Python程序设计课程大作业

班级：2018211310

学号：2018211514

姓名：许子康

本地系统

【文字描述】

首先，在作业一和作业二中，实现了LocalProxy双协议（SOCKS5和HTTP tunnel）本地代理。

第一次作业，使用asyncio的streams（coroutine based API）实现SOCKS5服务器，实现了CONNECT、NO AUTHENTICATION REQUIRED功能。

第二次作业，实现了LocalProxy双协议（SOCKS5和HTTP tunnel）本地代理，支持HTTP tunnel可用于HTTPS代理，实现了HTTP CONNECT功能。

然后，在作业三，实现了localProxy和remoteProxy分离式代理，支持SOCKS5代理和HTTPS代理，localProxy收到的每个TCP连接单独建立代理TCP连接。

之后，实现localProxy命令行参数账号登录，有两个选项，分别为：

- 注册，python -s localProxy.py
- 登陆，python -l localProxy.py

最后，实现localProxy的图形管理界面localGui，可通过图形界面（可以使用QDialog）关闭和开启localProxy

界面上提供remoteProxy的主机地址和端口、认证的用户名和密码（掩码显示），可以实时查看localProxy的运行状态（是否运行、实时吞吐率）。

localGui与localProxy之间采用WebSocket连接（localGui为client）。

SOCKS5服务

```python
import asyncio
import struct

async def ExchangeData(reader, writer):
    #交换数据
    while True:
        try:
            data = await reader.read(2048)
            if not data:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(data)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break



async def handle_echo(reader, writer):
    data = await reader.read(2048)
    version, _, methods = struct.unpack('!BBB', data[: 3])
    if methods == 0:
        print('no method!')
    writer.write(struct.pack('!BB', version, 0))
    await writer.drain()

    data = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', data[: 4])

    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', data[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', data[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(data[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', data[4: 5])[0]
        addr = data[5: domainSize + 5].decode()
        port = struct.unpack('!H', data[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = ExchangeData(reader_out, writer)
    in_to_out = ExchangeData(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)


async def work():
    sever_socks5 = await asyncio.start_server(handle_echo, '127.0.0.1', 1080)
    print('-------------------------------------------')
    print('|                IP: 127.0.0.1             |')
    print('|                 Port: 1080               |')
    print('|                                          |')
    print('|                         Designer: Xuzikang|')
    print('|                           Time: 2020.10.26|')
    print('-------------------------------------------')

    async with sever_socks5:
        await sever_socks5.serve_forever()


if __name__ == '__main__':
    asyncio.run(work())
    pass
```

【文字描述】

第一次作业代码，使用asyncio的streams（coroutine based API）实现SOCKS5服务器，实现了CONNECT、NO AUTHENTICATION REQUIRED功能。

先解析version、methods，然后根据methods进行不同的操作。

HTTPS隧道服务

```
import asyncio
import struct

async def Exchangemsg(reader, writer):
    #交换数据
    while True:
        try:
            msg = await reader.read(2048)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break
# http代理
async def handle_http_echo(reader ,writer):
    print('http proxy server')
    msg = await reader.readuntil(b' ')
    print(msg)
    if msg == b'CONNECT ':
        msg = await reader.readuntil(b':')
        addr = msg[: -1]
        msg = await reader.readuntil(b' ')
        port = msg[: -1]
        msg = await reader.readuntil(b'\r\n')
        version = msg[: -2]

        print('address: ', addr)
        print('port: ', port)
        print('version: ', version)

        msg = await reader.read(2048)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
            try:
                sendbuf = 'HTTP/1.1 200 Connection Established\r\n\r\n'
                writer.write(sendbuf.encode())
                await writer.drain()
                print('send succeeded')

                out_to_in = Exchangemsg(reader_out, writer)
                in_to_out = Exchangemsg(reader, writer_out)
                await asyncio.gather(in_to_out, out_to_in)

            except Exception as err:
                print('send failed: ', err)
        except Exception as err:
            print('connection failed: ', err)


# socks5代理
async def handle_socks5_echo(reader, writer):
    print('socks5 proxy server')
    msg = await reader.read(2048)
    version, _, methods = struct.unpack('!BBB', msg[: 3])
    if methods == 0:
        print('no method!')
    writer.write(struct.pack('!BB', version, 0))
    await writer.drain()

    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', msg[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', msg[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(msg[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)


async def work():
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
2080)
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
1080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 1080              |')
    print('|       http tunnel代理端口: 2080            |')
    print('|                                          |')
    print('|                         Designer: Xuzikang|')
    print('|                           Time: 2020.10.26|')
    print('-------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)

if __name__ == '__main__':
    asyncio.run(work())
    pass
```

【文字描述】

第二次作业，实现了LocalProxy双协议（SOCKS5和HTTP tunnel）本地代理，支持HTTP tunnel可用于HTTPS代理，实现了HTTP CONNECT功能。

与远端模块通信

- LocalProxy代理

```
#localProxy
```

```python
import asyncio
import struct
import sys
import asyncio,struct,socket,select,sys
async def Exchangemsg(reader, writer):
    #交换数据
    while True:
        try:
            msg = await reader.read(2048)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break


# socks5代理
async def handle_socks5_echo(reader, writer):
    print('lockal socks5 proxy server')
    print('send to remote proxy server, address 127.0.0.1, port 7080')
    msg = await reader.read(2048)
    version, _, methods = struct.unpack('!BBB', msg[: 3])
    if methods == 0:
        print('no method!')
    writer.write(struct.pack('!BB', version, 0))
    await writer.drain()

    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    reply = ""
    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
7080)
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            reply = await reader_out.read(2048)

        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
7080)
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            reply = await reader_out.read(2048)

        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)


# http代理
async def handle_http_echo(reader ,writer):
    print('local http proxy server')
    print('send to remote proxy server, address 127.0.0.1, port 8080')
    msg = await reader.readuntil(b' ')
    print(msg)
    if msg == b'CONNECT ':

        print('address: 127.0.0.1')
        print('port: 8080')

        msg = await reader.read(2048)
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
8080)
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            try:
                sendbuf = 'HTTP/1.1 200 Connection Established\r\n\r\n'
                writer.write(sendbuf.encode())
                await writer.drain()
                print('send succeeded')

                out_to_in = Exchangemsg(reader_out, writer)
                in_to_out = Exchangemsg(reader, writer_out)
                await asyncio.gather(in_to_out, out_to_in)

            except Exception as err:
                print('send failed: ', err)
        except Exception as err:
            print('connection failed: ', err)
```

```python
# tcp
uname = ''
async def handle_tcp(reader, writer):
    global uname
    option = sys.argv[1]
    name = sys.argv[2]
    uname = name
    password = sys.argv[3]
    #user sign in
    if option == '-s':
        data0 = struct.pack("!B", 0)
    #user log in
    elif option == '-l':
        data0 = struct.pack("!B", 1)
    reader_out, writer_out = await asyncio.open_connection('127.0.0.1', 5080)
    data0 += struct.pack("!B", len(name)) + name.encode() + struct.pack("!B",
len(password)) + password.encode()
    writer_out.write(data0)

    await writer_out.drain()
    data1 = ''

    data1 = await reader_out.read(1024)
    flag = struct.unpack("!B", data1[0:1])[0]

    if flag == 0:
        print('sign in success!')
    elif flag == 10:
        print('wrong password!')
        return
    elif flag == 2:
        print('username not find')
        return
    elif flag == 1:
        print(f'{name!r} log in sucess!')
        data = await reader.read(2048)
        choose = -1
        if data[0] == 67 and data[1] == 79:
            choose = 0
        else:
            version, _, methods = struct.unpack('!BBB', data[:3])
            if version == 5:
                choose = 1
        if choose == 0:  # http
            print('local http proxy server')
            data = data[8:]
            try:
                address = ''
                seq = 0
                for i in range(0, 50):
                    if data[i] == 58:
                        seq = i
                        break
                address = data[0:seq]
                seq1 = seq
                for i in range(seq, seq + 100):
                    if data[i] == 32:
                        seq1 = i
                        break
                port = data[seq + 1 : seq1]
                port = int(port.decode())
                data = struct.pack("!B", len(address)) + address +
struct.pack("!H", port)

                reader_out, writer_out = await
asyncio.open_connection('127.0.0.1', 8080)
                print(data)
                print('connection succeeded')
                writer_out.write(data)
                await writer_out.drain()
                try:
                    sendbuf = 'HTTP/1.1 200 Connection Established\r\n\r\n'
                    writer.write(sendbuf.encode())
                    await writer.drain()
                    print('send succeeded')

                    in_to_out = Exchangemsg(reader, writer_out)
                    out_to_in = Exchangemsg(reader_out, writer)
                    await asyncio.gather(in_to_out, out_to_in)
                except Exception as err:
                    print('send failed: ', err)
            except Exception as err:
                print('connection failed: ', err)

        elif choose == 1:  # socks5
            print('lockal socks5 proxy server')
            if methods == 0:
                print('no method!')
            writer.write(struct.pack('!BB', version, 0))
            await writer.drain()

            data = await reader.read(2048)
            _, command, _, address_type = struct.unpack('!BBBB', data[:4])
            #处理Ipv4类型
            if address_type == 1 and command == 1:
                try:
                    address = '.'.join([str(a) for a in struct.unpack('!BBBB',
data[4:8])])

                    print('address: ', address)
                    port = struct.unpack('!H', data[8:10])[0]
                    data = struct.pack('!B', len(address)) + address.encode() +
struct.pack('!H', port)
                    reader_out, writer_out = await
asyncio.open_connection('127.0.0.1', 7080)

                    print('connection succeeded')

                    writer_out.writer(data)
                    await writer_out.drain()
                    reply = await reader_out.read(2048)
                except Exception as err:
                    print('Ipv4 connection failed:', err)
            # 处理DomainName类型
            elif address_type == 3 and command == 1:
                try:
                    addr_len = struct.unpack('!B', data[4:5])[0]
                    address = data[5:5 + addr_len].decode()
                    port = struct.unpack('!H', data[5 + addr_len:5 + addr_len +
2])[0]
                    data = struct.pack('!B', addr_len) + address.encode() +
struct.pack('!H', port)
                    reader_out, writer_out = await
asyncio.open_connection('127.0.0.1', 7080)

                    print('connection succeeded')

                    writer_out.write(data)
                    await writer_out.drain()

                    reply = await reader_out.read(2048)
                except Exception as err:
                    print('DomainName connection failed:', err)

            writer.write(reply)
            await writer.drain()

            in_to_out = Exchangemsg(reader, writer_out)
            out_to_in = Exchangemsg(reader_out, writer)
            await asyncio.gather(in_to_out, out_to_in)


async def work():
    """
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
2080)
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
1080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 1080              |')
    print('|       http tunnel代理端口: 2080            |')
    print('|                                           |')
    print('|                         Designer: Xuzikang|')
    print('|                           Time: 2020.11.09|')
    print('-------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)
    """
    if sys.argv[1]=='-l':
        print('serving on 127.0.0.1 : 8080')
        my_tcp = await asyncio.start_server(handle_tcp, '127.0.0.1', 8080)
        async with my_tcp:
            await my_tcp.serve_forever()
    else:

        opt = sys.argv[1]
        name = sys.argv[2]
        password = sys.argv[3]
        data0 = ''
        # user sign in
        if opt == '-s':
            data0 = struct.pack("!B", 0)
        Reader, Writer = await asyncio.open_connection('127.0.0.1', 5080)
        data0 += struct.pack("!B", len(name)) + name.encode() +
struct.pack("!B", len(password)) + password.encode()
        Writer.write(data0)

        await Writer.drain()
        data1 = ''

        data1 = await Reader.read(1024)
        flag = struct.unpack("!B", data1[0:1])[0]

        if flag == 0:
            print('Sign in success!')

if __name__ == '__main__':
    asyncio.run(work())
    pass
```

【文字描述】

第三、四、五次作业结合。

首先实现了localProxy和remoteProxy分离式代理，支持SOCKS5代理和HTTPS代理，localProxy收到的每个TCP连接单独建立代理TCP连接。

之后实现localProxy命令行参数账号登录，有两个选项，分别为：

- 注册，python -s localProxy.py
- 登陆，python -l localProxy.py

图形管理界面

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtNetwork import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebSockets import *
import sys,logging,traceback

class Window(QWidget):
    #def __init__(self, parent=None):
    def __init__(self):
        super().__init__()
        #QDialog.__init__(self, parent)

        self.resize(1000, 1000)
        self.move(1000, 200)

        self.sendBandwidthLabel = QLabel(self)
        self.sendBandwidthLabel.setText('发送带宽')
        self.sendBandwidthLabel.resize(500,100)
        self.sendBandwidthLabel.move(450,100)
        self.sendBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: yellow")

        self.recvBandwidthLabel = QLabel(self)
        self.recvBandwidthLabel.setText('接收带宽')
        self.recvBandwidthLabel.resize(500,100)
        self.recvBandwidthLabel.move(450,300)
        self.recvBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: yellow")

        self.listenportlabel = QLabel(self)
        self.listenportlabel.setText('console port')
        self.listenportlabel.move(50,50)
        self.listenPortLine = QLineEdit(self)
        self.listenPortLine.move(50,80)
        self.listenPortLine.setText('')

        self.consolePortlabel = QLabel(self)
        self.consolePortlabel.setText('listen Port')
        self.consolePortlabel.move(50,150)
        self.consolePortLine = QLineEdit(self)
        self.consolePortLine.move(50,180)
        self.consolePortLine.setText('')

        self.usernamelabel = QLabel(self)
        self.usernamelabel.setText('user name')
        self.usernamelabel.move(50,250)
        self.usernameLine = QLineEdit(self)
        self.usernameLine.move(50,280)
        self.usernameLine.setText('')

        self.passwordlabel = QLabel(self)
        self.passwordlabel.setText('password')
        self.passwordlabel.move(50,350)
        self.passwordLine = QLineEdit(self)
        self.passwordLine.move(50,380)
        self.passwordLine.setText('')
        self.passwordLine.setEchoMode(QLineEdit.Password)

        self.startBtn = QPushButton(self)
        self.startBtn.move(50,450)
        self.startBtn.setText('start button')

        self.startBtn.clicked.connect(self.startClicked)

        # 此处省略界面布局

        self.process = QProcess()
        self.process.setProcessChannelMode(QProcess.MergedChannels)
        self.process.finished.connect(self.processFinished)
        self.process.started.connect(self.processStarted)
        self.process.readyReadStandardOutput.connect(self.processReadyRead)

    def processReadyRead(self):
        data = self.process.readAll()

        try:
            print(data.data().strip())
        except Exception as exc:
            log.error(f'{traceback.format_exc()}')
            exit(1)

    def processStarted(self):
        process = self.sender()
        processId = process.processId()
        print('pid = ',processId)
        log.debug(f'pid={processId}')
        self.startBtn.setText('Stop')
        # self.processIdLine.setText(str(processId))

        self.websocket = QWebSocket()
        self.websocket.connected.connect(self.websocketConnected)
        self.websocket.disconnected.connect(self.websocketDisconnected)

        try:
            self.websocket.open(QUrl(f'ws://127.0.0.1:
{self.listenPortLine.text()}/'))
            self.websocket.textMessageReceived.connect(self.websocketMsgRcvd)
            print('conn')
        except:
            print('conn err')

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

    def startClicked(self):
        btn = self.sender()
        text = btn.text().lower()
        if text.startswith('start'):
            listenPort = self.listenPortLine.text()
            username = self.usernameLine.text()
            password = self.passwordLine.text()
            consolePort = self.consolePortLine.text()
            cmdLine = 'python local.py -l ' + username + ' '+  password + ' ' +
listenPort + ' ' + consolePort
            print(cmdLine)
            log.debug(f'cmd={cmdLine}')
            self.process.start(cmdLine)
        else:
            self.process.kill()

    def websocketConnected(self):
        pass

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

    def websocketMsgRcvd(self, msg):
        print('recved msg')
        log.debug(f'msg={msg}')
        sendBandwidth, recvBandwidth, *_= msg.split()
        print(sendBandwidth, recvBandwidth)
        nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
        self.sendBandwidthLabel.setText(f'发送的带宽为：{nowTime} {sendBandwidth}')
        self.recvBandwidthLabel.setText(f'接收的带宽为：{nowTime} {recvBandwidth}')
```

```python
def work():
    app = QApplication(sys.argv)
    w = Window()
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    work()
    pass
```

【文字描述】

作业六，实现localProxy的图形管理界面localGui，可通过图形界面（可以使用QDialog）关闭和开启localProxy

界面上提供remoteProxy的主机地址和端口、认证的用户名和密码（掩码显示），可以实时查看localProxy的运行状态（是否运行、实时吞吐率）。

localGui与localProxy之间采用WebSocket连接（localGui为client）。

远端系统

【文字描述】

从作业三开始逐步编写完善remoteProxy.py代码。

首先，在作业三中实现最基础的远端系统，支持SOCKS5代理和HTTPS代理。

然后，在作业四中，实现了remoteProxy多账号认证。

remoteProxy采用SQLite3数据库进行用户账号管理（用户名、密码），使用aiosqlite操作SQLite3数据库.

示例有两个账号，分别为：

- 账号一：用户名：xuzikang　　　　密码：2018211514
- 账号二：用户名：gofire　　　密码：2147483648

之后，在作业五中，实现了remoteProxy对每个用户进行单独流控。

SQLite3数据库的每个用户的账号信息中增加带宽信息（用户名、密码、带宽）

带宽的单位为BPS（Bytes / Second，字节每秒），该带宽为某个用户的所有连接的转发数据总和带宽。

- RomoteProxy运行方法python RemoteProxy.py [bandwidth]
    - bandwidth表示控制用户的带宽，单位是Bytes/s。
- 实现方法
    - 数据库中每一个用户有两个个属性cur_amount、$last\_time$，cur_amount表示该用户当前的令牌数量，$last_time$表示上次取出令牌的时间。
    - 每个用户有一个令牌桶，初始时每个用户的令牌桶都是满的，即$cur_amount = capacity$。令牌以一定速度均匀产生，但是令牌桶有最大容量，每次读多少字节的数据就取出多少令牌。
    - 在Exchangemsg中，先计算用户从上次取出令牌到现在的令牌增长量inc，同时更新令牌桶的当前容量$cur_amount = min(cur_amount + inc, capacity)$。
    - 比较cur_amount和2048，如果当前令牌数量少，则重复上面步骤，否则就从流中读取数据$data$，再更新$cur_amount = cur_amount - len(data)$，更新$last_time$为当前时间。每次更新都要用互斥锁控制，防止多个协程同时修改用户的两个属性。

最后，在第七次作业中，实现remoteProxy的用户数据库的REST管理接口，基于Sanic实现对user.db数据库（SQLite）的管理接口，支持对user用户的增、删、改、查操作。

与本地模块通信

```
#remoteProxy
import asyncio
import struct

async def Exchangemsg(reader, writer):
    #交换数据
    while True:
        try:
            msg = await reader.read(2048)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break


# socks5代理
async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')
    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', msg[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', msg[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(msg[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)


# http代理
async def handle_http_echo(reader ,writer):
    print('remote http proxy server')

    msg = await reader.readuntil(b':')
    addr = msg[: -1]
    msg = await reader.readuntil(b' ')
    port = msg[: -1]
    msg = await reader.readuntil(b'\r\n')
    version = msg[: -2]

    print('address: ', addr)
    print('port: ', port)
    print('version: ', version)

    msg = await reader.read(2048)
    try:
        reader_out, writer_out = await asyncio.open_connection(addr, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            out_to_in = Exchangemsg(reader_out, writer)
            in_to_out = Exchangemsg(reader, writer_out)
            await asyncio.gather(in_to_out, out_to_in)

        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
            print('connection failed: ', err)



async def work():
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
7080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|          socks5代理Port: 7080              |')
    print('|        http tunnel代理端口: 8080             |')
    print('|                                            |')
    print('|                          Designer: Xuzikang|')
    print('|                            Time: 2020.11.09|')
    print('--------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)

if __name__ == '__main__':
    asyncio.run(Work())
    pass
```

【文字描述】

作业三的remoteProxy.py代码，最基础的远端系统，支持SOCKS5代理和HTTPS代理。

多用户管理

```
#remoteProxy
import asyncio
import struct
import sqlite3
import aiosqlite3

async def Exchangemsg(reader, writer):
    #交换数据
    while True:
        try:
            msg = await reader.read(2048)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

"""
# socks5代理
async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')
    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])
    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', msg[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', msg[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(msg[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)
"""

async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')

    data = await reader.read(2048)

    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1 : addr_len + 1].decode()

    port = struct.unpack('!H', data[addr_len + 1:addr_len + 3])[0]

    reply = struct.pack('!5B', 5, 0, 0, 3, addr_len) + address.encode() +
struct.pack('!H', port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
    except Exception as err:
        print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    in_to_out = Exchangemsg(reader, writer_out)
    out_to_in = Exchangemsg(reader_out, writer)
    await asyncio.gather(in_to_out, out_to_in)

"""
# http代理
async def handle_http_echo(reader ,writer):
    print('remote http proxy server')

    msg = await reader.readuntil(b':')
    addr = msg[: -1]
    msg = await reader.readuntil(b' ')
    port = msg[: -1]
    msg = await reader.readuntil(b'\r\n')
    version = msg[: -2]

    print('address: ', addr)
    print('port: ', port)
    print('version: ', version)

    msg = await reader.read(2048)
    try:
        reader_out, writer_out = await asyncio.open_connection(addr, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            out_to_in = Exchangemsg(reader_out, writer)
            in_to_out = Exchangemsg(reader, writer_out)
            await asyncio.gather(in_to_out, out_to_in)

        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
            print('connection failed: ', err)
"""


async def handle_http_echo(reader, writer):
    print('remote http proxy server')

    data = await reader.read(2048)
    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1:1 + addr_len].decode()
    print(address)
    port = struct.unpack('!H', data[1 + addr_len:1 + addr_len + 2])[0]
    print(port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            in_to_out = Exchangemsg(reader, writer_out)
            out_to_in = Exchangemsg(reader_out, writer)
            await asyncio.gather(in_to_out, out_to_in)
```

```python
        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
        print('connection failed: ', err)


async def handle_confirm(reader, writer):
    flag = 1
    data = await reader.read(1024)
    option = struct.unpack("!B", data[0:1])[0]
    name_len = struct.unpack("!B", data[1:2])[0]

    name = data[2:2 + name_len].decode()
    password_len = struct.unpack("!B", data[2 + name_len:3 + name_len])[0]
    password = data[3 + name_len:3 + name_len + password_len].decode()

    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f"select usrname,usrpassword  from user") as 
cursor:
            print("all info:")
            for row in cursor:
                print(row[0], row[1])
        if option == 0:
            await db.execute(f"insert into user (usrname,usrpassword) \
                values ({name!r},{password!r})")
            await db.commit()
            print('after insert:')
            async with db.execute(f"select usrname,usrpassword  from user") as 
cursor:
                print("all info:")
                for row in cursor:
                    print(row[0], row[1])
            flag = 0
        elif option == 1:
            find = 1
            async with db.execute(f'select usrpassword  from user where 
usrname="{name}"') as cursor:
                if len(list(cursor)) == 0:
                    flag = 2
                    find = 0

            if find == 1:
                async with db.execute(f'select usrpassword from user where 
usrname="{name}"') as cursor:
                    for row in cursor:
                        if row[0] != password:
                            print(row[0], password)
                            flag = 10
                            print('wrong pwd')
    db.close()
    data0 = struct.pack("!B", flag)
    writer.write(data0)
    await writer.drain()

async def work():
    """
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1', 
8080)
```

```
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
7080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 7080              |')
    print('|       http tunnel代理端口: 8080             |')
    print('|                                            |')
    print('|                           Designer: Xuzikang|')
    print('|                             Time: 2020.11.14|')
    print('-------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)
    """
    async with aiosqlite3.connect("user.db") as db:
        await db.execute("delete from user")
        await db.commit()
        print("清除数据")
        await db.execute("drop table user")
        await db.commit()

        await db.execute('''create table user
        (usrname test primary key not null,
        usrpassword text not null);''')
        await db.execute("insert into user (usrname,usrpassword) \
        values ('xuzikang','2018211514')")
        await db.execute("insert into user (usrname,usrpassword) \
        values ('gofire','2147483648')")
        await db.commit()

    while True:
        confirm = await asyncio.start_server(handle_confirm, '127.0.0.1', 5080)
        server_socks5 = await asyncio.start_server(handle_socks5_echo,
'127.0.0.1', 7080)
        server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
        print('-------------------------------------------')
        print('|        代理服务器IP: 127.0.0.1             |')
        print('|         socks5代理Port: 7080              |')
        print('|       http tunnel代理端口: 8080             |')
        print('|                                            |')
        print('|                           Designer: Xuzikang|')
        print('|                             Time: 2020.11.16|')
        print('-------------------------------------------')

        async with server_socks5:
            await server_socks5.serve_forever()
        async with server_http:
            await server_http.serve_forever()

        await asyncio.gather(server_socks5, server_http)

if __name__ == '__main__':
    asyncio.run(work())
    pass
```

【文字描述】

作业四的remoteProxy.py代码，实现了remoteProxy多账号认证。

remoteProxy采用SQLite3数据库进行用户账号管理（用户名、密码），使用aiosqlite操作SQLite3数据库.

示例有两个账号，分别为：

- 账号一：用户名：xuzikang　　　密码：2018211514
- 账号二：用户名：gofire　　　密码：2147483648

用户流控

```
#remoteProxy
import sys
import time
import struct
import asyncio
import sqlite3
import aiosqlite3

async def Exchangemsg(reader, writer, _addr, lock):
    #交换数据
    while True:
        try:
            msg = await myRead(reader, _addr, lock)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

rate = 1000000
capacity = 1000000

async def myRead(reader, _addr, lock):
    cur_amount = 0
    last_time = int(time.time())
    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f"select cur_amount,last_time from user where
usrname={_addr!r}") as cursor:
            for row in cursor:
                cur_amount = row[0]
                last_time = row[1]
                print(f'cur_amount of client {_addr!r}: ',cur_amount)
    increment = (int(time.time())-last_time) * rate
    #lock
    await lock.acquire()
    cur_amount = min(cur_amount + increment, capacity)
    #unlock
    lock.release()
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()

    increment = (int(time.time())-last_time) * rate
    #lock
    await lock.acquire()
    cur_amount = min(cur_amount + increment, capacity)
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    while cur_amount < 2048:
        increment = (int(time.time())-last_time) * rate
        #lock
        await lock.acquire()
        cur_amount = min(cur_amount + increment, capacity)
        async with aiosqlite3.connect("user.db") as db:
            await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
            await db.commit()
        #unlock
        lock.release()
    #lock
    await lock.acquire()
    last_time = int(time.time())
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set last_time={last_time!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    data = await reader.read(2048)
    #lock
    await lock.acquire()
    cur_amount -= len(data)
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    return data

"""
# socks5代理
async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')
    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', msg[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', msg[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(msg[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)
"""

async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')

    data = await reader.read(2048)

    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1 : addr_len + 1].decode()

    port = struct.unpack('!H', data[addr_len + 1:addr_len + 3])[0]

    reply = struct.pack('!5B', 5, 0, 0, 3, addr_len) + address.encode() +
struct.pack('!H', port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
    except Exception as err:
            print('DomainName connection failed:', err)

        writer.write(reply)
        await writer.drain()

        in_to_out = Exchangemsg(reader, writer_out)
        out_to_in = Exchangemsg(reader_out, writer)
        await asyncio.gather(in_to_out, out_to_in)

"""
# http代理
async def handle_http_echo(reader ,writer):
    print('remote http proxy server')

    msg = await reader.readuntil(b':')
    addr = msg[: -1]
    msg = await reader.readuntil(b' ')
    port = msg[: -1]
    msg = await reader.readuntil(b'\r\n')
    version = msg[: -2]

    print('address: ', addr)
    print('port: ', port)
    print('version: ', version)

    msg = await reader.read(2048)
    try:
        reader_out, writer_out = await asyncio.open_connection(addr, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            out_to_in = Exchangemsg(reader_out, writer)
            in_to_out = Exchangemsg(reader, writer_out)
            await asyncio.gather(in_to_out, out_to_in)

        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
            print('connection failed: ', err)
"""


async def handle_http_echo(reader, writer):
    print('remote http proxy server')

    data = await reader.read(2048)
    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1:1 + addr_len].decode()
    print(address)
    port = struct.unpack('!H', data[1 + addr_len:1 + addr_len + 2])[0]
    print(port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
        try:
            print('send succeeded')
            in_to_out = Exchangemsg(reader, writer_out)
            out_to_in = Exchangemsg(reader_out, writer)
            await asyncio.gather(in_to_out, out_to_in)
        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
        print('connection failed: ', err)


async def handle_tcp_echo(reader, writer):
    _addr = ''
    data = await reader.read(2048)

    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1:addr_len + 1].decode()
    port = struct.unpack('!H', data[addr_len + 1:addr_len + 3])[0]
    _addr_len = struct.unpack('!B', data[addr_len + 3:addr_len + 4])[0]
    _addr = data[addr_len + 4:addr_len + 4 + _addr_len].decode()
    print(address, port)
    print(f'client {_addr!r} exchange data')
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection success')
    except:
        print('domainname connect error!')
    lock = asyncio.Lock()
    in_to_out = Exchangemsg(reader, writer_out, _addr, lock)
    out_to_in = Exchangemsg(reader_out, writer, _addr, lock)
    await asyncio.gather(in_to_out, out_to_in)


async def handle_confirm(reader, writer):
    flag = 1
    data = await reader.read(1024)
    option = struct.unpack("!B", data[0:1])[0]
    name_len = struct.unpack("!B", data[1:2])[0]

    name = data[2:2 + name_len].decode()
    password_len = struct.unpack("!B", data[2 + name_len:3 + name_len])[0]
    password = data[3 + name_len:3 + name_len + password_len].decode()

    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f"select usrname,usrpassword  from user") as
cursor:
            print("all info:")
            for row in cursor:
                print(row[0], row[1])
        if option == 0:
            await db.execute(f"insert into user (usrname,usrpassword) \
                values ({name!r},{password!r})")
            await db.commit()
            print('after insert:')
            async with db.execute(f"select usrname,usrpassword  from user") as
cursor:
                print("all info:")
                for row in cursor:
                    print(row[0], row[1])
            flag = 0
        elif option == 1:
            find = 1
            async with db.execute(f'select usrpassword  from user where
usrname="{name}"') as cursor:
                if len(list(cursor)) == 0:
                    flag = 2
                    find = 0

            if find == 1:
                async with db.execute(f'select usrpassword from user where
usrname="{name}"') as cursor:
                    for row in cursor:
                        if row[0] != password:
                            print(row[0], password)
                            flag = 10
                            print('wrong pwd')
    db.close()
    data0 = struct.pack("!B", flag)
    writer.write(data0)
    await writer.drain()
    if flag == 1:
        await handle_tcp_echo(reader, writer)

async def Work():
    """
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
7080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 7080               |')
    print('|       http tunnel代理端口: 8080             |')
    print('|                                           |')
    print('|                            Designer: Xuzikang|')
    print('|                              Time: 2020.11.09|')
    print('-------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)
    """
    global rate, capacity
    if sys.argv[1] == '-d':
        rate = 1000000
    else:
        rate = int(sys.argv[1])
    capacity = rate
    async with aiosqlite3.connect("user.db") as db:
        await db.execute("delete from user")
        await db.commit()
        print("清除数据")
        await db.execute("drop table user")
        await db.commit()
        await db.execute('''create table user
        (usrname test primary key not null,
        usrpassword text not null);''')
        await db.execute("insert into user (usrname,usrpassword) \
        values ('xuzikang','2018211514')")
        await db.execute("insert into user (usrname,usrpassword) \
        values ('gofire','2147483648')")
        await db.commit()

    while True:
        confirm = await asyncio.start_server(handle_confirm, '127.0.0.1', 5080)
        """
        server_socks5 = await asyncio.start_server(handle_socks5_echo,
'127.0.0.1', 7080)
        server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
        print('-------------------------------------------')
        print('|        代理服务器IP: 127.0.0.1            |')
        print('|         socks5代理Port: 7080              |')
        print('|       http tunnel代理端口: 8080            |')
        print('|                                          |')
        print('|                         Designer: Xuzikang|')
        print('|                           Time: 2020.11.16|')
        print('-------------------------------------------')

        async with server_socks5:
            await server_socks5.serve_forever()
        async with server_http:
            await server_http.serve_forever()
        """
        async with confirm:
            await confirm.serve_forever()
        # await asyncio.gather(server_socks5, server_http)

if __name__ == '__main__':
    asyncio.run(work())
```

【文字描述】

作业五的remoteProxy代码，实现了remoteProxy对每个用户进行单独流控。

SQLite3数据库的每个用户的账号信息中增加带宽信息（用户名、密码、带宽）

带宽的单位为BPS（Bytes / Second，字节每秒），该带宽为某个用户的所有连接的转发数据总和带宽。

- RomoteProxy运行方法python RemoteProxy.py [bandwidth]
  - bandwidth表示控制用户的带宽，单位是Bytes/s。
- 实现方法
  - 数据库中每一个用户有两个个属性cur_amount、$last\_time$，cur_amount表示该用户当前的令牌数量，$last_time$表示上次取出令牌的时间。
  - 每个用户有一个令牌桶，初始时每个用户的令牌桶都是满的，即$cur_amount = capacity$。令牌以一定速度均匀产生，但是令牌桶有最大容量，每次读多少字节的数据就取出多少令牌。
  - 在Exchangemsg中，先计算用户从上次取出令牌到现在的令牌增长量inc，同时更新令牌桶的当前容量$cur_amount = min(cur_amount + inc, capacity)$。
  - 比较cur_amount和2048，如果当前令牌数量少，则重复上面步骤，否则就从流中读取数据$data$，再更新$cur_amount = cur_amount - len(data)$，更新$last_time$为当前时间。每

次更新都要用互斥锁控制，防止多个协程同时修改用户的两个属性。

用户数据库管理接口

```
import sys
import time
import asyncio
import aiosqlite

from sanic import Sanic
from sanic import response
from sanic import exceptions

app = Sanic('RemoteProxyAdmin')
app.config.DB_userName = 'user.db'

@app.delete('/user/<userName>')
async def userDel(req, userName):
    async with aiosqlite.connect(app.config.DB_userName) as db:
        await db.execute("DELETE FROM user WHERE userName=?", (userName,))
        await db.commit()
    return response.json({})

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

@app.post('/user')
async def userAdd(req):
    userName = req.json.get('userName')
    password = req.json.get('password')
    dataRate = req.json.get('dataRate')
    if not userName or not password or not dataRate:
        return response.text(f'err userName={userName} password={password}
dataRate={dataRate}',status=400)
    async with aiosqlite.connect(app.config.DB_userName) as db:
        await db.execute("INSERT INTO user(userName,password,dataRate)
VALUES(?,?,?)", (userName,password,dataRate))
        await db.commit()
    return response.json({})

@app.put('/user/<userName>')
async def userModify(req, userName):
    password = req.json.get('password')
    dataRate = req.json.get('dataRate')
    if not userName or not password or not dataRate:
        return resopnse.text(f'err userName={userName} password={password}
dataRate={dataRate}', stauts = 400)
    async with aiosqlite.connect(app.config.DB_userName) as db:
        await db.execute("UPDATE user SET passowrd=?, dataRate=? WHERE
userName=?", (password,dataRate,userName))
        await db.commit()
    return response.json({})

@app.get('/user')
async def userList(req):
    userList = list()
    async with aiosqlite.connect(app.config.DB_userName) as db:
        async with db.execute("SELECT userName,password,dataRate FROM user;") as
cursor:
            async for row in cursor:
                user = {'userName':row[0], 'password':row[1], 'dataRate':row[2]}
                userList.append(user)
    return response.json(userList)

@app.get('/user/<userName>')
async def userList(req, userName):
    async with aiosqlite.connect(app.config.DB_userName) as db:
        async with db.execute("SELECT password,dataRate FROM user WHERE
userName=?", (userName,)) as cursor:
            async for row in cursor:
                user = {'userName':userName, 'password':row[0],
'dataRate':row[[1]]}
                return response.json(user)
    return response.json({}, status = 404)



if __userName__ == '__main__':
    app.run(host='127.0.0.1', port = 8080)
```

【文字描述】

第七次作业，实现remoteProxy的用户数据库的REST管理接口，基于Sanic实现对user.db数据库（SQLite）的管理接口，支持对user用户的增、删、改、查操作。

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

```
#localProxy
import sys
import time
import struct
import socket
import select
import asyncio
import sqlite3
import aiosqlite3

uname = ''
len_now_rdata = 0
len_now_wdata = 0
gSendBrandwidth = 0
gRecvBrandwidth = 0
async def Exchangemsg(reader, writer):
    #交换数据
    global len_now_rdata
    global len_now_wdata
    while True:
        try:
            data = await reader.read(2048)
            len_now_rdata += len(data)
            if not data:
                writer.close()
                break
        except:
            writer.close()
            break
        try:
            writer.write(data)
            len_now_wdata += len(data)
            await writer.drain()
        except:
            writer.close()
            break


# socks5代理
async def handle_socks5_echo(reader, writer):
    print('lockal socks5 proxy server')
    print('send to remote proxy server, address 127.0.0.1, port 7080')
    msg = await reader.read(2048)
    version, _, methods = struct.unpack('!BBB', msg[: 3])
    if methods == 0:
        print('no method!')
    writer.write(struct.pack('!BB', version, 0))
    await writer.drain()

    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    reply = ""
    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
7080)
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            reply = await reader_out.read(2048)

        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
7080)
```

```
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            reply = await reader_out.read(2048)

        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)


# http代理
async def handle_http_echo(reader ,writer):
    print('local http proxy server')
    print('send to remote proxy server, address 127.0.0.1, port 8080')
    msg = await reader.readuntil(b' ')
    print(msg)
    if msg == b'CONNECT ':

        print('address: 127.0.0.1')
        print('port: 8080')

        msg = await reader.read(2048)
        try:
            reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
8080)
            print('connection succeeded')
            writer_out.write(msg)
            await writer_out.drain()
            try:
                sendbuf = 'HTTP/1.1 200 Connection Established\r\n\r\n'
                writer.write(sendbuf.encode())
                await writer.drain()
                print('send succeeded')

                out_to_in = Exchangemsg(reader_out, writer)
                in_to_out = Exchangemsg(reader, writer_out)
                await asyncio.gather(in_to_out, out_to_in)

            except Exception as err:
                print('send failed: ', err)
        except Exception as err:
            print('connection failed: ', err)

# tcp
async def handle_tcp(reader, writer):
    global len_now_rdata
    global len_now_wdata
    global uname
    option = sys.argv[1]
    name = sys.argv[2]
    uname = name
    password = sys.argv[3]
    #sign in
    if option == '-s':
        data0 = struct.pack("!B",0)
    #log in
    elif option == '-l':
        data0 = struct.pack("!B",1)
    reader_out, writer_out = await asyncio.open_connection('127.0.0.1', 5080)
    data0 +=
struct.pack("!B",len(name))+name.encode()+struct.pack("!B",len(password))+passwo
rd.encode()
    writer_out.write(data0)
    len_now_wdata += len(data0)

    await writer_out.drain()
    data1=''

    data1 = await reader_out.read(2048)
    len_now_rdata += len(data1)
    flag = struct.unpack("!B",data1[0:1])[0]

    if flag == 0:
        print('sign success')
    elif flag == 10:
        print('wrong password')
        return
    elif flag == 2:
        print('username not find')
        return
        ########33
    elif flag == 1:
        print(f'{name!r} log in sucess')
        data = await reader.read(2048)
        len_now_rdata += len(data)
        choose = -1
        if data[0] == 67 and data[1] == 79:
            choose = 0
        else:
            version,nmethods,methods=struct.unpack('!BBB',data[:3])
            if version == 5:
                choose = 1
        if choose == 0:#http
            print('local http proxy server')
            data = data[8:]
            try:
                address = ''
                seq=0
                for i in range(0,50):
                    if data[i]==58:
                        seq=i
                        break
                address=data[0:seq]
                seq1=seq
                for i in range(seq,seq+100):
                    if data [i] == 32:
                        seq1 = i
                        break
                port = data[seq+1:seq1]
                port = int(port.decode())
                data = struct.pack("!B",len(address)) + address+
struct.pack("!H", port) + struct.pack("!B",len(uname))+uname.encode()

                print(data)
                writer_out.write(data)
                len_now_wdata += len(data)
                await writer_out.drain()
                try:
                    sendbuf='HTTP/1.1 200 Connection Established\r\n\r\n'
                    writer.write(sendbuf.encode())
                    len_now_wdata += len(sendbuf.encode())
                    await writer.drain()
                    print('send sucess!')
                    in_to_out = Exchangemsg(reader, writer_out)
                    out_to_in = Exchangemsg(reader_out, writer)
                    await asyncio.gather(in_to_out,out_to_in)
                except:
                    print("fail to send!')

            except:
                print('http send err')
        elif choose == 1:#socks5
            print('lockal socks5 proxy server')
            if methods == 0:
                print("hi")
            writer.write(struct.pack('!BB',version,0))
            len_now_wdata += 2
            await writer.drain()

            data = await reader.read(2048)
            len_now_rdata += len(data)
            _, command, _, address_type = struct.unpack('!BBBB', data[:4])
            #ipv4
            if address_type == 1 and command == 1:
                try:
                    address = '.'.join([str(a) for a in
struct.unpack('!BBBB',data[4:8])])
                    print("address")
                    print(address)
                    port = struct.unpack('!H',data[8:10])[0]
                    data = struct.pack('!B',
len(address))+address.encode()+struct.pack('!H', port) +
struct.pack("!B",len(uname))+uname.encode()
                    print('connect success')

                    writer_out.writer(data)
                    len_now_wdata += len(data)
                    await writer_out.drain()

 reply=struct.pack('!4B',5,0,0,1)+address.encode()+struct.pack('!H',port)
                except:
                    print('ipv4 connect err!')
            #域名
            elif address_type == 3 and command == 1:
                try:
                    addr_len = struct.unpack('!B', data[4:5])[0]
                    address = data[5:5+addr_len].decode()
                    port = struct.unpack('!H', data[5+addr_len:5+addr_len+2])[0]
                    data = struct.pack('!B',
addr_len)+address.encode()+struct.pack('!H', port) +
struct.pack("!B",len(uname))+uname.encode()
                    #reader_out,writer_out=await
asyncio.open_connection('127.0.0.1', 7878)
                    print('连接成功')

                    writer_out.write(data)
                    len_now_wdata += len(data)
                    await writer_out.drain()

 reply=struct.pack('!5B',5,0,0,3,addr_len)+address.encode()+struct.pack('!H',por
t)
                    #reply = await reader_out.read(2048)
                except:
                    print('!!!!!!!!!!!!!!!!!!!domainname connect err')

            writer.write(reply)
            len_now_wdata += len(reply)
            await writer.drain()

            in_to_out = Exchangemsg(reader, writer_out)
            out_to_in = Exchangemsg(reader_out, writer)
            await asyncio.gather(in_to_out,out_to_in)

async def clacbrandwidth():
    global gSendBrandWidth
    global gRecvBrandWidth
    global len_now_rdata
    global len_now_wdata
    gSendBrandWidth = 0
    gRecvBrandWidth = 0
    while True:
        gSendBrandWidth = len_now_wdata
        gRecvBrandWidth = len_now_rdata
        len_now_rdata = 0
        len_now_wdata = 0

        print(f'接收的带宽为: {gRecvBrandWidth!r}')
        print(f'发送的带宽为: {gSendBrandWidth!r}')
        await asyncio.sleep(1)


async def localConsole(ws, path):
    global gRecvBrandWidth
    global gSendBrandWidth
    try:
        while True:
            await asyncio.sleep(1)
            msg = await ws.send(f'{gSendBrandWidth} {gRecvBrandWidth}')
    except websockets.exceptions.ConnectionClosedError as exc:
        log.error(f'exc')
    except websockets.exceptions.ConnectionClosedOK as exc:
        log.error(f'exc')
    except Exception:
        log.error(f'{traceback.format_exc()}')
        exit(1)
```

```
async def Work():
    """
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
2080)
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
1080)
    print('------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 1080              |')
    print('|       http tunnel代理端口: 2080            |')
    print('|                                          |')
    print('|                         Designer: Xuzikang|')
    print('|                           Time: 2020.11.09|')
    print('------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)
    """

    global len_now_rdata
    global len_now_wdata
    print('local started')
    asyncio.create_task(clacbrandwidth())
    ws_server = await websockets.serve(localConsole, '127.0.0.1',
int(sys.argv[4]))

    if sys.argv[1]=='-l':
        print('serving on 127.0.0.1:8080')
        my_tcp = await asyncio.start_server(handle_tcp, '127.0.0.1',
int(sys.argv[5]))
        async with my_tcp:
            await my_tcp.serve_forever()
    else:
        option = sys.argv[1]
        name = sys.argv[2]
        password = sys.argv[3]
        #sign in
        if option == '-s':
            data0 = struct.pack("!B",0)
        reader_out, writer_out = await asyncio.open_connection('127.0.0.1',
5080)
        data0 +=
struct.pack("!B",len(name))+name.encode()+struct.pack("!B",len(password))+passwo
rd.encode()
        writer_out.write(data0)
        len_now_wdata += len(data0)

        await writer_out.drain()
        data1=''

        data1 = await reader_out.read(2048)
        len_now_rdata += len(data1)
        flag = struct.unpack("!B",data1[0:1])[0]

        if flag == 0:
            print('sign success!')

if __name__ == '__main__':
    log = logging.getLogger(__file__)
    asyncio.run(work())
    pass
```

localGui.py

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtNetwork import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebSockets import *
import sys,logging,traceback

class window(QWidget):
    def __init__(self):
        super().__init__()

        self.resize(1000, 1000)
        self.move(1000, 200)

        self.sendBandwidthLabel = QLabel(self)
        self.sendBandwidthLabel.setText('发送带宽')
        self.sendBandwidthLabel.resize(500,100)
        self.sendBandwidthLabel.move(450,100)
        self.sendBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: yellow")

        self.recvBandwidthLabel = QLabel(self)
        self.recvBandwidthLabel.setText('接收带宽')
        self.recvBandwidthLabel.resize(500,100)
        self.recvBandwidthLabel.move(450,300)
        self.recvBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: yellow")

        self.listenportlabel = QLabel(self)
        self.listenportlabel.setText('console port')
        self.listenportlabel.move(50,50)
        self.listenPortLine = QLineEdit(self)
        self.listenPortLine.move(50,80)
        self.listenPortLine.setText('')

        self.consolePortlabel = QLabel(self)
        self.consolePortlabel.setText('listen Port')
        self.consolePortlabel.move(50,150)
        self.consolePortLine = QLineEdit(self)
        self.consolePortLine.move(50,180)
        self.consolePortLine.setText('')

        self.usernamelabel = QLabel(self)
        self.usernamelabel.setText('user name')
        self.usernamelabel.move(50,250)
        self.usernameLine = QLineEdit(self)
        self.usernameLine.move(50,280)
        self.usernameLine.setText('')

        self.passwordlabel = QLabel(self)
        self.passwordlabel.setText('password')
        self.passwordlabel.move(50,350)
        self.passwordLine = QLineEdit(self)
        self.passwordLine.move(50,380)
        self.passwordLine.setText('')
        self.passwordLine.setEchoMode(QLineEdit.Password)

        self.startBtn = QPushButton(self)
        self.startBtn.move(50,450)
        self.startBtn.setText('start button')

        self.startBtn.clicked.connect(self.startClicked)

        # 此处省略界面布局

        self.process = QProcess()
        self.process.setProcessChannelMode(QProcess.MergedChannels)
        self.process.finished.connect(self.processFinished)
        self.process.started.connect(self.processStarted)
        self.process.readyReadStandardOutput.connect(self.processReadyRead)

    def processReadyRead(self):
        data = self.process.readAll()

        try:
            print(data.data().strip())
        except Exception as exc:
            log.error(f'{traceback.format_exc()}')
            exit(1)

    def processStarted(self):
        process = self.sender()
        processId = process.processId()
        print('pid = ',processId)
        log.debug(f'pid={processId}')
        self.startBtn.setText('Stop')
        # self.processIdLine.setText(str(processId))

        self.websocket = QWebSocket()
        self.websocket.connected.connect(self.websocketConnected)
        self.websocket.disconnected.connect(self.websocketDisconnected)

        try:
            self.websocket.open(QUrl(f'ws://127.0.0.1:
{self.listenPortLine.text()}/'))
            self.websocket.textMessageReceived.connect(self.websocketMsgRcvd)
            print('conn')
        except:
            print('conn err')

    def processFinished(self):
        self.process.kill()
    def startClicked(self):
        btn = self.sender()
        text = btn.text().lower()
        if text.startswith('start'):
            listenPort = self.listenPortLine.text()
            username = self.usernameLine.text()
            password = self.passwordLine.text()
            consolePort = self.consolePortLine.text()
            cmdLine = 'python local.py -l ' + username + ' '+  password + ' ' +
listenPort + ' ' + consolePort
            print(cmdLine)
            log.debug(f'cmd={cmdLine}')
            self.process.start(cmdLine)
        else:
            self.process.kill()

    def websocketConnected(self):
        pass

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

    def websocketMsgRcvd(self, msg):
        print('recved msg')
        log.debug(f'msg={msg}')
        sendBandwidth, recvBandwidth, *_= msg.split()
        print(sendBandwidth, recvBandwidth)
        nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
        self.sendBandwidthLabel.setText(f'发送的带宽为：{nowTime} {sendBandwidth}')
        self.recvBandwidthLabel.setText(f'接收的带宽为：{nowTime} {recvBandwidth}')

def Work():
    app = QApplication(sys.argv)
    w = Window()
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    Work()
    pass
```

remote.py

```
#remoteProxy
import sys
import time
import struct
import socket
import select
import asyncio
import sqlite3
import aiosqlite3

async def Exchangemsg(reader, writer, _addr, lock):
    #交换数据
    while True:
        try:
            msg = await myRead(reader, _addr, lock)
            if not msg:
                writer.close()
                break
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

        try:
            writer.write(msg)
            await writer.drain()
        except Exception as err:
            writer.close()
            print('Error:', err)
            break

rate = 1000000
capacity = 1000000

"""
# socks5代理
async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')
    msg = await reader.read(2048)
    _, cmd, _, addrType = struct.unpack('!BBBB', msg[: 4])

    #处理Ipv4类型
    if cmd == 1 and addrType == 1:
        addr = '.'.join([str(a) for a in struct.unpack('!BBBB', msg[4: 8])])
        print('address: ', addr)
        port = struct.unpack('!H', msg[-2: ])[0]
        reply = struct.pack('!8B', 5, 0, 0, 3, struct.unpack(msg[4: 8])) +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('Ipv4 connection failed:', err)

    #处理DomainName类型
    elif cmd == 1 and addrType == 3:
        domainSize = struct.unpack('!B', msg[4: 5])[0]
        addr = msg[5: domainSize + 5].decode()
        port = struct.unpack('!H', msg[-2:])[0]

        print('address: ', addr)
        print('port: ', port)
        reply = struct.pack('!5B', 5, 0, 0, 3, domainSize) + addr.encode() +
struct.pack('!H', port)
        try:
            reader_out, writer_out = await asyncio.open_connection(addr, port)
            print('connection succeeded')
        except Exception as err:
            print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()
    out_to_in = Exchangemsg(reader_out, writer)
    in_to_out = Exchangemsg(reader, writer_out)

    await asyncio.gather(in_to_out, out_to_in)
"""


async def myRead(reader, _addr, lock):
    cur_amount = 0
    last_time = int(time.time())
    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f"select cur_amount,last_time from user where
usrname={_addr!r}") as cursor:
            for row in cursor:
                cur_amount = row[0]
                last_time = row[1]
                print(f'cur_amount of client {_addr!r}: ',cur_amount)
    increment = (int(time.time())-last_time) * rate
    #lock
    await lock.acquire()
    cur_amount = min(cur_amount + increment, capacity)
    #unlock
    lock.release()
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()

    increment = (int(time.time())-last_time) * rate
    #lock
    await lock.acquire()
    cur_amount = min(cur_amount + increment, capacity)
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    while cur_amount < 2048:
        increment = (int(time.time())-last_time) * rate
        #lock
        await lock.acquire()
        cur_amount = min(cur_amount + increment, capacity)
        async with aiosqlite3.connect("user.db") as db:
            await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
            await db.commit()
        #unlock
        lock.release()
    #lock
    await lock.acquire()
    last_time = int(time.time())
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set last_time={last_time!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    data = await reader.read(2048)
    #lock
    await lock.acquire()
    cur_amount -= len(data)
    async with aiosqlite3.connect("user.db") as db:
        await db.execute(f"update user set cur_amount={cur_amount!r} where
usrname={_addr!r}")
        await db.commit()
    #unlock
    lock.release()
    return data


async def handle_socks5_echo(reader, writer):
    print('remote socks5 proxy server')

    data = await reader.read(2048)

    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1 : addr_len + 1].decode()

    port = struct.unpack('!H', data[addr_len + 1:addr_len + 3])[0]

    reply = struct.pack('!5B', 5, 0, 0, 3, addr_len) + address.encode() +
struct.pack('!H', port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
    except Exception as err:
        print('DomainName connection failed:', err)

    writer.write(reply)
    await writer.drain()

    in_to_out = Exchangemsg(reader, writer_out)
    out_to_in = Exchangemsg(reader_out, writer)
    await asyncio.gather(in_to_out, out_to_in)

"""
# http代理
async def handle_http_echo(reader ,writer):
    print('remote http proxy server')

    msg = await reader.readuntil(b':')
    addr = msg[: -1]
    msg = await reader.readuntil(b' ')
    port = msg[: -1]
    msg = await reader.readuntil(b'\r\n')
    version = msg[: -2]

    print('address: ', addr)
    print('port: ', port)
    print('version: ', version)

    msg = await reader.read(2048)
    try:
        reader_out, writer_out = await asyncio.open_connection(addr, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            out_to_in = Exchangemsg(reader_out, writer)
            in_to_out = Exchangemsg(reader, writer_out)
            await asyncio.gather(in_to_out, out_to_in)

        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
            print('connection failed: ', err)
"""


async def handle_http_echo(reader, writer):
    print('remote http proxy server')

    data = await reader.read(2048)
    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1:1 + addr_len].decode()
    print(address)
    port = struct.unpack('!H', data[1 + addr_len:1 + addr_len + 2])[0]
    print(port)
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection succeeded')
        try:
            print('send succeeded')

            in_to_out = Exchangemsg(reader, writer_out)
            out_to_in = Exchangemsg(reader_out, writer)
            await asyncio.gather(in_to_out, out_to_in)
        except Exception as err:
            print('send failed: ', err)
    except Exception as err:
        print('connection failed: ', err)


async def handle_tcp_echo(reader, writer):
    _addr = ''
    data = await reader.read(2048)

    addr_len = struct.unpack('!B', data[0:1])[0]
    address = data[1:addr_len + 1].decode()
    port = struct.unpack('!H', data[addr_len + 1:addr_len + 3])[0]
    _addr_len = struct.unpack('!B', data[addr_len + 3:addr_len + 4])[0]
    _addr = data[addr_len + 4:addr_len + 4 + _addr_len].decode()
    print(address, port)
    print(f'client {_addr!r} exchange data')
    try:
        reader_out, writer_out = await asyncio.open_connection(address, port)
        print('connection success')
    except:
        print('domainname connect error!')
    lock = asyncio.Lock()
    in_to_out = Exchangemsg(reader, writer_out, _addr, lock)
    out_to_in = Exchangemsg(reader_out, writer, _addr, lock)
    await asyncio.gather(in_to_out, out_to_in)


async def handle_confirm(reader, writer):
    flag = 1
    data = await reader.read(2048)
    option = struct.unpack("!B", data[0:1])[0]
    name_len = struct.unpack("!B", data[1:2])[0]

    name = data[2:2 + name_len].decode()
    password_len = struct.unpack("!B", data[2 + name_len:3 + name_len])[0]
    password = data[3 + name_len:3 + name_len + password_len].decode()

    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f"select usrname,usrpassword  from user") as
cursor:
            print("all info:")
            for row in cursor:
                print(row[0], row[1])
        if option == 0:
            await db.execute(f"insert into user (usrname,usrpassword) \
                values ({name!r},{password!r})")
            await db.commit()
            print('after insert:')
            async with db.execute(f"select usrname,usrpassword  from user") as
cursor:
                print("all info:")
                for row in cursor:
                    print(row[0], row[1])
            flag = 0
        elif option == 1:
            find = 1
            async with db.execute(f'select usrpassword  from user where
usrname="{name}"') as cursor:
                if len(list(cursor)) == 0:
                    flag = 2
                    find = 0

            if find == 1:
                async with db.execute(f'select usrpassword from user where
usrname="{name}"') as cursor:
                    for row in cursor:
                        if row[0] != password:
                            print(row[0], password)
                            flag = 10
                            print('wrong pwd')
    db.close()
    data0 = struct.pack("!B", flag)
    writer.write(data0)
    await writer.drain()
    if flag == 1:
        await handle_tcp_echo(reader, writer)

async def work():
    """
    server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
```

```
    server_socks5 = await asyncio.start_server(handle_socks5_echo, '127.0.0.1',
7080)
    print('-------------------------------------------')
    print('|        代理服务器IP: 127.0.0.1             |')
    print('|         socks5代理Port: 7080              |')
    print('|       http tunnel代理端口: 8080             |')
    print('|                                          |')
    print('|                         Designer: Xuzikang|')
    print('|                           Time: 2020.11.09|')
    print('-------------------------------------------')

    async with server_http:
        await server_http.serve_forever()
    async with server_socks5:
        await server_socks5.serve_forever()

    await asyncio.gather(server_socks5, server_http)
    """
    global rate, capacity
    if sys.argv[1] == '-d':
        rate = 1000000
    else:
        rate = int(sys.argv[1])
    capacity = rate
    async with aiosqlite3.connect("user.db") as db:
        await db.execute("delete from user")
        await db.commit()
        print("清除数据")
        await db.execute("drop table user")
        await db.commit()

        await db.execute('''create table user
        (usrname test primary key not null,
        usrpassword text not null);''')
        await db.execute("insert into user (usrname,usrpassword) \
        values ('xuzikang','2018211514')")
        await db.execute("insert into user (usrname,usrpassword) \
        values ('gofire','2147483648')")
        await db.commit()

    while True:
        confirm = await asyncio.start_server(handle_confirm, '127.0.0.1', 5080)
        """
        server_socks5 = await asyncio.start_server(handle_socks5_echo,
'127.0.0.1', 7080)
        server_http = await asyncio.start_server(handle_http_echo, '127.0.0.1',
8080)
        print('-------------------------------------------')
        print('|        代理服务器IP: 127.0.0.1             |')
        print('|         socks5代理Port: 7080              |')
        print('|       http tunnel代理端口: 8080             |')
        print('|                                          |')
        print('|                         Designer: Xuzikang|')
        print('|                           Time: 2020.11.16|')
        print('-------------------------------------------')

        async with server_socks5:
            await server_socks5.serve_forever()
        async with server_http:
            await server_http.serve_forever()
        """
        async with confirm:
            await confirm.serve_forever()
        # await asyncio.gather(server_socks5, server_http)

if __name__ == '__main__':
    asyncio.run(work())

    pass
```

remoteRest.py

```
import sys
import time
import struct
import socket
import select
import asyncio
import sqlite3
import aiosqlite3

from sanic import Sanic
from sanic import response
from sanic import exceptions

app = Sanic('RemoteAdmin')
app.config.DB_NAME = 'user.db'


@app.get('/user/<name>')
async def FindUser(req, name):
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        async with db.execute("SELECT password,cur_amount FROM user WHERE
userName=?",(name,)) as cursor:
            async for row in cursor:
                user = {'userName':name, 'password':row[0],'cur_amount':row[1]}
                return response.json(user)
    return response.json({}, status = 404)

@app.get('/user')
async def listUsers(req):
    userList = list()
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        async with db.execute("SELECT userName,password,cur_amount FROM user;")
as cursor:
            async for row in cursor:
                user = {'userName':row[0], 'password':row[1],
'cur_amount':row[2]}
                userList.append(user)
    return response.json(userList)

@app.delete('/user/<name>')
async def deleteUser(req, name):
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        await db.execute("DELETE FROM user WHERE userName=?",(name,))
```

```python
        await db.commit()
    return response.json({})

@app.post('/user')
async def addUser(req):
    userName = req.json.get('userName')
    password = req.json.get('password')
    if not userName or not password:
        return response.text(f'err userName={userName} password={password}',
status = 400)
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        await db.execute("INSERT INTO
user(userName,password,cur_amount,last_time) VALUES(?,?,?,?)",
(userName,password,int(sys.argv[1]),time.time()))
        await db.commit()
    return response.json({})

@app.put('/user/<userName>')
async def updatePassword(req, userName):
    password = req.json.get('password')
    if not userName or not password:
        return response.text(f'err userName={userName} password={password}',
status = 400)
    async with aiosqlite.connect(app.config.DB_NAME) as db:
        await db.execute("UPDATE user SET password=? WHERE userName=?",
(password, userName))
        await db.commit()
    return response.json({})

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

if __name__ == '__main__':
    app.run(host='0.0.0.0', port = 8788)
```