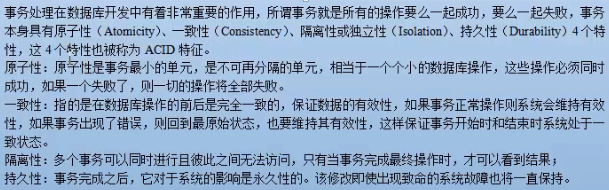
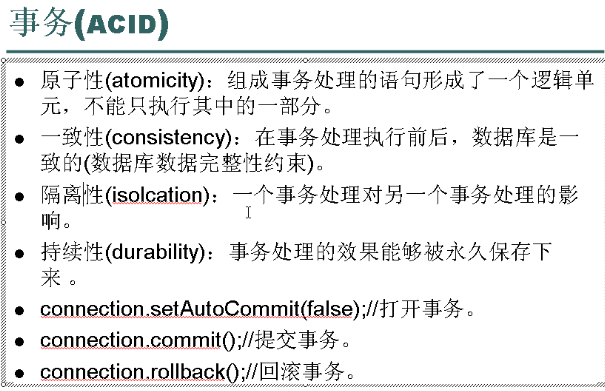
JDBC事务处理

1. **事务的ACID 这4个特征**：



1. 事务：ACID**原子性、一致性、隔离性、持续性。**



注意：**conn.setAutoCommit(false);//方法就是利用关闭自动提交功能，其实也就是打开事务。**

**con.rollback();就是返回到conn.setAutoCommit(false);处。**

1. **打开事务：con.setAutoCommit(false);**

**设置保存点：con.savepoint();**

**回滚事务：con.rollback();//con.rollback(sp);**

**提交事务：con.commit();**

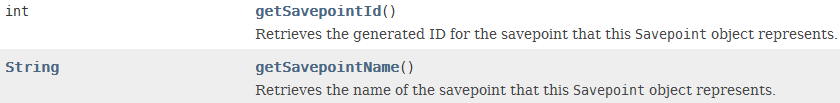
**注意：最后一定要对事务进行提交，否则，得不到相应的操作结果。**

1. **MySQL对事务的支持**：



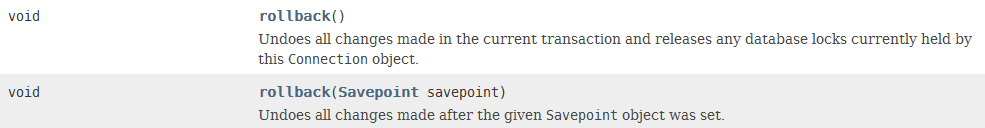
1. **Savepoint保存点**是一个接点，存在于java.sql包中：

其只有两个方法：**getSavepointId和getSavepointName**

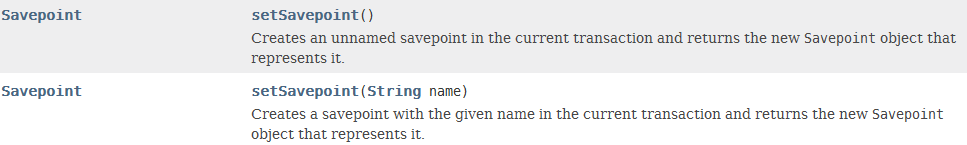


保存点与事务处理的滚回即**Connection的rollback方法**一起使用，滚回到此保存点时 ，此保存点之后执行的语句将会返回原来的状态。

**Connection接口的rollback方法**：



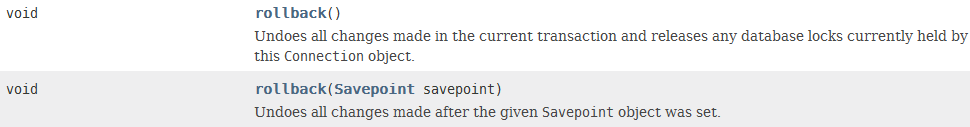
1. 通过**Connection接口的setSavepoint方法**设置一个保存点，返回一个**Savepoint接口的实现对象**：



1. **Connection接口的setAutoCommit（Boolean）方法**：



1. Connection接口的rollback方法：



1. **JDBC事务处理**：
2. **未使用事务处理**：

public class Demo\_Transaction {

static private DbUtil du = new DbUtil();

public static void outAccount(Connection con, String accountName,

int account) throws Exception {

String sql = "update t\_account set accountBalance = accountBalance - ? where accountName = ?";

PreparedStatement psm = con.prepareStatement(sql);

psm.setInt(1, account);

psm.setString(2, accountName);

psm.executeUpdate();

}

public static void inAccount(Connection con, String accountName,

int account) throws Exception {

String sql = "update t\_account set accountBalance = accountBalance + ? where accountName = ?";

PreparedStatement psm = con.prepareStatement(sql);

psm.setInt(1, account);

psm.setString(2, accountName);

psm.executeUpdate();

}

public static void main(String[] args) throws Exception {

Connection con = du.getConnection();

System.out.println("开始转账");

outAccount(con,"张三",500);

inAccount(con,"李四",500);

System.out.println("转账成功");

}

1. **使用事务处理**：

public class Demo\_Transaction {

static private DbUtil du = new DbUtil();

public static void outAccount(Connection con, String accountName,

int account) throws Exception {

String sql = "update t\_account set accountBalance = accountBalance - ? where accountName = ?";

PreparedStatement psm = con.prepareStatement(sql);

psm.setInt(1, account);

psm.setString(2, accountName);

psm.executeUpdate();

}

public static void inAccount(Connection con, String accountName,

int account) throws Exception {

String sql = "update t\_account set accountBaance = accountBalance + ? where accountName = ?";

PreparedStatement psm = con.prepareStatement(sql);

psm.setInt(1, account);

psm.setString(2, accountName);

psm.executeUpdate();

}

public static void main(String[] args) {

Connection con = null;

try {

con = du.getConnection();

**con.setAutoCommit(false);//取消自动提交，打开事务**

System.out.println("开始转账");

outAccount(con,"张三",500);

System.out.println("李三的钱已经扣除");

inAccount(con,"李四",500);

System.out.println("转账成功");

}catch(Exception e) {

try {

**con.rollback();//回滚**

} catch (SQLException e1) {

e1.printStackTrace();

}

}finally {

try {

con.commit();

} catch (SQLException e) {

e.printStackTrace();

}

if(con != null) {

try {

con.close();

} catch (SQLException e) {

e.printStackTrace();

}}}}}

1. **通过设置保存点来实现滚回**：

public static void main(String[] args) {

Connection con = null;

Savepoint sp = null;

try {

con = du.getConnection();

**con.setAutoCommit(false);**

**sp = con.setSavepoint();**

System.out.println("开始转账");

outAccount(con,"张三",500);

System.out.println("李三的钱已经扣除");

inAccount(con,"李四",500);

System.out.println("转账成功");

}catch(Exception e) {

try {

**con.rollback(sp);**

} catch (SQLException e1) {

e1.printStackTrace();

}

}finally {

try {

con.commit();

} catch (SQLException e) {

e.printStackTrace();

}

if(con != null) {

try {

con.close();

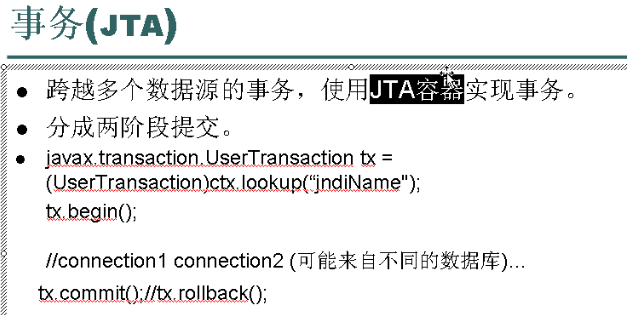
} catch (SQLException e) {

e.printStackTrace();

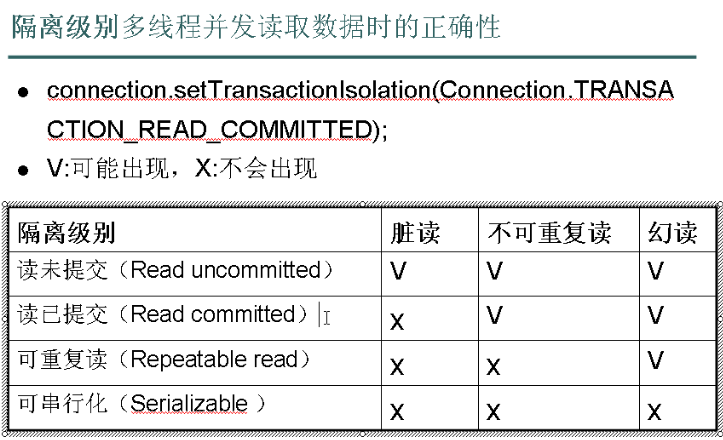
}}} }

1. 事务JTA：Java Transaction API

如果一个事务需要跨越多个数据库，显然上述方法就不可了。例如工资卡为工行，信用卡为建行，则操作必须借助于JTA容器才可以完成。



1. **事务隔离的级别：**



**脏读：读到了其他操作还未提交的数据，而其他操作有可能会执行回滚操作。**

**不可重复读：不可重复读意味着，在数据库访问中，一个事务范围内两个相同的查询却返回了不同数据。这是由于查询时系统中其他事务修改的提交而引起的。**

**幻读：幻读,是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.**

**隔离级别最高的是可串行化，对数据操作最正确，但是需要付出一定的代价。**

1. **如何设置事务的隔离级别？**

**利用Connection的setTransactionIsolation（级别）方法设置。**

**级别就是Connection的Fields：如Connection.TRANSACTION\_NONE等。**



**一般不需要修改数据库连接的隔离级别，使用数据库的缺省级别即可。有些，即使修改了级别，也不一定会起作用。**

1. **不可重复读与幻读的区别：**

**不可重复读：重点强调的是对数据的修改，使一个事务中，多次读到的数据不一致；**

**如第一次读到小明wage=10000，另外一个操作对此修改，使wage=20000，则再次读取发现wage=20000个，不可重复读。在一个事务中前后两次读取的结果并不致，导致了不可重复读。**

**幻读：强调的是对数据的增添或删除，同一个事务中，多次独到的数据的数量不一致，如同产生了幻觉一样。**

**如第一次读到wage>10000的为10个，另外一个操作对此添加一个wage>10000的，则再次读取发现是11个，如同产生了幻觉。**

1) "不可重复读" 是指在一个事务内，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。例如，一个编辑人员两次读取同一文档，但在两次读取之间，作者重写了该文档。当编辑人员第二次读取文档时，文档已更改。原始读取不可重复。如果只有在作者全部完成编写后编辑人员才可以读取文档，则可以避免该问题。  
  
 要避免这种情况，通常可以用 set tran isolation level repeatable read 来设置隔离级别，这样事务A 在两次读取表T中的数据时，事务B如果企图更改表T中的数据（细节到事务A读取数据）时，就会被阻塞，知道事务A提交！ 这样就保证了，事务A两次读取的数据的一致性。  
2）幻觉读是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。例如，一个编辑人员更改作者提交的文档，但当生产部门将其更改内容合并到该文档的主复本时，发现作者已将未编辑的新材料添加到该文档中。如果在编辑人员和生产部门完成对原始文档的处理之前，任何人都不能将新材料添加到文档中，则可以避免该问题。  
 还是上面的例子，事务A要两次读取表T的中数据，虽然设置 repeatable read  可以防止事务B对数据进行修改，但是事务B却可以向表T中插入新的数据。如何防止这个问题，我们可以考虑设置最高的事务隔离级别 set tran isolation level serializable。于是乎，事务B就只能乖乖的等待事务A的提交，才能想表T中插入新的数据，从而避免了幻读！