程序设计实践大作业

——设计中继服务器

班号：2018211310

姓名：陈泊任

学号：2018211527

实验日期：2020.12.3

1、环境

系统：windows 10

语言：python

2、实现功能：

①使用asyncio的streams(coroutine based API)实现SOCKS5服务器

(1) 只实现CMD X'01'（即CONNECT）

(2) 只实现METHOD X'00'（即NO AUTHENTICATION REQUIRED）

(3) 客户端和目的服务器之间的数据转发为并发进行

②支持HTTP tunnel(即HTTP CONNECT method)，可用于HTTPS代理

客户端和目的服务器之间的数据转发为并发进行

③实现localProxy和remoteProxy

(1) 使用asyncio的streams API实现分离式的localProxy和remoteProxy两个单独程序

(2) localProxy收到的每个TCP连接单独建立代理TCP连接

(3) localProxy处理客户端发来的请求类型、连接remoteProxy、转发数据包

(4) remoteProxy解析客户端发来的请求包、连接目的服务器、转发数据包

(5) 客户端和目的服务器之间的数据转发为并发进行

④实现remoteProxy多账号认证

(1) 创建数据库

(2) 采用SQLite3数据库进行用户账号管理（用户名、密码）

(3) 使用aiosqlite操作SQLite3数据库

⑤实现remoteProxy对每个用户进行单独流控

(1) SQLite3数据库的每个用户的账号信息中增加宽带信息

(2) 使用令牌桶实现对每个用户进行单独流控

(3) 所有数据包对令牌桶的访问是互斥的

⑥设计localProxy的图形管理界面localGui

(1) 可通过图形界面关闭和开启localProxy

(2) 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）

(3) 使用QProcess类管理localProxy进程

(4) 可以实时查看localProxy的运行状态（是否运行、实时吞吐率）

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

3、特点：

协程

在这次的大作业中，我采用的是协程而不是多线程。

利用asyncio来构造协程，协程是异步的，需要加入到实践循环，然后由后者调用

协程的优点：

①协程有极高的执行效率，因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。

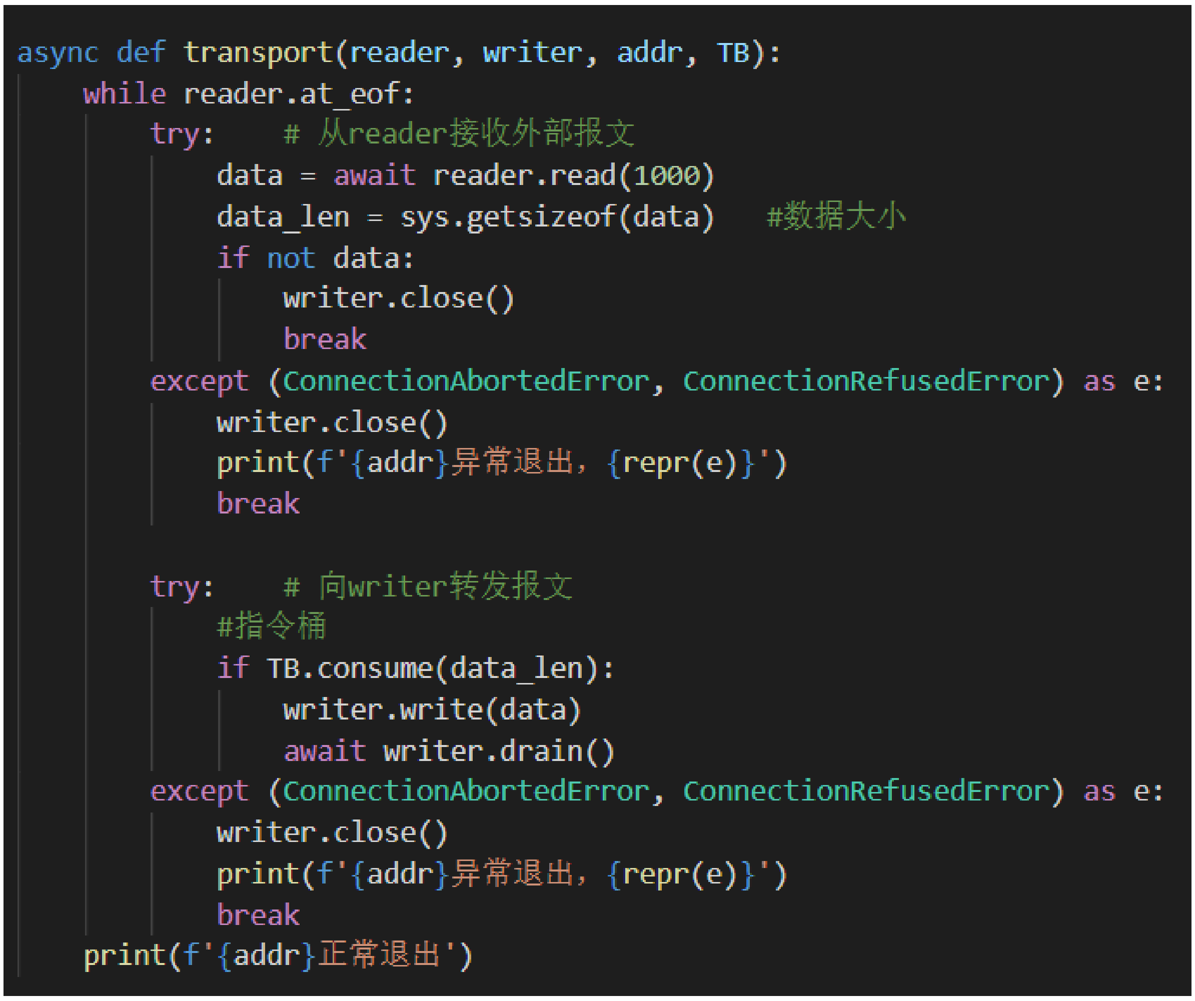
②不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多

多进程并发

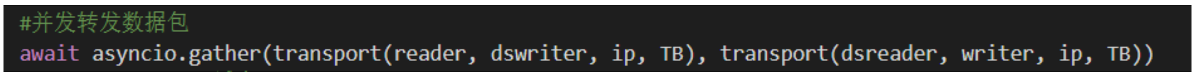
因为协程是一个线程执行，那怎么利用多核CPU呢？

我用的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

用asyncio实现并发，需要有多个协程来完成任务，每当有任务阻塞的时候就await



```python
async def transport(reader, writer, addr, TB):
    while reader.at_eof:
        try:    # 从reader接收外部报文
            data = await reader.read(1000)
            data_len = sys.getsizeof(data)   #数据大小
            if not data:
                writer.close()
                break
        except (ConnectionAbortedError, ConnectionRefusedError) as e:
            writer.close()
            print(f'{addr}异常退出, {repr(e)}')
            break

        try:    # 向writer转发报文
            #指令桶
            if TB.consume(data_len):
                writer.write(data)
                await writer.drain()
        except (ConnectionAbortedError, ConnectionRefusedError) as e:
            writer.close()
            print(f'{addr}异常退出, {repr(e)}')
            break
    print(f'{addr}正常退出')
```

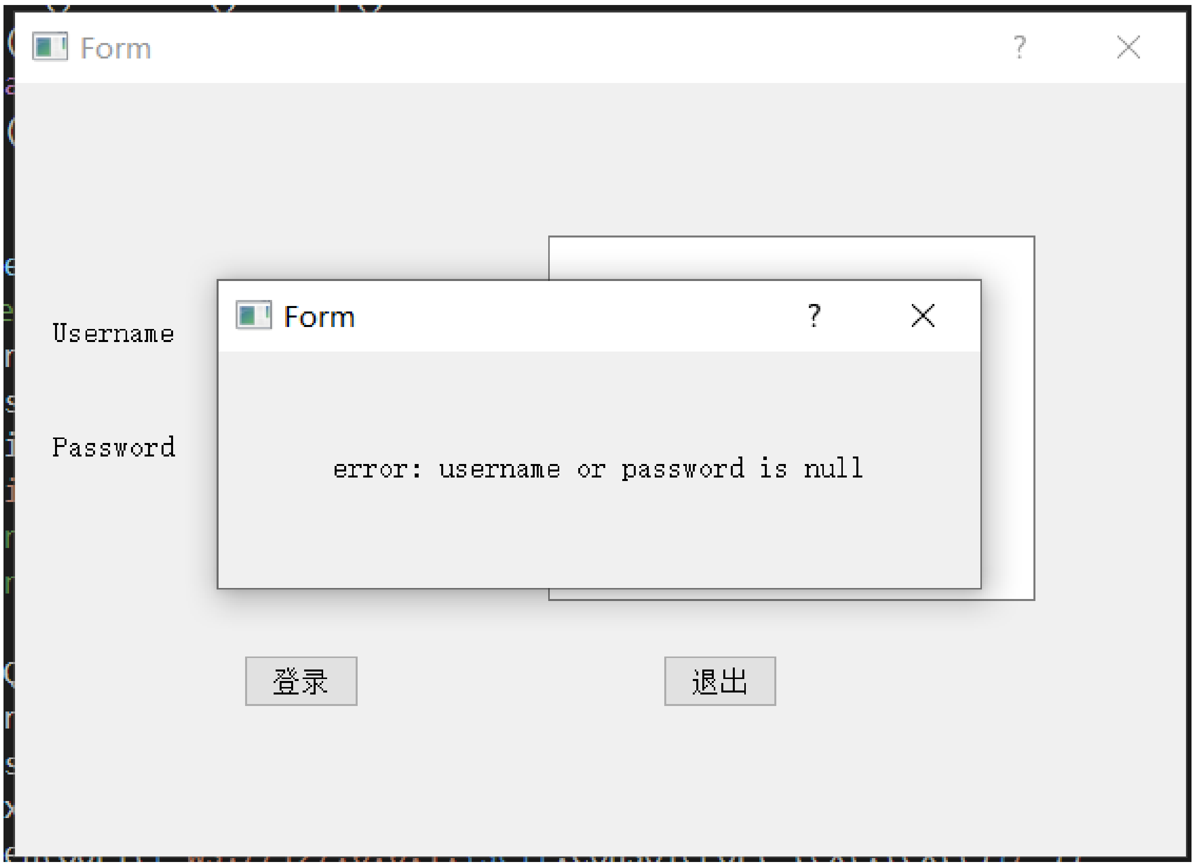


```python
#并发转发数据包
await asyncio.gather(transport(reader, dswriter, ip, TB), transport(dsreader, writer, ip, TB))
```

明显看出，这些进程是并发进行，直到全部的数据包都读写完毕（不止两个进程，因为客户端和服务端一直都在收发数据包）

健壮性

在界面上输入用户名和密码错误时，会弹出一个报错窗口。


Form

Username

Password

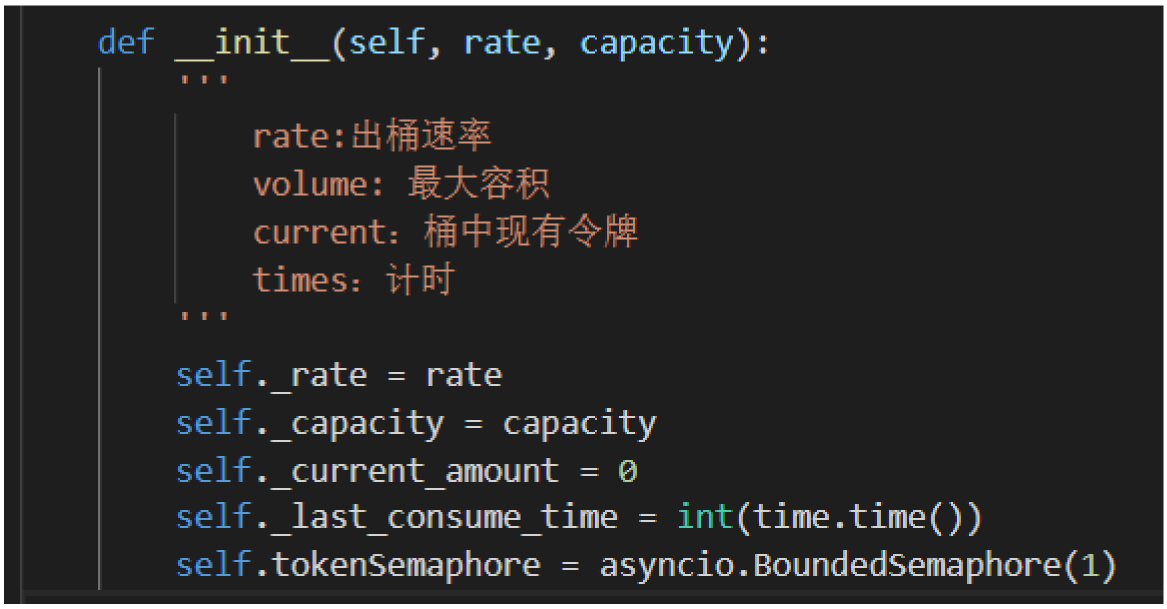
Form

error: username or password is null

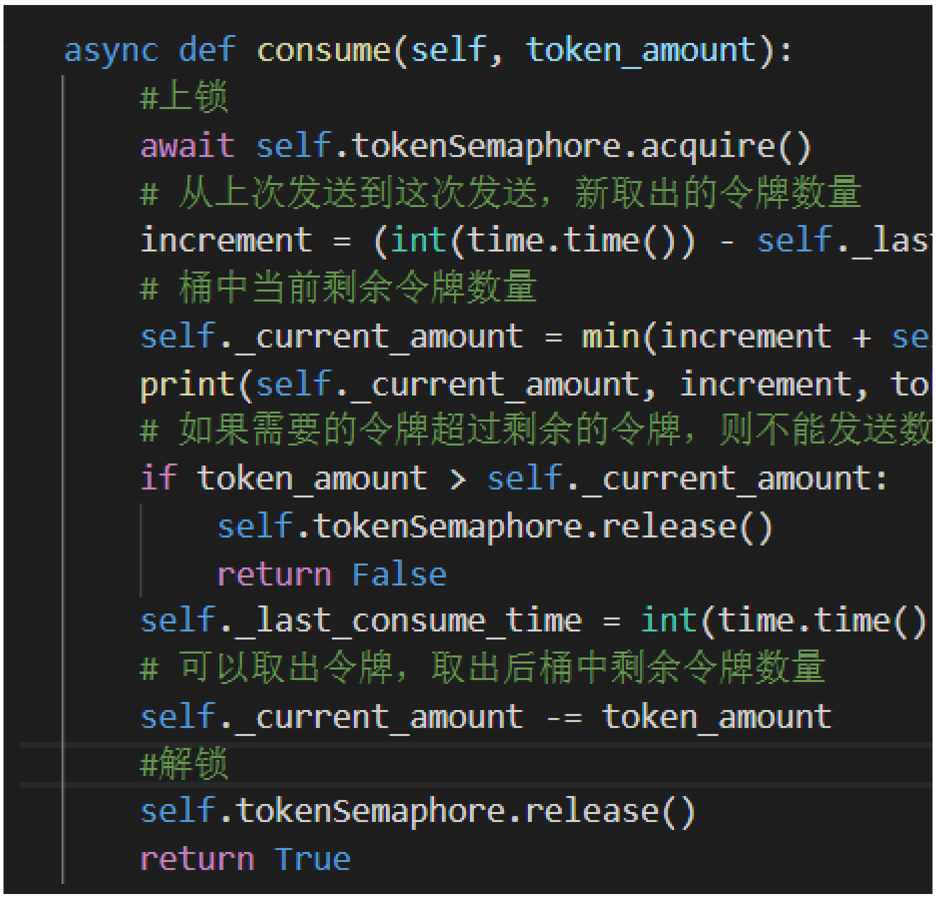
拓展性

在这个项目里，localProxy和remoteProxy使用的是多进程+协程并发。

互斥访问令牌桶



```python
def __init__(self, rate, capacity):
    '''
        rate:出桶速率
        volume: 最大容积
        current: 桶中现有令牌
        times: 计时
    '''
    self._rate = rate
    self._capacity = capacity
    self._current_amount = 0
    self._last_consume_time = int(time.time())
    self.tokenSemaphore = asyncio.BoundedSemaphore(1)
```



```
async def consume(self, token_amount):
    #上锁
    await self.tokenSemaphore.acquire()
    # 从上次发送到这次发送，新取出的令牌数量
    increment = (int(time.time()) - self._las
    # 桶中当前剩余令牌数量
    self._current_amount = min(increment + se
    print(self._current_amount, increment, to
    # 如果需要的令牌超过剩余的令牌，则不能发送数
    if token_amount > self._current_amount:
        self.tokenSemaphore.release()
        return False
    self._last_consume_time = int(time.time()
    # 可以取出令牌，取出后桶中剩余令牌数量
    self._current_amount -= token_amount
    #解锁
    self.tokenSemaphore.release()
    return True
```

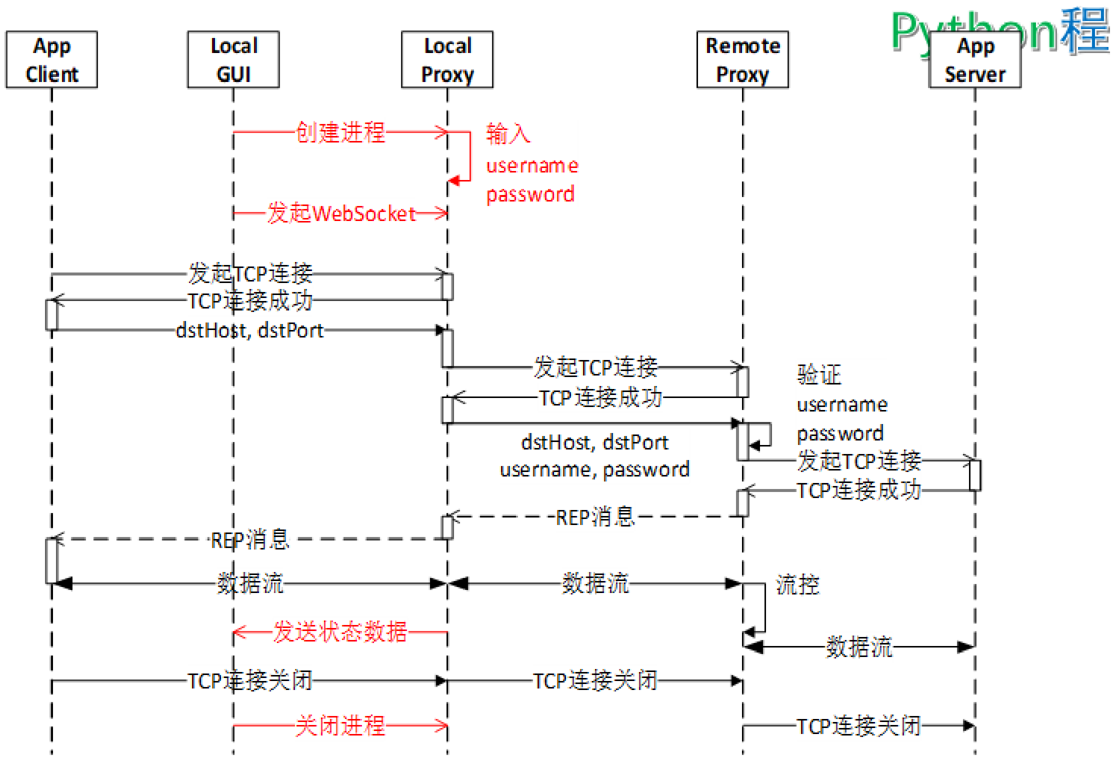
其中，tokenSemaphore是互斥锁，只允许一个进程访问令牌桶

await self.tokenSemaphore.acquire()　给令牌桶上锁

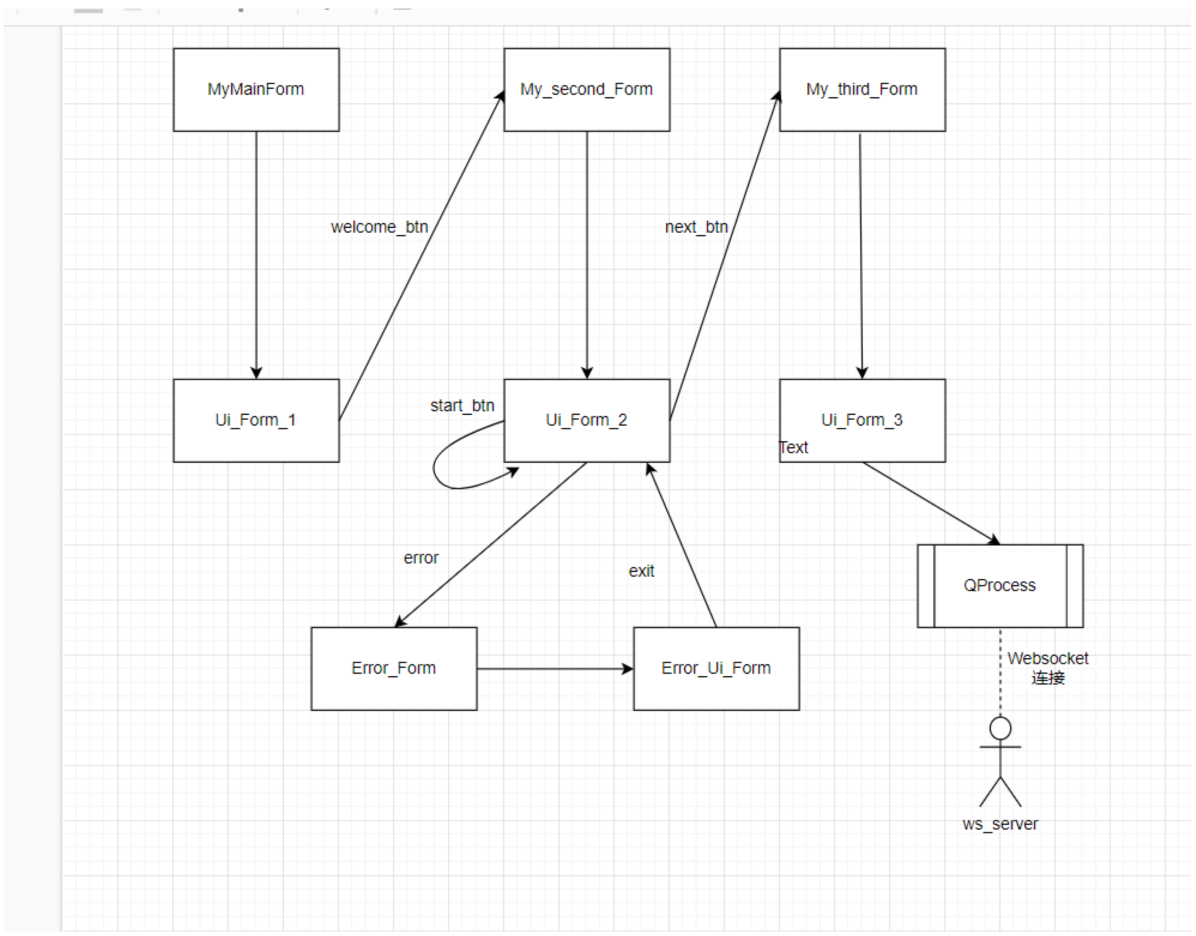
await self.tokenSemaphore.release()　给令牌桶解锁

4、模块图

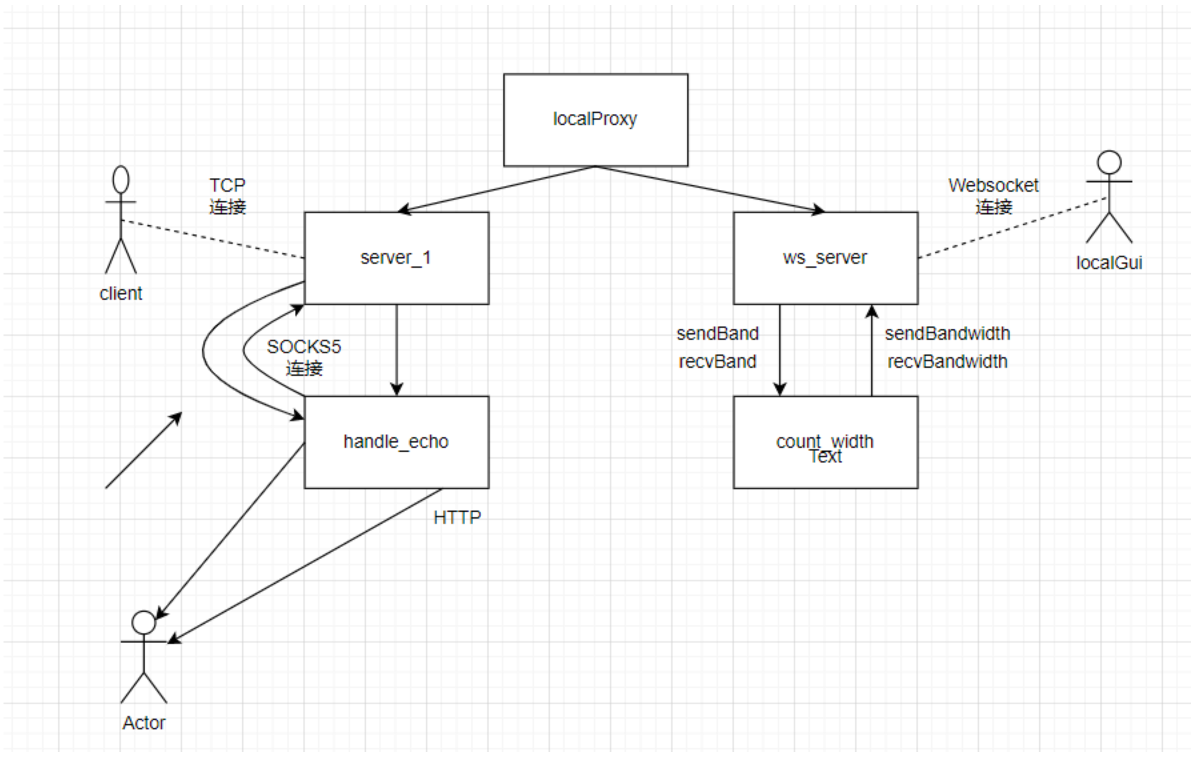
总览：



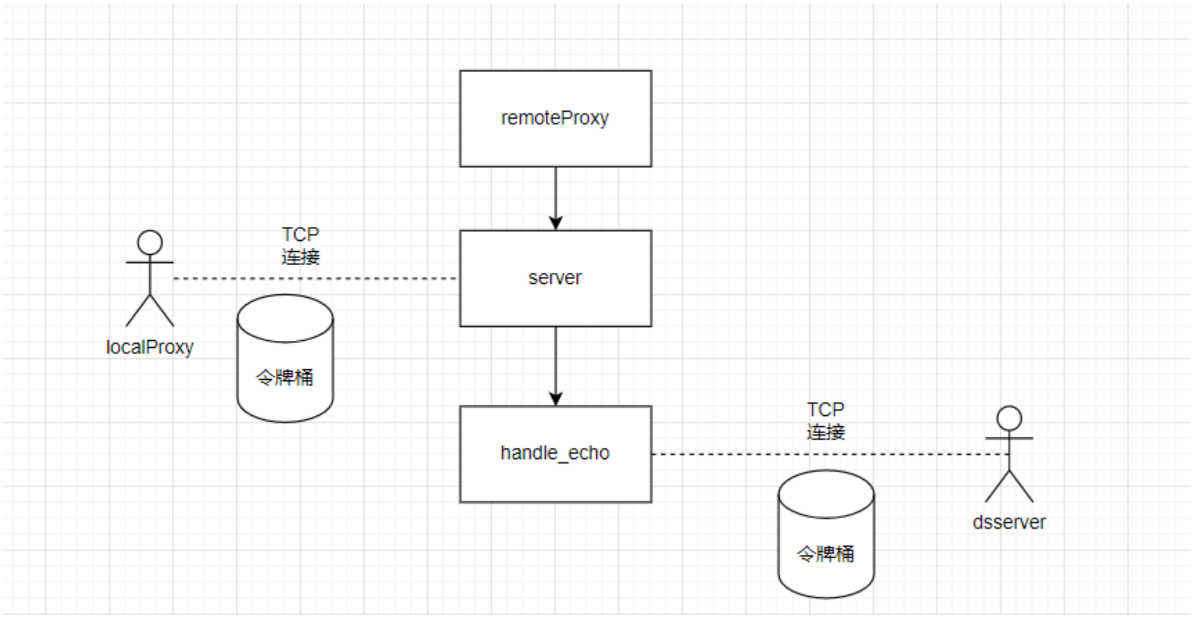
localGui:



localProxy



remoteProxy



5、具体实现过程：

localProxy

建立TCP连接

建立服务器并运行：（handle_echo是localProxy作为服务器处理数据调用的函数）

```
server_1 = await asyncio.start_server(handle_echo, '127.0.0.1', 1080)


addr = server_1.sockets[0].getsockname()
print(f'Serving on {addr}')

async with server_1:
    await server_1.serve_forever()
```

从客户端读取第一个信息，并判断连接的类型

```
data = await reader.read(5000) #将第一个信息读至EOF
data1 = data.decode()
httpdata = data1.split(' ')
# socks5协议
if data[0] == 5:
    pass
# http协议
#只处理http的connect包，其余包暂时不处理
elif httpdata[0] == 'CONNECT':
    pass
# 其他类型的连接不进行处理
else:
    print("we can't handle this type of request")
    writer.close()
    return
```

解析socks5的连接请求包，分为两种情况，ipv4和域名，还有一种情况是ipv6，没写出来

ipv4:

```
if header[0] == 5 and header[1] == 1:
    if header[3] == 1:  #IPv4 X'01'
        try:
            dsreader, dswriter = await asyncio.open_connection('127.0.0.5',
1005)   #向远端代理发起连接请求

            dswriter.write(usermas)
            await writer.drain()
            VER = await dsreader.read(50000)
            if VER[0] != 1:
                writer.close()

            dswriter.write(data)   #将数据包发给remoteproxy
            await dswriter.drain()
            REP = await dsreader.read(1024) #从远端代理处接收应答包
            writer.write(REP)   #将应答包转发给客户端
            await writer.drain()
            print(f'connect success with 127.0.0.5 and 1085 !')
        except (TimeoutError, ConnectionRefusedError) as e:
            print(f'connect failed with 127.0.0.5 and 1085 !')
            print(f'{repr(e)}')
            writer.close()
            return
        #并发转发数据包
        await asyncio.gather(transport(reader, dswriter, '127.0.0.1'),
transport(dsreader, writer, '127.0.0.1'), count_width())
```

域名：

```
if header[3] == 3: #域名 X'03'
        try:
            dsreader, dswriter = await asyncio.open_connection('127.0.0.5',
1085)
```

```
            dswriter.write(usermas)
            await writer.drain()
            VER = await dsreader.read(50000)
            if VER[0] != 1:
                writer.close()

            dswriter.write(data)   #将数据包发给remoteproxy
            await dswriter.drain()
            REP = await dsreader.read(1024) #从远端代理处接收应答包
            writer.write(REP)   #将应答包转发给客户端
            await writer.drain()

            print(f'connect success with 127.0.0.5 and 1085 in SOCKS5!')
        except (TimeoutError, ConnectionRefusedError) as e:
            print(f'connect failed with 127.0.0.5 and 1085 in SOCKS5!')
            print(f'{repr(e)}')
            writer.close()
            return
        #并发转发数据包
        await asyncio.gather(transport(reader, dswriter, '127.0.0.5'),
transport(dsreader, writer, '127.0.0.5'), count_width())
```

解析http协议：

```
# http协议
elif httpdata[0] == 'CONNECT': #只处理http的connect包，其余包暂时不处理
    try:
        dsreader, dswriter = await asyncio.open_connection('127.0.0.5', 1085)  #
与remoteproxy建立连接

        #账号、密码认证
        dswriter.write(usermas)
        await dswriter.drain()

        VER = await dsreader.read(50000)
        if VER[0] != 1:
            writer.close()

        dswriter.write(data)   #将数据包发给remoteproxy
        await dswriter.drain()
        REP = await dsreader.read(1024) #从远端代理处接收应答包
        writer.write(REP)    #将应答包转发给客户端
        await writer.drain()
        print(f'connect success with 127.0.0.5 and 1085 in HTTP!')
    except (TimeoutError, ConnectionRefusedError) as e:
        print(f'connect failed with 127.0.0.1 and 1085 in HTTP!')
        print(f'{repr(e)}')
        writer.close()

        return
    await asyncio.gather(transport(reader, dswriter, '127.0.0.5'),
transport(dsreader, writer, '127.0.0.5'), count_width())
```

实现并发转发数据包

```
async def transport(reader, writer, addr):
    global SendBand
    global RecvBand
    while reader.at_eof:
        try:    # 从reader接收外部报文
            data = await reader.read(1000)
            RecvBand += len(data)
            if not data:
                writer.close()
                break
        except (ConnectionAbortedError, ConnectionRefusedError) as e:
            writer.close()
            print(f'{addr}异常退出，{repr(e)}')
            break
        try:    # 向writer转发报文
            SendBand += len(data)
            writer.write(data)
            await writer.drain()
        except (ConnectionAbortedError, ConnectionRefusedError) as e:
            writer.close()
            print(f'{addr}异常退出，{repr(e)}')
            break
    print(f'{addr}正常退出')

# 实现并发转发数据包
await asyncio.gather(transport(reader, dswriter, '127.0.0.5'),
transport(dsreader, writer, '127.0.0.5'), count_width())
```

建立Websocket连接

建立服务器

```
if args.consolePort:
        ws_server = await websockets.serve(localConsole, '127.0.0.1',
args.consolePort)
        logging.info(f'CONSOLE LISTEN {ws_server.sockets[0].getsockname()}')

    #asyncio.create_task(calcBandwidth())

    server_1 = await asyncio.start_server(handle_echo, '127.0.0.1', 1080)
```

测试运行状态 (计算宽带)

```
async def count_width():
    global SendBand
    global RecvBand
    global SendBandwidth
    global RecvBandwidth
    time.sleep(1)
    SendBandwidth = SendBand/1
    SendBand = 0
    print('SendBandwidth:', SendBandwidth)
    RecvBandwidth = RecvBand/1
    RecvBand = 0
    print('RecvBandwidth:', RecvBandwidth)
```

回传运行状态给localGui

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

remoteProxy

建立数据库

```
async with aiosqlite.connect('user.db') as db:
    #await db.execute("DROP TABLE user_table") #删除表
    await db.execute("CREATE TABLE if not exists user_table ( username
VARCHAR(50) primary key, password VARCHAR(50), rate INT)")
    #await db.execute("DELETE FROM user_table")#删除表中的数据
    #await db.execute("INSERT INTO user_table ( username, password) VALUES
('R52125', 'R100214')")
    #await db.execute("INSERT INTO user_table ( username, password) VALUES
('F444627549', 'XJBRCBR')")

    await db.execute("DELETE FROM user_table where username = 1")
    await db.execute("INSERT INTO user_table (username, password, rate) VALUES
(?, ?, ?)" ,(args.username, args.password, args.rate))
    await db.execute("DELETE FROM user_table where username = 1")
    await db.commit()

    async with db.execute("SELECT username, password, rate FROM user_table") as
uap:
        async for u in uap:
            print(f'f1={u[0]} f2={u[1]} f3={u[2]}')
```

身份验证

```
async with aiosqlite.connect('user.db') as db:
    async with db.execute("SELECT username, password, rate FROM user_table") as 
uap:
        async for u in uap:
            if usermessage[0]==u[0] and usermessage[1]==u[1] and flag==0:
                Rate = u[2] #速率
                flag = 1
                print("认证成功!!!")
                writer.write(b'\x01')
                await writer.drain()

if flag==0:
    writer.close()
    print("认证失败!!!")
```

建立TCP连接

建立服务器并运行：（handle_echo为）

```
server = await asyncio.start_server(handle_echo, '127.0.0.5', 1085)

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

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

处理localProxy和目的服务器发来的数据包

socks5连接：ipv4和域名，ipv6没有实现

ipv4

```
if header[3] == 1:  #IPv4 X'01'
    ip = '.'.join([str(i) for i in unpack('!BBBB', data[4:8])]) #获取IP地址，IPv4的
地址4个字节
    port = unpack('!H', data[8:10])[0]  #获取端口号，2字节
    print(f'ip:{ip}, port{port}')
    try:
        dsreader, dswriter = await asyncio.open_connection(ip, port)
        writer.write(b'\x05\x00\x00' + data[3:11])  #将信息重新打包发回给
localproxy，告知客户端代理服务器与目标服务器连接成功
        await writer.drain()
        print(f'connect success with {ip} and {port}!')
    except (TimeoutError, ConnectionRefusedError) as e:
        print(f'connect failed with{ip} and {port}!')
        print(f'{repr(e)}')
        writer.close()
        return
    #指令桶实例化
    TB = TokenBucket(Rate, capacity)
    #并发转发数据包
    await asyncio.gather(transport(reader, dswriter, ip, TB), 
transport(dsreader, writer, ip, TB))
```

域名

```
if header[3] == 3: #域名 X'03'
    Hostlen = unpack('!B', data[4:5])[0] #域名长度
    Host = data[5: Hostlen+5].decode('utf-8') #解析域名
    port = unpack('!H', data[5+Hostlen: Hostlen+7])[0] #获取端口号，2字节
    print(f'Hostlen:{Hostlen}, Host:{Host}, port:{port}')
    try:
        dsreader, dswriter = await asyncio.open_connection(Host, port)
        writer.write(b'\x05\x00\x00'+data[3: Hostlen+7])  #将信息重新打包发回给
locoalproxy，告知客户端：代理服务器与目标服务器连接成功
        await writer.drain()
        print(f'connect success with {Host} and {port} in SOCKS5!')
    except (TimeoutError, ConnectionRefusedError) as e:
        print(f'connect failed with{Host} and {port} in SOCKS5!')
        print(f'{repr(e)}')
        writer.close()
        return

    #指令桶
    TB = TokenBucket(Rate, capacity)
    #并发转发数据包
    await asyncio.gather(transport(reader, dswriter, Host, TB),
transport(dsreader, writer, Host, TB))
```

http连接

```
# http协议
elif httpdata[0] == 'CONNECT': #只处理http的connect包，其余包暂时不处理
    try:
        httpdata1 = httpdata[1].split(':')
        Host = httpdata1[0] # 获取主机地址
        port = httpdata1[1] # 获取端口号
        dsreader, dswriter = await asyncio.open_connection(Host, port)  # 与目的服
务器建立连接
        writer.write(b'HTTP/1.1 200 Connection established\r\n\r\n')    # 将应答包
发给localproxy，告知客户端：代理服务器与目标服务器连接成功
        await writer.drain()
        print(f'connect success with {Host} and {port} in HTTP!')
    except (TimeoutError, ConnectionRefusedError) as e:
        print(f'connect failed with {Host} and {port} in HTTP!')
        print(f'{repr(e)}')
        writer.close()
        return

    #指令桶实例化
    TB = TokenBucket(Rate, capacity)
    #并发转发数据包
    await asyncio.gather(transport(reader, dswriter, Host, TB),
transport(dsreader, writer, Host, TB))
```

实现并发转发数据包，并且利用指令桶进行限流

```
async def transport(reader, writer, addr, TB):
    while reader.at_eof:
        try:    # 从reader接收外部报文
            data = await reader.read(1000)
            data_len = sys.getsizeof(data)   #数据大小
```

```
                if not data:
                    writer.close()
                    break
            except (ConnectionAbortedError, ConnectionRefusedError) as e:
                writer.close()
                print(f'{addr}异常退出，{repr(e)}')
                break

            try:    # 向writer转发报文
                #指令桶
                if TB.consume(data_len):
                    writer.write(data)
                    await writer.drain()
            except (ConnectionAbortedError, ConnectionRefusedError) as e:
                writer.close()
                print(f'{addr}异常退出，{repr(e)}')
                break
        print(f'{addr}正常退出')

# 实现并发
await asyncio.gather(transport(reader, dswriter, Host, TB), transport(dsreader,
writer, Host, TB))
```

令牌桶实现

```
capacity = 10000000 #桶容量
class TokenBucket:
    '''
        令牌桶算法：出
            令牌出桶速率恒定，当桶中无剩余令牌，则不能取出
    '''
    def __init__(self, rate, capacity):
        '''
            rate:出桶速率
            volume: 最大容积
            current：桶中现有令牌
            times：计时
        '''
        self._rate = rate
        self._capacity = capacity
        self._current_amount = 0
        self._last_consume_time = int(time.time())
        self.tokenSemaphore = asyncio.BoundedSemaphore(1)

    async def consume(self, token_amount):
        await self.tokenSemaphore.acquire()#上锁
        # 从上次发送到这次发送，新取出的令牌数量
        increment = (int(time.time()) - self._last_consume_time) * self._rate
        # 桶中当前剩余令牌数量
        self._current_amount = min(increment + self._current_amount,
self._capacity)
        print(self._current_amount, increment, token_amount,
self._last_consume_time, int(time.time()))
        # 如果需要的令牌超过剩余的令牌，则不能发送数据
        if token_amount > self._current_amount:
            self.tokenSemaphore.release()
            return False
        self._last_consume_time = int(time.time())
        # 可以取出令牌，取出后桶中剩余令牌数量
        self._current_amount -= token_amount
        self.tokenSemaphore.release()
        return True
```

localGui

主窗口

```
# 主窗口
class Ui_Form_1(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(484, 390)
        self.Go = QtWidgets.QPushButton(Form)
        self.Go.setGeometry(QtCore.QRect(190, 170, 93, 28))
        self.Go.setObjectName("Go")
        self.Go.clicked.connect(self.enter)

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.Go.setText(_translate("Form", "welcome"))
```

一级窗口

```
# 一级界面
class Ui_Form_2(object):
    def setupUi(self, Form):
        #global gl_username
        #global gl_password
        Form.setObjectName("Form")
        Form.resize(615, 406)
        self.username_lable = QtWidgets.QLabel(Form)
        self.username_lable.setGeometry(QtCore.QRect(20, 120, 101, 21))
        self.username_lable.setObjectName("username_lable")
        self.input_username = QtWidgets.QLineEdit(Form)
        self.input_username.setGeometry(QtCore.QRect(110, 120, 113, 21))
        self.input_username.setText("")
        self.password_lable = QtWidgets.QLabel(Form)
        self.password_lable.setGeometry(QtCore.QRect(20, 180, 101, 20))
        self.password_lable.setObjectName("password_lable")
        self.input_password = QtWidgets.QLineEdit(Form)
        self.input_password.setGeometry(QtCore.QRect(110, 180, 113, 21))
        self.input_password.setObjectName("input_password")
        self.input_password.setEchoMode(QLineEdit.Password)
        self.show_text = QtWidgets.QTextBrowser(Form)
        self.show_text.setGeometry(QtCore.QRect(280, 80, 256, 192))
        self.show_text.setObjectName("show_text")
        self.go_btn = QtWidgets.QPushButton(Form)
        self.go_btn.setGeometry(QtCore.QRect(120, 300, 61, 28))
        self.go_btn.setObjectName("go_btn")
        self.exit_btn = QtWidgets.QPushButton(Form)
        self.exit_btn.setGeometry(QtCore.QRect(340, 300, 61, 28))
        self.exit_btn.setObjectName("exit_btn")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.username_lable.setText(_translate("Form", "Username"))
        self.password_lable.setText(_translate("Form", "Password"))
        self.go_btn.setText(_translate("Form", "登录"))
        self.exit_btn.setText(_translate("Form", "退出"))
```

二级窗口

```
# 二级界面
class Ui_Form_3(Ui_Form_2):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(762, 529)
        self.scrollArea = QtWidgets.QScrollArea(Form)
        self.scrollArea.setGeometry(QtCore.QRect(99, 76, 561, 341))
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 559, 339))
        self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
        self.tableWidget = QtWidgets.QTableWidget(self.scrollAreaWidgetContents)
        self.tableWidget.setGeometry(QtCore.QRect(0, 0, 561, 341))
        self.tableWidget.setColumnCount(4)
        self.tableWidget.setObjectName("tableWidget")
        self.tableWidget.setRowCount(0)
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(3, item)
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        self.Start_btn = QtWidgets.QPushButton(Form)
        self.Start_btn.setGeometry(QtCore.QRect(160, 430, 93, 28))
        self.Start_btn.setObjectName("Start_btn")
        self.Stop_btn = QtWidgets.QPushButton(Form)
        self.Stop_btn.setGeometry(QtCore.QRect(480, 430, 93, 28))
        self.Stop_btn.setObjectName("Stop_btn")
        self.Exit_btn = QtWidgets.QPushButton(Form)
        self.Exit_btn.setGeometry(QtCore.QRect(310, 480, 93, 28))
        self.Exit_btn.setObjectName("Exit_btn")
        self.Host_label = QtWidgets.QLabel(Form)
        self.Host_label.setGeometry(QtCore.QRect(410, 20, 71, 21))
        self.Host_label.setObjectName("Host_label")
        self.ConsolePort_lable = QtWidgets.QLabel(Form)
        self.ConsolePort_lable.setGeometry(QtCore.QRect(410, 50, 91, 21))
        self.ConsolePort_lable.setObjectName("ConsolePort_lable")
        self.Host_text = QtWidgets.QLineEdit(Form)
        self.Host_text.setGeometry(QtCore.QRect(500, 20, 141, 21))
        self.Host_text.setText("")
        self.Host_text.setObjectName("Host_text")
        self.ConsolePort_text = QtWidgets.QLineEdit(Form)
        self.ConsolePort_text.setGeometry(QtCore.QRect(500, 50, 131, 21))
        self.ConsolePort_text.setObjectName("ConsolePort_text")
        self.username_lable = QtWidgets.QLabel(Form)
        self.username_lable.setGeometry(QtCore.QRect(100, 20, 72, 21))
        self.username_lable.setObjectName("username_lable")
        self.label = QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(100, 50, 72, 21))
        self.label.setObjectName("label")
        self.username_text = QtWidgets.QLineEdit(Form)
        self.username_text.setGeometry(QtCore.QRect(170, 20, 151, 21))
        self.username_text.setObjectName("username_text")
        self.password_text = QtWidgets.QLineEdit(Form)
        self.password_text.setGeometry(QtCore.QRect(170, 50, 151, 21))
        self.password_text.setObjectName("password_text")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        item = self.tableWidget.horizontalHeaderItem(0)
        item.setText(_translate("Form", "Time"))
        item = self.tableWidget.horizontalHeaderItem(1)
        item.setText(_translate("Form", "Connect or not"))
        item = self.tableWidget.horizontalHeaderItem(2)
        item.setText(_translate("Form", "SendBandWidth"))
        item = self.tableWidget.horizontalHeaderItem(3)
        item.setText(_translate("Form", "RecvBandWidth"))
        self.Start_btn.setText(_translate("Form", "Start"))
        self.Stop_btn.setText(_translate("Form", "Stop"))
        self.Exit_btn.setText(_translate("Form", "Exit"))
        self.Host_label.setText(_translate("Form", "Host"))
        self.ConsolePort_lable.setText(_translate("Form", "ConsolePort"))
        self.username_lable.setText(_translate("Form", "username"))
        self.label.setText(_translate("Form", "password"))
```

错误界面

```
# 错误界面
class Error_Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(400, 124)
        self.error_lable = QtWidgets.QLabel(Form)
        self.error_lable.setGeometry(QtCore.QRect(60, 40, 291, 41))
        self.error_lable.setFrameShadow(QtWidgets.QFrame.Plain)
        self.error_lable.setObjectName("error_lable")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.error_lable.setText(_translate("Form", "error: username or password
is null"))
```

界面与业务逻辑分离实现

```
class MyMainForm(QMainWindow, Ui_Form_1):
    def __init__(self, parent=None):
        super(MyMainForm, self).__init__(parent)
        self.setupUi(self)

class My_second_Form(QDialog, Ui_Form_2):
    def __init__(self, parent=None):
        super(My_second_Form, self).__init__(parent)
        self.setupUi(self)

class My_third_Form(QDialog, Ui_Form_3):
    def __init__(self, parent=None):
        super(My_third_Form, self).__init__(parent)
        self.setupUi(self)

class Error_Form(QDialog, Error_Ui_Form):
    def __init__(self, parent=None):
        super(Error_Form, self).__init__(parent)
        self.setupUi(self)
```

添加信号和槽，实现业务逻辑

```
self.Go.clicked.connect(self.enter)

def enter(self):
    self.hide()
    self.s = My_second_Form()
    self.s.show()

-------------------------------------------------------------------------------

self.go_btn.clicked.connect(self.show_msg)
self.exit_btn.clicked.connect(self.Hide)

# 显示欢迎信息
def show_msg(self):
    self.username = self.input_username.text()
    self.password = self.input_password.text()
    if self.username!='' and self.password!='':
        self.show_text.setText(f'Welcome, {self.username}')
        self.show_text.show()

        self.go_btn.setText('next')
```

```
            self.go_btn.clicked.connect(self.monitor)
            # 显示错误界面
            else:
                self.e = Error_Form()
                self.e.show()

    # 显示监控（图形化）
    def monitor(self):
        self.hide()
        self.m = My_third_Form()
        self.m.show()

    # 退出
    def Hide(self):
        self.hide()
        self.h = MyMainForm()
        self.h.show()

    -------------------------------------------------------------------------------

    self.Stop_btn.clicked.connect(self.Stop)
    self.Start_btn.clicked.connect(self.Start)
    self.Exit_btn.clicked.connect(self.Hide)

    # 开始监控
    def Start(self):
        self.host = self.Host_text.text()
        self.consolePort = self.ConsolePort_text.text()
        msg = Ui_Form_2()
        self.username = self.username_text.text()
        self.password = self.password_text.text()
        pythonExec = os.path.basename(sys.executable)

        cmdLine = f'{pythonExec} localproxy.py -u {self.username} -p {self.password}
    -c {self.consolePort}'
        logging.debug(f'cmd={cmdLine}')
        self.process.start(cmdLine)

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

    def process_started(self):
        # 等同于self.process，使用sender适应性更好
        process = self.sender()
        processId = process.processId()
        logging.basicConfig(filename='example.log', level=logging.DEBUG)
        logging.debug(f'pid={processId}')
        #self.processIdLine = QLineEdit()
        #self.processIdLine.setText(str(processId))

        self.websocket = QWebSocket()
```

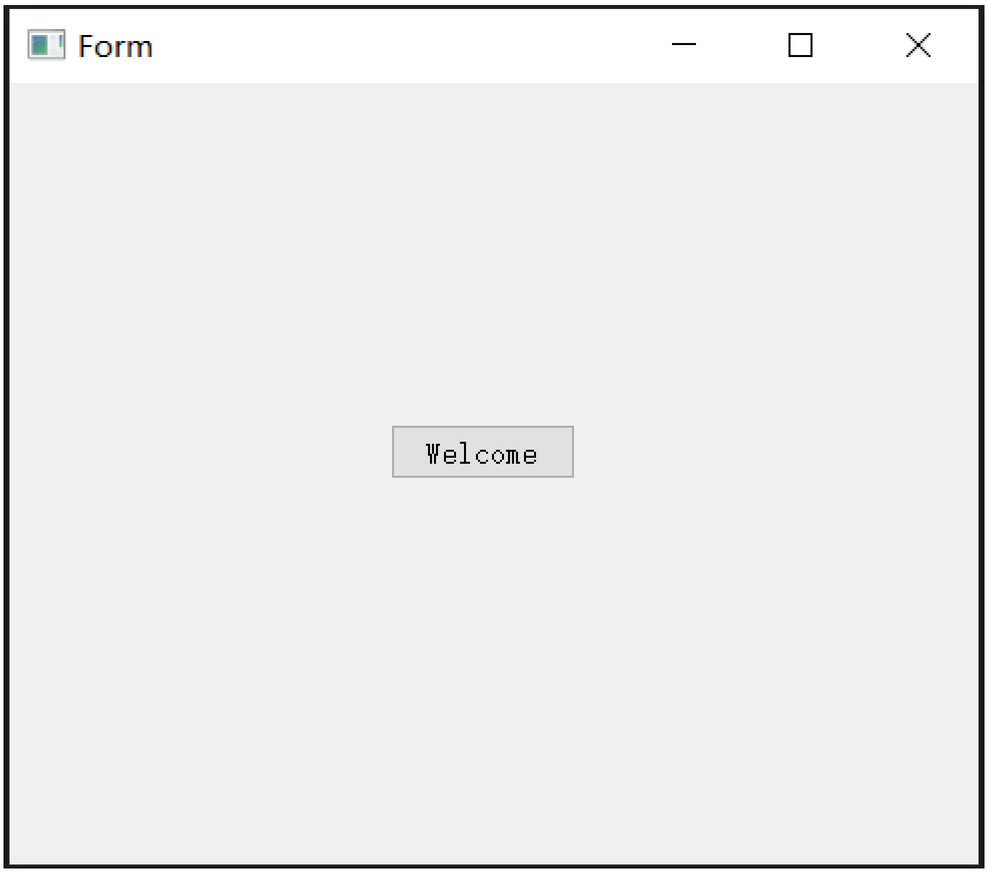
```python
        self.websocket.connected.connect(self.websocket_connected)
        self.websocket.disconnected.connect(self.websocket_disconnected)
        self.websocket.textMessageReceived.connect(self.websocket_message_rec)
        self.websocket.open(QUrl(f'ws://127.0.0.1:{self.ConsolePort_text.text()}/'))

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

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

    def websocket_message_rec(self, msg):
        logging.debug(f'msg={msg}')
        send_Bandwidth, recv_Bandwidth, *_ = msg.split()
        self.nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
        self.sendmsg_input = f'{humanfriendly.format_size(int(send_Bandwidth))}'
        self.recvmsg_input = f'{humanfriendly.format_size(int(recv_Bandwidth))}'
        row = self.tableWidget.rowCount()   # 返回当前行数
        self.tableWidget.insertRow(row)     # 尾部插入一行新行表格
        col = self.tableWidget.columnCount()# 返回当前列数
        self.tableWidget.setItem(row, 0, QTableWidgetItem(self.nowTime))
        self.tableWidget.setItem(row, 1, QTableWidgetItem('connect'))
        self.tableWidget.setItem(row, 2, QTableWidgetItem(self.sendmsg_input))
        self.tableWidget.setItem(row, 3, QTableWidgetItem(self.recvmsg_input))

    # 进程停止
    def Stop(self):
        self.process.kill()

    # 退出
    def Hide(self):
        self.hide()
        self.h = My_second_Form()
        self.h.show()
```

QProcess类（管理localProxy）

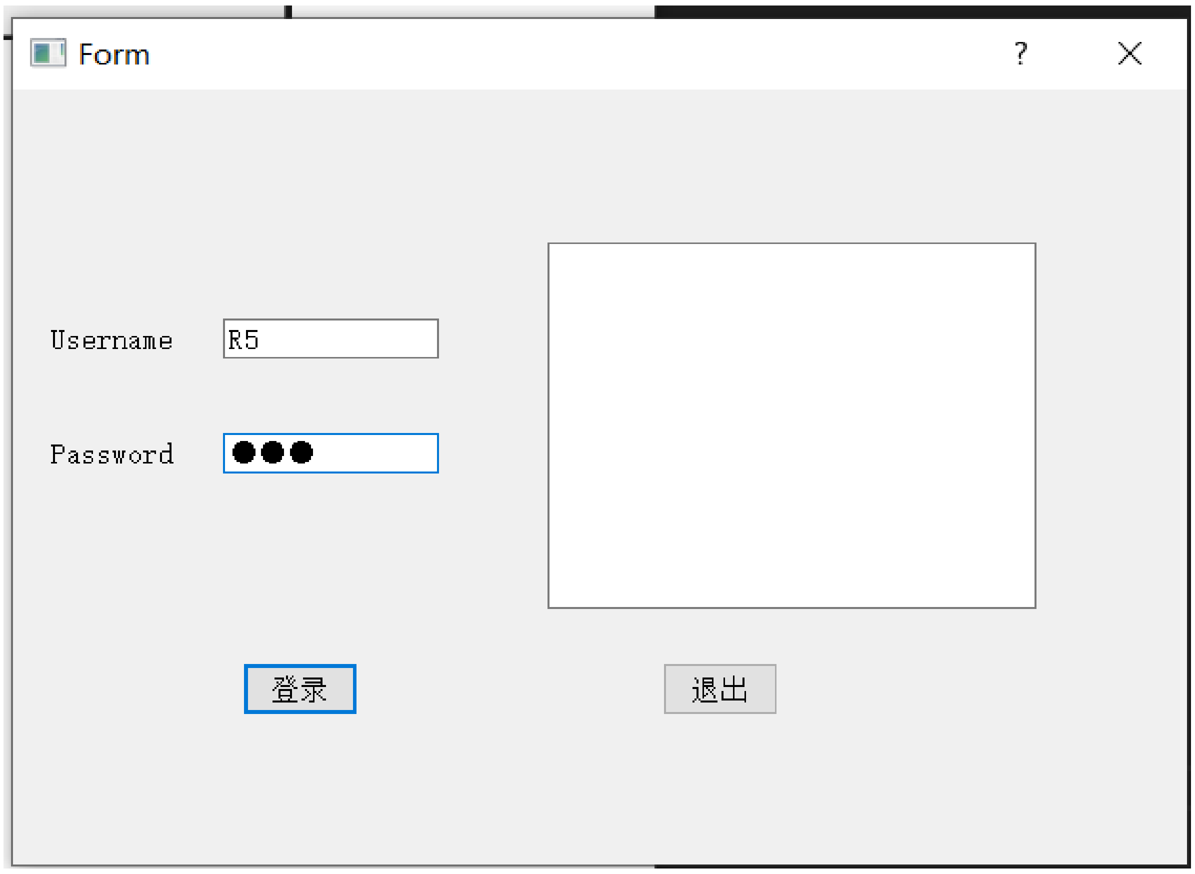
```python
# -----使用QProcess类管理localProxy-----
self.process = QProcess()
self.process.setProcessChannelMode(QProcess.MergedChannels)
#self.process.finished.connect(self.process_finished)
self.process.started.connect(self.process_started)
self.process.readyReadStandardOutput.connect(self.process_readyread)
```

6、测试样例

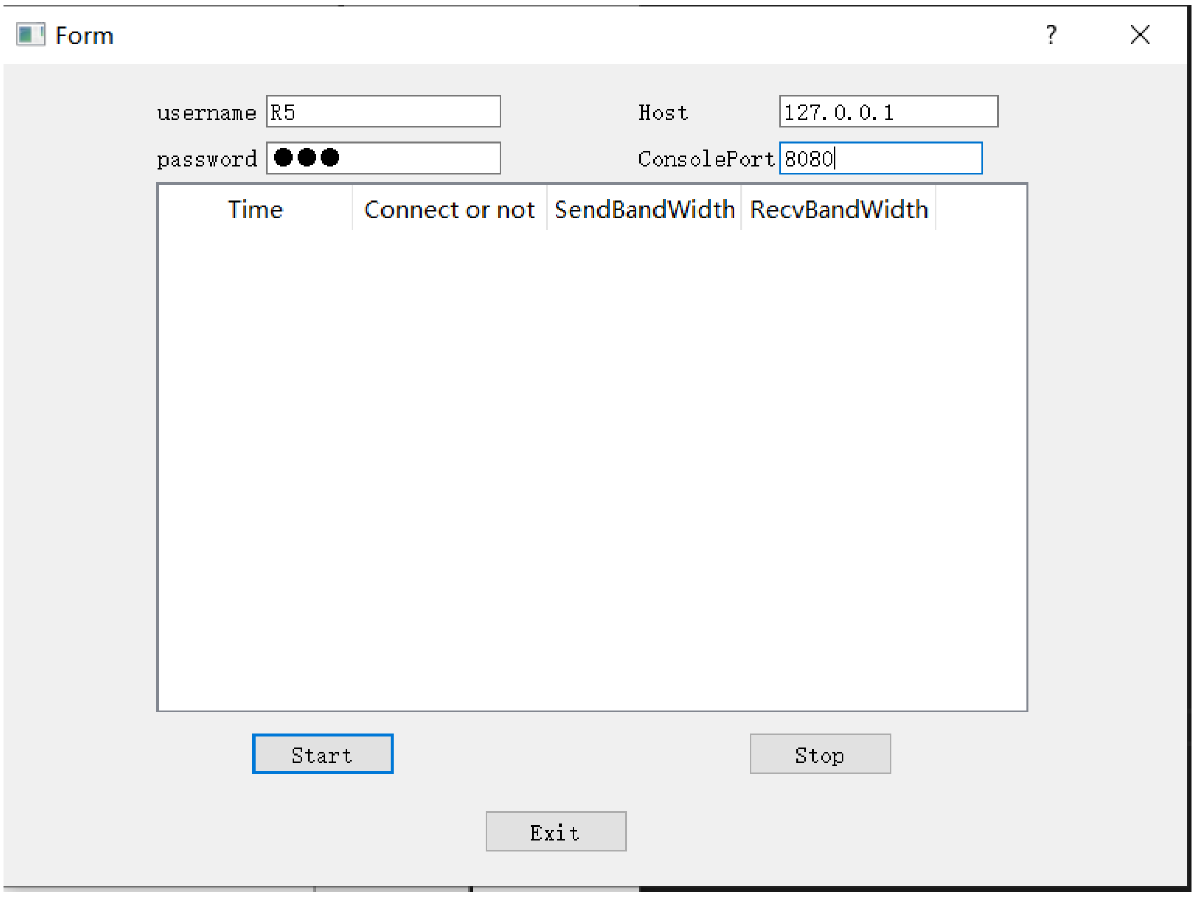
主窗口



一级窗口

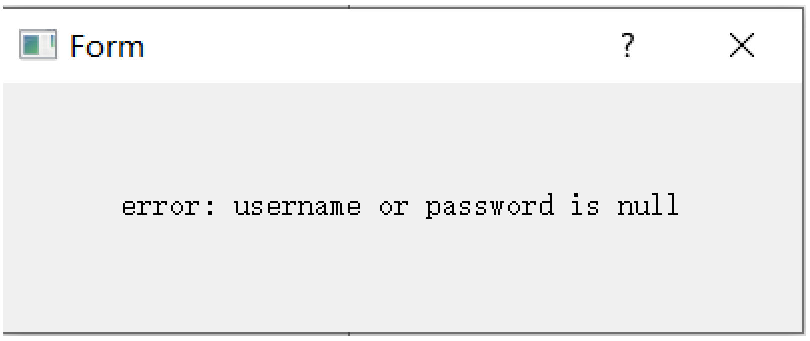


二级窗口



username	R5	Host	127.0.0.1
password	●●●	ConsolePort	8080

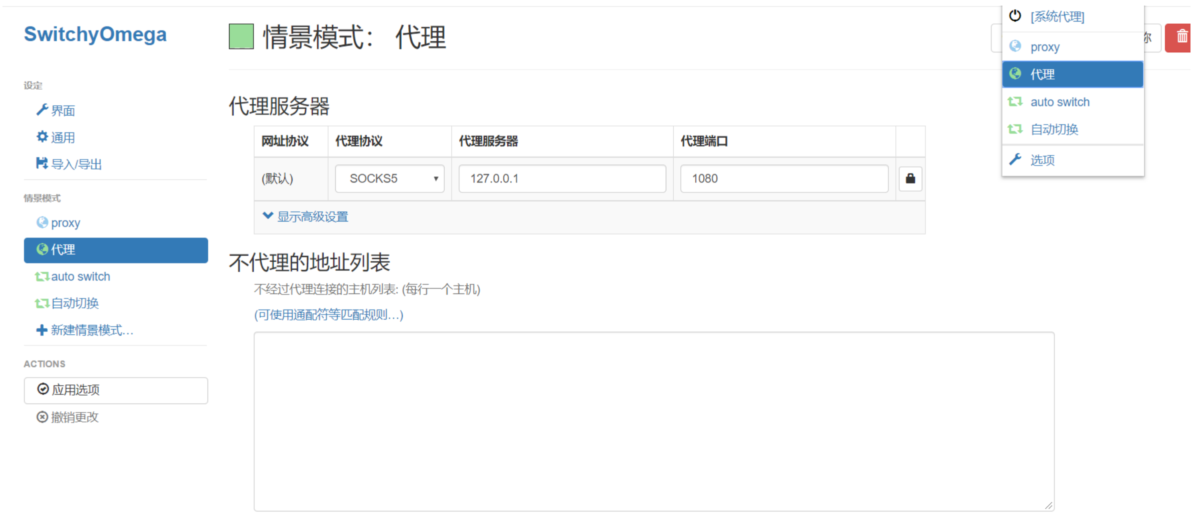
错误窗口



```
error: username or password is null
```

代理设置

SOCKS5代理设置



情景模式：代理

代理服务器

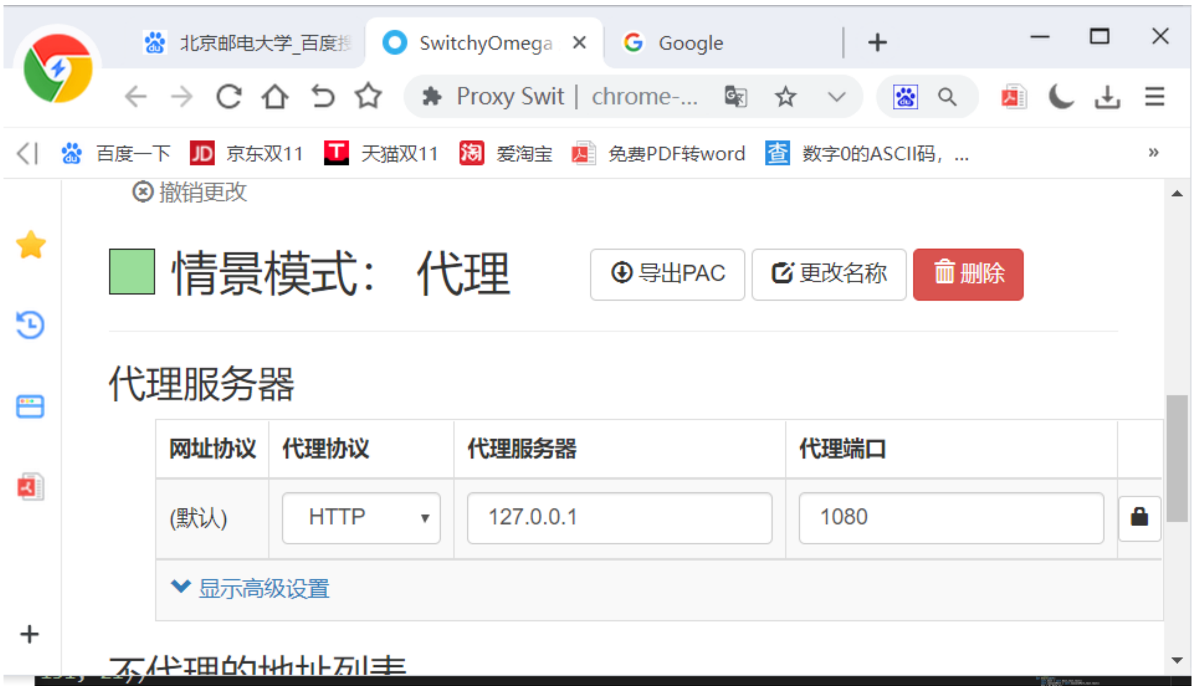
网址协议	代理协议	代理服务器	代理端口
(默认)	SOCKS5	127.0.0.1	1080

不代理的地址列表

不经过代理连接的主机列表: (每行一个主机)

(可使用通配符等匹配规则...)

HTTP代理设置



撤销更改

情景模式：代理

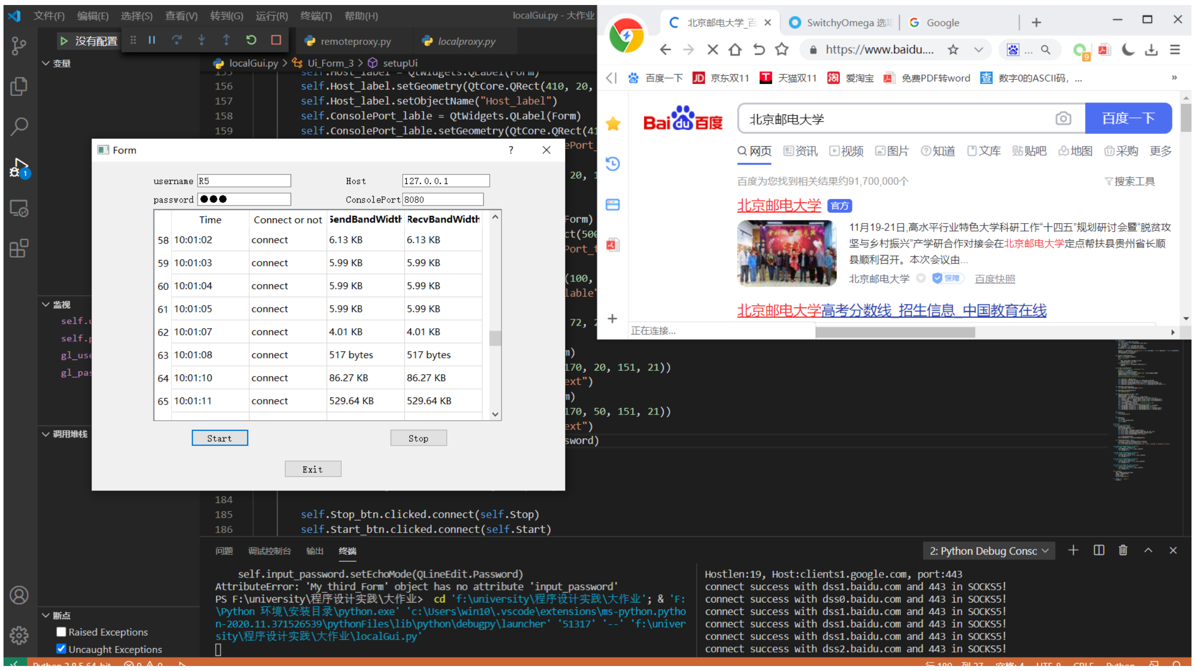
代理服务器

网址协议	代理协议	代理服务器	代理端口
(默认)	HTTP	127.0.0.1	1080

不代理的地址列表

代理服务器运行结果（是否能上网）及localProxy的状态

SOCKS5成功联网



username R5　　Host 127.0.0.1
password ●●●　　ConsolePort 8080

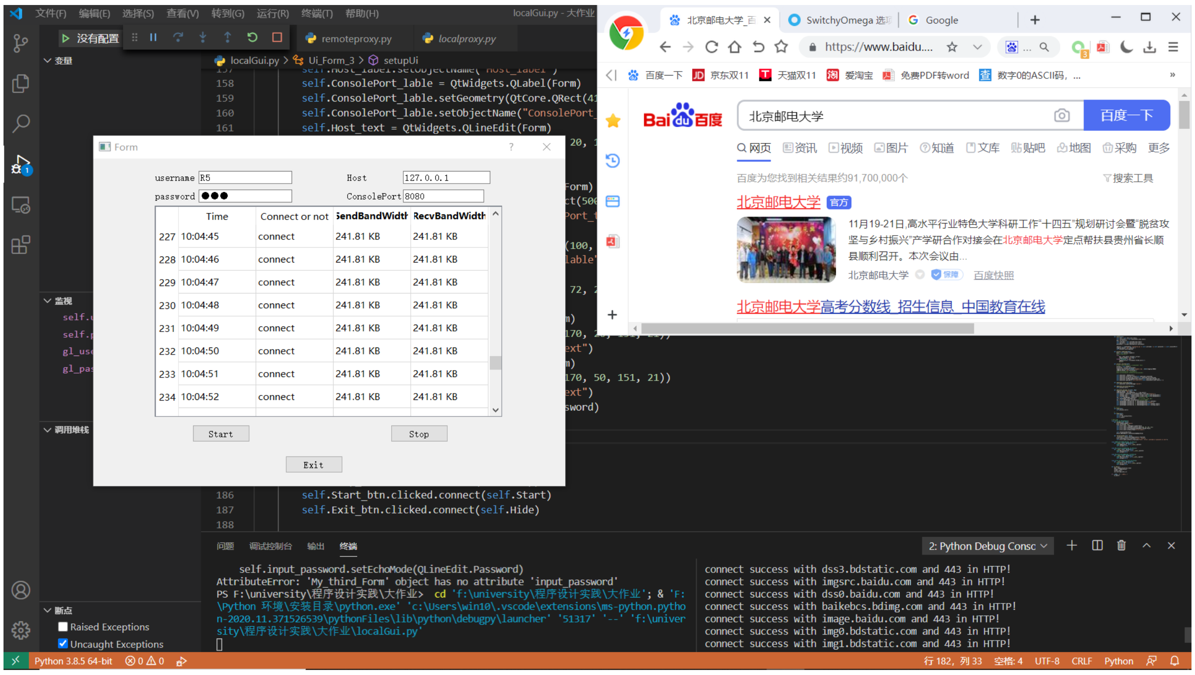
	Time	Connect or not	SendBandWidth	RecvBandWidth
58	10:01:02	connect	6.13 KB	6.13 KB
59	10:01:03	connect	5.99 KB	5.99 KB
60	10:01:04	connect	5.99 KB	5.99 KB
61	10:01:05	connect	5.99 KB	5.99 KB
62	10:01:07	connect	4.01 KB	4.01 KB
63	10:01:08	connect	517 bytes	517 bytes
64	10:01:10	connect	86.27 KB	86.27 KB
65	10:01:11	connect	529.64 KB	529.64 KB

```
        self.Host_label.setGeometry(QtCore.QRect(410, 20,
        self.Host_label.setObjectName("Host_label")
        self.ConsolePort_lable = QtWidgets.QLabel(Form)
        self.ConsolePort_lable.setGeometry(QtCore.QRect(41

        self.Stop_btn.clicked.connect(self.Stop)
        self.Start_btn.clicked.connect(self.Start)
```

```
    self.input_password.setEchoMode(QLineEdit.Password)
AttributeError: 'My_third_Form' object has no attribute 'input_password'
PS F:\university\程序设计实践\大作业>  cd 'f:\university\程序设计实践\大作业'; & 'F:
\Python 环境\安装目录\python.exe' 'c:\Users\win10\.vscode\extensions\ms-python.pytho
n-2020.11.371526539\pythonFiles\lib\python\debugpy\launcher' '51317' '--' 'f:\univer
sity\程序设计实践\大作业\localGui.py'
```

```
Hostlen:19, Host:clients1.google.com, port:443
connect success with dss1.baidu.com and 443 in SOCKS5!
connect success with dss0.baidu.com and 443 in SOCKS5!
connect success with dss1.baidu.com and 443 in SOCKS5!
connect success with dss1.baidu.com and 443 in SOCKS5!
connect success with dss1.baidu.com and 443 in SOCKS5!
connect success with dss2.baidu.com and 443 in SOCKS5!
```

HTTP成功联网



```
        self.ConsolePort_lable = QtWidgets.QLabel(Form)
        self.ConsolePort_lable.setGeometry(QtCore.QRect(41
        self.ConsolePort_lable.setObjectName("ConsolePort_
        self.Host_text = QtWidgets.QLineEdit(Form)
```

username	R5	Host	127.0.0.1
password	●●●	ConsolePort	8080

	Time	Connect or not	SendBandWidth	RecvBandWidth
227	10:04:45	connect	241.81 KB	241.81 KB
228	10:04:46	connect	241.81 KB	241.81 KB
229	10:04:47	connect	241.81 KB	241.81 KB
230	10:04:48	connect	241.81 KB	241.81 KB
231	10:04:49	connect	241.81 KB	241.81 KB
232	10:04:50	connect	241.81 KB	241.81 KB
233	10:04:51	connect	241.81 KB	241.81 KB
234	10:04:52	connect	241.81 KB	241.81 KB

```
        self.Start_btn.clicked.connect(self.Start)
        self.Exit_btn.clicked.connect(self.Hide)
```

```
    self.input_password.setEchoMode(QLineEdit.Password)
AttributeError: 'My_third_Form' object has no attribute 'input_password'
PS F:\university\程序设计实践\大作业>  cd 'f:\university\程序设计实践\大作业'; & 'F:
\Python 环境\安装目录\python.exe' 'c:\Users\win10\.vscode\extensions\ms-python.pytho
n-2020.11.371526539\pythonFiles\lib\python\debugpy\launcher' '51317' '--' 'f:\univer
sity\程序设计实践\大作业\localGui.py'
```

```
connect success with dss3.bdstatic.com and 443 in HTTP!
connect success with imgsrc.baidu.com and 443 in HTTP!
connect success with dss0.baidu.com and 443 in HTTP!
connect success with baikebcs.bdimg.com and 443 in HTTP!
connect success with image.baidu.com and 443 in HTTP!
connect success with img0.bdstatic.com and 443 in HTTP!
connect success with img1.bdstatic.com and 443 in HTTP!
```

北京邮电大学

百度为您找到相关结果约91,700,000个

北京邮电大学 官方

11月19-21日,高水平行业特色大学科研工作"十四五"规划研讨会暨"脱贫攻坚与乡村振兴"产学研合作对接会在北京邮电大学定点帮扶县贵州省长顺县顺利召开。本次会议由...

北京邮电大学高考分数线_招生信息_中国教育在线

7、遇到的问题

（1）因为我是这个学期才学习的python，一开始对于一些python的图形化和协程方面的东西理解得不是特别深入，导致我遇到了很多困难。比如，PyQt5关于QWdget类的属性和方法了解得不够，导致我绕了很多弯路，明明是一个内置函数就可以解决的事情

（2）因为结合到了数据库，需要使用一些简单地SQL语句，SQL语句很简单，但是在实际操作时总会报错，比如找不到表、表重定义、找不到某一属性等等，解决后发现这些其实都是很简单的问题

8、总结

（1）熟练掌握了python的基础应用

（2）对SOCKS5协议和HTTP协议了解得更加透彻

（3）学会了把数据库和python结合

（4）了解了协程和并发