● 事务指的是逻辑上的一组操作(多条sql语句),组成这组操作的各个单元要么全都成功,要么全都失败.

事务作用：保证在一个事务中多次操作要么全都成功,要么全都失败

MySql的事务

手动控制事务

* start transaction;| begin; -- 开启事务
* commit; --提交事务
* rollback; -- 回滚事务

```
mysql> use test001;
Database changed
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE affair SET money=money-999 WHERE aname='luo';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE affair SET money=money+999 WHERE aname='rong';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

```
begin;   //开启事务
UPDATE affair SET money=money-999 WHERE aname='luo';   //执行更新
UPDATE affair SET money=money+999 WHERE aname='rong';
commit;        //提交事务
```

JDBC的事务

conn.setAutoCommit(false); //开启事务

执行SQL语句 数据更新操作.....

```
UPDATE affair SET money=money-999 WHERE aname='luo';   //执行更新
UPDATE affair SET money=money+999 WHERE aname='rong';
```

conn.commit(); //提交事务

conn.rollback(); //回滚事务

连接池配合Dutils的事务

```
Connection con = null;
        try {
          //创建无参对象  手动进行事务的开启   提交    回滚操作
            QueryRunner runner = new QueryRunner();
            con = JdbcUtils.getConnection2();
            // 开启事务
            con.setAutoCommit(false);
            // 执行更新数据操作
            runner.update(con, "UPDATE affair SET money=money-? WHERE aname=?", f.getMoney(), f.getAnam
            runner.update(con, "UPDATE affair SET money=money+? WHERE aname=?", t.getMoney(), t.getAnam
            // 提交事务
            con.commit();
            return true;
        } catch (SQLException e) {
            try {
                // 回滚事务
                con.rollback();
```

```
            } catch (Exception e1) {
            }
        } finally {
            try {
                con.close();
            } catch (SQLException e) {
            }
        }
```

正常的查询使用有参的对象 传入连接池对象

```
QueryRunner qr = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "select * from affair where aname=?";
        User query = null;
        try {
            query = qr.query(sql, new BeanHandler<User>(User.class), name);
```

多线程操作JDBC的事务处理方法：

引入一个本地线程类来处理**Connection**连接对象

```
ThreadLocal<Connection> tl = new ThreadLocal<>();
```

具体方法：

```
    //创建一个线程局部变量
    private static ThreadLocal<Connection> tl = new ThreadLocal<>();
    //得到连接
    public static Connection getConnectionFromTl() throws SQLException{
        Connection conn = tl.get();
        if(conn==null){
            conn = JdbcUtils.getConnection2();   //使用了C3P0获取Connection连接对象
            tl.set(conn);
        }
        return conn;
    }

    //开启事务    通过连接对象开启事务
    public static void begin() throws SQLException{
        getConnectionFromTl().setAutoCommit(false);
    }

    //提交事务
    public static void commit() throws SQLException{
        getConnectionFromTl().commit();
    }

    //回滚事务
    public static void rollback() throws SQLException{
        getConnectionFromTl().rollback();
    }

    //释放资源
    public static void close() throws SQLException{
        getConnectionFromTl().close();
        tl.remove();
    }
```

使用：

在进行对数据库的操作前调用相应的方法

```
ConnectionManager.begin();//开启事务
//使用多线程技术
runner.update(ConnectionManager.getConnectionFromTl(), "UPDATE affair SET money=money-? WHERE aname=?",
runner.update(ConnectionManager.getConnectionFromTl(), "UPDATE affair SET money=money+? WHERE aname=?",

//使用多线程方式
ConnectionManager.commit();
```

事务的隔离级别（了解）

作用：避免多个线程操作同一个表里的数据时，一个线程可能会读取到另一个线程还没有处理完的数据

如：A线程开启事务查询表里的数据　B线程也开启了事务对表里的数据进行插入或改变 而B线程在还没有完成提交前 A线程对表数据的查询会看到B线程改变的数据，但是有可能B线程没有提交还是中断了事务，当A线程再去查询时会表里数据又恢复了，这样会导致A线程的查询出现一些错误

事务的隔离级别就是用来解决以上问题的

设置事务的隔离级别:　设置隔离级别必须在事务之前

***1 read uncommitted** :未提交读.脏读，不可重复读，虚读都可能发生.

***2 read committed** :已提交读.避免脏读.但是不可重复读和虚读有可能发生.（**oracle**默认）

***4 repeatable read** :可重复读.避免脏读,不可重复读.但是虚读有可能发生.（**mysql**默认）

***8 serializable** :串行化的.避免脏读，不可重复读，虚读的发生.

级别越高、越安全、效率越低。

mysql中：

查看当前的事务隔离级别：SELECT @@TX_ISOLATION;

更改当前的事务隔离级别：SET　TRANSACTION　ISOLATION　LEVEL 四个级别之一。