**基于IoTDB和WISE的盆栽养护系统**

**学生姓名：周锐淇、马小龙、周旋 指导老师：王静**

**摘 要** 物联网是一个基于互联网、传统电信网等的信息承载体，它让所有能够被独立寻址的普通物理对象形成互联互通的网络，随着5G等高速、低延时网络技术的应用，物联网也将迸发更大的生机。我们以盆栽养护为选题，旨在实现从数据采集到客户使用这一整套流程，体验传感器的开发利用。

本系统的设计开发主要基于SpringBoot、Android、IoTDB、WISE通信、传感器数据采集、B/S架构、MVC设计模式等技术或者思想，开发工具为Intellij IDEA、Android Studio、Ecplise、后台测试工具为Postman，客户端运行系统为android 9.0以上。整个盆栽养护系统分为三个部分：后端、客户端、数据采集端。数据采集端通过温度传感器与湿度传感器对温湿度数据进行采集，获得数据后通过WISE通信将数据传送至后端，也可以收到后端发送的指令，对异常情况进行处理。android客户端与后端进行交互获取温湿度实时数据、历史时段平均数据以及发送指令等。后端对数据进行处理、存储，将数据持久化，同时，对前端的请求作出响应。

经过多轮测试、改进，系统运行状况良好，响应迅速。基本具有盆栽养护选题所需要实现的所有功能，完成了既定目标，后续如果进一步完善改进可以在实际生活中得到广泛的应用。

**关键词** SpringBoot后台框架、IoTDB数据库、Android、WISE通信、传感器数据采集

目录

[1 引 言 3](#_Toc60582068)

[1.1 本文主要内容 3](#_Toc60582069)

[1.2 项目概况 3](#_Toc60582070)

[1.3设计平台 4](#_Toc60582071)

[2 设计原理 5](#_Toc60582072)

[2.1 后台基础框架：SpringBoot 5](#_Toc60582073)

[2.2 依赖管理工具：Maven 5](#_Toc60582074)

[2.3 数据库：IoTDB 6](#_Toc60582075)

[2.4 后端开发思想MVC 6](#_Toc60582076)

[2.5 android开发 6](#_Toc60582077)

[2.6 数据采集端 7](#_Toc60582078)

[3设计步骤 8](#_Toc60582079)

[3.1系统分析 8](#_Toc60582080)

[3.1.1 需求分析 8](#_Toc60582081)

[3.1.2可行性分析 8](#_Toc60582082)

[3.2 系统设计 9](#_Toc60582083)

[3.3详细设计 10](#_Toc60582084)

[3.3.1数据库设计 10](#_Toc60582085)

[3.3.2功能详细实现 10](#_Toc60582086)

[3.4测试 21](#_Toc60582087)

[3.4.1功能测试 21](#_Toc60582088)

[3.4.1测试结果 24](#_Toc60582089)

# 1 引 言

## 1.1 本文主要内容

本文主要用以汇报基于IoTDB和WISE的盆栽养护系统的设计开发过程以及运行结果。

项目分为三个端：后端、数据采集端、客户端。

后端对数据进行处理、存储，将数据持久化，同时，对前端的请求作出响应。

数据采集端通过温度传感器与湿度传感器对温湿度数据进行采集，获得数据后再通过WISE通信将数据传送至后端提供的接口，也可以收到来自后端的指令，并进行处理。

android客户端与后端进行交互获取温湿度实时数据、实时平均数据、历史时段平均数据以及发送指令等。

本文第二节介绍了基于IoTDB和WISE的盆栽养护系统的设计原理。第三节详细描述了设计开发步骤，包括可行性分析、需求分析、功能模块设计、数据库设计、详细实现方法等。第四节对本项目进行了总结。

## 1.2 项目概况

·项目名称：基于IoTDB和WISE的盆栽养护系统

·人员分工：周锐淇（后端、文档、负责人）

马小龙（客户端）

周 旋（数据采集端）

·目标用户：盆栽养护人员、农场管理人员等 。

·系统介绍：本项目旨在实现基于IoTDB和WISE的盆栽养护系统的各项功能，完成选题。

·目 标：基于选题，实现基于IoTDB和WISE的盆栽养护系统的各项功能，同时需要符合题目的其他各项明确或隐含需求。前后端进行数据交互，完成数据采集端、后端、客户端的设计开发。在功能实现的基础上提升用户使用体验。

## 1.3设计平台

**硬件平台**：系统为windows10的笔记本、android 9.0以上的安卓手机、WISE 1024E、温度传感器、湿度传感器。

**软件平台**：Intellij IDEA、Android Studio、Ecplise、IoTDB数据库、Postman等

# 2 设计原理

## 2.1 后台基础框架：SpringBoot

SpringBoot是一个快速整合的第三方框架，它简化了XML配置完全采用注解化、内置http服务器（Jetty和Tomcat），最终以java应用程序进行执行。SpringBoot基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决[1][3]。SpringBoot也是当前主流的java开发框架。使用SpringBoot框架具有以下优势：

* 可以创建独立的[Spring](https://baike.baidu.com/item/Spring/85061)应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；
* 内嵌Tomcat或Jetty等Servlet容器；
* 提供自动配置的“starter”项目对象模型（POMS）以简化[Maven](https://baike.baidu.com/item/Maven/6094909)配置；
* 尽可能自动配置Spring容器，SpringBoot的自动配置特性利用了Spring4对条件化配置的支持，合理地推测应用所需的bean并自动化配置他们；
* 提供准备好的特性，如指标、健康检查和外部化配置；
* 绝对没有代码生成，不需要XML配置。

## 2.2 依赖管理工具：Maven

Maven是一个项目管理和综合工具。Maven提供了开发人员构建一个完整的生命周期框架。开发团队可以自动完成项目的基础工具建设，Maven使用标准的目录结构和默认构建生命周期。

使用maven能够使我们方便快捷地导入依赖、构建项目。

## 2.3 数据库：IoTDB

IoTDB是针对时间序列数据收集、存储与分析一体化的数据管理引擎。它具有体量轻、性能高、易使用的特点，完美对接Hadoop与Spark生态，适用于工业物联网应用中海量时间序列数据高速写入和复杂分析查询的需求。与我们此前所接触的关系型数据库MySQL不一样，IoTDB属于时序数据库。

时序数据库，其全称为时间序列数据库。时间序列数据库主要用于指处理带时间标签（按照时间的顺序变化，即时间序列化）的数据，带时间标签的数据也称为时间序列数据，即时序数据。

时间序列是IoTDB中的核心概念。时间序列可以被看作产生时序数据的传感器的所在完整路径，在IoTDB中所有的时间序列必须以root开始、以传感器作为结尾。一个时间序列也可称为一个全路径。

通过其查询引擎，IoTDB提供了 JDBC 访问 API，简单易用。

## 2.4 后端开发思想MVC

MVC 模式代表 Model-View-Controller（模型-视图-控制器） 模式。这种模式用于应用程序的分层开发[4]：

* Model（模型）：模型代表一个存取数据的对象或 JAVA POJO。它也可以带有逻辑，在数据变化时更新控制器；
* View（视图）：视图代表模型包含的数据的可视化；
* Controller（控制器）：控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。

模型－视图－控制器（MVC）是Xerox PARC在八十年代为编程语言Smalltalk－80发明的一种软件设计模式，至今已被广泛使用。

## 2.5 android开发

安卓项目使用Gradle构建，支持maven， Ivy仓库，支持传递性依赖管理，而不需要远程仓库或者是pom.xml和ivy.xml配置文件，基于Groovy，build脚本使用Groovy编写，项目采用MVP架构，底层依赖Linux 内核。

## 2.6 数据采集端

使用WISE\_4012E与传感器连接获取传感器的实时状态，将设备加入局域网后通过http协议和Basic身份认证从wise不断获取json类型数据解析处理后发给后台服务器，并通过搭建http服务器接收来自后台和用户的命令。

# 3设计步骤

## 3.1系统分析

### 3.1.1 需求分析

* **编写目的**：本需求分析对 “基于IoTDB和WISE的盆栽养护系统”项目进行了规范，明确了项目需求可作为系统设计、项目目标及项目验收的依据。
* **适用范围：**本需求分析适用于“基于IoTDB和WISE的盆栽养护系统” 的应用模型和功能需求描述。
* **系统功能模块划分：**需要实现基于IoTDB和WISE的盆栽养护系统的各项功能，同时需要符合题目的其他各项明确或隐含需求。系统从功能上划分为三个模块，分别为后端、数据采集端以及客户端：

①后端对数据进行处理、存储，将数据持久化，同时，对前端的请求作出响应。

②数据采集端通过温度传感器与湿度传感器对温湿度数据进行采集，获得数据后再通过WISE通信将数据传送至后端提供的接口，也可以收到来自后端的指令，并进行处理。

③android客户端与后端进行交互获取温湿度实时数据、实时平均数据、历史时段平均数据以及发送指令等。

### 3.1.2可行性分析

本盆栽养护系统涉及从数据获取、后端处理、客户端展示一整套流程，涉及技术点都可以得到解决，具有技术可行性；

基于IoTDB和WISE的盆栽养护系统开发之后再进一步完善可以有广泛的应用，具有商业潜力和应用可行性。

## 3.2 系统设计

系统从开发上划分三个部分，分别是后端、数据采集端、客户端。各个部分完成相应的功能：

后端对数据进行处理、存储，将数据持久化，同时，对前端的请求作出响应。

数据采集端通过温度传感器与湿度传感器对温湿度数据进行采集，获得数据后再通过WISE通信将数据传送至后端提供的接口，也可以收到来自后端的指令，并进行处理。

android客户端与后端进行交互获取温湿度实时数据、实时平均数据、历史时段平均数据以及发送指令等。

系统的具体功能如下：

* 历史时段数据（通过折线图显示当天历史某一频率时段内的平均温湿度值，如早上8:00进入程序，若显示频率为1小时一次，则应显示0到8之间的每个小时的平均温湿度数据）；
* 实时温湿度数据显示（通过折线图以两秒一次的频率实时显示温度、湿度数据）；
* 异常通知（出现异常情况时发送通知，并在保存在通知列表，通知列表可删除）；
* 发送指令（用户可通过客户端向后端发送请求，后端再向数据采集端发送控制指令，以调控温湿度）；

系统模块图如下图3.1所示：

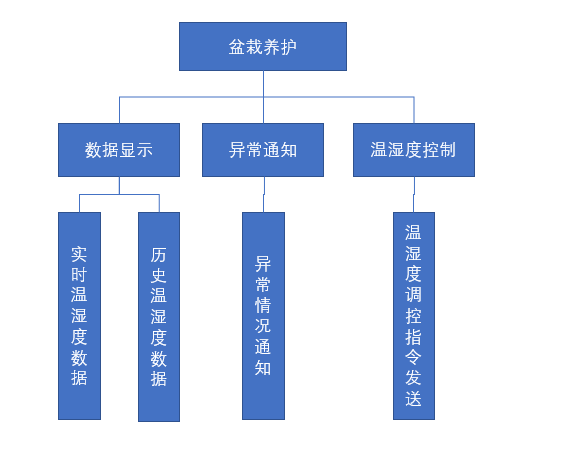
****

图3.1系统模块设计图

## 3.3详细设计

### 3.3.1数据库设计

本系统采用时序数据库IoTDB。

根据对系统所做的需求分析、系统设计、规划出基于IoTDB和WISE的盆栽养护系统相关存储组为root.plant，设备为d1，传感结点为s1/s2/s3，分别为温度、湿度以及情况记录，时间序列为：root.plant.d1.s1, root.plant.d1.s2, root.plant.d1.s3,相关配置如下图所示：

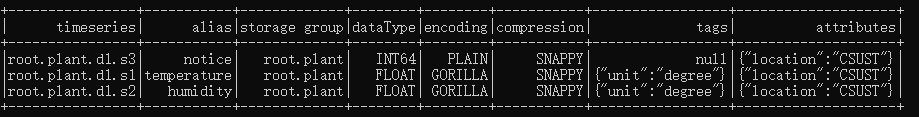


图3.2 系统时间序列配置图

### 3.3.2功能详细实现

本系统运行时前端和数据采集端通过url向后台发送请求和参数，后台controller调用service层，service层调用dao层，dao实现对请求和数据进行处理，并返回处理结果，service实现类对处理结果进行判断、封装并将封装结果传回controller，由controller发送出去。前端接收后台数据后在页面上动态渲染，由此实现前后端的数据交互。

**1.后端数据封装格式**

客户端请求后获取的数据都会封装成以下格式，code为状态码，msg为携带信息，data为传送数据。

1. @Data
2. @AllArgsConstructor
3. @NoArgsConstructor
4. **public** **class** Result {
5. /\*\*
6. \* -1获取失败，0成功，1温度过高，2温度过低，3表示过湿，4表示湿度不够
7. \*/
8. **private** Integer code;
9. **private** String msg;
10. **private** Object data;
11. }

**2.状态判断**

在本系统中对温湿度的异常情况封装成一个Warn类，并设置了对应的判断方法。Warn类中有四个属性，分别表示温、湿度的阈值。

1. **public** **static** **final** Float MaxTemperature = 30.0f;
2. **public** **static** **final** Float MinTemperature = 10.0f;
3. **public** **static** **final** Float MaxHumidity = 0.7f;
4. **public** **static** **final** Float MinHumidity = 0.4f;

温度判断，根据温度所在的区间返回不同的数值。

1. **public** **static** Integer isTempSuit(Float tem) {
2. **if** (tem == **null**) {
3. **return** -1;
4. }
5. **if** (tem >= MaxTemperature) {
6. **return** 1;
7. } **else** **if** (tem <= MinTemperature) {
8. **return** 2;
9. } **else** {
10. **return** 0;
11. }
12. }

湿度判断，根据湿度所在的区间返回不同的数值。

1. **public** **static** Integer isHumSuit(Float humidity) {
2. **if** (humidity == **null**) {
3. **return** -1;
4. }
5. **if** (humidity >= MaxHumidity) {
6. **return** 3;
7. } **else** **if** (humidity <= MinHumidity) {
8. **return** 4;
9. } **else** {
10. **return** 0;//适宜
11. }
12. }

**3.数据库连接**

①连接IoTDB：IoTDB 通过查询引擎提供了JDBC访问 API，数据库连接驱动为org.apache.IoTDB.jdbc.IoTDBDriver，url前缀为 jdbc:IoTDB: ，username和password均默认为root。下面代码即可返回一次连接。

1. **public** **static** Connection getConnection() {
2. Connection connection = **null**;
3. **try** {
4. Class.forName(driver);
5. connection = DriverManager.getConnection(url, username, password);
6. } **catch** (ClassNotFoundException e) {
7. e.printStackTrace();
8. } **catch** (SQLException e) {
9. e.printStackTrace();
10. }
11. **return** connection;
12. }

②连接池：为了提升系统的并发性能让系统响应速度更快、提高资源重用，同时统一管理连接，避免数据库泄露，系统中创建了数据库连接池。在IoTDBPool类中具有两个属性

1. **private** **static** LinkedList<Connection> pool;
2. **private** **static** Integer MAX\_NUM = 5;

LinkedList<Connection> pool是连接链表，Integer MAX\_NUM是最大连接数，暂定为5。系统启动后首先会创建5个连接存放在连接链表中，需要时会首先从连接链表中获取，使用完后再返回给链表，可以避免大量的重复创建连接过程，提高系统响应速度。

1. **public** **static** Connection getConnection() **throws** SQLException {
2. **if** (pool == **null**) {
3. pool = **new** LinkedList<>();
4. **for** (**int** i = 0; i < MAX\_NUM; i++) {
5. pool.add(IoTDBUtil.getConnection());
6. }
7. }
8. **if** (pool.size() <= 0) {
9. //创建零时连接，可以防止出现大量空闲连接
10. **return** IoTDBUtil.getConnection();
11. }
12. //返回第一个连接，并从列表中去除
13. **return** pool.remove();

当连接数超过最大连接数时采用创建零时连接的方式创建新连接，防止出现大量空闲连接，耗费系统资源。返回连接则如下所示。

1. **public** **static** **void** remove(Connection connection) {
2. pool.add(connection);
3. }

**4.部分功能详解**

①数据存储：数据采集端通过后端提供接口发送数据，后端获取数据后标记其状态并将数据存入IoTDB中。controller层从请求体中获取json格式时间戳以及对应的温湿度数据并解析，然后调用service层。

1. @PostMapping("/insert")
2. **public** Result insert(@RequestBody JSONObject jsonObject) {
3. Long time = **null**;
4. Float tem = **null**;
5. Float humidity = **null**;
6. **try** {
7. time = Long.parseLong(valueOf(jsonObject.get("time")));
8. tem = Float.parseFloat(valueOf(jsonObject.get("temperature")));
9. humidity = Float.parseFloat(valueOf(jsonObject.get("humidity")));
10. } **catch** (ClassCastException e) {
11. e.printStackTrace();
12. **return** **new** Result(-1, "插入参数有误", **null**);
13. }
14. **return** iotService.insert(time, tem, humidity);
15. }

在service层的对应方法public Result insert(Long time, Float tem, Float hum)中首先对温湿度状态进行判断，并调用dao层方法public boolean addNotice(Long time, int status)将异常情况存入IoTDB数据库中。异常情况用二进制数表示，01温度过高，00温度过低，11湿度过高，10湿度过低。

1. **int** t = Warn.isTempSuit(tem);
2. **int** h = Warn.isHumSuit(hum);
4. **if** (t == 1) {
5. iotDao.addNotice(time, 1);
6. } **else** **if** (t == 2) {
7. iotDao.addNotice(time, 0);
8. }
9. **if** (h == 3) {
10. iotDao.addNotice(time, 11);
11. } **else** **if** (h == 4) {
12. iotDao.addNotice(time, 10);
13. }

然后调用dao中的方法，将数据插入。

1. **if** (iotDao.insert(time, tem, hum)) {
2. **return** **new** Result(0, "存取成功", **null**);
3. } **else** {
4. **return** **new** Result(-1, "存取失败", **null**);
5. }

在dao层，分别有insert和addNotice方法被调用。插入温湿度数据时首先从连接池中获取连接，通过prepareStatement类对sql语句进行预处理，设置参数，再调用executeUpdate()方法将数据插入数据库。

sql语句为：

1. String sql = "insert into root.plant.d1(timestamp ,temperature, humidity) values (?,?,?)";

获取连接：

1. connection = IoTDBPool.getConnection();

预处理：

1. pstmt = connection.prepareStatement(sql);
2. pstmt.setLong(1, time);
3. pstmt.setFloat(2, tem);
4. pstmt.setFloat(3, humidity);

执行：

1. row = pstmt.executeUpdate();

回收连接：

1. IoTDBPool.remove(connection);

addNotice（）方法中sql语句为：

1. String sql = "insert into root.plant.d1(timestamp,notice) values(?,?)";

②实时获取温度，客户端发送请求，通过JSON格式提交时间戳数据，controller调用service层getTemp（）方法，service从dao层获取数据，并对数据进行判断、封装后返回给controller，再发送给客户端。

调用dao层方法获取数据：

1. Map<String, Object> map = iotDao.getTemp(questTime);
2. Float tem = (Float) map.get("value");
3. Integer suit = Warn.isTempSuit(tem);

对数据进行封装：

1. **switch** (suit) {
2. **case** 0:
3. **return** **new** Result(0, "温度适宜", map);
4. **case** 1:
5. **return** **new** Result(1, "温度过高", map);
6. **case** 2:
7. **return** **new** Result(2, "温度过低", map);
8. **default**:
9. **return** **new** Result(-1, "未知错误", **null**);
10. }

dao层方法：根据前端请求时间戳选择3秒内的平均值作为实时数据传送，因为如果只按照单一时间戳条件选择的话返回结果集可能为空。

1. **public** Map<String, Object> getTemp(**long** questTime) {
2. **long** gap = 1000 \* 3;//3s
3. **long** begin = questTime - gap;
4. String sql = "select avg(temperature) from root.plant.d1 where time <? and time>?";
5. Float temperature = **null**;
6. **try** {
7. connection = IoTDBPool.getConnection();
8. pstmt = connection.prepareStatement(sql);
9. pstmt.setLong(1, questTime);
10. pstmt.setLong(2, begin);
11. ResultSet resultSet = pstmt.executeQuery();
12. **while** (resultSet.next()) {
13. temperature = Float.parseFloat(resultSet.getString("avg(root.plant.d1.temperature)"));
14. }
15. } **catch** (SQLException e) {
16. e.printStackTrace();
17. } **finally** {
18. //释放回连接池
19. IoTDBPool.remove(connection);
20. Map<String, Object> map = **new** HashMap<>();
21. map.put("value", temperature);
22. **return** map;
23. }
24. }

③获取当天历史数据，每次用户打开客户端时都会向后端发送请求获取当天各个时段内的平均数据，以温度为例。在dao层中，需要传入一个参数，即间隔时间，单位为秒，类型为长整型，首先获取当天零点的时间戳，通过滑动窗口函数，设置窗口大小为传入参数，窗口移动步长也为传入参数，窗口时间范围为当天零点到当前时间。如果传入参数为900（秒），即15分钟，进入时间为早上8点，则会将当天0点到8点间每15分钟的平均温度返回给客户端。

1. **public** List<Map<String, Object>> allTem(Long gap) {
2. **long** now = System.currentTimeMillis();
3. **long** dayBegin = now - (now + 60 \* 60 \* 8 \* 1000) % (60 \* 60 \* 24 \* 1000);
4. List<Map<String, Object>> list = **new** ArrayList<>();
5. String sql = " select avg(temperature) from root.plant.d1 group by([?, ?), ?s,?s)";
6. **try** {
7. connection = IoTDBPool.getConnection();
8. pstmt = connection.prepareStatement(sql);
9. pstmt.setLong(1, dayBegin);
10. pstmt.setLong(2, now);
11. pstmt.setLong(3, gap);
12. pstmt.setLong(4, gap);
13. ResultSet resultSet = pstmt.executeQuery();
14. Timestamp time = **null**;
15. String temperature = **null**;
16. **while** (resultSet.next()) {
17. Map<String, Object> map = **new** HashMap<>();
18. time = resultSet.getTimestamp("Time");
19. map.put("time", time);
20. temperature = resultSet.getString("avg(root.plant.d1.temperature)");
21. **if** (StringUtils.hasLength(temperature)) {
22. map.put("value", Float.parseFloat(temperature));
23. } **else** {
24. map.put("value", temperature);
25. }
26. list.add(map);
27. }
28. IoTDBUtil.outputResult(resultSet);
29. } **catch** (SQLException e) {
30. e.printStackTrace();
31. } **finally** {
32. IoTDBPool.remove(connection);
33. **return** list;
34. }
35. }

④发送指令：当出现温湿度情况异常时，用户可通过客户端发送指令，经过后端将指令发送给数据采集端。

1. @PostMapping("/order")
2. **public** Result order(@RequestBody JSONObject jsonObject) {
3. Integer i = Integer.valueOf(String.valueOf(jsonObject.get("order")));
4. System.out.println(jsonObject.toString());
5. **return** iotService.order(i);
6. }

controller层调用service层的order（）方法

service中首先创建一个HttpClient客户端用来发送请求，以json格式发送body参数将指令发送给数据采集端。

1. CloseableHttpClient httpClient = HttpClientBuilder.create().build();
2. // 创建Post请求
3. HttpPost httpPost = **new** HttpPost("http://192.168.43.112:8765/order");
4. // 响应模型
5. CloseableHttpResponse response = **null**;
6. //json参数
7. JSONObject jsonObject = **new** JSONObject();
8. jsonObject.put("order", order);
9. **try** {
10. StringEntity jsonParam = **new** StringEntity(jsonObject.toString());
11. jsonParam.setContentEncoding("UTF-8");//发送数据编码为utf-8
12. jsonParam.setContentType("application/json");//发送json数据需要设置contentType
13. // 由客户端执行(发送)Post请求
14. httpPost.setEntity(jsonParam);
15. response = httpClient.execute(httpPost);
16. // 从响应模型中获取响应实体
17. HttpEntity responseEntity = response.getEntity();
18. **if** (response.getStatusLine().getStatusCode() != 200) {
19. //请求失败则返回
20. **return** **new** Result(-1, "操作失败", response.getStatusLine());
21. }
22. **if** (responseEntity != **null**) {
23. System.out.println("响应内容为:" + JSONObject.parseObject(EntityUtils.toString(responseEntity)).getInteger("data"));
24. }
25. } **catch** (ClientProtocolException e) {
26. e.printStackTrace();
27. }

## 3.4测试

### 3.4.1功能测试

①获取实时温度，图中有两条线，红色线为从记录开始的平均温度数据，蓝色线为实时温度数据（由于同一时间内温度变化不大，蓝色线被红色线覆盖），点击节点可查看数值大小，如图所示。

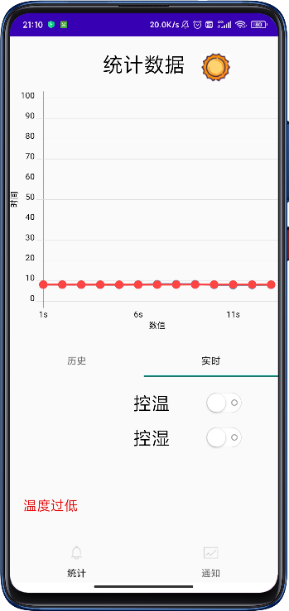


图3.3 实时温度数据

②获取实时湿度，通过改变湿度获取湿度变化图，蓝色线为实时湿度，红色为平均湿度，如图所示。

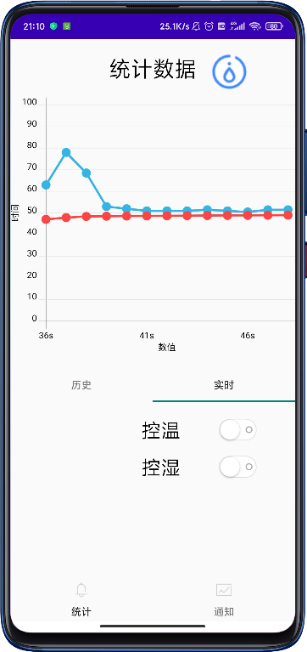


图3.4 实时湿度数据

③获取当天历史温度数据，21点之前由于无数据，数值为0 , 21点开始接受数据，如图所示。

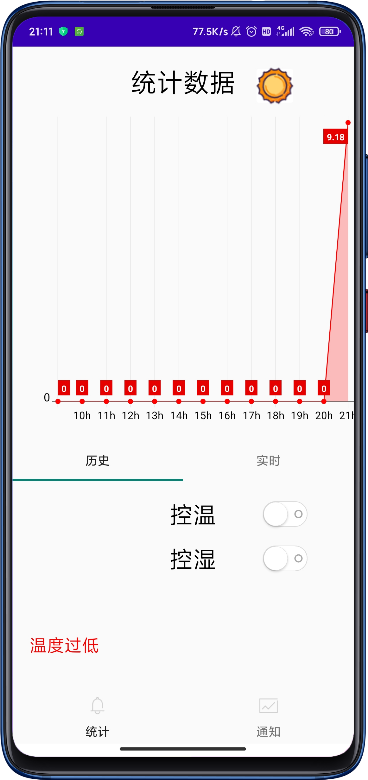


图3.5 当天历史温度数据

④获取当天历史湿度数据，如图所示。

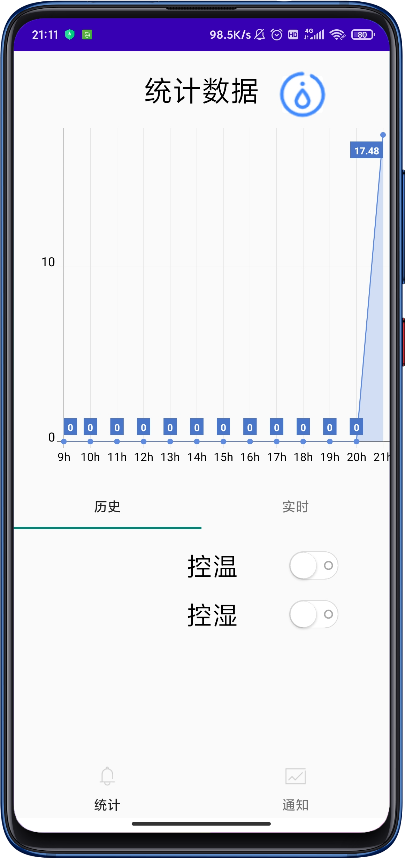


图3.6 当天历史湿度数据

⑤异常状态通知，状态异常会用红色字体显示出来，并保存在手机本地，可通过历史记录表查看，也可删除历史记录。如图所示。

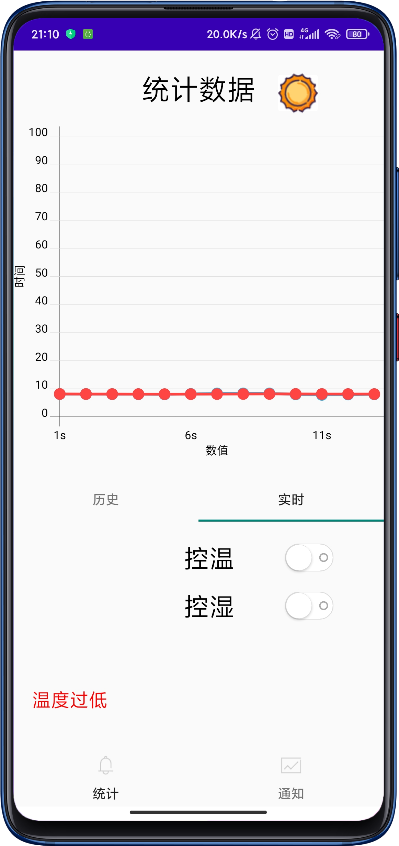
 

图3.7 温度过低报警 图3.8 历史情况记录

⑥异常状态处理，当出现异常状况时可通过客户端发送指令，数据采集端获得指令后会进行相应处理。

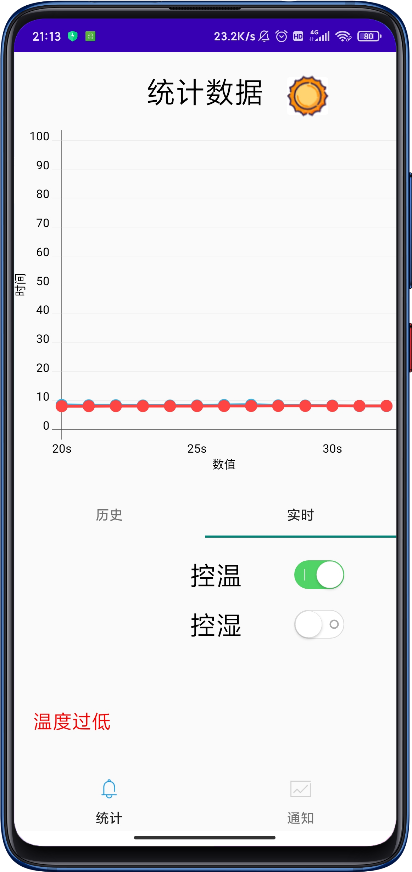
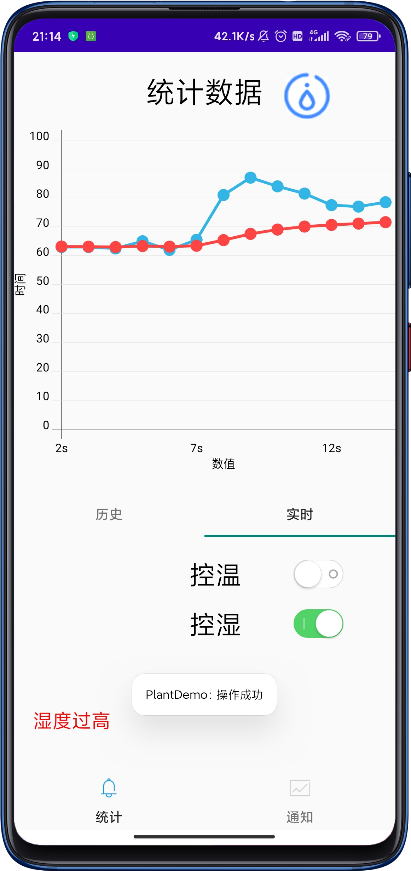
 

图3.9 控温 图3.10 控湿

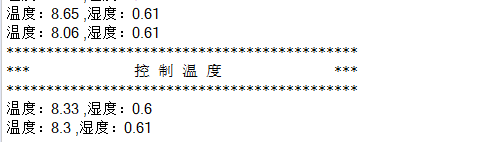


图3.11 数据采集端控制温度

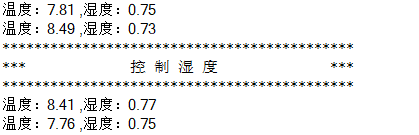


图3.12 数据采集端控制湿度

### 3.4.1测试结果

经过反复测试，系统运行状况良好，各项数据准确，数据响应时间快速，达成预期目标。