

# 目录

<b>一、 设计背景与目标</b>	<b>1</b>
1.1 设计背景	1
1.2 设计目标	1
<b>二、 课程设计器材</b>	<b>1</b>
2.1 硬件平台	2
2.2 软件平台	2
<b>三、 设计方案</b>	<b>3</b>
3.1 思路分析	3
3.2 系统架构	3
3.3 功能设计	3
3.4 技术栈选型	5
<b>四、 设计过程</b>	<b>5</b>
4.1 服务器搭建	6
4.1.1 服务器选用	6
4.1.2 数据库搭建	6
4.1.3 MQTT 服务器搭建	7
4.2 硬件端搭建	8
4.2.1 NB-IoT 联网	8
4.2.2 DHT22 模块安装	10
4.2.3 MQTT 消息发布	10
4.2.4 WiFi 版本编写	10
4.2.5 子功能测试	10
4.3 前端设计	12
4.3.1 页面布局	12
4.3.2 数据请求	12
4.3.3 设备管理	14
4.3.4 数据分析	14
4.3.5 历史数据	15
4.4 后端开发	15
4.4.1 数据库的连接	15
4.4.2 MQTT 服务器的配置	16
4.4.3 服务器连接与数据获取	16
4.4.4 接口设计	19

<b>五、 系统测试</b>	<b>20</b>
5.1 系统部署 . . . . .	20
5.2 功能测试 . . . . .	20
5.2.1 主页视图 . . . . .	20
5.2.2 动态视图 . . . . .	21
5.2.3 报表导出 . . . . .	22
5.2.4 设备管理 . . . . .	23
5.3 效果分析 . . . . .	23
<b>六、 心得与体会</b>	<b>24</b>

## 摘 要

环境温湿度监测在许多行业中具有重要意义，包括农业、物流、仓储、医疗等众多重要领域，因此本次课程设计选定的主题是基于 NB-IoT 的无线温湿度监测系统设计与实现。在项目中，我们团队着眼于无线监测节点的设计，通过整合 ESP8266、SIM7020 模块和 DHT22 模块，成功实现了 NB-IoT 联网和温湿度数据的获取。此外，前后端的协同工作也是项目中不可或缺的一部分，我们实现了从硬件到前后端的全面开发，包括数据的存储与渲染，并实现了一系列功能需求，体现了系统功能的复杂性与可用性。

在硬件设计方面，ESP8266 的编程为监测节点提供了智能控制和数据处理功能。而 SIM7020 模块的选择使得监测节点能够实现 NB-IoT 联网，为远程通信提供了高效且可靠的手段。结合 DHT22 模块，成功实现了环境温湿度信息的高精度采集，为系统提供了全面感知环境变化的能力。这一节点设计不仅是系统的核心，也为整个监测系统的稳定运行提供了可靠保障。

在软件平台的选择上，我们采用了 Visual Studio Code、ArduinoIDE 和 IDEA 等工具，分别用于前端代码的编写与测试，硬件代码的编写与烧录，以及后端代码的编写。这些工具的灵活运用使得我们能够高效地实现系统的各个模块，体现了团队在物联网开发技巧上的熟练掌握。与此同时，我们成功配置了云服务器，安装了数据库，并搭建了 MQTT 服务器用于消息汇总与传递。这一系列操作使得我们更深刻地理解了云计算和物联网的结合，为系统的可扩展性和数据处理提供了强大支持。

最后，通过对整个设计过程的总结，我们团队获得了丰富的物联网开发经验。从硬件到前后端，我们不仅掌握了各类开发技巧，还培养了解决实际问题的能力。项目的成功实施不仅是对所学知识的应用，更是对团队整体实际能力的全面提升。

**关键词：**NB-IoT   ESP8266 模块   MQTT 消息队列   前后端分离   温湿度监测

# Abstract

Environmental temperature and humidity monitoring holds significant importance in various industries, including agriculture, logistics, warehousing, healthcare, and many other critical sectors. Therefore, the chosen theme for this course project is the design and implementation of a wireless temperature and humidity monitoring system based on NB-IoT (Narrowband Internet of Things). In this project, our team focused on the design of wireless monitoring nodes, successfully integrating the ESP8266, SIM7020 module, and DHT22 module to achieve NB-IoT connectivity and temperature-humidity data acquisition. Additionally, the collaboration between the frontend and backend was an integral part of the project. We accomplished comprehensive development from hardware to frontend and backend, including data storage and rendering, implementing a series of functional requirements that showcase the complexity and usability of the system.

In terms of hardware design, the programming of the ESP8266 provides intelligent control and data processing capabilities for the monitoring nodes. The selection of the SIM7020 module enables NB-IoT connectivity, offering an efficient and reliable means for remote communication. Combining the DHT22 module, we successfully achieved high-precision collection of environmental temperature and humidity information, providing the system with the ability to comprehensively perceive changes in the environment. This node design is not only the core of the system but also ensures the stable operation of the entire monitoring system.

For the software platform selection, we used tools such as Visual Studio Code, Arduino IDE, and IDEA for frontend code writing and testing, hardware code writing and burning, and backend code writing, respectively. The flexible use of these tools allowed us to efficiently implement various modules of the system, demonstrating the team's proficiency in IoT development skills. At the same time, we successfully configured a cloud server, installed a database, and set up an MQTT server for message aggregation and transmission. This series of operations deepened our understanding of the integration of cloud computing and IoT, providing robust support for the system's scalability and data processing.

In conclusion, through summarizing the entire design process, our team gained rich experience in IoT development. From hardware to frontend and backend, we not only mastered various development skills but also cultivated the ability to solve practical problems. The successful implementation of the project is not only an application of acquired knowledge but also a comprehensive enhancement of the overall capabilities of the team.

**Keywords:** NB-IoT   ESP8266 module   MQTT message queue   Frontend and backend separation   Temperature and humidity monitoring

## 一、 设计背景与目标

### 1.1 设计背景

环境温湿度监测在许多行业中具有重要意义，包括农业、物流、仓储、医疗等众多重要领域。通过实时监测环境的温度和湿度，可以提前预警并及时采取措施，保证设备的正常运行，提高生产效率，防止质量问题的发生。除此之外，全球气候变化和环境问题引起了广泛关注，气温升高、极端天气和自然灾害频发成为不容忽视的现实。

在当下，许多企业和机构依然采用传统的手工监测模式，投入数量有限的监测点，并投入一定量的人力，导致监测数据出现获取不及时、准确性低等状况。这类方式存在一定的弊端，包括高昂的成本、效率低下、难以实现全面覆盖等问题。但随着对环境温湿度的要求不断提高，许多行业迫切需要一种可靠且智能化的环境温湿度监测报警系统。

这种系统将基于物联网技术，通过部署一定数量的传感器设备，在关键位置实时监测环境的温度和湿度，通过数据采集和处理，不仅可以实时监测环境参数，还可以根据预设条件自动触发报警机制，实现对温湿度的全面监测和分析，为用户提供实时的环境状态信息、报警提示和环境状态预测，帮助用户尽早发现异常情况并采取相应的措施。

### 1.2 设计目标

本次课程设计题目为《基于 NB-IoT 的无线温湿度监测系统设计与实现》，包含以下若干需要实现的设计目标。

1. 基于 NB-IoT 技术进行应用开发，实现一个完善的智能温湿度监测系统；
2. 掌握基本的物联网开发技术，包括无线节点的代码设计与写入、通过无线信道进行信息传输；
3. 熟悉 NB-IoT 的基本原理与常见硬件模块的使用方式；
4. 熟悉各类常见传感器的电气特性与使用方法，例如 DHT 系列传感器、蜂鸣器等；
5. 掌握基础全栈开发技术，学习常用前后端框架，以完成应用平台的开发；
6. 培养在实践中发现问题与解决问题的能力。

## 二、 课程设计器材

选择合适的硬件平台和软件平台可以为本次课程设计提供极大的设计便利，利于最终的智能温湿度监测系统的实现。

## 2.1 硬件平台

硬件平台的选用将综合考虑性能、稳定性、成本、使用门槛、平台拓展性等方面。节点的开发板方面，本次实验选用 ESP8266 模块，NB-IoT 联网方面选用 SIM7020 模块，温湿度监测方面选用 DHT22 模块。这些器件的主要特点如下。

- **ESP8266**: ESP8266 是一款低成本、低功耗的开发模组，具有高性能的 Wi-Fi 模块，内部集成 MCU 以实现串口通信。选用该模块的原因包括：成本较低，试验代价较小；具有 Wi-Fi 模块，联网方便，便于开发过程中的信息传输调试。
- **SIM7020**: SIM7020 模块是一种低功耗、小型化的 NB-IoT 模块，专为物联网应用设计，可以用于远程设备的低功耗、长距离通信。具有低功耗、广覆盖、高连接性的特点。选用 SIM7020 模块可以实现设备与云端的可靠、高效的通信，使得温湿度监测系统可以在不同地点进行数据传输。
- **DHT22**: DHT22 是一种数字温湿度传感器，能够测量环境中的温度和湿度，并通过数字信号输出这些数据，是一种成本效益高且易于使用的温湿度传感器。它提供了对环境条件的准确监测，是制作低成本、高效率温湿度监测系统的理想选择。

## 2.2 软件平台

- **Arduino IDE**: Arduino 是一种开源电子原型平台，包括硬件和软件，它使用简单的 C/C++ 语言编写，为物联网和嵌入式系统提供了一种易于使用的开发环境。使用其集成开发环境（IDE）能够非常快速地开发并测试物联网设备的硬件代码，故本次课程设计主要使用 Arduino 作为硬件的开发平台。
- **IDEA: IntelliJ**: IDEA 是一款由 JetBrains 开发的 Java 集成开发环境，适用于后端的代码编写、测试、打包，有助于后端的开发效率。
- **Visual Studio Code**: VSCode 是一款轻量级的开源代码编辑器，支持多种编程语言，本次课程设计主要使用其进行前端代码的开发，它提供了丰富的前端插件，简化了开发流程。
- **EMQX**: EMQX 是一个开源的分布式物联网消息中间件，支持 MQTT 协议，适用于大规模物联网应用。EMQX 作为消息队列服务器，为各节点提供了可靠的消息传递机制，且可以自主进行 EMQX 服务器的搭建，因而本次实验也选用了 EMQX 服务器作为消息传输的载体。

## 三、 设计方案

### 3.1 思路分析

对于无线智能温湿度监测系统，需要考虑以下若干方面：不同位置的温湿度信息获取、温湿度信息的汇总与存储、数据的合理处理与呈现。

对于温湿度信息的获取，需要使用相关的温湿度信息监测模块，并连接到某一节点上，节点需要通过温湿度传感器获取温湿度信息，并对信息进行合理的处理，包括异常值的筛选、监测时间的记录。且每个节点之间是独立且可分辨的，可以互不干扰地进行温湿度信息的获取，通过多节点实现不同空间位置上的监测。

对于温湿度信息的汇总，需要考虑节点之间的信息共享，因此需要网络的建立，本次课程设计需要使用窄带物联网进行通信，因此各节点需要配备 NB-IoT 的通信模块。除此之外，各传感节点中获取的信息各不相同，且具有异步特性，故考虑使用消息队列服务器进行信息的汇总处理，本次设计采用 EMQX 服务器作为消息汇总的中心，各节点共同订阅某一主题，并在获取到温湿度数据后将数据进行打包（添加节点编号、时间戳等信息），并统一通过 NB-IoT 发送至 EMQX 服务器的特定主题，以实现信息的共同发布与汇总。

对于数据的存储，考虑使用后端程序（本次课程设计选用 springboot 框架）实现，与各传感节点类似，后端程序也应作为用户订阅 EMQX 服务器中的指定主题，但应作为信息的接收者，在接收到传感器的信息发布后进行消息的接收，并连接数据库（选用 MySQL）进行数据的存储。

对于数据的处理与呈现，考虑使用前端 Web 应用进行报表的呈现，同时提供给用户操作数据的接口，包括阈值设定，动态报表呈现，数据的导出等，前端代码需向后端进行数据的请求，后端程序也应当根据前端的需求提供合理可用的接口。

### 3.2 系统架构

根据 3.1 节中所述的设计思路，可以进行系统整体的架构图的呈现，大致结构如图 3.1 所示。

为更好地说明系统整体的运行方式，考虑对以下过程进行流程分析：从某一节点采集数据开始直至数据呈现于用户。流程图如图 3.2 所示。

可以确定，硬件端与软件端之间的运行相对独立，即前文所提及的硬件端的信息发布具有异步性，因此软件段需要在运行后持续等待硬件层面的数据传输。

### 3.3 功能设计

在正式进行课程设计的开发前，我们进行了系统的系列需求分析，确定了一系列面向用户的功能。

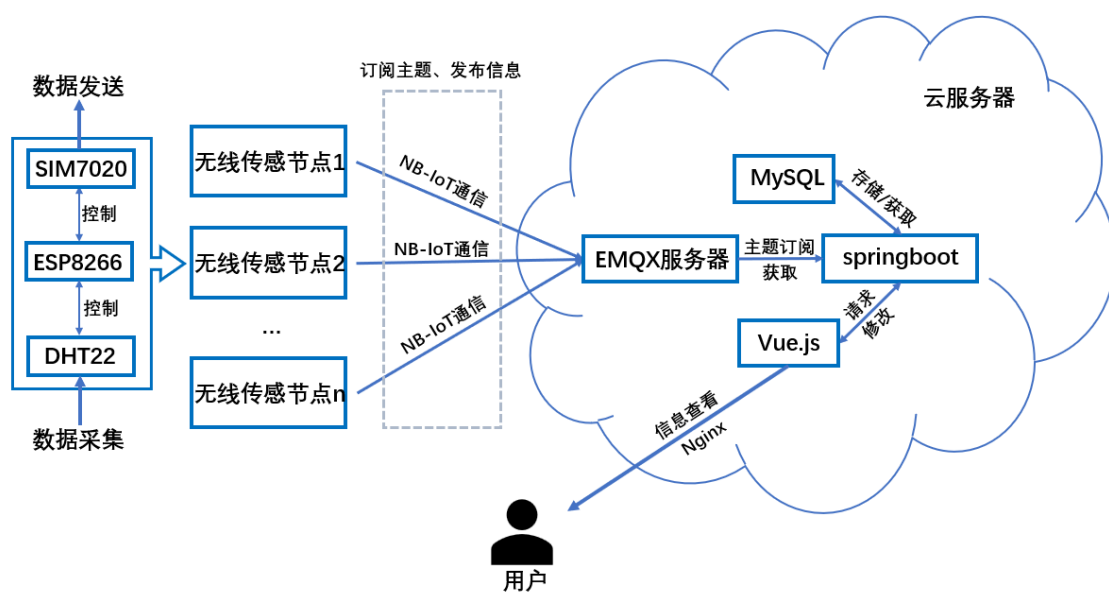


图 3.1 系统整体逻辑结构

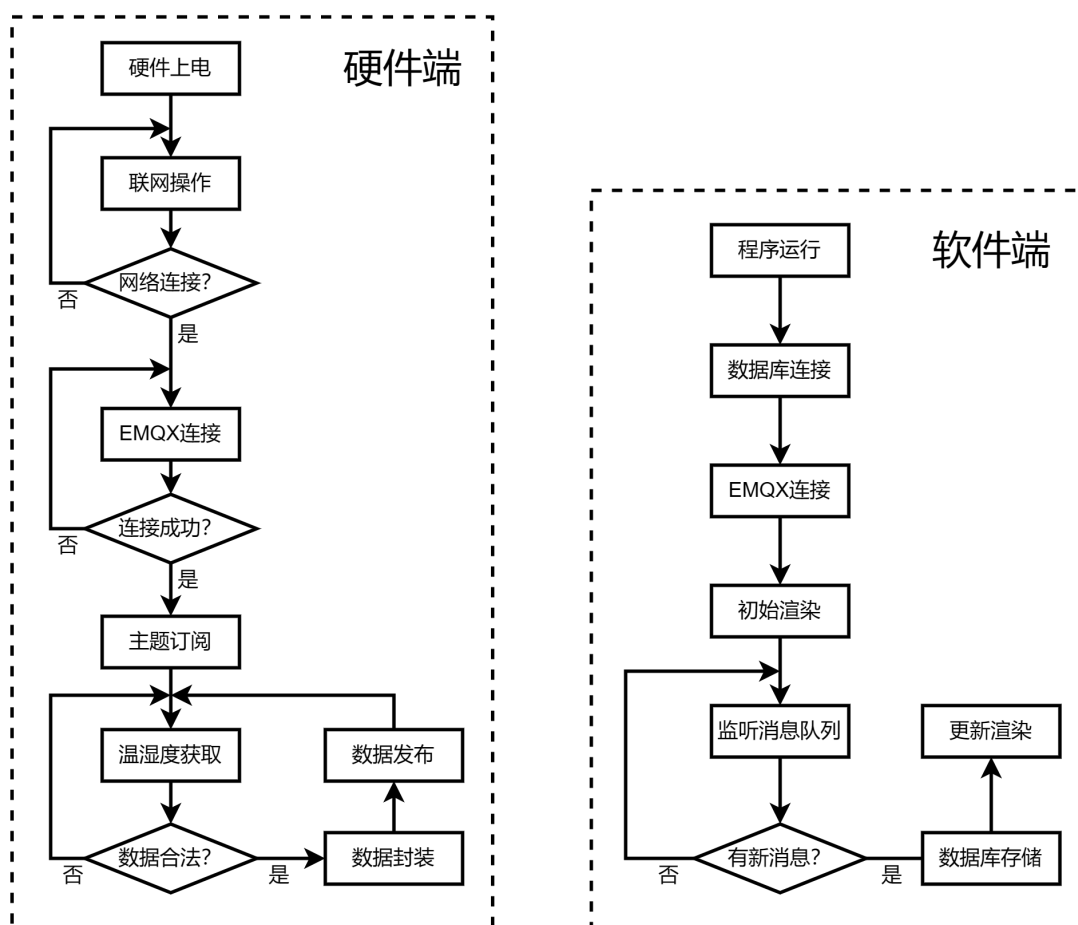


图 3.2 系统运行流程图



该系统的核心功能包括：实时监测、数据分析与渲染呈现、异常报警设置与通知、历史数据查询、日志报告导出等部分。具体的功能描述如下表 3.1 所示。

**表 3.1 系统功能需求表**

功能概述	详细描述
实时监测	系统具备实时监测环境温湿度的功能，具有时效性。
数据分析与渲染呈现	系统能够对采集到的温湿度数据进行分析，提供直观的趋势展示，例如二维折线、三维热力图等。
异常报警设置与通知	系统支持自定义的报警规则设置。温湿度数据超出设定的阈值将触发报警机制，并以短信、电子邮件等形式通知相关人员。
历史数据查询	系统将保存一定时间范围内的历史数据，可以通过用户界面查询历史温湿度数据
日志报告导出	系统根据指定的时间间隔进行报告的生成，并提供导出服务器，包括数据分析表、异常情况统计
设备管理	系统提供对所有监测设备的统一管理服务平台，供管理员进行设备管理、系统设置

### 3.4 技术栈选型

本次课程设计选用的技术栈列举如下。

- **服务器端：**Spring boot、MyBatis、Spring Security、Mysql、Redis、EMQX
- **Web 端：**ES6、Vue、Vuex、Vue-router、Vue-cli、Axios、Element-ui
- **硬件端：**ESP、Arduino、SIM、DHT

## 四、 设计过程

设计实现包括以下若干步骤：服务器搭建、硬件端设计实现、前端页面设计、后端接口提供。

## 4.1 服务器搭建

### 4.1.1 服务器选用

考虑到数据库的配置、MQTT 服务器的搭建需要对外提供便捷的接口，同时方便对其进行维护，本次课设考虑将上述内容部署于云服务器中。选用的云服务器来自于华为云 ECS，配置 4 核 CPU 与 8G 内存，预装操作系统为 CentOS7.6 64 位。如图 4.1 所示。



图 4.1 云服务器选用规格信息

### 4.1.2 数据库搭建

考虑到无线节点获取的温湿度数据需要在数据库中进行存储，因此需要在 ECS 上进行数据库的配置与安装，本次设计选用的数据库版本为 MySQL8.0.35 for Linux on aarch64。在云服务器上进行数据库的安装，并保证数据库服务启动并能够通过 3306 端口进行访问，如图 4.2 所示，数据库服务启动。

```
[root@esc-mqtt ~]# systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2024-01-05 10:52:02 CST; 3min 23s ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
   Process: 888 ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
  Main PID: 1028 (mysqld)
    Status: "Server is operational"
   CGroup: /system.slice/mysqld.service
           └─1028 /usr/sbin/mysqld
```

图 4.2 MySQL 安装与服务启动

由于给定的云服务器具有公网 IP，因此可以远程对该数据库进行访问连接，本次实验采用 HeidiSQL 作为数据设计的管理工具，使用 HeidiSQL 连接至云端数据库，首

先通过 root 用户创建了一个名为 mysql\_mqtt 的数据库，以存储后续的数据表，并作为后端连接的主数据库，进行数据表的设计，考虑到这类应用场景数据类型单一，只考虑设置两张数据表，分别为记录 record 与硬件信息表 hardware。

对于硬件信息表，其中包含字段为：硬件 ID（主键）、温度阈值上下限、湿度阈值上下限、硬件类型、硬件状态、收集数据总数、硬件描述，如图 4.3 所示。

#	名称	数据类型	长度/集合
1	hardid	INT	10
2	tempmax	FLOAT	
3	tempmin	FLOAT	
4	humimax	FLOAT	
5	humimin	FLOAT	
6	model	VARCHAR	50
7	state	BIT	1
8	datanum	INT	10
9	description	VARCHAR	50

图 4.3 硬件信息表 hardware 内容

对于记录表，其中包含字段为：记录 ID（主键）、来源设备（依赖于硬件信息表）、时间戳信息、温度信息、湿度信息，如图 4.4 所示。

字段: + 添加 × 删除 ▲ 向上 ▼ 向下

#	名称	数据类型	长度/集合
 1	id	INT	10
 2	froms	VARCHAR	50
3	timestamp	VARCHAR	50
4	temperature	VARCHAR	50
5	humidity	VARCHAR	50

图 4.4 记录表 record 内容

### 4.1.3 MQTT 服务器搭建

本次课设选用 MQTT 服务器作为消息队列服务器，考虑到维护和测试的方便，故自主进行 MQTT 服务器的搭建，与数据库服务器类似对外提供 EMQX 服务，接受各用户的订阅与消息共享。选用的 EMQX 服务的版本为 5.1.6，在云服务器上完成 EMQX 服务的安装后，确认 EMQX 服务运行，如图 4.5 所示。

使用 EMQX 服务提供的 18083 端口可以访问 MQTT 服务器的管理后台，便于对集群信息进行查看，例如连接数目、连接状况、数据传输量等，便于后续的设计。后台内

```
[root@esc-mqtt ~]# systemctl status emqx
● emqx.service - emqx daemon
   Loaded: loaded (/usr/lib/systemd/system/emqx.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2024-01-05 10:52:00 CST; 16min ago
   Main PID: 715 (beam.smp)
   CGroup: /system.slice/emqx.service
           └─ 715 emqx -Bd -spp true -A 4 -I0t 4 -SDio 8 -e 262144 -zdbbl 8192 -Q 1048576 -P 2097152 ...
              1788 erl_child_setup 1048576
              1846 /usr/lib/emqx/lib/os_mon-2.8.2/priv/bin/memsup
              1847 /usr/lib/emqx/lib/os_mon-2.8.2/priv/bin/cpu_sup
              1911 /usr/lib/emqx/erts-13.2.2/bin/inet_gethost 4
              1912 /usr/lib/emqx/erts-13.2.2/bin/inet_gethost 4

Jan 05 10:52:00 esc-mqtt systemd[1]: Started emqx daemon.
Jan 05 10:52:01 esc-mqtt bash[715]: WARNING: Default (insecure) Erlang cookie is in use.
Jan 05 10:52:01 esc-mqtt bash[715]: WARNING: Configure node.cookie in /etc/emqx/emqx.conf or over...OKIE
Jan 05 10:52:01 esc-mqtt bash[715]: WARNING: NOTE: Use the same cookie for all nodes in the cluster.
Jan 05 10:52:04 esc-mqtt bash[715]: Listener ssl:default on 0.0.0.0:8883 started.
Jan 05 10:52:04 esc-mqtt bash[715]: Listener tcp:default on 0.0.0.0:1883 started.
Jan 05 10:52:04 esc-mqtt bash[715]: Listener ws:default on 0.0.0.0:8083 started.
Jan 05 10:52:04 esc-mqtt bash[715]: Listener wss:default on 0.0.0.0:8084 started.
Jan 05 10:52:04 esc-mqtt bash[715]: Listener http:dashboard on :18083 started.
Jan 05 10:52:04 esc-mqtt bash[715]: EMQX 5.1.6 is running now!
```

图 4.5 EMQX 服务运行

容概览如图 4.6 所示。为方便对具体的消息内容进行查看与确认，我们选用了 MQTTX 客户端进行服务器的连接与主题的监测，从而能够更好地确定消息发布状态。

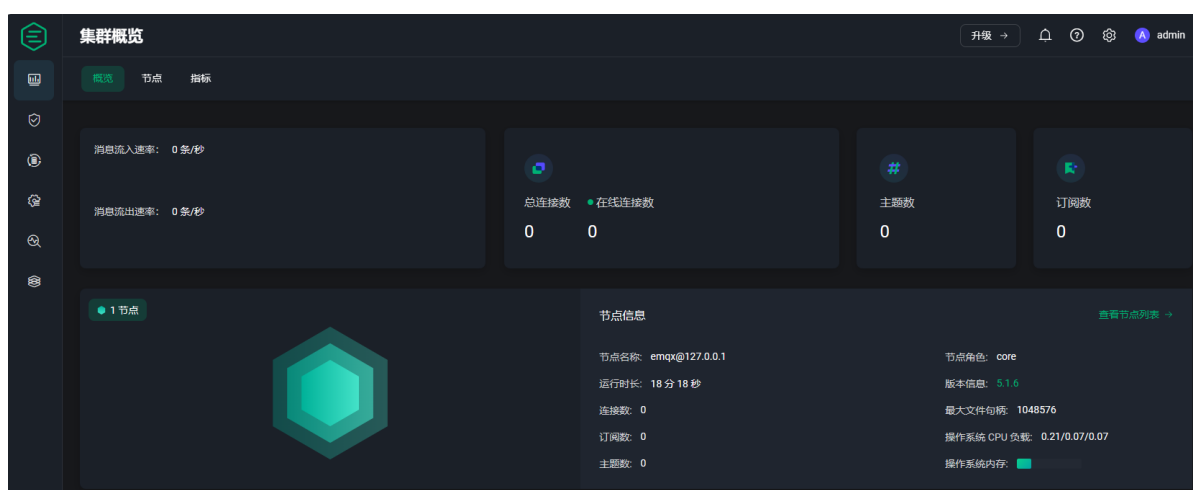


图 4.6 EMQX 服务的管理后台

## 4.2 硬件端搭建

### 4.2.1 NB-IoT 联网

使用 SIM7020 模块进行，选用合适的 NB-IoT 的 SIM 卡，插入 SIM7020 模块中，按照图 4.7 的形式将 SIM7020 模块与 ESP8266 开发板进行连接。

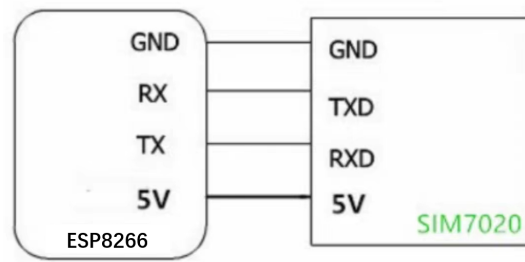


图 4.7 ESP8266 模块与 SIM7020 连接

通过 AT 指令集对 SIM7020 模块进行控制，通过 ESP8266 模块进行 AT 指令集的发送，相关代码内容如图 4.8 所示。

```

SIM7020 sim(&Serial1);
//NB-IOT联网及mqtt服务器连接
void connectnbmqtt(){
    Serial.begin(115200);
    Serial1.begin(115200);
    sim.reboot(true);
    sim.waitReady();
    String resp = sim.execCommand("AT+CMQNEW=\"123.249.197.82\",1883,12000,1024");
    Serial.println(resp);
    resp = sim.execCommand("AT+CMQCON=0,4,\"slb-sim7020\",3600,0,0,\"admin\", \"mqtt@1912\");
    Serial.println(resp);
}
    
```

服务器地址、端口号、超时时间

NB-IOT设备唯一标识      登录用户名和密码

图 4.8 通过 AT 指令集进行 NB-IoT 联网并连接至 MQTT 服务器

```

#define DHTPIN 2 // 设置连接到esp8266的GPIO引脚 其实是D4!!
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

float currentTemperature = 0;
float currentHumidity = 0;
unsigned long currentUnixTime = 0;
//获取温湿度信息
void getTH() {
    delay(3000);
    float RH = dht.readHumidity();
    float T = dht.readTemperature();
    currentTemperature = T;
    currentHumidity = RH;
}
    
```

图 4.9 DHT22 的连接与温湿度获取

### 4.2.2 DHT22 模块安装

DHT22 模块仅包含 VCC、GND 和一个单线的数据 IO 口，将其连接到 ESP8266 上（引脚 4，对应的代码编号为 2）。代码层面，加载 DHT 相关的库，并定义一个 DHT 对象，设置 DHT 类型与 DHT 的 IO 端口号。完成连接后可以通过 DHT 对象的 readHumidity() 和 readTemperature() 函数获取温湿度信息，若温湿度信息异常，则返回 NaN，需要对其异常值进行处理。代码内容如图 4.9 所示。通过 getTH 函数可以获取环境的温湿度，并存储于全局变量中，以供其余功能函数进行数据的获取。

### 4.2.3 MQTT 消息发布

通过代码设置模块对某个主题进行订阅，这里选用的主题名称为“eps8266-1912”，将 messageBuff 中的内容通过窄带物联网发送至 MQTT 服务器的指定主题中，其中 messageBuff 数组中的内容为 getJson 函数生成的，该函数用于将一系列硬件与温湿度信息封装成 Json 格式存储到 messageBuff 之中。代码内容如图 4.10 所示。

```
// 信息发布
void postmsg()
{
    getJson();
    String cmd = "AT+CMQPPUB=0,\"eps8266-1912\",0,0,0,\" + String(strlen(messageBuff)) + "\",\"\" + messageBuff + "\"";
    String resp = sim.execCommand(cmd);
    Serial.println(resp);
    delay(30000);
    memset(messageBuff,0,1024);
}
```

图 4.10 通过 NB-IoT 进行信息订阅与发布

### 4.2.4 WiFi 版本编写

在设计过程中，发现 NB-IoT 相关组件的连接不良，考虑编写一份使用 WiFi 联网的硬件代码，仅与 NB-IoT 版本在联网部分的部分代码有所差异。因此在测试过程中，也使用了基于 WiFi 联网的硬件获取方式。该部分内容主要进行了联网函数 connectWiFi() 函数的编写，其中设置了 WiFi 信息以供 ESP8266 进行循环连接。主函数内容如图 4.11 所示。由于篇幅受限，不对每一个函数的细节进行展示，以注释的形式说明其功能。

### 4.2.5 子功能测试

以上的内容实现了无线节点的温湿度信息获取，并能够通过窄带物联网进行 MQTT 服务器的连接，从而进行数据的发布，为测试该子功能的正确性，考虑使用 MQTTeX 客户端进行同一主题的订阅，并观察获取的数据准确性。如图 4.12 所示，可以看到成功实现了节点环境温度的监测，并以 Json 格式将其发布到了 MQTT 服务器中，且其余订阅该主题模块或软件均能够获取该信息。

```
void setup(){
    Serial.begin(9600);
    //传感器启用
    dht.begin();
    //连接网络
    connectWiFi();
    //连接到NTP，用于时间戳的获取
    timeClient.begin();
    //设置MQTT服务器和端口号
    mqttClient.setServer(mqttServer,mqttPort);
    //连接MQTT服务器
    connectMQTTServer();
}
void loop() {
    //若MQTT客户连接
    if(mqttClient.connected()) {
        //获取温湿度、时间戳
        getTH();
        getUnixTime();
        //串口打印
        dataTest();
        //信息发布
        publishMQTTmsg();
    } else {
        //重连MQTT服务器
        connectMQTTServer();
    }
}
```

图 4.11 WiFi 版本的主函数



图 4.12 数据发布功能的测试



## 4.3 前端设计

### 4.3.1 页面布局

基于表 3.1 中所列的各项功能模块，使用 VSCode 与 Vue.js 前端框架进行前端界面的代码编写，并使用了 ElementUI 辅助进行页面的美化与设计。

网页界面的前端布局大体如图 4.13 所示。包含侧边导航栏，其中包含了系统主页、设备管理、数据分析、历史数据等子组件。

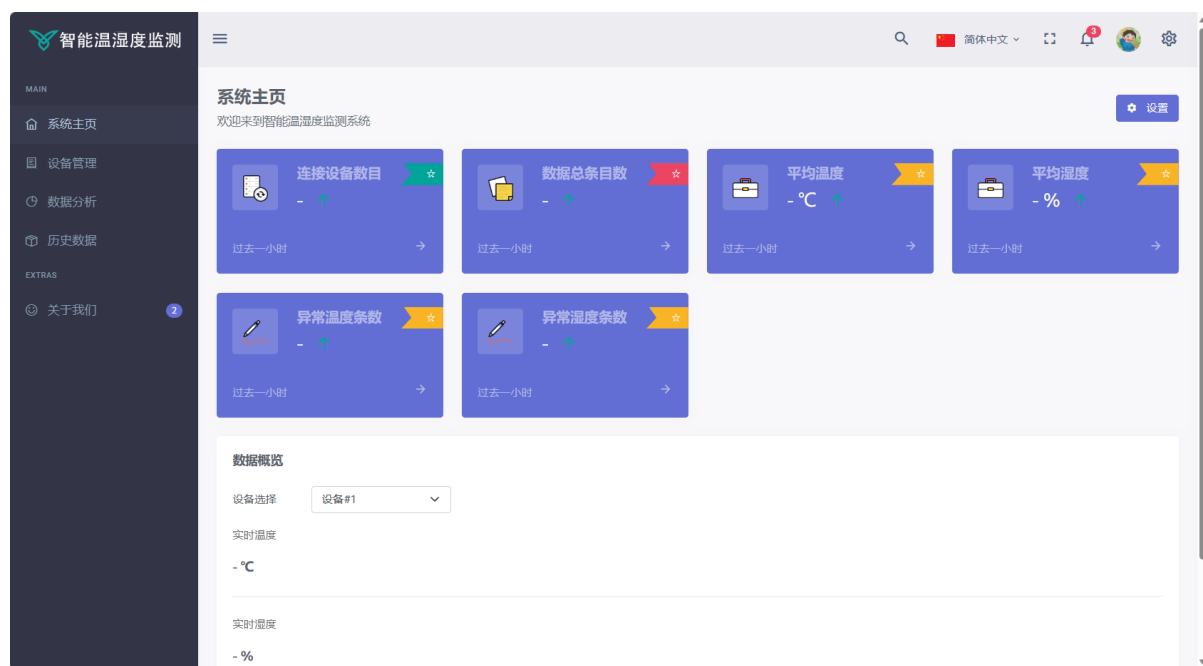


图 4.13 前端网页布局

对于页面的跳转，采用 vue-router 实现，路由信息如下图 4.14 所示，通过导航栏的点击，跳转到不同的路径，从而显示不同的组件内容。

### 4.3.2 数据请求

如图 4.13 所示，数据尚未进行显示，需要前端向后端发送数据请求，并在前端使用 vue 的双向绑定功能进行数据渲染，数据请求层面，使用 axios 进行异步请求。具体地，首先进行 axios 的二次封装，设置请求的地址与端口（后端服务的地址与端口），再进行 api 接口的设置，需要定义请求方式（get、put、post、delete）与请求路径，如图 4.15 所示，为部分数据请求函数。

在各组件中，只需引入相应的数据请求函数，即可通过 axios 向指定的后端发送数据请求，并能够在 request 中获取数据的内容。后续需要在后端中依次提供这类数据的请求接口。



```
{
  path: '/management',
  name: 'Management',
  component: () => import('./views/management/index'),
  meta: {
    authRequired: true,
  },
},
{
  path: '/analysis',
  name: 'Analysis',
  component: () => import('./views/analysis/index'),
  meta: {
    authRequired: true,
  },
},
{
  path: '/histroydata',
  name: 'HistroyData',
  component: () => import('./views/histroy/index'),
  meta: {
    authRequired: true,
  },
},
{
  path: '/abouts',
  name: 'About',
  component: () => import('./views/abouts/index'),
  meta: {
    authRequired: true,
  },
},
},
```

图 4.14 使用 vue-router 实现页面跳转

```
export const getIotCount = () => {
  return http.get('api/getIotCount')
}
export const getAbnormalTempData = () => {
  return http.get('api/abnormaltempdata')
}
export const getAbnormalHumiData = () => {
  return http.get('api/abnormalhumidata')
}
export const editIot = (data) => {
  return http.put('/api/updateIot', data)
}
export const deleteIot = (params) => {
  return http.delete('/api/deleteIot/' + params)
}
export const appendIot = (data) => {
  return http.post('api/addIot', data)
}
```

图 4.15 前端部分数据请求

### 4.3.3 设备管理

在设备管理界面，进行了所有导入系统的无线节点信息的显示，包括唯一标识的设备 ID、设备型号、设备状态、发送数据流、温度阈值上下限、湿度阈值上下限、设备描述等，如图 4.16 所示。同时对每一个设备信息可以进行编辑、删除操作，在此界面中进行设备的添加操作，需注意设备 ID 需要非重复，系统将会自动监测是否存在该 ID 的设备发送的信息，若有则进行存储与整合，并实施渲染到前端以提示用户。

对于该部分的实现，主要采用 ElementUI 组件进行，数据主要来自于 hardware 表，其中发送数据量的值筛选自 record 表。



设备ID	设备型号	设备状态	发送数据量	温度报警上限	温度报警下限	湿度报警上限	湿度报警下限	设备描述	修改	删除
1	ESP8266	已连接	937	30	-10	74.2	20	库房1监测	修改	删除
2	ESP8266	已断开	798	40	10	70	20	库房2监测	修改	删除
3	ESP8266	已断开	0	30	-10	80	50	库房3监测	修改	删除

图 4.16 设备管理界面设计



图 4.17 数据分析界面设计

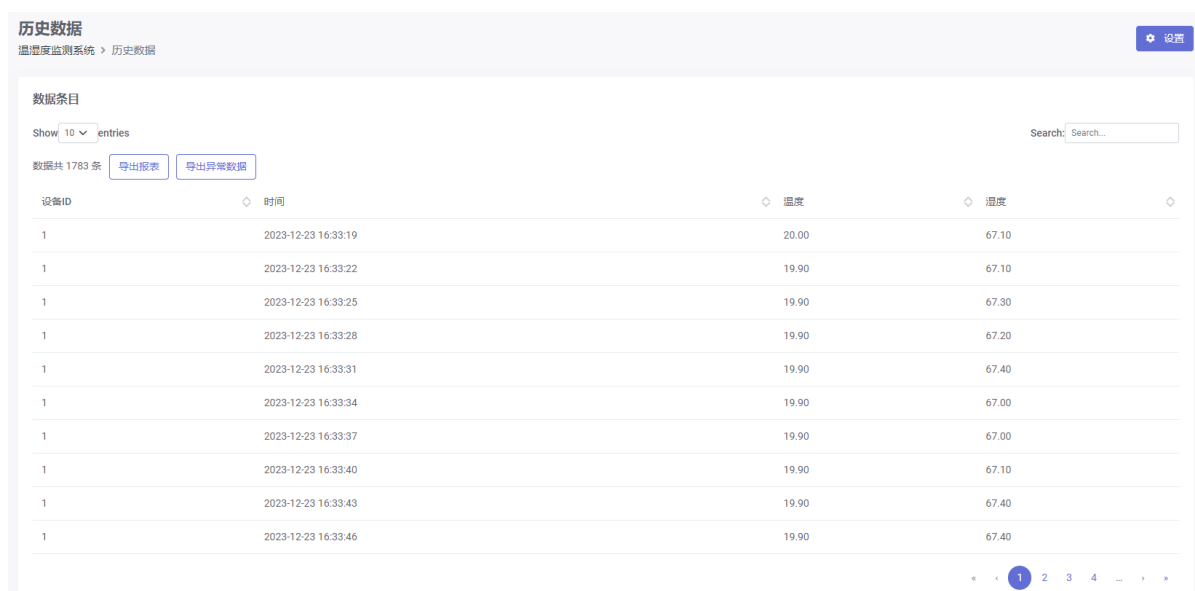
### 4.3.4 数据分析

在数据分析界面，以动态报表的形式呈现所有正在工作的监测节点处的温湿度信息，并在图线中给出阈值范围，使得用户能够清晰地观察到温湿度所处的范围，如图

4.17 所示。具体地，该部分采用 echarts 组件进行报表的呈现，前端每个一段时间（1s）向后端发送数据请求，查询数据库中新增的数据条目，并添加到显示队列即可实现报表的动态更新，且请求到数据后需要进行数据的筛选，以实现多个设备之间的工作独立性。

### 4.3.5 历史数据

在该界面中实现的内容较为简单，即将所有 records 表中的数据以表格的形式在前端进行呈现，并提供排序和筛选操作，使用户可以按照时间或者设备 ID 等信息查询指定时间段的数据信息，此外提供了报表导出与异常数据导出的功能，能够实现所有数据或异常数据的 excel 表格导出，方便后续的数据处理。该部分界面设计效果如图 4.18 所示。



历史数据

温湿度监测系统 > 历史数据

数据条目

Show 10 entries

数据共 1789 条

导出报表 导出异常数据

设备ID	时间	温度	湿度
1	2023-12-23 16:33:19	20.00	67.10
1	2023-12-23 16:33:22	19.90	67.10
1	2023-12-23 16:33:25	19.90	67.30
1	2023-12-23 16:33:28	19.90	67.20
1	2023-12-23 16:33:31	19.90	67.40
1	2023-12-23 16:33:34	19.90	67.00
1	2023-12-23 16:33:37	19.90	67.00
1	2023-12-23 16:33:40	19.90	67.10
1	2023-12-23 16:33:43	19.90	67.40
1	2023-12-23 16:33:46	19.90	67.40

1 2 3 4

图 4.18 历史数据界面设计

## 4.4 后端开发

前端的待渲染数据通过 axios 向后端请求，因此后端程序需要提供合理的数据请求接口，具体地，后端程序需要实现数据库的连接，以在获取温湿度信息后进行数据的转存，同时也需要进行 MQTT 服务器的连接与主题订阅，以消费者的形式进行消息的获取。

### 4.4.1 数据库的连接

由于后端框架采用 springboot 实现，需要在资源文件中进行相关配置的设置（application.yml），包括对数据库的信息配置，在 springboot 项目的初始化时已经选用添

加了 MySQL Driver 依赖。由于数据库来自于云服务器，根据 4.1.2 节中的数据库配置，在此需要设置 IP 为云服务器的公网 IP，并设置数据库名为 mysql\_mqtt，连接到 root 用户，并给定密码，如图 4.19 所示。

在完成配置后 springboot 程序将在运行时连接至该数据库，且后续 Mapper 层中的 SQL 语句将被发送到该数据库中执行，以实现数据的存储与查询。

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://123.2 82:3306/mysql_mqtt?userSSL=false
    username: root
    password: Mqtt@1912
```

图 4.19 后端程序连接数据库

#### 4.4.2 MQTT 服务器的配置

若想要 springboot 能够连接 MQTT 服务器，首先需要导入 Maven 依赖，具体的依赖名为：spring-integration-mqtt，重新加载 Maven 依赖后即可实现相关资源的加载。

与数据库连接类似，MQTT 服务器的连接也需要在 application.yml 文件中进行相关配置的设置，包括服务器所在的 IP 与服务提供端口，对于 MQTT 服务端口，在 TCP 协议下均默认为 1883，此外还需要设定连接的 id 信息，该 id 需要保证在所有连接到目标服务器的客户端中非重复，最后需要设置订阅的主题。具体配置如下图 4.20 所示。

需注意的是，本次课程设计过程中搭建的 MQTT 服务器并未进行节点的认证操作，因此这里的用户名和密码随机指定即可，服务器端并未对此进行认证。

```
mqtt:
  url: tcp://123.2 82:1883
  username: springboot
  password: 123456
  client:
    id: springboot_f07be17a_196825
  default:
    topic: eps8266-1912
```

图 4.20 后端程序连接 MQTT 服务器的相关配置

#### 4.4.3 服务器连接与数据获取

完成 MQTT 服务器的基本配置后，需要编写相关函数实现服务器的连接、主题订阅、消息监听与获取等操作，考虑设置一个 MQTT 消费者的控制类 MqttConsumer-

Comfig, 提供一些基本的连接、重连函数。

在加载 spring-integration-mqtt 依赖后, 可以导入 MqttClient 类, 该类的对象实现了一个虚拟的 MQTT 客户, 在实例化对象时, 需要设置目标 MQTT 服务器的 IP 与端口信息, 同时给定唯一标识的客户 id, 这类信息可以在图 4.20 中的配置信息中获取, 可以通过 springboot 的 @Configuration 与 @Value 注解将配置信息加载到类中的私有变量中, 以供类中函数的调用。

对于连接 MQTT 服务器的具体配置, 可以通过 MqttConnectOptions 对象进行设置, 包括用户名、密码、超时时长、保活时间、断开连接后的处理, 最后通过 MqttClient 对象的 connect 函数进行配置的设置与连接操作执行即可完成连接。连接成功后再通过 subscribe 函数进行主题的订阅。

除此之外, 最为重要的操作即为对 MqttClient 对象进行回调函数的设置, 该依赖提供了完整的回调函数的设置格式, 抽象于 MqttCallback 接口中, 只需要实现其中的三个抽象函数即可, 分别为 connectionLost (用于连接丢失后的处理操作设置)、messageArrived (获取信息后的处理)、deliveryComplete (完成发送后的后续处理)。对于 deliveryComplete 函数, 本次课程设置无需进行实现; 对于 connectionLost 函数, 通过 MqttClient 对象的 isConnected 函数获取连接状态, 并调用 reconnect 函数进行重连, 或者直接重新执行 connect 函数。

对于 messageArrived 函数, 在主题中有新消息时, 会以 MqttMessage 对象的形式将数据传入该回调函数, MqttMessage 对象提供了 getPayload 函数来获取其中加载的信息内容字符串, 随后通过一系列 Java 操作将其转换为 Json 格式并获取图 4.12 中所示的所有信息。

在获取到所需的所有的数据后, 考虑将数据进行封装并存入数据库, 这里使用了 springboot 的三层架构, 首先设置了实体类 Record, 成员变量与数据表 record 中的字段一一对应, 以用于数据的封装, 后续定义 Service 层与 Mapper 层, 在 Service 层中对外提供插入数据的服务, 以 insertRecord 函数的形式呈现, 具体在 Mapper 层中通过 SQL 语句进行数据的插入。SQL 语句即为: insert into record (froms ,timestamp,temperature,humidty) values (froms ,timestamp,temperature,humidty), 其中 values 中的数据来自于封装好的 Record 对象。

综上完成了 MQTT 服务器的连接、主题的订阅、消息的监听、数据的存储操作。核心代码如下:

```
public void connect() {
    try {
        //MQTT客户对象
        client = new MqttClient(hostUrl,clientId,new MemoryPersistence());
        //连接配置信息
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(false);
```

```

options.setUserName(username);
options.setPassword(password.toCharArray());
options.setConnectionTimeout(100);
options.setKeepAliveInterval(20);
options.setWill("willTopic",(clientId +
    "与服务器断开连接").getBytes(),0,false);
//重写回调函数内容
client.setCallback(new MqttCallback() {
    @Override //重连的处理方法
    public void connectionLost(Throwable throwable) {
        System.out.println("MQTT连接断开，发起重连.....");
        try {
            if(client!=null && !client.isConnected()) {
                client.reconnect();
                System.out.println("尝试重新连接");
            } else {
                client.connect(options);
                System.out.println("尝试建立新连接");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @Override //消息获取后的处理方法
    public void messageArrived(String topic, MqttMessage message) throws
        Exception {
        String msg = new String(message.getPayload());
        ObjectMapper objectMapper = new ObjectMapper();
        Map<String,Object> map = objectMapper.readValue(msg, Map.class);
        //获取消息中的关键信息
        String Timestamp = String.valueOf(map.get("Timestamp"));
        String from = String.valueOf(map.get("ID"));
        String Humidity = String.valueOf(map.get("Humidity"));
        String Temperature = String.valueOf(map.get("Temperature"));
        //信息封装
        Record record = new Record();
        record.setFroms(from);
        record.setTimestamp(Timestamp);
        record.setTemperature(Temperature);
        record.setHumidity(Humidity);
        //处理异常数据
    }
}

```

```
        if(Long.parseLong(Timestamp)<100) return;
        //调用服务以插入数据
        recordService.insertRecord(record);
    }
    @Override
    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {}
});
//设置连接配置，随后发送连接
client.connect(options);
//订阅主题
client.subscribe(defaultTopic,0);
} catch (MqttException e) {
    e.printStackTrace();
}
}
```

运行 springboot 程序，同时进入到 MQTT 服务器的后端（18083 端口），可以查看到 springboot 程序已连接至 MQTT 服务器，并显示了图 4.20 中指定的用户名和唯一标识的客户 ID，如图 4.21 所示。至此实现了服务器的连接以及消息主题的订阅。

客户端 ID	用户名	状态	IP 地址
CENSYS		未连接	167.248.133.53:53456
mqttx_f07be17a		已连接	223.104.161.252:5242
springboot_f07be17a_196...	springboot	已连接	123.218.17.88:16078

图 4.21 springboot 连接 MQTT

#### 4.4.4 接口设计

该部分实现后端对前端提供的各类 api 接口的代码编写，采用 springboot 的三层架构，在 Controller 层提供对外的请求接口。以增添设备操作为例，需要前端发送 post 请求，后端层面通过 @PostMapping 注解设置请求路径为 '/addIot'，并在函数参数处通过 @RequestBody 获取请求数据，即为待插入的 IoT 信息，随后调用 espService 中的 insertIot 服务，将待插入的对象传入，在其中继续调用 Mapper 层的特定函数，最后在 Mapper 层使用 SQL 语句进行数据插入即可。

需要注意的是，插入操作需要保证完整性依赖，即设备的 ID 不能重复，否则数据

库层面会拒绝插入操作并以 Error 的形式返回后端，故后端代码需要使用 try-catch 语句捕获该异常，并向前端返回异常代码，若插入正常则返回成功代码。

该部分的代码如下。以此为例实现了前端所需的所有请求接口，完成了所有的前后端互通操作。

```
@PostMapping("/addIot")
public Result add(@RequestBody EspIoT iot) {
    try {
        espService.insertIot(iot);
    } catch (Exception e) {
        if(e instanceof DuplicateKeyException) {
            return Result.error("Insert Wrong");
        } else {
            return Result.error("System Wrong");
        }
    }
    return Result.success();
}
```

## 五、 系统测试

### 5.1 系统部署

代码编写时均在本地进行测试运行，完成了系统的开发后，为方便测试系统功能的完整性与可用性，将前后端项目部署到云服务器中，该云服务器即为 MySQL 数据库和 MQTT 服务器所在的 ECS，更改前端中 axios 的默认请求 IP 为 127.0.0.1:9090 或者具体的 ECS 公网 IP，若使用 localhost 则无法被解析。打包前端项目生成 dist 文件，打包后端项目生成 jar 文件，将其上传至云服务器之中，配置 nginx 路径于该 dist 文件中，并通过 nohup 命令在后台运行该 jar 文件，即可实现使用公网 IP 访问系统。

后续的功能测试均在该部署完成的温湿度监测系统上进行。

### 5.2 功能测试

该部分将对各功能模块进行效果呈现。

#### 5.2.1 主页视图

主页视图如图 5.1 所示。其中显示了连接设备数目、数据总条目数、平均温度、平均湿度、异常温度条数、异常湿度条数的概览信息，同时也提供了各设备实时温度的数



值与简易视图，便于进行一些简单信息的查阅。

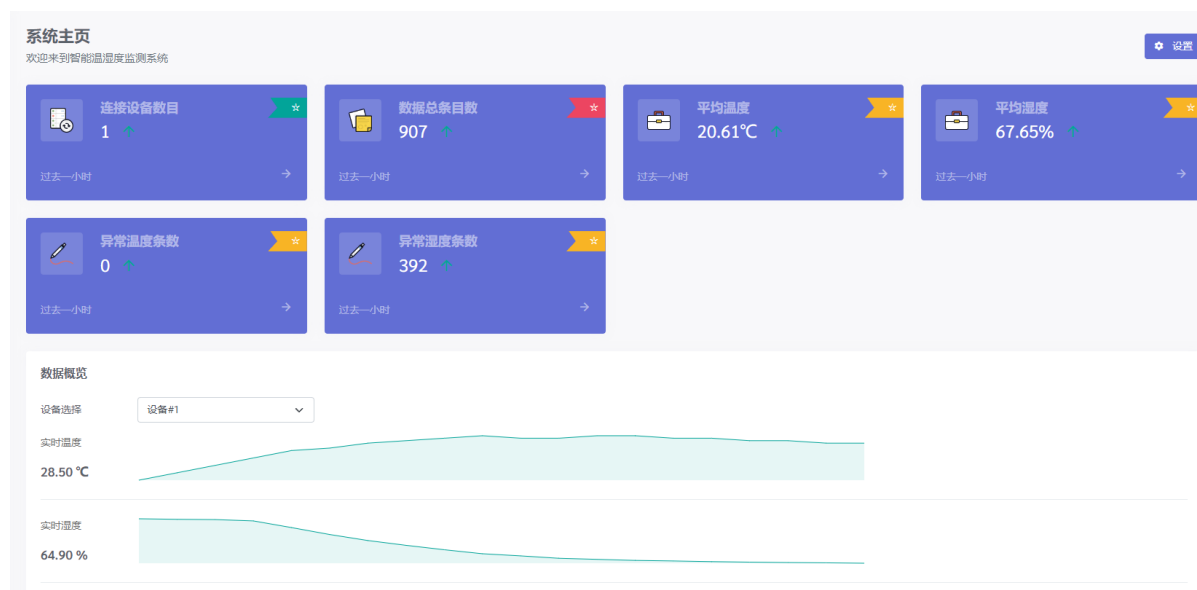


图 5.1 系统主页视图效果展示

## 5.2.2 动态视图

数据分析模块中进行了动态视图的显示，由于环境室温较为稳定，难以看出视图的动态性，因此考虑变化硬件节点的所处环境，可以看到数据发生了明显变化，但略延迟于实际时间点约 1s 左右，这部分延迟来自于传感节点的信息发布周期以及信息发布、获取所消耗的时间。视图效果如图 5.2 所示。

除此之外可以看到视图中展示了阈值范围与平均温度线，便于进行数据的监测与进一步的分析。

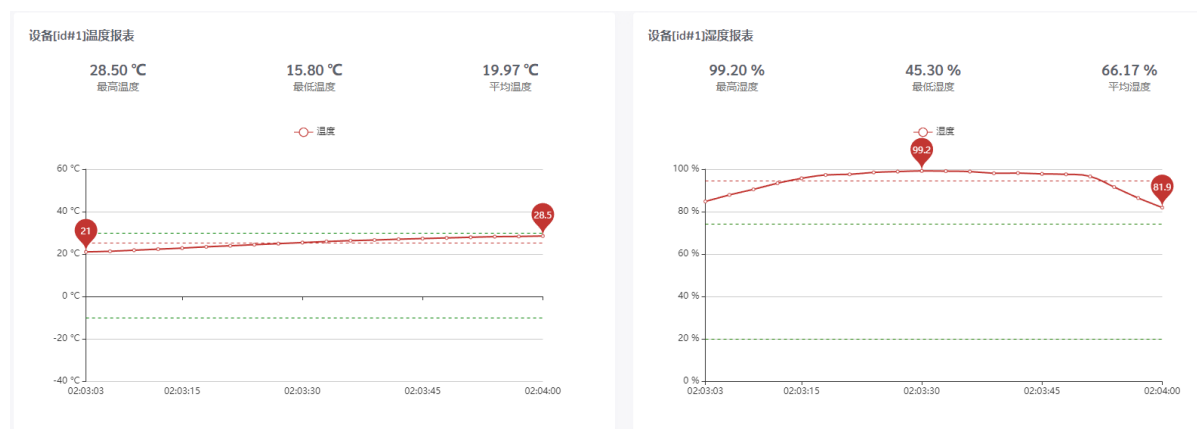


图 5.2 动态视图效果展示

### 5.2.3 报表导出

对于历史数据模块，呈现了所有存储于数据库中的温湿度信息，并记录了其来源设备、具体时间等信息，同时也告知用户总共的数据条数，该数据与系统主页中呈现的一致。点击”导出报表“按钮，系统会自动下载一个 Excel 文件，其中包含了所有的温湿度信息，如图 5.3 所示；点击”导出异常数据“按钮，系统会自动下载一个额外的 Excel 文件，其中内容如图 5.4 所示，Excel 中包含了具体的时间、来源设备、温湿度信息、实际状态与正常的温湿度范围，以供使用者进行异常分析的参考。

	A	B	C	D
1	设备ID	时间	温度	湿度
2	1	2023-12-25 14:19:55	16.00	48.40
3	1	2023-12-25 14:19:58	15.90	48.30
4	1	2023-12-25 14:20:1	15.90	48.30
5	1	2023-12-25 14:20:4	15.90	48.40
6	1	2023-12-25 14:20:7	15.90	48.20
7	1	2023-12-25 14:20:10	15.90	48.40
8	1	2023-12-25 14:20:13	15.90	48.50
9	1	2023-12-25 14:20:16	15.90	48.20
10	1	2023-12-25 14:20:19	15.80	48.30
11	1	2023-12-25 14:20:22	15.80	48.40
12	1	2023-12-25 14:20:25	15.90	48.20
13	1	2023-12-25 14:20:28	15.90	48.40
14	1	2023-12-25 14:20:31	15.80	48.30
15	1	2023-12-25 14:20:34	15.80	48.30

图 5.3 所有数据信息的报表导出

设备ID	时间	温度	湿度	状态	正常温度范围	正常湿度范围
1	2023-12-25 14:23:0	17.30	74.80	湿度异常	-10~30	20~74.2
1	2023-12-25 14:23:3	17.70	80.20	湿度异常	-10~30	20~74.2
1	2023-12-25 14:23:6	18.10	82.10	湿度异常	-10~30	20~74.2
1	2023-12-25 14:23:9	18.60	79.70	湿度异常	-10~30	20~74.2
1	2023-12-25 14:23:12	18.90	74.50	湿度异常	-10~30	20~74.2
1	2023-12-26 12:17:31	20.40	74.30	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:1	20.50	74.30	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:4	20.50	75.00	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:7	20.40	75.40	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:10	20.50	75.50	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:13	20.50	75.30	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:16	20.40	75.30	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:19	20.50	74.80	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:22	20.50	74.90	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:25	20.50	74.50	湿度异常	-10~30	20~74.2
1	2023-12-26 12:18:32	20.50	74.30	湿度异常	-10~30	20~74.2

图 5.4 异常数据的导出

### 5.2.4 设备管理

设备管理模块中，由用户自行进行设备的添加，同时也可以进行基本的设备信息修改与删除操作，其中设备的修改界面如图 5.5 所示，其中设备 ID、设备型号、设备状态、数据条数等内容不允许进行修改，用户仅能够进行设备描述的内容修改以及温湿度阈值上下限的修改。

当完成对上下限修改后，该设备所对应的动态报表中的阈值返回以及异常判断标准也会随着发生改变。需注意设备 ID 会与实际硬件烧录中的 ID 信息一一对应。

提示

×

设备ID

1

设备型号

ESP8266

设备状态

已连接

数据条数

694

\* 温度下限

-10

\* 温度上限

30

\* 湿度下限

20

\* 湿度上限

74.2

设备描述

库房1监测

取消

确定

图 5.5 设备的信息修改

## 5.3 效果分析

通过一系列的功能测试，效果符合需求分析中的各项功能要求，由于 MQTT 服务器为自我搭建且与后端代码、数据库等部件位于同一台云服务器中，运行效率较高，且硬件通过 NB-IoT 进行联网，信息传输稳定且受地域干扰较小。本次课程设计的开发在功能实现层面体现了一定的复杂性与较高的可用性，符合预期。

## 六、心得与体会

在这次的课程设计中，我们团队获得了丰富的物联网开发经验，取得了一系列可喜的成果。首先，通过对 ESP8266 的编程、ArduinoIDE 的灵活运用以及各类传感器模块的巧妙连接，我们深刻理解并掌握了物联网设备的开发技巧。这使得我们能够高效地实现温湿度监测系统的硬件层面，充分发挥了各种传感器的功能，为系统的数据采集提供了可靠的基础。

无线监测节点的设计为本次课设设计的重点，这一部分工作涉及到 ESP8266、SIM7020 模块以及 DHT22 模块的协同工作，为系统的 NB-IoT 联网和 MQTT 服务器连接提供了可靠的基础。

首先，通过 ESP8266 的灵活编程，我们成功实现了监测节点的智能控制和数据处理功能。ESP8266 作为微控制器，不仅令我们熟悉了硬件编程的精要，还为监测节点的智能化提供了可靠支持。其次，我们选用 SIM7020 模块，通过它实现了 NB-IoT 的联网操作。这一步骤是整个监测系统的关键，使得监测节点能够远程连接至云端，实现了数据的远程传输。SIM7020 模块的低功耗、高效性能为我们的系统提供了可靠的网络通信能力。同时，我们结合 DHT22 模块，实现了对环境温湿度信息的高精度采集。DHT22 模块通过数字信号输出，为监测系统提供了可靠的环境数据，使得我们的智能监测节点能够更全面地感知环境的变化。最终，通过 MQTT 服务器的连接，我们成功将监测节点获取的数据整合并发送至云端。这不仅实现了监测功能，也为整个系统提供了远程信息采集的强大功能。通过这一设计，我们深刻体会到了无线监测节点在智能系统中的关键作用，以及其对于物联网应用的不可或缺性。

在软件方面，我们成功地熟悉和掌握了前后端的开发技巧。通过使用 Visual Studio Code 和 ArduinoIDE，我们能够迅速进行前端代码的编写和硬件代码的烧录，确保系统的复杂功能能够被可靠实现。与此同时，对于后端的开发，我们通过使用 IDEA，成功搭建了一个功能强大的系统后台，实现了数据的处理、存储和管理。这为系统的稳健性和可用性提供了有力的支持。

服务器的搭建是我们团队在这次设计中的一项关键任务。我们成功地配置了云服务器，安装了数据库，并搭建了 MQTT 服务器进行消息的汇总和传递。这一系列操作不仅加深了我们对服务器架构的理解，也为系统的远程通信提供了坚实的基础。通过这一过程，我们对于云计算和物联网的结合有了更深刻的认识。

总的来说，这次课程设计不仅让我们掌握了丰富的技术知识，更锻炼了我们解决问题的能力。我们学会了发现并解决各种工程开发中可能遇到的问题，培养了团队协作和沟通的能力。最终，我们能够顺利完成课程设计的要求，这不仅是对我们所学知识的巩固，也是对我们工程开发能力的一次全面提升。这次设计不仅是知识的积累，更是团队协作和实践的宝贵经验。

## 参考文献

- [1] 苗君臣, 王睿昊, 尤达等. 基于物联网的智能农业温室大棚监测系统 [J]. 物联网技术, 2023, 13(10): 16-18. DOI: 10.16667/j.issn.2095-1302.2023.10.004.
- [2] 赵学作, 刘敏, 薛艳茹等. 基于树莓派的温湿度系统设计 [J]. 网络安全和信息化, 2022(04): 94-97.
- [3] 杨海龙, 寇健, 温晓东等. 基于 ESP8266 的智能建筑温湿度检测系统设计 [J]. 河北建筑工程学院学报, 2023, 41(03): 177-181+188.
- [4] 胡浩鸣, 张胜利, 赵思等. 基于 ESP8266 的地窖环境监测系统设计 [J]. 现代信息技术, 2023, 7(22): 150-155. DOI: 10.19850/j.cnki.2096-4706.2023.22.033.
- [5] 饶云. 基于 NBIOT 的工业环境监测平台设计与实现 [D]. 武汉大学, 2021. DOI: 10.27727/d.cnki.gwhxc.2019.000092.
- [6] 徐毅. 基于 NBIOT 技术的环境监测系统的实验设计与实现 [J]. 科技创新与应用, 2020(34): 29-30.
- [7] 王伟. 基于 NBIOT 的冷链运输温湿度监控管理系统设计与实现 [J]. 科技视界, 2022(08): 34-36. DOI: 10.19694/j.cnki.issn2095-2457.2022.08.08.
- [8] 王宇健. NBIOT 技术在工业环境监测中的应用 [J]. 现代工业经济和信息化, 2022, 12(10): 96-97+99. DOI: 10.16525/j.cnki.14-1362/n.2022.10.040.