## ThreadLocal

ThreadLocal类只有三个方法：

void set(T value)：保存值；

T get()：获取值；

void remove()：移除值。

ThreadLocal内部其实是个Map来保存数据。虽然在使用ThreadLocal时只给出了值，没有给出键，其实它内部使用了当前线程做为键。

## DbUtils

|  |
| --- |
| **public** **class** DbUtils {  // 四大参数  **private** **static** String *drivername*;  **private** **static** String *url*;  **private** **static** String *user*;  **private** **static** String *pwd*;  **static** {  **try** {  Properties prop = **new** Properties();  prop.load(**new** FileReader("JDBC.conf"));  //准备四大参数  *drivername* = prop.getProperty("drivername");  *url* = prop.getProperty("url");  *user* = prop.getProperty("user");  *pwd* = prop.getProperty("pwd");  //加载驱动  Class.*forName*(*drivername*);  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (ClassNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }    //创建连接  **public** **static** Connection getConnection() {  Connection conn = **null**;  **try** {  conn = DriverManager.*getConnection*(*url*, *user*, *pwd*);  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** conn;  }    //关闭连接  **public** **static** **void** closeConn(Connection conn) {  **if**(conn != **null**) {  **try** {  conn.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }    //关闭statement  **public** **static** **void** closeStmt(Statement stmt) {  **if**(stmt != **null**) {  **try** {  stmt.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  //关闭preparedstatement  **public** **static** **void** closePstmt(PreparedStatement pstmt) {  **if**(pstmt != **null**) {  **try** {  pstmt.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  //关闭结果集  **public** **static** **void** closeRset(ResultSet rset) {  **if**(rset != **null**) {  **try** {  rset.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  } |

## 学习dbutils后对工具类的改进

|  |
| --- |
| **public** **class** JdbcUtils {  **private** **static** DataSource *dataSource* = **new** ComboPooledDataSource();  **public** **static** DataSource getDataSource() {  **return** *dataSource*;  }    **public** **static** Connection getConnection() {  **try** {  **return** *dataSource*.getConnection();  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  } |

## 在Dao层处理事务

在Dao层处理事务非常简单

|  |
| --- |
| public void xxx() {  Connection con = null;  try {  con = JdbcUtils.getConnection();//获取连接  con.setAutoCommitted(false);//开启事务  QueryRunner qr = new QueryRunner();  String sql = …;  Object[] params = …;  qr.update(con, sql, params);//执行操作1，比如减去100元  sql = …;  Object[] params = …;  qr.update(con, sql, params); //执行操作2，比如加上100元，两个操作使用的同一个连接  con.commit();  } catch(Exception e) {  try {  if(con != null) {con.rollback();}//出现异常，失败回滚  } catch(Exception e) {}  } finally {  try {  con.close();//关闭连接  } catch(Exception e) {}  }  } |

## Service才是处理事务的地方

我们要清楚一件事，DAO中不是处理事务的地方，因为DAO中的每个方法都是对数据库的一次操作，而Service中的方法才是对应一个业务逻辑。也就是说我们需要在Service中的一方法中调用DAO的多个方法，而这些方法应该在一起事务中。

DAO层中的方法只进行最细粒度的增删改查；

Service层处理业务，对DAO层的方法进行组合。

怎么才能让DAO的多个方法使用相同的Connection呢？方法不能再自己来获得Connection，而是由外界传递进去。

|  |
| --- |
| public void daoMethod1(Connection con, …) {//操作1 减去xx  }  public void daoMethod2(Connection con, …) {//操作2 加上xx  } |

在Service中调用DAO的多个方法时，传递相同的Connection就可以了。

|  |
| --- |
| public class XXXService() {  private XXXDao dao = new XXXDao();  public void serviceMethod() {//转账  Connection con = null;//和jdbc相关的  try {  con = JdbcUtils.getConnection();//获取连接  con.setAutoCommitted(false);  dao.daoMethod1(con, …);//操作1  dao.doaMethod2(con, …);//操作2，两个操作使用同一个连接  com.commint();  } catch(Exception e) {  try {  con.rollback();  } catch(Exception e) {}  } finally {  try {  con.close();  } catch(Exception e) {}  }  }  } |

但是，在Service中不应该出现Connection，它应该只在DAO中出现，因为它是JDBC的东西，JDBC的东西是用来连接数据库的，连接数据库是DAO的事情，但是，事务是Service的事儿，不能放到DAO中。

## 修改JdbcUtils

我们把对事务的开启和关闭放到JdbcUtils中，在Service中调用JdbcUtils的方法来完成事务的处理，但在Service中就不会再出现Connection这一“禁忌”了。

DAO中的方法不用再让Service来传递Connection了。DAO会主动从JdbcUtils中获取Connection对象，这样，JdbcUtils成为了DAO和Service的中介！

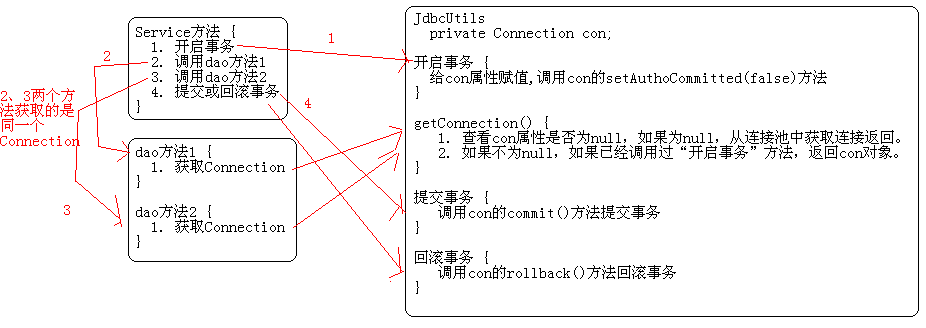
我们在JdbcUtils中添加beginTransaction()和rollbackTransaction()，以及commitTransaction()方法。这样在Service中的代码如下：

|  |
| --- |
| public class XXXService() {  private XXXDao dao = new XXXDao();  public void serviceMethod() {  try {  JdbcUtils.beginTransaction();//开启事务  dao.daoMethod1(…);//操作1  dao.daoMethod2(…); //操作2  JdbcUtils.commitTransaction();//提交事务  } catch(Exception e) {  JdbcUtils.rollbackTransaction();//回滚  }  }  } |

DAO

|  |
| --- |
| 要保证两个方法中获取的conn是同一个  public void daoMethod1(…) {  Connection con = JdbcUtils.getConnection();  }  public void daoMethod2(…) {  Connection con = JdbcUtils.getConnection();  } |

在Service中调用了JdbcUtils.beginTransaction()方法时，JdbcUtils要做准备好一个已经调用了setAuthCommitted(false)方法的Connection对象，因为在Service中调用JdbcUtils.beginTransaction()之后，马上就会调用DAO的方法，而在DAO方法中会调用JdbcUtils.getConnection()方法。这说明JdbcUtils要在getConnection()方法中返回刚刚准备好的，已经设置了手动提交的Connection对象。



在JdbcUtils中创建一个Connection con属性，当它为null时，说明没有事务！当它不为null时，表示开启了事务。

* 在没有开启事务时，可以调用“开启事务”方法；
* 在开启事务后，可以调用“提交事务”和“回滚事务”方法；
* getConnection()方法会在con不为null时返回con，再con为null时，从连接池中返回连接。

beginTransaction()

判断con是否为null，如果不为null，就抛出异常！

如果con为null，那么从连接池中获取一个Connection对象，赋值给con！然后设置它为“手动提交”。

getConnection()

判断con是否为null，如果为null说明没有事务，那么从连接池获取一个连接返回；

如果不为null，说明已经开始了事务，那么返回con属性返回。这说明在con不为null时，无论调用多少次getConnection()方法，返回的都是同个Connection对象。

commitTransaction()

判断con是否为null，如果为null，说明没有开启事务就提交事务，那么抛出异常；

如果con不为null，那么调用con的commit()方法来提交事务；

调用con.close()方法关闭连接；

con = null，这表示事务已经结束！

rollbackTransaction()

判断con是否为null，如果为null，说明没有开启事务就回滚事务，那么抛出异常；

如果con不为null，那么调用con的rollback()方法来回滚事务；

调用con.close()方法关闭连接；

con = null，这表示事务已经结束！

JdbcUtils.java

|  |
| --- |
| **public** **class** JdbcUtils {  **private** **static** DataSource *dataSource* = **new** ComboPooledDataSource();  **private** **static** Connection *con* = **null**;  **public** **static** DataSource getDataSource() {  **return** *dataSource*;  }  **public** **static** Connection getConnection() **throws** SQLException {  **if**(*con* == **null**) {  **return** *dataSource*.getConnection();  }  **return** *con*;  }    **public** **static** **void** beginTranscation() **throws** SQLException {  **if**(*con* != **null**) {  **throw** **new** SQLException("事务已经开启，在没有结束当前事务时，不能再开启事务！");  }  *con* = *dataSource*.getConnection();  *con*.setAutoCommit(**false**);  }    **public** **static** **void** commitTransaction() **throws** SQLException {  **if**(*con* == **null**) {  **throw** **new** SQLException("当前没有事务，所以不能提交事务！");  }  *con*.commit();  *con*.close();  *con* = **null**;  }    **public** **static** **void** rollbackTransaction() **throws** SQLException {  **if**(*con* == **null**) {  **throw** **new** SQLException("当前没有事务，所以不能回滚事务！");  }  *con*.rollback();  *con*.close();  *con* = **null**;  }  } |

## 再次修改JdbcUtils

现在JdbcUtils有个问题，如果有两个线程！第一个线程调用了beginTransaction()方法，另一个线程再调用beginTransaction()方法时，因为con已经不再为null，所以就会抛出异常了。

我们希望JdbcUtils可以多线程环境下被使用！这说明最好的方法是为每个线程提供一个Connection，这样每个线程都可以开启自己的事务了。

|  |
| --- |
| **public** **class** JdbcUtils {  **private** **static** DataSource *dataSource* = **new** ComboPooledDataSource();  **private** **static** ThreadLocal<Connection> *tl* = **new** ThreadLocal<Connection>();  **public** **static** DataSource getDataSource() {  **return** *dataSource*;  }  **public** **static** Connection getConnection() **throws** SQLException {  Connection con = *tl*.get();  **if**(con == **null**) {  **return** *dataSource*.getConnection();  }  **return** con;  }    **public** **static** **void** beginTranscation() **throws** SQLException {  Connection con = *tl*.get();  **if**(con != **null**) {  **throw** **new** SQLException("事务已经开启，在没有结束当前事务时，不能再开启事务！");  }  con = *dataSource*.getConnection();  con.setAutoCommit(**false**);  *tl*.set(con);  }    **public** **static** **void** commitTransaction() **throws** SQLException {  Connection con = *tl*.get();  **if**(con == **null**) {  **throw** **new** SQLException("当前没有事务，所以不能提交事务！");  }  con.commit();  con.close();  *tl*.remove();  }    **public** **static** **void** rollbackTransaction() **throws** SQLException {  Connection con = *tl*.get();  **if**(con == **null**) {  **throw** **new** SQLException("当前没有事务，所以不能回滚事务！");  }  con.rollback();  con.close();  *tl*.remove();  }  } |