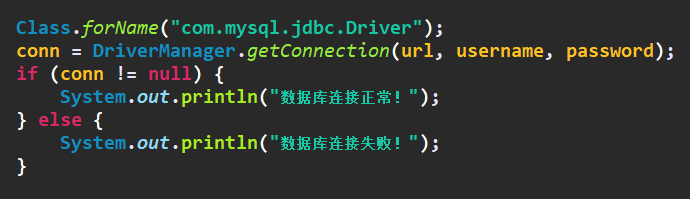
建立连接：

加载数据库驱动：Class.forName

获取数据库连接(会话)：DriverManager.getConnection



创建对象语句：执行SQL语句(Statement/ PreparedStatement/ CallableStatement)

Statement stmt = conn.createStatement();

DatabaseMetaData(数据库元数据)

DatabaseMetaData data = conn.getMetaData()：获取数据库的基本信息，名字，版本号等

Stmt.executeUpdate(sql)：更新/删除/添加操作，返回的是int

stmt.execute(sql)：执行SQL语句，返回的是Boolean，判断执行是否成功，一般都很少用

stmt.executeQuery(sql)：返回值是ResultSet，select操作rs.next/rs.previous

ResultSet：

ResultSetMetaDatar smd=rs.getMetaData():获取结果集的元数据，通元数据可以获取列名相关操作

rsmd.getColumnCount()：获取列的总数

rsmd.getColumnName(i)：获取指定列列名

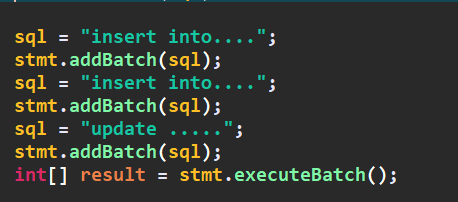
rsmd.getColumnLable(i)：和getColumnName差不多

Object obj=rs.getObject(i)：获取对应的列值

对象语句如果要对ResultSet集合进行游标操作，在创建的时候需要指定XxxStatement的类型，对游标的操作都可以对ResultSet操作

stmt =conn.createStatement(ResultSet.*TYPE\_SCROLL\_SENSITIVE*,ResultSet.*CONCUR\_UPDATABLE*);

executeBatch()：批量操作



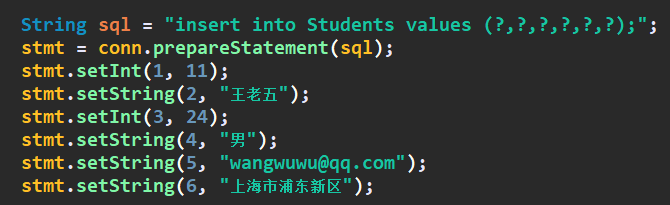
|  |
| --- |
| /\*  **设置不提交事务：只有commit之后才提交，默认是执行execute操作后就执行事务，提交到数据库中**  \*/  conn.setAutoCommit(false);  String sql = "insert...";  stmt.addBatch(sql);  /\*  **虽然这里执行了，但是操作不成功，还未commit**  \*/  stmt.executeBatch();  /\*  **设置保存点**  \*/  sp = conn.setSavepoint();  sql = "insert...";  stmt.addBatch(sql);  sql = "insert... 错误SQL";  stmt.addBatch(sql);  sql = "insert...";  stmt.addBatch(sql);  sql = "update...";  stmt.addBatch(sql);  /\*  **执行执行SQL出错，抛出异常，在异常中处理**  **rollback:回滚，sp表示回滚到保存点**  **保存点之前的操作，只要事务commit之后，都会提交到数据库中**  **保存点之后的操作，即时操作没错，commit之后，任然不会提交到数据库，会回滚**  \*/  conn.rollback(sp);  conn.commit(); |

执行完ps.executeBatch后，应该清除ps.clearBatch

Stmt.close()：关闭操作语句

Conn.close()：关闭数据库链接

PreparedStatement(对象语句块)：预编译，带参数



预编译：会对SQL语句进行预编译，下次执行相同SQL的时候，就直接到缓冲区获取，不重新编译(每个SQL对应一个PrepareStatement，所以获取的PrepareStatement),但是第一次执行效率慢，并且使用？传递参数，避免SQL注入，预编译的时候已经将sql发送给数据库了，就等参数，所以预编译是在数据库中进行的，这也说明了为啥预编译安全，SQL和数据分离传输

SQL注入：

Statement："select \* from user\_table where username=' "+userName+" ' and password=' "+password+" '";

输入用户名：‘or 1=1 # 密码：随意；拼接后的SQL

SELECT \* FROM user\_table WHERE username='’ or 1 = 1 # and password='’；#为注释，最终为

SELECT \* FROM user\_table WHERE username='’ or 1 = 1；这样就会查全表，跳过校验

PrepareStatement："select \* from user\_table where username= ? and password= ?";PrepareStatement会对SQL进行编译，sql注入只对sql语句的准备(编译)过程有破坏作用而PreparedStatement已经准备好了，执行阶段只是把输入串作为数据处理，而不再对sql语句进行解析

Statement：最基本的语句对象，只能执行SQL语句，不能动态添加参数

CallableStatement：可以执行存储过程

数据库连接池：

并发问题：使用Synchronized关键字既可以确保是同步的

public synchronized Connection getConnection()

多数据库/多用户：

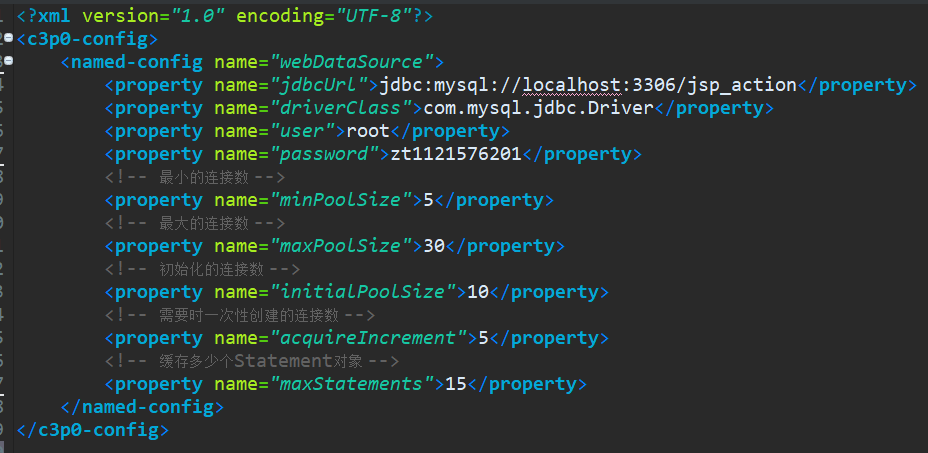
大型企业级应用常连接多个不同类型的数据库，不同数据库有不同的连接池，可以设计一个符合单例的连接池管理类来管理这些连接池，为每个连接池取一个名字

释放与连接：

先根据配置文件创建连接池的最小连接数，当有连接时，先检查连接池中是否有可用的，然后根据连接建立的时间，将建立时间最长的分配给请求(先检查连接是否可用，再分配给请求)，当连接池中无空闲连接，判断当前池中连接数是否达到最大连接数，如果没达到，创建一定的连接数(配置文件指定)，如果已经达到连接池的最大连接数，则等待一段时间，查看是否有释放连接的，超过等待时间则返回null。

C3P0的使用：

导包后配置C3P0-config.xml配置文件



设置maxStatements(缓存的PrepareStatement)：PrepareStatement具有预编译的功能，编译后的结果会缓存到数据库中，当下次执行相同SQL的时候，直接在缓存中调用，不用编译，只需传入参数，这里就是设置缓存的个数，默认为0

Jdbc获取connection

Class.forName

DriverManager.getConnection

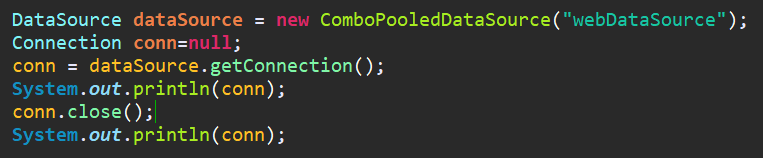
C3P0获取connection

DataSource *dataSource* = new ComboPooledDataSource("webDataSource")：webDataSource---named-config

*dataSource.getConnection*

释放connection：connection.close()，这里的connection不是Connection，而是一个代理的Connection，所以close并不是关闭连接，而是释放连接还给连接池

代理Connection：com.mchange.v2.c3p0.impl.NewProxyConnection@2ef9b8bc



虽然释放后任然可以输出conn但是不能执行操作，这里的conn属于当前类的，而执行操作是连接池执行的，连接池在分配conn的时候内部应该有一个映射表，释放conn后，conn对应的映射就为空，虽然在类中不为空，但是任然不能执行conn，因为在执行之前回去判断当前对象所持有的conn能不能在映射表中找到映射(连接池分配处于一个connection后，会跟踪当前connection，当用户通过connection来操作数据库时，连接池会判断当前用户是否拥有当前connection，释放connection之后，连接池就会断开用户和当前connection的连接)

性能最好的数据库连接池是阿里的 [Druid](https://blog.csdn.net/xiaoguo1001/article/details/51289490)

DBUtils：

一个基于JDBC的简单ORM类库

使用：

导包

获取connection

创建SQL执行工具：QueryRunner runner=new QueryRunner()

QueryRunner runner=new QueryRunner(conn)

runner.update(sql,params)

CRUD：

int i=runner.update(conn,sql,params)：增，删，改

sql：UPDATE bs\_book SET sales=? , stock=? WHERE id=?

params是参数，不是实体类，Object[]

List<Student> list=runner.query(conn,sql,new BeanListHandler<Student>(Student.class),params);

将结果集中的每一行封装到一个对应的JavaBean中，，并存放在list中

Student student=runner.query(conn,sql,new BeanHandler<Student>(Student.class),params);

将结果集中的第一行封装到一个对应的JavaBean中

Object[] obj=runner.query(conn,sql,new ArrayHandler,params);

将结果集中的第一行的数据转化为一个对象数组(即每一列为一个Object)

List<Object[]> objList=runner.query(conn,sql,new ArrayListHandler,params);

将结果集中的每一行的数据转化为一个对象数，并保存到list中

List<Object> columnRs= runner.query(conn,sql,new ColumnListHandler(“columnName”) ,params);

将结果集中指定列的每一行封装到List<Object>中

Map<Object, Map<String, Object>> keyedResult = runner.query(sql,new KeyedHandler("userName"));

返回值封装到Map中，第二个Map的String为userName，根据userName获取Object

MapHandler：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值

返回值：Map<String, Object>

MapListHandler：将结果集中的每一行数据都封装到一个Map里，然后再存放到List

返回值：List<Map<String,Object>>

ScalarHandler：将结果集中某一条记录的其中某一列的数据存成Object

这些Handler都是ResultSetHandler接口的实现类，用户对返回结果的处理

批量执行

int[] batch=Runner.batch(conn,sql,params);

params：Object[][] params，一维sql语句执行次数，二维参数

重数据库中获取单一的值(总记录数)

Object obj=runner.query(conn,sql,new ScalarHandler(),params)

关闭：

rs.cloase(),st.cloase(),conn.cloase()

DBUtils.close(rs), DBUtils.close(st), DBUtils.close(conn)