1.1.1 项目名称

可通过wifi远程控制的灯及传感器软件部分

1.1.2 项目开发人员

李井瑞

1.1.3 项目开发环境

`Intellij IDEA` + `NodeJS` + `NodeMCU` + `Lua`

1.1.4 系统功能设定

软件部分分为3大模块, `Gateway`部分, 灯泡部分, 传感器部分：

1.`Gateway`部分

> 用户控制接口，云端交互，数据记录相关功能：
> 这部分由NodeJS写成，用json与灯泡和传感器进行交互。并开放与用户交互的接口。

`LampServer.js`

该模块通过监听Gateway的18323端口, 获得灯泡的连接, 引入该模块后, 获取接入的灯泡实例, 并为特定灯泡添加事件监听。

引入该模块：

```
const lamps = require('./LampServer')
```

`lamps`为灯泡列表, 底层为`JavaScript`数组

获取灯泡实例的方法:

1.遍历数组

```
for(let item of lamps){
    //your code
}
```

2.get方法

```
let lamp = lamps.get(id:string)
//your code
```

3.监听器

```
lamps.onLogin = (msg: Object, lamp: Object) => {
    //your code
}
```

类似的监听器还有五个, 分别是:

监听器名称	触发事件
onResult	当收到某一灯泡返回命令结果时
onWarning	当收到某一灯泡低电压警告时
onMessage	当收到任何消息时
onError	当收到某一灯泡出现错误时

注意: lamps对象的监听器只有一个, 后设定的监听器会替换前面的监听器

也可以为特定灯泡添加事件监听, 事件只有特定灯泡触发:
(一个灯泡可以有多个监听器, 依次被调用)

```
lamp.addListener('eventType', function(msg: Object){
    //your code
})
```

eventType共有四种,分别是: message, info, warning, error

- 注意: eventType并不包含result, result消息的处理应当写在相关API的"callback"中

Lamp类:

```
Lamp :
{
    Id: 灯的ID,
    Birghtness: 白灯亮度值,
    Blue: 蓝灯色调值,
    Red: 红灯色调值,
    Green: 绿灯色调值,
    IP: 灯的网络IP,
```

```
    Position: 灯的位置,
    Name: 灯的名字,
    Voltage:灯的电压
}
```

相关API:

```
lamp.setPosition(postion:string, [function callback(result){})
lamp.setName(name:string, [function callback(result){})
lamp.setBrightness(brightness:number,[function callback(result){})
lamp.setColor(red:number, green:number, blue:number,[function callback(result){})
```

SensorServer.js

传感器服务通过监听18324端口, 进行与传感器的交互。相关监听器处理方式与LampServer相似。

相关API:

```
sensor.setPosition(postion:string, [function callback(result){})
sensor.setName(name:string, [function callback(result){})
sensor.setInterval(interval:number, [function callback(result){})
```

2.灯泡部分

wifi灯泡的软件部分主要架构为:

```
-src
|--INIT_DATA
|--init.lua
|--constants.lua
|--LampInit.lua
|--LampSocket.lua
```

其在NodeMCU的运行顺序同上.

1. INIT_DATA

该文件储存灯泡的初始化信息, 并且在用户设置亮度、位置和名字信息后进行持久化保存。
该文件只有一行, 而且为标准的JSON格式, id为"ABCD"灯泡的INIT_DATA示例:

```
{"blue":0,"id":"ABCD","position":"unknown","userLevel":0,"name":"unknown","green":
0,"IP":"null","brightness":500,"red":0,"voltage":3.153}
```

- 其中的voltage属性会在通电后自动获取, IP属性会在连接Wifi后自动获取

2. `init.lua`

该文件是NodeMCU启动后自动运行的文件。负责调用其它文件。

3. `constants.lua`

该文件储存灯泡运行的必要信息:

```
* wifi SSID 与 密码
* Gateway IP 与 端口号
* 各LED引脚信息
* 控制按钮引脚信息
* 各用户控制等级 与 PWM占空比 的对应关系
* 低电压报警下限
```

4. `LampInit.lua`

该文件负责各模块初始化工作.
各函数作用： - `initLED`: 初始化LED - `initData`: 初始化持久化数据 - 其他: 初始化按钮控制, wifi连接, 低电压监控

5. `LampSocket.lua`

该文件负责与服务器的交互:

```
当运行该文件后, 启动一个名为skcTimer的计时器, 该计时器负责每3秒尝试连接一次服务器,直到连接成功.

连接服务器成功后, 以`JSON`格式发送`INIT_DATA`数据, 进行登陆.

下为一次正确的登陆信息:
```

```
{"type":"login","blue":0,"id":"ABCD","position":"unknown","userLevel":0,"name":"unknown","green":0,"IP":"192.168.1.1","brightness":500,"red":0,"voltage":3.153}
```

6. 开放API

注意: 硬件端口API只对Gateway开放.

以下为Lua API以及Gateway调用该API发送的JSON示例

```
function getVoltage() end
function setName(string) end
function setBrightness(string) end
function setColor(string) end
function setPosition(string) end
```

以上方法都可以通过发送如下JSON调用:

```
{"cmd": "getVoltage/setName/setBrightness/setColor",
"arg": "***"}
```

调用该api可以发送lua脚本到NodeMCU, 达到在线升级的效果.

```
function script(script) end
```

调用方式:

```
{"cmd": "script", "script": "***"}
```

3.传感器部分

`wifi传感器`的软件部分主要架构为:

```
-src
|--INIT_DATA
|--init.lua
|--constants.lua
|--GY30.lua
|--SensorInit.lua
|--SensorSocket.lua
```

其在NodeMCU的运行顺序同上.

1. **`INIT_DATA`**, **`init.lua`**, **`constans.lua`**

该模块`INIT_DATA`, `init.lua`, `constants.lua`与灯泡模块大体相同, 请参阅上一部分.

2. **`GY30.lua`**

该模块通过I2C接口对GY30传感器进行读取

```
### 主要函数:
```

- `initGY30(sda, scl)`:

  参数:
  sda: sda针脚; scl: scl针脚

  功能:
  初始化传感器

- `readGY30(mode, func)`:
  参数:
  mode: 读取模式; func: 回调函数

  功能:
  读取GY30

3. `SensorInit.lua`

负责初始化数据, 初始化低电压警报, 初始化wifi连接.

4. `SensorSocket.lua`

该文件负责与服务器的交互:

```
当运行该文件后，启动一个名为skcTimer的计时器，该计时器负责每3秒尝试连接一次服务器,直到连接成功.

连接服务器成功后，以`JSON`格式发送`INIT_DATA`数据，进行登陆.

下为一次正确的登陆信息:
```

```
{"type":"login","id":"poiuytrewq","position":"unknown","voltage":0,"name":"unknown","IP":"192.168.1.1","interval":2000}
```

5. 开放API

注意: 硬件API只对Gateway开放.

以下为Lua API以及Gateway调用该API发送的JSON示例

```
function getVoltage() end
function setName(string) end
function setBrightness(string) end
```

```
function setColor(string) end
function setPosition(string) end
```

以上方法都可以通过发送如下JSON调用:

```
{"cmd": "getVoltage/setName/setBrightness/setColor",
"arg": "***"}
```

调用该api可以发送lua脚本到NodeMCU, 达到在线升级的效果.

```
function script(script) end
```

调用方式:

```
{"cmd": "script", "script": "***"}
```

- 每收到一条服务器的命令, 硬件都会返回一条由`{"type":"result"}`扩充而成的消息.