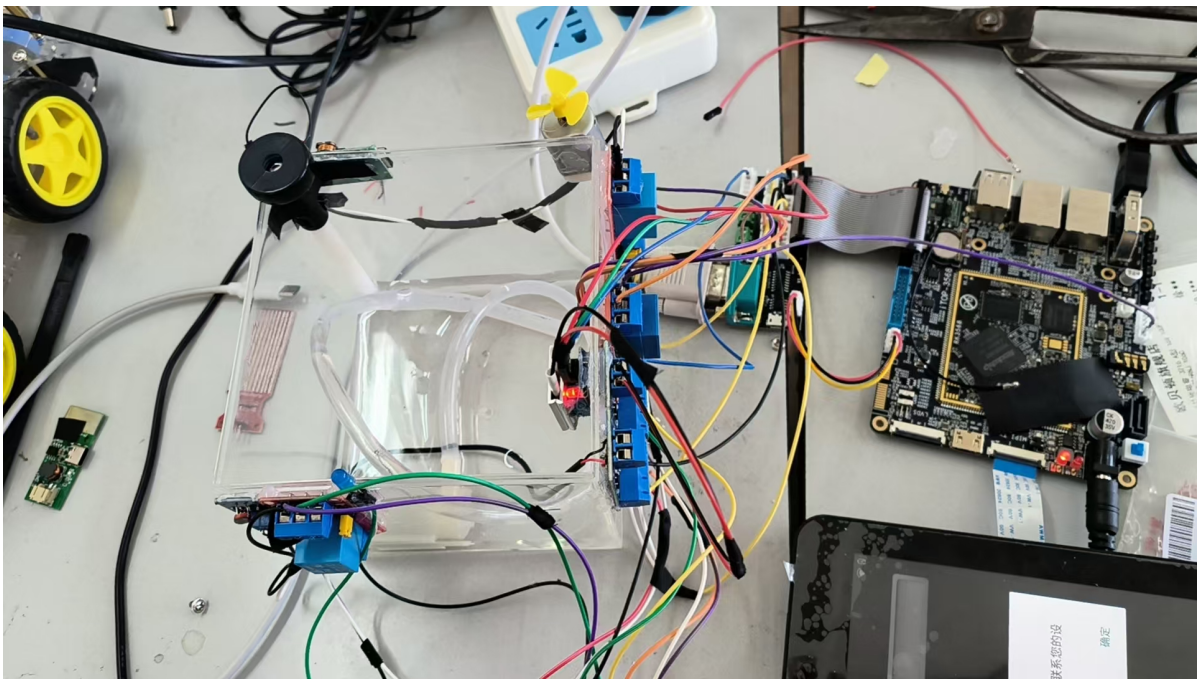
rk3568温湿度Android控制系统



1.使用的模块

加湿器、dht11温湿度传感器、加热模块、水泵、光敏模块、水位传感器、直流电机、继电器、3.3v/5v电源模块

涉及到单总线、GPIO、pinctrl

dht11原理

https://blog.csdn.net/m0_55849362/article/details/126426768?fromshare=blogdetail&sharetype=blogdetail&sharerId=126426768&sharerefer=PC&sharesource=xyydjjsbsn&sharefrom=from_link

2.设备树

```
//humiture
    humiture:humiture{
        compatible = "myhumiture";
        io-channel-names = "adc1";
        io-channels = <&saradc 7>;

        //I2S3_SCLK_M0
        dht11:dht11{
            compatible = "dht11";
            dht11-gpios = <&gpio3 RK_PA3 GPIO_ACTIVE_HIGH>;
            pinctrl-names = "default";
            pinctrl-0 = <&dht11_gpio3_a3_ctrl>;
        };

        //adc7
        water_level:water_level{
            compatible = "water_level";
            //io-channel-names = "adc1";
            //io-channels = <&saradc 7>;
        };

        //I2C3_SDA_M0
        Fan:Fan{
            compatible = "Fan";
            fan-gpios = <&gpio1 RK_PA0 GPIO_ACTIVE_HIGH>;
            pinctrl-names = "default";
            pinctrl-0 = <&my_gpio1_a0_ctrl>;
        };

        //I2C3_SCL_M0
        pump:pump{
            compatible = "pump";
            pump-gpios = <&gpio1 RK_PA1 GPIO_ACTIVE_HIGH>;
            pinctrl-names = "default";
            pinctrl-0 = <&my_gpio1_a1_ctrl>;
        };

        heating:heating{
```

```
            compatible = "heating";
            heating-gpios = <&gpio0 RK_PC0 GPIO_ACTIVE_HIGH>;
            pinctrl-names = "default";
            pinctrl-0 = <&my_gpio0_c0_ctrl>;
        };

        humidifying:humidifying{
            compatible = "humidifying";
            humidifying-gpios = <&gpio0 RK_PC1 GPIO_ACTIVE_HIGH>;
            pinctrl-names = "default";
            pinctrl-0 = <&my_gpio0_c1_ctrl>;
        };
    };


mygpio{
        my_gpio1_a0_ctrl:my-gpio-a0-ctrl{
            rockchip,pins = <1 RK_PA0 RK_FUNC_GPIO &pcfg_pull_none>;
        };
        my_gpio1_a1_ctrl:my-gpio-a1-ctrl{
            rockchip,pins = <1 RK_PA1 RK_FUNC_GPIO &pcfg_pull_none>;
        };
        my_gpio0_c0_ctrl:my-gpio-c0-ctrl{
            rockchip,pins = <0 RK_PC0 RK_FUNC_GPIO &pcfg_pull_none>;
        };
        my_gpio0_c1_ctrl:my-gpio-c1-ctrl{
            rockchip,pins = <0 RK_PC1 RK_FUNC_GPIO &pcfg_pull_none>;
        };
    };
dht11{
        dht11_gpio3_a3_ctrl:dht11-gpio-a3-ctrl{
            rockchip,pins = <3 RK_PA3 RK_FUNC_GPIO &pcfg_pull_none>;
        };
    };
```

3.驱动层

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/platform_device.h>
#include <linux/mod_devicetable.h>
#include <linux/of.h>
#include <linux/gpio/consumer.h>
#include <linux/gpio.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/pwm.h>
#include <linux/delay.h>
#include <linux/iio/consumer.h>


#define MYDEVICE_NAME "myhumiture"
```

```
#define SET_HEATING     _IOW('myhumiture', 0, int)
#define SET_HUMIDIFYING _IOW('myhumiture', 1, int)
#define SET_FAN         _IOW('myhumiture', 2, int)
#define SET_PUMP        _IOW('myhumiture', 3, int)

#define GET_TEMPERATURE _IOR('myhumiture', 0, int)
#define GET_HUMIDITY _IOR('myhumiture', 1, int)
#define GET_WATER_LEVEL _IOR('myhumiture', 2, int)


int ret;
unsigned int rec_data[5];// dht11数据
int temp_temperature = -1;// 温度暂存

// dht11 函数声明
void dht11_reset(void);
unsigned char dht11_readbit(void);
int dht11_readbyte(void);
int dht11_readHumidityAndTemp(void);

// 字符设备函数声明
static int mydevice_open(struct inode *inode, struct file *file);
static int mydevice_release(struct inode *inode, struct file *file);
static ssize_t mydevice_read(struct file *file, char __user *buf, size_t count,
loff_t *ppos);
static ssize_t mydevice_write(struct file *file, const char __user *buf, size_t
count, loff_t *ppos);
static long mydevice_ioctl(struct file *file, unsigned int cmd, unsigned long
arg);

// 字符设备相关
static dev_t mydevice_dev;
static struct cdev mydevice_cdev;
static struct class *mydevice_class;
static struct file_operations mydevice_fops = {
    .open = mydevice_open,
    .release = mydevice_release,
    .read = mydevice_read,
    .write = mydevice_write,
    .unlocked_ioctl = mydevice_ioctl,
};

// gpio描述符
struct gpio_desc *gpio_desc_dht11;          // dht11
struct gpio_desc *gpio_desc_Fan;            // 风扇
struct gpio_desc *gpio_desc_pump;           // 水泵
struct gpio_desc *gpio_desc_heating;        // 加热
struct gpio_desc *gpio_desc_humidifying;    // 加湿
// adc
struct iio_channel *water_level;            // 水位

// 获取dht11温度
int get_dht11_temperature(void) {
    int temp;
    //printk("get_dht11_temperature!\n");
    temp = rec_data[2];
```

```
    return temp;
}
//获取dht11湿度
int get_dht11_humidity(void) {
    int humidity;
    //printk("get_dht11_humidity!\n");
    humidity = dht11_readHumidityAndTemp();
    return humidity;
}

static int mydevice_open(struct inode *inode, struct file *file)
{
    printk("myhumiture_open!\n");
    return 0;
}
static int mydevice_release(struct inode *inode, struct file *file)
{
    printk("myhumiture_release!\n");
    return 0;
}
static ssize_t mydevice_read(struct file *file, char __user *buf, size_t count,
loff_t *ppos)
{
    return 0;
}
static ssize_t mydevice_write(struct file *file, const char __user *buf, size_t
count, loff_t *ppos)
{
    return 0;
}
static long mydevice_ioctl(struct file *file, unsigned int cmd, unsigned long
arg)
{
    int ret;
    int value;
    ret = arg;
    switch (cmd) {
        // 加热
        case SET_HEATING:
            if (ret == 0)
            {
                printk("tem_stop\n");
                gpiod_set_value(gpio_desc_heating,0);
            }else if (ret == 1)
            {
                printk("tem_up\n");
                gpiod_set_value(gpio_desc_heating,1);
            }
            break;
        // 加湿
        case SET_HUMIDIFYING:
            if (ret == 0)
            {
                printk("hum_stop\n");
                gpiod_set_value(gpio_desc_humidifying,0);
            }else if (ret == 1)
```

```
            {
                printk("hum_up\n");
                gpiod_set_value(gpio_desc_humidifying,1);
            }
            break;
        // 风扇
        case SET_FAN:
            if (ret == 0)
            {
                printk("fan_off\n");
                gpiod_set_value(gpio_desc_Fan,0);
                printk("fan value is %d",gpiod_get_value(gpio_desc_Fan));
            }else if (ret == 1)
            {
                printk("fan_on\n");
                gpiod_set_value(gpio_desc_Fan,1);
                printk("fan value is %d",gpiod_get_value(gpio_desc_Fan));
            }
            break;
        // 水泵
        case SET_PUMP:
            if (ret == 0)
            {
                printk("pump_off\n");
                gpiod_set_value(gpio_desc_pump,0);
            }else if (ret == 1)
            {
                printk("pump_on\n");
                gpiod_set_value(gpio_desc_pump,1);
            }
            break;

        // 返回温度
        case GET_TEMPERATURE:
            ret = get_dht11_temperature();
            if (copy_to_user((int __user *)arg, &ret, sizeof(ret))) {
                return -EFAULT;
            }
            break;

        // 返回湿度
        case GET_HUMIDITY:
            ret = get_dht11_humidity();
            if (copy_to_user((int __user *)arg, &ret, sizeof(ret))) {
                return -EFAULT;
            }
            break;

        //返回水位
        case GET_WATER_LEVEL:

           iio_read_channel_raw(water_level, &value);
            // 转换成百分比，0-1024 映射到 0-100
            printk("ADC原始值: %d\n", value);
            // 确保值在有效范围内
            if (value < 0) value = 0;
```

```
            if (value > 1024) value = 1024;
            // 进行映射转换
            value = (value * 100) / 1024;

            printk("水位: %d%%\n", value);

            if (copy_to_user((int __user *)arg, &value, sizeof(value))) {
                return -EFAULT;
            }
            break;

        default:
            break;
    }
    return 0;
}


int mydriver_probe(struct platform_device *pdev)
{
    printk("mydriver_probe!\n");

    // 注册字符设备（带错误处理）
    ret = alloc_chrdev_region(&mydevice_dev, 0, 1, MYDEVICE_NAME);
    if (ret < 0) {
        printk("Failed to allocate chrdev\n");
        return ret;
    }
    // 初始化字符设备
    cdev_init(&mydevice_cdev, &mydevice_fops);
    ret = cdev_add(&mydevice_cdev, mydevice_dev, 1);
    if (ret < 0) {
        printk("Failed to add cdev\n");
        goto chrdev_err;
    }
    // 创建设备类（带错误处理）
    mydevice_class = class_create(THIS_MODULE, MYDEVICE_NAME);
    if (IS_ERR(mydevice_class)) {
        ret = PTR_ERR(mydevice_class);
        printk("Failed to create class\n");
        goto class_err;
    }
    device_create(mydevice_class, NULL, mydevice_dev, NULL, MYDEVICE_NAME);

    struct device_node *humiture_node;
    struct device_node *child;
    int ret;

    // 获取设备树节点
    humiture_node = pdev->dev.of_node;
    if (!humiture_node) {
        printk("Failed to find humiture_node\n");
        return -ENODEV;
    }

    // 遍历子节点，打印GPIO节点信息，初始化
```

```
    for_each_child_of_node(humiture_node, child) {

        // 打印子节点名称
        printk("Child node: %s\n", child->name);

        // dht11
        if (of_device_is_compatible(child, "dht11")) {
            // 获取 dht11-gpios
            gpio_desc_dht11 = devm_gpiod_get_from_of_node(&pdev->dev, child,
"dht11-gpios", 0, GPIOD_OUT_LOW, NULL);

            if (IS_ERR(gpio_desc_dht11)) {
                printk("Get gpio_desc_dht11 failed for %s\n", child->name);
                gpio_desc_dht11 = NULL;
            } else {
                // 设置 GPIO 方向为输出
                gpiod_direction_output(gpio_desc_dht11, 0);
                printk("gpio_desc_dht11 %s get successful\n", child->name);
            }
        }
        //water_level
        if (of_device_is_compatible(child, "water_level")) {
            //  获取水位节点
            water_level = iio_channel_get(&pdev->dev, "adc1");

            if (IS_ERR(water_level)) {
                printk("Get water_level failed for %s\n", child->name);
                water_level = NULL;
            }else{
                printk("water_level %s get successful\n", child->name);
            }
        }
        //Fan
        if(of_device_is_compatible(child, "Fan")){
            // 获取风扇节点
            gpio_desc_Fan = devm_gpiod_get_from_of_node(&pdev->dev, child, "fan-
gpios", 0, GPIOD_OUT_LOW, NULL);

            if (IS_ERR(gpio_desc_Fan)) {
                printk("Get gpio_desc_Fan failed for %s\n", child->name);
                gpio_desc_Fan = NULL;
            } else {
                // 设置 GPIO 方向为输出
                gpiod_direction_output(gpio_desc_Fan, 0);
                printk("value is %d",gpiod_get_value(gpio_desc_Fan));
                printk("gpio_desc_Fan %s get successful\n", child->name);
            }
        }
        //pump
        if(of_device_is_compatible(child, "pump")){
            // 获取水泵节点
            gpio_desc_pump = devm_gpiod_get_from_of_node(&pdev->dev, child, "pump-
gpios", 0, GPIOD_OUT_LOW, NULL);

            if (IS_ERR(gpio_desc_pump)) {
                printk("Get gpio_desc_pump failed for %s\n", child->name);
```

```
                gpio_desc_pump = NULL;
            } else {
                // 设置 GPIO 方向为输出
                gpiod_direction_output(gpio_desc_pump, 0);
                printk("gpio_desc_pump %s get successful\n", child->name);
            }
        }
        //heating
        if(of_device_is_compatible(child, "heating")){
            // 获取加热节点
            gpio_desc_heating = devm_gpiod_get_from_of_node(&pdev->dev, child,
"heating-gpios", 0, GPIOD_OUT_LOW, NULL);

            if (IS_ERR(gpio_desc_heating)) {
                printk("Get gpio_desc_heating failed for %s\n", child->name);
                gpio_desc_heating = NULL;
            }else{
                // 设置 GPIO 方向为输出
                gpiod_direction_output(gpio_desc_heating, 0);
                printk("gpio_desc_heating %s get successful\n", child->name);
            }
        }
        //humidifying
        if(of_device_is_compatible(child, "humidifying")){
            // 获取加湿节点
            gpio_desc_humidifying = devm_gpiod_get_from_of_node(&pdev->dev, child,
"humidifying-gpios", 0, GPIOD_OUT_LOW, NULL);

            if (IS_ERR(gpio_desc_humidifying)) {
                printk("Get gpio_desc_humidifying failed for %s\n", child->name);
                gpio_desc_humidifying = NULL;
            }else{
                //设置 GPIO 方向为输出
                gpiod_direction_output(gpio_desc_humidifying, 0);
                printk("gpio_desc_humidifying %s get successful\n", child->name);
            }
        }
    }
    return 0;

class_err:
    cdev_del(&mydevice_cdev);
chrdev_err:
    unregister_chrdev_region(mydevice_dev, 1);
    return ret;
}

int mydriver_remove(struct platform_device *dev)
{
    printk("mydriver_remove!\n");
    // 卸载字符设备
    device_destroy(mydevice_class, mydevice_dev);
    class_destroy(mydevice_class);
    cdev_del(&mydevice_cdev);
    unregister_chrdev_region(mydevice_dev, 1);
    return 0;
```

```
}

// 平台驱动相关
const struct platform_device_id mydriver_id_table = {
    .name = "myhumiture", // 设备名称，用于匹配特定的设备
};
const struct of_device_id of_match_table_id[] = {
    {
        .compatible = "myhumiture",
    },
    {}
};
struct platform_driver platform_driver_test = {
    .driver = {
        .name = "myhumiture",
        .owner = THIS_MODULE,
        .of_match_table = of_match_table_id,
    },
    .id_table = &mydriver_id_table,
    .probe = mydriver_probe,
    .remove = mydriver_remove,
};

module_platform_driver(platform_driver_test);

/**
 * DHT11传感器复位函数
 *
 * 本函数通过对GPIO引脚的操作，实现DHT11传感器的复位过程
 * 复位操作包括输出高电平、输出低电平和输入状态的转换，以触发传感器的响应
 */
void dht11_reset(void){
    gpiod_direction_output(gpio_desc_dht11, 1);
    udelay(30);
    gpiod_set_value(gpio_desc_dht11, 0);
    mdelay(20);
    gpiod_set_value(gpio_desc_dht11, 1);
    udelay(30);
    gpiod_direction_input(gpio_desc_dht11);
}

/**
 * 从DHT11传感器读取单个数据位
 *
 * 本函数实现DHT11传感器的单比特读取时序，通过检测GPIO引脚电平变化来判断数据位值
 * 遵循DHT11通信协议，包含等待起始信号、测量高电平持续时间等关键步骤
 *
 * @return 读取到的数据位值，返回0或1
 */
unsigned char dht11_readbit(void){
    unsigned char bit = 0;

    // 等待低电平结束（起始信号）
    while(gpiod_get_value(gpio_desc_dht11) == 0);

    // 测量高电平持续时间
```

```
    udelay(30); // 等待30us后检测电平状态

    // 根据高电平持续时间判断数据位值
    if(gpiod_get_value(gpio_desc_dht11) == 1) {
        bit = 1;
    }

    // 等待高电平结束（准备接收下一位）
    while(gpiod_get_value(gpio_desc_dht11) == 1);

    return bit;
}

/**
 * 从DHT11传感器读取完整字节数据
 *
 * 本函数通过连续调用8次dht11_readbit函数读取单个字节，遵循DHT11的串行通信协议
 * 每个字节由8个数据位组成，从最高有效位(MSB)到最低有效位(LSB)依次读取并组合
 *
 * @return 返回读取的完整字节数据，数据类型为整型
 */
int dht11_readbyte(void){
    int data = 0;
    int i;
    // 循环读取8个数据位，组成完整字节
    for(i=0;i<8;i++){
        // 每次读取后左移并组合数据，保持数据位的正确顺序
        data |= dht11_readbit() << (7 - i);  // 高位先接收
    }
    return data;
}

/**
 * 从DHT11传感器读取湿度值
 *
 * 该函数完成DHT11传感器的初始化、数据读取和校验流程，返回湿度百分比值
 *
 * @return 湿度百分比值（0-100），读取失败时返回负数错误码
 */

int dht11_readHumidityAndTemp(void)
{
    int humidity = 0;
    int temperature = 0;
    unsigned char checksum = 0;
    int i;

    // 初始化传感器
    dht11_reset();

    // 等待传感器响应（拉低总线）
    if (gpiod_get_value(gpio_desc_dht11) == 0) {
       // 等待响应信号结束（总线拉高）
        while (gpiod_get_value(gpio_desc_dht11) == 0);
        while (gpiod_get_value(gpio_desc_dht11) == 1);
```

```
        // 读取40位数据（湿度整数、湿度小数、温度整数、温度小数、校验和）
        for (i = 0; i < 5; i++) {
            rec_data[i] = dht11_readbyte();
        }
        // 校验数据有效性
        checksum = rec_data[0] + rec_data[1] + rec_data[2] + rec_data[3];
        if (rec_data[4] != checksum) {
            printk("DHT11 checksum error\n");
        }
        //记录第一次温度
        if(temp_temperature == -1){
            temp_temperature = rec_data[2];
        }

        // 判断温度变化是否超过8度，如果超过则使用上一次的温度值
        if(abs(temp_temperature - rec_data[2]) > 10){
            rec_data[2] = temp_temperature;
        }else{//没有超过就记录温度
            temp_temperature = rec_data[2];
        }

        // 组合湿度数据（DHT11小数位通常为0）
        humidity = rec_data[0]; // 湿度整数部分
        temperature = rec_data[2];// 温度整数部分

        printk("DHT11 humidity: %d\n", humidity);
        printk("DHT11 temperature: %d\n", temperature);
        return humidity;
    }else{
        //使用之前的数据
        return rec_data[0];
    }
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("XYY");
MODULE_VERSION("V0.1");
```

4.应用层

1.手机端

```
package com.example.humiture;

import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

```java
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.util.Locale;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.regex.Pattern;

public class MainActivity extends AppCompatActivity {

    private Button btnHeat;
    private Button btnHumidify;
    private Button btnFan;
    private Button btnPump;
    private EditText ipEditText;
    private EditText portEditText;
    private Button connectButton;
    private TextView temperatureTextView;
    private TextView humidityTextView;
    private TextView waterLevelTextView;


    // 功能按钮状态
    private boolean isHeating = false;
    private boolean isHumidifying = false;
    private boolean isFaning = false;
    private boolean isPumping = false;

    // 网络参数
    private String serverIp;
    private int port;

    // 网络连接
    private Socket clientSocket;
    private PrintWriter writer;

    // 线程管理
    private final ExecutorService executor = Executors.newSingleThreadExecutor();
    private final Handler handler = new Handler(Looper.getMainLooper());
    private final Object connectionLock = new Object();

    // 状态标志（使用原子类型保证线程安全）
    private final AtomicBoolean isConnected = new AtomicBoolean(false);
    private final AtomicBoolean isSending = new AtomicBoolean(false);
```

```
    private long lastSendTime = 0;
    private final AtomicBoolean isReceiving = new AtomicBoolean(false);

    // 读取服务器的响应
    private BufferedReader reader;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);// 加载布局文件
        setupViews();// 初始化控件
        setupWindowInsets();// 设置窗口 insets
        ClickListener();// 设置点击事件监听器
    }

    private void setupViews() {// 初始化控件
        btnHeat = findViewById(R.id.btnHeat);
        btnHumidify = findViewById(R.id.btnHumidify);
        btnFan = findViewById(R.id.btnFan);
        btnPump = findViewById(R.id.btnPump);
        connectButton = findViewById(R.id.btnConnect);
        ipEditText = findViewById(R.id.etIp);
        portEditText = findViewById(R.id.etPort);
        temperatureTextView = findViewById(R.id.tvTemperature);
        humidityTextView = findViewById(R.id.tvHumidity);
        waterLevelTextView = findViewById(R.id.tvWaterLevel);
    }

    private void setupWindowInsets() {// 设置窗口 insets
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });
    }

    // 设置点击事件监听器，处理点击事件
    private void ClickListener() {
        connectButton.setOnClickListener(v -> handleConnectClick());//处理连接的点击
事件
        btnHeat.setOnClickListener(v -> handeClick(v)); // 处理功能点击事件
        btnHumidify.setOnClickListener(v -> handeClick(v));
        btnFan.setOnClickListener(v -> handeClick(v));
        btnPump.setOnClickListener(v -> handeClick(v));
    }

    private void handeClick(View v) {
    if (isConnected.get()){// 如果已经连接，才执行点击事件
        if (v.getId() == R.id.btnHeat) {
            if (isHeating) {
                isHeating = false;
```

```
                btnHeat.setText("加热");
                btnHeat.setBackgroundResource(R.drawable.bg);
                sendData("tem_stop"); // 发送停止加热指令
            } else {
                isHeating = true;
                btnHeat.setText("加热中...");
                btnHeat.setBackgroundResource(R.drawable.bg_disconnect);
                sendData("tem_up"); // 发送开始加热指令
            }
        } else if (v.getId() == R.id.btnHumidify) {
            if (isHumidifying) {
                isHumidifying = false;
                btnHumidify.setText("加湿");
                btnHumidify.setBackgroundResource(R.drawable.bg);
                sendData("hum_stop"); // 发送停止加湿指令
            } else {
                isHumidifying = true;
                btnHumidify.setText("加湿中...");
                btnHumidify.setBackgroundResource(R.drawable.bg_disconnect);
                sendData("hum_up"); // 发送开始加湿指令
            }
        }else if (v.getId() == R.id.btnFan) {
            if (isFaning){
                isFaning = false;
                btnFan.setText("风扇");
                btnFan.setBackgroundResource(R.drawable.bg);
                sendData("fan_off");// 发送关闭风扇指令
            }else{
                isFaning = true;
                btnFan.setText("散热中...");
                btnFan.setBackgroundResource(R.drawable.bg_disconnect);
                sendData("fan_on");// 发送开启风扇指令
            }
        }else if (v.getId() == R.id.btnPump) {
            if (isPumping){
                isPumping = false;
                btnPump.setText("水泵");
                btnPump.setBackgroundResource(R.drawable.bg);
                sendData("pump_off");// 发送关闭水泵指令
            }else{
                isPumping = true;
                btnPump.setText("抽水中...");
                btnPump.setBackgroundResource(R.drawable.bg_disconnect);
                sendData("pump_on");// 发送开启水泵指令
            }
        }
    }else{
        showToast("请先连接服务器");
    }
}

    /**
     * 处理连接或断开连接点击事件
     * 此方法旨在验证用户输入的IP和端口，并尝试连接到指定的服务器，或断开当前的连接
     */
    private void handleConnectClick() {
```

```
        if (isConnected.get()) {
            // 如果已经连接，则断开连接
            disconnectFromServer();
        } else {
            // 如果未连接，则尝试连接到服务器
            // 获取并修剪IP输入框中的内容
            String ip = ipEditText.getText().toString().trim();
            // 获取并修剪端口输入框中的内容
            String portStr = portEditText.getText().toString().trim();

            // 检查IP和端口是否都已输入，如果任一为空，则提示用户并返回
            if (ip.isEmpty() || portStr.isEmpty()) {
                showToast("请输入IP和端口");
                return;
            }
            // 验证IP地址格式，如果不正确，则提示用户并返回
            if (!isValidIp(ip)) {
                showToast("IP地址格式错误！");
                return;
            }
            try {
                // 将端口字符串转换为整数
                int port = Integer.parseInt(portStr);
                // 检查端口号是否在有效范围内（1到65535），如果不在，则提示用户并返回
                if (port < 1 || port > 65535) {
                    showToast("端口不存在！");
                    return;
                }
                // 如果IP和端口都有效，则设置服务器IP和端口为输入的值
                this.serverIp = ip;
                this.port = port;
                // 禁用连接按钮以防止重复点击
                setConnectButtonState(false);
                // 尝试连接到服务器
                connectToServer();
                // 显示连接中的提示信息
                showToast("连接中...");
            } catch (NumberFormatException e) {
                // 如果端口转换为整数时发生错误，则提示用户
                showToast("端口错误！");
            }
        }
    }


/***********************************客户端***********************************/
    // region 网络连接管理
    /**
     * 连接到服务器的方法
     * 该方法在一个单独的线程中执行，以避免网络操作阻塞主线程
     * 使用同步块来确保连接的互斥性，防止多个线程同时修改连接状态
     */
    private void connectToServer() {
        executor.execute(() -> {
            synchronized (connectionLock) {
                try {
```

```
                    // 在尝试新连接之前，关闭任何现有的连接
                    closeConnection();

                    // 创建到服务器的新Socket连接
                    clientSocket = new Socket(serverIp, port);
                    // 初始化用于向服务器发送数据的PrintWriter
                    writer = new PrintWriter(clientSocket.getOutputStream(),
true);
                    // 初始化用于从服务器接收数据的BufferedReader
                    reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                    // 更新应用内部的连接状态并显示连接成功的通知
                    updateConnectionStatus(true);
                    showToast("连接成功");

                    // 设置接收标志为true
                    isReceiving.set(true);
                    // 启动数据接收线程
                    startReceivingData();

                } catch (IOException e) {
                    // 如果连接失败，更新连接状态并显示错误信息
                    updateConnectionStatus(false);
                    showToast("连接失败: " + e.getMessage());
                } finally {
                    // 重新启用连接按钮，无论连接成功还是失败
                    setConnectButtonState(true);
                    // 通知其他等待连接锁的线程
                    connectionLock.notifyAll();
                }
            }
        });
    }

    /**
     * 断开与服务器的连接
     * 该方法在一个单独的线程中执行，以避免网络操作阻塞主线程
     * 使用同步块来确保连接的互斥性，防止多个线程同时修改连接状态
     */
    private void disconnectFromServer() {
        executor.execute(() -> {
            synchronized (connectionLock) {
                try {
                    isReceiving.set(false); // 停止接收线程
                    // 关闭当前的连接
                    closeConnection();

                    // 更新应用内部的连接状态并显示断开连接的通知
                    updateConnectionStatus(false);
                    showToast("已断开连接");
                } catch (Exception e) {
                    // 如果断开连接时发生错误，更新连接状态并显示错误信息
                    updateConnectionStatus(false);
                    showToast("断开连接失败: " + e.getMessage());
                } finally {
                    // 重新启用连接按钮
```

```
                    setConnectButtonState(true);
                    // 通知其他等待连接锁的线程
                    connectionLock.notifyAll();
                }
            }
        });
    }

    /**
     * 发送数据到远程服务器
     * 该方法主要负责将信息打包成特定格式的字符串，并通过网络发送出去
     *
     * @param data 要发送的指
令,"tem_up"、"tem_stop"、"hum_up"、"hum_stop"、"pump_on"、"pump_off"、"fan_on"、"fan
_off"
     */
    private void sendData(String data) {
        // 发送频率限制（50ms）
        if (System.currentTimeMillis() - lastSendTime < 50) return;
        lastSendTime = System.currentTimeMillis();

        // 避免重复发送
        if (isSending.get()) return;
        isSending.set(true);

        // 使用线程池执行发送任务，避免阻塞当前线程
        executor.execute(() -> {
            synchronized (connectionLock) {
                try {
                    // 确保连接有效，否则尝试重新连接
                    if (!isConnectionValid()) {
                        connectToServer();
                        connectionLock.wait(1000);
                    }

                    // 发送数据
                    if (writer != null) {
                        String message = String.format(Locale.US,
                                "%s",data);
                        writer.println(message);
                        //打印log
                        Log.d("MainActivity", "Sent: " + message);
                    }
                } catch (Exception e) {
                    // 异常处理
                    handleSendError(e);
                } finally {
                    // 重置发送状态标志
                    isSending.set(false);
                }
            }
        });
    }
    // endregion
/*********************************************************************************
*********/
```

```
/*********************************************服务端
*********************************************/

    /**
     * 开始接收数据
     * 此方法启动一个新任务来接收客户端套接字的数据
     * 它读取数据直到线程被中断或套接字断开连接
     */
private void startReceivingData() {
    new Thread(() -> {
        while (isReceiving.get()) {
            Log.d("Network", "数据接收线程启动");
            try {
                // 检查连接是否有效
                if (clientSocket == null || clientSocket.isClosed()) {
                    Log.e("Network", "连接未建立或已关闭");
                    handleConnectionLost("连接已断开，请检查网络后重新连接");
                    break;  // 退出接收循环
                }

                String data;
                while (isReceiving.get() && (data = reader.readLine()) != null) {
                    Log.d("Network", "收到原始数据: " + data);
                    parseSensorData(data);
                }
            } catch (IOException e) {
                Log.e("Network", "接收数据失败: " + e.getMessage());
                handleConnectionLost("连接异常: " + e.getMessage());
                break;  // 退出接收循环
            }
        }
    }).start();
}

    /**
     * 解析传感器数据字符串
     * 此方法旨在处理格式化的传感器数据字符串，提取温度和湿度信息，并更新UI显示
     * 预期的数据格式为"Temperature:XX,Humidity:YY"，其中XX和YY是整数值
     *
     * @param data 格式化的传感器数据字符串
     */
    private void parseSensorData(String data) {
    try {
        // 格式示例: Temperature:25,Humidity:50,WaterLevel:75
        String[] parts = data.split(",");
        if (parts.length >= 3) { // 确保有三个数据段
            int temperature = Integer.parseInt(parts[0].split(":")[1].trim());
            int humidity = Integer.parseInt(parts[1].split(":")[1].trim());
            int waterLevel = Integer.parseInt(parts[2].split(":")[1].trim());

            Log.d("Parse", String.format("解析数据: %d°C, %d%%, 水位 %d%%",
                temperature, humidity, waterLevel));

            // 更新UI（需要先在布局中添加水位显示控件）
            runOnUiThread(() -> {
```

```
                temperatureTextView.setText(String.format("温度: %d°C",
temperature));
                humidityTextView.setText(String.format("湿度: %d%%", humidity));
                waterLevelTextView.setText(String.format("水位: %d%%",
waterLevel));
            });
        }
    } catch (Exception e) {
        Log.e("Parse", "数据解析失败: " + data, e);
        // 可选: 显示错误状态的UI
        runOnUiThread(() -> {
            temperatureTextView.setText("温度: --");
            humidityTextView.setText("湿度: --");
            waterLevelTextView.setText("水位: --");
        });
    }
}

    /**
     * 处理连接丢失的情况
     * @param message 要显示给用户的消息
     */
    private void handleConnectionLost(String message) {
        // 确保停止接收
        isReceiving.set(false);

        // 关闭连接
        closeConnection();

        // 更新UI状态
        updateConnectionStatus(false);

        // 重置按钮状态
        setConnectButtonState(true);

        // 通知用户
        showToast(message);

        // 如果正在加热或加湿，重置这些状态
        runOnUiThread(() -> {
            if (isHeating) {
                isHeating = false;
                btnHeat.setText("加热");
                btnHeat.setBackgroundResource(R.drawable.bg);
            }
            if (isHumidifying) {
                isHumidifying = false;
                btnHumidify.setText("加湿");
                btnHumidify.setBackgroundResource(R.drawable.bg);
            }
            if (isFaning) {
                isFaning = false;
                btnFan.setText("风扇");
                btnFan.setBackgroundResource(R.drawable.bg);
            }
            if (isPumping) {
```

```
                isPumping = false;
                btnPump.setText("水泵");
                btnPump.setBackgroundResource(R.drawable.bg);
            }
        });
    }

// endregion

// 其他部分保持不变

// endregion


    // region 工具方法
    /**
     * 处理发送错误的方法
     * 当发送操作失败时调用此方法，它执行以下操作：
     * 1. 显示一个toast消息，提示发送失败的原因（异常消息）
     * 2. 关闭连接
     * 3. 更新连接状态为未连接
     *
     * @param e 异常对象，包含发送失败的原因
     */
    private void handleSendError(Exception e) {
        showToast("发送失败: " + e.getMessage());
        closeConnection();
        updateConnectionStatus(false);
    }

    /**
     * 检查当前客户端套接字的连接是否有效
     *
     * 此方法通过一系列条件判断来确定客户端与服务器之间的连接状态是否正常
     * 它首先确保clientSocket对象不为空，以防止空指针异常
     * 接着检查套接字是否已连接，且没有被关闭，以及输入输出流是否都处于正常工作状态
     * 只有当所有这些条件都满足时，才认为连接是有效的
     *
     * @return boolean 表示连接是否有效的布尔值如果连接有效则返回true，否则返回false
     */
    private boolean isConnectionValid() {
        return clientSocket != null
                && clientSocket.isConnected()// 确保套接字已连接
                && !clientSocket.isClosed()// 确保套接字没有被关闭
                && !clientSocket.isInputShutdown()// 确保输入流处于正常工作状态
                && !clientSocket.isOutputShutdown();// 确保输出流处于正常工作状态
    }

    /**
     * 关闭连接资源
     * 此方法旨在确保在不再需要时正确关闭连接和流，以防止资源泄露和潜在的连接问题
     */
    private void closeConnection() {
        try {
            isReceiving.set(false); // 确保接收线程停止
            if (reader != null) {
```

```
                reader.close();
                reader = null;
            }
            if (writer != null) {
                writer.close();
                writer = null;
            }
            if (clientSocket != null) {
                clientSocket.close();
                clientSocket = null;
            }
        } catch (IOException e) {
            Log.e("Network", "关闭连接时出错", e);
        }
    }

    /**
     * 更新应用界面的连接状态
     * 此方法根据连接状态更新应用界面，包括连接按钮的文本和背景颜色
     *
     * @param connected 一个布尔值，表示当前的连接状态 true表示已连接，false表示未连接
     */
    private void updateConnectionStatus(boolean connected) {
        // 更新内部状态变量以反映当前的连接状态
        isConnected.set(connected);

        // 在主线程中运行UI更新操作，以确保线程安全
        runOnUiThread(() -> {
            // 根据连接状态更新连接按钮的文本
            connectButton.setText(connected ? "断开连接" : "连接");

            // 根据连接状态更新连接按钮的背景颜色
            if (connected){//如果连接成功，设置背景为红色
                connectButton.setBackgroundResource(R.drawable.bg_disconnect);
            }else{//如果断开连接,恢复背景bg.xml
                connectButton.setBackgroundResource(R.drawable.bg);
                // 更新温度和湿度的文本
                temperatureTextView.setText("温度: --°C");
                humidityTextView.setText("湿度: --%");
                waterLevelTextView.setText("水位: --%");
            }
        });
    }

    /**
     * 设置连接按钮的状态
     * 此方法通过在主线程中运行来更新UI，确保按钮的状态更改在主线程中执行，
     * 以避免多线程环境下对UI组件进行操作时可能出现的问题
     *
     * @param enabled 指定按钮是否启用 true表示按钮将被启用，false表示按钮将被禁用
     */
    private void setConnectButtonState(boolean enabled) {
        runOnUiThread(() -> connectButton.setEnabled(enabled));
    }

    /**
```

```
     * 在主线程中显示短时间的Toast消息
     *
     * @param text 要显示的Toast消息文本
     */
    private void showToast(String text) {
        handler.post(() -> Toast.makeText(this, text,
Toast.LENGTH_SHORT).show());
    }

    /**
     * 验证输入的字符串是否为有效的IPv4地址
     * 该方法使用正则表达式来匹配IPv4地址的格式
     *
     * @param ip 要验证的IP地址字符串
     * @return 如果输入的字符串是有效的IPv4地址，则返回true；否则返回false
     */
    private boolean isValidIp(String ip) {
        // 编译一个正则表达式，用于匹配IPv4地址的格式
        Pattern pattern = Pattern.compile(
                "^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)$");
        // 使用编译的正则表达式创建一个匹配器，并验证输入的IP地址是否匹配IPv4地址的格式
        return pattern.matcher(ip).matches();
    }
    // endregion
    @Override
    protected void onDestroy() {
        super.onDestroy();
    }
}
```

2.开发板端

```
package com.example.humiturejni;

import static android.content.ContentValues.TAG;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.util.Log;

import com.example.humiturejni.databinding.ActivityMainBinding;

import java.io.BufferedReader;
import java.io.IOException;

import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Locale;
```

```
public class MainActivity extends AppCompatActivity {

    private Handler handler = new Handler(Looper.getMainLooper());// 用于在主线程中
更新UI
    private Socket client = null;

    // 定义日志标签
    private static final String TAG = "TempHumiditySender";

    // Used to load the 'humiturejni' library on application startup.
    static {
        System.loadLibrary("humiturejni");
    }

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        startServer();//启动服务器
        MyDeviceOpen();//打开设备
        updateTempHumidity();//更新显示
    }

    // 更新显示的Runnable任务
    private Runnable updateRunnable = new Runnable() {
        @Override
        public void run() {
            new Thread(() -> {
                try {
                    int temperature = getTemperature();
                    int humidity = getHumidity();
                    int waterLevel = getWaterLevel();

                    handler.post(() -> {

 binding.tvTemperature.setText(String.format(Locale.getDefault(), "%d°C",
temperature));

 binding.tvHumidity.setText(String.format(Locale.getDefault(), "%d%%",
humidity));

 binding.tvWaterLevel.setText(String.format(Locale.getDefault(), "%d%%",
waterLevel));
                    });

                    sendTempHumidityToClient();
                } catch (Exception e) {
                    Log.e(TAG, "Error updating data", e);
```

```
                }
            }).start();

            handler.postDelayed(this, 2000);
        }
    };


    // 更新温湿度显示
    private void updateTempHumidity() {
        handler.post(updateRunnable);
    }


    /*****************************************服务端
*************************************************/
    /**
     * 启动服务器
     * 该方法在一个新的线程中启动一个服务器，监听指定端口，并接受客户端连接请求
     * 选择在新线程中运行是因为服务器通常需要长时间运行，且处理客户端请求时不应阻塞主线程
     * 使用ServerSocket来监听8888端口，这是服务器与客户端通信的入口
     * 该方法不接受参数，也无返回值
     */
    private void startServer() {
        new Thread(() -> {
            try {
                ServerSocket serverSocket = new ServerSocket(8888);
                while (!Thread.interrupted()) {
                    Socket newClient = serverSocket.accept();
                    // 关闭旧连接并保存新客户端
                    synchronized (this) {
                        if (client != null && !client.isClosed()) {
                            client.close();
                        }
                        client = newClient;
                    }
                    handleClient(newClient); // 处理新客户端
                }
            } catch (IOException e) {
                Log.e(TAG, "Server error: " + e.getMessage());
            }
        }).start();
    }

    /**
     * 处理客户端请求
     * 当客户端连接到服务器时，此方法将被调用以处理客户端的请求
     * 它读取客户端发送的每一行指令，并将其解析和执行
     *
     * @param client 代表客户端的Socket对象
     */
    private void handleClient(Socket client) {
        // 为每个客户端启动一个新的线程，以实现并发处理
        new Thread(() -> {
            try (BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()))) {
```

```
                String inputLine;
                // 持续监听客户端指令
                while ((inputLine = in.readLine()) != null) {
                    // 当收到指令时，记录日志
                    Log.d(TAG, "收到指令: " + inputLine);
                    // 解析并执行接收到的指令
                    parseAndExecuteCommand(inputLine.trim());
                }
            } catch (IOException e) {
                // 如果发生IO异常，记录错误日志
                Log.e(TAG, "客户端断开连接: " + e.getMessage());
            } finally {
                // 确保最终关闭客户端连接
                try {
                    client.close();
                } catch (IOException e) {
                    // 如果关闭连接时发生异常，记录错误日志
                    Log.e(TAG, "关闭连接失败: " + e.getMessage());
                }
            }
        }).start();
    }


    /**
     * 解析并执行客户端发送的指令
     * @param command 客户端发送的指令字符串
     */
    private void parseAndExecuteCommand(String command) {
        command = command.trim().toLowerCase(); // 转为小写并去除空格
        if ("tem_stop".equals(command)) {
            // 停止加热
            Log.d(TAG, "停止加热");
            CmdIoctl(0, 0);
        } else if ("tem_up".equals(command)) {
            // 开始加热
            Log.d(TAG, "开始加热");
            CmdIoctl(0, 1);
        } else if ("hum_stop".equals(command)) {
            // 停止加湿
            Log.d(TAG, "停止加湿");
            CmdIoctl(1, 0);
        } else if ("hum_up".equals(command)) {
            // 开始加湿
            Log.d(TAG, "开始加湿");
            CmdIoctl(1, 1);
        }else if ("fan_off".equals(command)){
            Log.d(TAG, "关闭风扇");
            CmdIoctl(2, 0);
        }else if ("fan_on".equals(command)) {
            //  开启风扇
            Log.d(TAG, "开启风扇");
            CmdIoctl(2, 1);
        }else if ("pump_off".equals(command)){
            Log.d(TAG, "关闭水泵");
            CmdIoctl(3, 0);
```

```
        }else if ("pump_on".equals(command)){
            Log.d(TAG, "开启水泵");
            CmdIoctl(3, 1);
        }
    }
/*****************************************************************************************
******/

/****************************************客户端
*************************************************/

    /**
     * 将当前的温度、湿度、水位发送给客户端
     * 此方法在一个新的线程中执行，以避免阻塞主线程
     * 需要同步以确保在同一时间只有一个线程可以发送数据
     */
    private void sendTempHumidityToClient() {
        // 创建并启动一个新的线程，以异步方式发送数据
        new Thread(() -> {
            // 同步代码块，确保线程安全
            synchronized (this) {
                try {
                    // 检查客户端连接是否处于活动状态
                    if (client == null || client.isClosed()) {
                        Log.e(TAG, "No active client connection");
                        return;
                    }
                    // 准备输出流以发送数据
                    PrintWriter out = new PrintWriter(client.getOutputStream(),
true);
                    // 获取当前的温度和湿度
                    int temperature = getTemperature();
                    int humidity = getHumidity();
                    int waterLevel = getWaterLevel();
                    // 格式化信息
                    String message = String.format(Locale.getDefault(),
"Temperature:%d,Humidity:%d,WaterLevel:%d", temperature, humidity,waterLevel);
                    // 发送格式化后的数据
                    out.println(message);
                    Log.d(TAG, "Data sent: " + message);
                } catch (IOException e) {
                    Log.e(TAG, "Failed to send data: " + e.getMessage());
                }
            }
        }).start();
    }


    @Override
    protected void onDestroy() {
        MyDeviceClose();
        super.onDestroy();
    }

    /**
     * A native method that is implemented by the 'humiturejni' native library,
```

```
     * which is packaged with this application.
     */
    public native String MyDeviceOpen();
    public native String MyDeviceClose();
    public native void CmdIoctl(int cmd, int arg);

    public native int getTemperature();
    public native int getHumidity();
    public native int getWaterLevel();
}
```

```
#include <jni.h>
#include <string>
#include <android/log.h>
#include <asm-generic/fcntl.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define SET_HEATING     _IOW('myhumiture', 0, int)
#define SET_HUMIDIFYING _IOW('myhumiture', 1, int)
#define SET_FAN         _IOW('myhumiture', 2, int)
#define SET_PUMP        _IOW('myhumiture', 3, int)

#define GET_TEMPERATURE _IOR('myhumiture', 0, int)
#define GET_HUMIDITY _IOR('myhumiture', 1, int)
#define GET_WATER_LEVEL _IOR('myhumiture', 2, int)

int fd;
extern "C" JNIEXPORT jstring JNICALL
Java_com_example_humiturejni_MainActivity_MyDeviceOpen(
        JNIEnv* env,
        jobject /* this */) {
    fd = open("/dev/myhumiture", O_RDWR | O_NDELAY | O_NOCTTY);

    if (fd < 0) {
        __android_log_print(ANDROID_LOG_INFO, "serial", "open error");
    }else {
        __android_log_print(ANDROID_LOG_INFO, "serial", "open success fd=%d",fd);
    }
    return 0;
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_humiturejni_MainActivity_MyDeviceClose(
        JNIEnv* env,
        jobject /* this */) {
    if (fd != -1) {
        __android_log_print(ANDROID_LOG_INFO, "serial", "Closing device fd=%d",
fd); // 添加关闭日志
        close(fd);
        fd = -1;
    }
```

```
    return 0;
}

extern "C" JNIEXPORT void JNICALL
Java_com_example_humiturejni_MainActivity_CmdIoctl(JNIEnv *env, jobject, jint cmd,
jint arg) {
    if(cmd == 0){
        ioctl(fd, SET_HEATING, arg);
    }else if(cmd == 1){
        ioctl(fd, SET_HUMIDIFYING, arg);
    }else if(cmd == 2){
        ioctl(fd, SET_FAN, arg);
    }else if(cmd == 3){
        ioctl(fd, SET_PUMP, arg);
    }
}

extern "C" JNIEXPORT jint JNICALL
Java_com_example_humiturejni_MainActivity_getTemperature(JNIEnv *env, jobject) {
    int temperature;
    ioctl(fd, GET_TEMPERATURE, &temperature);
    __android_log_print(ANDROID_LOG_INFO, "serial", "c_temperature=%d",
temperature);
    return temperature;
}

extern "C" JNIEXPORT jint JNICALL
Java_com_example_humiturejni_MainActivity_getHumidity(JNIEnv *env, jobject) {
    int humidity;
    ioctl(fd, GET_HUMIDITY, &humidity);
    __android_log_print(ANDROID_LOG_INFO, "serial", "c_humidity=%d", humidity);
    return humidity;
}

extern "C" JNIEXPORT jint JNICALL
Java_com_example_humiturejni_MainActivity_getWaterLevel(JNIEnv *env, jobject) {
    int water_level;
    ioctl(fd, GET_WATER_LEVEL, &water_level);
    __android_log_print(ANDROID_LOG_INFO, "serial", "c_water_level=%d",
water_level);
    return water_level;
}
```