作业

用SocketServer来改写ChatServer

使用ThreadingTCPServer改写ChatServer

```python
import threading
from socketserver import ThreadingTCPServer, StreamRequestHandler
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatHandler(StreamRequestHandler):
    clients = {}
    def setup(self):
        super().setup()
        self.event = threading.Event()
        self.clients[self.client_address] = self.wfile

    def handle(self):
        super().handle() # 虽然父类什么都没做，但是调用是个好习惯
        while not self.event.is_set():
            data = self.rfile.read1().strip()
            if data == b'quit' or data == b'':
                break
            msg = "From {}:{}. data={}".format(*self.client_address, data)
            for f in self.clients.values():
                f.write(msg.encode())
                f.flush()

    def finish(self):
        self.clients.pop(self.client_address)
        super().finish()
        self.event.set()


class ChatServer:
    def __init__(self, ip='127.0.0.1', port=9999):
        self.server = ThreadingTCPServer((ip, port), ChatHandler)
        self.server.daemon_threads = True

    def start(self):
        threading.Thread(
            target=self.server.serve_forever, name='chatserver',
            daemon=True).start()

    def stop(self):
        self.server.server_close()

if __name__ == '__main__':
    cs = ChatServer()
    cs.start()
```

```
    while True:
        cmd = input('>>').strip()
        if cmd == 'quit':
            cs.stop()
            break
        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

```
import threading
import selectors
import socket
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatServer:
    def __init__(self, ip='127.0.0.1', port=9999):
        self.addr = ip, port
        self.sock = socket.socket()
        self.sock.setblocking(False) # 非阻塞
        self.event = threading.Event()
        # 构建本系统最优Selector
        self.selector = selectors.DefaultSelector()

    def start(self):
        self.sock.bind(self.addr)
        self.sock.listen()
        key = self.selector.register(self.sock, selectors.EVENT_READ,
self.accept)

        threading.Thread(target=self.select, name='select').start()

    def select(self):
        with self.selector:
            while not self.event.is_set():
                events = self.selector.select(0.5) # 超时返回[]
                # 监听注册的对象的事件，发生被关注事件则返回events
                for key, mask in events:
                    key.data(key.fileobj, mask)

    def accept(self, server:socket.socket, mask):
```

```
        conn, raddr = server.accept()
        conn.setblocking(False)
        logging.info("New client {} accepted. fd={}".format(raddr,
conn.fileno()))

        key = self.selector.register(conn, selectors.EVENT_READ, self.recv)

    def recv(self, conn:socket.socket, mask):
        data = conn.recv(1024).strip()
        if data == b'' or data == b'quit':
            self.selector.unregister(conn)
            conn.close() # 关闭前一定要注销
            return
        msg = "Your msg={}".format(data.decode()).encode()
        logging.info(msg)
        for key in self.selector.get_map().values():
            print(key.data.__name__)
            # 特别注意，绑定的方法==和is的区别
            print(key.data is self.accept, key.data == self.accept)
            print(key.data is self.recv, key.data == self.recv)
            if key.data == self.recv:
                key.fileobj.send(msg)

    def stop(self):
        self.event.set()


if __name__ == '__main__':
    cs = ChatServer()
    cs.start()
    while True:
        cmd = input('>>').strip()
        if cmd == 'quit':
            cs.stop()
            break
        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/

```
import threading
import selectors
import socket
import logging
import webob

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
```

```
logging.basicConfig(format=FORMAT, level=logging.INFO)

html_content = """
<html>
<head><title></title></head>
<body>
    欢迎访问马哥教育
</body>
</html>
"""

class WebServer:
    def __init__(self, ip='0.0.0.0', port=80):
        self.addr = ip, port
        self.sock = socket.socket()
        self.sock.setblocking(False) # 非阻塞
        self.event = threading.Event()
        # 构建本系统最优Selector
        self.selector = selectors.DefaultSelector()

    def start(self):
        self.sock.bind(self.addr)
        self.sock.listen()
        key = self.selector.register(self.sock, selectors.EVENT_READ,
self.accept)

        threading.Thread(target=self.select, name='select').start()

    def select(self):
        with self.selector:
            while not self.event.is_set():
                events = self.selector.select(1)# 超时返回[]
                # 监听注册的对象的事件，发生被关注事件则返回events
                print(events)
                for key, mask in events:
                    key.data(key.fileobj, mask)

    def accept(self, server:socket.socket, mask):
        conn, raddr = server.accept()
        conn.setblocking(False)
        logging.info("New client {} accepted. fd={}".format(raddr,
conn.fileno()))

        key = self.selector.register(conn, selectors.EVENT_READ, self.recv)

    def recv(self, conn:socket.socket, mask):
        with conn: # 用完就断
            try:
                data = conn.recv(1024).strip()
                # 收到request报文，下面要做url映射等，此处都省略
                request = webob.Request.from_bytes(data)
                print(request.url)
                print('=' * 30)

                response = webob.Response(html_content, status=201)
                response.headers.add('Server', 'MageServer')
                firstline = 'HTTP/1.1 {}'.format(response.status)
                print(response.headerlist)
```

```
                headers = "\r\n".join(
                    [firstline] + ["{}: {}".format(k, v) for k, v in
response.headerlist] + ['', '']
                ) # 响应头：第一行、头部字段、2个回车换行
                body = response.body
                print(type(headers), type(body))
                content = headers.encode() + body

                conn.send(content)
            finally:
                self.selector.unregister(conn)

    def stop(self):
        self.event.set()


if __name__ == '__main__':
    cs = WebServer()
    cs.start()
    while True:
        cmd = input('>>').strip()
        if cmd == 'quit':
            cs.stop()
            break
```

还要实现的话就是，路径映射