

作业

用SocketServer来改写ChatServer

使用ThreadingTCPServer改写ChatServer

```
1  import threading
2  from socketserver import ThreadingTCPServer, StreamRequestHandler
3  import logging
4
5  FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
6  logging.basicConfig(format=FORMAT, level=logging.INFO)
7
8
9  class ChatHandler(StreamRequestHandler):
10     clients = {}
11     def setup(self):
12         super().setup()
13         self.event = threading.Event()
14         self.clients[self.client_address] = self.wfile
15
16     def handle(self):
17         super().handle() # 虽然父类什么都没做，但是调用是个好习惯
18         while not self.event.is_set():
19             data = self.rfile.readline().strip()
20             if data == b'quit' or data == b'':
21                 break
22             msg = "From {}:{}. data={}".format(*self.client_address, data)
23             for f in self.clients.values():
24                 f.write(msg.encode())
25                 f.flush()
26
27     def finish(self):
28         self.clients.pop(self.client_address)
29         super().finish()
30         self.event.set()
31
32
33  class ChatServer:
34     def __init__(self, ip='127.0.0.1', port=9999):
35         self.server = ThreadingTCPServer((ip, port), ChatHandler)
36         self.server.daemon_threads = True
37
38     def start(self):
39         threading.Thread(
40             target=self.server.serve_forever, name='chatserver',
41             daemon=True).start()
42
43     def stop(self):
44         self.server.server_close()
45
46  if __name__ == '__main__':
47     cs = ChatServer()
48     cs.start()
```

```

49     while True:
50         cmd = input('>>').strip()
51         if cmd == 'quit':
52             cs.stop()
53             break
54         print(threading.enumerate())

```

问题

上例 self.clients.pop(self.client_address) 能执行到吗？

如果连接的线程中handle方法中抛出异常，例如客户端主动断开导致的异常，线程崩溃，self.clients的pop方法还能执行吗？

当然能执行，基类源码保证了即使异常，也能执行finish方法。但不代表不应该不捕获客户端各种异常。

注意：此程序线程不安全

使用IO多路复用改写群聊软件

不需要启动多线程来执行socket的accept、recv方法了

```

1  import threading
2  import selectors
3  import socket
4  import logging
5
6  FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
7  logging.basicConfig(format=FORMAT, level=logging.INFO)
8
9
10 class ChatServer:
11     def __init__(self, ip='127.0.0.1', port=9999):
12         self.addr = ip, port
13         self.sock = socket.socket()
14         self.sock.setblocking(False) # 非阻塞
15         self.event = threading.Event()
16         # 构建本系统最优Selector
17         self.selector = selectors.DefaultSelector()
18
19     def start(self):
20         self.sock.bind(self.addr)
21         self.sock.listen()
22         key = self.selector.register(self.sock, selectors.EVENT_READ,
self.accept)
23
24         threading.Thread(target=self.select, name='select').start()
25
26     def select(self):
27         with self.selector:
28             while not self.event.is_set():
29                 events = self.selector.select(0.5) # 超时返回[]
30                 # 监听注册的对象的事件，发生被关注事件则返回events
31                 for key, mask in events:
32                     key.data(key.fileobj, mask)
33
34     def accept(self, server:socket.socket, mask):

```

```

35         conn, raddr = server.accept()
36         conn.setblocking(False)
37         logging.info("New client {} accepted. fd={}".format(raddr,
conn.fileno()))
38
39         key = self.selector.register(conn, selectors.EVENT_READ, self.recv)
40
41     def recv(self, conn:socket.socket, mask):
42         data = conn.recv(1024).strip()
43         if data == b'' or data == b'quit':
44             self.selector.unregister(conn)
45             conn.close() # 关闭前一定要注销
46             return
47         msg = "Your msg={}".format(data.decode()).encode()
48         logging.info(msg)
49         for key in self.selector.get_map().values():
50             print(key.data.__name__)
51             # 特别注意，绑定的方法==和is的区别
52             print(key.data is self.accept, key.data == self.accept)
53             print(key.data is self.recv, key.data == self.recv)
54             if key.data == self.recv:
55                 key.fileobj.send(msg)
56
57     def stop(self):
58         self.event.set()
59
60
61 if __name__ == '__main__':
62     cs = ChatServer()
63     cs.start()
64     while True:
65         cmd = input('>>').strip()
66         if cmd == 'quit':
67             cs.stop()
68             break
69         print(*cs.selector.get_map().values())

```

本例只完成基本功能，其他功能如有需要，请自行完成。

注意使用IO多路复用，使用了几个线程？

特别注意key.data == self.recv

自己实现HTTPServer

<https://webob.org/>

<https://docs.pylonsproject.org/projects/webob/en/stable/>

```

1 import threading
2 import selectors
3 import socket
4 import logging
5 import webob
6
7 FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"

```

```

8 logging.basicConfig(format=FORMAT, level=logging.INFO)
9
10 html_content = """
11 <html>
12 <head><title></title></head>
13 <body>
14     欢迎访问马哥教育
15 </body>
16 </html>
17 """
18
19 class WebServer:
20     def __init__(self, ip='0.0.0.0', port=80):
21         self.addr = ip, port
22         self.sock = socket.socket()
23         self.sock.setblocking(False) # 非阻塞
24         self.event = threading.Event()
25         # 构建本系统最优Selector
26         self.selector = selectors.DefaultSelector()
27
28     def start(self):
29         self.sock.bind(self.addr)
30         self.sock.listen()
31         key = self.selector.register(self.sock, selectors.EVENT_READ,
self.accept)
32
33         threading.Thread(target=self.select, name='select').start()
34
35     def select(self):
36         with self.selector:
37             while not self.event.is_set():
38                 events = self.selector.select(1) # 超时返回[]
39                 # 监听注册的对象的事件，发生被关注事件则返回events
40                 print(events)
41                 for key, mask in events:
42                     key.data(key.fileobj, mask)
43
44     def accept(self, server:socket.socket, mask):
45         conn, raddr = server.accept()
46         conn.setblocking(False)
47         logging.info("New client {} accepted. fd={}".format(raddr,
conn.fileno()))
48
49         key = self.selector.register(conn, selectors.EVENT_READ, self.recv)
50
51     def recv(self, conn:socket.socket, mask):
52         with conn: # 用完就断
53             try:
54                 data = conn.recv(1024).strip()
55                 # 收到request报文，下面要做url映射等，此处都省略
56                 request = webob.Request.from_bytes(data)
57                 print(request.url)
58                 print('=' * 30)
59
60                 response = webob.Response(html_content, status=201)
61                 response.headers.add('Server', 'MageServer')
62                 firstline = 'HTTP/1.1 {}'.format(response.status)
63                 print(response.headerlist)

```

```

64         headers = "\r\n".join(
65             [firstline] + ["{}: {}".format(k, v) for k, v in
response.headerlist] + ['', ''])
66         ) # 响应头: 第一行、头部字段、2个回车换行
67         body = response.body
68         print(type(headers), type(body))
69         content = headers.encode() + body
70
71         conn.send(content)
72     finally:
73         self.selector.unregister(conn)
74
75     def stop(self):
76         self.event.set()
77
78
79 if __name__ == '__main__':
80     cs = WebServer()
81     cs.start()
82     while True:
83         cmd = input('>>').strip()
84         if cmd == 'quit':
85             cs.stop()
86             break

```

还要实现的话就是，路径映射

