天津理工大学 实验报告

学院（系）名称：计算机科学与工程学院

姓名	王帆	学号	20152180	专业	银行业务管理软件
班级	2015 级 1 班	实验项目	实验四：温度传感器模拟设计		
课程名称		Java 程序设计		课程代码	0667056
实验时间		2018 年 11 月 15 日第 1、2 节		实验地点	7-219

考核标准	实验过程 25 分	程序运行 20 分	回答问题 15 分	实验报告 30 分	特色功能 5 分	考勤违纪情况 5 分	成绩	
成绩栏							其它批改意见:	
考核内容	评价在实验课堂中的表现，包括实验态度、编写程序过程等内容等。	□功能完善, □功能不全 □有小错 □无法运行	○正确 ○基本正确 ○有提示 ○无法回答	○完整 ○较完整 ○一般 ○内容极少 ○无报告	○有 ○无	○有 ○无	教师签字：	

一、实验目的

使用 Java，实现定时任务以及对数据库数据的控制与访问。

二、实验题目与要求

使用 Java 开发两个软件，要求如下：

1.设计一个模拟温度传感器：实现每十秒钟向数据库发送一个温度值，要求温度值在 20±5℃范围内随机变化

2.设计一个温度显示模块，在 Console 上每 10 秒显示一次实时温度，每 1 分钟显示最近 1 分钟的平均温度。

数据库表名：sample，包含两个字段：

```
sample_time: timestamp;
sample_data: decimal(5,1);
```

三、实验过程与实验结果

设计思路：

1.实现模拟温度传感器，通过 Java 定时任务与 JDBC，定时向数据库发送随机温度值（20±5）

2.实现温度显示模块，通过 Java 定时任务与 JDBC，定时从数据库中获取温度（实时温度与一分钟内温度集合），通过格式化处理与组织展示给用户。

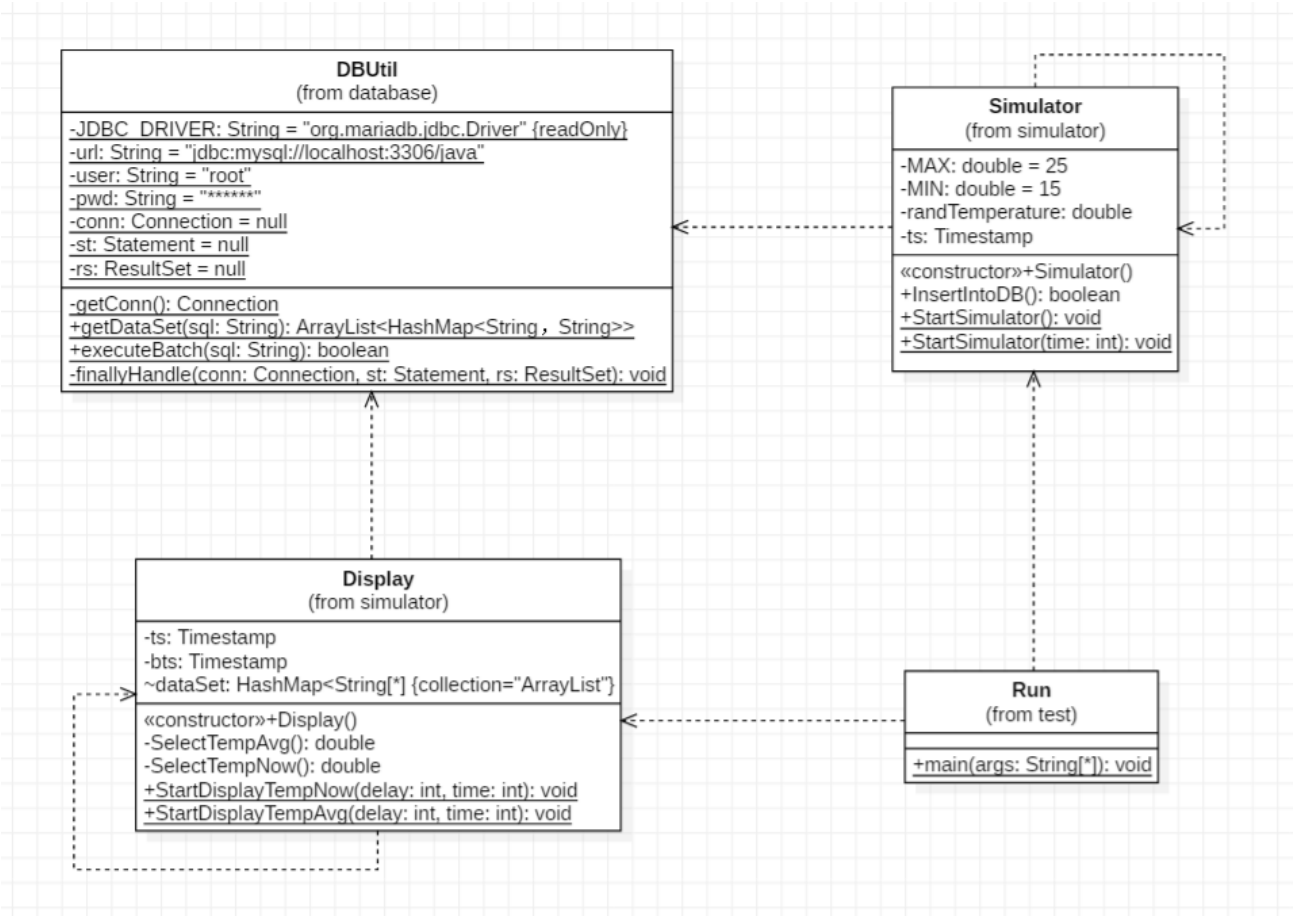

图 1 项目类图

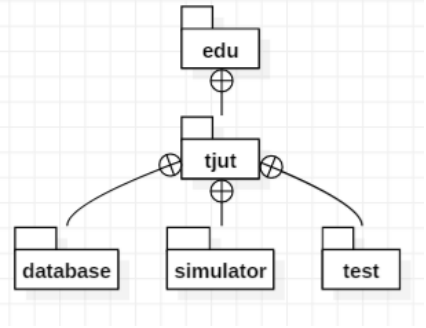

图 2 项目包图

实现过程：

1． 封装数据库操作类 DBUTil，实现 JDBC 对数据库的连接访问、数据查询与数据修改。

```
package edu.tjut.database;

//数据库操作工具类

import java.sql.Connection;
```

```java
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;

public class DBUtil {
    // 1.定义并声明常用字段
    private static final String JDBC_DRIVER = "org.mariadb.jdbc.Driver";
    private static String url = "jdbc:mysql://localhost:3306/java";
    private static String user = "root";
    private static String pwd = "*******";
    // 2.定义并声明SQL操作对象
    private static Connection conn = null;
    private static Statement st = null;
    private static ResultSet rs = null;

    // 3.获取连接
    private static Connection getConn() {
        try {
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(url, user, pwd);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return conn;
    }

    // 4.SQL查询
    public static ArrayList<HashMap<String, String>> getDataSet(String sql) {
        HashMap<String, String> hash = null;
        ArrayList<HashMap<String, String>> list = new ArrayList<>();
        ResultSetMetaData rsma = null;
        int columncount = 0;

        try {
            conn = DBUtil.getConn();
            st = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
            rs = st.executeQuery(sql);
            rsma = rs.getMetaData();
            while (rs.next()) {
                hash = new HashMap<>();
                columncount = rsma.getColumnCount();
```

```
                for (int i = 1; i <= columncount; i++) {
                    hash.put(rsma.getColumnName(i), rs.getString(i));
                }
                list.add(hash);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            finallyHandle(conn, st, rs);
        }
        return list;
    }

    // 5.SQL控制
    public static boolean executeBatch(String sql) {
        boolean flag = true;// 返回值默认为true
        try {
            conn = getConn();// 调用getConn()方法，初始化数据库连接
            conn.setAutoCommit(false);
            st = conn.createStatement();
            st.addBatch(sql);
            st.executeBatch();
            conn.commit();// 执行事务
            conn.setAutoCommit(true);

        } catch (Exception ex) {
            try {
                conn.rollback();// 事务回滚
            } catch (SQLException e) {
                e.printStackTrace();
            }
            flag = false;// 执行失败，返回false
            ex.printStackTrace();
        } finally {
            finallyHandle(conn, st, rs);// 关闭数据库连接
        }
        return flag;
    }

    // 6.最终处理（关闭数据库连接）
    private static void finallyHandle(Connection conn, Statement st, ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
                rs = null;
            }
```

```
            if (st != null) {
                st.close();
                st = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

}
```

2. 实现温度模拟类 Simulator，对特定范围的温度进行随机产生，实现 SQL 插入语句的组织与数据插入。

```
package edu.tjut.simulator;

import java.sql.Timestamp;
import java.util.Date;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

import edu.tjut.database.DBUtil;


public class Simulator {

    private double MAX = 25;
    private double MIN = 15;
    private double randTemperature;
    private Timestamp ts;

    public Simulator() {
        randTemperature = (MIN + new Random().nextDouble() * (MAX - MIN));
        randTemperature = (double) Math.round(randTemperature * 10) / 10;
        ts = new Timestamp(new Date().getTime());
    }

    public boolean InsertIntoDB() {
        try {
            //字符串 组织SQL语句
            String sql = "INSERT INTO sample(sample_time,sample_data)
VALUES('"+ts+"','"+randTemperature+"')";
            //执行插入操作
```

```
            if(DBUtil.executeBatch(sql)){
                return true;
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    public static void StartSimulator() {
        Timer timer = new Timer();
        //前一次执行程序结束后 10s 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                new Simulator().InsertIntoDB();
            }
        }, 0,10000);
    }

    public static void StartSimulator(int time) {
        Timer timer = new Timer();
        //前一次执行程序结束后 (time)ms 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                new Simulator().InsertIntoDB();
            }
        }, 0,time);
    }
}
```

3. 实现演示类 Display，对数据库中的温度数据进行定时获取并格式化输出

```
package edu.tjut.simulator;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;

import edu.tjut.database.DBUtil;

public class Display {
```

```
    private Timestamp ts;
    private Timestamp bts;
    ArrayList<HashMap<String, String>> dataSet;

    public Display() {
        ts = new Timestamp(new Date().getTime());
        bts = new Timestamp(ts.getTime() - 60000);
    }

    private double SelectTempAvg() {
        try {
            //字符串 组织SQL语句
            String sql = "SELECT sample_data FROM sample WHERE sample_time BETWEEN '"+ bts
+"' AND '"+ ts +"'";
            //执行查询操作
            dataSet=DBUtil.getDataSet(sql);
            double temp = 0;
            for(int i=0;i<dataSet.size();i++){
                temp+=Double.parseDouble(dataSet.get(i).get("sample_data"));
            }
            temp/=dataSet.size();
            return temp;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    private double SelectTempNow() {
        try {
            //字符串 组织SQL语句
            String sql = "SELECT sample_data FROM sample ORDER BY sample_time DESC LIMIT 1";
            //执行查询操作
            dataSet=DBUtil.getDataSet(sql);
            return Double.parseDouble(dataSet.get(0).get("sample_data"));
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    public static void StartDisplayTempNow(int delay,int time) {
        Timer timer = new Timer();
```

```
        // 前一次执行程序结束后 60s 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.out.println("==============================");
                System.out.println("当前时间:" + new Timestamp(new Date().getTime()));

                System.out.printf("当前温度：%.1f℃\n",new Display().SelectTempNow());

                System.out.println("==============================");
            }
        }, delay, time);
    }

    public static void StartDisplayTempAvg(int delay,int time) {
        Timer timer = new Timer();

        // 前一次执行程序结束后 (time)ms 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.out.println("==============================");
                System.out.println("当前时间:" + new Timestamp(new Date().getTime()));

                System.out.printf("平均温度：%.1f℃\n",new Display().SelectTempAvg());

                System.out.println("==============================");
            }
        }, delay, time);

    }
}
```

4. 测试类 Run 实现对 Similator 和 Display 的调用

```
package edu.tjut.test;

import edu.tjut.simulator.Display;
import edu.tjut.simulator.Simulator;

public class Run {

    public static void main(String[] args) {
        System.out.println("模拟器开始");
        Simulator.StartSimulator(10000);
        Display.StartDisplayTempNow(0,10000);
        Display.StartDisplayTempAvg(60000,60000);
    }
```

}

示例与演示：

```
Run [Java Application] C:\Java\jdk1.8.0_161\bin\javaw.exe (2018年11月19日 下午5:25:41)
模拟器开始,一分钟后显示第一组数据
==============================
当前时间:2018-11-19 17:25:41.879
当前温度: 21.5℃
==============================
==============================
当前时间:2018-11-19 17:25:51.88
当前温度: 24.2℃
==============================
==============================
当前时间:2018-11-19 17:26:01.88
当前温度: 16.4℃
==============================
==============================
当前时间:2018-11-19 17:26:11.881
当前温度: 15.2℃
==============================
==============================
当前时间:2018-11-19 17:26:21.881
当前温度: 24.9℃
==============================
==============================
当前时间:2018-11-19 17:26:31.882
当前温度: 24.7℃
==============================
==============================
当前时间:2018-11-19 17:26:41.879
==============================
当前时间:2018-11-19 17:26:41.882
平均温度: 21.5℃
==============================
当前温度: 23.5℃
==============================
```

图3 演示结果

四、收获与体会

1. 掌握了 Java 中 JDBC 编程的基本思路与操作；
2. 掌握了 Java 定时调度的基本使用方法。

五、源代码清单

```
public class DBUtil {
    // 1.定义并声明常用字段
    private static final String JDBC_DRIVER = "org.mariadb.jdbc.Driver";
    private static String url = "jdbc:mysql://localhost:3306/java";
    private static String user = "root";
    private static String pwd = "*******";
    // 2.定义并声明SQL操作对象
    private static Connection conn = null;
    private static Statement st = null;
    private static ResultSet rs = null;

    // 3.获取连接
```

```
    private static Connection getConn() {
        try {
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(url, user, pwd);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return conn;
    }

    // 4.SQL查询
    public static ArrayList<HashMap<String, String>> getDataSet(String sql) {
        HashMap<String, String> hash = null;
        ArrayList<HashMap<String, String>> list = new ArrayList<>();
        ResultSetMetaData rsma = null;
        int columncount = 0;

        try {
            conn = DBUtil.getConn();
            st = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
            rs = st.executeQuery(sql);
            rsma = rs.getMetaData();
            while (rs.next()) {
                hash = new HashMap<>();
                columncount = rsma.getColumnCount();
                for (int i = 1; i <= columncount; i++) {
                    hash.put(rsma.getColumnName(i), rs.getString(i));
                }
                list.add(hash);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            finallyHandle(conn, st, rs);
        }
        return list;
    }

    // 5.SQL控制
    public static boolean executeBatch(String sql) {
        boolean flag = true;// 返回值默认为true
        try {
            conn = getConn();// 调用getConn()方法，初始化数据库连接
            conn.setAutoCommit(false);
            st = conn.createStatement();
```

```
            st.addBatch(sql);
            st.executeBatch();
            conn.commit();// 执行事务
            conn.setAutoCommit(true);

        } catch (Exception ex) {
            try {
                conn.rollback();// 事务回滚
            } catch (SQLException e) {
                e.printStackTrace();
            }
            flag = false;// 执行失败，返回false
            ex.printStackTrace();
        } finally {
            finallyHandle(conn, st, rs);// 关闭数据库连接
        }
        return flag;
    }

    // 6.最终处理（关闭数据库连接）
    private static void finallyHandle(Connection conn, Statement st, ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
                rs = null;
            }
            if (st != null) {
                st.close();
                st = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

}
public class Simulator {

    private double MAX = 25;
    private double MIN = 15;
    private double randTemperature;
    private Timestamp ts;
```

```
    public Simulator() {
        randTemperature = (MIN + new Random().nextDouble() * (MAX - MIN));
        randTemperature = (double) Math.round(randTemperature * 10) / 10;
        ts = new Timestamp(new Date().getTime());
    }

    public boolean InsertIntoDB() {
        try {
            //字符串 组织SQL语句
            String sql = "INSERT INTO sample(sample_time,sample_data)
VALUES('"+ts+"','"+randTemperature+"')";
            //执行插入操作
            if(DBUtil.executeBatch(sql)){
                return true;
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    public static void StartSimulator() {
        Timer timer = new Timer();
        //前一次执行程序结束后 10s 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                new Simulator().InsertIntoDB();
            }
        }, 0,10000);
    }

    public static void StartSimulator(int time) {
        Timer timer = new Timer();
        //前一次执行程序结束后 (time)ms 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                new Simulator().InsertIntoDB();
            }
        }, 0,time);
    }
}
public class Display {
```

```
    private Timestamp ts;
    private Timestamp bts;
    ArrayList<HashMap<String, String>> dataSet;

    public Display() {
        ts = new Timestamp(new Date().getTime());
        bts = new Timestamp(ts.getTime() - 60000);
    }

    private double SelectTempAvg() {
        try {
            //字符串 组织SQL语句
            String sql = "SELECT sample_data FROM sample WHERE sample_time BETWEEN '"+ bts
+"' AND '"+ ts +"'";
            //执行查询操作
            dataSet=DBUtil.getDataSet(sql);
            double temp = 0;
            for(int i=0;i<dataSet.size();i++){
                temp+=Double.parseDouble(dataSet.get(i).get("sample_data"));
            }
            temp/=dataSet.size();
            return temp;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    private double SelectTempNow() {
        try {
            //字符串 组织SQL语句
            String sql = "SELECT sample_data FROM sample ORDER BY sample_time DESC LIMIT 1";
            //执行查询操作
            dataSet=DBUtil.getDataSet(sql);
            return Double.parseDouble(dataSet.get(0).get("sample_data"));
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return -1;
    }

    public static void StartDisplayTempNow(int delay,int time) {
        Timer timer = new Timer();
```

```
        // 前一次执行程序结束后 60s 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.out.println("==============================");
                System.out.println("当前时间:" + new Timestamp(new Date().getTime()));

                System.out.printf("当前温度：%.1f℃\n",new Display().SelectTempNow());

                System.out.println("==============================");
            }
        }, delay, time);
    }

    public static void StartDisplayTempAvg(int delay,int time) {
        Timer timer = new Timer();

        // 前一次执行程序结束后 (time)ms 后开始执行下一次程序
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                System.out.println("==============================");
                System.out.println("当前时间:" + new Timestamp(new Date().getTime()));

                System.out.printf("平均温度：%.1f℃\n",new Display().SelectTempAvg());

                System.out.println("==============================");
            }
        }, delay, time);

    }
}
public class Run {

    public static void main(String[] args) {
        System.out.println("模拟器开始");
        Simulator.StartSimulator(10000);
        Display.StartDisplayTempNow(0,10000);
        Display.StartDisplayTempAvg(60000,60000);
    }
}
```