JDBC是通过JAVA连接数据库的编程

6步骤

1.加载驱动

JAVA有个驱动大管家

实例化时自动向drivermanager注册

//加载驱动类

Class.*forName*("com.mysql.jdbc.Driver");

Class.forName("oracle.jdbc.driver.OracleDriver");

new oracle.jdbc.driver.OracleDriver();

2.链接数据库

Connection conn = DriverManager.getConnection(数据库的链接字符串"jdbc:oracle:thin:@192.168.0.1: 1521:服务名SXT"，用户名"scott"，密码"tiger");

Connection conn = DriverManager.get.Connection("jdbc:mysql://localhost:3306/tt","root","root");

//创建链接（内部包含了Socket对象，远程连接，非常耗时）

//真正开发为了提高效率，会通过连接池来做

Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/testjdbc","root","123456");

3.创建SQL

//创建SQL语句

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select \* from dept");

4.开始循环取结果 5.实现数据结果

//开始循环取结果，实现数据结果

//移动一下游标

**while**(rs.next()){

//rs.getInt(index).index指的是表结构里面的列位置，从左往右数，从1开始 System.***out***.println(rs.getInt(1)+"---"+rs.getString(3)+"---"+rs.getString(2));

}

6.关闭链接

**finally**{

//关闭链接

//必须要rs，stmt，conn分别单独关闭。不能放在一起

**if**(rs!=**null**){

**try** {

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(stmt!=**null**){

**try** {

stmt.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(conn!=**null**){

**try** {

conn.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

} //后打开的先关

完美的JDBC



JDBC进阶

灵活指定SQL语句中的变量

常用Statement方法

execute()运行语句返回boolean

executeQuery()运行select语句，返回ResultSet结果集

executeUpdate()运行insert/update/delete操作，返回更新的行数int

关于SQL注入问题

//SQL注入，通过字符串拼接，恶意造成全部记录删除

String id = "4 or 1=1";//1=1，所以全部满足条件为真，然后直接全部清空

String sql = "delete from T\_user where id = " + id;

stmt.execute(sql);

PreparedStatement（常用）

对存储过程进行调用

提高效率，防止SQL注入

//创建SQL。PreparedStatement通过占位符，预编译来防止SQL恶意注入

String sql = "delete from T\_user where id = ?";//?占位符

PreparedStatement ps = conn.prepareStatement(sql);

//这个时候只会删除id=6的数据，通过占位符来防止SQL恶意注入

ps.setString(1, "6 or 1=1");//索引和？号位置匹配，从1开始。

// ps.setObject(1, "7");//当不确定参数的时候，统一用object来处理

ps.execute();

CallableStatement

批处理Batch

//不推荐使用PreparedStatement，因为预编译空间有限，大批量处理会报错

//大批量处理还是使用Statement

stmt = conn.createStatement();

//因为是Statement所以？占位符不能使用

**for** (**int** i = 0; i < 100; i++) {

String sql = "insert into t\_user (username,classroom) values('yang"+i+"','初1一班')";

//把相关SQL语句添加到Batch里面

stmt.addBatch(sql);

//防止大量Batch造成内存溢出，如果批处理量大，设定每100条处理一次

**if**(i%100==0){

//执行Batch处理

stmt.executeBatch();

//防止大量Batch造成内存溢出，清空Batch

stmt.clearBatch();

}

}

运用事务处理Transaction

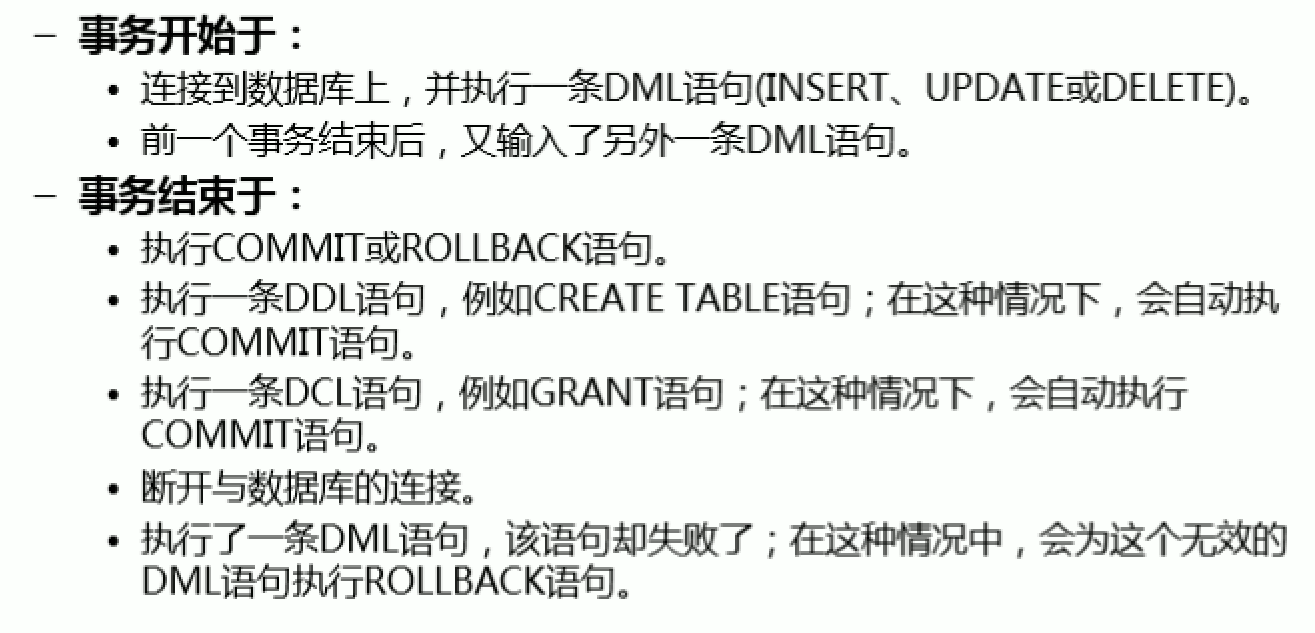
事物是一组同时成功或者同时失败的SQL语句组。整体是数据库操作的执行单元

因为增加外键会影响数据库执行效率，所以手动维护数据库，不使用外键。

就通过事务处理，自己来手动维护数据库。当一条SQL语句更新删除时，手动保证其他相关地方也做处理

事务开始：set autocommit=off; 或者 start transaction;

结束于：执行成功就commit;执行失败就rollback;



事务四大特点ACID：原子性，一致性，隔离性，持久性

Atomicity

原子性保证一个事务为一个最小的单元，内部不可分割；

Consistency

一致性保证事务中的每个操作线程不可单独提交，成功则一起提交，不成功则事务回滚；

Isolation

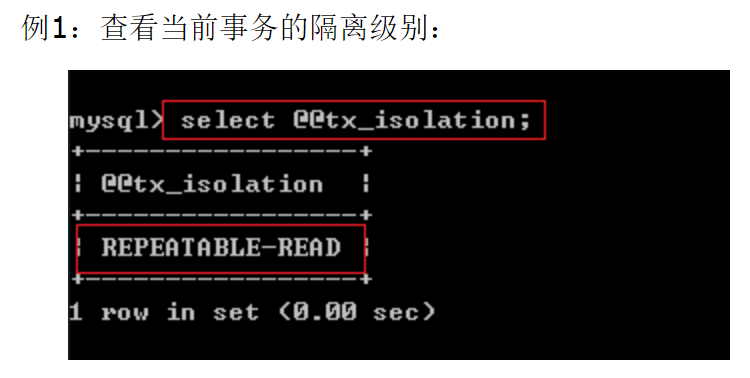
隔离性保证不同事务间看到的数据视图相互独立，相互隔离（隔离级别可设置）；

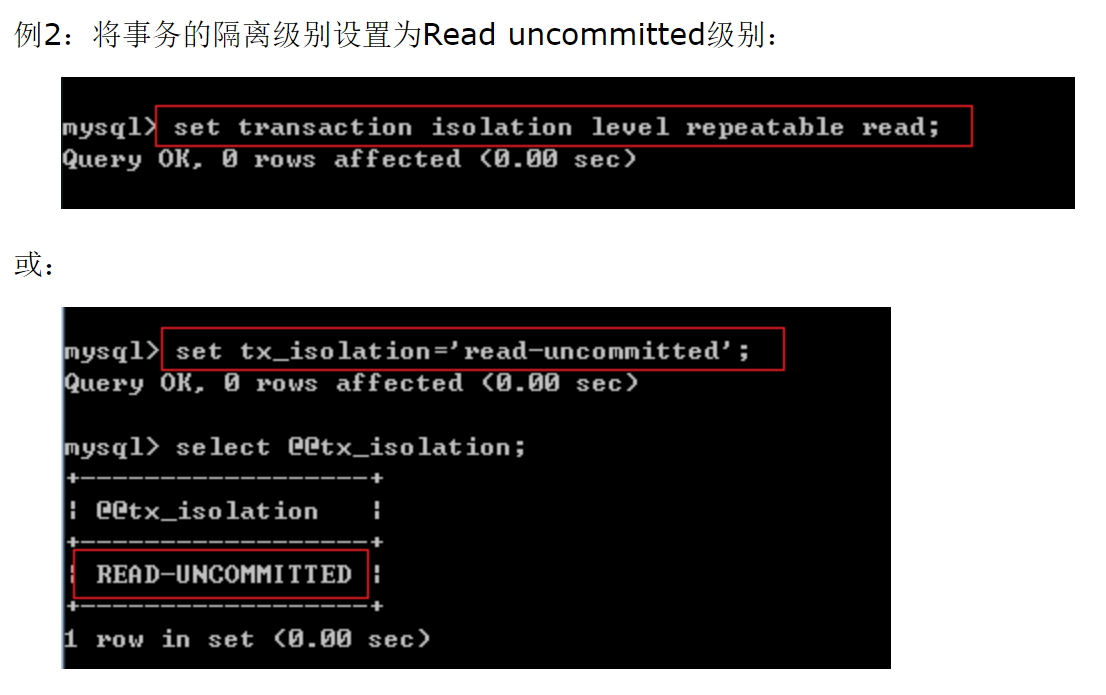
durability

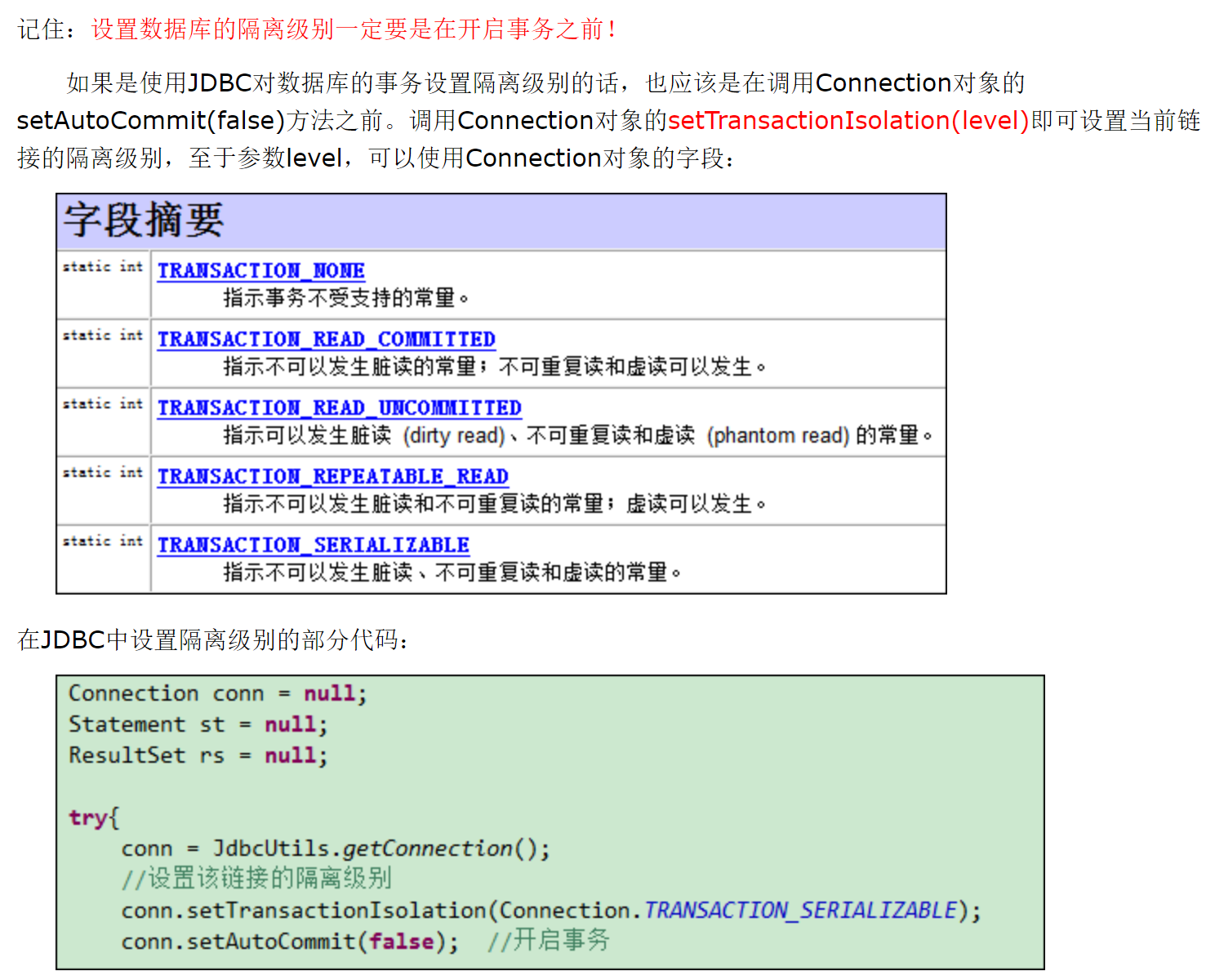
持久性保证事务提交后数据会持久的保存下来；

1.查看当前会话隔离级别  
select @@tx\_isolation;  
2.查看系统当前隔离级别  
select @@global.tx\_isolation;  
3.设置当前会话隔离级别  
set session transaction isolation level repeatable read;  
4.设置系统当前隔离级别  
set global transaction isolation level repeatable read;

隔离级别的设置只对当前链接有效。对于使用MySQL命令窗口而言，一个窗口就相当于一个链接，当前窗口设置的隔离级别只对当前窗口中的事务有效；对于JDBC操作数据库来说，一个Connection对象相当于一个链接，而对于Connection对象设置的隔离级别只对该Connection对象有效，与其他链接Connection对象无关。







关于事务的隔离级别

关键点：描述多个事务和事务并发访问的关系

前提：A和B两个事务同时访问同一个数据库的同一个表

read uncommitted

事务A正在查询，且事务B做的任何操作增查改删，不管事务B有没有提交commit，事务A都可以立即查看到。

read committed

事务A正在查询，且事务B做的任何操作增查改删，只有事务B提交commit后，事务A可以立即查看到。

repeatable read(MySQL默认隔离级别)

事务A正在查询，且事务B做的任何操作增查改删，不管事务B有没有提交commit，事务A都无法查看到。只有事务A自己提交commit后，才能查看B修改的结果。

serializable

事务A正在查询，且事务B也只能做查询操作，如果事务B要做增改删操作，事务A必须先commit。也就是说增改删操作，同一时刻有且只能有一个事务操作。

//把自动提交事物改为false，变成手动提交事务

conn.setAutoCommit(**false**);//事务开始

stmt = conn.createStatement();

**for** (**int** i = 500; i < 510; i++) {

String sql = "insert into idnum (id,num) values('"+i+"','"+i\*2+"')";

stmt.execute(sql);

}

//事务结束

conn.commit();

} **catch** (Exception e) {

**try** {

conn.rollback();//有异常事务就回滚

} **catch** (SQLException e1) {

e1.printStackTrace();

}

e.printStackTrace();

}

conn.setAutoCommit(false);//把自动提交取消

conn.commit();//自己提交

conn.setAutoCommit(true);//设置回默认

可滚动的结果集

CLOB用于存储大量文本数据 存字符

Tinytext 2^8-1=255

Text 2^16-1=65535

Mediumtext 2^24-1=16777215

Longtext 2^32-1=4GB

BLOB用于存储大量的二进制数据 存字节

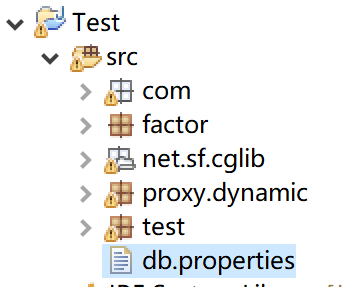
Tinyblob 2^8-1=255

blob 2^16-1=65535

Mediumblob 2^24-1=16777215

Longblob 2^32-1=4GB

封装JDBC



db.properties下的内容：

mysqlDriver = com.mysql.jdbc.Driver//key和value直接用=号隔开。结尾用回车换行区分

mysqlURL = jdbc:mysql://localhost:3306/testjdbc

mysqlUser = root

mysqlPwd = 123456

**public** **class** JDBCUtil {

//配置文件一定要放在src目录下！！

**static** Properties *prop* = **null**;

**static**{

*prop* = **new** Properties();

**try** {

//加载配置文件输入流

*prop*.load(Thread.*currentThread*().getContextClassLoader().getResourceAsStream("db.properties"));

} **catch** (IOException e) {

e.printStackTrace();

}

}

//封装链接数据库

**public** **static** Connection getConnection(){

**try** {

//直接调用配置文件prop.getProperty(String str)来获取相关值

Class.*forName*(*prop*.getProperty("mysqlDriver"));

**return** DriverManager.*getConnection*(*prop*.getProperty("mysqlURL"),

*prop*.getProperty("mysqlUser"),*prop*.getProperty("mysqlPwd"));

} **catch** (Exception e) {

e.printStackTrace();

**return** **null**;

}

}

//封装关闭数据库

**public** **static** **void** close(ResultSet rs, Statement stmt,Connection conn){

**if**(rs!=**null**){

**try** {

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(stmt!=**null**){

**try** {

stmt.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**if**(conn!=**null**){

**try** {

conn.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

}