智能灯控后端

使用 GO（Golang）+ Gin 框架开发。使用 TCP 协议，端口 9528 和智能灯硬件通信。用 Http 端口 9527 和前端通信。\可以查看电灯开关状态，电流，故障状态，可以远程控制电灯开和关。

API

和硬件通信使用 TCP 通信。
设备控制指令下发如下：

	帧头		设备地址		开关	帧尾
开灯	0xFE	0x06 0x91 0x90	0x09 0x1C	0x00	0x01	0xFF
关灯	0xFE	0x06 0x91 0x90	0x09 0x1C	0x00	0x00	0xFF

设备状态数据上报如下（XX表示任意）：

帧头		设备地址	状态字符串	帧尾
0xFE	XX XX XX	0x09 0x1C	0x30 0x2C 0x31 0x2C 0x30 0x2E 0x36	0xFF

状态字符串格式：(故障状态),(开关状态),(电流值)
如 `0,1,0.2`，故障状态和开关状态用 0 和 1 表示。故障状态为 1 表示故障，0 表示正常；开关状态为 1 表示开，0 表示关。
电流值用小数表示，固定3位。
字符串一共使用7个字节。

和http接口如下：

请求地址	方法	功能
/api/ping	GET	测试，返回 "pong"
/api/allDevices	GET	获取所有设备信息，详情见下文
/api/setDevices	POST	设置设备状态，详情见下文
/api/openAll	GET	打开所有灯，无参数
/api/closeAll	GET	关闭所有灯，无参数
/api/open	GET	打开灯，使用url参数
/api/close	GET	关闭灯，使用url参数
/api/deleteDevice	GET	删除灯，使用url参数
/api/setRemark	POST	设置备注，详情见下文

`/api/allDevices` 获取所有设备信息，GET方法，无参数，返回如下结果。

```
{
```

```
  "devices": [
    {
      "addr": 26709,
      "current": 2,
      "switch": true,
      "fault": false,
      "update_at": "1636108091",
      "remark": "办公室"
    },
    {
      "addr": 23125,
      "current": 2,
      "switch": true,
      "fault": false,
      "update_at": "1634116076",
      "remark": ""
    }
  ]
}
```

- `devices`
  - `addr` 设备地址，10进制整数类型。
  - `current` 电流大小，2表示0.2A。
  - `switch` 开关，布尔类型，true为开，false为关。
  - `fault` 故障，true为故障，false为正常。
  - `update_at` 设备最后一次上传数据的数据，用时间戳表示，字符串类型。
  - `remark` 设备备注，字符串。如果没有备注则为空字符串。

`/api/setDevices` 设置设备开关，POST方法，用json传递参数，参数如下。

```
{
  "addr": 26709,
  "switch": true
}
```

成功则返回 `OK`

- `addr` 设备地址，10进制整数类型。
- `switch` 开关，true为开，false为关。

axios例子如下。

```
axios.post("/api/setDevices", {addr: 26709, switch: true})
    .then(res => {
            console.log("成功")
        }
    ).catch(err => {
        console.log(err)
    })
```

`/api/open` 打开灯，GET方法，使用url参数，如下。
`/api/open?addr=26709` addr 为设备地址。成功返回 `OK`
关灯，删除接口方法类似。

axios例子如下。

```
axios.get("/api/open", {params: {addr: 26709}})
    .then(res => {
            console.log("成功")
        }
    ).catch(err => {
        console.log(err)
    })
```

`/api/setRemark` 设置设备备注，POST方法，用json传递参数，参数如下。

```
{
  "addr": 26709,
  "remark": "办公室"
}
```

成功则返回 `OK`

- `addr` 设备地址，10进制整数类型。
- `remark` 备注字符串，空字符表示不使用备注。

部署

首先你需要安装 Docker 和 Docker-Compose

Docker 官方安装教程 https://docs.docker.com/engine/install/
通过脚本安装:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Docker-Compose 官方安装教程 https://docs.docker.com/compose/install/
安装 Docker-Compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

如果找不到 `docker-compose` 命令

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

在树莓派等 Arm 平台

```
sudo apt install docker-compose
```

开始构建

```
sudo docker-compose up -d
```

接下来会自动构建镜像 `smartlightbackend_backend`。
并创建服务 `smart-light-backend`。
可以用 `sudo docker logs -f smart-light-backend` 来查看日志

完成后会监听 9527 和 9528 端口，请保证防火墙放行，如果要改为其它端口请修改 `docker-compose.yml` 文件

```yaml
services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: smart-light-backend
    hostname: smart-light-backend
    ports:
      - "9527:9527"  # 左边的9527为外部访问端口，用于和智能硬件通信
      - "9528:9528"  # 用于和前端通信
    restart: always
```

一键部署脚本

```
sudo sh ./deploy.sh
```

开发

项目目录如下。

```
SmartLightBackend
├── control
│   └── light.go
├── deploy.sh
├── docker-compose.yml
├── Dockerfile
├── go.mod
├── go.sum
├── main.go
├── models
│   └── models.go
├── network
│   └── network.go
├── pkg
│   └── logging
│       └── log.go
├── README.md
├── router
│   ├── api
│   │   └── lamp.go
│   └── router.go
├── runtime
│   └── sqlite.db
└── util
    └── jwt.go
```

`control` 下面是灯控控制逻辑相关代码。
`models` 下面是数据库模型相关代码。
`network` 下面是与硬件通信相关代码。
`pkg` 下面是一些模块相关代码，如日志模块。

`router` 下面是路由相关代码。
`runtime` 下面存放数据库文件。

为了方便接入，后端允许跨域访问。可以在`router.go`中的`InitRouter`函数内取消允许跨域。
后端使用 SQLite 数据库。