张岩林
路漫漫其修远兮，吾将上下而求索!

全屏 目录

python之socket编程

本章内容

1、socket
2、IO多路复用
3、socketserver

Socket

socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，对于文件用【打开】【读写】【关闭】模式来操作。socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作（读/写IO、打开、关闭）

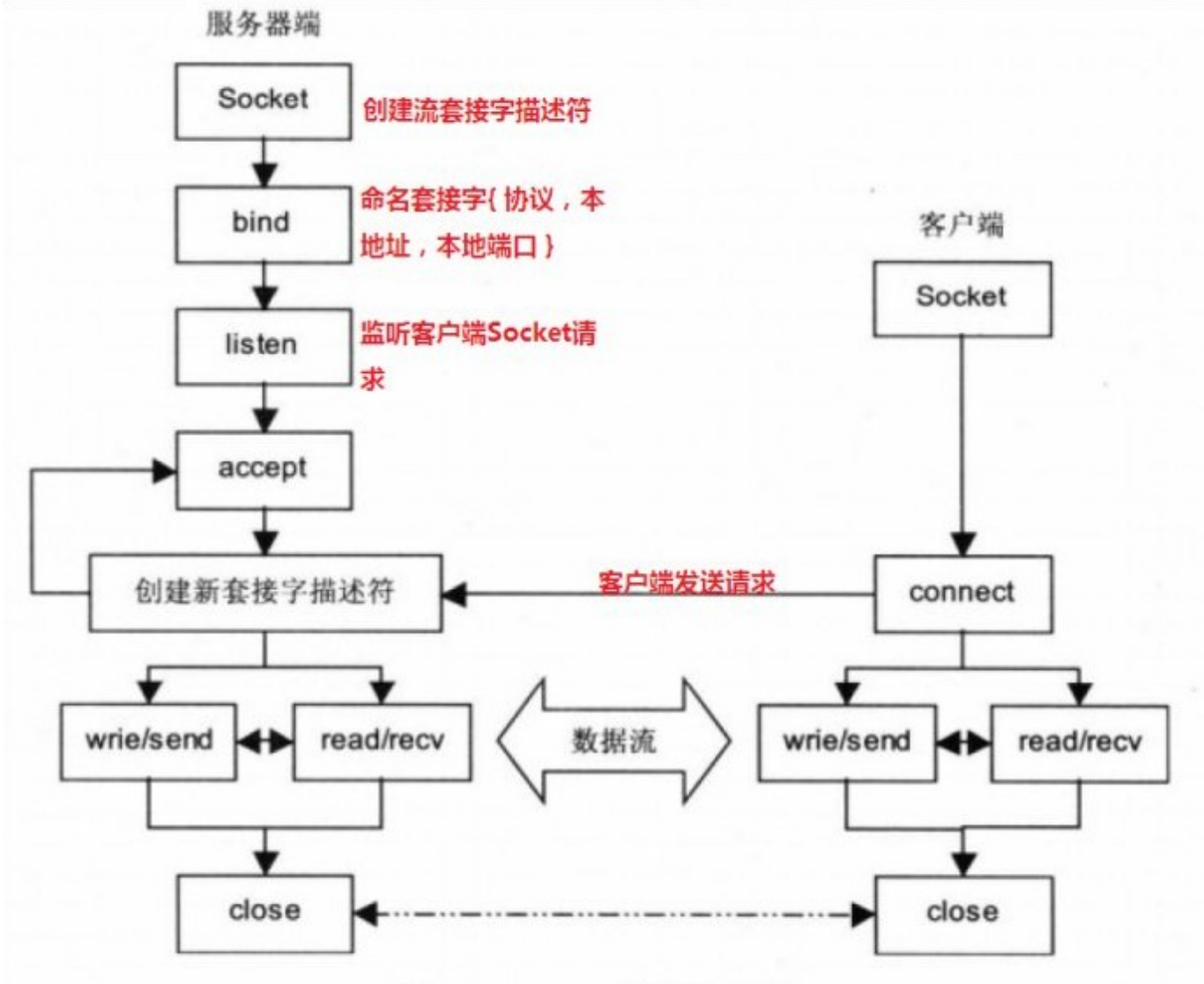
基本上，Socket 是任何一种计算机网络通讯中最基础的内容。例如当你在[illegible]入 http://www.cnblogs.com/ 时，你会打开一个套接字，然后连接到 http:[illegible]/ 并读取响应的页面然后然后显示出来。而其他一些聊天客户端如 gtalk 和 sk[illegible]可网络通讯都是通过 Socket 来完成的。

Python 官方关于 Socket 的函数请看 http://docs.python.org/library/s[illegible]

socket和file的区别：

1、file模块是针对某个指定文件进行【打开】【读写】【关闭】

2、socket模块是针对 服务器端 和 客户端Socket 进行【打开】【读写】【关闭】



那我们就先来创建一个socket服务端吧

```
import socket

sk = socket.socket()
sk.bind(("127.0.0.1",8080))
sk.listen(5)

conn,address = sk.accept()
sk.sendall(bytes("Hello world",encoding="utf-8"))
```

```
import socket

obj = socket.socket()
obj.connect(("127.0.0.1",8080))

ret = str(obj.recv(1024),encoding="utf-8")
print(ret)
```

socket更多功能

```
    def bind(self, address): # real signature unknown; restored from __doc__
        """
        bind(address)

        Bind the socket to a local address.  For IP sockets, the address is a
        pair (host, port); the host must refer to the local host. For raw packet
        sockets the address is a tuple (ifname, proto [,pkttype [,hatype]])
        """
'''将套接字绑定到本地地址。是一个IP套接字的地址对(主机、端口),主机必须参考本地主机。'''
        pass

    def close(self): # real signature unknown; restored from __doc__
        """
        close()

        Close the socket.  It cannot be used after this call.
        """
        '''关闭socket'''
        pass

    def connect(self, address): # real signature unknown; restored fr
        """
        connect(address)

        Connect the socket to a remote address.  For IP sockets, the a
        is a pair (host, port).
```

```
        """
        '''将套接字连接到远程地址。IP套接字的地址'''
        pass

    def connect_ex(self, address): # real signature unknown; restored from __doc__
        """
        connect_ex(address) -> errno

        This is like connect(address), but returns an error code (the errno value)
        instead of raising an exception when an error occurs.
        """
        pass

    def detach(self): # real signature unknown; restored from __doc__
        """
        detach()

        Close the socket object without closing the underlying file descriptor.
        The object cannot be used after this call, but the file descriptor
        can be reused for other purposes.  The file descriptor is returned.
        """
'''关闭套接字对象没有关闭底层的文件描述符。'''
        pass

    def fileno(self): # real signature unknown; restored from __doc__
        """
        fileno() -> integer

        Return the integer file descriptor of the socket.
        """
        '''返回整数的套接字的文件描述符。'''
        return 0

    def getpeername(self): # real signature unknown; restored from __doc__
        """
        getpeername() -> address info

        Return the address of the remote endpoint.  For IP sockets, th
        info is a pair (hostaddr, port).
            """
        '''返回远程端点的地址。IP套接字的地址'''
        pass

    def getsockname(self): # real signature unknown; restored from __doc__
```

```
        """
        getsockname() -> address info

        Return the address of the local endpoint.  For IP sockets, the address
        info is a pair (hostaddr, port).
        """
        '''返回远程端点的地址。IP套接字的地址'''
        pass

    def getsockopt(self, level, option, buffersize=None): # real signature unknown; restored
from __doc__
        """
        getsockopt(level, option[, buffersize]) -> value

        Get a socket option.  See the Unix manual for level and option.
        If a nonzero buffersize argument is given, the return value is a
        string of that length; otherwise it is an integer.
        """
        '''得到一个套接字选项'''
        pass

    def gettimeout(self): # real signature unknown; restored from __doc__
        """
        gettimeout() -> timeout

        Returns the timeout in seconds (float) associated with socket
        operations. A timeout of None indicates that timeouts on socket
        operations are disabled.
        """
        '''返回的超时秒数(浮动)与套接字相关联'''
        return timeout

    def ioctl(self, cmd, option): # real signature unknown; restored from __doc__
        """
        ioctl(cmd, option) -> long

        Control the socket with WSAIoctl syscall. Currently supported
        SIO_RCVALL:  'option' must be one of the socket.RCVALL_* const
        SIO_KEEPALIVE_VALS:  'option' is a tuple of (onoff, timeout,
        """
        return 0

    def listen(self, backlog=None): # real signature unknown; restored from __doc__
        """
```

```
        listen([backlog])

        Enable a server to accept connections.  If backlog is specified, it must be
        at least 0 (if it is lower, it is set to 0); it specifies the number of
        unaccepted connections that the system will allow before refusing new
        connections. If not specified, a default reasonable value is chosen.
        """
        '''使服务器能够接受连接。'''
        pass

    def recv(self, buffersize, flags=None): # real signature unknown; restored from __doc__
        """
        recv(buffersize[, flags]) -> data

        Receive up to buffersize bytes from the socket.  For the optional flags
        argument, see the Unix manual.  When no data is available, block until
        at least one byte is available or until the remote end is closed.  When
        the remote end is closed and all data is read, return the empty string.
        """
'''当没有数据可用,阻塞,直到至少一个字节是可用的或远程结束之前关闭。'''
        pass

    def recvfrom(self, buffersize, flags=None): # real signature unknown; restored from __doc__
        """
        recvfrom(buffersize[, flags]) -> (data, address info)

        Like recv(buffersize, flags) but also return the sender's address info.
        """
        pass

    def recvfrom_into(self, buffer, nbytes=None, flags=None): # real signature unknown;
restored from __doc__
        """
        recvfrom_into(buffer[, nbytes[, flags]]) -> (nbytes, address info)

        Like recv_into(buffer[, nbytes[, flags]]) but also return the sender's address info.
        """
        pass

    def recv_into(self, buffer, nbytes=None, flags=None): # real signature unknown; restored
from __doc__
        """
        recv_into(buffer, [nbytes[, flags]]) -> nbytes_read
```

```
        A version of recv() that stores its data into a buffer rather than creating
        a new string.  Receive up to buffersize bytes from the socket.  If buffersize
        is not specified (or 0), receive up to the size available in the given buffer.

        See recv() for documentation about the flags.
        """
        pass

    def send(self, data, flags=None): # real signature unknown; restored from __doc__
        """
        send(data[, flags]) -> count

        Send a data string to the socket.  For the optional flags
        argument, see the Unix manual.  Return the number of bytes
        sent; this may be less than len(data) if the network is busy.
        """
        '''发送一个数据字符串到套接字。'''
        pass

    def sendall(self, data, flags=None): # real signature unknown; restored from __doc__
        """
        sendall(data[, flags])

        Send a data string to the socket.  For the optional flags
        argument, see the Unix manual.  This calls send() repeatedly
        until all data is sent.  If an error occurs, it's impossible
        to tell how much data has been sent.
        """
        '''发送一个数据字符串到套接字，直到所有数据发送完成'''
        pass

    def sendto(self, data, flags=None, *args, **kwargs): # real signature unknown; NOTE:
unreliably restored from __doc__
        """
        sendto(data[, flags], address) -> count

        Like send(data, flags) but allows specifying the destination a
        For IP sockets, the address is a pair (hostaddr, port).
        """
        pass

    def setblocking(self, flag): # real signature unknown; restored f
        """
        setblocking(flag)
```

```
        Set the socket to blocking (flag is true) or non-blocking (false).
        setblocking(True) is equivalent to settimeout(None);
        setblocking(False) is equivalent to settimeout(0.0).
        """
'''是否阻塞（默认True），如果设置False，那么accept和recv时一旦无数据，则报错。'''
        pass

    def setsockopt(self, level, option, value): # real signature unknown; restored from __doc__
        """
        setsockopt(level, option, value)

        Set a socket option.  See the Unix manual for level and option.
        The value argument can either be an integer or a string.
        """
        pass

    def settimeout(self, timeout): # real signature unknown; restored from __doc__
        """
        settimeout(timeout)

        Set a timeout on socket operations.  'timeout' can be a float,
        giving in seconds, or None.  Setting a timeout of None disables
        the timeout feature and is equivalent to setblocking(1).
        Setting a timeout of zero is the same as setblocking(0).
        """
        pass

    def share(self, process_id): # real signature unknown; restored from __doc__
        """
        share(process_id) -> bytes

        Share the socket with another process.  The target process id
        must be provided and the resulting bytes object passed to the target
        process.  There the shared socket can be instantiated by calling
        socket.fromshare().
        """
        return b""

    def shutdown(self, flag): # real signature unknown; restored from
        """
        shutdown(flag)

        Shut down the reading side of the socket (flag == SHUT_RD), the writing side
```

```
        of the socket (flag == SHUT_WR), or both ends (flag == SHUT_RDWR).
        """
        pass

    def _accept(self): # real signature unknown; restored from __doc__
        """
        _accept() -> (integer, address info)

        Wait for an incoming connection.  Return a new socket file descriptor
        representing the connection, and the address of the client.
        For IP sockets, the address info is a pair (hostaddr, port).
        """
        pass
```

注：撸主知道大家懒，所以把全部功能的中文标记在每个功能的下面啦。下面撸主列一些经常用到的吧

sk.bind(address)

s.bind(address) 将套接字绑定到地址。address地址的格式取决于地址族。在AF_INET下，以元组（host,port）的形式表示地址。

sk.listen(backlog)

开始监听传入连接。backlog指定在拒绝连接之前，可以挂起的最大连接数量。

backlog等于5，表示内核已经接到了连接请求，但服务器还没有调用accept进行处理的连接个数最大为5

这个值不能无限大，因为要在内核中维护连接队列

sk.setblocking(bool)

是否阻塞（默认True），如果设置False，那么accept和recv时一旦无数据，则报错。

sk.accept()

接受连接并返回（conn,address），其中conn是新的套接字对象，可以用来接收和发送数据。address是连接客户端的地址。

接收TCP 客户的连接（阻塞式）等待连接的到来

sk.connect(address)

连接到address处的套接字。一般，address的格式为元组（hostname 出错，返回socket.error错误。

sk.connect_ex(address)

同上，只不过会有返回值，连接成功时返回 0 ，连接失败时候返回编码，例如：10061

sk.close()

关闭套接字

sk.recv(bufsize[,flag])

接受套接字的数据。数据以字符串形式返回，bufsize指定最多可以接收的数量。flag提供有关消息的其他信息，通常可以忽略。

sk.recvfrom(bufsize[.flag])

与recv()类似，但返回值是（data,address）。其中data是包含接收数据的字符串，address是发送数据的套接字地址。

sk.send(string[,flag])

将string中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于string的字节大小。即：可能未将指定内容全部发送。

sk.sendall(string[,flag])

将string中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。

内部通过递归调用send，将所有内容发送出去。

sk.sendto(string[,flag],address)

将数据发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地址。返回值是发送的字节数。该函数主要用于UDP协议。

sk.settimeout(timeout)

设置套接字操作的超时期，timeout是一个浮点数，单位是秒。值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如 client 连接最多等待5s）

sk.getpeername()

返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。

sk.getsockname()

返回套接字自己的地址。通常是一个元组(ipaddr,port)

sk.fileno()

套接字的文件描述符

TCP：

```
import  socketserver
```

服务端

```
class Myserver(socketserver.BaseRequestHandler):

    def handle(self):

        conn = self.request
        conn.sendall(bytes("你好，我是机器人",encoding="utf-8"))
        while True:
            ret_bytes = conn.recv(1024)
            ret_str = str(ret_bytes,encoding="utf-8")
            if ret_str == "q":
                break
            conn.sendall(bytes(ret_str+"你好我好大家好",encoding="utf-8"))

if __name__ == "__main__":
    server = socketserver.ThreadingTCPServer(("127.0.0.1",8080),Myserver)
    server.serve_forever()
```

客户端

```
import socket

obj = socket.socket()

obj.connect(("127.0.0.1",8080))

ret_bytes = obj.recv(1024)
ret_str = str(ret_bytes,encoding="utf-8")
print(ret_str)

while True:
    inp = input("你好请问您有什么问题？ \n >>>")
    if inp == "q":
        obj.sendall(bytes(inp,encoding="utf-8"))
        break
    else:
        obj.sendall(bytes(inp, encoding="utf-8"))
        ret_bytes = obj.recv(1024)
        ret_str = str(ret_bytes,encoding="utf-8")
```

```
        print(ret_str)
```

服务端

```
import socket

sk = socket.socket()

sk.bind(("127.0.0.1",8080))
sk.listen(5)

while True:
    conn,address = sk.accept()
    conn.sendall(bytes("欢迎光临我爱我家",encoding="utf-8"))

    size = conn.recv(1024)
    size_str = str(size,encoding="utf-8")
    file_size = int(size_str)

    conn.sendall(bytes("开始传送", encoding="utf-8"))

    has_size = 0
    f = open("db_new.jpg","wb")
    while True:
        if file_size == has_size:
            break
        date = conn.recv(1024)
        f.write(date)
        has_size += len(date)

    f.close()
```

客户端

```
import socket
import os

obj = socket.socket()

obj.connect(("127.0.0.1",8080))
```

```
ret_bytes = obj.recv(1024)
ret_str = str(ret_bytes,encoding="utf-8")
print(ret_str)

size = os.stat("yan.jpg").st_size
obj.sendall(bytes(str(size),encoding="utf-8"))

obj.recv(1024)

with open("yan.jpg","rb") as f:
    for line in f:
        obj.sendall(line)
```

UdP

```
import socket
ip_port = ('127.0.0.1',9999)
sk = socket.socket(socket.AF_INET,socket.SOCK_DGRAM,0)
sk.bind(ip_port)

while True:
    data = sk.recv(1024)
    print data
```

```
import socket
ip_port = ('127.0.0.1',9999)

sk = socket.socket(socket.AF_INET,socket.SOCK_DGRAM,0)
while True:
    inp = input('数据: ').strip()
    if inp == 'exit':
        break
    sk.sendto(bytes(inp,encoding = "utf-8"),ip_port)

sk.close()
```

WEB服务应用:

```
#!/usr/bin/env python
#coding:utf-8
import socket

def handle_request(client):
    buf = client.recv(1024)
    client.send("HTTP/1.1 200 OK\r\n\r\n")
    client.send("Hello, World")

def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost',8080))
    sock.listen(5)

    while True:
        connection, address = sock.accept()
        handle_request(connection)
        connection.close()

if __name__ == '__main__':
  main()
```

IO多路复用

I/O（input/output），即输入/输出端口。每个设备都会有一个专用的I/O地址，用来处理自己的输入输出信息首先什么是I/O:

I/O分为磁盘io和网络io，这里说的是网络io

IO多路复用:

I/O多路复用指：通过一种机制，可以监视多个描述符(socket)，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。

Linux

Linux中的 select，poll，epoll 都是IO多路复用的机制。

Linux下网络I/O使用socket套接字来通信，普通I/O模型只能监听一个soc 用可同时监听多个socket.

I/O多路复用避免阻塞在io上，原本为多进程或多线程来接收多个连接的消息变为单进程或单线程保

存多个socket的状态后轮询处理.

Python

Python中有一个select模块，其中提供了：select、poll、epoll三个方法，分别调用系统的select，poll，epoll 从而实现IO多路复用。

```
Windows Python:

    提供： select

Mac Python:

    提供： select

Linux Python:

    提供： select、poll、epoll
```

对于select模块操作的方法:

```
句柄列表11, 句柄列表22, 句柄列表33 = select.select(句柄序列1, 句柄序列2, 句柄序列3, 超时时间)

参数： 可接受四个参数（前三个必须）
返回值：三个列表

select方法用来监视文件句柄，如果句柄发生变化，则获取该句柄。
1、当 参数1 序列中的句柄发生可读时（accetp和read），则获取发生变化的句柄并添加到 返回值1 序列中
2、当 参数2 序列中含有句柄时，则将该序列中所有的句柄添加到 返回值2 序列中
3、当 参数3 序列中的句柄发生错误时，则将该发生错误的句柄添加到 返回值3 序列中
4、当 超时时间 未设置，则select会一直阻塞，直到监听的句柄发生变化
5、当 超时时间 = 1时，那么如果监听的句柄均无任何变化，则select会阻塞 1 秒，之后返回三个空列表，如果监听的句柄
```

利用select监听终端操作实例

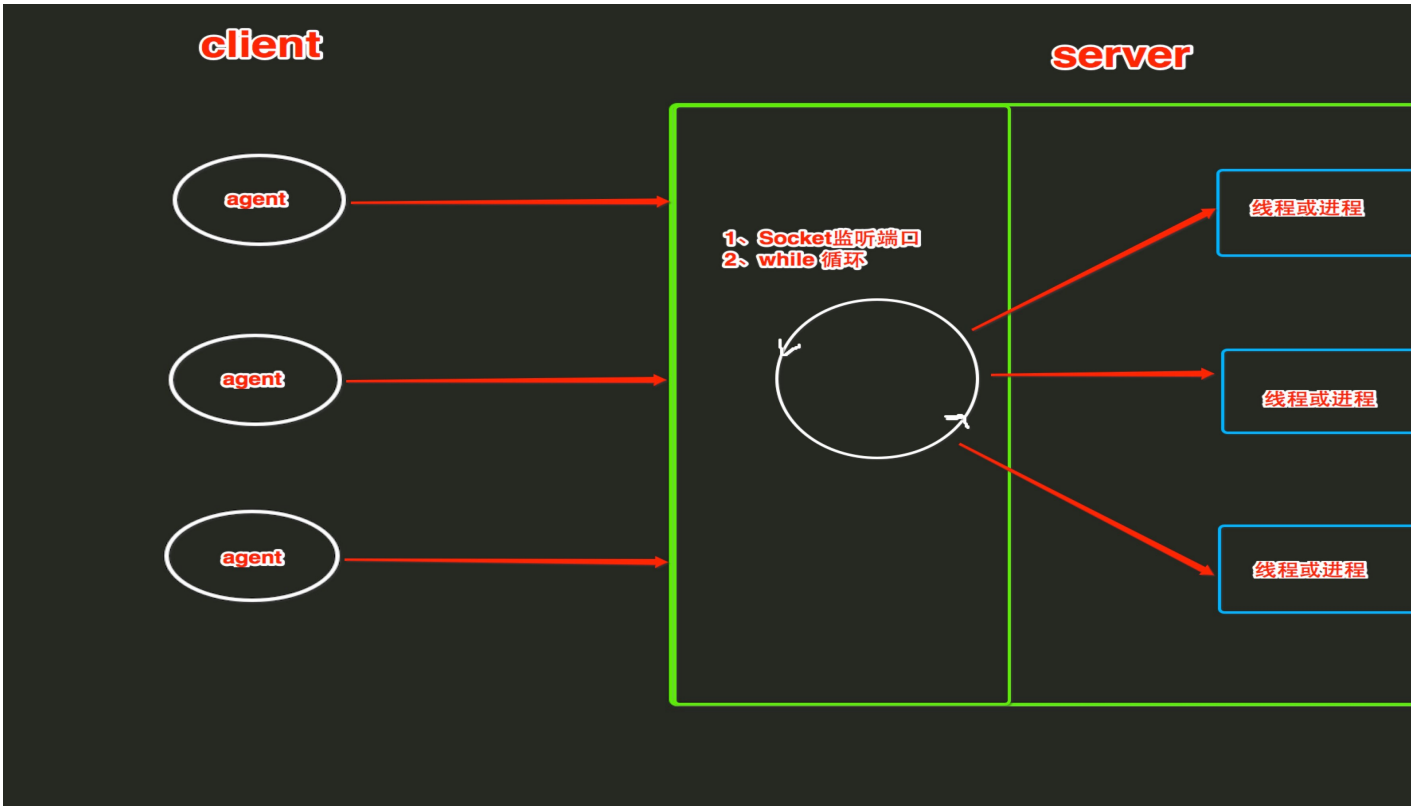
利用select实现伪同时处理多个Socket客户端请求

利用select实现伪同时处理多个Socket客户端请求读写分离

41

关注 | 顶部 | 评论

socketserver



SocketServer内部使用 IO多路复用 以及 “多线程” 和 “多进程” ，从而实现并发处理多个客户端请求的Socket服务端。即：每个客户端请求连接到服务器时，Socket服务端都会在服务器是创建一个“线程”或者“进程” 专门负责处理当前客户端的所有请求。

ThreadingTCPServer

ThreadingTCPServer实现的Soket服务器内部会为每个client创建一个 “线程”，该线程用来和客户端进行交互。

1、ThreadingTCPServer基础

使用ThreadingTCPServer：

创建一个继承自 SocketServer.BaseRequestHandler 的类

类中必须定义一个名称为 handle 的方法

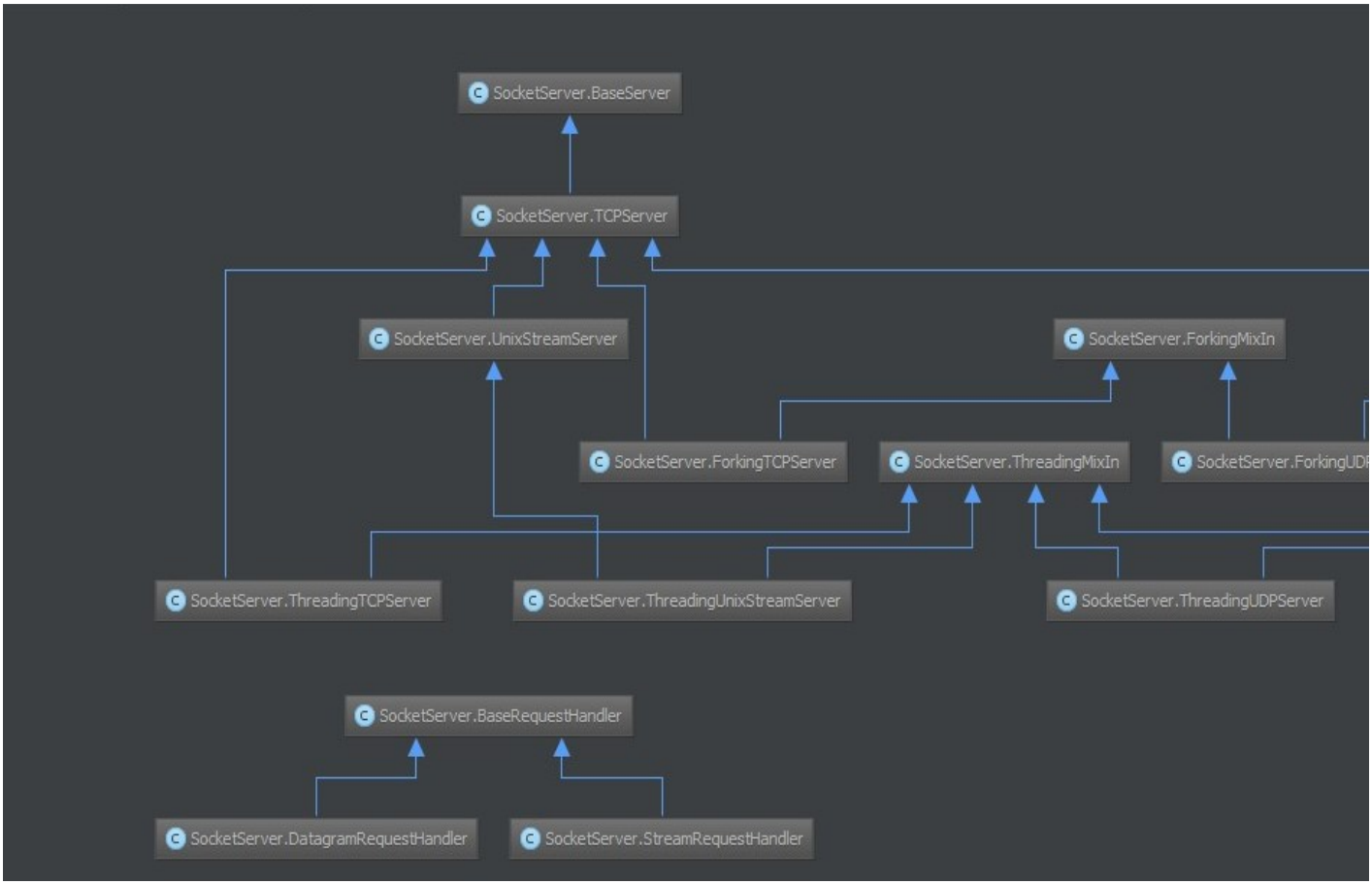
启动ThreadingTCPServer

服务端

客户端

2、ThreadingTCPServer源码剖析

ThreadingTCPServer的类图关系如下：



内部调用流程为：

启动服务端程序

执行 TCPServer.__init__ 方法，创建服务端Socket对象并绑定 IP 和 端口

执行 BaseServer.__init__ 方法，将自定义的继承自SocketServer.BaseRequestHandler 的类MyRequestHandle赋值给 self.RequestHandlerClass

执行 BaseServer.server_forever 方法，While 循环一直监听是否有客户端请求到达 ...

当客户端连接到达服务器

执行 ThreadingMixIn.process_request 方法，创建一个 “线程” 用来处理请求

执行 ThreadingMixIn.process_request_thread 方法

执行 BaseServer.finish_request 方法，执行 self.RequestHandlerClass() 即：执行 自定义 MyRequestHandler 的构造方法（自动调用基类BaseRequestHandle 的构造方法，在该构造方法中又会调用 MyRequestHandler的handle方法）

相对应的源码如下：

```
Baseserver
```

```
TCP server
```

```
ThreadingMixIn
```

```
SocketServer.BaseRequestHandler
```

SocketServer的ThreadingTCPServer之所以可以同时处理请求得益于 select 和 Threading 两个东西，其实本质上就是在服务器端为每一个客户端创建一个线程，当前线程用来处理对应客户端的请求，所以，可以支持同时n个客户端链接（长连接）。

分类: python

张岩林
关注 - 5
粉丝 - 346
+加关注

« 上一篇：python面向对象编程
» 下一篇：python进程、线程、协程

posted @ 2016-06-09 11:04 张岩林 阅读(44849) 评论(5) 编辑 收藏

评论列表

#1楼 2017-03-25 20:22 sndnvaps

文章写得不错，学习了。

支持(0) 反对(0)

#2楼 2017-05-04 10:03 23云恋49枫

hao

支持(0) 反对(0)

#3楼 2017-10-15 22:59 卡罗特

多谢!

反对(0)

#4楼 2017-12-11 20:05 娜yilianyoumeng

#5楼 2018-02-01 10:34 凯特春

```
import socket

sk = socket.socket()
sk.bind(("127.0.0.1",8080))
sk.listen(5)

conn,address = sk.accept()
sk.sendall(bytes("Hello world",encoding="utf-8"))


这个有个错误的地方:
sk.sendall应该换成conn.sendall
```

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· 三星S9/S9+电池百分百确认：照抄S8 毫无提升
· A站已经无法打开！动荡不已的"猴山"再度来到至暗时刻
· 12306为黄牛推出慢速排队机制
· 苹果/谷歌/亚马逊总市值超2万亿美元 在全球经济中支配力日增
· 年费19.99美元！任天堂宣布9月推出Switch在线服务
» 更多新闻...

最新知识库文章:
· 领域驱动设计在互联网业务开发中的实践
· 步入云计算
· 以操作系统的角度述说线程与进程
· 软件测试转型之路
· 门内门外看招聘
» 更多知识库文章...

公告

有[illegible]

昵称：张岩林
园龄：1年9个月
粉丝：346
关注：5
+加关注

<		2018年2月				>
日	**一**	**二**	**三**	**四**	**五**	**六**
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	1	2	3

41

关注 | 顶部 | 评论

4 5 6 7 8 9 10

搜索

找找看

必应搜索

打开标签

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
快速链接

随笔分类

ajax(1)
Go语言(2)
mysql(3)
python(20)
前端技术(6)
项目实战篇(2)

随笔档案

2016年12月 (3)
2016年11月 (1)
2016年10月 (2)
2016年9月 (3)
2016年8月 (7)
2016年7月 (3)
2016年6月 (6)
2016年5月 (8)
2016年4月 (4)

文章分类

Django(1)

积分与排名

积分 - 69036
排名 - 4886

最新评论

1. Re:python之socket编程

import socketsk = socket.socket()sk.bind(("127.0.0.1",8080))sk.listen(5)conn,address =

sk.accept()sk.....

--凯特春

2. Re:python正则表达式

可以分享几篇爬虫相关的知识吗？？？

--枫飞飞

3. Re:MYSQL(一)

谢谢分享!

--孤云落日-夕阳情

4. Re:python之socket编程

不错

--娜yilianyoumeng

5. Re:ajax应用篇

路漫漫其修远兮，吾将上下而求索，不错

--小通

6. Re:ajax应用篇

初出茅庐的菜鸟路过，表示看不懂，楼主，有详细的实例没（各个部分），感谢

--小通

7. Re:HTML基础做出属于自己的完美网页

@TDX借鉴一位大神的...

--张岩林

8. Re:玩转Jquery，告别前端知道思路忘记知识点的痛苦

学习了
Jquery插件集

--mangju59907

9. Re:HTML基础做出属于自己的完美网页

写的很详细

--find007

10. Re:python之socket编程

多谢!

--卡罗特

11. Re:HTTP图解

谢谢~

--quickli

12. Re:玩转Jquery，告别前端知道思路忘记知识点的痛苦

谢谢分享，受益

--蜚IT人

13. Re:玩转Jquery，告别前端知道思路忘记知识点的痛苦

多谢！对初学者来说太棒了!

--哲子兄

14. Re:HTML基础做出属于自己的完美网页

楼主，实力非凡，谢谢分享!
楼主的博客样式很漂亮，请问下博客的模板是自己设计的还是博客园提供的?

--MXTA

15. Re:python之socket编程

hao

--23云恋49枫

16. Re:Python Set集合，函数，深入拷贝，浅入拷贝,文件处理

元组的深拷贝，要看所含元素是否有可变类型对象。

--Eliefly

17. Re:python之socket编程

文章写得不错，学习了。

--sndnvaps

18. Re:MYSQL(一)

很不错

--追阳

19. Re:HTTP图解（大牛必经之路）

前端是要了解http协议 才算是入了门。

--.伊泽瑞尔

20. Re:HTTP图解（大牛必经之路）

tomcat服务器中 set-cookie 会放入session信息

--程序人生0407

21. Re:HTTP图解（大牛必经之路）

一直想找此书的pdf，感谢分享!

--FarmerQing

22. Re:MYSQL(二)

mark

--黄金时代1.0

23. Re:python模拟登陆之下载

始终不会post 账号密码登录 哎

--archie’s

24. Re:HTTP图解（大牛必经之路）

请问传图片的时候，是用Base64还是二进制格式传图片?

谢谢

--We_Are_Friends!

25. Re:HTTP图解（大牛必经之路）

还是有点迷糊

--孔三胖

26. Re:玩转Jquery，告别前端知道思路忘记知识点的痛苦

收藏

--dgdyq

27. Re:HTTP图解（大牛必经之路）

几张图很形象

--雨知

28. Re:HTTP图解（大牛必经之路）

学习了

--java_lover

29. Re:HTTP图解（大牛必经之路）

好文学习了。

--tt_Vincen

30. Re:HTTP图解（大牛必经之路）

厉害了

--music180

31. Re:HTTP图解（大牛必经之路）

好文，顶一个，勾起了上学的回忆

--Double_K

32. Re:HTTP图解（大牛必经之路）

图解好有趣

--Zoctopus

33. Re:HTTP图解（大牛必经之路）

看过这本书

--朝向远方

34. Re:HTTP图解（大牛必经之路）

@张岩林似乎有好几处.....

--SzeCheng

35. Re:HTTP图解（大牛必经之路）

隐藏目录按钮单击没用啊

--ycyzharry

36. Re:HTTP图解（大牛必经之路）

@SzeCheng老脸一红，当没看见，...

--张岩林

37. Re:HTTP图解（大牛必经之路）

Neo荣

错别字吧

--SzeCheng

38. Re:HTTP图解（大牛必经之路）

@孙长宇自己的总结，是摘抄，原书我已经标明...

--张岩林

39. Re:HTTP图解（大牛必经之路）

博主您确定您的内容不是从这本书中摘抄

的?

--孙长宇

40. Re:HTTP图解（大牛必经之路）

@普通的地球人这就尴尬了，哈哈...

--张岩林

阅读排行榜

1. python之socket编程(44844)
2. ajax应用篇(9998)
3. django多条件筛选搜索(项目实例)(9447)
4. HTTP图解(9124)
5. MYSQL(一)(7719)

评论排行榜

1. HTTP图解(46)
2. ajax应用篇(36)
3. MYSQL(一)(24)
4. HTML基础做出属于自己的完美网页(13)
5. MYSQL(二)(10)

推荐排行榜

1. ajax应用篇(285)
2. MYSQL(一)(157)
3. HTTP图解(119)
4. MYSQL(二)(81)
5. MYSQL(三)(41)