39 min. read

Python程序设计课程大作业

2021-07-16 | 2020-12-28 | 4 | 0

Python程序设计课程大作业

班级：2018211308

学号：2018211430

姓名：周瑞发

本地系统

【文字描述】

local-proxy提供socks5和http-tunnel服务，接收来自客户端的请求。同时为了方便用户操作，提供了图形界面供用户登录连接。

SOCKS5服务

```
async def socks5(first, reader, writer):
    addr_from = writer.get_extra_info('peername')
    logging.info(f'connect from{addr_from}')
    header = await reader.read(1)
    header = first + header
    ver, num_method = struct.unpack("!BB", header)
    logging.info(f'ver == VERSION:{ver == VERSION}')
    logging.info('num_method = %d' % num_method)
    methods = []
    for i in range(num_method):
```

```
            methods.append(ord(await reader.read(1)))
        if 0 not in methods:#无需认证
            writer.close()
            writer.wait_closed()
            return
        #回应一个数据包，包括协议版本号，指定认证方法
        writer.write(struct.pack("!BB", VERSION, 0))
        await writer.drain()
        request = await reader.read(4)
        ver, cmd, rsv, atype = struct.unpack("!BBBB", request)
        assert ver == VERSION
        assert cmd == 1
        #ipv4
        if atype == 1:
            address = socket.inet_ntoa(await reader.read(4))
        #域名
        elif atype == 3:
            domain_length = await reader.read(1)
            address = await reader.read(domain_length[0])
        #ipv6
        elif atype == 4:
            address = socket.inet_ntop(socket.AF_INET6, await reader.read(16))
        else:
            writer.close()
            writer.wait_closed()
            return
        port = struct.unpack('!H', await reader.read(2))
        reply = struct.pack("!BBBBIH", VERSION, 0, 0, 1, 0, 0)
        writer.write(reply)
        await writer.drain()
        print(address.decode(), str(port[0]))
        # 得到了地址和端口号，sock5连接建立完成,然后调用函数，与remote-proxy连接
        await xfer_remote(reader, writer, address.decode(), str(port[0]))
```

【文字描述】

1、根据socks5协议，首先收到客户端发送过来的协议版本及认证方式，格式为

VER	NMETHODS	METHODS
1	1	1TO255

第一个字段如果是0x05，则表示为socks5协议

第二个字段表示支持的认证方式的数量

第三个字段是一个数组，包含了支持的认证方式列表，我们只考虑了无需认证的方式，即METHOD为0x00

2、local-proxy收到客户端的代理请求后，选择双方都支持的加密方式回复给客户端：

VER	METHOD
1	1

我们这里回复的应该是5，0

此时客户端收到服务端的响应请求后，双方握手完成，开始进行协议交互。

3、认证结束后，客户端发送请求信息，服务端通过解析客户端发过来的请求，获取地址和端口号。

4、获取地址和端口号之后，就可以与remote-proxy建立连接了

HTTPS隧道服务

```
async def httptunnel(first, reader, writer):
    http_connect = (await reader.read(1024))
    http_connect = (first + http_connect).decode()
    #从http请求中解析出ip和端口号
    i = 0
    while(http_connect[i] != ':'):
        i += 1
    domain_name = http_connect[8 : i]
    j = i
    while(http_connect[j] != ' '):
        j += 1
    port = http_connect[i + 1 : j]
    reply = 'HTTP/1.1 200 OK\r\n\r\n'
    writer.write(reply.encode())
    await writer.drain()
    await xfer_remote(reader, writer, domain_name, port)
```

【文字描述】

1、 CONNECT www.example.com:443 HTTP/1.1 http协议很简单，只需要从请求中解析出地址和端口号就可以了。

2、回复 'HTTP/1.1 200 OK\r\n\r\n' 表明连接建立成功，注意一定要加上 \r\n\r\n

3、连接建立好后，就可以与remote-proxy通信了

与远端模块通信

```
async def xfer_remote(reader, writer, address, port):
    #与remote通信的协议是自己定的，为了方便，我直接使用了http协议
    reader_remote,writer_remote = await asyncio.open_connection('123.56.111.64', 1
    http_connect = 'CONNECT ' + address + ':' + port + ' HTTP/1.1'
    http_connect += ' %' + username + '%' + password + '%'
    logging.info(http_connect)
    writer_remote.write(http_connect.encode())
    await writer_remote.drain()
    reply = await reader_remote.read(1024)
    if('HTTP/1.1 200 OK' in reply.decode()):#与remote连接建立成功
        tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, wri
        await asyncio.wait(tasks)
    else:
        logging.info('connect to remote failed!')

async def read_trans(reader, writer_remote):
    while True:
        data = await reader.read(4096)
        if not data:
            logging.info('disconnect')
            break
        writer_remote.write(data)
        await writer_remote.drain()

async def write_trans(reader_remote, writer):
    while True:
        data = await reader_remote.read(4096)
        if not data:
            logging.info('disconnect')
            break
        writer.write(data)
        await writer.drain()
```

【文字描述】

1、local-proxy与remote-proxy之间的通信协议是自己制定的，为了方便，我这里直接使用了http协议。

2、该函数与remote-proxy建立连接后，得到reader_remote和writer_remote。我们的目标是，将从本地reader读到的信息写进writer_remote，将reader_remote读到的信息写进本地writer。

3、建立一个task，使从远端读和像远端写并发执行。 `tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, writer)]`

4、read_trans是用来将本地读到的消息转发给远端，write_trans是将从远端读到的消息转发给本地

图形管理界面

```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow,self).__init__(parent)
        uic.loadUi("mainwindow.ui", self) # 加载界面
        self.pushButton.clicked.connect(self.startClicked)

        self.process = QProcess()
        self.process.setProcessChannelMode(QProcess.MergedChannels)
        self.process.finished.connect(self.processFinished)#当进程结束，触发processF
        self.process.started.connect(self.processStarted)# 当进程已经开始了，触发pro
        self.process.readyReadStandardOutput.connect(self.processReadyRead) #信号

    def processReadyRead(self):
        data = self.process.readAll()
        try:
            msg = data.data().decode().strip()
            logging.debug(f'msg={msg}')
        except Exception as exc:
            logging.error(f'{traceback.format_exc()}')
            exit(1)

    def processStarted(self):  #进程开始后，调用该函数
        process = self.sender() # 此处等同于 self.process 只不过使用sender适应性更好
        processId = process.processId()
        logging.debug(f'pid={processId}')
```

```
        self.pushButton.setText('stop')

        self.websocket = QWebSocket()
        self.websocket.connected.connect(self.websocketConnected)
        self.websocket.disconnected.connect(self.websocketDisconnected)
        self.websocket.textMessageReceived.connect(self.websocketMsgRcvd) #当收到对
        self.websocket.open(QUrl(f'ws://127.0.0.1:{self.consolePortLine.text()}/')

    def startClicked(self):  #当点击开始按钮时
        btn = self.sender()
        text = btn.text().lower()
        if text.startswith('start'):
            listenPort = self.listenPortLine.text()  #本地端口10086
            username = self.usernameLine.text()      #用户名
            password = self.passwordLine.text()      #密码
            consolePort = self.consolePortLine.text() #websockts端口
            remoteHost = self.remoteHostLine.text() #远程主机ip
            remotePort = self.remotePortLine.text() #远程主机端口
            pythonExec = os.path.basename(sys.executable)
            cmdLine = f'{pythonExec} local-proxy.py -p {listenPort} -u {username}
            logging.info(cmdLine)
            logging.debug(f'cmd={cmdLine}')
            self.process.start(cmdLine)
        else:
            self.process.kill()
    def processFinished(self):
        process = self.sender()
        log.debug(f'pid={process.processId()}')
        self.startBtn.setText('Start')
        self.processIdLine.setText('')
    def websocketConnected(self):
        self.websocket.sendTextMessage('connect successful') #连接建立后发送

    def websocketDisconnected(self):
        self.process.kill()

    def websocketMsgRcvd(self, msg):
        logging.debug(f'msg={msg}')
        sendBandwidth, recvBandwidth, *_ = msg.split()
        nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
        logging.info(sendBandwidth)
        logging.info(recvBandwidth)
        self.sendBandwidthLine.setText(f'{nowTime} {sendBandwidth} Bps')
        self.lineEdit_2.setText(f'{nowTime} {recvBandwidth} Bps')
```

【文字描述】

1、界面使用QT的designer设计的，然后将其转换为python的ui代码，↑
`uic.loadUi("mainwindow.ui", self) # 加载界面` 加载ui文件，将设计好的界面导入。

2、当点击开始按钮时，会触发信号startClicked，该函数会打开local-proxy.py程序 `self.process.start(cmdLine)`，当程序开始运行时，又会触发信号processStarted，该函数在界面模块和local-proxy之间建立websocket连接，之后在界面进行的操作就可以发送给local-proxy了。

3、websocket负责与local-proxy通信，将用户信息发送给local-proxy，然后接收来自local-proxy的速率。

远端系统

【文字描述】

remote-proxy接收来自local-proxy的连接，验证local-proxy的用户，并且提供了流控措施。同时，为了方便对用户数据库的管理，，提供了用户数据库的REST管理接口。

与本地模块通信

```
async def handle(reader_local, writer_local):
    logging.info('start working')
    global username_to_token_bucket
    db = await aiosqlite.connect('user.db')
    http_connect = (await reader_local.read(1024))
    http_connect = http_connect.decode()
    logging.info(http_connect)

    i = 0
    while(http_connect[i] != ':'):
        i += 1
    domain_name = http_connect[8 : i]
    j = i
    while(http_connect[j] != ' '):
        j += 1
    port = http_connect[i + 1 : j]
    i = 0
    while(http_connect[i] != '%'):
```

```
            i += 1
        j = i + 1
        while(http_connect[j] != '%'):
            j += 1
        k = j + 1
        while(http_connect[k] != '%'):
            k += 1
        username = http_connect[i + 1: j]
        password = http_connect[j + 1: k]
        sql = 'SELECT * FROM user where name = \'' + username + '\' and password = \''
        cursor = await db.execute(sql)
        row = await cursor.fetchall()
        await cursor.close()
        if(len(row) != 1):
            logging.error('wrong account')
            return
        else:
            logging.info('right account')
        if(username not in username_to_token_bucket.keys()):#用户名不在dict中，要创建令牌
            username_to_token_bucket[username] = 0
            logging.info('init bucket')
        reader_remote,writer_remote = await asyncio.open_connection(domain_name,port)
        sql = 'select dataRate FROM user where name = \'' + username + '\''
        cursor = await db.execute(sql)
        row = await cursor.fetchall()
        await cursor.close()
        speed = row[0][0]
        reply = 'HTTP/1.1 200 OK\r\n\r\n'
        writer_local.write(reply.encode())
        await writer_local.drain()

        tasks = [read_trans(reader_local, writer_remote), write_trans(reader_remote, w
        await asyncio.wait(tasks,return_when=asyncio.FIRST_COMPLETED)
        await db.close()

    async def read_trans(reader, writer_remote):
        while True:
            data = await reader.read(4096)
            if not data:
                logging.info('disconnect from clinet')
                return
            writer_remote.write(data)
            await writer_remote.drain()

    async def write_trans(reader_remote, writer, username, speed):
        global username_to_token_bucket
        data = ''
```

```
66      while True:
67          if(data == ''):
68              data = await reader_remote.read(int(0.01 * speed))# 10.01s    speed  B
69          if not data:
70              logging.info('disconnect from server')
71              return
72          if(username_to_token_bucket[username] < 10):#如果桶里的令牌不够那么就等待，注
73              # 目的是让cpu去执行其他部分的代码，防止在此处阻塞
74              await asyncio.sleep(0)
75              continue
76          else:#令牌够了，把data发出去，同时把data清空
77              username_to_token_bucket[username] -= 10
78              writer.write(data)
79              await writer.drain()
80              data = ''
```

【文字描述】

1、local-proxy和remote-proxy之间的通信协议是自己定的，我这里直接使用了http协议。

2、在收到local-proxy的消息后，从消息中解析出地址、端口号、用户名和密码。

3、然后进行用户名密码验证，验证通过之后，从数据库中查询该用户的速率。

4、向local-proxy回复 `'HTTP/1.1 200 OK\r\n\r\n'` ，表明连接建立成功

5、接下来local-proxy和remote-proxy就可以进行通信了。

6、在remote-proxy向local-proxy发送数据时，要在此处执行限速。只有当令牌的数量足够，才会发送数据包，这也就实现了下载限速。

多用户管理

```
1  async def main():
2      asyncio.create_task(token_bucket_plus_one())
3      server = await asyncio.start_server(handle, '0.0.0.0', 10010)
4      async with server:
5          await server.serve_forever()
```

【文字描述】

1、利用start_server函数，监听来自所有IP地址对端口10010的连接。

2、连接建立后回调handle函数，在handle函数对连接进行处理

↑

3、每来一个用户，就会新建一个连接并且调用handle函数，这样就实现了多用户管理。

用户流控

```
async def write_trans(reader_remote, writer, username, speed):
    global username_to_token_bucket
    data = ''
    while True:
        if(data == ''):
            data = await reader_remote.read(int(0.01 * speed))# 10.01s    speed  B
        if not data:
            logging.info('disconnect from server')
            username_to_token_bucket.pop(username)#去掉该用户
            return
        if(username_to_token_bucket[username] < 10):#如果桶里的令牌不够那么就等待，注
            # 目的是让cpu去执行其他部分的代码，防止在此处阻塞
            await asyncio.sleep(0)
            continue
        else:#令牌够了，把data发出去，同时把data清空
            username_to_token_bucket[username] -= 10
            writer.write(data)
            await writer.drain()
            data = ''
async def token_bucket_plus_one():
    global username_to_token_bucket
    while True:
        for k in username_to_token_bucket.keys():# 每1秒可以攒够1000个令牌，所以平均i
            if username_to_token_bucket[k] < 10000:
                username_to_token_bucket[k] += 10
                print(username_to_token_bucket[k])
        await asyncio.sleep(0.01)
```

【文字描述】

1、username_to_token_bucket是一个dict，键为用户名，因为用户名是唯一的，值是该用户对应的令牌桶。

2、为token_bucket_plus_one创建一个task，每隔一定时间将所有用户对应的令牌桶里令牌的数量加一

3、限速发生在下载时。当有数据发过来时，不会立即发送到local-proxy，而是等待令牌的数量，只有令牌数量够了，才会将数据转发出去。

4、每个用户都会建立一个连接，向local-proxy转发数据时，首先根据用户名创建相应的令牌桶，然后根据该用户的限制速度来进行限速。

5、当用户断开连接时，要在username_to_token_bucket里去掉该用户。

用户数据库管理接口

```
@app.exception(exceptions.NotFound)
async def ignore_404(req, exc):
    return response.text('err_url', status=404)

# 获得所有用户信息
@app.get('/user')
async def userList(req):
    userList = list()
    sql = 'select name, password, dataRate from user;'
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        cursor = await db.execute(sql)
        async for row in cursor:
            user = {'name' : row[0], 'password' : row[1], 'dataRate:' : row[2]}
            logging.debug(f'{user}')
            userList.append(user)
    return response.json(userList)

# 更新用户信息
@app.put('/user')
async def updateUserInfo(req):
    update_user_info = req.json
    name = update_user_info.get('name')
    password = update_user_info.get('password')
    dataRate = update_user_info.get('dataRate')
    db = await aiosqlite.connect(app.config.DB_NAME)
    sql = 'select * from user where name = \'' + name + '\';'
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    if(len(row) == 0):
        await cursor.close()
```

```
31          return response.json({"msg":"user not exist"})
32      sql = 'update user set password = \'' + password + '\', dataRate = ' + str(dat
33      await db.execute(sql)
34      await db.commit()
35      await db.close()
36      return response.json({"msg":"update successful"})
37
38  # 增加用户信息
39  @app.post('/user')
40  async def insertUserInfo(req):
41      insert_user_info = req.json
42      name = insert_user_info.get('name')
43      password = insert_user_info.get('password')
44      dataRate = insert_user_info.get('dataRate')
45      sql = 'select * from user where name = \'' + name + '\';'
46      db = await aiosqlite.connect(app.config.DB_NAME)
47      cursor = await db.execute(sql)
48      row = await cursor.fetchall()
49      if(len(row) != 0):
50          await cursor.close()
51          return response.json({"msg":"user already exist"})
52      sql = 'insert into user values(\'' + name + '\',\'' + password + '\',' + str(d
53      await db.execute(sql)
54      await db.commit()
55      await db.close()
56      return response.json({"msg":"insert successful"})
57
58  #删除用户信息
59  @app.delete('/user')
60  async def deleteUserInfo(req):
61      delete_user_info = req.json
62      name = delete_user_info.get('name')
63      sql = 'select * from user where name = \'' + name + '\';'
64      db = await aiosqlite.connect(app.config.DB_NAME)
65      cursor = await db.execute(sql)
66      row = await cursor.fetchall()
67      if(len(row) == 0):
68          await cursor.close()
69          return response.json({"msg":"user not exist"})
70      sql = 'delete from user where name = \'' + name + '\';'
71      await db.execute(sql)
72      await db.commit()
73      await db.close()
74      return response.json({"msg":"delete successful"})
```

【文字描述】

1、通过不同的http请求来实现对用户数据库的增删改查，对应的http请求分别为post,delete,put,get。

2、在进行删除、增加、更新用户数据时均有**错误处理**，例如，对于更新来说，首先查询待更新的用户是否存在，如果不存在，则返回错误信息，如果存在，则更新并且返回更新成功信息。

程序完整源码

【此处根据个人具体情况粘贴程序源码，可以是单个文件或多个文件，但是只限于自己编写的源程序】

local.py

```
import asyncio
import struct
import socket
import logging
logging.basicConfig(level=logging.INFO)
import sys
import getopt
VERSION = 5
async def socks5(first, reader, writer):
    addr_from = writer.get_extra_info('peername')
    logging.info(f'connect from{addr_from}')
    header = await reader.read(1)
    header = first + header
    ver, num_method = struct.unpack("!BB", header)
    logging.info(f'ver == VERSION:{ver == VERSION}')
    logging.info('num_method = %d' % num_method)
    methods = []
    for i in range(num_method):
        methods.append(ord(await reader.read(1)))
    if 0 not in methods:#无需认证
        writer.close()
        writer.wait_closed()
        return
    #回应一个数据包，包括协议版本号，指定认证方法
    writer.write(struct.pack("!BB", VERSION, 0))
    await writer.drain()
    request = await reader.read(4)
    ver, cmd, rsv, atype = struct.unpack("!BBBB", request)
```

```
    assert ver == VERSION
    assert cmd == 1
    #ipv4
    if atype == 1:
        address = socket.inet_ntoa(await reader.read(4))
    #域名
    elif atype == 3:
        domain_length = await reader.read(1)
        address = await reader.read(domain_length[0])
    #ipv6
    elif atype == 4:
        address = socket.inet_ntop(socket.AF_INET6, await reader.read(16))
    else:
        writer.close()
        writer.wait_closed()
        return
    port = struct.unpack('!H', await reader.read(2))
    reply = struct.pack("!BBBBIH", VERSION, 0, 0, 1, 0, 0)
    writer.write(reply)
    await writer.drain()
    logging.info(address.decode(), str(port[0]))
    # 得到了地址和端口号，sock5连接建立完成,然后调用函数，与remote-proxy连接
    await xfer_remote(reader, writer, address.decode(), str(port[0]))


async def httptunnel(first, reader, writer):
    http_connect = (await reader.read(1024))
    http_connect = (first + http_connect).decode()
    #从http请求中解析出ip和端口号
    i = 0
    while(http_connect[i] != ':'):
        i += 1
    domain_name = http_connect[8 : i]
    j = i
    while(http_connect[j] != ' '):
        j += 1
    port = http_connect[i + 1 : j]
    reply = 'HTTP/1.1 200 OK\r\n\r\n'
    writer.write(reply.encode())
    await writer.drain()
    await xfer_remote(reader, writer, domain_name, port)

async def xfer_remote(reader, writer, address, port):
    #与remote通信的协议是自己定的，为了方便，我直接使用了http协议
    reader_remote,writer_remote = await asyncio.open_connection('123.56.111.64',
    http_connect = 'CONNECT ' + address + ':' + port + ' HTTP/1.1'
    http_connect += ' %' + username + '%' + password + '%'
```

```
76      logging.info(http_connect)
77      writer_remote.write(http_connect.encode())
78      await writer_remote.drain()
79      reply = await reader_remote.read(1024)
80      if('HTTP/1.1 200 OK' in reply.decode()):#与remote连接建立成功
81          tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, wr
82          await asyncio.wait(tasks)
83      else:
84          logging.info('connect to remote failed!')
85  async def read_trans(reader, writer_remote):
86      while True:
87          data = await reader.read(4096)
88          if not data:
89              logging.info('disconnect')
90              break
91          writer_remote.write(data)
92          await writer_remote.drain()
93
94  async def write_trans(reader_remote, writer):
95      while True:
96          data = await reader_remote.read(4096)
97          if not data:
98              logging.info('disconnect')
99              break
100         writer.write(data)
101         await writer.drain()
102
103 async def start(reader, writer):
104     first = await reader.read(1)
105     if(first == b'\x05'):
106         await socks5(first, reader, writer)
107     elif(first == b'C'):
108         await httptunnel(first, reader, writer)
109
110 username = str()
111 password = str()
112
113 async def main():
114     global username, password
115     if(len(sys.argv) != 3):
116         logging.info('usage: local-proxy.py username, password')
117     else:
118         username = sys.argv[1]
119         password = sys.argv[2]
120     logging.info(username)
121     logging.info(password)
122     server = await asyncio.start_server(start, '127.0.0.1', 10086)
```

```
    async with server:
        await server.serve_forever()

asyncio.run(main())
```

localGui.py

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtNetwork import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebSockets import *
from mainwindow import Ui_MainWindow
import sys
from PyQt5 import QtWidgets, uic
import logging
import os
import humanfriendly
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow,self).__init__(parent)
        uic.loadUi("mainwindow.ui", self) # 加载界面
        self.pushButton.clicked.connect(self.startClicked)

        self.process = QProcess()
        self.process.setProcessChannelMode(QProcess.MergedChannels)
        self.process.finished.connect(self.processFinished)#当进程结束，触发processF
        self.process.started.connect(self.processStarted)# 当进程已经开始了，触发pro
        self.process.readyReadStandardOutput.connect(self.processReadyRead) #信号

    def processReadyRead(self):
        data = self.process.readAll()
        try:
            msg = data.data().decode().strip()
            logging.debug(f'msg={msg}')
        except Exception as exc:
            logging.error(f'{traceback.format_exc()}')
            exit(1)

    def processStarted(self):  #进程开始后，调用该函数
        process = self.sender() # 此处等同于 self.process 只不过使用sender适应性更好
        processId = process.processId()
        logging.debug(f'pid={processId}')
        self.pushButton.setText('stop')
```

```
38
39          self.websocket = QWebSocket()
40          self.websocket.connected.connect(self.websocketConnected)
41          self.websocket.disconnected.connect(self.websocketDisconnected)
42          self.websocket.textMessageReceived.connect(self.websocketMsgRcvd) #当收到对
43          self.websocket.open(QUrl(f'ws://127.0.0.1:{self.consolePortLine.text()}/')
44
45      def startClicked(self):  #当点击开始按钮时
46          btn = self.sender()
47          text = btn.text().lower()
48          if text.startswith('start'):
49              listenPort = self.listenPortLine.text()  #本地端口10086
50              username = self.usernameLine.text()      #用户名
51              password = self.passwordLine.text()      #密码
52              consolePort = self.consolePortLine.text() #websockts端口
53              remoteHost = self.remoteHostLine.text() #远程主机ip
54              remotePort = self.remotePortLine.text() #远程主机端口
55              pythonExec = os.path.basename(sys.executable)
56              cmdLine = f'{pythonExec} local-proxy.py -p {listenPort} -u {username}
57              print(cmdLine)
58              logging.debug(f'cmd={cmdLine}')
59              self.process.start(cmdLine)
60          else:
61              self.process.kill()
62      def processFinished(self):
63          process = self.sender()
64          log.debug(f'pid={process.processId()}')
65          self.startBtn.setText('Start')
66          self.processIdLine.setText('')
67      def websocketConnected(self):
68          self.websocket.sendTextMessage('connect successful') #连接建立后随便发的
69
70      def websocketDisconnected(self):
71          self.process.kill()
72
73      def websocketMsgRcvd(self, msg):
74          logging.debug(f'msg={msg}')
75          sendBandwidth, recvBandwidth, *_ = msg.split()
76          nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
77          logging.info(sendBandwidth)
78          logging.info(recvBandwidth)
79          self.sendBandwidthLine.setText(f'{nowTime} {sendBandwidth} Bps')
80          self.lineEdit_2.setText(f'{nowTime} {recvBandwidth} Bps')
81
82  app = QApplication(sys.argv)
83  app.aboutToQuit.connect(app.deleteLater)
84  form = MainWindow()
```

```
form.show()
app.exec_()
```

↑

remote.py

```
import asyncio
import struct
import socket
import logging
import time
logging.basicConfig(level=logging.INFO)
import nest_asyncio
nest_asyncio.apply()
import aiosqlite
async def handle(reader_local, writer_local):
    logging.info('start working')
    global username_to_token_bucket
    db = await aiosqlite.connect('user.db')
    http_connect = (await reader_local.read(1024))
    http_connect = http_connect.decode()
    logging.info(http_connect)

    i = 0
    while(http_connect[i] != ':'):
        i += 1
    domain_name = http_connect[8 : i]
    j = i
    while(http_connect[j] != ' '):
        j += 1
    port = http_connect[i + 1 : j]
    i = 0
    while(http_connect[i] != '%'):
        i += 1
    j = i + 1
    while(http_connect[j] != '%'):
        j += 1
    k = j + 1
    while(http_connect[k] != '%'):
        k += 1
    username = http_connect[i + 1: j]
    password = http_connect[j + 1: k]
    sql = 'SELECT * FROM user where name = \'' + username + '\' and password = \'
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
```

```
    await cursor.close()
    if(len(row) != 1):
        logging.error('wrong account')
        return
    else:
        logging.info('right account')
    if(username not in username_to_token_bucket.keys()):#用户名不在dict中，要创建令
        username_to_token_bucket[username] = 0
        logging.info('init bucket')
    reader_remote,writer_remote = await asyncio.open_connection(domain_name,port)
    sql = 'select dataRate FROM user where name = \'' + username + '\''
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    await cursor.close()
    speed = row[0][0]
    reply = 'HTTP/1.1 200 OK\r\n\r\n'
    writer_local.write(reply.encode())
    await writer_local.drain()

    tasks = [read_trans(reader_local, writer_remote), write_trans(reader_remote,
    await asyncio.wait(tasks,return_when=asyncio.FIRST_COMPLETED)
    await db.close()


async def read_trans(reader, writer_remote):
    while True:
        data = await reader.read(4096)
        if not data:
            logging.info('disconnect from clinet')
            return
        writer_remote.write(data)
        await writer_remote.drain()

async def write_trans(reader_remote, writer, username, speed):
    global username_to_token_bucket
    data = ''
    while True:
        if(data == ''):
            data = await reader_remote.read(int(0.01 * speed))# 10.01s    speed
        if not data:
            logging.info('disconnect from server')
            return
        if(username_to_token_bucket[username] < 10):#如果桶里的令牌不够那么就等待，
            # 目的是让cpu去执行其他部分的代码，防止在此处阻塞
            await asyncio.sleep(0)
            continue
        else:#令牌够了，把data发出去，同时把data清空
```

```
            username_to_token_bucket[username] -= 10
            writer.write(data)
            await writer.drain()
            data = ''
async def token_bucket_plus_one():
    global username_to_token_bucket
    while True:
        for k in username_to_token_bucket.keys():# 每1秒可以攒够1000个令牌，所以平均
            if username_to_token_bucket[k] < 10000:
                username_to_token_bucket[k] += 10
                print(username_to_token_bucket[k])
        await asyncio.sleep(0.01)
async def main():
    asyncio.create_task(token_bucket_plus_one())
    server = await asyncio.start_server(handle, '0.0.0.0', 10010)
    async with server:
        await server.serve_forever()
username_to_token_bucket = {}
asyncio.run(main())
```

remoteRest.py

```
from sanic import Sanic
from sanic import response
from sanic import exceptions
from sanic.response import json
import aiosqlite
import logging

app = Sanic("RemoteProxyAdmin")
app.config.DB_NAME = 'user.db'

@app.exception(exceptions.NotFound)
async def ignore_404(req, exc):
    return response.text('err_url', status=404)

# 获得所有用户信息
@app.get('/user')
async def userList(req):
    userList = list()
    sql = 'select name, password, dataRate from user;'
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        cursor = await db.execute(sql)
        async for row in cursor:
```

```
            user = {'name' : row[0], 'password' : row[1], 'dataRate:' : row[2]}
            logging.debug(f'{user}')
            userList.append(user)
    return response.json(userList)

# 更新用户信息
@app.put('/user')
async def updateUserInfo(req):
    update_user_info = req.json
    name = update_user_info.get('name')
    password = update_user_info.get('password')
    dataRate = update_user_info.get('dataRate')
    db = await aiosqlite.connect(app.config.DB_NAME)
    sql = 'select * from user where name = \'' + name + '\';'
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    if(len(row) == 0):
        await cursor.close()
        return response.json({"msg":"user not exist"})
    sql = 'update user set password = \'' + password + '\', dataRate = ' + str(dat
    await db.execute(sql)
    await db.commit()
    await db.close()
    return response.json({"msg":"update successful"})

# 增加用户信息
@app.post('/user')
async def insertUserInfo(req):
    insert_user_info = req.json
    name = insert_user_info.get('name')
    password = insert_user_info.get('password')
    dataRate = insert_user_info.get('dataRate')
    sql = 'select * from user where name = \'' + name + '\';'
    db = await aiosqlite.connect(app.config.DB_NAME)
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    if(len(row) != 0):
        await cursor.close()
        return response.json({"msg":"user already exist"})
    sql = 'insert into user values(\'' + name + '\',\'' + password + '\',' + str(d
    await db.execute(sql)
    await db.commit()
    await db.close()
    return response.json({"msg":"insert successful"})

#删除用户信息
@app.delete('/user')
```

```
async def deleteUserInfo(req):
    delete_user_info = req.json
    name = delete_user_info.get('name')
    sql = 'select * from user where name = \'' + name + '\';'
    db = await aiosqlite.connect(app.config.DB_NAME)
    cursor = await db.execute(sql)
    row = await cursor.fetchall()
    if(len(row) == 0):
        await cursor.close()
        return response.json({"msg":"user not exist"})
    sql = 'delete from user where name = \'' + name + '\';'
    await db.execute(sql)
    await db.commit()
    await db.close()
    return response.json({"msg":"delete successful"})

@app.route("/zrf")
async def test(request):
    return response.json({"hello": "world"})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)

```

Python程序设计作业#8 Python程序设计作业#7