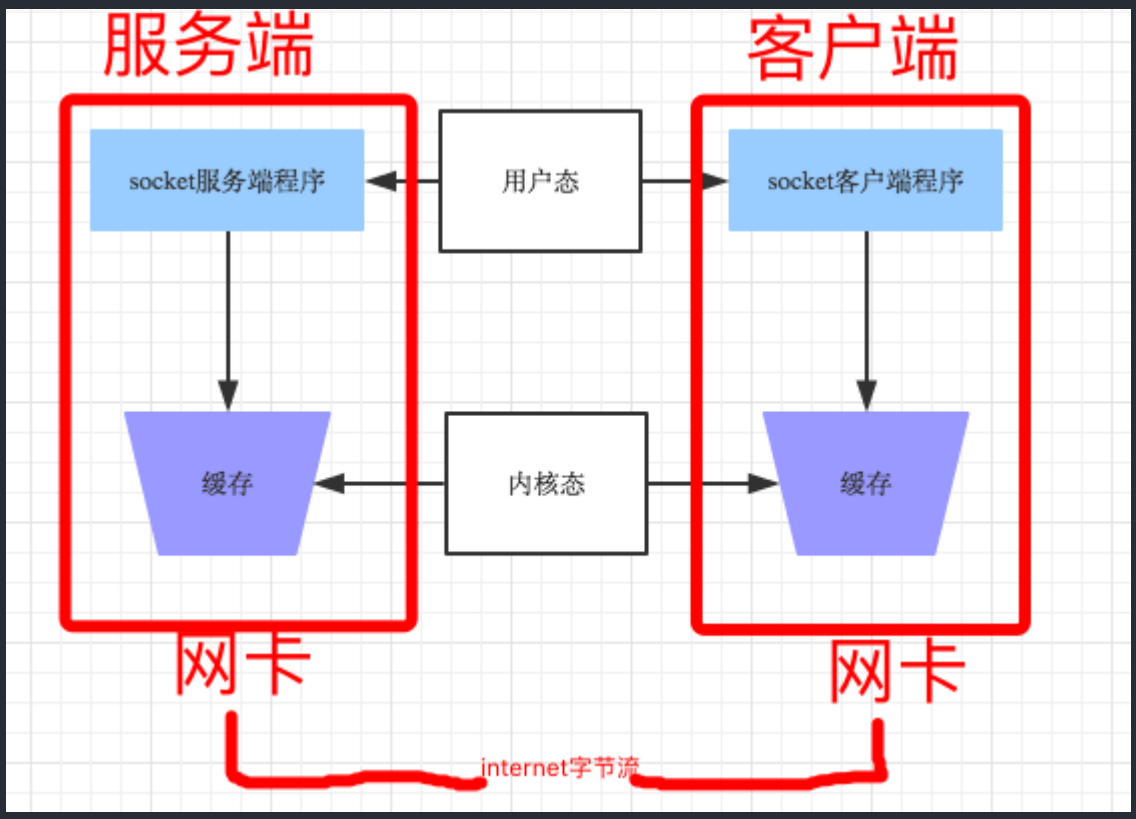
1. 粘包问题



1. 什么是粘包：
    - 接收方不知道消息之间的界限，不知道一次性提取多少字节的数据所造成的
2. TCP 粘包原因:
    - 发送方引起的粘包是由TCP协议本身造成的，TCP为提高传输效率，发送方往往要收集到足够多的数据后才发送一个TCP段。若连续几次需要send的数据都很少，通常TCP会根据优化算法把这些数据合成一个TCP段后一次发送出去，这样接收方就收到了粘包数据。
3. 总结：
    1. TCP（transport control protocol，传输控制协议）是面向连接的，面向流的，提供高可靠性服务。收发两端（客户端和服务器端）都要有一一成对的socket，因此，发送端为了将多个发往接收端的包，更有效的发到对方，使用了优化方法（Nagle算法），将多次间隔较小且数据量小的数据，合并成一个大的数据块，然后进行封包。这样，接收端，就难于分辨出来了，必须提供科学的拆包机制。 即面向流的通信是无消息保护边界的。
    2. UDP（user datagram protocol，用户数据报协议）是无连接的，面向消息的，提供高效率服务。不会使用块的合并优化算法，，由于UDP支持的是一对多的模式，所以接收端的skbuff(套接字缓冲区）采用了链式结构来记录每一个到达的UDP包，在每个UDP包中就有了消息头（消息来源地址，端口等信息），这样，对于接收端来说，就容易进行区分处理了。 即面向消息的通信是有消息保护边界的。
    3. tcp是基于数据流的，于是收发的消息不能为空，这就需要在客户端和服务端都添加空消息的处理机制，防止程序卡住，而udp是基于数据报的，即便是你输入的是空内容（直接回车），那也不是空消息，udp协议会帮你封装上消息头。
4. TCP/UDP可靠问题
    1. udp的recvfrom是阻塞的，一个recvfrom(x)必须对唯一一个sendinto(y),收完了x个字节的数据就算完成,若是y>x数据就丢失，这意味着udp根本不会粘包，但是会丢数据，不可靠

    2. tcp的协议数据不会丢，没有收完包，下次接收，会继续上次继续接收，己端总是在收到ack时才会清除缓冲区内容。数据是可靠的，但是会粘包。
5. 两种情况下会发生粘包：
    1. 发送端需要等缓冲区满才发送出去，造成粘包（发送数据时间间隔很短，数据了很小，会合到一起，产生粘包）
    2. 接收方不及时接收缓冲区的包，造成多个包接收（客户端发送了一段数据，服务端只收了一小部分，服务端下次再收的时候还是从缓冲区拿上次遗留的数据，产生粘包）
6. 拆包的发生情况
    - 当发送端缓冲区的长度大于网卡的MTU时，tcp会将这次发送的数据拆成几个数据包发送出去
7. 补充问题一：为何tcp是可靠传输，udp是不可靠传输
    - tcp在数据传输时，发送端先把数据发送到自己的缓存中，然后协议控制将缓存中的数据发往对端，对端返回一个ack=1，发送端则清理缓存中的数据，对端返回ack=0，则重新发送数据，所以 tcp是可靠的，而udp发送数据，对端是不会返回确认信息的，因此不可靠
8. 补充问题二：send(字节流)和recv(1024)及sendall
    - recv里指定的1024意思是从缓存里一次拿出1024个字节的数据；
    - send的字节流是先放入己端缓存，然后由协议控制将缓存内容发往对端，如果待发送的字节流大小大于缓存剩余空间，那么数据丢失，用sendall就会循环调用send，数据不会丢失

2. subprocess模块

1. 注意:

```
res=subprocess.Popen(cmd.decode('utf-8'),
shell=True,
stderr=subprocess.PIPE,
stdout=subprocess.PIPE)

#结果的编码是以当前所在的系统为准的，如果是windows，那么res.stdout.read()读出的就是GBK编码的，
#在接收端需要用GBK解码
#且只能从管道里读一次结果
```

3. struct模块

1. 为何要用struct模块:
    - 解决粘包方法，为字节流加上自定义固定长度报头，报头中包含字节流长度，然后一次send到对端，对端在接收时，先从缓存中取出定长的报头，然后再取真实数据
2. struct模块作用:
    - 可以把一个类型，如数字，转成固定长度的bytes，

```
struct.pack('i',1111111111111)
#struct.error: 'i' format requires -2147483648 <= number <= 2147483647 #这个是范围
```

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	string of length 1	1	
b	signed char	integer	1	(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
f	float	float	4	(4)
d	double	float	8	(4)
s	char[]	string		
p	char[]	string		
P	void *	integer		(5), (3)

3. struct示例

```
import json,struct
#假设通过客户端上传1T:1073741824000的文件a.txt

#为避免粘包,必须自定制报头
header={'file_size':1073741824000,'file_name':'/a/b/c/d/e/a.txt',
        'md5':'8f6fbf8347faa4924a76856701edb0f3'}
#1T数据,文件路径和md5值

#为了该报头能传送,需要序列化并且转为bytes
head_bytes=bytes(json.dumps(header),encoding='utf-8') #序列化并转成bytes,用于传输



#为了让客户端知道报头的长度,用struck将报头长度这个数字转成固定长度:4个字节
head_len_bytes=struct.pack('i',len(head_bytes)) #这4个字节里只包含了一个数字,该数字是报头的长度



#客户端开始发送
conn.send(head_len_bytes) #先发报头的长度,4个bytes
conn.send(head_bytes) #再发报头的字节格式
conn.sendall(文件内容) #然后发真实内容的字节格式



#服务端开始接收
```

```
head_len_bytes=s.recv(4) #先收报头4个bytes,得到报头长度的字节格式

x=struct.unpack('i',head_len_bytes)[0] #提取报头的长度



head_bytes=s.recv(x) #按照报头长度x,收取报头的bytes格式

header=json.loads(json.dumps(header)) #提取报头



#最后根据报头的内容提取真实的数据,比如

real_data_len=s.recv(header['file_size'])

s.recv(real_data_len)
```

4. 自定制报头

1. FTP服务端

```
import socket,struct,json
import subprocess
phone=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
phone.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1) #就是它，在bind前加

phone.bind(('127.0.0.1',8080))

phone.listen(5)

while True:
    conn,addr=phone.accept()
    while True:
        cmd=conn.recv(1024)
        if not cmd:break
        print('cmd: %s' %cmd)

        res=subprocess.Popen(cmd.decode('utf-8'),
                             shell=True,
                             stdout=subprocess.PIPE,
                             stderr=subprocess.PIPE)
        err=res.stderr.read()
        print(err)
        if err:
            back_msg=err
        else:
            back_msg=res.stdout.read()

        conn.send(struct.pack('i',len(back_msg))) #先发back_msg的长度
        conn.sendall(back_msg) #在发真实的内容
```

```
30
31      conn.close()
```

2. FTP客户端

```
#_*_coding:utf-8_*_
__author__ = 'Linhaifeng'
import socket,time,struct

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
res=s.connect_ex(('127.0.0.1',8080))

while True:
    msg=input('>>: ').strip()
    if len(msg) == 0:continue
    if msg == 'quit':break

    s.send(msg.encode('utf-8'))



    l=s.recv(4)
    x=struct.unpack('i',l)[0]
    print(type(x),x)
    # print(struct.unpack('I',l))
    r_s=0
    data=b''
    while r_s < x:
        r_d=s.recv(1024)
        data+=r_d
        r_s+=len(r_d)

    # print(data.decode('utf-8'))
    print(data.decode('gbk')) #windows默认gbk编码
```

5. json报头

1. 可以把报头做成字典，字典里包含将要发送的真实数据的详细信息，然后json序列化，然后用struck将序列化后的数据长度打包成4个字节（4个自己足够用了）
    - 发送时

```
先发报头长度
再编码报头内容然后发送
最后发真实内容
```

    - 接收时

```
先手报头长度，用struct取出来
根据取出的长度收取报头内容，然后解码，反序列化
从反序列化的结果中取出待取数据的详细信息，然后去取真实的数据内容
```

2. FTP服务端

```python
import socket,struct,json
import subprocess
phone=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
phone.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1) #就是它，在bind前加

phone.bind(('127.0.0.1',8080))

phone.listen(5)

while True:
    conn,addr=phone.accept()
    while True:
        cmd=conn.recv(1024)
        if not cmd:break
        print('cmd: %s' %cmd)

        res=subprocess.Popen(cmd.decode('utf-8'),
                             shell=True,
                             stdout=subprocess.PIPE,
                             stderr=subprocess.PIPE)
        err=res.stderr.read()
        print(err)
        if err:
            back_msg=err
        else:
            back_msg=res.stdout.read()

        headers={'data_size':len(back_msg)}
        head_json=json.dumps(headers)
        head_json_bytes=bytes(head_json,encoding='utf-8')

        conn.send(struct.pack('i',len(head_json_bytes))) #先发报头的长度
        conn.send(head_json_bytes) #再发报头
        conn.sendall(back_msg) #在发真实的内容

    conn.close()
```

3. FTP客户端

```python
from socket import *
import struct,json

ip_port=('127.0.0.1',8080)
client=socket(AF_INET,SOCK_STREAM)
client.connect(ip_port)

while True:
    cmd=input('>>: ')
    if not cmd:continue
    client.send(bytes(cmd,encoding='utf-8'))
```

```
    head=client.recv(4)
    head_json_len=struct.unpack('i',head)[0]
    head_json=json.loads(client.recv(head_json_len).decode('utf-8'))
    data_len=head_json['data_size']

    recv_size=0
    recv_data=b''
    while recv_size < data_len:
        recv_data+=client.recv(1024)
        recv_size+=len(recv_data)

    print(recv_data.decode('utf-8'))
    #print(recv_data.decode('gbk')) #windows默认gbk编码
```

6. FTP作业

1. 服务端

```
import socket
import struct
import json
import subprocess
import os

class MYTCPServer:
    address_family = socket.AF_INET

    socket_type = socket.SOCK_STREAM

    allow_reuse_address = False

    max_packet_size = 8192

    coding='utf-8'

    request_queue_size = 5

    server_dir='file_upload'

    def __init__(self, server_address, bind_and_activate=True):
        """Constructor.  May be extended, do not override."""
        self.server_address=server_address
        self.socket = socket.socket(self.address_family,
                                    self.socket_type)
        if bind_and_activate:
            try:
                self.server_bind()
                self.server_activate()
            except:
                self.server_close()
                raise
```

```
34
35      def server_bind(self):
36          """Called by constructor to bind the socket.
37          """
38          if self.allow_reuse_address:
39              self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
40          self.socket.bind(self.server_address)
41          self.server_address = self.socket.getsockname()
42
43      def server_activate(self):
44          """Called by constructor to activate the server.
45          """
46          self.socket.listen(self.request_queue_size)
47
48      def server_close(self):
49          """Called to clean-up the server.
50          """
51          self.socket.close()
52
53      def get_request(self):
54          """Get the request and client address from the socket.
55          """
56          return self.socket.accept()
57
58      def close_request(self, request):
59          """Called to clean up an individual request."""
60          request.close()
61
62      def run(self):
63          while True:
64              self.conn,self.client_addr=self.get_request()
65              print('from client ',self.client_addr)
66              while True:
67                  try:
68                      head_struct = self.conn.recv(4)
69                      if not head_struct:break
70
71                      head_len = struct.unpack('i', head_struct)[0]
72                      head_json = self.conn.recv(head_len).decode(self.coding)
73                      head_dic = json.loads(head_json)
74
75                      print(head_dic)
76                      #head_dic={'cmd':'put','filename':'a.txt','filesize':123123}
77                      cmd=head_dic['cmd']
78                      if hasattr(self,cmd):
79                          func=getattr(self,cmd)
80                          func(head_dic)
81                  except Exception:
82                      break
83
84      def put(self,args):
85          file_path=os.path.normpath(os.path.join(
86              self.server_dir,
```

```
            args['filename']
        ))

        filesize=args['filesize']
        recv_size=0
        print('----->',file_path)
        with open(file_path,'wb') as f:
            while recv_size < filesize:
                recv_data=self.conn.recv(self.max_packet_size)
                f.write(recv_data)
                recv_size+=len(recv_data)
                print('recvsize:%s filesize:%s' %(recv_size,filesize))


tcpserver1=MYTCPServer(('127.0.0.1',8080))

tcpserver1.run()

```

2. 客户端

```
import socket
import struct
import json
import os



class MYTCPClient:
    address_family = socket.AF_INET

    socket_type = socket.SOCK_STREAM

    allow_reuse_address = False

    max_packet_size = 8192

    coding='utf-8'

    request_queue_size = 5

    def __init__(self, server_address, connect=True):
        self.server_address=server_address
        self.socket = socket.socket(self.address_family,
                                    self.socket_type)
        if connect:
            try:
                self.client_connect()
            except:
                self.client_close()
                raise

    def client_connect(self):
```

```
        self.socket.connect(self.server_address)

    def client_close(self):
        self.socket.close()

    def run(self):
        while True:
            inp=input(">>: ").strip()
            if not inp:continue
            l=inp.split()
            cmd=l[0]
            if hasattr(self,cmd):
                func=getattr(self,cmd)
                func(l)


    def put(self,args):
        cmd=args[0]
        filename=args[1]
        if not os.path.isfile(filename):
            print('file:%s is not exists' %filename)
            return
        else:
            filesize=os.path.getsize(filename)

        head_dic={'cmd':cmd,'filename':os.path.basename(filename),'filesize':filesize}
        print(head_dic)
        head_json=json.dumps(head_dic)
        head_json_bytes=bytes(head_json,encoding=self.coding)

        head_struct=struct.pack('i',len(head_json_bytes))
        self.socket.send(head_struct)
        self.socket.send(head_json_bytes)
        send_size=0
        with open(filename,'rb') as f:
            for line in f:
                self.socket.send(line)
                send_size+=len(line)
                print(send_size)
            else:
                print('upload successful')



client=MYTCPClient(('127.0.0.1',8080))

client.run()
```