

昵称： 又见阿郎  
园龄： 4年2个月  
粉丝： 35  
关注： 60  
+加关注

< 2020年8月 >						
日	一	二	三	四	五	六
26	27	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	29
30	31	1	2	3	4	5

搜索

找找看

谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

我的标签

- C#(13)
- Python(12)
- abp(11)
- .net core(9)
- DI(8)
- redis(6)
- 并发(6)
- 设计模式(6)
- 线程安全(6)
- EF Core(5)
- 更多

python select.select模块通信全过程详解

要理解select.select模块其实主要就是要理解它的参数, 以及其三个返回值。  
select()方法接收并监控3个通信列表, 第一个是所有的输入的数据,就是指外部发过来的数据, 第2个是监控和接收所有要发出去的数据(outgoing data),第3个监控错误信息

在网上一一直在找这个select.select的参数解释, 但实在是没有, 哎...自己硬着头皮分析了一下。  
readable, writable, exceptional = select.select(inputs, outputs, inputs)

select 函数的参数其实很好理解, 前提是我们对unix 网络编程有了解. select 模型是unix 系统中的网络模型, python 将其封装了,因此我们使用起来就比较方便, 但是面试官就不会这么觉得了(最近被面试逼疯了, 考虑问题都从面试官的角度考虑), 先说下unix 系统中的select 模型吧, 参数原型:  
int select(int maxfdpl, fd\_set \* readset, fd\_set \*writset, fd\_set \*exceptset, const struct timeval \* tiomeout)

- 第一个是最大的文件描述符长度
- 第二个是监听的可读集合
- 第三个是监听的可写集合
- 第四个是监听的异常集合
- 第五个是时间限制

对struct fd\_set结构体操作的宏  
FD\_SETSIZE 容量, 指定fd\_array数组大小, 默认为64, 也可自己修改宏  
FD\_ZERO(\*set) 置空, 使数组的元素值都为3435973836, 元素个数为0.  
FD\_SET(s, \*set) 添加, 向 struct fd\_set结构体添加套接字s  
FD\_ISSET(s, \*set) 判断, 判断s是否为 struct fd\_set结构体中的一员  
FD\_CLR(s, \*set) 删除, 从 struct fd\_set结构体中删除成员s

因为此模型主要是在网络中应用, 我们不考虑文件, 设备, 单从套接字来考虑, 可读条件如下:  
a) 该套接字接收缓冲区中的数据字节数大于等于套接字接收缓冲区低水位标记的当前大小. 对这样的套接字执行读操作不会阻塞并将返回一个大于0的值 (也就是返回准备好读入的数据)。我们可以使用SO\_RCVLOWAT套接字选项设置该套接字的低水位标记。对于TCP和UDP套接字而言，其默认值为1。  
b) 该连接的读半部关闭 (也就是接收了FIN的TCP连接)。对这样的套接字的读操作将不阻塞并返回0 (也就是返回EOF)。  
c) 该套接字是一个监听套接字且已完成的连接数不为0。对这样的套接字的accept通常不会阻塞，不过我们将在15.6节讲解accept可能阻塞的一种时序条件。  
d) 其上有一个套接字错误待处理。对这样的套接字的读操作将不阻塞并返回-1 (也就是返回一个错误)，同时把errno设置成确切的错误条件。这些待处理错误 (pending error) 也可以通过指定SO\_ERROR套接字选项调用getsockopt获取并清除。

可写条件如下:  
a) 该套接字发送缓冲区中的可用空间字节数大于等于套接字发送缓冲区低水位标记的当前大小, 并且或者该套接字已连接, 或者该套接字不需要连接 (如UDP套接字)。这意味着如果我们把这样的套接字设置成非阻塞 (第16章), 写操作将不阻塞并返回一个正值 (例如由传输层接受的字节数)。我们可以使用SO\_SNDLOWAT套接字选项来设置该套接字的低水位标记。对于TCP和UDP套接字而言，其默认值通常为2048。  
b) 该连接的写半部关闭。对这样的套接字的写操作将产生SIGPIPE信号 (5.12节)。  
c) 使用非阻塞式connect的套接字已建立连接, 或者connect已经以失败告终。  
d) 其上有一个套接字错误待处理。对这样的套接字的写操作将不阻塞并返回-1 (也就是返回一个错误)，同时把errno设置成确切的错误条件。这些待处理的错误也可以通过指定SO\_ERROR套接字选项调用getsockopt获取并清除。

我看C 示例的时候, 看的有点懵逼, 应该需要跑一遍代码就好, python 就简单了, 直接调用封装好的select , 其底层处理好了文件描述符的相关读写监听(回头再研究下), 我们在Python 中只需这么写:  
can\_read, can\_write, \_ = select.select(inputs, outputs, None, None)

第一个参数是我们需要监听可读的套接字, 第二个参数是我们需要监听可写的套接字, 第三个参数使我们需要监听异常的套接字, 第四个则是时间限制设置。


如果监听的套接字满足了可读可写条件, 那么所返回的can\_read 或是 can\_write就会有值了, 然后我们就可以利用这些返回值进行随后的操作了。相比较unix 的select模型, 其select函数的返回值是

随笔档案
2020年8月(1)
2020年5月(1)
2020年4月(2)
2020年3月(1)
2019年12月(7)
2019年11月(1)
2019年10月(1)
2019年7月(4)
2019年6月(4)
2019年5月(4)
2019年4月(5)
2019年3月(2)
2019年2月(3)
2019年1月(3)
2018年12月(1)
2018年11月(2)
2018年10月(1)
2018年9月(1)
2018年8月(4)
2018年7月(16)
2018年6月(5)
2018年5月(8)
2018年4月(14)
2018年3月(5)
2018年2月(4)
2018年1月(8)
2017年12月(7)
2017年11月(7)
2017年10月(9)
2017年9月(9)
2017年8月(3)

一个整型, 用以判断是否执行成功.

第一个参数就是服务器端的socket, 第二个是我们在运行过程中存储的客户端的socket, 第三个存储错误信息。  
重点是在返回值, 第一个返回的是可读的list, 第二个存储的是可写的list, 第三个存储的是错误信息的list。  
网上所有关于select.select的代码都是差不多的, 但是有些不能运行, 或是不全。我自己重新写了一份能运行的程序, 做了很多注释, 好好看看就能搞懂

服务器端:



```
# coding: utf-8
import select
import socket
import Queue
from time import sleep

# Create a TCP/IP
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(False)

# Bind the socket to the port
server_address = ('localhost', 8090)
print ('starting up on %s port %s' % server_address)
server.bind(server_address)

# Listen for incoming connections
server.listen(5)

# Sockets from which we expect to read
inputs = [server]

# Sockets to which we expect to write
# 处理要发送的消息
outputs = []

# Outgoing message queues (socket: Queue)
message_queues = {}

while inputs:
    # Wait for at least one of the sockets to be ready for processing
    print ('waiting for the next event')
    # 开始select 监听, 对input_list 中的服务器端server 进行监听
    # 一旦调用socket的send, recv函数, 将会再次调用此模块
    readable, writable, exceptional = select.select(inputs, outputs, inputs)

    # Handle inputs
    # 循环判断是否有客户端连接进来, 当有客户端连接进来时select 将触发
    for s in readable:
        # 判断当前触发的是不是服务端对象, 当触发的对象是服务端对象时,说明有新客户端连接进来了
        # 表示有新用户来连接
        if s is server:
            # A "readable" socket is ready to accept a connection
            connection, client_address = s.accept()
            print ('connection from', client_address)
            # this is connection not server
            connection.setblocking(0)
            # 将客户端对象也加入到监听的列表中, 当客户端发送消息时 select 将触发
            inputs.append(connection)

            # Give the connection a queue for data we want to send
            # 为连接的客户端单独创建一个消息队列, 用来保存客户端发送的消息
            message_queues[connection] = Queue.Queue()
        else:
            # 有老用户发消息, 处理接受
            # 由于客户端连接进来时服务端接收客户端连接请求, 将客户端加入到了监听列表中(input_list)
            # 所以判断是否是客户端对象触发
            data = s.recv(1024)
            # 客户端未断开
            if data != '':
                # A readable client socket has data
                print ('received "%s" from %s' % (data, s.getpeername()))
                # 将收到的消息放入到相对应的socket客户端的消息队列中
                message_queues[s].put(data)
```

2017年7月(8)
2017年6月(8)
2017年5月(2)

最新评论
1. Re:你真的知道.NET Framework中的阻塞队列BlockingCollection的妙用吗?
这个讲的比较详细
--JWCJW
2. Re:easynetq发布订阅demo实现注意事项
你这个obj是哪里的啊
--初来乍到，多多指导
3. Re:DispatchProxy实现动态代理及AOP
啥原理 测试过性能吗
--Gamain
4. Re:asp.net core上使用Redis探索(2)
博主，请问这个怎么做条件查询呢?
--湫湫秋
5. Re:我应该跟libuv说声对不起，我错怪了libuv(转)
@ 溪源More看你说的话还以为是几年前的人发的，都9102年了，大把知识付费的，君不见而已...
--铖览IT

阅读排行榜
1. python snownlp情感分析简易demo (15207)
2. windows下Python 3.x图形图像处理库PIL的安装(13128)
3. 我应该跟libuv说声对不起，我错怪了libuv(转)(12640)
4. C# 中的线程安全集合类(12078)
5. python select.select模块通信全过程详解(9219)

```
# Add output channel for response
# 将需要进行回复操作socket放到output 列表中，让select监听
if s not in outputs:
    outputs.append(s)
else:
    # 客户端断开了连接，将客户端的监听从input列表中移除
    # Interpret empty result as closed connection
    print ('closing', client_address)
    # Stop listening for input on the connection
    if s in outputs:
        outputs.remove(s)
    inputs.remove(s)
    s.close()

# Remove message queue
# 移除对应socket客户端对象的消息队列
del message_queues[s]

# Handle outputs
# 如果现在没有客户端请求，也没有客户端发送消息时，开始对发送消息列表进行处理，是否需要发送消息
# 存储哪个客户端发送过消息
for s in writable:
    try:
        # 如果消息队列中有消息，从消息队列中获取要发送的消息
        message_queue = message_queues.get(s)
        send_data = ''
        if message_queue is not None:
            send_data = message_queue.get_nowait()
        else:
            # 客户端连接断开了
            print "has closed "
    except Queue.Empty:
        # 客户端连接断开了
        print "%s" % (s.getpeername())
        outputs.remove(s)
    else:
        # print "sending %s to %s " % (send_data, s.getpeername())
        # print "send something"
        if message_queue is not None:
            s.send(send_data)
        else:
            print "has closed "
        # del message_queues[s]
        # writable.remove(s)
        # print "Client %s disconnected" % (client_address)

# # Handle "exceptional conditions"
# 处理异常的情况
for s in exceptional:
    print ('exception condition on', s.getpeername())
    # Stop listening for input on the connection
    inputs.remove(s)
    if s in outputs:
        outputs.remove(s)
    s.close()

# Remove message queue
del message_queues[s]

sleep(1)
```

客户端:

```
# coding: utf-8
import socket

messages = ['This is the message ', 'It will be sent ', 'in parts ', ]

server_address = ('localhost', 8090)

# Create aTCP/IP socket
```

评论排行榜

1. 我应该跟libuv说声对不起，我错怪了libuv(转)(5)

2. 项目开发中的一些注意事项以及技巧总结(3)

3. asp.net 分布式探讨之Session共享问题(3)

4. asp.net core Session的测试使用心得及注意事项(3)

5. asp.net core 上使用redis探索(3)--redis示例demo(3)

推荐排行榜

1. asp.net 下的中文分词检索工具 - jieba.net(4)

2. 我应该跟libuv说声对不起，我错怪了libuv(转)(3)

3. 一个你不能错过的第三方.net集合库(3)

4. 从零开始搭建基于CEFGlue的CB/S的winform项目(2)

5. 你真的知道.NET Framework中的阻塞队列BlockingCollection的妙用吗？(1)

```
socks = [socket.socket(socket.AF_INET, socket.SOCK_STREAM), socket.socket(socket.AF_INET, socket.SOCK_STREAM)]

# Connect the socket to the port where the server is listening

print ('connecting to %s port %s' % server_address)
# 连接到服务器
for s in socks:
    s.connect(server_address)

for index, message in enumerate(messages):
    # Send messages on both sockets
    for s in socks:
        print ('%s: sending "%s"' % (s.getsockname(), message + str(index)))
        s.send(bytes(message + str(index)).decode('utf-8'))
    # Read responses on both sockets

for s in socks:
    data = s.recv(1024)
    print ('%s: received "%s"' % (s.getsockname(), data))
    if data != "":
        print ('closing socket', s.getsockname())
        s.close()
```

写代码过程中遇到了两个问题，一是如何判断客户端已经关闭了socket连接，后来自己分析了下，如果关闭了客户端socket，那么此时服务器端接收到的data就是"，加个这个判断。二是如果服务器端关闭了socket，一旦在调用socket的相关方法都会报错，不管socket是不是用不同的容器存储的(意思是说list\_1存储了socket1，list\_2存储了socket1，我关闭了socket1，两者都不能在调用这个socket了)

服务器端：

```
问题 输出 调试控制台 终端
starting up on localhost port 8090
waiting for the next event
('connection from', ('127.0.0.1', 10719))
waiting for the next event
('connection from', ('127.0.0.1', 10720))
received "This is the message 0It will be sent 1in parts 2" from ('127.0.0.1', 10719)
waiting for the next event
received "This is the message 0It will be sent 1in parts 2" from ('127.0.0.1', 10720)
waiting for the next event
('closing', ('127.0.0.1', 10720))
has closed
has closed
waiting for the next event
('closing', ('127.0.0.1', 10720))
has closed
has closed
waiting for the next event
```

客户端：

```
PS E:\python Program\web> python .\select_client.py
connecting to localhost port 8090
('127.0.0.1', 10719): sending "This is the message 0"
('127.0.0.1', 10720): sending "This is the message 0"
('127.0.0.1', 10719): sending "It will be sent 1"
('127.0.0.1', 10720): sending "It will be sent 1"
('127.0.0.1', 10719): sending "in parts 2"
('127.0.0.1', 10720): sending "in parts 2"
('127.0.0.1', 10719): received "This is the message 0It will be sent 1in parts 2"
('closingsocket', ('127.0.0.1', 10719))
('127.0.0.1', 10720): received "This is the message 0It will be sent 1in parts 2"
('closingsocket', ('127.0.0.1', 10720))
```

标签: [python socket](#), [python select](#)

好文要顶 关注我 收藏该文





又见阿郎

关注 - 60

粉丝 - 35

+加关注

0 0

« 上一篇: [深入浅出Web服务器与python应用程序之间的联系](#)  
» 下一篇: [python socketserver框架解析](#)

发表评论

编辑

预览

B

支持 Markdown

提交评论

退出

订阅评论

[Ctrl+Enter快捷键提交]