1. maven依赖

   - mysql-connector-java 8.0.16
   - lombok 1.18.8

2. 自定义连接池包装类

```
package top.soliloquize;

import lombok.AllArgsConstructor;
import lombok.Data;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
* @author wb
* @date 2019/7/17
*/
@Data
@AllArgsConstructor
public class PooledConnection {
    /**
    * 表示繁忙标志      复用的标志
    */
    private boolean busy;
    /**
    * 真正的sql 连接connection(java.sql.Connection)
    */
    private Connection connection;
    /**
    * 只是用来测试当前connectionName,便于观察
    */
    private String connName;
/**
* 将该连接置为不可用,而不是真正关掉连接
*/
public void close() {
    this.busy = false;
}

/**
* 对外提供一个简单的测试方法，也就是获得了连接之后，就可以使用statement进行执行Sql语句
*
* @param sql sql语句
* @return ResultSet
*/
public ResultSet queryBySql(String sql) {
    Statement sttm;
    ResultSet rs = null;
    try {
        sttm = connection.createStatement();
        rs = sttm.executeQuery(sql);
        System.out.println("当前连接编号是:" + connName);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rs;
}
}
```

3. 新建连接池接口SoliloquizeThreadPool.java

```
package top.soliloquize;


/**
* @author wb
* @date 2019/7/17
* 连接池架构接口
*/
public interface SoliloquizeThreadPool {
    /**
    * 对外提供可复用连接
    *
    * @return PooledConnection
    */
    PooledConnection getConnection();
```

```
/**
* 对内创建连接
*
* @param count 连接数量
*/
void createConnections(int count);
}
```

4. 新建SoliloquizeThreadPoolImpl.java实现SoliloquizeThreadPool

```
package top.soliloquize;

import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import java.util.Vector;
import java.util.concurrent.CopyOnWriteArrayList;

/**
* @author wb
* @date 2019/7/17
*/
public class SoliloquizeThreadPoolImpl implements SoliloquizeThreadPool {
private static String driver = null;
private static String url = null;
private static String user = null;
private static String password = null;
/**
* 连接池中管道参数
**/
private static int initCount = 5;
private static int stepSize = 10;
private static int poolMaxSize = 55;
private static int expandTime = 0;
/**
* 连接池
* 线程安全集合,用来放(可复用)数据库连接管道
*/
private static volatile CopyOnWriteArrayList&lt;PooledConnection&gt; pooledConnections = new CopyOnWriteArrayList&lt;&gt;();

public SoliloquizeThreadPoolImpl() {
    this.init();
}

@Override
public PooledConnection getConnection() {
    //要拿连接,先判断连接池中是否有连接
    if (pooledConnections.size() == 0) {
        System.out.println("连接池没有连接! ");
        //如果没有就手动再建一把连接池
        this.createConnections(initCount);
    }
    PooledConnection connection = getRealConnection();
    //如果还是没有拿到,说明全部线程都处于busy状态,得扩容
    while (connection == null) {
        this.createConnections(stepSize);
        connection = getRealConnection();
        try {//拿到连接等待一会,防止连接又被别的线程抢夺
            Thread.sleep(50);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    return connection;
}

/**
* 同步方法,真正的获取连接(连接包装内包括:connection和标志位busy)
*
* @return PooledConnection
*/
private  PooledConnection getRealConnection() {
    for (PooledConnection conn : pooledConnections) {
        //判断该连接是否已被占用,false为可用(空闲),true为占用(繁忙)
        if (!conn.isBusy()) {
            Connection connection = conn.getConnection();
            try {
                //判断该连接是否在设定时间连接通数据库(连接通为true)
                if (!connection.isValid(2000)) {
                    connection = DriverManager.getConnection(url, user, password);
                }
            conn.setConnection(connection);
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
            conn.setBusy(true);
            return conn;
        }
    }
    //如果连接池中的连接都被占用,则返回null,由调用函数处理(即扩容)
    return null;
}

@Override
public synchronized void createConnections(int count) {
    expandTime++;
    System.out.println("第" + expandTime + "次扩容,扩容量为:" + count);
    if ((pooledConnections.size() + count) &lt;= poolMaxSize) {
        for (int i = 0; i &lt; count; i++) {
            try {
                //获取连接放入线程安全的连接池中
                Connection conn = DriverManager.getConnection(url, user, password);
                PooledConnection pooledConnection = new PooledConnection(false, conn, String.valueOf(i));
                pooledConnections.add(pooledConnection);
                System.out.println("初始化" + (i + 1) + "个连接");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    System.out.println("当前连接池连接数量:" + pooledConnections.size());
    System.out.println("最大连接池数量为:" + poolMaxSize);
}

private void init() {
    InputStream inStream = this.getClass().getClassLoader().getResourceAsStream("jdbc.properties");
    Properties properties = new Properties();
    try {
        properties.load(inStream);
    } catch (IOException e) {
        //若这里抛出异常则下面不运行
        e.printStackTrace();
    }
    driver = properties.getProperty("jdbc_driver");
    url = properties.getProperty("jdbc_url");
    user = properties.getProperty("jdbc_username");
    password = properties.getProperty("jdbc_password");
    String inCountStr = properties.getProperty("initCount");
    String stepSizeStr = properties.getProperty("stepSize");
    String poolMaxSizeStr = properties.getProperty("poolMaxSize");
    if (inCountStr != null &amp;&amp; Integer.valueOf(inCountStr) &gt; 0) {
        initCount = Integer.valueOf(properties.getProperty("initCount"));
    } else if (stepSizeStr != null &amp;&amp; Integer.valueOf(stepSizeStr) &gt; 0) {
        stepSize = Integer.valueOf(properties.getProperty("stepSize"));
    } else if (poolMaxSizeStr != null &amp;&amp; Integer.valueOf(poolMaxSizeStr) &gt; 0) {
        poolMaxSize = Integer.valueOf(properties.getProperty("poolMaxSize"));
    }
    //准备创建DriverManager
    try {
        Driver dbDriver = (Driver) Class.forName(driver).newInstance();
        DriverManager.registerDriver(dbDriver);
    } catch (Exception e) {
        e.printStackTrace();
    }
    //获取连接,用create方法获取
    this.createConnections(initCount);
}
}
```

5. 新建PoolManager.java

```
package top.soliloquize;

/**
* @author wb
* @date 2019/7/17
* 提供线程池实例
*/
public enum PoolManager {
    /**
    * 连接池实例
    */
    INSTANCE;
public SoliloquizeThreadPool get() {
    return new SoliloquizeThreadPoolImpl();
}
}
```

6. 配置(在resources目录下新建jdbc.properties)

```
jdbc_driver = com.mysql.cj.jdbc.Driver
jdbc_url = jdbc:mysql://192.168.0.120:3306/master?characterEncoding=utf8&useSSL=false
jdbc_username = root
jdbc_password = wangBin_123
```

7. 测试

```
package top.soliloquize;

import java.sql.ResultSet;

/**
* @author wb
* @date 2019/7/17
*/
public class Test {
    public static void main(String[] args) {
        SoliloquizeThreadPool pool = PoolManager.INSTANCE.get();
        for (int i = 0; i < 2000; i++) {
            new Thread(() -> {
                pool.getConnection();
                ResultSet resultSet = pool.getConnection().queryBySql("select * from user");
                try {
                    while (resultSet.next()) {
                        String name = resultSet.getString("name");
                        System.out.println(name);
                    }
                }catch (Exception e) {}
            }).start();
        }
    }
}
```