

周瑞发的网站

欢迎访问



39 min. read

Python程序设计课程大作业

📅 2021-07-16 | 📅 2020-12-28 | 👁 4 | 💬 0

Python程序设计课程大作业

班级：2018211308

学号：2018211430

姓名：周瑞发

本地系统

【文字描述】

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

SOCKS5服务

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

```

11         methods.append(ord(await reader.read(1)))
12     if 0 not in methods:#无需认证
13         writer.close()
14         writer.wait_closed()
15     return
16     #回应一个数据包, 包括协议版本号, 指定认证方法
17     writer.write(struct.pack("!BB", VERSION, 0))
18     await writer.drain()
19     request = await reader.read(4)
20     ver, cmd, rsv, atype = struct.unpack("!BBBB", request)
21     assert ver == VERSION
22     assert cmd == 1
23     #ipv4
24     if atype == 1:
25         address = socket.inet_ntoa(await reader.read(4))
26     #域名
27     elif atype == 3:
28         domain_length = await reader.read(1)
29         address = await reader.read(domain_length[0])
30     #ipv6
31     elif atype == 4:
32         address = socket.inet_ntop(socket.AF_INET6, await reader.read(16))
33     else:
34         writer.close()
35         writer.wait_closed()
36     return
37     port = struct.unpack('!H', await reader.read(2))
38     reply = struct.pack("!BBBBIH", VERSION, 0, 0, 1, 0, 0)
39     writer.write(reply)
40     await writer.drain()
41     print(address.decode(), str(port[0]))
42     # 得到了地址和端口号, sock5连接建立完成,然后调用函数, 与remote-proxy连接
43     await xfer_remote(reader, writer, address.decode(), str(port[0]))

```

【文字描述】

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

VER	NMETHODS	METHODS
1	1	1TO255

第一个字段如果是0x05, 则表示为socks5协议

第二个字段表示支持的认证方式的数量

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

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

VER	METHOD
1	1

我们这里回复的应该是5， 0

此时客户端收到服务端的响应请求后，双方握手完成，开始进行协议交互。

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

4、获取地址和端口号之后，就可以与remote-proxy建立连接了

HTTPS隧道服务

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

【文字描述】

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

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

3、连接建立好后，就可以与remote-proxy通信了

与远端模块通信

```
1  async def xfer_remote(reader, writer, address, port):
2      #与remote通信的协议是自己定的，为了方便，我直接使用了http协议
3      reader_remote,writer_remote = await asyncio.open_connection('123.56.111.64', 1
4      http_connect = 'CONNECT ' + address + ':' + port + ' HTTP/1.1'
5      http_connect += ' %' + username + '%' + password + '%'
6      logging.info(http_connect)
7      writer_remote.write(http_connect.encode())
8      await writer_remote.drain()
9      reply = await reader_remote.read(1024)
10     if('HTTP/1.1 200 OK' in reply.decode()):#与remote连接建立成功
11         tasks = [read_trans(reader, writer_remote), write_trans(reader_remote, wri
12         await asyncio.wait(tasks)
13     else:
14         logging.info('connect to remote failed!')
15
16  async def read_trans(reader, writer_remote):
17      while True:
18          data = await reader.read(4096)
19          if not data:
20              logging.info('disconnect')
21              break
22          writer_remote.write(data)
23          await writer_remote.drain()
24
25  async def write_trans(reader_remote, writer):
26      while True:
27          data = await reader_remote.read(4096)
28          if not data:
29              logging.info('disconnect')
30              break
31          writer.write(data)
32          await writer.drain()
```

【文字描述】

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

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

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

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

图形管理界面

```

1  class MainWindow(QMainWindow):
2      def __init__(self, parent=None):
3          super(MainWindow, self).__init__(parent)
4          uic.loadUi("mainwindow.ui", self) # 加载界面
5          self.pushButton.clicked.connect(self.startClicked)
6
7          self.process = QProcess()
8          self.process.setProcessChannelMode(QProcess.MergedChannels)
9          self.process.finished.connect(self.processFinished)#当进程结束，触发processF
10         self.process.started.connect(self.processStarted)# 当进程已经开始了，触发pro
11         self.process.readyReadStandardOutput.connect(self.processReadyRead) #信号
12
13         def processReadyRead(self):
14             data = self.process.readAll()
15             try:
16                 msg = data.data().decode().strip()
17                 logging.debug(f'msg={msg}')
18             except Exception as exc:
19                 logging.error(f'{traceback.format_exc()}')
20                 exit(1)
21
22         def processStarted(self): #进程开始后，调用该函数
23             process = self.sender() # 此处等同于 self.process 只不过使用sender适应性更好
24             processId = process.processId()
25             logging.debug(f'pid={processId}')
```

```
26         self.pushButton.setText('stop')
27
28         self.websocket = QWebSocket()
29         self.websocket.connected.connect(self.websocketConnected)
30         self.websocket.disconnected.connect(self.websocketDisconnected)
31         self.websocket.textMessageReceived.connect(self.websocketMsgRcvd) #当收到对
32         self.websocket.open(QUrl(f'ws://127.0.0.1:{self.consolePortLine.text()}'))
33
34     def startClicked(self): #当点击开始按钮时
35         btn = self.sender()
36         text = btn.text().lower()
37         if text.startswith('start'):
38             listenPort = self.listenPortLine.text() #本地端口10086
39             username = self.usernameLine.text() #用户名
40             password = self.passwordLine.text() #密码
41             consolePort = self.consolePortLine.text() #websockets端口
42             remoteHost = self.remoteHostLine.text() #远程主机ip
43             remotePort = self.remotePortLine.text() #远程主机端口
44             pythonExec = os.path.basename(sys.executable)
45             cmdLine = f'{pythonExec} local-proxy.py -p {listenPort} -u {username}'
46             logging.info(cmdLine)
47             logging.debug(f'cmd={cmdLine}')
48             self.process.start(cmdLine)
49         else:
50             self.process.kill()
51     def processFinished(self):
52         process = self.sender()
53         log.debug(f'pid={process.processId()}')
54         self.startBtn.setText('Start')
55         self.processIdLine.setText('')
56     def websocketConnected(self):
57         self.websocket.sendTextMessage('connect successful') #连接建立后发送
58
59     def websocketDisconnected(self):
60         self.process.kill()
61
62     def websocketMsgRcvd(self, msg):
63         logging.debug(f'msg={msg}')
64         sendBandwidth, recvBandwidth, *_ = msg.split()
65         nowTime = QDateTime.currentDateTime().toString('hh:mm:ss')
66         logging.info(sendBandwidth)
67         logging.info(recvBandwidth)
68         self.sendBandwidthLine.setText(f'{nowTime} {sendBandwidth} Bps')
69         self.lineEdit_2.setText(f'{nowTime} {recvBandwidth} Bps')
```

【文字描述】

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

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

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

远端系统

【文字描述】

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

与本地模块通信

```

1  async def handle(reader_local, writer_local):
2      logging.info('start working')
3      global username_to_token_bucket
4      db = await aiosqlite.connect('user.db')
5      http_connect = (await reader_local.read(1024))
6      http_connect = http_connect.decode()
7      logging.info(http_connect)
8
9      i = 0
10     while(http_connect[i] != ':'):
11         i += 1
12     domain_name = http_connect[8 : i]
13     j = i
14     while(http_connect[j] != ' '):
15         j += 1
16     port = http_connect[i + 1 : j]
17     i = 0
18     while(http_connect[i] != '%'):

```

```

19         i += 1
20     j = i + 1
21     while(http_connect[j] != '%'):
22         j += 1
23     k = j + 1
24     while(http_connect[k] != '%'):
25         k += 1
26     username = http_connect[i + 1: j]
27     password = http_connect[j + 1: k]
28     sql = 'SELECT * FROM user where name = \'' + username + '\'' and password = \''
29     cursor = await db.execute(sql)
30     row = await cursor.fetchall()
31     await cursor.close()
32     if(len(row) != 1):
33         logging.error('wrong account')
34         return
35     else:
36         logging.info('right account')
37     if(username not in username_to_token_bucket.keys()):#用户名不在dict中, 要创建令牌
38         username_to_token_bucket[username] = 0
39         logging.info('init bucket')
40     reader_remote,writer_remote = await asyncio.open_connection(domain_name,port)
41     sql = 'select dataRate FROM user where name = \'' + username + '\''
42     cursor = await db.execute(sql)
43     row = await cursor.fetchall()
44     await cursor.close()
45     speed = row[0][0]
46     reply = 'HTTP/1.1 200 OK\r\n\r\n'
47     writer_local.write(reply.encode())
48     await writer_local.drain()
49
50     tasks = [read_trans(reader_local, writer_remote), write_trans(reader_remote, w
51     await asyncio.wait(tasks,return_when=asyncio.FIRST_COMPLETED)
52     await db.close()
53
54     async def read_trans(reader, writer_remote):
55         while True:
56             data = await reader.read(4096)
57             if not data:
58                 logging.info('disconnect from client')
59                 return
60             writer_remote.write(data)
61             await writer_remote.drain()
62
63     async def write_trans(reader_remote, writer, username, speed):
64         global username_to_token_bucket
65         data = ''

```



```

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

```

【文字描述】

- 1、local-proxy和remote-proxy之间的通信协议是自己定的，我这里直接使用了http协议。
- 2、在收到local-proxy的消息后，从消息中解析出地址、端口号、用户名和密码。
- 3、然后进行用户名密码验证，验证通过之后，从数据库中查询该用户的速率。
- 4、向local-proxy回复 'HTTP/1.1 200 OK\r\n\r\n'，表明连接建立成功
- 5、接下来local-proxy和remote-proxy就可以进行通信了。
- 6、在remote-proxy向local-proxy发送数据时，要在此处执行限速。只有当令牌的数量足够，才会发送数据包，这也就实现了下载限速。

多用户管理

```

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

```

【文字描述】

- 1、利用start_server函数，监听来自所有IP地址对端口10010的连接。
- 2、连接建立后回调handle函数，在handle函数对连接进行处理
↑
- 3、每来一个用户，就会新建一个连接并且调用handle函数，这样就实现了多用户管理。

用户流控

```

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

```

【文字描述】

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

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

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

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

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

用户数据库管理接口

```

1  @app.exception(exceptions.NotFound)
2  async def ignore_404(req, exc):
3      return response.text('err_url', status=404)
4
5  # 获得所有用户信息
6  @app.get('/user')
7  async def userList(req):
8      userList = list()
9      sql = 'select name, password, dataRate from user;'
10     async with aiosqlite.connect(app.config.DB_NAME) as db:
11         cursor = await db.execute(sql)
12         async for row in cursor:
13             user = {'name' : row[0], 'password' : row[1], 'dataRate:' : row[2]}
14             logging.debug(f'{user}')
15             userList.append(user)
16     return response.json(userList)
17
18 # 更新用户信息
19 @app.put('/user')
20 async def updateUserInfo(req):
21     update_user_info = req.json
22     name = update_user_info.get('name')
23     password = update_user_info.get('password')
24     dataRate = update_user_info.get('dataRate')
25     db = await aiosqlite.connect(app.config.DB_NAME)
26     sql = 'select * from user where name = \'' + name + '\'';'
27     cursor = await db.execute(sql)
28     row = await cursor.fetchall()
29     if(len(row) == 0):
30         await cursor.close()

```

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

【文字描述】

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

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

程序完整源码

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

local.py

```
1  import asyncio
2  import struct
3  import socket
4  import logging
5  logging.basicConfig(level=logging.INFO)
6  import sys
7  import getopt
8  VERSION = 5
9  async def socks5(first, reader, writer):
10     addr_from = writer.get_extra_info('peername')
11     logging.info(f'connect from{addr_from}')
12     header = await reader.read(1)
13     header = first + header
14     ver, num_method = struct.unpack("!BB", header)
15     logging.info(f'ver == VERSION:{ver == VERSION}')
16     logging.info('num_method = %d' % num_method)
17     methods = []
18     for i in range(num_method):
19         methods.append(ord(await reader.read(1)))
20     if 0 not in methods:#无需认证
21         writer.close()
22         writer.wait_closed()
23         return
24     #回应一个数据包，包括协议版本号，指定认证方法
25     writer.write(struct.pack("!BB", VERSION, 0))
26     await writer.drain()
27     request = await reader.read(4)
28     ver, cmd, rsv, atype = struct.unpack("!BBBB", request)
```

```

29     assert ver == VERSION
30     assert cmd == 1
31     #ipv4
32     if atype == 1:
33         address = socket.inet_ntoa(await reader.read(4))
34     #域名
35     elif atype == 3:
36         domain_length = await reader.read(1)
37         address = await reader.read(domain_length[0])
38     #ipv6
39     elif atype == 4:
40         address = socket.inet_ntop(socket.AF_INET6, await reader.read(16))
41     else:
42         writer.close()
43         writer.wait_closed()
44         return
45     port = struct.unpack('!H', await reader.read(2))
46     reply = struct.pack("!BBBBIH", VERSION, 0, 0, 1, 0, 0)
47     writer.write(reply)
48     await writer.drain()
49     logging.info(address.decode(), str(port[0]))
50     # 得到了地址和端口号, sock5连接建立完成,然后调用函数,与remote-proxy连接
51     await xfer_remote(reader, writer, address.decode(), str(port[0]))
52
53
54 async def httptunnel(first, reader, writer):
55     http_connect = (await reader.read(1024))
56     http_connect = (first + http_connect).decode()
57     #从http请求中解析出ip和端口号
58     i = 0
59     while(http_connect[i] != ':'):
60         i += 1
61     domain_name = http_connect[8 : i]
62     j = i
63     while(http_connect[j] != ' '):
64         j += 1
65     port = http_connect[i + 1 : j]
66     reply = 'HTTP/1.1 200 OK\r\n\r\n'
67     writer.write(reply.encode())
68     await writer.drain()
69     await xfer_remote(reader, writer, domain_name, port)
70
71 async def xfer_remote(reader, writer, address, port):
72     #与remote通信的协议是自己定的,为了方便,我直接使用了http协议
73     reader_remote,writer_remote = await asyncio.open_connection('123.56.111.64',
74     http_connect = 'CONNECT ' + address + ':' + port + ' HTTP/1.1'
75     http_connect += ' %' + username + '%' + password + '%'

```

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

```

123     async with server:
124         await server.serve_forever()
125
126 asyncio.run(main())
↑

```

localGui.py

```

1  from PyQt5.QtCore import *
2  from PyQt5.QtGui import *
3  from PyQt5.QtNetwork import *
4  from PyQt5.QtWidgets import *
5  from PyQt5.QtWebSockets import *
6  from mainwindow import Ui_MainWindow
7  import sys
8  from PyQt5 import QtWidgets, uic
9  import logging
10 import os
11 import humanfriendly
12 class MainWindow(QtWidgets.QMainWindow):
13     def __init__(self, parent=None):
14         super(MainWindow, self).__init__(parent)
15         uic.loadUi("mainwindow.ui", self) # 加载界面
16         self.pushButton.clicked.connect(self.startClicked)
17
18         self.process = QProcess()
19         self.process.setProcessChannelMode(QProcess.MergedChannels)
20         self.process.finished.connect(self.processFinished)#当进程结束，触发processF
21         self.process.started.connect(self.processStarted)# 当进程已经开始了，触发pro
22         self.process.readyReadStandardOutput.connect(self.processReadyRead) #信号
23
24     def processReadyRead(self):
25         data = self.process.readAll()
26         try:
27             msg = data.data().decode().strip()
28             logging.debug(f'msg={msg}')
29         except Exception as exc:
30             logging.error(f'{traceback.format_exc()}')
31             exit(1)
32
33     def processStarted(self): #进程开始后，调用该函数
34         process = self.sender() # 此处等同于 self.process 只不过使用sender适应性更好
35         processId = process.processId()
36         logging.debug(f'pid={processId}')
37         self.pushButton.setText('stop')

```



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

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

↑

remote.py

```
1  import asyncio
2  import struct
3  import socket
4  import logging
5  import time
6  logging.basicConfig(level=logging.INFO)
7  import nest_asyncio
8  nest_asyncio.apply()
9  import aiosqlite
10 async def handle(reader_local, writer_local):
11     logging.info('start working')
12     global username_to_token_bucket
13     db = await aiosqlite.connect('user.db')
14     http_connect = (await reader_local.read(1024))
15     http_connect = http_connect.decode()
16     logging.info(http_connect)
17
18     i = 0
19     while(http_connect[i] != ':'):
20         i += 1
21     domain_name = http_connect[8 : i]
22     j = i
23     while(http_connect[j] != ' '):
24         j += 1
25     port = http_connect[i + 1 : j]
26     i = 0
27     while(http_connect[i] != '%'):
28         i += 1
29     j = i + 1
30     while(http_connect[j] != '%'):
31         j += 1
32     k = j + 1
33     while(http_connect[k] != '%'):
34         k += 1
35     username = http_connect[i + 1: j]
36     password = http_connect[j + 1: k]
37     sql = 'SELECT * FROM user where name = \'' + username + '\'' and password = \''
38     cursor = await db.execute(sql)
39     row = await cursor.fetchall()
```

```

40     await cursor.close()
41     if(len(row) != 1):
42         logging.error('wrong account')
43         return
44     else:
45         logging.info('right account')
46     if(username not in username_to_token_bucket.keys()):#用户名不在dict中, 要创建令
47         username_to_token_bucket[username] = 0
48         logging.info('init bucket')
49     reader_remote,writer_remote = await asyncio.open_connection(domain_name,port)
50     sql = 'select dataRate FROM user where name = \'' + username + '\''
51     cursor = await db.execute(sql)
52     row = await cursor.fetchall()
53     await cursor.close()
54     speed = row[0][0]
55     reply = 'HTTP/1.1 200 OK\r\n\r\n'
56     writer_local.write(reply.encode())
57     await writer_local.drain()
58
59     tasks = [read_trans(reader_local, writer_remote), write_trans(reader_remote,
60     await asyncio.wait(tasks,return_when=asyncio.FIRST_COMPLETED)
61     await db.close()
62
63
64     async def read_trans(reader, writer_remote):
65         while True:
66             data = await reader.read(4096)
67             if not data:
68                 logging.info('disconnect from clinet')
69                 return
70             writer_remote.write(data)
71             await writer_remote.drain()
72
73     async def write_trans(reader_remote, writer, username, speed):
74         global username_to_token_bucket
75         data = ''
76         while True:
77             if(data == ''):
78                 data = await reader_remote.read(int(0.01 * speed))# 10.01s      speed
79             if not data:
80                 logging.info('disconnect from server')
81                 return
82             if(username_to_token_bucket[username] < 10):#如果桶里的令牌不够那么就等待, 注
83                 # 目的是让cpu去执行其他部分的代码, 防止在此处阻塞
84                 await asyncio.sleep(0)
85                 continue
86             else:#令牌够了, 把data发出去, 同时把data清空

```

```

87         username_to_token_bucket[username] -= 10
88         writer.write(data)
89         await writer.drain()
90         data = ''
↑ 91 async def token_bucket_plus_one():
92     global username_to_token_bucket
93     while True:
94         for k in username_to_token_bucket.keys():# 每1秒可以攒够1000个令牌, 所以平均
95             if username_to_token_bucket[k] < 10000:
96                 username_to_token_bucket[k] += 10
97                 print(username_to_token_bucket[k])
98             await asyncio.sleep(0.01)
99 async def main():
100     asyncio.create_task(token_bucket_plus_one())
101     server = await asyncio.start_server(handle, '0.0.0.0', 10010)
102     async with server:
103         await server.serve_forever()
104     username_to_token_bucket = {}
105     asyncio.run(main())

```

remoteRest.py

```

1  from sanic import Sanic
2  from sanic import response
3  from sanic import exceptions
4  from sanic.response import json
5  import aiosqlite
6  import logging
7
8  app = Sanic("RemoteProxyAdmin")
9  app.config.DB_NAME = 'user.db'
10
11 @app.exception(exceptions.NotFound)
12 async def ignore_404(req, exc):
13     return response.text('err_url', status=404)
14
15 # 获得所有用户信息
16 @app.get('/user')
17 async def userList(req):
18     userList = list()
19     sql = 'select name, password, dataRate from user;'
20     async with aiosqlite.connect(app.config.DB_NAME) as db:
21         cursor = await db.execute(sql)
22         async for row in cursor:

```

```
23         user = {'name' : row[0], 'password' : row[1], 'dataRate:' : row[2]}
24         logging.debug(f'{user}')
25         userList.append(user)
26     return response.json(userList)
27
28 # 更新用户信息
29 @app.put('/user')
30 async def updateUserInfo(req):
31     update_user_info = req.json
32     name = update_user_info.get('name')
33     password = update_user_info.get('password')
34     dataRate = update_user_info.get('dataRate')
35     db = await aisqlite.connect(app.config.DB_NAME)
36     sql = 'select * from user where name = \'' + name + '\'';
37     cursor = await db.execute(sql)
38     row = await cursor.fetchall()
39     if(len(row) == 0):
40         await cursor.close()
41         return response.json({"msg":"user not exist"})
42     sql = 'update user set password = \'' + password + '\', dataRate = ' + str(dataRate) + ' where name = \'' + name + '\'';
43     await db.execute(sql)
44     await db.commit()
45     await db.close()
46     return response.json({"msg":"update successful"})
47
48 # 增加用户信息
49 @app.post('/user')
50 async def insertUserInfo(req):
51     insert_user_info = req.json
52     name = insert_user_info.get('name')
53     password = insert_user_info.get('password')
54     dataRate = insert_user_info.get('dataRate')
55     sql = 'select * from user where name = \'' + name + '\'';
56     db = await aisqlite.connect(app.config.DB_NAME)
57     cursor = await db.execute(sql)
58     row = await cursor.fetchall()
59     if(len(row) != 0):
60         await cursor.close()
61         return response.json({"msg":"user already exist"})
62     sql = 'insert into user values(\'' + name + '\', \'' + password + '\', ' + str(dataRate) + ' where name = \'' + name + '\'';
63     await db.execute(sql)
64     await db.commit()
65     await db.close()
66     return response.json({"msg":"insert successful"})
67
68 #删除用户信息
69 @app.delete('/user')
```

```
70 async def deleteUserInfo(req):
71     delete_user_info = req.json
72     name = delete_user_info.get('name')
73     sql = 'select * from user where name = \'' + name + '\'';
74     db = await aiosqlite.connect(app.config.DB_NAME)
75     cursor = await db.execute(sql)
76     row = await cursor.fetchall()
77     if(len(row) == 0):
78         await cursor.close()
79         return response.json({"msg": "user not exist"})
80     sql = 'delete from user where name = \'' + name + '\'';
81     await db.execute(sql)
82     await db.commit()
83     await db.close()
84     return response.json({"msg": "delete successful"})
85
86 @app.route("/zrf")
87 async def test(request):
88     return response.json({"hello": "world"})
89
90 if __name__ == "__main__":
91     app.run(host="0.0.0.0", port=8000)
92
```

[Python程序设计作业#8](#)[Python程序设计作业#7](#) >

昵称	邮箱	网址(http://)
<div>Just go go</div>		
<div><div><div><div></div></div></div><div><div></div></div></div>		
<div><div><div></div></div><div>提交</div></div>		

来发评论吧~



© 2021 周瑞发
由 [Hexo](#) & [NexT.Muse](#) 强力驱动