。

因此，我们可以总结出下面这些好处：

²  Service层的方法只管开启事务（如果讲究点的还会设一个Transaction）；

²  在该Service层中的所有dao使用该service方法中开启的事务（即connection）；

²  Dao中每次只管getCurrentConnection（获取当前的connection），与进行数据处理

²  Dao层中如果发生错误就抛回Service层

²  Service层中接到exception，在catch{}中rollback，在try{}未尾commit，在finally块中关闭整个connection。

这。。。就是我们所说的ThreadLocal。

**举个更实际的例子再次来说明ThreadLocal：**

我们有3个用户访问同一个service方法，该service方法内有3个dao方法为一个完整事务，那么整个web容器内只因该有3个connection，并且每个connection之间的状态，彼此“隔离”。

**我们下面一起来看我们如何用代码实现类似于Spring的这种做法。**

首先，根据我们的ThreadLocal的概念，我们先声明一个ConnectionManager的类。

**2.4 利用ThreadLocal制作ConnectionManager**

|  |
| --- |
| **public class ConnectionManager {**  **private static ThreadLocal tl = new ThreadLocal();**  **private static Connection conn = null;**  **public static void BeginTrans(boolean beginTrans) throws Exception {**  **if (tl.get() == null || ((Connection) tl.get()).isClosed()) {**  **conn = SingletonDBConnection.getInstance().getConnection();**  **conn = new ConnectionSpy(conn);**  **if (beginTrans) {**  **conn.setAutoCommit(false);**  **}**  **tl.set(conn);**  **}**  **}**  **public static Connection getConnection() throws Exception {**  **return (Connection) tl.get();**  **}**  **public static void close() throws SQLException {**  **try {**  **((Connection) tl.get()).setAutoCommit(true);**  **} catch (Exception e) {**  **}**  **((Connection) tl.get()).close();**  **tl.set(null);**  **}**  **public static void commit() throws SQLException {**  **try {**  **((Connection) tl.get()).commit();**  **} catch (Exception e) {**  **}**  **try {**  **((Connection) tl.get()).setAutoCommit(true);**  **} catch (Exception e) {**  **}**  **}**  **public static void rollback() throws SQLException {**  **try {**  **((Connection) tl.get()).rollback();**  **} catch (Exception e) {**  **}**  **try {**  **((Connection) tl.get()).setAutoCommit(true);**  **} catch (Exception e) {**  **}**  **}**  **}** |

**2.5 利用ThreadLocal改造Service与Dao层**

**Service层（注意红色标粗-好粗yeah，的地方）**

|  |
| --- |
| package sky.org.service.impl;  public class StudentServiceImpl implements StudentService {           public void addStudent(Student std) throws Exception {  **StudentDAO studentDAO = new StudentDAOImpl();**  **ClassRoomDAO classRoomDAO = new ClassRoomDAOImpl();**                     try {                              ConnectionManager.BeginTrans(true);  **studentDAO.addStudent(std);**  **classRoomDAO**  **.addStudentClassRoom(std.getClassRoomId(), std.getsNo());**  **ConnectionManager.commit();**                     } catch (Exception e) {  **try {**  **ConnectionManager.rollback();**  **} catch (Exception de) {**  **}**                              throw new Exception(e);                     }**finally {**  **try {**  **ConnectionManager.close();**  **} catch (Exception e) {**  **}**  **}**           }  } |

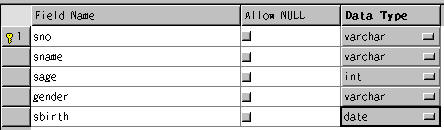
Look，如果我把上述标粗（没有加**红色**）的地方，全部用AOP的方式从这块代码的外部“切”进去。。。是不是一个Spring里的Service方法就诞生了？

下面来看一个完整的例子

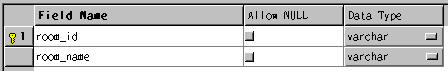
**2.6 使用ThreadLocal分离Service、DAO层**

**先来看表结构：**

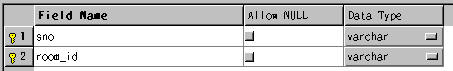
**T\_Student表**



**T\_ClassRoom表**



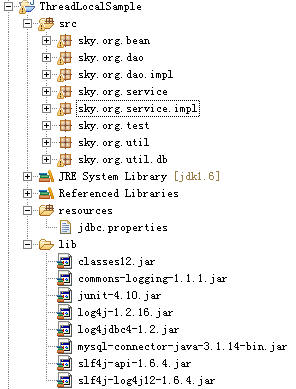
**T\_Student\_ClassRoom表**

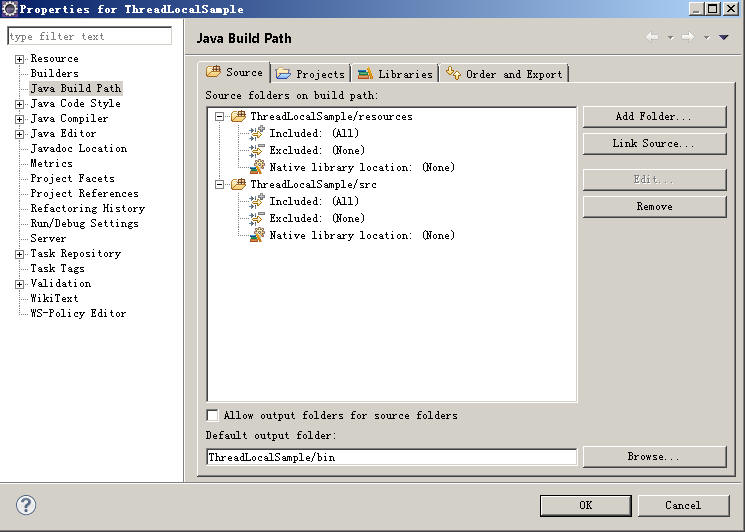


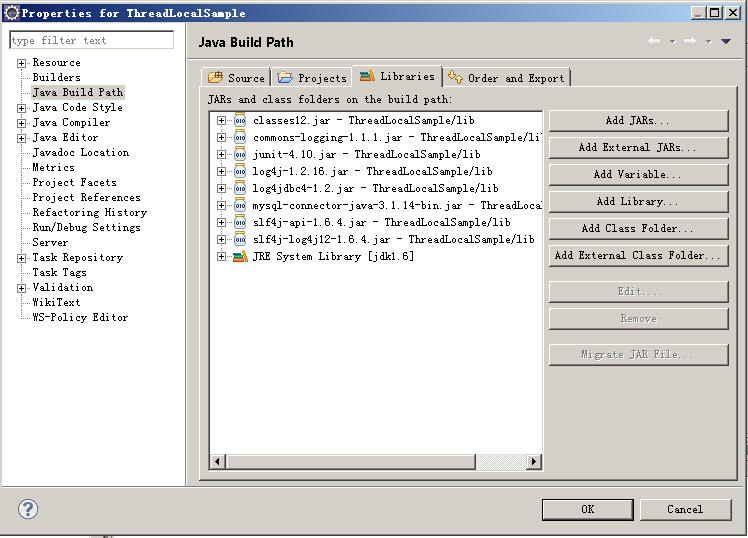
**需求：**

很简单，T\_ClassRoom表里已经有值了，在插入T\_Student表的数据时同时要给这个学生分配一个班级并且插入T\_Student\_ClassRoom表，这就是一个事务，这两步中有任何一步出错，事务必须回滚。

**看来工程的结构吧：**







下面开始放出所有源代码：

**2.6.1 ConnectionManager类**

|  |
| --- |
| package sky.org.util.db;  import java.sql.\*;  public class ConnectionManager {           private static ThreadLocal tl = new ThreadLocal();           private static Connection conn = null;           public static void BeginTrans(boolean beginTrans) throws Exception {                     if (tl.get() == null || ((Connection) tl.get()).isClosed()) {                              conn = DBConnection.getInstance().getConnection();                              if (beginTrans) {                                       conn.setAutoCommit(false);                              }                              tl.set(conn);                     }           }           public static Connection getConnection() throws Exception {                    return (Connection) tl.get();           }           public static void close() throws SQLException {                     try {                              ((Connection) tl.get()).setAutoCommit(true);                     } catch (Exception e) {                     }                     ((Connection) tl.get()).close();                     tl.set(null);           }           public static void commit() throws SQLException {                     try {                              ((Connection) tl.get()).commit();                     } catch (Exception e) {                     }                     try {                              ((Connection) tl.get()).setAutoCommit(true);                     } catch (Exception e) {                     }           }           public static void rollback() throws SQLException {                     try {                              ((Connection) tl.get()).rollback();                     } catch (Exception e) {                     }                     try {                              ((Connection) tl.get()).setAutoCommit(true);                     } catch (Exception e) {                     }           }  } |

**2.6.2 DBConnection类**

|  |
| --- |
| package sky.org.util.db;  public class DBConnection {           private static DBConnection instance = null;           private static String driverClassName = null;           private static String connectionUrl = null;           private static String userName = null;           private static String password = null;           private static Connection conn = null;           private static Properties jdbcProp = null;           private DBConnection() {           }           private static Properties getConfigFromPropertiesFile() throws Exception {                     Properties prop = null;                     prop = JdbcProperties.getPropObjFromFile();                     return prop;           }           private static void initJdbcParameters(Properties prop) {                     driverClassName = prop.getProperty(Constants.DRIVER\_CLASS\_NAME);                     connectionUrl = prop.getProperty(Constants.CONNECTION\_URL);                     userName = prop.getProperty(Constants.DB\_USER\_NAME);                     password = prop.getProperty(Constants.DB\_USER\_PASSWORD);           }           private static void createConnection() throws Exception {                     Class.forName(driverClassName);                     conn = DriverManager.getConnection(connectionUrl, userName, password);           }           public static Connection getConnection() throws Exception {                     return conn;           }           public synchronized static DBConnection getInstance()throws Exception{                     if (instance == null) {                              jdbcProp = getConfigFromPropertiesFile();                              instance = new DBConnection();                     }                     initJdbcParameters(jdbcProp);                     createConnection();                     return instance;           }  } |

**2.6.3 JdbcProperties类**

|  |
| --- |
| package sky.org.util.db;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStream;  import java.net.URL;  import java.util.\*;  public class JdbcProperties {           private static Log logger = LogFactory.getLog(JdbcProperties.class);           public static Properties getPropObjFromFile() {                     Properties objProp = new Properties();                     ClassLoader classLoader = Thread.currentThread()                                       .getContextClassLoader();                     URL url = classLoader.getResource(Constants.JDBC\_PROPERTIES\_FILE);                     if (url == null) {                              classLoader = ClassLoader.getSystemClassLoader();                              url = classLoader.getResource(Constants.JDBC\_PROPERTIES\_FILE);                     }                     File file = new File(url.getFile());                     InputStream inStream = null;                     try {                              inStream = new FileInputStream(file);                              objProp.load(inStream);                     } catch (FileNotFoundException e) {                              objProp = null;                              e.printStackTrace();                     } catch (IOException e) {                              e.printStackTrace();                     } finally {                              try {                                       if (inStream != null) {                                                 inStream.close();                                                 inStream = null;                                       }                              } catch (Exception e) {                              }                     }                     return objProp;           }  } |

**2.6.4 Resource目录下的jdbc.properties**

|  |
| --- |
| jdbc.driverClassName=com.mysql.jdbc.Driver  jdbc.databaseURL=jdbc:mysql://localhost:3306/mydb?useUnicode=true&characterEncoding=utf8  jdbc.username=mysql  jdbc.password=password\_1 |

**2.6.5 StudentService接口**

|  |
| --- |
| package sky.org.service;  import java.util.List;  import java.util.Vector;  import sky.org.bean.\*;  public interface StudentService {           public void addStudent(Student std) throws Exception;  } |

**2.6.6 StudentServiceImpl类**

|  |
| --- |
| package sky.org.service.impl;  import java.util.ArrayList;  import java.util.List;  import java.util.Vector;  import sky.org.util.db.ConnectionManager;  import sky.org.util.\*;  import sky.org.bean.\*;  import sky.org.dao.\*;  import sky.org.dao.impl.\*;  import sky.org.service.\*;  public class StudentServiceImpl implements StudentService {             public void addStudent(Student std) throws Exception {                     StudentDAO studentDAO = new StudentDAOImpl();                     ClassRoomDAO classRoomDAO = new ClassRoomDAOImpl();                     try {                              ConnectionManager.BeginTrans(true);                              studentDAO.addStudent(std);                              classRoomDAO                                                 .addStudentClassRoom(std.getClassRoomId(), std.getsNo());                              ConnectionManager.commit();                     } catch (Exception e) {                              try {                                       ConnectionManager.rollback();                              } catch (Exception de) {                              }                              throw new Exception(e);                     } finally {                              try {                                       ConnectionManager.close();                              } catch (Exception e) {                              }                     }           }  } |

**2.6.7 ClassRoomDAO接口**

|  |
| --- |
| package sky.org.dao;  import java.util.HashMap;  import java.util.List;  public interface ClassRoomDAO {           public void addStudentClassRoom(String roomId, String sNo) throws Exception;  } |

**2.6.8 ClassRoomDAOImpl类**

|  |
| --- |
| package sky.org.dao.impl;    import java.sql.\*;  import java.util.\*;    import sky.org.dao.ClassRoomDAO;  import sky.org.util.db.ConnectionManager;    public class ClassRoomDAOImpl implements ClassRoomDAO {             public void addStudentClassRoom(String roomId, String sNo) throws Exception {                     Connection conn = null;                     PreparedStatement pstmt = null;                     try {                              conn = ConnectionManager.getConnection();                              pstmt = conn                                                 .prepareStatement(ClassRoomDAOSql.ADD\_STUDENT\_CLASSROOM);                              pstmt.setString(1, roomId);                              pstmt.setString(2, sNo);                              pstmt.executeUpdate();                     } catch (Exception e) {                              throw new Exception("addStudentClassRoom:" + e.getMessage(), e);                     } finally {                              try {                                       if (pstmt != null) {                                                 pstmt.close();                                                 pstmt = null;                                       }                              } catch (Exception e) {                              }                     }           }  } |

**2.6.9 StudentDAO接口**

|  |
| --- |
| package sky.org.dao;  import java.util.\*;  import sky.org.bean.Student;  public interface StudentDAO {           public void addStudent(Student std) throws Exception;  } |

**2.6.10 StudentDAOImpl类**

|  |
| --- |
| package sky.org.dao.impl;    import java.sql.\*;  import javax.sql.\*;    import org.apache.commons.logging.Log;  import org.apache.commons.logging.LogFactory;    import sky.org.bean.Student;  import sky.org.dao.StudentDAO;  import sky.org.util.db.ConnectionManager;    import java.util.List;  import java.util.ArrayList;  import java.util.Vector;  import java.text.\*;  import sky.org.util.StringUtil;    public class StudentDAOImpl implements StudentDAO {           private Log logger = LogFactory.getLog(this.getClass());             public void addStudent(Student std) throws Exception {                     Connection conn = null;                     PreparedStatement pstmt = null;                     try {                              conn = ConnectionManager.getConnection();                              pstmt = conn.prepareStatement(StudentDAOSql.ADD\_STUDENT);                              pstmt.setString(1, std.getsNo());                              pstmt.setString(2, std.getsName());                              pstmt.setString(3, std.getsAge());                              pstmt.setString(4, std.getGender());                              pstmt.setDate(5, StringUtil.convertStrToDate(std.getSbirth()));                                pstmt.executeUpdate();                     } catch (Exception e) {                              throw new Exception("addStudent:" + e.getMessage(), e);                     } finally {                              try {                                       if (pstmt != null) {                                                 pstmt.close();                                                 pstmt = null;                                       }                              } catch (Exception e) {                              }                     }           }             public void delStudent(String sNo) throws Exception {                     Connection conn = null;                     PreparedStatement pstmt = null;                     try {                              conn = ConnectionManager.getConnection();                              pstmt = conn.prepareStatement(StudentDAOSql.DEL\_STUDENT);                              pstmt.setString(1, sNo);                              pstmt.executeUpdate();                     } catch (Exception e) {                              throw new Exception("delStudent:" + e.getMessage(), e);                     } finally {                              try {                                       if (pstmt != null) {                                                 pstmt.close();                                                 pstmt = null;                                       }                              } catch (Exception e) {                              }                     }           }  } |

**2.6.11 StudentDAOSql类**

|  |
| --- |
| package sky.org.dao.impl;  public class StudentDAOSql {  public final static String ADD\_STUDENT = "insert into t\_student(sno, sname, sage, gender,  sbirth)values(?,?,?,?,?)";  } |

**2.6.12 ClassRoomDAOSql类**

|  |
| --- |
| package sky.org.dao.impl;  public class ClassRoomDAOSql {           public static String ADD\_STUDENT\_CLASSROOM = "insert into  t\_student\_classroom(room\_id,sno)values(?,?)";  } |

**2.6.13 ClassRoom 类**

|  |
| --- |
| package sky.org.bean;  import java.io.\*;    public class ClassRoom implements Serializable {             private String roomId = "";           private String roomName = "";             public String getRoomId() {                     return roomId;           }             public void setRoomId(String roomId) {                     this.roomId = roomId;           }             public String getRoomName() {                     return roomName;           }             public void setRoomName(String roomName) {                     this.roomName = roomName;           }  } |

**2.6.14 Student类**

|  |
| --- |
| package sky.org.bean;    import java.io.\*;    public class Student implements Serializable {             public String getsNo() {                     return sNo;           }             public void setsNo(String sNo) {                     this.sNo = sNo;           }             public String getsName() {                     return sName;           }             public void setsName(String sName) {                     this.sName = sName;           }             public String getsAge() {                     return sAge;           }             public void setsAge(String sAge) {                     this.sAge = sAge;           }             public String getGender() {                     return gender;           }             public void setGender(String gender) {                     this.gender = gender;           }             private String sNo = "";           private String sName = "";           private String sAge = "";           private String gender = "";           private String sbirth = "";           private String classRoomId = "";           private String classRoomName = "";             public String getClassRoomId() {                     return classRoomId;           }             public void setClassRoomId(String classRoomId) {                     this.classRoomId = classRoomId;           }             public String getClassRoomName() {                     return classRoomName;           }             public void setClassRoomName(String classRoomName) {                     this.classRoomName = classRoomName;           }             public String getSbirth() {                     return sbirth;           }             public void setSbirth(String sbirth) {                     this.sbirth = sbirth;           }    } |

**2.6.15 StudentCRUD类（运行主类）**

|  |
| --- |
| package sky.org.test;    import sky.org.bean.Student;  import sky.org.service.StudentService;  import sky.org.service.impl.StudentServiceImpl;    public class StudentCRUD {             public void addStudent() throws Exception {                     StudentService stdService = new StudentServiceImpl();                     Student std = new Student();                     std.setsNo("101");                     std.setsName("abc");                     std.setSbirth("1977/01/01");                     std.setsAge("35");                     std.setGender("m");                     std.setClassRoomId("1");                     std.setClassRoomName("class1");                     stdService.addStudent(std);           }             public static void main(String[] args) {                     StudentCRUD testStudentCRUD = new StudentCRUD();                     try {                              testStudentCRUD.addStudent();                     } catch (Exception e) {                              e.printStackTrace();                              System.exit(-1);                     }             }    } |

**三、Hibernate里的ThreadLocal**

Hibernate在事务操作中也支持ThreadLocal的作法，我们这边指的是不用Spring去做代理，而直接用Hibernate。即：

|  |
| --- |
| **Service Method{**  **hbDAO1.doSomething();**  **hbDAO2.doSomething();**  **hbDAO3.doSomething();**  **。。。**  **}** |

Hibernate版本3后增加了新特性，即getCurrentSession()。

**3.1 getCurrentSession与openSession的区别**

**3.1.1 openSession**

我们传统的做法是openSession即：

|  |
| --- |
| public void testUser() throws Exception {                     Transaction tran = null;                     SessionFactory factory = null;                     UserDAO userDAO = new UserDAOImpl();                     try {                              factory = HibernateUtil.getInstance().getSessionFactory();                              Session session = factory.openSession();                              tran = session.beginTransaction();                              TUser testUser = new TUser();                          testUser.setId(new Integer(i));                              testUser.setName("abc");                              userDAO.addUser(testUser);                              tran.commit();                     } catch (Exception e) {                              tran.rollback();                              throw new Exception(e);                     } finally {                              try{      if(session!=null){      session.close();      session=null();  }  }catch(Excepton e){}                     }           } |

这样做，能够保证我们每次在finally块中正确关闭session，但是，如果我们也遇上了这样的case即：

|  |
| --- |
| **Service Method{**  **hbDAO1.doSomething();**  **hbDAO2.doSomething();**  **hbDAO3.doSomething();**  **。。。**  **}** |

这时，我们如果用的是openSession，应该怎么办？

**解决方案一：**

自己用ThreadLocal模式写一个SessionManagement类，来维护这个session。

**解决方案二：**

把在Service方法中打开的session，传到每个dao方法中，使每个dao方法使用同一个session，最后在Service方法中去关闭它（很烂的做法）。

下面我们来看看Hibernate自身提供的getCurrentSession()的做法吧

**3.1.2 getCurrentSession**

要使用这个getCurrentSession，你的hibernate的设置必须如下（红色加粗部分显示-就喜欢粗J）：

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8'?>  <!DOCTYPE hibernate-configuration PUBLIC            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"            "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>           <session-factory>                     <property name="connection.url">                              jdbc:oracle:thin:@localhost:1521:myorcl                     </property>                     <property name="dialect">                              org.hibernate.dialect.Oracle9Dialect                     </property>                     <property name="connection.username">abc</property>                     <property name="connection.password">abc</property>                     <property name="connection.driver\_class">                              oracle.jdbc.OracleDriver                     </property>                     <property name="show\_sql">true</property>                     <property name="hibernate.hbm2ddl.auto">update</property>  **<property name="hibernate.current\_session\_context\_class">thread</property>**                     <mapping resource="com/cts/testhb/model/TUser.hbm.xml" />           </session-factory>  </hibernate-configuration> |

然后上述代码将变成如下的样子：

|  |
| --- |
| public void testUser() throws Exception {                     Transaction tran = null;                     SessionFactory factory = null;                     UserDAO userDAO = new UserDAOImpl();                     try {                              factory = HibernateUtil.getInstance().getSessionFactory();  **Session session = factory.getCurrentSession();**                              tran = session.beginTransaction();                              for (int i = 0; i < 100; i++) {                                       TUser testUser = new TUser();                                       testUser.setId(new Integer(i));                                       testUser.setName("abc");                                       userDAO.addUser(testUser);                              }                              tran.commit();                     } catch (Exception e) {                              tran.rollback();                              throw new Exception(e);                     } finally {  **ThreadLocalSessionContext.unbind(factory);**                     }           } |

而你的每个DAO方法中的代码是这样实现的：

|  |
| --- |
| public void addUser(TUser user) throws Exception {           SessionFactory factory = HibernateUtil.getInstance()                              .getSessionFactory();           Session session = factory.getCurrentSession();           session.save(user);  } |

是不是很方便的哈。

**3.1.3 openSession与getCurrentSession的区别**

**严重注意下面3点：**

²  openSession一旦被调用，必须且一定要在finally块中close，要不然你就等着out of memory吧；

²  如果你使用的是getCurrentSession，那么你不能在finally块中调用”session.close()”，不行你可以在finally块中用try-catch把session.close();包起来，然后在catch{}块中抛出这个exception，这个exception将会是：sessionhas been already closed。

因为：

l   如果你用的是getCurrentSession，那么它在session.commit()或者是session.rollback()时就已经调用了一次session.close()了，因此你只要正确放置session.commit()与rollback()即可。

l   你必须在finally块中调用”ThreadLocalSessionContext.unbind(factory);”，以使得当前的事务结束时把session（即dbconnection）还回db connection pool中

²  如果你使用的是getCurrentSession，那么就算你是一个简单的select语句，也必须包含在：

|  |
| --- |
| tran = session.beginTransaction();  **//your select hibernate query**  tran.commit(); |

这样的事务块中，要不然它将会抛出这样的一个错误：

**NoHibernate Session bound to thread, and configuration does not allow creation ofnon-transactional**

看下面的例子：

|  |
| --- |
| try {                      factory = HibernateUtil.getInstance().getSessionFactory();                      Session session = factory.getCurrentSession();  **tran = session.beginTransaction();**                      TUser testUser = userDAO.getUserByID("1");                      log.info("user id===="+testUser.getId()+"  user name===="+testUser.getName());  **tran.commit();**             } catch (Exception e) {  **tran.rollback();**                      throw new Exception(e);             } finally {                      ThreadLocalSessionContext.unbind(factory);                     } |

可以看到我们的查询是被tran=session.beginTransaction一直到tran.commit()或者是tran.rollback()结束的，如果，你把你的hibernate查询移到了tran=session.beginTransaction的上面。。。就会抛上述这个错误。

**3.1.4 getCurrentSession带来的问题**

getCurrentSession非常好，不需要我们自己写ThreadLocal只需要在hibernate.cfg的配置文件中声音一下就可以获得ThreadLocal的好处，便于我们划分我们的程序的层次与封装，带也带来了一定的性能问题。

特别是“**如果你使用的是getCurrentSession，那么就算你是一个简单的select语句，也必须包含在事务块中**”。这给我们带来了很大的问题。

因此，本人建议，在碰到如果：

1.       一个service方法中只有单个dao操作且此操作是一个select类的操作，请使用openSession，并且即时在finally块中关闭它；

2.       如果一个service方法中涉及到多个dao操作，请一定使用getCurrentSession；

3.       如果一个service方法中混合着select操作，delete, update, insert操作。请按照下述原则：

1)      将属于select的操作，单独做成一个dao方法，该dao使用openSession并且在finally块中及时关闭session,该dao只需要返回一个java的object如：List<Student>即可，如果出错将exception抛回给调用它的service方法。

2)      对于其它的delete, insert, update的dao操作，请使用getCurrentSession。

4.       忌讳，把select类的操作放在“事务”中；