TCP编程

socket编程

Python中提供了socket标准库，非常底层的接口库。 Socket是一种通用的网络编程接口，和网络层次没有一一对应的关系。

协议族 AF表示Address Family，用于socket()第一个参数

名称	含义
AF_INET	IPV4
AF_INET6	IPV6
AF_UNIX	Unix Domain Socket，windows没有

Socket类型

名称	含义
SOCK_STREAM	面向连接的流套接字。默认值，TCP协议
SOCK_DGRAM	无连接的数据报文套接字。UDP协议

TCP协议是流协议，也就是一大段数据看做字节流，一段段持续发送这些字节。

UDP协议是数据报协议，每一份数据封在一个单独的数据报中，一份一份发送数据。

CS编程

Socket编程，是完成一端和另一端通信的，注意一般来说这两端分别处在不同的进程中，也就是说网络通信是一个进程发消息到另外一个进程。

我们写代码的时候，每一个socket对象只表示了其中的一端。

从业务角度来说，这两端从角色上分为：

- 主动发送请求的一端，称为客户端Client
- 被动接受请求并回应的一端，称为服务端Server

这种编程模式也称为**C/S编程。**

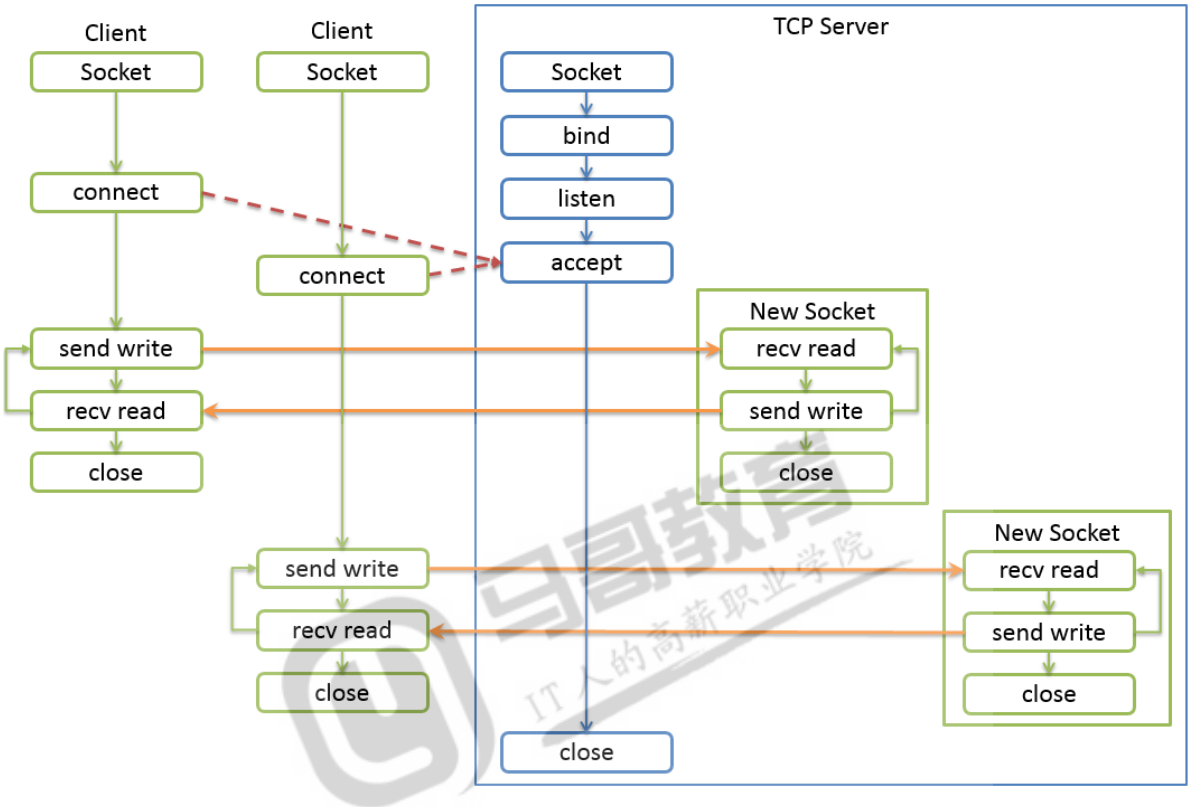
TCP服务端编程

服务器端编程步骤

- 创建Socket对象
- 绑定IP地址Address和端口Port。bind()方法 IPv4地址为一个二元组('IP地址字符串', Port)
- 开始监听，将在指定的IP的端口上监听
    - listen([backlog])方法。未完成连接队列和完成连接队列长度不能超过backlog，如果accept不拿走就满了，就会直接拒绝连接请求。backlog可以不写，默认为5

- 获取用于传送数据的**新的**Socket对象 socket.accept() -> (socket object, address info) accept方法阻塞等待客户端建立连接，返回一个新的Socket对象和客户端地址的二元组 地址是远程客户端的地址，IPv4中它是一个二元组(clientaddr, port)
    - 接收数据 recv(bufsize[, flags]) 使用缓冲区接收数据
    - 发送数据 send(bytes)发送数据

```
Server端开发
socket对象 --> bind((IP, PORT)) --> listen --> accept --> close
                                                  |--> recv or send --> close
```




```python
# 最简单的服务器例子
import socket

# TCP服务端编程
server = socket.socket()  # 创建socket对象
laddr = ('0.0.0.0', 9999)  # 地址和端口的元组
server.bind(laddr)  # 绑定
server.listen(1024)  # 监听
# 等待建立连接的客户端
conn, raddr = server.accept() # 阻塞

data = conn.recv(4096)  # 接收客户端信息
print(conn.getpeername(), data)
conn.send(b"Hello magedu.com")  # 回应客户端

conn.close()
server.close()
```

实战：实现WEB服务器——多线程阻塞IO版

```python
import threading
import time
import socket

html = """\
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>magedu</title>
</head>
<body>
    <h1>马哥教育www.magedu.com -- Multithread + Blocking IO</h1>
</body>
</html>\
"""

response = """\
HTTP/1.1 200 OK
Date: Mon, 24 Oct 2022 20:04:23 GMT
Content-Type: text/html
Content-Length: {}
Connection: keep-alive
Server: wayne.magedu.com

{}""".format(len(html), html).encode()


def accept(server):
    i = 1
    while True:
        conn, raddr = server.accept()
        threading.Thread(target=recv, name="recv-{}".format(i), args=(conn, raddr)).start()
        i += 1


def recv(conn: socket.socket, raddr):
    try:
        data = conn.recv(4096)
        if not data:
            print(raddr, 'bye~~~~')
            return
        # print(data)
        conn.send(response)
    except Exception as e:
        print(e, '~~~~~~~~~~~~~~')


if __name__ == '__main__':
    server = socket.socket()
    laddr = ('0.0.0.0', 9999)
    server.bind(laddr)
    server.listen(1024)

```

```
    threading.Thread(target=accept, name="accept", args=(server,), daemon=True).start()

    while True:
        time.sleep(60)
        print(threading.active_count())

```

阻塞的IO导致该线程进入阻塞态，就该让出CPU，这对性能影响不大。此多线程程序最大的问题在于，当高并发到来，连接非常多，多线程的频繁地创建和销毁。

接下来，我们用线程池来简单优化一下，看看能否提升性能？IO多路复用又是什么东西，它能提高多少性能？

实战：实现WEB服务器——线程池版

上例实现了多线程加阻塞IO版本

- 一个客户端请求到达后端，开启一个线程为之服务
- 线程内运行函数代码，接收HTTP请求并解析，返回HTTP响应报文

问题

- 大量的线程为HTTP连接服务，用完就断，而创建和销毁线程的代价太高
  - 解决的方案就是利用线程池
- 如果拥有海量线程来处理并发客户端请求，线程调度时上下文切换将给系统造成巨大的性能消耗
  - 程序层面解决不了
  - 操作系统解决：非阻塞IO、IO多路复用

下面用Python高级异步线程池ThreadPoolExecutor来改造代码。

```
import threading
import time
import socket

html = """\
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>magedu</title>
</head>
<body>
    <h1>马哥教育www.magedu.com -- Multithread Pool</h1>
</body>
</html>\
"""

response = """\
HTTP/1.1 200 OK
Date: Mon, 24 Oct 2022 20:04:23 GMT
Content-Type: text/html
Content-Length: {}

```

```
Connection: keep-alive
Server: wayne.magedu.com

{}""".format(len(html), html).encode()

from concurrent.futures import ThreadPoolExecutor

count = 10
executor = ThreadPoolExecutor(count)
# executor = ThreadPoolExecutor(max_workers=count)

def accept(server):
    # i = 1
    while True:
        conn, raddr = server.accept()
        # threading.Thread(target=recv, name="recv-{}".format(i), args=
(conn, raddr)).start()
        # i += 1
        executor.submit(recv, conn, raddr)


def recv(conn: socket.socket, raddr):
    try:
        data = conn.recv(4096)
        if not data:
            print(raddr, 'bye~~~~')
            return
        # print(data)
        conn.send(response)
    except Exception as e:
        print(e, '~~~~~~~~~~~~~')


if __name__ == '__main__':
    server = socket.socket()
    laddr = ('0.0.0.0', 9999)
    server.bind(laddr)
    server.listen(1024)

    # threading.Thread(target=accept, name="accept", args=(server,),
daemon=True).start()
    executor.submit(accept, server)

    while True:
        time.sleep(60)
        print(threading.active_count())
```