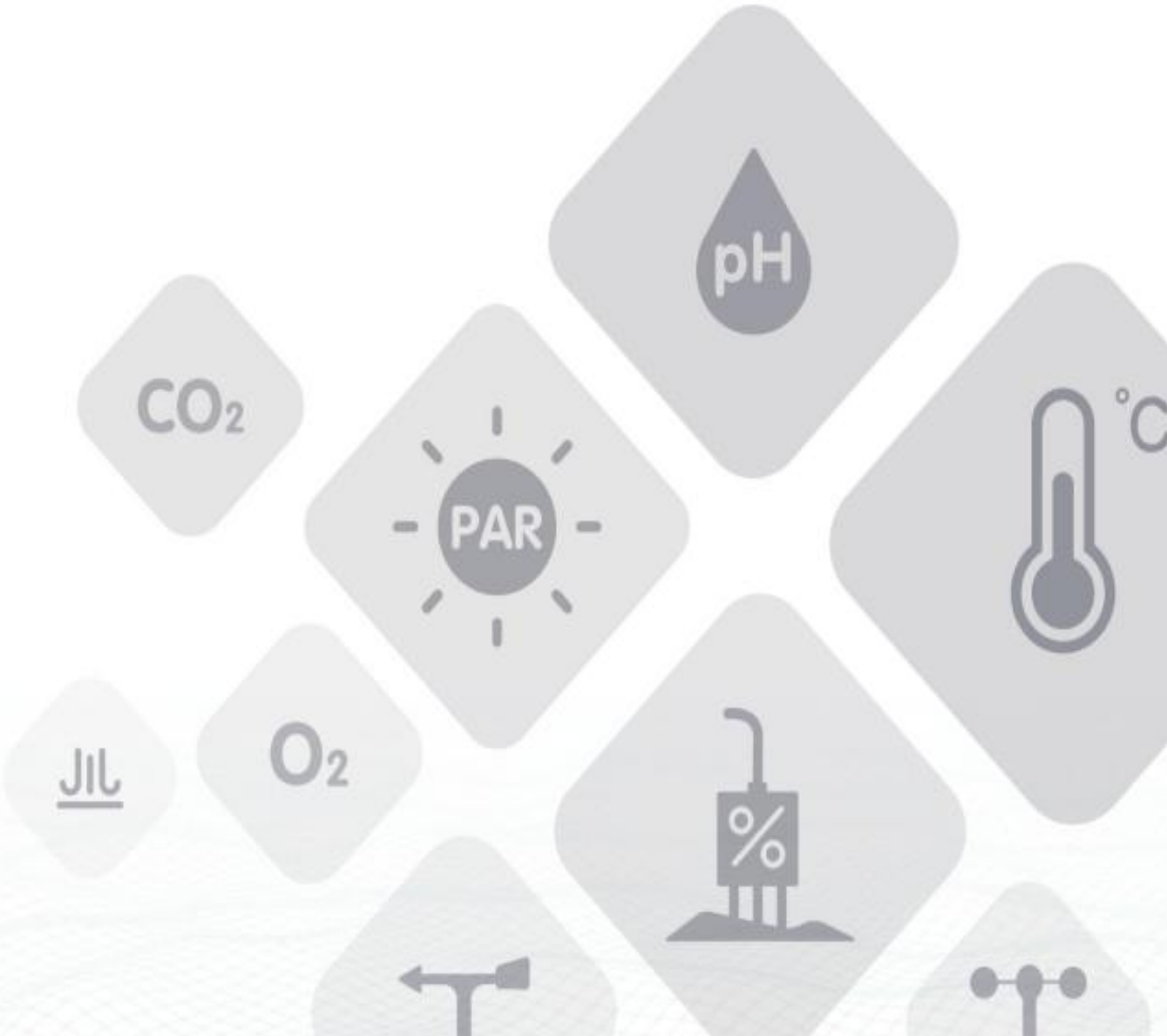
seeed studio

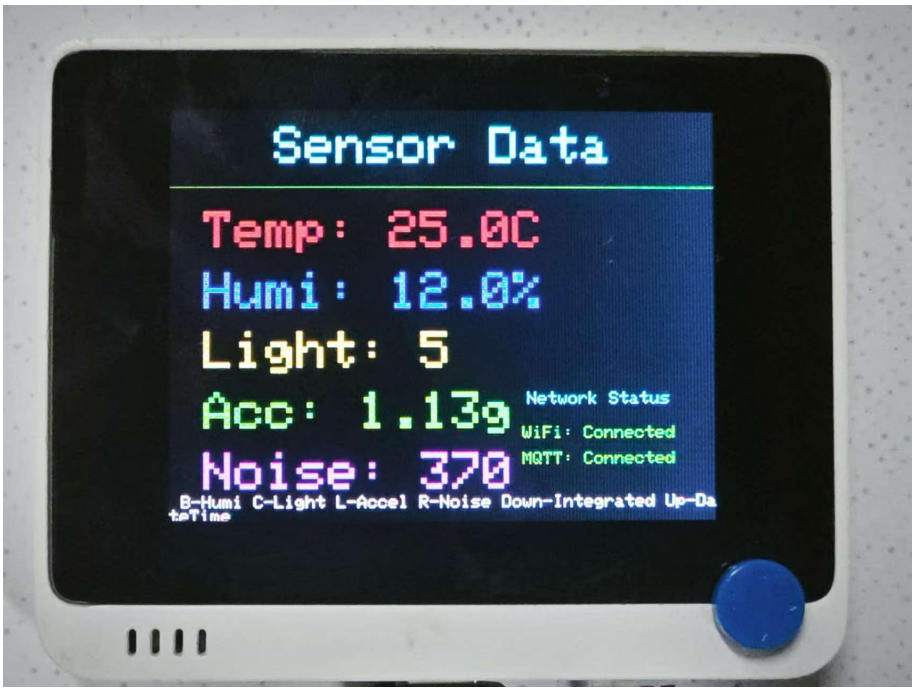
多功能环境监测仪

用户手册



目　录

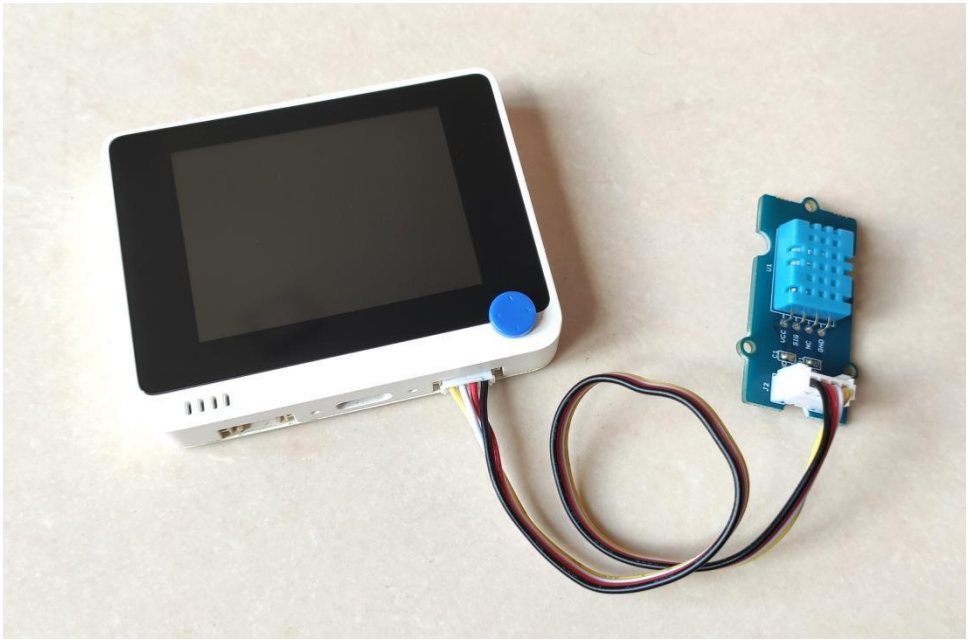
1. 产品介绍



此设计为基于 Wio 终端的多功能环境检测仪。它集成了多个传感器（温湿度、光照、麦克风、加速度）来实时监控环境参数，并通过 WiFi 将数据上传到 MQTT 服务器。系统具有本地显示屏，可以切换不同界面查看数据，并且在检测到异常（如温度超标、设备移动等）时触发声光报警。同时，系统支持 NTP 时间同步，提供日期时间显示。用户可以通过物理按键与设备交互，例如切换显示界面或关闭报警。

应用十分广泛，如温室大棚环境监测、粮仓温湿度监控、野外种植基地防盗系统、育苗室光照管理。通过多传感器融合与实时告警机制，实现农业生产环境的无人化智能监护。

1.1. 产品清单



- Wio 终端：https://wiki.seeedstudio.com/Wio_Terminal_Intro/
- Grove-温湿度传感器(DHT11)：

  https://wiki.seeedstudio.com/Grove-TemperatureAndHumidity_Sensor/
- 其他：双向端子连接线（可参考产品介绍图）、USB-C 电缆；
- 软件工具：MQTTX：https://mqttx.app/zh/features

  Arduino IDE：https://www.arduino.cc/en/software/#ide

  注：使用前请先准备好以上硬件设备及软件工具；

2. 产品规格

2.1. Wio 终端

核心配置	主控芯片	微控制器： ARM® Cortex®-M4
		主频： 120 MHz
		存储： 192KB SRAM + 4MB 外部闪存

	显示屏	尺寸：2.4 英寸
		分辨率：320×240 像素
无线连接	Wi-Fi	RTL8720DN 模块 支持 2.4GHz / 5GHz 双频 协议：802.11 a/b/g/n
	BLE	BLE 5.0（兼容 4.2）
传感器与输入	内置传感器	光传感器（LTR-553ALS） 6 轴 IMU（LSM6DS3TR-C，加速度计+陀螺仪） 红外发射器（IR 940nm）
	交互控件	5 向摇杆（方向+按下） 2 个可编程按键（侧面） 复位/电源按钮
扩展接口	Grove 生态系统	1× Grove I²C 接口（背面） 1× Grove 数字/模拟接口（底部）
	GPIO 引脚	兼容 Raspberry Pi 40-pin（支持 UART/I2C/SPI/ADC）
	其他接口	USB Type-C（供电/编程） MicroSD 卡槽（最大支持 32GB） 2.4G 天线接口（外接天线）
电源	输入	USB Type-C（5V/2A）
	电池扩展	支持锂电池（通过底部接口充电）
开发支持	编程环境	Arduino IDE / PlatformIO MicroPython / CircuitPython

	兼容框架	Arduino、TinyML、Edge Impulse 等
物理规格	尺寸	72mm * 57mm * 12mm
	外壳	ABS + PC

2.2. Grove-温湿度传感器（DHT11）

温度	测量范围	0°C ~50°C
	精度	±2°C
	分辨率	0.1°C
湿度	测量范围	20% ~ 90% RH
	精度	±5% RH
	分辨率	0.1% RH
采样率		1 Hz (每秒采样 1 次)
接口	类型	Grove 4-pin 标准接口
	协议	数字信号 (单总线协议)
	引脚	VCC, GND, NC (未连接), SIG (信号)
供电	电压电流	5V/2A
规格	尺寸	40mm x 20mm x 12mm
	重量	约 4.5 克

2.3. 软件工具

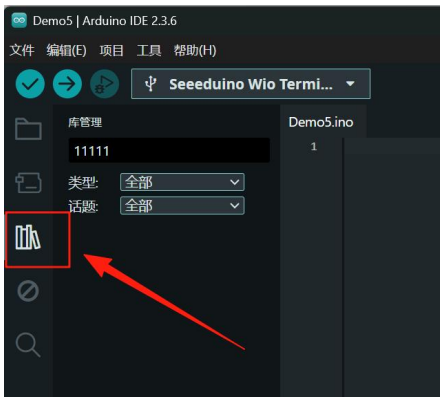
下载 MQTTX 和 Arduino IDE 第三方工具（下载链接见产品清单）

MQTTX：用于订阅 Wio 终端主题查看上行数据及下行指令；

Arduino IDE：用于给 Wio 终端烧录编程代码；

3. 操作部署流程

1) 准备所需软硬件设备（Wio 终端、Grove-温湿度传感器(DHT11)、双向端子连接线、USB-C 电缆、MQTTX、Arduino IDE）；
2) 将 Grove-温湿度传感器(DHT11)连接到 Wio 终端的 D0 端口（五向开关下方端口）；
3) 通过 USB-C 电缆将 Wio 终端连接到 PC；
4) 打开 Arduino IDE，点击如图所示图标，进入“库管理”，或者可点击“项目—导入库—管理库”进入“库管理”；



5) 安装运行代码所需的所有库：

TFT_eSPI

功能：驱动 Wio Terminal 的 TFT 显示屏

安装方式：Arduino 库管理器搜索 TFT_eSPI（作者：Bodmer）

LIS3DHTR

功能：驱动 LIS3DHTR 加速度传感器

安装方式：搜索 LIS3DHTR（作者：Seeed Studio）

DHT Sensor Library

功能：读取 DHT11 温湿度传感器数据

安装方式：搜索 DHT sensor library（作者：Adafruit）

注意：需同时安装 Adafruit Unified Sensor 库

PubSubClient

功能：MQTT 客户端通信

安装方式：搜索 PubSubClient（作者：Nick O'Leary）

NTPClient

功能：从 NTP 服务器同步时间

安装方式：搜索 NTPClient（作者：Fabrice Weinberg）

WiFi (内置)

功能：Wio Terminal 的 WiFi 连接（通过 rpcWiFi.h）

说明：包含在 Seeed Studio SAMD Boards 板支持包中，无需单独安装

WiFiUDP (内置)

功能：UDP 协议支持（NTP 依赖）

说明：Arduino 核心库，无需单独安装

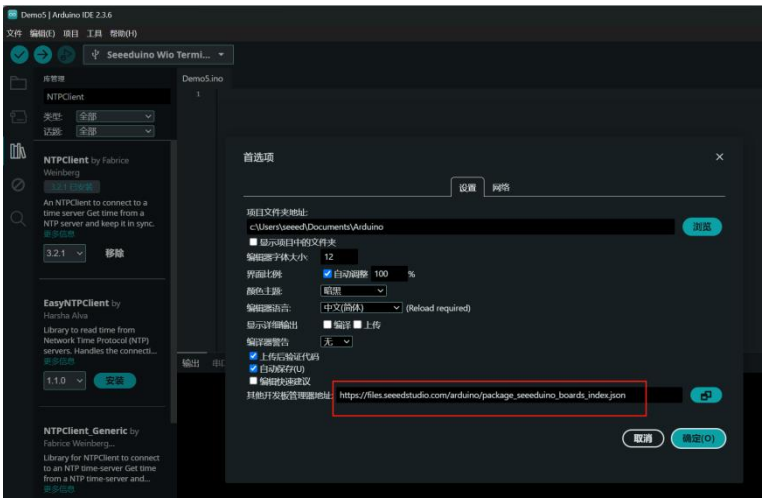
6) 安装步骤：

- 安装板支持包

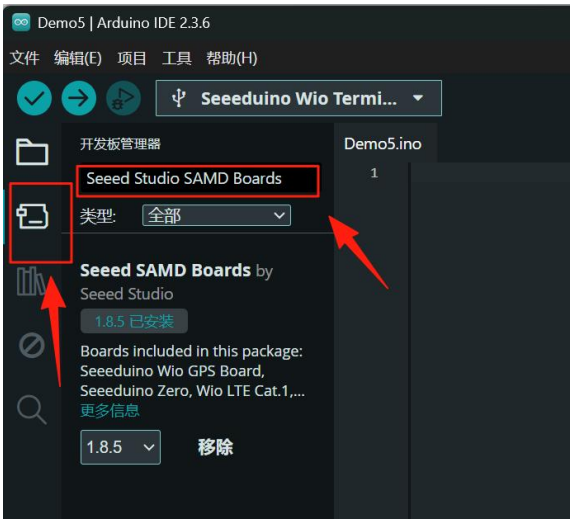
Wio Terminal 需要 Seeed Studio SAMD Boards 支持包：

① Arduino IDE → 文件 → 首选项 → 附加开发板管理器网址 添加：

https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json

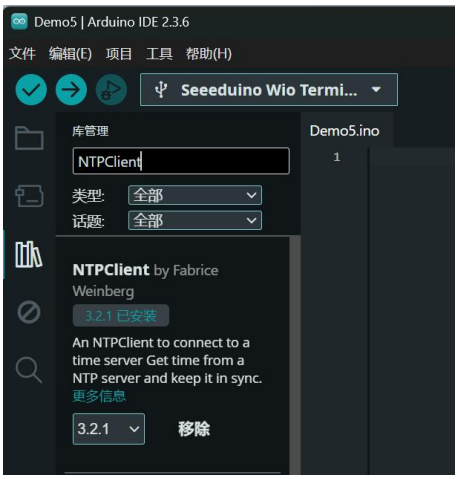


② 工具 → 开发板 → 开发板管理器 → 搜索 Seeed Studio SAMD Boards → 安装最新版。



- 通过库管理器安装依赖库

① 打开 Arduino IDE → 项目 → 加载库 → 管理库...



② 搜索并安装以下库：

TFT_eSPI

LIS3DHTR

DHT sensor library（同时安装 Adafruit Unified Sensor）

PubSubClient

NTPClient

- 验证库安装

① 选择开发板：

工具 → 开发板 → Seeed Studio SAMD Boards → Wio Terminal

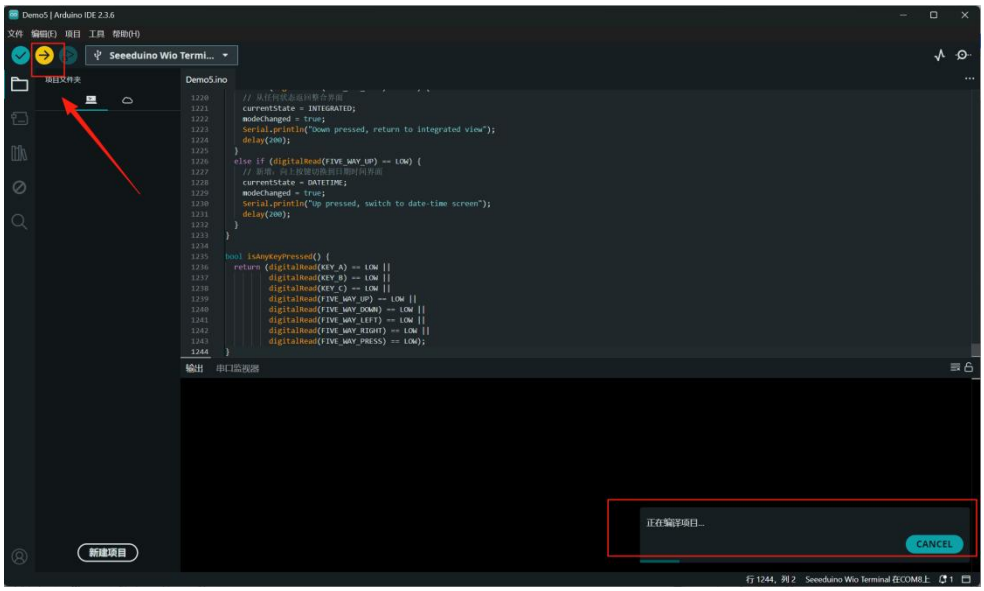
② 连接 Wio Terminal 并上传示例代码：

确保所有 #include 语句无报错

编译时无 undefined reference 错误（如有报错，可根据报错信息提示进行处理）

7) 上传代码，从以下链接直接复制到 Arduino IDE 上传即可；

https://github.com/huakaiyang/Multifunctional-Environmental-Monitoring-Instrument/blob/main/code.txt



4. 代码解析

4.1. 代码功能解析

1) 硬件与传感器初始化

```
// 硬件引脚定义
#define BUZZER_PIN WIO_BUZZER
#define DHTPIN 0
// ...其他引脚定义...

// 传感器对象
TFT_eSPI tft;           // TFT显示屏
LIS3DHTR<TwoWire> lis; // 加速度计
DHT dht(DHTPIN, DHTTYPE); // 温湿度传感器
```

- 显示屏：使用 TFT_eSPI 库驱动 320x240 分辨率屏幕
- 加速度计：LIS3DHTR 传感器，配置为 4G 量程和 50Hz 采样率
- 温湿度：DHT11 传感器，通过数字引脚 0 读取

2) 网络与时间同步

```
// NTP时间客户端
NTPClient timeClient(ntpUDP, "pool.ntp.org", 8 * 3600);

// WiFi和MQTT配置
const char *ssid = "abcd";
const char *password = "123456789";
PubSubClient mqttClient(wifiClient);
```

- NTP 时间同步：每 60 秒同步一次 UTC+8 时间
- WiFi 连接：实现自动重连机制
- MQTT 通信：连接 test.mosquitto.org 服务器，支持数据发布/订阅

3) 核心功能架构

```
enum DisplayState {
  INTEGRATED,      // 综合数据显示
  TEMPERATURE,     // 温度单独显示
  HUMIDITY,        // 湿度单独显示
  // ...其他状态...
  DATETIME         // 日期时间界面
};
```

- 8 种显示模式：通过物理按键切换不同数据显示界面
- 状态管理：使用 currentState 变量跟踪当前显示模式

4) 报警系统

```
enum AlarmType {
  NONE, TEMP_LOW_ALARM, TEMP_HIGH_ALARM,
  // ...7种报警类型...
};

const float TEMP_HIGH = 28.0;   // 温度高阈值
const float ACCEL_THRESHOLD = 1.3; // 加速度变化阈值
```

- 多类型报警：温度/湿度/光照/噪声/运动等异常检测
- 阈值配置：可自定义各传感器报警阈值
- 报警处理：
  - 蜂鸣器播放高低音交替警报
  - 屏幕显示红色警报界面
  - 自动上传 MQTT 报警信息
  - 支持按键手动解除报警

5) 传感器数据处理

```
struct SensorData {
  float temperature;
  float humidity;
  int lightLevel;
  // ...其他数据...
};

void readSensors() {
  // 读取所有传感器数据
  sensorData.temperature = dht.readTemperature();
  // ...其他传感器读取...
}
```

- 结构化存储：使用 SensorData 结构体统一管理数据
- 定时采集：每秒读取一次所有传感器
- 数据滤波：对 DHT11 数据做 NaN 检查

6) 显示系统

```
void drawIntegratedScreen() {
  // 显示所有传感器数据
  // 网络状态
  // 操作提示
}

void drawDateTimeScreen() {
  // 显示Seeed Studio LOGO
  // 当前日期和时间
  // NTP同步状态
}
```

- 综合界面：同时显示温度/湿度/光照/噪声/加速度数据

- 单传感器界面：大字体显示特定传感器数据
- 专业日期界面：
  - 绿色"Seeed Studio"LOGO
  - 居中显示日期(YYYY/MM/DD)和时间(HH:MM)
  - 显示下次 NTP 同步时间

7) 网络通信

```
void publishSensorData() {
  // 发布JSON格式传感器数据
  // 示例：{"temp":25.5,"humi":45.0,...}
}

void publishAlarmData(const char* alarmType) {
  // 发布报警信息含详细传感器数据
}
```

- MQTT 协议：使用 PubSubClient 库
- 数据格式：标准 JSON 格式
- 主题区分：
  - 普通数据发布到"WioTerminal"主题
  - 报警数据包含额外 alarm 字段

8) 按键控制

```
void checkButtons() {
  if (digitalRead(KEY_A)==LOW) currentState=TEMPERATURE;
  // ...其他按键检测...
}
```

- 物理按键映射：

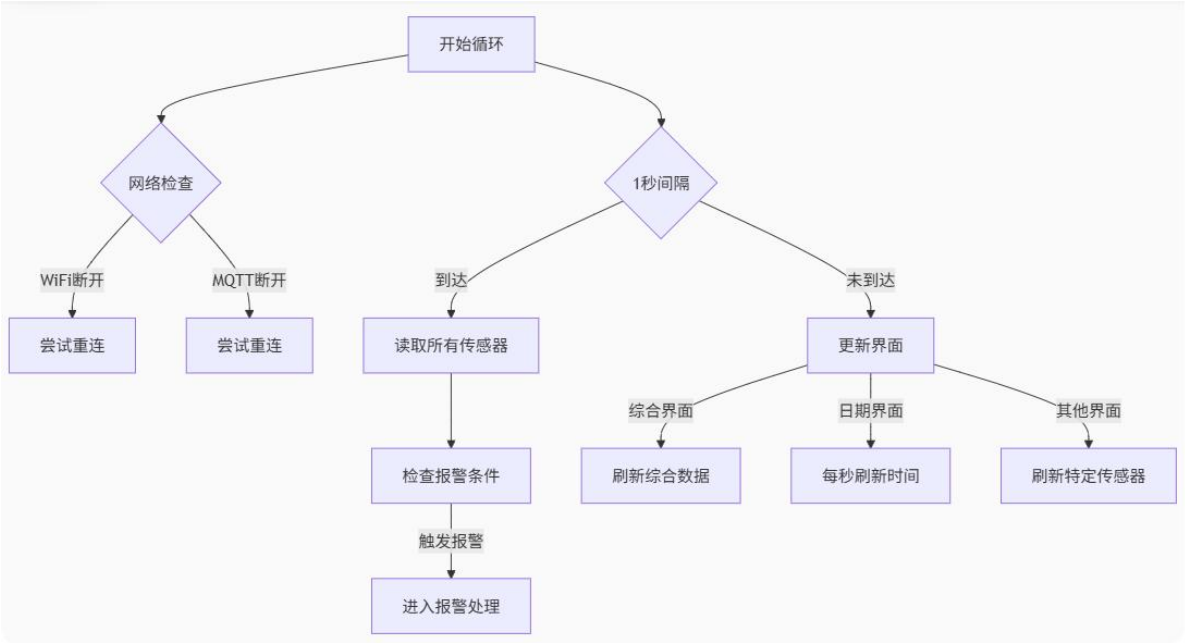
- KEY_A：温度界面
- KEY_B：湿度界面
- KEY_C：光照界面
- 五向键左：加速度界面
- 五向键右：噪声界面
- 五向键下：返回综合界面
- 五向键上：日期时间界面

4.2. 系统工作流程

1) 初始化阶段：

- 启动屏幕显示初始化信息
- 初始化所有传感器和网络连接
- 显示综合数据界面

2) 主循环



graph TD

A[开始循环] --> B{网络检查}

B -->|WiFi 断开| C[尝试重连]

B -->|MQTT 断开| D[尝试重连]

A --> E{1 秒间隔}

E -->|到达| F[读取所有传感器]

F --> G[检查报警条件]

G -->|触发报警| H[进入报警处理]

E -->|未到达| I[更新界面]

I -->|综合界面| J[刷新综合数据]

I -->|日期界面| K[每秒刷新时间]

I -->|其他界面| L[刷新特定传感器]

3) 报警处理流程

- 检测到传感器超阈值
- 记录报警类型和开始时间
- 切换到红色警报界面
- 蜂鸣器播放警报音
- 上传 MQTT 报警信息
- 5 秒后或按键按下时解除报警

4.3. 代码总结

该代码实现了一个完整的环境监测系统，具有：

- 多传感器数据采集（温湿度/光照/噪声/加速度）
- 8 种显示界面灵活切换
- 智能阈值报警系统
- 网络时间同步功能
- MQTT 物联网通信
- 直观的用户界面和交互

5. 常见问题

- 上传代码失败？

✓ 根据报错提示检查是否未按要求安装库，可复制报错信息使用 AI 工具搜索问题。

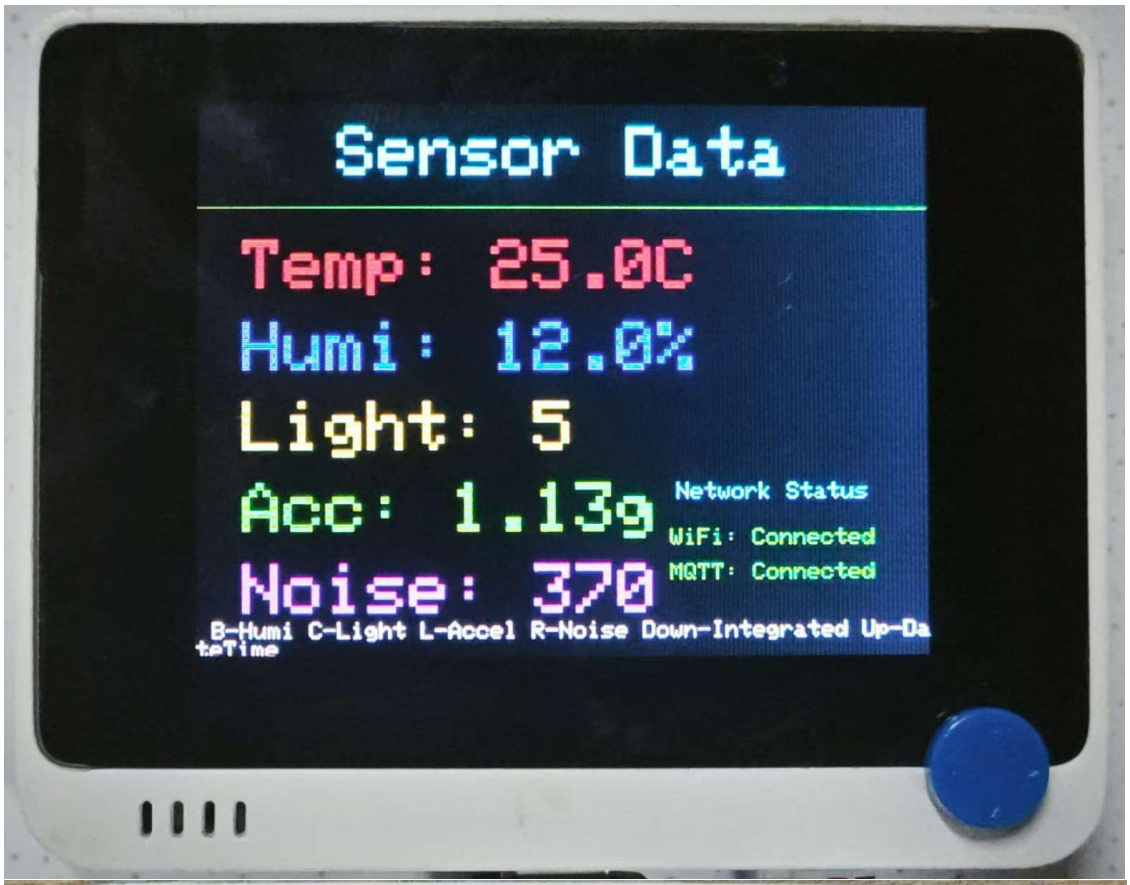
- Wio 终端屏幕卡住？

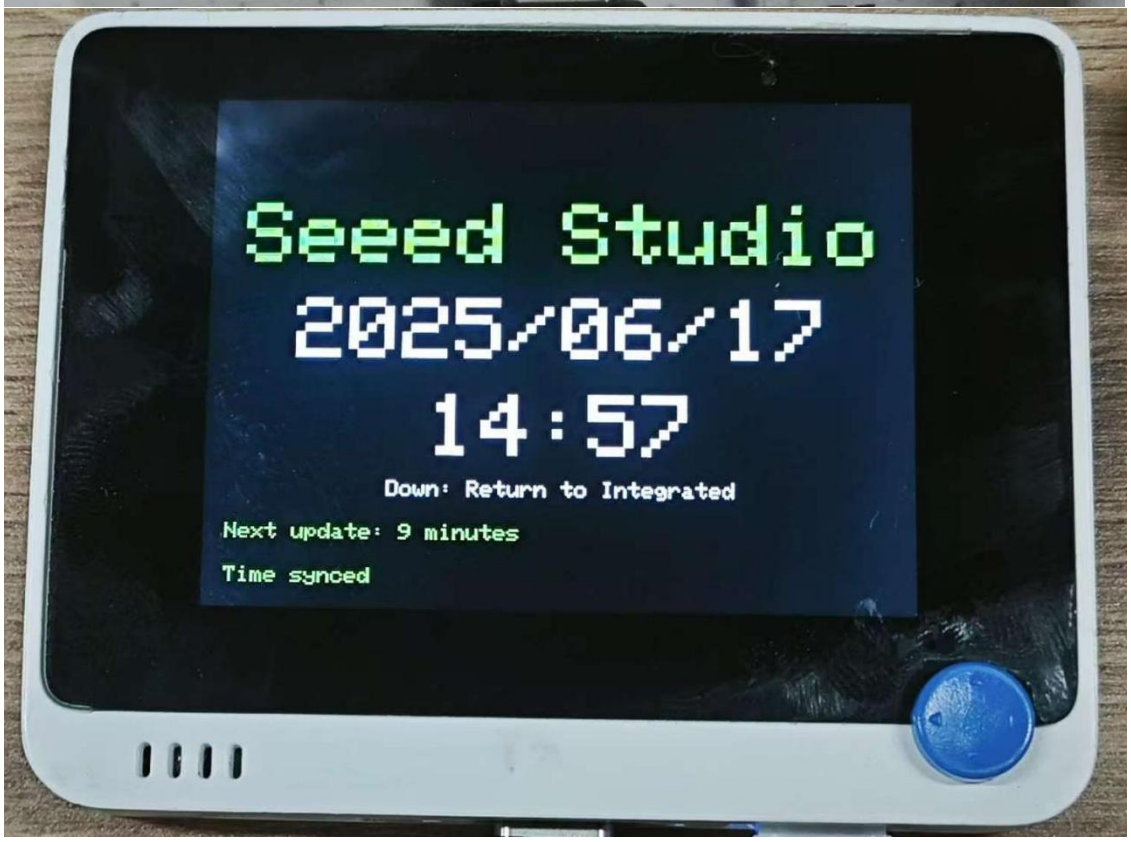
✓ 向下拨动侧边按钮重启设备。

- 网络异常，未能连接上 WiFi 和 MQTT 服务器？

✓ 可修改代码内 WiFi 和服务器配置，改为您需要链接的网络和服务器。

6. 示例界面


Sensor Data
Temp: 25.0C
Humi: 12.0%
Light: 5
Acc: 1.13g
Noise: 370
Network Status
WiFi: Connected
MQTT: Connected
B-Humi C-Light L-Accel R-Noise Down-Integrated Up-Da
teTime


Seeed Studio
2025/06/17
14:57
Down: Return to Integrated
Next update: 9 minutes
Time synced

7. 总代码

```
#include <TFT_eSPI.h>
#include "LIS3DHTR.h"
#include <DHT.h>
#include "rpcWiFi.h"
#include <PubSubClient.h>
#include <WiFiUdp.h> // 用于 NTP 时间同步
#include <NTPClient.h> // NTP 客户端库

// 硬件引脚定义
#define BUZZER_PIN WIO_BUZZER
#define DHTPIN 0
#define DHTTYPE DHT11
#define MIC_PIN WIO_MIC

// 传感器对象
TFT_eSPI tft;
LIS3DHTR<TwoWire> lis;
DHT dht(DHTPIN, DHTTYPE);

// NTP 时间客户端
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 8 * 3600, 60000); // UTC+8 时区，每 60 秒更新一次

// 按键定义
#define KEY_A WIO_KEY_A
#define KEY_B WIO_KEY_B
#define KEY_C WIO_KEY_C
#define FIVE_WAY_UP WIO_5S_UP
#define FIVE_WAY_DOWN WIO_5S_DOWN
#define FIVE_WAY_LEFT WIO_5S_LEFT
#define FIVE_WAY_RIGHT WIO_5S_RIGHT
#define FIVE_WAY_PRESS WIO_5S_PRESS

// 屏幕尺寸
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240

// 显示模式和状态
enum DisplayState { INTEGRATED, TEMPERATURE, HUMIDITY, LIGHT, ACCEL, NOISE, ALARM, DATETIME };
DisplayState currentState = INTEGRATED; // 初始为整合界面
DisplayState preAlarmState = INTEGRATED; // 报警前状态
bool modeChanged = true;
```

```
// 报警相关
bool isAlarmActive = false;
unsigned long alarmStartTime = 0;
const unsigned long ALARM_DURATION = 5000; // 5 秒报警时长
const int ALARM_SIREN_COUNT = 10; // 报警音效循环次数

// 报警类型
enum AlarmType { NONE, TEMP_LOW_ALARM, TEMP_HIGH_ALARM, HUMI_LOW_ALARM,
                 HUMI_HIGH_ALARM, LIGHT_ALARM, NOISE_ALARM, MOTION_ALARM };
AlarmType activeAlarmType = NONE;

// 传感器数据
struct SensorData {
  float temperature = 0;
  float humidity = 0;
  int lightLevel = 0;
  int noiseLevel = 0;
  float accelX = 0;
  float accelY = 0;
  float accelZ = 0;
  float accelMagnitude = 0;
  float lastAccelMagnitude = 0;
  unsigned long lastUpdate = 0;
};

SensorData sensorData;

// 阈值设置
const float TEMP_HIGH = 28.0;
const float TEMP_LOW = 10.0;
const float HUMI_HIGH = 80.0;
const float HUMI_LOW = 1.0;
const int LIGHT_HIGH = 60;
const int NOISE_HIGH = 1500; // 噪声阈值
const float ACCEL_THRESHOLD = 1.3; // 2g 变化阈值

// 使用您提供的 WiFi 和 MQTT 配置
const char *ssid = "abcd";                  // 您的 WiFi SSID
const char *password = "123456789"; // 您的 WiFi 密码
const char *clientID = "Wio-Terminal-Client";   // 设备名称，必须唯一
const char *pubTopic = "WioTerminal"; // 发布主题
const char *subTopic = "inTopic";         // 订阅主题
const char *mqttServer = "test.mosquitto.org"; // MQTT 服务器
```

```
const int mqttPort = 1883;                    // MQTT 端口

WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

// 蜂鸣器音效参数
const int SIREN_HIGH = 1000;
const int SIREN_LOW = 600;
const int SIREN_DURATION = 300;
int sirenCounter = 0;
bool isHighTone = true;
unsigned long lastToneTime = 0;

// 网络状态
bool wifiConnected = false;
bool mqttConnected = false;

// 时间同步状态
bool timeSynced = false;
unsigned long lastTimeSyncAttempt = 0;
const unsigned long TIME_SYNC_INTERVAL = 3600000; // 每小时同步一次时间

// 函数声明
void connectWiFi();
void connectMQTT();
void publishData(const char* topic, const char* payload);
void handleAlarmSound();
void drawIntegratedScreen(); // 新增：整合数据显示界面
void drawAlarmScreen();
void drawSensorUI();
void updateSensorDisplay();
void readSensors();
void checkButtons();
bool isAnyKeyPressed();
void handleAlarm();
void checkAlarmConditions();
void publishSensorData();
void publishAlarmData(const char* alarmType); // 新增：专门发布报警数据
void mqttCallback(char* topic, byte* payload, unsigned int length);
int centerTextX(const char* text, int fontSize);
int centerTextY(int lineNumber, int totalLines, int fontSize);
void drawDateTimeScreen(); // 新增：日期时间界面
void syncTime(); // 新增：同步网络时间
```

```
void setup() {
    Serial.begin(115200);
    while (!Serial); // 等待串口连接
    Serial.println("System starting...");

    // 初始化屏幕
    tft.begin();
    tft.setRotation(3);
    tft.fillScreen(TFT_BLACK);
    tft.setTextSize(2);
    tft.setTextColor(TFT_WHITE);
    tft.setCursor(50, 100);
    tft.print("Initializing...");
    delay(500);

    // 初始化传感器
    dht.begin();
    pinMode(WIO_LIGHT, INPUT);
    pinMode(MIC_PIN, INPUT);

    // 初始化加速度计
    Wire1.begin();
    lis.begin(Wire1); // 直接调用 begin()
    lis.setOutputDataRate(LIS3DHTR_DATARATE_50HZ); // 提高采样率
    lis.setFullScaleRange(LIS3DHTR_RANGE_4G);          // 提高量程
    Serial.println("Accelerometer initialized");

    // 初始化按键
    const int keys[] = {KEY_A, KEY_B, KEY_C, FIVE_WAY_UP, FIVE_WAY_DOWN,
                        FIVE_WAY_LEFT, FIVE_WAY_RIGHT, FIVE_WAY_PRESS};
    for (int i = 0; i < sizeof(keys)/sizeof(keys[0]); i++) {
        pinMode(keys[i], INPUT_PULLUP);
    }

    // 初始化蜂鸣器
    pinMode(BUZZER_PIN, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);

    // 初始读取加速度值
    sensorData.lastAccelMagnitude = sqrt(
        lis.getAccelerationX() * lis.getAccelerationX() +
        lis.getAccelerationY() * lis.getAccelerationY() +
        lis.getAccelerationZ() * lis.getAccelerationZ()
    );
```

```
    // 设置 MQTT 回调
    mqttClient.setServer(mqttServer, mqttPort);
    mqttClient.setCallback(mqttCallback);

    // 开始连接网络
    connectWiFi();

    // 初始显示整合界面
    drawIntegratedScreen();

    Serial.println("System initialization complete");
}

void loop() {
    static unsigned long lastSensorRead = 0;
    static unsigned long lastWiFiCheck = 0;
    static unsigned long lastMQTTCheck = 0;

    // 定期检查 WiFi 连接
    if (!wifiConnected && millis() - lastWiFiCheck > 5000) {
        connectWiFi();
        lastWiFiCheck = millis();
    }

    // 维护 MQTT 连接
    if (wifiConnected && !mqttConnected && millis() - lastMQTTCheck > 5000) {
        connectMQTT();
        lastMQTTCheck = millis();
    }

    // 处理 MQTT 消息
    if (mqttConnected) {
        mqttClient.loop();
    }

    // 每 1000ms 读取一次传感器数据（1 秒间隔）
    if (millis() - lastSensorRead > 1000) {
        readSensors();
        lastSensorRead = millis();

        // 检查报警条件
        checkAlarmConditions();
```

```
    // 发布传感器数据
    if (mqttConnected) {
      publishSensorData();
    }

    // 在整合界面下更新显示
    if (currentState == INTEGRATED) {
      drawIntegratedScreen();
    }
  }

  // 处理时间同步
  if (wifiConnected && (!timeSynced || millis() - lastTimeSyncAttempt > TIME_SYNC_INTERVAL)) {
    syncTime();
  }

  // 处理报警状态
  if (isAlarmActive) {
    handleAlarm();
  } else {
    // 检查按键切换模式
    checkButtons();

    // 更新显示
    if (modeChanged) {
      switch(currentState) {
        case INTEGRATED:
          drawIntegratedScreen();
          break;
        case DATETIME: // 新增日期时间界面
          drawDateTimeScreen();
          break;
        default:
          drawSensorUI();
      }
      modeChanged = false;
    }

    // 更新日期时间界面每秒刷新
    if (currentState == DATETIME) {
      static unsigned long lastDateTimeUpdate = 0;
      if (millis() - lastDateTimeUpdate > 1000) {
        drawDateTimeScreen();
        lastDateTimeUpdate = millis();
```

```
            }
        }
        // 更新其他传感器界面
        else if (currentState != INTEGRATED) {
            updateSensorDisplay();
        }
    }

    delay(50);
}

// 同步网络时间
void syncTime() {
    if (!wifiConnected) return;

    Serial.println("Syncing network time...");

    // 更新显示
    tft.fillScreen(TFT_BLACK);
    tft.setTextSize(2);
    tft.setTextColor(TFT_YELLOW);

    int x = centerTextX("Syncing network time...", 2);
    tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
    tft.print("Syncing network time...");

    timeClient.begin();
    if (timeClient.forceUpdate()) {
        timeSynced = true;
        lastTimeSyncAttempt = millis();
        Serial.println("Time sync successful!");

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_GREEN);

        x = centerTextX("Time sync successful!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("Time sync successful!");

        // 获取当前时间
        timeClient.update();
        Serial.print("Current time: ");
        Serial.println(timeClient.getFormattedTime());
```

```
        delay(1000);
    } else {
        timeSynced = false;
        Serial.println("Time sync failed!");

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_RED);

        x = centerTextX("Time sync failed!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("Time sync failed!");

        delay(1000);
    }

    // 返回整合界面
    drawIntegratedScreen();
}

// 计算文本水平居中位置
int centerTextX(const char* text, int fontSize) {
    tft.setTextSize(fontSize);
    int textWidth = tft.textWidth(text);
    return (SCREEN_WIDTH - textWidth) / 2;
}

// 计算文本垂直位置（多行居中）
int centerTextY(int lineNumber, int totalLines, int fontSize) {
    // 估算行高（根据字体大小）
    int lineHeight = fontSize * 8;
    int totalHeight = totalLines * lineHeight;
    int startY = (SCREEN_HEIGHT - totalHeight) / 2;
    return startY + (lineNumber - 1) * lineHeight;
}

// MQTT 回调函数
void mqttCallback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");

    // 将 payload 转换为字符串
```

```
    char message[length + 1];
    for (int i = 0; i < length; i++) {
        message[i] = (char)payload[i];
    }
    message[length] = '\0';

    Serial.println(message);
}

void connectWiFi() {
    if (WiFi.status() == WL_CONNECTED) {
        wifiConnected = true;
        return;
    }

    Serial.print("Connecting to WiFi: ");
    Serial.println(ssid);

    // 更新显示
    tft.fillScreen(TFT_BLACK);
    tft.setTextSize(2);
    tft.setTextColor(TFT_YELLOW);

    int x = centerTextX("Connecting to WiFi...", 2);
    tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
    tft.print("Connecting to WiFi...");

    WiFi.begin(ssid, password);

    unsigned long startTime = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
        delay(500);
        Serial.print(".");
    }

    if (WiFi.status() == WL_CONNECTED) {
        wifiConnected = true;
        Serial.println("\nConnected! IP address: ");
        Serial.println(WiFi.localIP());

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_GREEN);
```

```
        x = centerTextX("WiFi connected!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("WiFi connected!");

        delay(1000);
    } else {
        wifiConnected = false;
        Serial.println("\nConnection failed!");

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_RED);

        x = centerTextX("WiFi connection failed!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("WiFi connection failed!");

        delay(1000);
    }

    // 返回整合界面
    drawIntegratedScreen();
}

void connectMQTT() {
    if (mqttClient.connected()) {
        mqttConnected = true;
        return;
    }

    Serial.print("Connecting to MQTT server...");

    // 更新显示
    tft.fillScreen(TFT_BLACK);
    tft.setTextSize(2);
    tft.setTextColor(TFT_YELLOW);

    int x = centerTextX("Connecting to MQTT server...", 2);
    tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
    tft.print("Connecting to MQTT server...");

    if (mqttClient.connect(clientID)) {
        mqttConnected = true;
        Serial.println("Success");
```

```
        // 订阅主题
        mqttClient.subscribe(subTopic);
        Serial.print("Subscribed to topic: ");
        Serial.println(subTopic);

        // 发布连接消息
        publishData(pubTopic, "{\"message\": \"Wio Terminal is connected!\"}");

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_GREEN);

        x = centerTextX("MQTT connected!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("MQTT connected!");

        delay(1000);
    } else {
        mqttConnected = false;
        Serial.print("Failed, state=");
        Serial.println(mqttClient.state());

        // 更新显示
        tft.fillScreen(TFT_BLACK);
        tft.setTextColor(TFT_RED);

        x = centerTextX("MQTT connection failed!", 2);
        tft.setCursor(x, SCREEN_HEIGHT / 2 - 20);
        tft.print("MQTT connection failed!");

        delay(1000);
    }

    // 返回整合界面
    drawIntegratedScreen();
}

void readSensors() {
    // 读取温湿度
    float temp = dht.readTemperature();
    float humi = dht.readHumidity();

    if (!isnan(temp)) {
```

```
        sensorData.temperature = temp;
    }

    if (!isnan(humi)) {
        sensorData.humidity = humi;
    }

    // 读取光照
    sensorData.lightLevel = analogRead(WIO_LIGHT);

    // 读取噪声（麦克风）
    sensorData.noiseLevel = analogRead(MIC_PIN);

    // 读取加速度
    sensorData.accelX = lis.getAccelerationX();
    sensorData.accelY = lis.getAccelerationY();
    sensorData.accelZ = lis.getAccelerationZ();

    // 计算加速度向量长度
    sensorData.accelMagnitude = sqrt(
        sensorData.accelX * sensorData.accelX +
        sensorData.accelY * sensorData.accelY +
        sensorData.accelZ * sensorData.accelZ
    );

    Serial.print("Sensor Data: ");
    Serial.print("Temp="); Serial.print(sensorData.temperature);
    Serial.print(" Humi="); Serial.print(sensorData.humidity);
    Serial.print(" Light="); Serial.print(sensorData.lightLevel);
    Serial.print(" Noise="); Serial.print(sensorData.noiseLevel);
    Serial.print(" AccelMag="); Serial.println(sensorData.accelMagnitude);
}

void checkAlarmConditions() {
    static unsigned long lastAccelAlarmTime = 0;

    // 1. 检查设备移动超过 2g
    // 计算加速度变化量
    float deltaAccel = fabs(sensorData.accelMagnitude - sensorData.lastAccelMagnitude);
    sensorData.lastAccelMagnitude = sensorData.accelMagnitude;

    // 添加调试信息
    Serial.print("Acceleration change: ");
    Serial.print(deltaAccel);
```

```
Serial.print(" / Threshold: ");
Serial.println(ACCEL_THRESHOLD);

if (deltaAccel > ACCEL_THRESHOLD && !isAlarmActive) {
   // 添加防抖动机制
   if (millis() - lastAccelAlarmTime > 1000) {
      isAlarmActive = true;
      alarmStartTime = millis();
      preAlarmState = currentState;
      activeAlarmType = MOTION_ALARM;
      Serial.println("Motion alarm triggered");
      publishAlarmData("Motion"); // 使用专用报警上传函数
      lastAccelAlarmTime = millis();
   }
}

// 2. 检查光照超过 200
if (sensorData.lightLevel > LIGHT_HIGH && !isAlarmActive) {
   isAlarmActive = true;
   alarmStartTime = millis();
   preAlarmState = currentState;
   activeAlarmType = LIGHT_ALARM;
   Serial.println("Light alarm triggered");
   publishAlarmData("Light");
}

// 3. 检查噪声超过阈值
if (sensorData.noiseLevel > NOISE_HIGH && !isAlarmActive) {
   isAlarmActive = true;
   alarmStartTime = millis();
   preAlarmState = currentState;
   activeAlarmType = NOISE_ALARM;
   Serial.println("Noise alarm triggered");
   publishAlarmData("Noise");
}

// 4. 检查温湿度超出阈值
if (sensorData.temperature < TEMP_LOW && !isAlarmActive) {
   isAlarmActive = true;
   alarmStartTime = millis();
   preAlarmState = currentState;
   activeAlarmType = TEMP_LOW_ALARM;
   Serial.println("Low temperature alarm triggered");
   publishAlarmData("LowTemperature");
```

```
}

if (sensorData.temperature > TEMP_HIGH && !isAlarmActive) {
    isAlarmActive = true;
    alarmStartTime = millis();
    preAlarmState = currentState;
    activeAlarmType = TEMP_HIGH_ALARM;
    Serial.println("High temperature alarm triggered");
    publishAlarmData("HighTemperature");
}

if (sensorData.humidity < HUMI_LOW && !isAlarmActive) {
    isAlarmActive = true;
    alarmStartTime = millis();
    preAlarmState = currentState;
    activeAlarmType = HUMI_LOW_ALARM;
    Serial.println("Low humidity alarm triggered");
    publishAlarmData("LowHumidity");
}

if (sensorData.humidity > HUMI_HIGH && !isAlarmActive) {
    isAlarmActive = true;
    alarmStartTime = millis();
    preAlarmState = currentState;
    activeAlarmType = HUMI_HIGH_ALARM;
    Serial.println("High humidity alarm triggered");
    publishAlarmData("HighHumidity");
}

// 恢复正常时停止报警（基于报警类型）
if (isAlarmActive) {
    bool shouldStopAlarm = false;

    switch(activeAlarmType) {
        case TEMP_LOW_ALARM:
            if (sensorData.temperature >= TEMP_LOW) shouldStopAlarm = true;
            break;

        case TEMP_HIGH_ALARM:
            if (sensorData.temperature <= TEMP_HIGH) shouldStopAlarm = true;
            break;

        case HUMI_LOW_ALARM:
            if (sensorData.humidity >= HUMI_LOW) shouldStopAlarm = true;
```

```
            break;

        case HUMI_HIGH_ALARM:
            if (sensorData.humidity <= HUMI_HIGH) shouldStopAlarm = true;
            break;

        case LIGHT_ALARM:
        case NOISE_ALARM:
        case MOTION_ALARM:
            // 瞬时报警 5 秒后自动停止
            if (millis() - alarmStartTime > ALARM_DURATION) shouldStopAlarm = true;
            break;

        default:
            // 未知报警类型 5 秒后停止
            if (millis() - alarmStartTime > ALARM_DURATION) shouldStopAlarm = true;
      }

      if (shouldStopAlarm) {
        isAlarmActive = false;
        activeAlarmType = NONE;
        Serial.println("Alarm resolved");
        publishData(pubTopic, "{\"alarm\":\"resolved\"}");
      }
    }
}

void handleAlarm() {
    // 处理报警声音
    handleAlarmSound();

    // 显示报警界面
    drawAlarmScreen();

    // 检查按键终止报警
    if (isAnyKeyPressed()) {
      isAlarmActive = false;
      activeAlarmType = NONE;
      sirenCounter = 0;
      noTone(BUZZER_PIN);
      currentState = preAlarmState;
      modeChanged = true;
      Serial.println("Alarm manually canceled");
      publishData(pubTopic, "{\"alarm\":\"canceled\"}");
```

```
        return;
    }
}

void handleAlarmSound() {
    if (millis() - lastToneTime < SIREN_DURATION) return;

    if (isHighTone) {
        tone(BUZZER_PIN, SIREN_HIGH, SIREN_DURATION);
    } else {
        tone(BUZZER_PIN, SIREN_LOW, SIREN_DURATION);
    }

    isHighTone = !isHighTone;
    lastToneTime = millis();
    sirenCounter++;

    if (sirenCounter >= ALARM_SIREN_COUNT * 2) {
        sirenCounter = 0;
    }
}

void publishData(const char* topic, const char* payload) {
    if (mqttClient.connected()) {
        mqttClient.publish(topic, payload);
        Serial.print("Publish message: ");
        Serial.print(topic);
        Serial.print(" - ");
        Serial.println(payload);
    }
}

// 新增：专门发布报警数据
void publishAlarmData(const char* alarmType) {
    char payload[100];
    snprintf(payload, sizeof(payload),
                "{\"alarm\":\"%s\",\"temp\":%.1f,\"humi\":%.1f,\"light\":%d,\"noise\":%d,\"accelMag\":%.2f}",
                alarmType, sensorData.temperature, sensorData.humidity,
                sensorData.lightLevel, sensorData.noiseLevel, sensorData.accelMagnitude);

    publishData(pubTopic, payload);
}

void publishSensorData() {
```

```
    char payload[150];
    snprintf(payload, sizeof(payload),
                "{\"temp\":%.1f,\"humi\":%.1f,\"light\":%d,\"noise\":%d,\"accelMag\":%.2f}",
                sensorData.temperature, sensorData.humidity, sensorData.lightLevel,
                sensorData.noiseLevel, sensorData.accelMagnitude);

    publishData(pubTopic, payload);
}

// 修改后的整合数据显示界面（网络状态与加速度数据对齐）
void drawIntegratedScreen() {
    tft.fillScreen(TFT_BLACK);

    // 显示标题
    tft.setTextColor(TFT_CYAN);
    tft.setTextSize(3);

    const char* title = "Sensor Data";
    int xTitle = centerTextX(title, 3);
    tft.setCursor(xTitle, 10);
    tft.print(title);

    // 分隔线
    tft.drawFastHLine(0, 45, SCREEN_WIDTH, TFT_DARKGREEN);

    // 显示所有传感器数据（左侧区域）
    tft.setTextSize(3);
    int startY = 60;
    int lineHeight = 35;

    // 温度
    tft.setTextColor(TFT_RED);
    char tempStr[30];
    sprintf(tempStr, "Temp: %.1fC", sensorData.temperature);
    tft.setCursor(20, startY);
    tft.print(tempStr);

    // 湿度
    tft.setTextColor(TFT_BLUE);
    char humiStr[30];
    sprintf(humiStr, "Humi: %.1f%%", sensorData.humidity);
    tft.setCursor(20, startY + lineHeight);
    tft.print(humiStr);
```

```
// 光照
tft.setTextColor(TFT_YELLOW);
char lightStr[30];
sprintf(lightStr, "Light: %d", sensorData.lightLevel);
tft.setCursor(20, startY + lineHeight * 2);
tft.print(lightStr);

// 加速度 - 网络状态将显示在这一行的右侧
tft.setTextColor(TFT_GREEN);
char accelStr[30];
sprintf(accelStr, "Acc: %.2fg", sensorData.accelMagnitude);
tft.setCursor(20, startY + lineHeight * 3);
tft.print(accelStr);

// 噪声
tft.setTextColor(TFT_MAGENTA);
char noiseStr[30];
sprintf(noiseStr, "Noise: %d", sensorData.noiseLevel);
tft.setCursor(20, startY + lineHeight * 4);
tft.print(noiseStr);

// 在加速度行右侧显示网络状态（与加速度行对齐）
int statusBoxX = 180; // 右侧区域起始 X 坐标
int statusBoxY = startY + lineHeight * 3; // 与加速度行对齐

// 显示状态标题（使用小字体）
tft.setTextSize(1);
tft.setTextColor(TFT_CYAN);
const char* statusTitle = "Network Status";
int xStatusTitle = statusBoxX + (SCREEN_WIDTH - statusBoxX - tft.textWidth(statusTitle)) / 2;
tft.setCursor(xStatusTitle, statusBoxY);
tft.print(statusTitle);

// 显示 WiFi 状态（使用小字体）
tft.setTextSize(1);
if (wifiConnected) {
    tft.setTextColor(TFT_GREEN);
    const char* wifiStatus = "WiFi: Connected";
    int xWifi = statusBoxX + (SCREEN_WIDTH - statusBoxX - tft.textWidth(wifiStatus)) / 2;
    tft.setCursor(xWifi, statusBoxY + 20);
    tft.print(wifiStatus);
} else {
    tft.setTextColor(TFT_RED);
    const char* wifiStatus = "WiFi: Disconnected";
```

```
    int xWifi = statusBoxX + (SCREEN_WIDTH - statusBoxX - tft.textWidth(wifiStatus)) / 2;
    tft.setCursor(xWifi, statusBoxY + 20);
    tft.print(wifiStatus);
  }

  // 显示 MQTT 状态（使用小字体）
  if (mqttConnected) {
    tft.setTextColor(TFT_GREEN);
    const char* mqttStatus = "MQTT: Connected";
    int xMqtt = statusBoxX + (SCREEN_WIDTH - statusBoxX - tft.textWidth(mqttStatus)) / 2;
    tft.setCursor(xMqtt, statusBoxY + 35);
    tft.print(mqttStatus);
  } else {
    tft.setTextColor(TFT_RED);
    const char* mqttStatus = "MQTT: Disconnected";
    int xMqtt = statusBoxX + (SCREEN_WIDTH - statusBoxX - tft.textWidth(mqttStatus)) / 2;
    tft.setCursor(xMqtt, statusBoxY + 35);
    tft.print(mqttStatus);
  }

  // 操作提示（底部）
  tft.setTextColor(TFT_WHITE);
  tft.setTextSize(1);
  const char* prompt = "A-Temp B-Humi C-Light L-Accel R-Noise Down-Integrated Up-DateTime";
  int xPrompt = centerTextX(prompt, 1);
  tft.setCursor(xPrompt, SCREEN_HEIGHT - 15);
  tft.print(prompt);
}

// 修改后的日期时间界面（使用 NTP 时间）
void drawDateTimeScreen() {
  tft.fillScreen(TFT_BLACK);

  // 更新时间（每小时更新一次）
  static unsigned long lastUpdateTime = 0;
  const unsigned long UPDATE_INTERVAL = 3600000; // 1 小时 = 3,600,000 毫秒

  if (timeSynced && (millis() - lastUpdateTime > UPDATE_INTERVAL)) {
    timeClient.update();  // 每小时更新一次时间
    lastUpdateTime = millis();
  }

  // 计算行高（增加行间距）
  int titleSize = 4; // Seeed Studio 的字体大小
```

```
int datetimeSize = 4; // 日期时间的字体大小
int lineHeight = 45; // 增加行间距为 45 像素

// 显示 Seeed Studio（绿色字体）
tft.setTextColor(TFT_GREEN);
tft.setTextSize(titleSize);

const char* seeedText = "Seeed Studio";
int xSeeed = centerTextX(seeedText, titleSize);
int ySeeed = 50; // 固定位置，顶部留出空间
tft.setCursor(xSeeed, ySeeed);
tft.print(seeedText);

// 获取并格式化日期时间
String dateStr = "N/A";
String timeStr = "N/A";

if (timeSynced) {
  // 获取当前日期
  time_t rawtime = timeClient.getEpochTime();
  struct tm *ti;
  ti = localtime(&rawtime);

  // 格式化日期为 YYYY/MM/DD
  char dateBuffer[20];
  sprintf(dateBuffer, "%04d/%02d/%02d", ti->tm_year + 1900, ti->tm_mon + 1, ti->tm_mday);
  dateStr = String(dateBuffer);

  // 格式化时间为 HH:MM
  char timeBuffer[10];
  sprintf(timeBuffer, "%02d:%02d", ti->tm_hour, ti->tm_min);
  timeStr = String(timeBuffer);
}

// 显示日期（白色字体）
tft.setTextColor(TFT_WHITE);
tft.setTextSize(datetimeSize);

int xDate = centerTextX(dateStr.c_str(), datetimeSize);
int yDate = ySeeed + lineHeight; // 在 Seeed Studio 下方 45 像素处
tft.setCursor(xDate, yDate);
tft.print(dateStr);

// 显示时间（白色字体）
```

```
    int xTime = centerTextX(timeStr.c_str(), datetimeSize);
    int yTime = yDate + lineHeight; // 在日期下方 45 像素处
    tft.setCursor(xTime, yTime);
    tft.print(timeStr);

    // 显示时间同步状态
    tft.setTextSize(1);
    if (timeSynced) {
        tft.setTextColor(TFT_GREEN);
        tft.setCursor(10, SCREEN_HEIGHT - 20);
        tft.print("Time synced");
    } else {
        tft.setTextColor(TFT_RED);
        tft.setCursor(10, SCREEN_HEIGHT - 20);
        tft.print("Time not synced");
    }

    // 显示下次更新时间
    unsigned long secondsUntilUpdate = (lastUpdateTime + UPDATE_INTERVAL - millis()) / 1000;
    unsigned long minutesUntilUpdate = secondsUntilUpdate / 60;
    unsigned long hoursUntilUpdate = minutesUntilUpdate / 60;

    char updateInfo[50];
    if (hoursUntilUpdate > 0) {
        sprintf(updateInfo, "Next update: %lu hours", hoursUntilUpdate);
    } else if (minutesUntilUpdate > 0) {
        sprintf(updateInfo, "Next update: %lu minutes", minutesUntilUpdate);
    } else {
        sprintf(updateInfo, "Next update: %lu seconds", secondsUntilUpdate);
    }

    tft.setCursor(10, SCREEN_HEIGHT - 40);
    tft.print(updateInfo);

    // 操作提示（底部）
    tft.setTextColor(TFT_WHITE);
    tft.setTextSize(1);
    const char* prompt = "Down: Return to Integrated";
    int xPrompt = centerTextX(prompt, 1);
    tft.setCursor(xPrompt, SCREEN_HEIGHT - 60);
    tft.print(prompt);
}

void drawAlarmScreen() {
```

```
tft.fillScreen(TFT_RED);

// 显示大 X
tft.setTextColor(TFT_WHITE);
tft.setTextSize(20);

const char* xChar = "X";
int xPos = centerTextX(xChar, 20);
int yPos = SCREEN_HEIGHT / 2 - 40;
tft.setCursor(xPos, yPos);
tft.print(xChar);

// 显示报警文本
tft.setTextSize(3);
const char* alarmText = "ALARM!";
int xAlarm = centerTextX(alarmText, 3);
tft.setCursor(xAlarm, yPos + 60);
tft.print(alarmText);

// 显示报警原因
tft.setTextSize(2);
const char* reason = "";

switch(activeAlarmType) {
  case TEMP_LOW_ALARM:
    reason = "LOW TEMPERATURE ALARM!";
    break;
  case TEMP_HIGH_ALARM:
    reason = "HIGH TEMPERATURE ALARM!";
    break;
  case HUMI_LOW_ALARM:
    reason = "LOW HUMIDITY ALARM!";
    break;
  case HUMI_HIGH_ALARM:
    reason = "HIGH HUMIDITY ALARM!";
    break;
  case LIGHT_ALARM:
    reason = "HIGH LIGHT LEVEL ALARM!";
    break;
  case NOISE_ALARM:
    reason = "HIGH NOISE LEVEL ALARM!";
    break;
  case MOTION_ALARM:
    reason = "MOTION DETECTED ALARM!";
```

```
            break;
        default:
            reason = "UNKNOWN ALARM!";
    }

    int xReason = centerTextX(reason, 2);
    tft.setCursor(xReason, SCREEN_HEIGHT - 40);
    tft.print(reason);
}

void drawSensorUI() {
    tft.fillScreen(TFT_BLACK);

    // 显示传感器名称
    tft.setTextSize(3);
    const char* title = "";

    switch(currentState) {
        case TEMPERATURE:
            tft.setTextColor(TFT_RED);
            title = "TEMPERATURE";
            break;

        case HUMIDITY:
            tft.setTextColor(TFT_BLUE);
            title = "HUMIDITY";
            break;

        case LIGHT:
            tft.setTextColor(TFT_YELLOW);
            title = "LIGHT LEVEL";
            break;

        case ACCEL:
            tft.setTextColor(TFT_GREEN);
            title = "ACCELERATION";
            break;

        case NOISE:
            tft.setTextColor(TFT_MAGENTA);
            title = "NOISE LEVEL";
            break;
    }
```

```
    int xTitle = centerTextX(title, 3);
    tft.setCursor(xTitle, 20);
    tft.print(title);

    // 分隔线
    tft.drawFastHLine(0, 60, SCREEN_WIDTH, TFT_DARKGREEN);

    // 操作提示
    tft.setTextColor(TFT_WHITE);
    tft.setTextSize(1);
    const char* prompt = "Down: Return to Integrated";
    int xPrompt = centerTextX(prompt, 1);
    tft.setCursor(xPrompt, SCREEN_HEIGHT - 20);
    tft.print(prompt);
}

// 修复后的函数：为每个 case 添加独立作用域
void updateSensorDisplay() {
    // 清除主显示区域
    tft.fillRect(0, 70, SCREEN_WIDTH, SCREEN_HEIGHT - 90, TFT_BLACK);

    switch(currentState) {
        case TEMPERATURE: {
            // 显示温度
            tft.setTextSize(8);

            // 根据阈值设置颜色
            if (sensorData.temperature < TEMP_LOW) {
                tft.setTextColor(TFT_BLUE);
            } else if (sensorData.temperature > TEMP_HIGH) {
                tft.setTextColor(TFT_RED);
            } else {
                tft.setTextColor(TFT_GREEN);
            }

            // 创建温度字符串
            char tempStr[20];
            sprintf(tempStr, "%.1fC", sensorData.temperature);
            int xTemp = centerTextX(tempStr, 8);
            tft.setCursor(xTemp, SCREEN_HEIGHT / 2 - 20); // 下移 20 像素
            tft.print(tempStr);
            break;
        }
```

```
case HUMIDITY: {
    // 显示湿度
    tft.setTextSize(8);

    // 根据阈值设置颜色
    if (sensorData.humidity < HUMI_LOW) {
        tft.setTextColor(TFT_BLUE);
    } else if (sensorData.humidity > HUMI_HIGH) {
        tft.setTextColor(TFT_RED);
    } else {
        tft.setTextColor(TFT_CYAN);
    }

    // 创建湿度字符串
    char humiStr[20];
    sprintf(humiStr, "%.1f%%", sensorData.humidity);
    int xHumi = centerTextX(humiStr, 8);
    tft.setCursor(xHumi, SCREEN_HEIGHT / 2 - 20); // 下移 20 像素
    tft.print(humiStr);
    break;
}

case LIGHT: {
    // 显示光照值
    tft.setTextSize(8);
    tft.setTextColor(sensorData.lightLevel > LIGHT_HIGH ? TFT_RED : TFT_YELLOW);

    char lightStr[10];
    sprintf(lightStr, "%d", sensorData.lightLevel);
    int xLight = centerTextX(lightStr, 8);
    tft.setCursor(xLight, SCREEN_HEIGHT / 2 - 40); // 下移 20 像素
    tft.print(lightStr);

    // 显示光照条
    int barWidth = map(sensorData.lightLevel, 0, 1023, 0, 240);
    int barX = (SCREEN_WIDTH - 240) / 2;
    int barY = SCREEN_HEIGHT / 2 + 40; // 下移 20 像素

    tft.fillRect(barX, barY, barWidth, 20, TFT_YELLOW);
    tft.fillRect(barX + barWidth, barY, 240 - barWidth, 20, TFT_DARKGREY);
    tft.drawRect(barX, barY, 240, 20, TFT_WHITE);

    // 显示阈值线
    int thresholdX = barX + map(LIGHT_HIGH, 0, 1023, 0, 240);
```

```
    tft.drawFastVLine(thresholdX, barY - 10, 40, TFT_RED);

    // 显示范围标签
    tft.setTextSize(1);
    tft.setTextColor(TFT_WHITE);
    tft.setCursor(barX - 15, barY + 25);
    tft.print("0");
    tft.setCursor(barX + 225, barY + 25);
    tft.print("1023");
    break;
}

case ACCEL: {
    // 显示加速度向量长度
    tft.setTextSize(4);
    tft.setTextColor(TFT_GREEN);

    char magStr[30];
    sprintf(magStr, "Mag: %.2f g", sensorData.accelMagnitude);
    int xMag = centerTextX(magStr, 4);
    tft.setCursor(xMag, SCREEN_HEIGHT / 2 - 40); // 下移 20 像素
    tft.print(magStr);

    // 显示加速度分量
    tft.setTextSize(3);

    // X 轴
    char xStr[20];
    sprintf(xStr, "X: %.2f", sensorData.accelX);
    int xX = centerTextX(xStr, 3);
    tft.setTextColor(TFT_RED);
    tft.setCursor(xX, SCREEN_HEIGHT / 2); // 下移 20 像素
    tft.print(xStr);

    // Y 轴
    char yStr[20];
    sprintf(yStr, "Y: %.2f", sensorData.accelY);
    int xY = centerTextX(yStr, 3);
    tft.setTextColor(TFT_GREEN);
    tft.setCursor(xY, SCREEN_HEIGHT / 2 + 30); // 下移 20 像素
    tft.print(yStr);

    // Z 轴
    char zStr[20];
```

```
        sprintf(zStr, "Z: %.2f", sensorData.accelZ);
        int xZ = centerTextX(zStr, 3);
        tft.setTextColor(TFT_BLUE);
        tft.setCursor(xZ, SCREEN_HEIGHT / 2 + 60); //  下移 20 像素
        tft.print(zStr);
        break;
      }

      case NOISE: {
        //  显示噪声值
        tft.setTextSize(8);
        tft.setTextColor(sensorData.noiseLevel > NOISE_HIGH ? TFT_RED : TFT_MAGENTA);

        char noiseStr[10];
        sprintf(noiseStr, "%d", sensorData.noiseLevel);
        int xNoise = centerTextX(noiseStr, 8);
        tft.setCursor(xNoise, SCREEN_HEIGHT / 2 - 40); //  下移 20 像素
        tft.print(noiseStr);

        //  显示噪声条
        int barWidth = map(sensorData.noiseLevel, 0, 1023, 0, 240);
        int barX = (SCREEN_WIDTH - 240) / 2;
        int barY = SCREEN_HEIGHT / 2 + 40; //  下移 20 像素

        tft.fillRect(barX, barY, barWidth, 20, TFT_MAGENTA);
        tft.fillRect(barX + barWidth, barY, 240 - barWidth, 20, TFT_DARKGREY);
        tft.drawRect(barX, barY, 240, 20, TFT_WHITE);

        //  显示阈值线
        int thresholdX = barX + map(NOISE_HIGH, 0, 1023, 0, 240);
        tft.drawFastVLine(thresholdX, barY - 10, 40, TFT_RED);

        //  显示范围标签
        tft.setTextSize(1);
        tft.setTextColor(TFT_WHITE);
        tft.setCursor(barX - 15, barY + 25);
        tft.print("0");
        tft.setCursor(barX + 225, barY + 25);
        tft.print("1023");
        break;
      }
    }
}
```

```
void checkButtons() {
  // 正常模式下的按键检测
  if (digitalRead(KEY_A) == LOW) {
    currentState = TEMPERATURE;
    modeChanged = true;
    Serial.println("A pressed, switch to temperature view");
    delay(200);
  }
  else if (digitalRead(KEY_B) == LOW) {
    currentState = HUMIDITY;
    modeChanged = true;
    Serial.println("B pressed, switch to humidity view");
    delay(200);
  }
  else if (digitalRead(KEY_C) == LOW) {
    currentState = LIGHT;
    modeChanged = true;
    Serial.println("C pressed, switch to light view");
    delay(200);
  }
  else if (digitalRead(FIVE_WAY_LEFT) == LOW) {
    currentState = ACCEL;
    modeChanged = true;
    Serial.println("Left pressed, switch to acceleration view");
    delay(200);
  }
  else if (digitalRead(FIVE_WAY_RIGHT) == LOW) {
    currentState = NOISE;
    modeChanged = true;
    Serial.println("Right pressed, switch to noise view");
    delay(200);
  }
  else if (digitalRead(FIVE_WAY_DOWN) == LOW) {
    // 从任何状态返回整合界面
    currentState = INTEGRATED;
    modeChanged = true;
    Serial.println("Down pressed, return to integrated view");
    delay(200);
  }
  else if (digitalRead(FIVE_WAY_UP) == LOW) {
    // 新增：向上按键切换到日期时间界面
    currentState = DATETIME;
    modeChanged = true;
    Serial.println("Up pressed, switch to date-time screen");
```

```
        delay(200);
    }
}

bool isAnyKeyPressed() {
    return (digitalRead(KEY_A) == LOW ||
            digitalRead(KEY_B) == LOW ||
            digitalRead(KEY_C) == LOW ||
            digitalRead(FIVE_WAY_UP) == LOW ||
            digitalRead(FIVE_WAY_DOWN) == LOW ||
            digitalRead(FIVE_WAY_LEFT) == LOW ||
            digitalRead(FIVE_WAY_RIGHT) == LOW ||
            digitalRead(FIVE_WAY_PRESS) == LOW);
}
```