[](tencent://AddContact/?fromId=50&fromSubId=1&subcmd=all&uin=3440037784)[](https://jq.qq.com/?_wv=1027&k=50Xp8mW)

扫码获取更多课程资料，开通直播上课权限（点击下方课程直播即可免费观看）

 

学神IT教育微信公众号 学神IT官方 QQ群

关注学神公众号有机会获得Python官方书籍

[](https://jq.qq.com/?_wv=1027&k=50Xp8mW)

# 第十九章 并发服务器

本节所讲内容：

19.1 并发服务器socket编程

19.2 select版-TCP服务器

19.3 协程-TCP

## 19.1 并发服务器socket编程

### 19.1.1 完成一个简单的TCP服务器

from socket import \*  
serSocket = socket(AF\_INET,SOCK\_STREAM)

#设置可以重复使用绑定的信息  
serSocket.setsockopt(SOL\_SOCKET,SO\_REUSEADDR,1)  
localAddr = ('',7788)  
serSocket.bind(localAddr)  
serSocket.listen(5)  
while True:  
 print('----主进程等待新客户端的到来----')  
 newSocket,destAddr = serSocket.accept()  
 print('----主进程接下来负责处理[%s]----'%str(destAddr))  
 try:  
 while True:  
 recv\_data = newSocket.recv(1024)  
 if recv\_data:  
 print('recv:',recv\_data.decode())  
 else:  
 print('[%s]客户端已经关闭'%str(destAddr))  
 break  
 except Exception as e:  
 print(e)  
 finally:  
 newSocket.close()  
serSocket.close()

客户端：

import socket  
def work(i):  
 sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  
 sock.connect(('127.0.0.1',7788))  
 sock.send('我是client'.encode())  
 sock.close()  
for i in range(5):  
 work(i)

总结  
同⼀时刻只能为⼀个客户进行服务， 不能同时为多个客户服务，类似于找⼀个“明星”签字⼀样， 客户需要耐⼼等待才可以获取到服务。



当服务器为⼀个客户端服务时， ⽽另外的客户端发起了connect， 只要服务器listen的队列有空闲的位置， 就会为这个新客户端进行连接， 并且客户端可以发送数据， 但当服务器为这个新客户端服务时， 可能⼀次性把所有数据接收完毕当recv接收数据时， 返回值为空， 即没有返回数据， 那么意味着客户端已经调⽤了close关闭了；

因此服务器通过判断recv接收数据是否为空 来判断客户端是否已经下线 。

So\_reuseraddr说明：

**SO\_REUSEADDR允许启动一个监听服务器并捆绑其众所周知端口，即使以前建立的将此端口用做他们的本地端口的连接仍存在。这通常是重启监听服务器时出现，若不设置此选项，则bind时将出错。**

**SO\_REUSEADDR允许在同一端口上启动同一服务器的多个实例，只要每个实例捆绑一个不同的本地IP地址即可。对于TCP，我们根本不可能启动捆绑相同IP地址和相同端口号的多个服务器。**

### 19.1.2 多进程服务器

from socket import \*  
from multiprocessing import \*  
from time import sleep  
*# 处理客户端的请求并为其服务*def dealWithClient(newSocket,destAddr):  
 while True:  
 recvData = newSocket.recv(1024)  
 if recvData:  
 print('recv[%s]:%s'%(str(destAddr), recvData.decode()))  
 else:  
 print('[%s]客户端已经关闭'%str(destAddr))  
 break  
 newSocket.close()  
def main():  
 serSocket = socket(AF\_INET, SOCK\_STREAM)  
 serSocket.setsockopt(SOL\_SOCKET, SO\_REUSEADDR , 1)  
 localAddr = ('', 7788)  
 serSocket.bind(localAddr)  
 serSocket.listen(5)  
 try:  
 while True:  
 print('-----主进程，，等待新客户端的到来------')  
 newSocket,destAddr = serSocket.accept()  
 print('-----主进程，，接下来创建一个新的进程负责数据处理[%s]-----'%str(destAddr))  
 client = Process(target=dealWithClient, args=(newSocket,destAddr))  
 client.start()  
 *#因为已经向子进程中copy了一份（引用），并且父进程中这个套接字也没有用处了  
 #所以关闭* newSocket.close()  
 finally:  
 *#当为所有的客户端服务完之后再进行关闭，表示不再接收新的客户端的链接* serSocket.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

通过为每个客户端创建一个进程的方式，能够同时为多个客户端进行服务

当客户端不是特别多的时候，这种方式还行，如果有几百上千个，就不可取了，因为每次创建进程等过程需要好较大的资源

### 19.1.3 多线程服务器

from socket import \*  
from threading import Thread  
from time import sleep  
*# 处理客户端的请求并执行事情*def dealWithClient(newSocket,destAddr):  
 while True:  
 recvData = newSocket.recv(1024)  
 if recvData:  
 print('recv[%s]:%s'%(str(destAddr), recvData.decode()))  
 else:  
 print('[%s]客户端已经关闭'%str(destAddr))  
 break  
 newSocket.close()  
def main():  
 serSocket = socket(AF\_INET, SOCK\_STREAM)  
 serSocket.setsockopt(SOL\_SOCKET, SO\_REUSEADDR , 1)  
 localAddr = ('', 7788)  
 serSocket.bind(localAddr)  
 serSocket.listen(5)  
 try:  
 while True:  
 print('-----主进程，，等待新客户端的到来------')  
 newSocket,destAddr = serSocket.accept()  
 print('-----主进程，，接下来创建一个新的进程负责数据处理[%s]-----'%str(destAddr))  
 client = Thread(target=dealWithClient, args=(newSocket,destAddr))  
 client.start()  
 *#因为线程中共享这个套接字，如果关闭了会导致这个套接字不可用，  
 #但是此时在线程中这个套接字可能还在收数据，因此不能关闭  
 #newSocket.close()* finally:  
 serSocket.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

### 19.1.4 单进程服务器-非阻塞

服务端

from socket import \*  
import time  
*# 用来存储所有的新链接的socket*g\_socketList = []  
def main():  
 serSocket = socket(AF\_INET, SOCK\_STREAM)  
 serSocket.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)  
 localAddr = ('127.0.0.1', 8080)  
 serSocket.bind(localAddr)  
 *# 可以适当修改listen中的值来看看不同的现象* serSocket.listen(1000)  
 *# 将套接字设置为非堵塞  
 # 设置为非堵塞后，如果accept时，恰巧没有客户端connect，那么accept会  
 # 产生一个异常，所以需要try来进行处理* serSocket.setblocking(False)  
 while True:  
 *# 用来测试  
 # time.sleep(0.5)* try:  
 newClientInfo = serSocket.accept()  
 except Exception as result:  
 pass  
 else:  
 print("一个新的客户端到来:%s" % str(newClientInfo))  
 newClientInfo[0].setblocking(False)  
 g\_socketList.append(newClientInfo)  
 *# 用来存储需要删除的客户端信息* needDelClientInfoList = []  
 for clientSocket, clientAddr in g\_socketList:  
 try:  
 recvData = clientSocket.recv(1024)  
 if len(recvData) > 0:  
 print('recv[%s]:%s' % (str(clientAddr), recvData))  
 else:  
 print('[%s]客户端已经关闭' % str(clientAddr))  
 clientSocket.close()  
 needDelClientInfoList.append((clientSocket, clientAddr))  
 except Exception as result:  
 pass  
 for needDelClientInfo in needDelClientInfoList:  
 g\_socketList.remove(needDelClientInfo)  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

客户端

from socket import \*  
import random  
import time  
  
serverIp = input("请输入服务器的ip:")  
connNum = input("请输入要链接服务器的次数(例如1000):")  
g\_socketList = []  
for i in range(int(connNum)):  
 s = socket(AF\_INET, SOCK\_STREAM)  
 s.connect((serverIp, 8080))  
 g\_socketList.append(s)  
 print(i)  
  
while True:  
 for s in g\_socketList:  
 s.send(bytes(random.randint(0, 100)))

## 19.2 select版-TCP服务器

### 19.2.1 select原理

在多路复⽤的模型中， 比较常⽤的有select模型和epoll模型。 这两个都是系统接及接， 由操作系统提供。 当然， Python的select模块进行了更高级的封装。

⽹络通信被Unix系统抽象为⽂件的读写， 通常是⼀个设备， 由设备驱动程序提供， 驱动可以知道⾃身的数据是否可⽤。 ⽀持阻塞操作的设备驱动通常会实现⼀组⾃身的等待队列， 如读/写等待队列⽤于⽀持上层(用户层)所需的block或non-block操作。

设备的文件的资源如果可⽤（可读或者可写） 则会通知进程， 反之则会让进程睡眠， 等到数据到来可⽤的时候， 再唤醒进程。这些设备的⽂件描述符被放在⼀个数组中， 然后select调⽤的时候遍历这个数组， 如果对于的⽂件描述符可读则会返回改⽂件描述符。

当遍历结束之后，如果仍然没有⼀个可用设备⽂件描述符， select让⽤户进程则会睡眠， 直到等待资源可⽤的时候在唤醒， 遍历之前那个监视的数组。 每次遍历都是依次进⾏判断的。

### 19.2.2 select回显服务器

from socket import \*  
from select import select  
import sys  
def main():  
 serSocket = socket(AF\_INET, SOCK\_STREAM)  
 localAddr = ('127.0.0.1', 7788)  
 serSocket.bind(localAddr)

#设置服务器为非阻塞式！  
 serSocket.setblocking(False)  
 serSocket.listen(100)  
 inputs = [serSocket]  
 running = True  
 while True:  
 readable, writable, exceptionable = select(inputs, [], [])  
 for sock in readable:  
 if sock == serSocket:  
 clientSocket, clientAddr = serSocket.accept()  
 print('newClient[%s]'%str(clientAddr))  
 inputs.append(clientSocket)  
 elif sock == sys.stdin:  
 cmd = sys.stdin.readline()  
 running = False  
 break  
 else:  
 massage = sock.recv(1024)  
 if massage:  
 print('massage from [%s] is %s'%(str(clientAddr), massage.decode('utf-8')))  
 else:  
 print('[%s] was closed'%(str(clientAddr)))  
 inputs.remove(sock)  
 sock.close()  
 if not running:  
 break  
 serSocket.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

### 19.2.3 总结

### 优点：select⽬前几乎在所有的平台上支持， 其良好跨平台⽀持也是它的⼀个优点。 缺点：select的⼀个缺点在于单个进程能够监视的文件描述符的数量存在最大限制。

## 19.3 协程服务搭建

19.3.1 协程TCP

上一章我们已经讲解了，greenlet实现了协程， 但是这个还的人工切换，所以在这里我们使用gevent这个模块。

gevent其原理是当⼀个greenlet遇到IO(指的是input output 输⼊输出，比如网络、文件操作等操作时，就自动切换到其他的greenlet， 等到IO操作完成， 再在适当的时候切换回来继续执行。

由于IO操作⾮常耗时， 经常使程序处于等待状态， 有了gevent为我们自动切换协程， 就保证总有greenlet在运行， 而不是等待IO

gevent版-TCP服务器

import sys  
import time  
import gevent  
from gevent import socket,monkey  
monkey.patch\_all()  
def handle\_request(conn):  
 while True:  
 data = conn.recv(1024)  
 if not data:  
 conn.close()  
 break  
 print("recv",data)  
def server(port):  
 s = socket.socket()  
 s.bind(('127.0.0.1',port))  
 s.listen(5)  
 while True:  
 cli,addr = s.accept()  
 gevent.spawn(handle\_request,cli)  
if \_\_name\_\_ == '\_\_main\_\_':  
 server(8080)

总结：

19.1 并发服务器socket编程

19.2 select版-TCP服务器

19.3 协程-TCP