20190211

OSI七层参考模型是什么？为什么要这样?

OSI七层参考模型自上至下分别为：

> 应用层 常见协议有HTTP/HTTPS/FTP/SSH/DNS/SMTP/POP3 最多1460个字节
>
> 表示层 主要功能是定义数据格式以及加密/解密
>
> 会话层 定义了如何开始、控制和结束一个会话，包括对多个双向消息的控制和管理，没有协议产生
>
> 传输层 TCP/UDP 提供端到端的连接服务 协议数据单元（PDU）为**数据段**（Segment）
>
> 网络层 IP/ICMP/IGMP 把异构网络（使用不同协议的网络）联系在了一起
>
> 协议数据单元（PDU）为**数据包(packet)**
>
> 数据链路层 常见协议有ARP/RARP，PPP， 协议数据单元（PDU）为**数据帧（Frame）**
>
> 物理层 传输比特流 协议数据单元（PDU）为**数据流**（Bit Stream）
>
> *PDU：网络的对等实体传送的信息单元，包括了控制信息，地址信息，或者数据

进行分层是为了方便人们学习计算机网络协议的各种规范。网络模型的每一层都实现了不同的功能，并向高层次提供了接口，处在高层次的系统仅是利用低层次的系统所提供的接口和功能，不需了解低层实现该功能所采用的算法和协议等细节，有利于工程的模块化，降低了编程的复杂度，使得程序便于维护，提高了产品的开发速度。

进程间通信有哪些方式?

主要有：

> 管道通信
>
> 信号量
>
> 信号
>
> 消息队列
>
> 共享内存
>
> socket套接字编程

TCP协议的状态有哪些？TCP首部有哪些内容，其各自的作用是什么？

TCP协议状态有11种

其中客户端独有的：

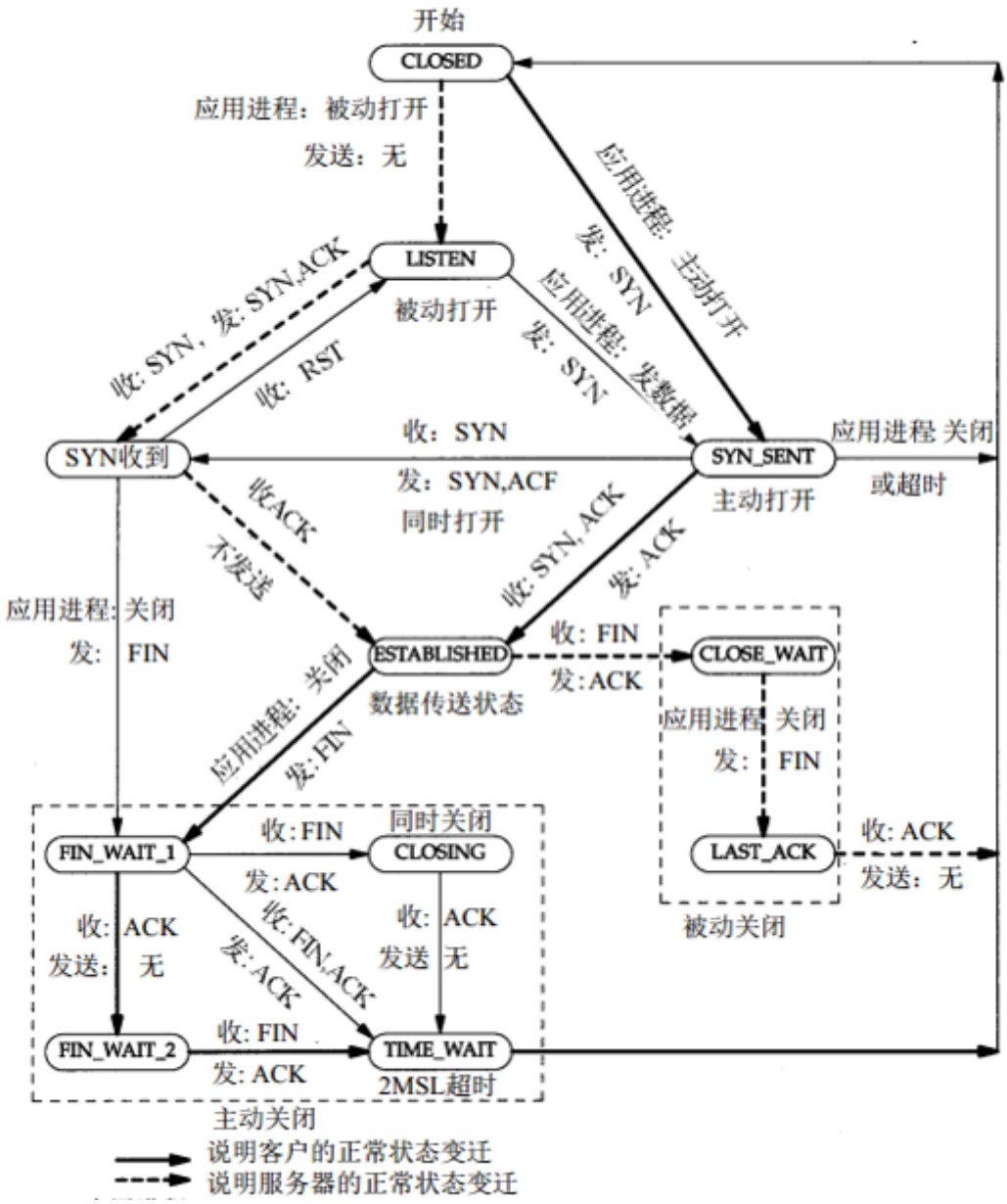
1. SYN_SENT 在发送连接请求后等待匹配的连接请求
2. FIN_WAIT1 等待远程TCP的连接中断请求，或先前的连接中断请求的确认
3. FIN_WAIT2 从远程TCP等待连接中断请求
4. CLOSING 等待远程TCP对连接中断的确认
5. TIME_WAIT 等待足够的时间以确保远程TCP接收到连接中断请求的确认

服务器独有的：

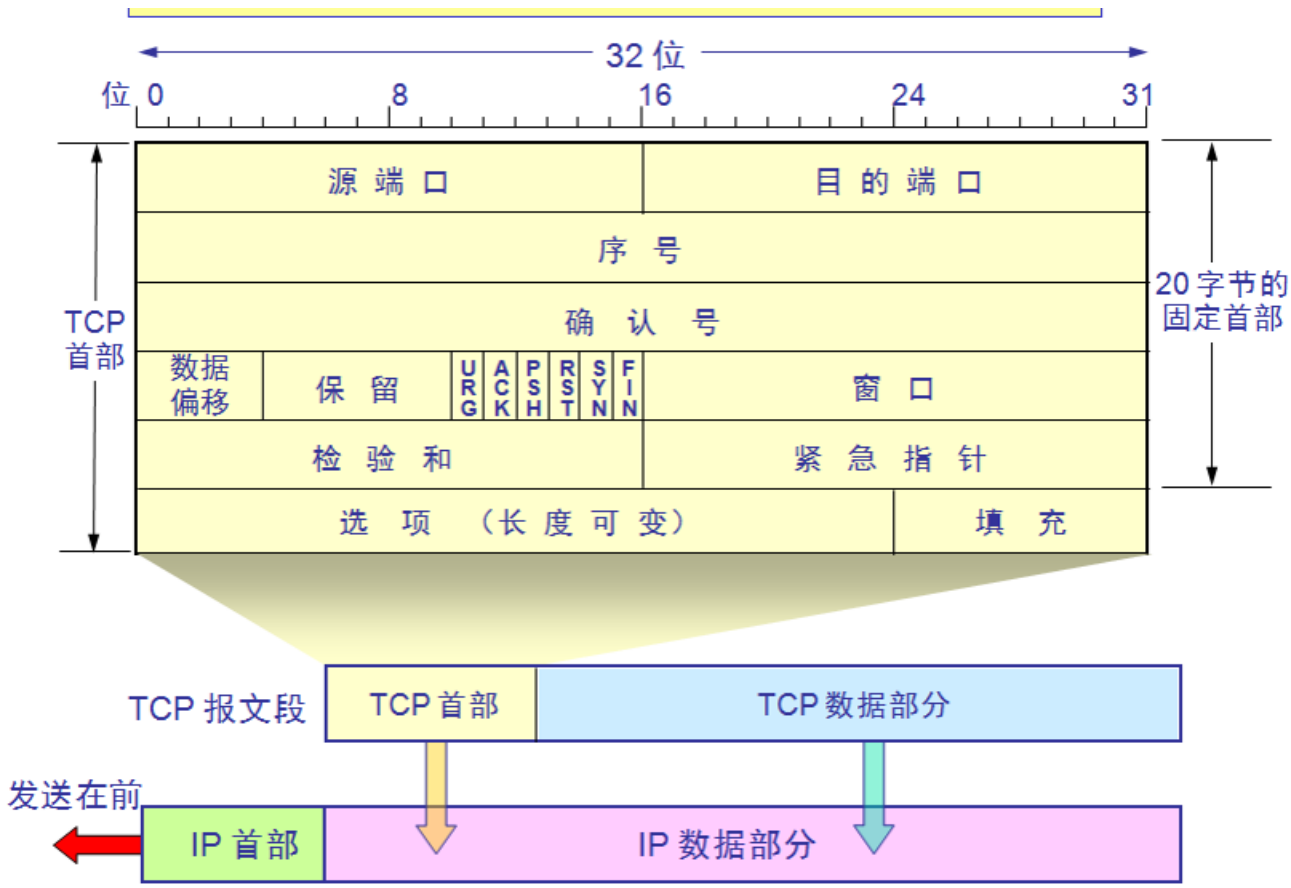
1. LISTEN 侦听来自远方TCP端口的连接请求
2. SYN_RCVD 在收到和发送一个连接请求后等待对连接请求的确认
3. CLOSE_WAIT 等待从本地用户发来的连接中断请求
4. LAST_ACK 等待原来发向远程TCP的连接中断请求的确认

共有的：

1. CLOSED 没有任何连接状态
2. ESTABLISHED 代表一个打开的连接，数据可以传送给用户


开始
CLOSED
应用进程：被动打开
发送：无
LISTEN
被动打开
收: SYN，发: SYN,ACK
收: RST
应用进程: 发数据
发: SYN
应用进程: 主动打开
发: SYN
SYN收到
收：SYN
发：SYN,ACF
同时打开
SYN_SENT
主动打开
应用进程 关闭
或超时
收ACK
不发送
收: SYN, ACK
发: ACK
应用进程: 关闭
发: FIN
ESTABLISHED
数据传送状态
收: FIN
发:ACK
CLOSE_WAIT
应用进程 关闭
发: FIN
应用进程: 关闭
发:FIN
同时关闭
FIN_WAIT_1
收:FIN
发:ACK
CLOSING
收: ACK
发送：无
LAST_ACK
被动关闭
收: ACK
发送: 无
收: FIN,ACK
发: ACK
收: ACK
发送 无
FIN_WAIT_2
收: FIN
发: ACK
TIME_WAIT
2MSL超时
主动关闭
说明客户的正常状态变迁
说明服务器的正常状态变迁

TCP首部内容包括：



1. TCP协议的报文头共有20个字节，前两个字节为

   端口号:（在0~ 65535间，包括源端口号和目的端口号），其中系统预留的端口号为： 0 ~ 1023，其余自定义服务器程序端口号都要大于1024。

2. 序号:是对发送进行编号， 代表的是即将要发送的数据的第一个序号，因为MTU 1500字节 ，所以TCP报文（IP首部减去TCP首部 ）能携带最大数据量为1460字节。当待传输的数据(文件)大于1460时，必须要进行分片，当分片的报文发送给对端时，要重组数据(文件)，就根据序号来进行重组。

3. 确认号:是当前已经收到对方的数据的序号，接下来对方可以发送确认号的数据

4. 数据偏移(TCP首部长度): 4位 0~15 5 * 4B = 20个字节 一般情况下

5. 窗口：16 位长。窗口大小字段表示在确认了字节之后还可以发送多少个字节,此字段用来进行流量控制。单位为字节数，这个值是本机期望一次接收的字节数（流量控制 滑动窗口机制 ）

6. URG：紧急指针（urgent pointer）有效，紧急指针指出在本报文段中的紧急数据的最后一个字节的序号

7. ACK：ACK 位置 1 表明确认号是合法的。如果 ACK 为 0，那么数据报不包含确认信息，确认字段被 省略。

8. PSH：表示是带有 PUSH 标志的数据。接收方因此请求数据报一到便可送往应用程序而不必等到缓冲区装满时才发送。当 PSH=1 时，则报文段会被尽快地交付给目的方，不会对这样的报文段使用缓策略

9. RST：用于复位由于主机崩溃或其他原因而出现的错误的连接。还可以用于拒绝非法的数据报或拒绝连接请求。当 RST 为 1 时，表明 TCP 连接中出现了严重的差错，必须释放连接，然后再重新建立连接。

10. SYN：用于建立连接。当 SYN=1 时，表示发起一个连接请求。

11. FIN：用于释放连接。当 FIN=1 时，表明此报文段的发送端的数据已发送完成，并要求释放连接。

12. 窗口：

13. 校验和：16 位长。是为了确保高可靠性而设置的。它校验头部、数据和伪 TCP 头部之和。
14. 可选项：0 个或多个 32 位字。包括最大 TCP 载荷，窗口比例、选择重复数据报等选项。
15. 填充：当数据不够20字节时补到20字节时用的

TCP链接建立的过程需要三次握手，为什么？

为了应对在服务端向客服端发送确认帧超时或是丢失的情况。

如果只进行两次握手只能保证建立一条单方向连接。

TCP链接断开的过程需要四次挥手，为什么？

TCP连接是一个全双工的协议,双方都既能够接收数据，也能发送数据，各自是独立的

1. 当主机A确认发送完数据且知道乙已经接受完了，想要关闭发送数据口（当然确认信号还是可以发），就会发FIN给主机B.
2. 主机乙收到甲发送的FIN，表示收到了，就会发送ACK回复。
3. 但这是乙可能还在发送数据，没有想要关闭数据口的意思，所以FIN与ACK不是同时发送的，而是等到乙数据发送完了，才会发送FIN给主机A.
4. A收到B发来的FIN，知道B的数据也发送完了，回复ACK，A等待2MSL以后，没有收到B传来的任何消息，知道B已经收到自己的ACK了，A就关闭链接，B也关闭链接了。

关闭链接时可不可以服务器端主动关闭?为什么？

一般情况下，服务器端不会主动关闭连接，因为主动关闭连接时，会进入到time_wait状态，该状态的超时时间是2MSL，此时服务器在2MSL时间内只能等待，不能执行其他操作，对服务器来说，资源就浪费了。

在TCP链接中，有一个time_wait的状态，该状态可以删掉吗？为什么？

不可以，如果删掉会产生两个问题

1. 如果client发送ACK，server端没有收到，此时server会重发，保证连接能够断开的
2. 造成窜链的情况，数据接收发生混乱

使用wireshark抓包分析TCP协议的三次握手和四次挥手，并截图提交

三次握手：

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.63.1	192.168.63.128	TCP	66	4694 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000536	192.168.63.128	192.168.63.1	TCP	66	22 → 4694 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.000647	192.168.63.1	192.168.63.128	TCP	54	4694 → 22 [ACK] Seq=1 Ack=1 Win=525568 Len=0
4	0.013612	192.168.63.128	192.168.63.1	SSHv2	95	Server: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.1)
5	0.054427	192.168.63.1	192.168.63.128	TCP	54	4694 → 22 [ACK] Seq=1 Ack=42 Win=525312 Len=0
6	0.123678	192.168.63.1	192.168.63.128	SSHv2	103	Client: Protocol (SSH-2.0-nsssh2_6.0.0009 NetSarang Computer, Inc.)
7	0.124050	192.168.63.128	192.168.63.1	TCP	60	22 → 4694 [ACK] Seq=42 Ack=50 Win=29312 Len=0
8	0.125427	192.168.63.1	192.168.63.128	SSHv2	1470	Client: Key Exchange Init
9	0.125724	192.168.63.128	192.168.63.1	TCP	60	22 → 4694 [ACK] Seq=42 Ack=1466 Win=32128 Len=0

四次挥手：

94	9.646773	192.168.63.1	192.168.63.128	SSHv2	106	Client: Encrypted packet (len=52)
95	9.646870	192.168.63.1	192.168.63.128	SSHv2	106	Client: Encrypted packet (len=52)
96	9.647343	192.168.63.128	192.168.63.1	TCP	60	22 → 4694 [ACK] Seq=5342 Ack=3006 Win=32128 Len=0
97	9.647726	192.168.63.1	192.168.63.128	SSHv2	122	Client: Encrypted packet (len=68)
98	9.664968	192.168.63.128	192.168.63.1	TCP	60	22 → 4694 [FIN, ACK] Seq=5342 Ack=3074 Win=32128 Len=0
99	9.665054	192.168.63.1	192.168.63.128	TCP	54	4694 → 22 [ACK] Seq=3074 Ack=5343 Win=525056 Len=0
00	9.665176	192.168.63.1	192.168.63.128	TCP	54	4694 → 22 [FIN, ACK] Seq=3074 Ack=5343 Win=525056 Len=0
01	9.665479	192.168.63.128	192.168.63.1	TCP	60	22 → 4694 [ACK] Seq=5343 Ack=3075 Win=32128 Len=0

使用tcpdump命令在linux下抓包，然后用wireshark进行分析，并截图提交

使tcpdum命令记录信息

```
$ifconfig //查看网络信息
$sudo tcpdump -i ens33   //抓取ens33网卡的数据包并打印
$sudo tcpdump -i ens33 -w test.pcap //抓取ens33网卡的数据包，并保存到test.pcap文件中
```

三次握手：

6	4.458937	Vmware_c0:00:08	Vmware_29:f2:6d	ARP	60	Who has 192.168.63.128? Tell 192.168.63.1
7	4.458951	Vmware_29:f2:6d	Vmware_c0:00:08	ARP	42	192.168.63.128 is at 00:0c:29:29:f2:6d
8	4.648801	192.168.63.1	192.168.63.128	TCP	66	7717 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
9	4.648823	192.168.63.128	192.168.63.1	TCP	66	22 → 7717 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
10	4.649007	192.168.63.1	192.168.63.128	TCP	60	7717 → 22 [ACK] Seq=1 Ack=1 Win=525568 Len=0
11	4.658534	192.168.63.128	192.168.63.1	SSHv2	95	Server: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.1)
12	4.699149	192.168.63.1	192.168.63.128	TCP	60	7717 → 22 [ACK] Seq=1 Ack=42 Win=525312 Len=0

四次挥手：

117	18.521334	192.168.63.1	192.168.63.128	SSHv2	106	Client: Encrypted packet (len=52)
118	18.521427	192.168.63.128	192.168.63.1	TCP	54	22 → 7717 [ACK] Seq=5118 Ack=3318 Win=32128 Len=0
119	18.522280	192.168.63.1	192.168.63.128	SSHv2	122	Client: Encrypted packet (len=68)
120	18.528652	192.168.63.128	192.168.63.1	TCP	54	22 → 7717 [FIN, ACK] Seq=5118 Ack=3386 Win=32128 Len=0
121	18.528989	192.168.63.1	192.168.63.128	TCP	60	7717 → 22 [ACK] Seq=3386 Ack=5119 Win=525312 Len=0
122	18.529810	192.168.63.1	192.168.63.128	TCP	60	7717 → 22 [FIN, ACK] Seq=3386 Ack=5119 Win=525312 Len=0
123	18.531643	192.168.63.128	192.168.63.1	TCP	54	22 → 7717 [ACK] Seq=5119 Ack=3387 Win=32128 Len=0
124	20.919384	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.63.254? Tell 192.168.63.1
125	21.424008	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.63.254? Tell 192.168.63.1
126	22.421014	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.63.254? Tell 192.168.63.1