一  粘包现象

首先我们要知道什么是粘包：

我们首先要知道，粘包是对于TCP来说的，UDP是不存在粘包一说的，那么TCP在传输数据的过程的特点是什么呢：

1 会将数据量较小，且发送时间间隔较短的的数据一起打包发送，那么这里所讲的时间较短是相比较网络延迟来说的，  
比如我们两次发送间隔为0.00001秒，那么网络延迟为0.001，这个时候两次的数据就会打包发送，这是一种优化机制。

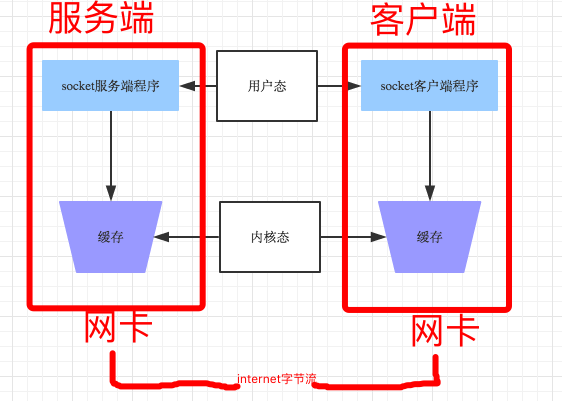
2 TCP协议发送数据时，是源源不断的发送，像水流一样，因此TCP又叫流式协议。

我们知道服务端在接收消息时是有一个最大限制的=====>conn.recv(1024),1024表示1024个bytes。那么如果我们一次传输的数据超过了1024bytes，剩余的数据会存在我们接收端计算机操作系统缓存中，也就是说，接收方并不知道发送方传输了多少数据，所以这个时候问题就出现了，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](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)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。

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。**

须知：只有TCP有粘包现象，UDP永远不会粘包，为何，且听我娓娓道来

首先需要掌握一个socket收发消息的原理



二 粘包发生的场景：

第一种：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）

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 服务端

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 客户端

第二种：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）

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 服务端

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 客户端

三 从根源解决粘包：

问题的根源在于，接收端不知道发送端将要传送的字节流的长度，所以解决粘包的方法就是围绕，如何让发送端在发送数据前，把自己将要发送的字节流总大小让接收端知晓，然后接收端来一个死循环接收完所有数据。

那么这个时候我们需要用到一个模块struct模块。struct模块的作用就是可以把一个类型，如数字，转成固定长度的bytes。比如：struct.pack('i',1111111111111)

**重点来了：！！！！！！！！！！！！！！！**

我们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

**首先我们来看struct模块的具体用法：**

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)