HTTP-Server Example

Routine explanation

Introduction

In this section, we use K230 as the HTTP server.

We open the example code and modify the parameters of the Connect_WIFI function in the main function to SSID (the name of the WIFI) and KEY (the password of the WIFI)

```
ip = Connect_WIFI ( "[SSID]" , "[KEY]" )
```

After the modification, we click the run button and you can see the serial interrupt output as follows

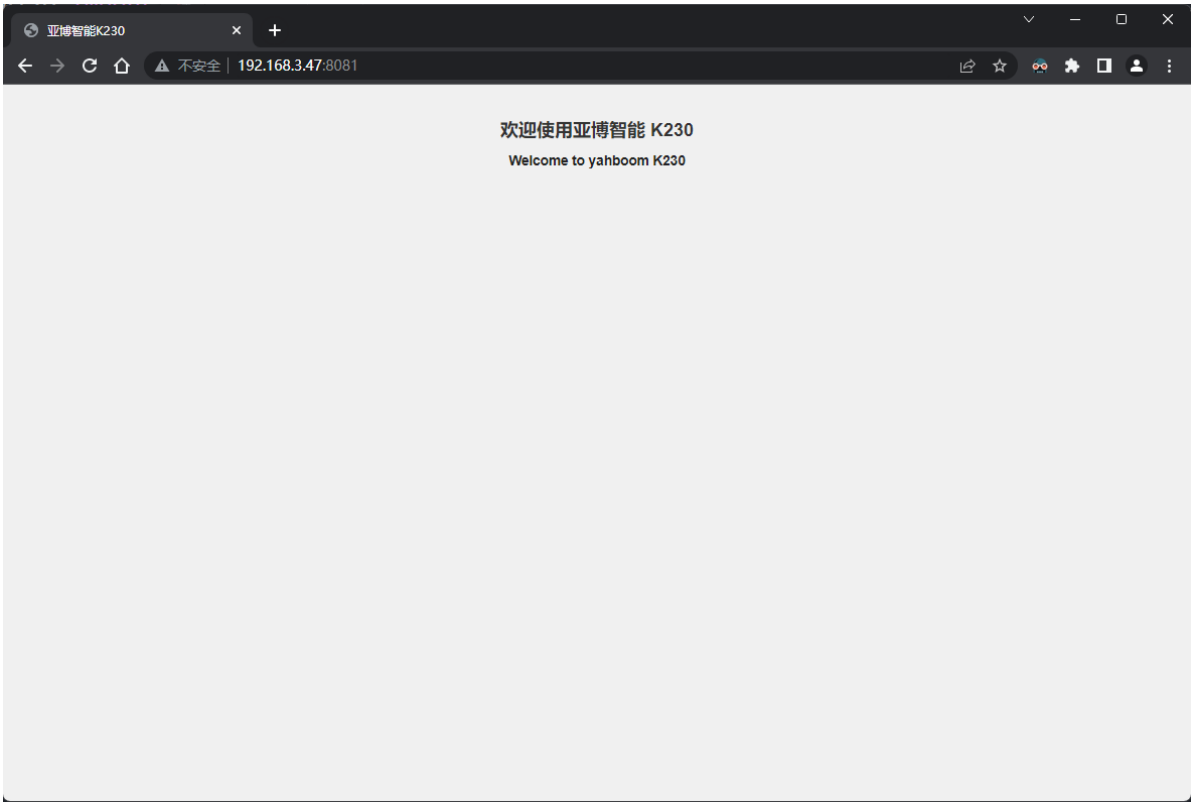
```
True
('192.168.3.47', '255.255.255.0', '192.168.3.1', '192.168.3.1')
Listening, connect your browser to http://192.168.3.47:8081/
```

The first line of True indicates that the WIFI connection is successful

The second line outputs the information of the connected WIFI

The third line indicates that the HTTP server has been successfully started. We can copy this address, paste it into the address bar of the browser, and press Enter.

You can see this page displayed in the browser, which means that the startup is successful.



Key part code

For the complete code, please refer to [Source Code/11.Network/05.http/http_server.py]

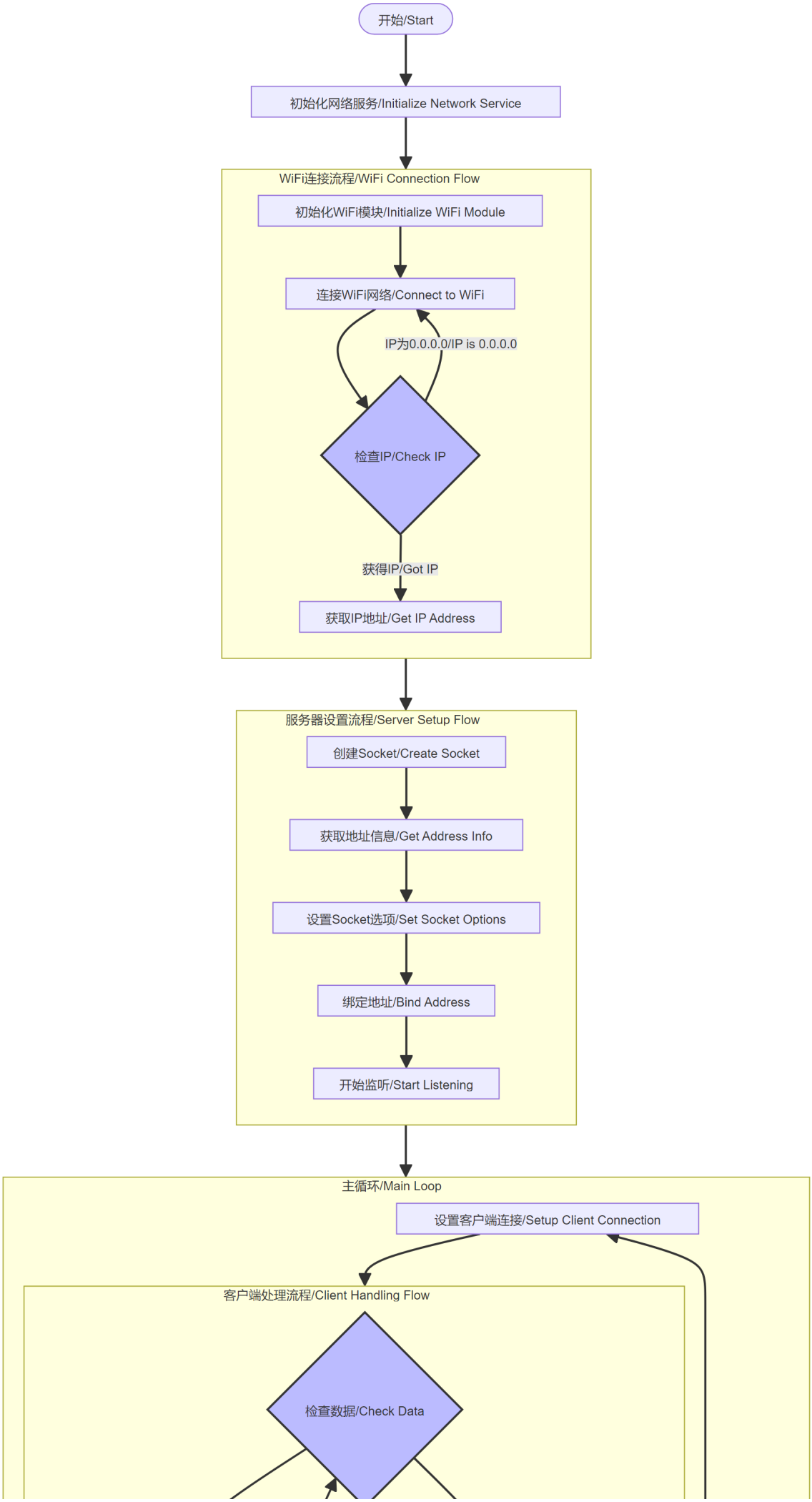
```
def main(micropython_optimize=True):
    # 连接WiFi并获取IP Connect to WiFi and get IP
    ip = Connect_WIFI("Yahboom", "yahboom890729")

    # 创建socket对象 Create socket object
    s = socket.socket()
    # 获取地址信息 Get address information
    ai = socket.getaddrinfo("0.0.0.0", 8081)
    addr = ai[0][-1]
    # 设置socket选项 Set socket options
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # 绑定地址 Bind address
    s.bind(addr)
    # 开始监听 Start listening
    s.listen(5)
    print("Listening, connect your browser to http://%s:8081/" % (ip))

    counter = 0
    # 主循环处理客户端连接 Main loop to handle client connections
    while True:
        # 接受客户端连接 Accept client connection
        res = s.accept()
        client_sock = res[0]
        client_addr = res[1]
        print("Client address:", client_addr)
        # 设置非阻塞模式 Set non-blocking mode
        client_sock.setblocking(False)
        # 获取客户端流 Get client stream
        client_stream = client_sock if micropython_optimize else
client_sock.makefile("rwb")
```

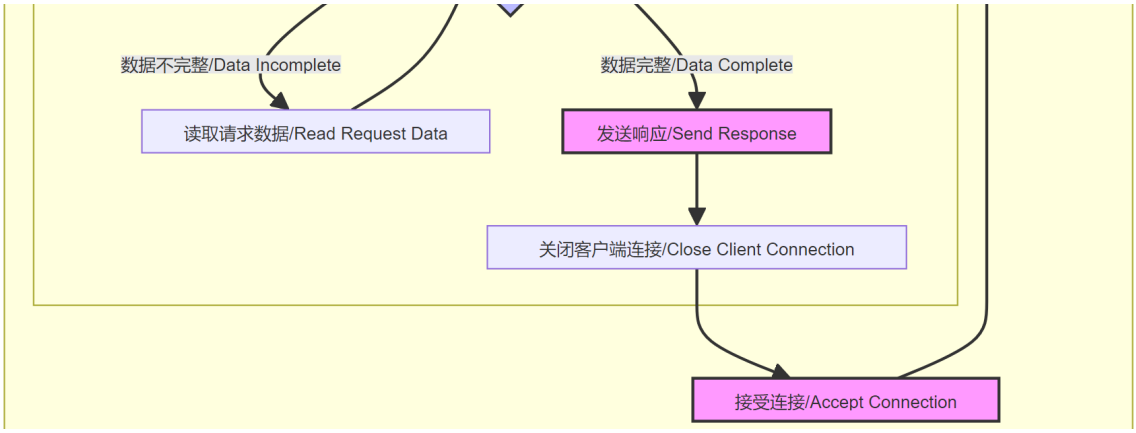
```
        # 读取HTTP请求头 Read HTTP request headers
        while True:
            h = client_stream.read()
            if h is None or h==b'':
                continue
            print(h)
            if h.endswith(b'\r\n\r\n'):
                break
            os.exitpoint()

        # 发送响应内容 Send response content
        client_stream.write(CONTENT)
        # 关闭客户端连接 Close client connection
        client_stream.close()

# 运行主函数 Run main function
main()
```

The CONTENT part is the page code displayed in our browser, using the HTML language

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>亚博智能K230</title>
        <style>
            body {
                font-family: Arial, sans-serif;
                margin: 0;
                padding: 20px;
                background-color: #f0f0f0;
            }
            h1 {
                color: #333;
                text-align: center;
            }
            h2 {
                color: #222;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <h1>欢迎使用亚博智能 K230</h1>
        <h2>Welcome to yahboom K230</h2>
    </body>
</html>
```

flow chart


开始/Start
初始化网络服务/Initialize Network Service
WiFi连接流程/WiFi Connection Flow
初始化WiFi模块/Initialize WiFi Module
连接WiFi网络/Connect to WiFi
IP为0.0.0.0/IP is 0.0.0.0
检查IP/Check IP
获得IP/Got IP
获取IP地址/Get IP Address
服务器设置流程/Server Setup Flow
创建Socket/Create Socket
获取地址信息/Get Address Info
设置Socket选项/Set Socket Options
绑定地址/Bind Address
开始监听/Start Listening
主循环/Main Loop
设置客户端连接/Setup Client Connection
客户端处理流程/Client Handling Flow
检查数据/Check Data



HTTP Server

Here we briefly describe the HTTP server

HTTP Request

> HTTP (Hypertext Transfer Protocol) is an application layer protocol for transferring data over the Internet.

We usually use browsers to access web pages. When we click or perform any operation on a web page, we actually send a request to the HTTP server through the browser, and then the HTTP server returns data (the data can be any type of data such as pictures, videos, or HTML pages. The specific return depends on the operation you requested). The HTTP server we implement in this tutorial receives http requests and returns an HTML page data.

HTML Page

HTML page is a description file that can be recognized and parsed by the browser. It can be simply understood that the web pages we usually see are written in HTML language. In this tutorial, the content of our CONTENT variable is a simple HTML code. When the browser receives the HTML file sent by the HTTP server, it will parse the content of the HTML file into the beautiful web page we see.

HTTP Status Codes

HTTP status codes are identifiers of the server's processing results for client requests and are divided into five categories:

1xx (Informational status code)

- 100 Continue: Continue to send the request
- 101 Switching Protocols:

2xx (successful status code)

- 200 OK: Request successful
- 201 Created: Successfully created
- 204 No Content: Success but no content returned
- 206 Partial Content: Partial content request is successful

3xx (redirect status code)

- 301 Moved Permanently: Permanent redirect

- 302 Found: Temporary redirect
- 304 Not Modified: The resource has not been modified
- 307 Temporary Redirect: Temporary redirect

4xx (Client Error)

- 400 Bad Request: Request syntax error
- 401 Unauthorized: Unauthorized
- 403 Forbidden: Access is forbidden
- 404 Not Found: The resource does not exist
- 405 Method Not Allowed: Method not allowed
- 429 Too Many Requests: Too many requests

5xx (Server Error)

- 500 Internal Server Error: Internal server error
- 502 Bad Gateway: Gateway error
- 503 Service Unavailable: Service unavailable
- 504 Gateway Timeout: Gateway timeout

> The beginning of CONTENT in this tutorial
>
> b"""\ HTTP/1.0 200 OK\r\n\ Content-Type: text/html; charset=utf-8\r\n\ \r\n\
>
> This section is the header of the HTTP response message, and the 200 in it means [200 OK: Request successful]