一、背景

现在，在生活中，无论是学习，工作，还是娱乐，我们几乎都离不开网络，但是能帮助我们个性化处理网络的工具似乎也不是很多，当我们不满足360等安全软件提供的网络数据信息，但又觉得利用抓包软件分析过于麻烦；当我们想像写日记一样记录我们的浏览记录，但又不满足"历史纪录"提供的那少的可怜的信息；当我们想同时处理多个需要用到网络的任务，但又想自己分配他们的带宽...
是的，大众软件不需要配置不需要操作，但它能提供的数据少的可怜。专业软件数据详尽，但可能过于详尽，缺乏统计。我们也许需要一种快捷而精致的软件，让我们快速的获取有关自己网络的细致信息，并且能够配置自己的网络。也许，我们所设计的这款软件，更多更多和我们有相同想法的人，都想要。

二、环境及技术

1、环境

windows

python 3.8

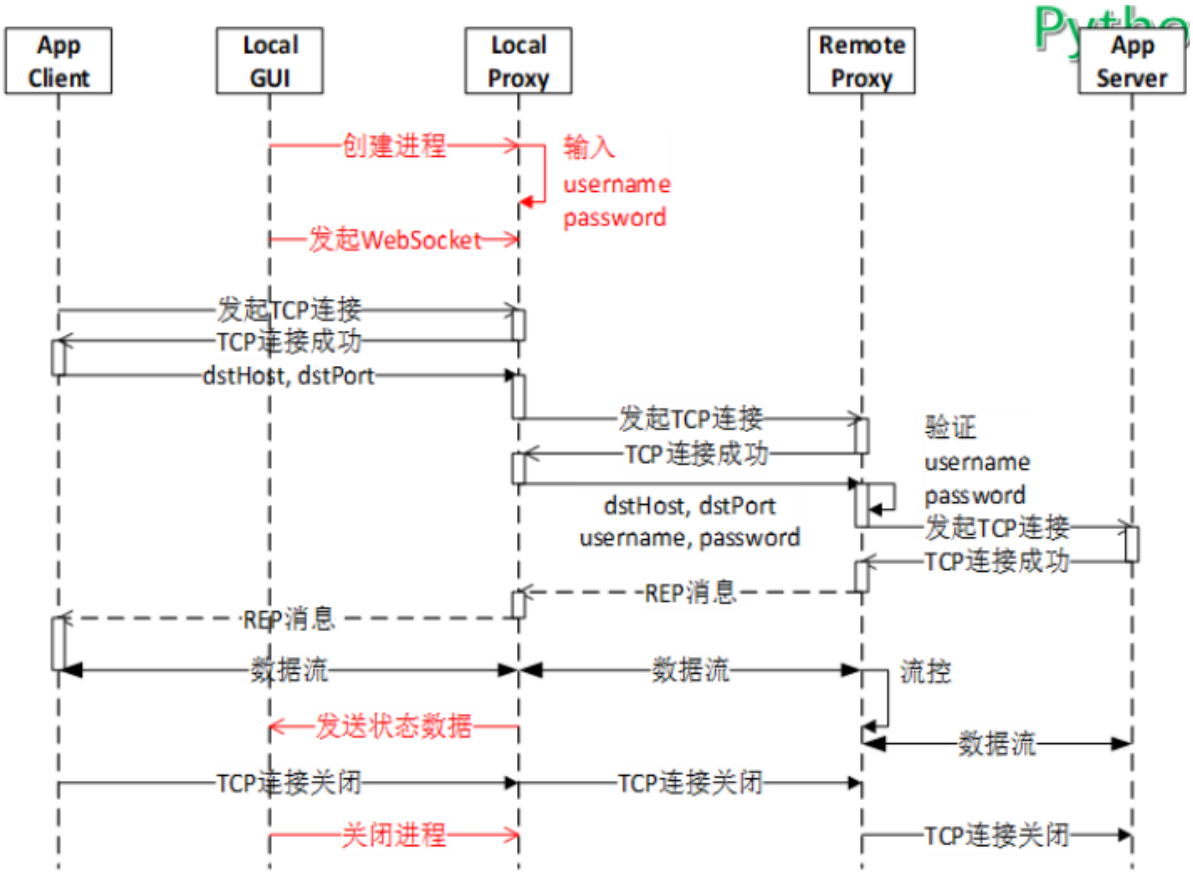
2、技术

协程

sqlite

三、产品介绍

1、基本功能



①支持代理

已实现功能

http和socks5

这是目前我们的软件支持的代理方式，在浏览器设置代理后，启动我们的软件，浏览器在使用的过程中，所有数据将经过我们的软件发送出去，代理方式可自动识别，用户无需做更多操作。在后续的版本中，我们将实现全局代理的方式，使得用户的使用更加方便。

计划实现功能

其他协议的通信

② 获取流量数据

在代理服务器local端进行处理，计算出数据传输的带宽，将传输的带宽数据传输到GUI端口进行显示

③ 数据库（内含身份验证）

Ⅰ用户信息系统

已实现功能

用户将通过用户名和密码登陆账号，用以区别。用户信息将被存储在数据库里面

计划功能：

将实现注册功能，用户登陆自己的账号之后，可以设置一些自己的偏好设置，，同时，网络监管模块将根据不同账号进行更加细致从处理

Ⅱ 网络监管系统

已实现功能：

通过代理，用户通过我们的软件可以获取浏览器接收和发送的所有信息，我们将记录用户访问过的域名及其ip和近期访问次数。

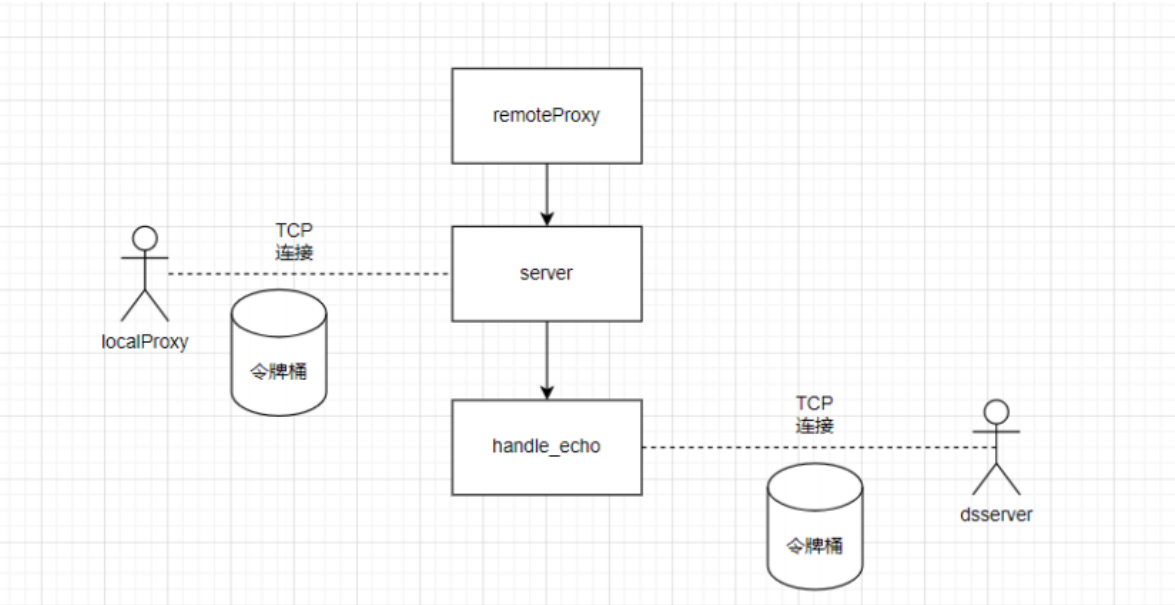
计划功能：

① 我们的软件可以根据近期访问次数，对近期经常访问的网页进行内部加速，网页的储存与替换采用LFU算法。内部加速将使得我们的软件对用户原网速的影响更小，从而在其有较高网速需求的时候，不会因为我们的软件被限制。同时，用户可以通过设置“临界网速”，配置软件，当需要的网速大于临界网速时，我们的软件将会逐渐降低检测力度，直到放弃检测，这将最大限度的在实现我们软件功能的同时，降低我们的软件对用户使用的影响。

② 我们的软件将根据统计信息，生成报表，用户可选择输出方式，比如csv等。

软件将给出访问信息的统计信息，显示主要网站的流量数据，按天，周或者月，统计新出现的网页，和消失的网页。用户在查看时，如果莫名出现陌生网页，则怀疑被攻击的可能性，当即时更改重要密码。同时用户也可以查看自己前一段时间的网络使用情况，类似于日记功能，方便用户的自我管理

④ 令牌桶



实现过程

因为每次发送信息，都会访问一次令牌桶，我们利用这个特性去实现令牌桶算法。

① 访问令牌桶时，加锁。

② 访问令牌桶动都会记录一次当前时刻的时间，用当前时刻的时间减去上一次访问令牌桶的时间，再乘以生成令牌桶的速率，可以得到现有的令牌数。（这个令牌数不能超过令牌桶容量，如果超出了，则令当前令牌数等于令牌桶最大容量）

③ 在访问令牌桶时，会传来一个参数，这个参数是需要消耗的令牌数量，如果当前令牌数量多于需要消耗的令牌数，则相减后得到剩余令牌，然后释放锁；反之，则释放锁之后，等待下一次访问令牌桶，直至当前令牌数量多余需要消耗的令牌数量。

计划实现功能

新增一个设置，用户可以控制是否限速。如果不限速。

⑤ 流量数据处理

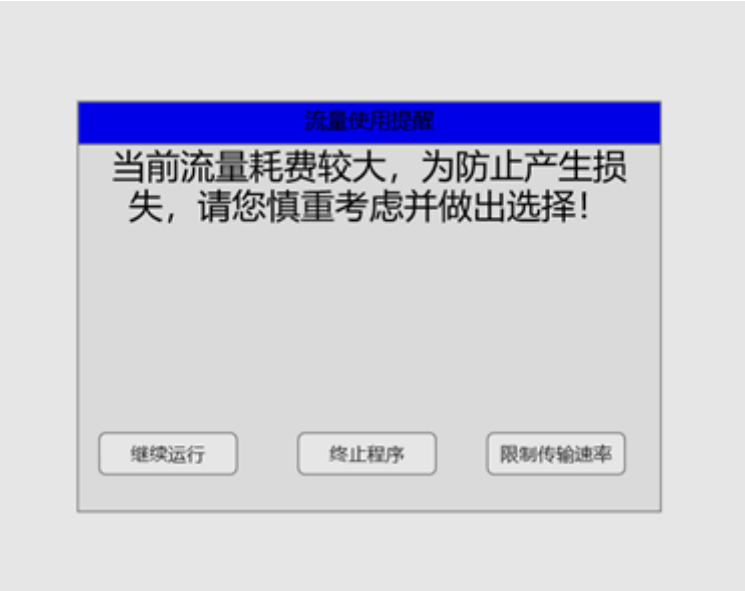
代理服务器的基本功能已经实现，在此基础上，根据对速率的监控，定期统计速率数据并对速率的增长趋势进行分析，若通过代理服务器的数据传输带宽有突发增长并持续一段时间，警示用户，用户可自己进行决策来确定是否继续，此功能的主要用途有两点：

① 防止流量使用过度而对用户带来损失。

② 进入部分网页点击不慎产生的捆绑下载。

计划实现功能

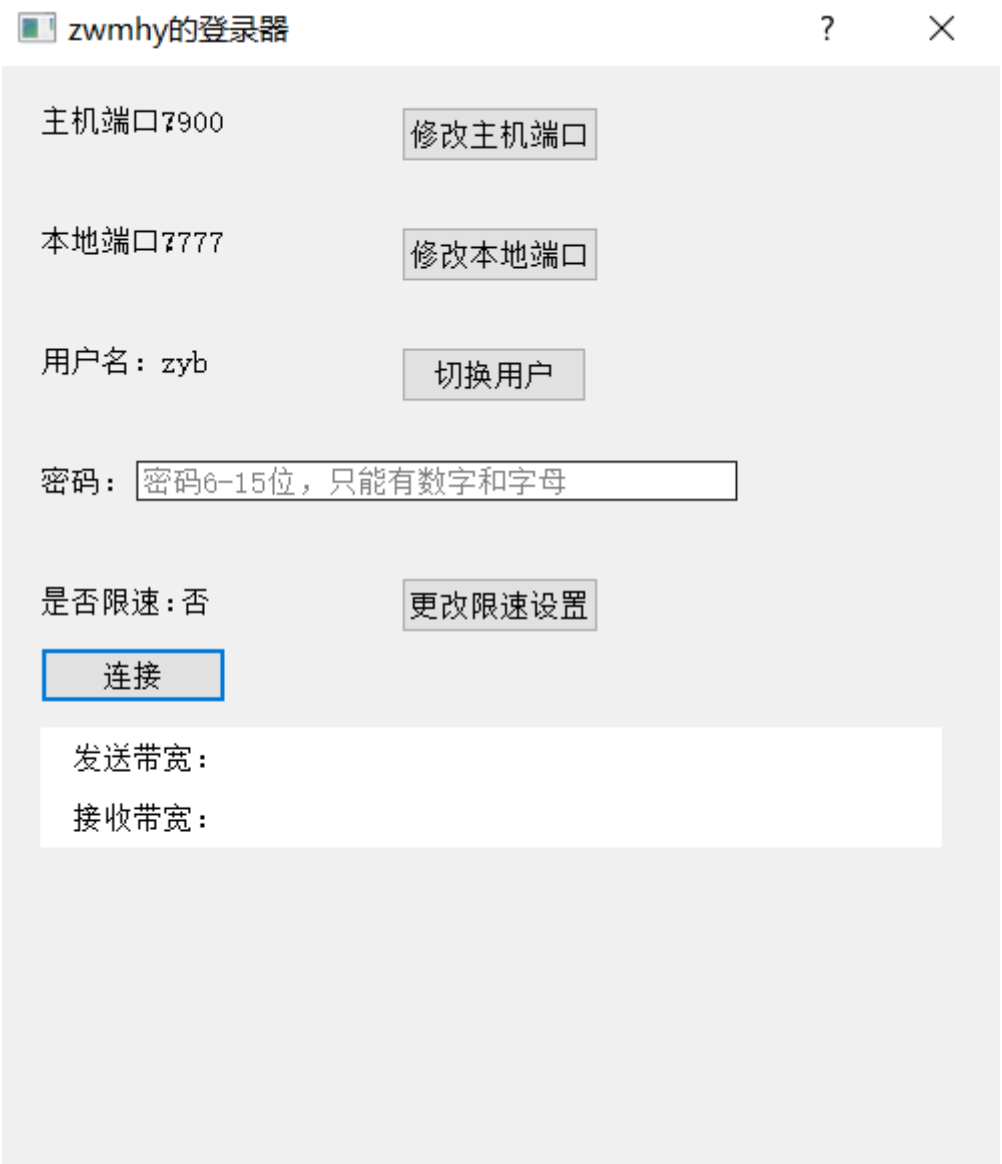
周期性检测传输带宽，用户在使用代理时如果带宽突发变大并且持续一定时间段，弹出警示框提醒用户，界面和选项初步计划如下图：

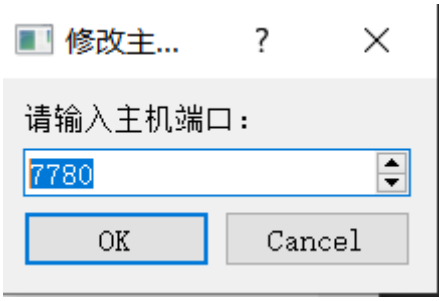


用户可选择继续使用代理传输，或者终止代理，或者通过限制传输速率来降低流量的消耗速度。限速连接另一个扩展功能，用户可以输入期望速率然后将数据传输到限速功能，采用令牌桶算法，对流量数据进行限速。用户可以根据GUI端的速率显示选择合理的限速速率。

⑥ GUI

GUI样式





使用技术

PyQt5

显示内容

① 主机端口

② 本地端口

③ 用户名

④ 密码（用掩码显示）

⑤ 显示带宽

使用方式

① 更改主机端口、本地端口、用户名、密码、是否限速。为了避免冲突，我们设置了本机端口和本地端口的范围。

② 点击连接，GUI将与 local 建立 websocket 连接，可以实时监控流量数据，并将其显示在 GUI 上。

2、拓展功能

① 完善令牌桶限速功能

原来的写法虽然也可以限速，但是本质上是错误的，这个版本将原来的错误修正了。

初始版本：每个用户有独属于自己的令牌数用来限速。

现在的版本：所有用户共用一个令牌数，只有被分配到令牌的用户才能用代理浏览网页。

② 提取信息

在 localproxy 和 remoteproxy 转发数据包的过程中，获取数据包的信息，并且进行数据的提取。

利用正则表达式，获取网址和 HTTP版本

```
第 0 个文件：   tsbrowser.xiangtatech, rwww.baidu
第 1 个文件：   tsbrowser.xiangtatech, rwww.baidu
第 2 个文件：   tsbrowser.xiangtatech, rwww.baidu
第 3 个文件：   tsbrowser.xiangtatech, rwww.baidu
```

```
第 0 个文件：   tsbrowser.xiangtatech.com, rwww.baidu.com
第 1 个文件：   tsbrowser.xiangtatech.com, rwww.baidu.com
第 2 个文件：   tsbrowser.xiangtatech.com, rwww.baidu.com
第 3 个文件：   tsbrowser.xiangtatech.com, rwww.baidu.com
```

```
第 0 个文件：   http/1.1
第 1 个文件：   http/1.1
第 2 个文件：   http/1.1
第 3 个文件：   http/1.1
```

③ 使用大数据知识分布式处理数据（未完全完成，只完成了框架）

环境：

Java：java-1.8.0

Python：python-3.8.5

Scala：scala-2.11.8

Flume：apach-flume-1.9.0

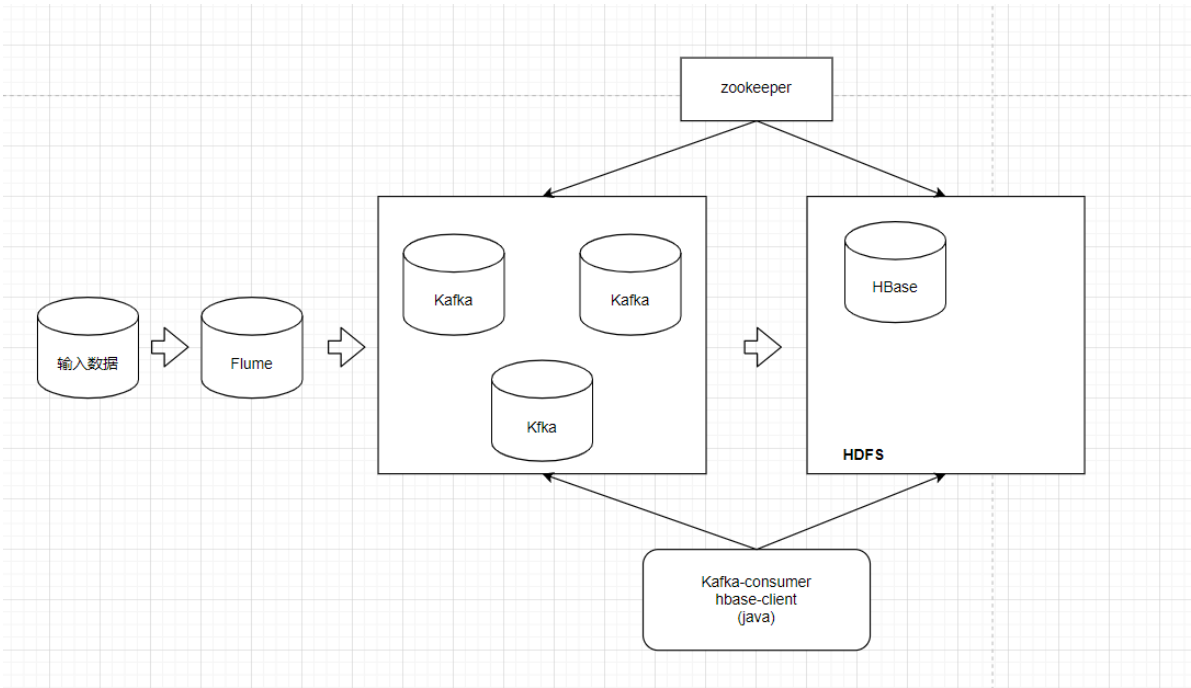
Kafka：kafka_2.11-2.4.1

Hadoop：hadoop-2.10.1

HBase：hbase-2.3.5

Spark：spark-2.4.7

在 linux 配置好相关环境之后，将提取出来的数据信息从flume输入，编写 java 程序，按照下面的流程图将输入的信息存入到HBase的表里。



接下来编写处理数据信息的java程序，将提交到 spark 上运行，进行分布式地处理这些数据，并将结果写入HDFS中。

这样我们可以更快、更准地获取到我们想得到的结果。比如：访问比较频繁的网页、存在安全问题无法访问的网页……

现在这个框架我们已经搭好了，但是因为时间的关系和基础功能出现了一些bug，我们没有继续去做这方面的内容，这个拓展部分只完成到了这个程度，没有去思考我们应该根据提取出来的数据去做什么样的数据处理。

四、优势

能获取更详细的数据

比起一些大众的浏览器自带的服务器，比如360，只可以简单地监控一下网速，而我们地产品，除了网速，还可以提供流量、网页访问次数等一些其他数据。

操作简单

wireshark等抓包软件呢，可以获取更为详细的数据，但是这些软件操作相对麻烦，不适合大众使用。相对的，我们的产品给这些数据提供一些接口，用户可以非常简单地使用。

五、未来发展方向

目前我们的软件主要应用于个人计算机，帮助个人统计网络的使用信息。经过一些修改，它同样可以用于一个局域网的流量管理与监控，我们的软件将具有应用到企业里面的潜力。在后续的版本中，我们将会考虑使用java或则c++重写我们的软件，实现更高的效率，同时优化我们的代码，进一步提高效率。

六、附录

GUI.py

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtNetwork import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebSockets import *
import sys
import os
import humanfriendly

class Window(QDialog):
    def __init__(self, parent=None):
        QDialog.__init__(self, parent)
        self.setGeometry(700,50,500,550)
        self.setWindowTitle('网络管理器')
        # self.resize(350,100)
        self.username='zyb'
        self.remoteport=7900
        self.localport=7777
        self.lim_flag = '否'
        self.startBtn = QPushButton('连接',self)
        self.startBtn.clicked.connect(self.startClicked)
        self.startBtn.move(20,290)

        self.lb1 = QLabel('主机端口：',self)
        self.lb1.move(20,20)

        self.lb2 = QLabel('本地端口：',self)
        self.lb2.move(20,80)

        self.lb3 = QLabel('用户名：',self)
        self.lb3.move(20,140)

        self.lb4 = QLabel('密码：',self)
        self.lb4.move(20,200)

        self.lb11 = QLabel('是否限速：', self)
        self.lb11.move(20, 260)

        self.bt1 = QPushButton('修改主机端口',self)
        self.bt1.move(200,20)

        self.bt2 = QPushButton('修改本地端口',self)
        self.bt2.move(200,80)

        self.bt3 = QPushButton('切换用户',self)
        self.bt3.move(200,140)

        self.bt5 = QPushButton('更改限速设置', self)
        self.bt5.move(200, 255)

        # self.bt4 = QPushButton('密码',self)
        # self.bt4.move(200,200)
```

```
        self.lb6 = QLabel('7900',self)
        self.lb6.move(80,20)

        self.lb7 = QLabel('7777',self)
        self.lb7.move(80,80)

        self.lb8 = QLabel('zyb',self)
        self.lb8.move(80,140)

        self.lb12 = QLabel('否', self)
        self.lb12.move(90, 260)

        self.edit = QLineEdit(self)
        self.edit.installEventFilter(self)

        #怎么布局在布局篇介绍过，这里代码省略...

        self.edit.setContextMenuPolicy(Qt.NoContextMenu)
        self.edit.setPlaceholderText("密码6-15位，只能有数字和字母")
        self.edit.move(68,197)
        self.edit.setFixedSize(300,20)
        self.edit.setEchoMode(QLineEdit.Password)

        regx = QRegExp("^[0-9A-Za-z]{14}$")
        validator = QRegExpValidator(regx, self.edit)
        self.edit.setValidator(validator)

        self.sendBandwidthLabel = QLabel(self)
        self.sendBandwidthLabel.setText('  发送带宽：')
        self.sendBandwidthLabel.resize(450,30)
        self.sendBandwidthLabel.move(20,330)
        self.sendBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: white")

        self.recvBandwidthLabel = QLabel(self)
        self.recvBandwidthLabel.setText('  接收带宽：')
        self.recvBandwidthLabel.resize(450,30)
        self.recvBandwidthLabel.move(20,360)
        self.recvBandwidthLabel.setStyleSheet("color: rgb(0, 0, 0);background-
color: white")
        # self.lb9 = QLabel('******',self)
        # self.lb9.move(120,200)

        self.bt1.clicked.connect(self.showDialog)
        self.bt2.clicked.connect(self.showDialog)
        self.bt3.clicked.connect(self.showDialog)
        self.bt5.clicked.connect(self.showDialog)
        # self.bt4.clicked.connect(self.showDialog2)

        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 showDialog(self):
        sender = self.sender()
```

```
        if sender == self.bt1:
            text, ok = QInputDialog.getInt(self, '修改主机端口', '请输入主机端
口：',min=7780,max=8000)
            if ok:
                self.lb6.setText(str(text))
                self.remoteport=text
        elif sender == self.bt2:
            text, ok = QInputDialog.getInt(self, '修改本地端口', '请输入本地端口：',
min = 7000,max=7779)
            if ok:
                self.lb7.setText(str(text))
                self.localport=text
        elif sender == self.bt3:
            text, ok = QInputDialog.getText(self, '切换用户', '请输入用户名')

            if ok:
                self.lb8.setText(text)
                self.username=text
        elif sender == self.bt5:
            if self.lim_flag == '否':
                self.lim_flag = '是'
            else:
                self.lim_flag = '否'
            self.lb12.setText(self.lim_flag)

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

    def processStarted(self):
        # print("p start")
        process = self.sender() # 此处等同于 self.process 只不过使用sender适应性更好
        processId = process.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.textMessageReceived.connect(self.websocketMsgRcvd)
            self.websocket.open(QUrl(f'ws://127.0.0.1:{int(self.localport)+1}'))
            print("connect")
        except:
            print("disconnect")

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

    def startClicked(self):
        btn = self.sender()
        # print("so?")
```

```
        text = btn.text().lower()
        if text.startswith('连接'):
            # listenPort = self.listenPortLine.text()
            # username = self.usernameLine.text()
            username = self.username
            remoteport = self.remoteport
            localport=self.localport
            password = self.edit.text()
            print(username,password,remoteport,localport)
            # consolePort = self.consolePortLine.text()
            # remoteHost = self.remoteHostLine.text()
            # remotePort = self.remotePortLine.text()
            # pythonExec = os.path.basename(sys.executable)
            # 从localgui启动localproxy直接使用-w 提供用户密码，不再使用命令行交互输入，因为有些许问题
            # cmdLine = f'{pythonExec} work6.py local -p {listenPort} -u {username} -w {password} -k {consolePort} {remoteHost} {remotePort}'
            # cmdLine="D:\v2py\.vscode\pydwork1\m6\local.py zyb 135800"
            cmdLine=f'python .vscode\pydwork1\m6\pyqtwebsocket\localtest.py {username} {password} {localport} {remoteport}'
            # .vscode\pydwork1\m6\finalv1\local.py
            # .vscode\pydwork1\m6\pyqtwebsocket\localtest.py
            # log.debug(f'cmd={cmdLine}')
            # print("now")
            self.process.start(cmdLine)
            # print("so")
        else:
            self.process.kill()
            self.startBtn.setText('连接')

    def websocketConnected(self):
        self.websocket.sendTextMessage('secret')

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

    def websocketMsgRcvd(self, msg):
        # log.debug(f'msg={msg}')
        # print("here sockets")
        sendBandwidth, recvBandwidth, *_ = msg.split()
        nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
        print(f'{nowTime} {humanfriendly.format_size(int(sendBandwidth))}')
        self.sendBandwidthLabel.setText(f'  发送带宽:   {nowTime} {humanfriendly.format_size(int(sendBandwidth))}')
        # print("something")
        # print(f'{nowTime} {humanfriendly.format_size(int(sendBandwidth))}')
        self.recvBandwidthLabel.setText(f'  接收带宽:   {nowTime} {humanfriendly.format_size(int(recvBandwidth))}')

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Window()
    ex.show()
    sys.exit(app.exec_())
```

local.py

```
import asyncio
import struct
import logging
import string
import sys
import websockets
import traceback

logging.basicConfig(level=logging.WARNING)  # 设置日志级别
#read和write的大小
rwsize=1024
username=str(sys.argv[1])
password=str(sys.argv[2])
gSendBandWidth = 0
gRecvBandWidth = 0
now_rdata_len = 0
now_wdata_len = 0

async def localConsole(ws, path):
    global gSendBandWidth
    global gRecvBandWidth
    global now_rdata_len
    global now_wdata_len
    try:
        while True:
            await asyncio.sleep(1)
            print(f'this is {gSendBandWidth} {gRecvBandWidth}')
            msg = await ws.send(f'{gSendBandWidth} {gRecvBandWidth}')
            now_rdata_len = 0
            now_wdata_len = 0
    except websockets.exceptions.ConnectionClosedError as exc:
        # log.error(f'{exc}')
        print(f'{exc}')
    except websockets.exceptions.ConnectionClosedOK as exc:
        logging.error(f'{exc}')
    except Exception:
        logging.error(f'{traceback.format_exc()}')
        exit(1)

async def clacbrandwidth():
    global gSendBandWidth
    global gRecvBandWidth
    global now_rdata_len
    global now_wdata_len
    # gSendBandWidth = 0
    # gRecvBandWidth = 0
    # last_rdata_len = 0
    # last_wdata_len = 0
    while True:
        gSendBandWidth = now_wdata_len# - last_wdata_len
        gRecvBandWidth = now_rdata_len# - last_rdata_len
        # last_wdata_len = now_wdata_len
        # last_rdata_len = now_rdata_len
        # gSendBandWidth=4
        # print(f'接收带宽：{gRecvBrandWidth!r}')
```

```
        # print(f'发送带宽：{gSendBrandWidth!r}')
        await asyncio.sleep(1)

async def my_connect(reader, writer):
    global now_rdata_len
    global now_wdata_len
    data1 = await reader.read(rwsize)#tcp1
    addr = writer.get_extra_info('peername')
    logging.info(f"connect from {addr!r}")
    if len(data1) < 3:
        logging.warning("too short!")
        writer.close()
        return
    request = struct.unpack('!BBB', data1[:3])
    #http
    if request[0] == 67  and request[1] == 79 and request[2] == 78:
        logging.info('in http')
        hdata=data1.decode('utf8')
        hdata1=hdata.split('\r\n')
        hdata2=hdata1[1].split(':')
        t=str.maketrans(string.ascii_letters,string.ascii_letters,' ')
        h_url=hdata2[1].translate(t)
        h_port=hdata2[2].translate(t)
        h_len=len(h_url)
        success=b"HTTP/1.0 200 Connection Established\r\n\r\n"
        try:
            reader_remote, writer_remote = await
asyncio.open_connection('127.0.0.2', int(sys.argv[4]))
            writer.write(success)#tcp2
            await writer.drain()
            logging.info(f'connect success ! {h_url}')
        except (TimeoutError, ConnectionRefusedError) as _:
            logging.warning(f'connect failed ! {h_url}')
            writer.write(success)
            await writer.drain()
            writer.close()
            return
        writer_remote.write(data1)#rtcp1
        await writer_remote.drain()
        #login
        login_info=username+'+'+password
        login_info=login_info.encode()
        writer_remote.write(login_info)
        await writer_remote.drain()
        data = await reader_remote.read(rwsize)
        request = struct.unpack('!BB', data[:2])
        if request[0]!=5:
            writer_remote.close()
            logging.warning('incorrect password')
            return
    # if request[1]==2:#cookie
    #     pass

        #
        writer_remote.write(data1)#rtcp1
        data = await reader_remote.read(rwsize)
        if data == success:
            logging.info('connection building successfully')
```

```
            up_stream = transport(reader, writer_remote, h_url)
            down_stream = transport(reader_remote, writer, h_url)
            await asyncio.gather(up_stream, down_stream)
    else:
        writer.write(b'\x05\x00') #tcp2
        await writer.drain()
        data2 = await reader.read(rwsize)#method1
        request = struct.unpack('!4B', data2[:4])
     #域名
        if request[0] == 5 and request[1] == 1 and request[3] == 3:
            host_len = struct.unpack('!B', data2[4:5])[0]
            host = data2[5:host_len + 5].decode()
            port = struct.unpack('!H', data2[host_len + 5:])[0]
            logging.info(f'len {host_len},host {host}, port {port}')
            try:
                reader_remote, writer_remote = await
asyncio.open_connection('127.0.0.2',int(sys.argv[4]))
                writer.write(struct.pack('!5B', 5, 0, 0, 3, host_len) +
host.encode() + struct.pack('!H', port))#tmethod2
                await writer.drain()
                logging.info(f'connect success ! {host}')
            except (TimeoutError, ConnectionRefusedError) as _:
                logging.warning(f'connect failed ! {host}')
                writer.write(struct.pack('!5B', 5, 3, 0, 3, host_len) +
host.encode() + struct.pack('!H', port))
                await writer.drain()
                writer.close()
                return
            await
connect_remote(writer_remote,reader_remote,writer,reader,data2,data1)

    #ipv4地址
        if request[0] == 5 and request[1] == 1 and request[3] == 1:
            ip = '.'.join([str(a) for a in struct.unpack('!BBBB', data2[4:8])])
            port = struct.unpack('H', data2[-2:])[0]
            test1=struct.unpack('!BBBB', data2[4:8])
            print(f'ip {ip}, port {port}, yuan{test1}')
            try:
                reader_remote, writer_remote = await
asyncio.open_connection('127.0.0.2', int(sys.argv[4]))
                writer.write(struct.pack('!8B', 5, 0, 0, 1,
*struct.unpack('!BBBB', data2[4:8])) + struct.pack('!H', port))
                await writer.drain()
                logging.info(f'connect success ! {ip}')
            except (TimeoutError, ConnectionRefusedError) as _:
                logging.warning(f'connect failed ! {ip}, {repr(_)}')
                writer.write(struct.pack('!8B', 5, 3, 0, 1,
*struct.unpack('!BBBB', data2[4:8])) + struct.pack('!H', port))
                await writer.drain()
                writer.close()
                return
            await
connect_remote(writer_remote,reader_remote,writer,reader,data2,data1)

async def connect_remote(writer_remote,reader_remote,writer,reader,data2,data1):
    writer_remote.write(data1)
    await writer_remote.drain()
    data = await reader_remote.read(rwsize)
```

```
    request = struct.unpack('!BB', data[:2])
    if request[0]==5 and request[1] == 0:#tcp successful
        #login in:
        login_info=username+'+'+password
        login_info=login_info.encode()
        writer_remote.write(login_info)
        await writer_remote.drain()
        data = await reader_remote.read(rwsize)
        request = struct.unpack('!BB', data[:2])
        if request[0]!=5:
            writer_remote.close()
            return
    # if request[1]==2:#cookie
    #     pass
    #
        writer_remote.write(data2)
        await writer_remote.drain()
        datat = await reader_remote.read(rwsize)
        host='local'
        up_stream = transport(reader, writer_remote, host)
        down_stream = transport(reader_remote, writer, host)
        await asyncio.gather(up_stream, down_stream)

async def transport(reader, writer, host):
    global now_rdata_len
    global now_wdata_len
    while reader.at_eof:
        try:
            data = await reader.read(rwsize)
            now_rdata_len+=len(data)
            if not data:
                writer.close()
                break
        except (ConnectionAbortedError, ConnectionResetError) as _:
            writer.close()
            logging.warning(f'{host} quit {repr(_)}')
            break
        try:
            writer.write(data)
            now_wdata_len+=len(data)
            await writer.drain()
        except (ConnectionAbortedError, ConnectionResetError) as _:
            writer.close()
            logging.warning(f'{host} abnormal quit {repr(_)}')
            break
        logging.info(f'{host} quit')

async def main():
    server = await asyncio.start_server(
    my_connect, '0.0.0.0', int(sys.argv[3]))
    addr = server.sockets[0].getsockname()
    logging.warning(f'Serving on {addr}')
    logging.info(f'username: {str(sys.argv[1])}  password: {str(sys.argv[2])}')
    # my_connect
    asyncio.create_task(clacbrandwidth())
    ws_server = await websockets.serve(localConsole, '127.0.0.1',
int(sys.argv[3])+1)
    asyncio.create_task(clacbrandwidth())
```

```
    # asyncio.create_task(clacbrandwidth())
    async with server:
        await server.serve_forever()

loop = asyncio.get_event_loop()
loop.run_until_complete(asyncio.gather(main()))
loop.run_forever()
```

remote.py

```
import asyncio
import struct
import logging
import string
import sqlite3
import aiosqlite3
import time
import sys
import socket


logging.basicConfig(level=logging.WARNING)  # 设置日志级别
#read和write的大小
rwsize=1024
CapacityTimes=1
ratetimes=2000
global lockall
checktime=1
lockall=asyncio.Lock()
fast_connect=1
if_find_ip=0
async def main1():
    async with aiosqlite3.connect("user.db") as db:
        if len(sys.argv)==1:
            print("清除数据,并新建表")
            await db.execute("DROP TABLE IF EXISTS list")
            await db.commit()
        else:
            print("新建表")
        await db.execute('''CREATE TABLE list
        (name VARCHAR(30) PRIMARY KEY     NOT NULL,
        passwd         TEXT    NOT NULL,
        bps            INT     NOT NULL,
        current_amount DOUBLE  NOT NULL,
        last_time      DOUBLE  NOT NULL
        );''')
        await db.execute("INSERT INTO list
(name,passwd,bps,current_amount,last_time) \
        VALUES ('zyb','135800',2000,0,0)")

        await db.commit()

async def fast_connection():
    async with aiosqlite3.connect("user1.db") as db:
        if len(sys.argv)==1:
            print("清除数据,并新建表")
```

```
            await db.execute("DROP TABLE IF EXISTS FAST_CONNECTION")
            await db.commit()
        else:
            print("新建表")
        await db.execute('''CREATE TABLE FAST_CONNECTION
        (name VARCHAR(30) PRIMARY KEY     NOT NULL,
        ip         TEXT    NOT NULL,
        fluence            INT     NOT NULL

        );''')

        await db.commit()

async def check_name(host):
    # print("进入加速系统")
    global if_find_ip
    async with aiosqlite3.connect("user1.db") as db:
        await db.execute(f'update FAST_CONNECTION set fluence=fluence-1 where
name="{host}"')
        await db.execute(f'delete from FAST_CONNECTION where fluence<0')
        await db.commit()
        async with db.execute(f'SELECT fluence from FAST_CONNECTION where name="
{host}"') as cursor:
            if len(list(cursor)) > 0:
                is_exist = 1
            else:
                is_exist = 0
        if is_exist == 1:
            async with db.execute(f'SELECT fluence,ip  from FAST_CONNECTION
where name="{host}"') as cursor:
                for row in cursor:
                    fluence = row[0]
                    the_ip = row[1]
                    if_find_ip = 1
                    logging.info(f'fluence:', fluence)
            await db.execute(f'update FAST_CONNECTION set fluence={fluence+2}
where name="{host}"')
            await db.commit()
            db.close()
            return
        else:
            myaddr = socket.getaddrinfo(host, 'http')
            # print(myaddr[0][4][0])
            if len(myaddr[0][4][0])<20:
                await db.execute(f'INSERT INTO FAST_CONNECTION (name,ip,fluence)
\
                    VALUES ("{host}","{myaddr[0][4][0]}",20)')
                await db.commit()


        db.close()
        if_find_ip=0
        return


async def my_connect(reader, writer):
    global lockall
    BaseRate=1000
```

```
    BaseCapacity=BaseRate*CapacityTimes
    data = await reader.read(rwsize)
    addr = writer.get_extra_info('peername')
    logging.info(f"connect from {addr!r}")
    if len(data) < 3:
        logging.warning("too short!")
        writer.close()
        return
    request = struct.unpack('!BBB', data[:3])
    if request[0] == 67  and request[1] == 79 and request[2] == 78:
        #identify
        login_info=await reader.read(rwsize)
        request =login_info.decode('utf8')
        id_info=request.split('+')
        logging.info(f'received request: username: {id_info[0]} ,password :
{id_info[1]}')
        #查数据库
        result=await checkID(id_info[0],id_info[1])
        if result>0:
            logging.info("legal user")
            checktime=result/10
            logging.info(f'get bps {BaseRate},capacity {BaseCapacity}')
            writer.write(b'\x05\x01')
            await writer.drain()
        else:
            writer.write(b'\x01\x01')
            writer.close()
            logging.info("illegal user")
            return
        data = await reader.read(rwsize)
        hdata=data.decode('utf8')
        hdata1=hdata.split('\r\n')
        hdata2=hdata1[1].split(':')
        t=str.maketrans(string.ascii_letters,string.ascii_letters,' ')
        h_url=hdata2[1].translate(t)
        h_port=hdata2[2].translate(t)
        h_len=len(h_url)
        success=b"HTTP/1.0 200 Connection Established\r\n\r\n"
        try:
            reader_remote, writer_remote = await asyncio.open_connection(h_url,
h_port)
            writer.write(success)
            await writer.drain()
            logging.info(f'connect success ! {h_url}')
        except (TimeoutError, ConnectionRefusedError) as _:
            logging.warning(f'connect failed ! {h_url}')
            writer.write(success)
            await writer.drain()
            writer.close()
            return
        # lock1=lockall
        lock1=asyncio.Lock()
        limit = TokenBucket(BaseRate,BaseCapacity,id_info[0],lock1)#设置获得令牌的
速度，令牌桶上限
        up_stream = transport(reader, writer_remote, h_url,limit,checktime)
        down_stream = transport(reader_remote, writer, h_url,limit,checktime)
        await asyncio.gather(up_stream, down_stream)
    else:
```

```
        writer.write(b'\x05\x00')
        await writer.drain()
        #identity
        login_info=data = await reader.read(rwsize)
        request =login_info.decode('utf8')
        id_info=request.split('+')
        logging.info(f'received request: username: {id_info[0]} ,password :
{id_info[1]}')
        #查数据库
        BaseRate=1000
        # BaseCapacity=10*BaseRate
        result=await checkID(id_info[0],id_info[1])
        if result>0:
            logging.info("legal user")
            checktime=result/10
            logging.info(f'get bps {BaseRate},capacity {BaseCapacity}')
            writer.write(b'\x05\x01')
            await writer.drain()
        else:
            writer.write(b'\x01\x01')
            logging.info("illegal user")
            writer.close()
            return

        data = await reader.read(rwsize)
        request = struct.unpack('!4B', data[:4])
        # print(data)

            #域名
        if request[0] == 5 and request[1] == 1 and request[3] == 3:
            host_len = struct.unpack('!B', data[4:5])[0]
            host = data[5:host_len + 5].decode()
            # 快速连接
            if fast_connect==1:
                pass
                # print("判断是否加速")
                await check_name(host)

            if if_find_ip==0:
                # print("未加速")
                port = struct.unpack('!H', data[host_len + 5:])[0]
                logging.info(f'len {host_len},host {host}, port {port}')
                try:
                    reader_remote, writer_remote = await
asyncio.open_connection(host, port)
                    writer.write(struct.pack('!5B', 5, 0, 0, 3, host_len) +
host.encode() + struct.pack('!H', port))
                    await writer.drain()
                    logging.info(f'connect success ! {host}')
                except (TimeoutError, ConnectionRefusedError) as _:
                    logging.warning(f'connect failed ! {host}')
                    writer.write(struct.pack('!5B', 5, 3, 0, 3, host_len) +
host.encode() + struct.pack('!H', port))
                    await writer.drain()
                    writer.close()
                    return
            # lock1=lockall
                lock1=asyncio.Lock()
```

```
                limit = TokenBucket(BaseRate,BaseCapacity,id_info[0],lock1)#设置获得令牌的速度，令牌桶上限
                up_stream = transport(reader, writer_remote, host,limit,checktime)
                down_stream = transport(reader_remote, writer, host,limit,checktime)
                await asyncio.gather(up_stream, down_stream)
            else:
                # 加速系统
                logging.info(f'speed up!')
                print("speed up")

        #ipv4地址
        if request[0] == 5 and request[1] == 1 and request[3] == 1:
            ip = '.'.join([str(a) for a in struct.unpack('!BBBB', data[4:8])])
            port = struct.unpack('H', data[-2:])[0]
            test1=struct.unpack('!BBBB', data[4:8])
            # print(f'ip {ip}, port {port}, yuan{test1}')
            try:
                reader_remote, writer_remote = await asyncio.open_connection('127.0.0.2', 7778)
                writer.write(struct.pack('!8B', 5, 0, 0, 1, *struct.unpack('!BBBB', data[4:8])) + struct.pack('!H', port))
                await writer.drain()
                logging.info(f'connect success ! {ip}')
            except (TimeoutError, ConnectionRefusedError) as _:
                logging.warning(f'connect failed ! {ip}, {repr(_)}')
                writer.write(struct.pack('!8B', 5, 3, 0, 1, *struct.unpack('!BBBB', data[4:8])) + struct.pack('!H', port))
                await writer.drain()
                writer.close()
                return
            # lock1=lockall
            lock1=asyncio.Lock()
            limit = TokenBucket(BaseRate,BaseCapacity,id_info[0],lock1)#设置获得令牌的速度，令牌桶上限
            up_stream = transport(reader, writer_remote, ip,limit,checktime)
            down_stream = transport(reader_remote, writer, ip,limit,checktime)
            await asyncio.gather(up_stream, down_stream)

async def transport(reader, writer, host,limit,checktime):
    ltime=0
    while reader.at_eof:
        # print("want token")
        if ltime>checktime:
            ltime=0
            # await limit.consume(rwsize/8)#获得令牌则继续
        ltime=ltime+1
        # print('got token')
        try:
            data = await reader.read(rwsize)
            if not data:
                writer.close()
                break
        except (ConnectionAbortedError, ConnectionResetError) as _:
            writer.close()
            logging.warning(f'{host} quit {repr(_)}')
            break
```

```
        try:
            writer.write(data)
            await writer.drain()
        except (ConnectionAbortedError, ConnectionResetError) as _:
            writer.close()
            logging.warning(f'{host} abnormal quit {repr(_)}')
            break
        logging.info(f'{host} quit')

async def checkID(username,passwd):
    lpasswd="1111"
    async with aiosqlite3.connect("user.db") as db:
        async with db.execute(f'SELECT passwd  from list where name="
{username}"') as cursor:
            if len(list(cursor))>0:
                is_exist=1
            else:
                is_exist=0
                passwd='0000'
        if is_exist==1:
            async with db.execute(f'SELECT passwd,bps  from list where name="
{username}"') as cursor:
                for row in cursor:
                    lpasswd=row[0]
                    Rate=row[1]
                    logging.info(f'passwd:',lpasswd)
                    logging.info(f'bps:',Rate)
    db.close()
    if lpasswd==passwd:
        return Rate
    else:
        return -1


class TokenBucket:

    # rate是令牌发放速度，capacity是桶的大小
    def __init__(self, rate, capacity,username,lock):
        self._rate = rate
        self._capacity = capacity
        self.username=username
        self.lock=lock
        self._current_amount=0
        self._last_consume_time=0
    # token_amount是发送数据需要的令牌数
    async def consume(self, token_amount):
        flag=1
        async with aiosqlite3.connect("user.db") as db:
            async with db.execute(f'SELECT current_amount,last_time  from list
where name="{self.username}"') as cursor:
                for row in cursor:
                    self._current_amount=row[0]
                    self._last_consume_time=row[1]
                    logging.info(f'current_amount:',row[0])
                    logging.info(f'last_time:',row[1])
        while flag==1:
            # await self.lock.acquire()
```

```
            increment = (time.time() - self._last_consume_time) * self._rate  # 计算从上次发送到这次发送，新发放的令牌数量
            logging.info(f'increment {increment}')
            # print("increment ",increment)
            self._current_amount = min(
                increment + self._current_amount, self._capacity)  # 令牌数量不能超过桶的容量
            # await self.lock.acquire()
            if self._current_amount>token_amount:
                # print("check")
                await self.lock.acquire()
                flag=0
                self._last_consume_time = time.time()
                self._current_amount -= token_amount
                async with aiosqlite3.connect("user.db") as db:
                    await db.execute(f'update list set current_amount={self._current_amount} where name="{self.username}"')
                    await db.execute(f'update list set last_time={self._last_consume_time} where name="{self.username}"')
                    await db.commit()
                self.lock.release()

async def main():
    await main1()
    await fast_connection()
    server = await asyncio.start_server(
    my_connect, '127.0.0.2', 7900)

    addr = server.sockets[0].getsockname()
    logging.warning(f'Serving on {addr}')

    async with server:
        await server.serve_forever()


loop = asyncio.get_event_loop()
loop.run_until_complete(asyncio.gather(main()))
loop.run_forever()
```