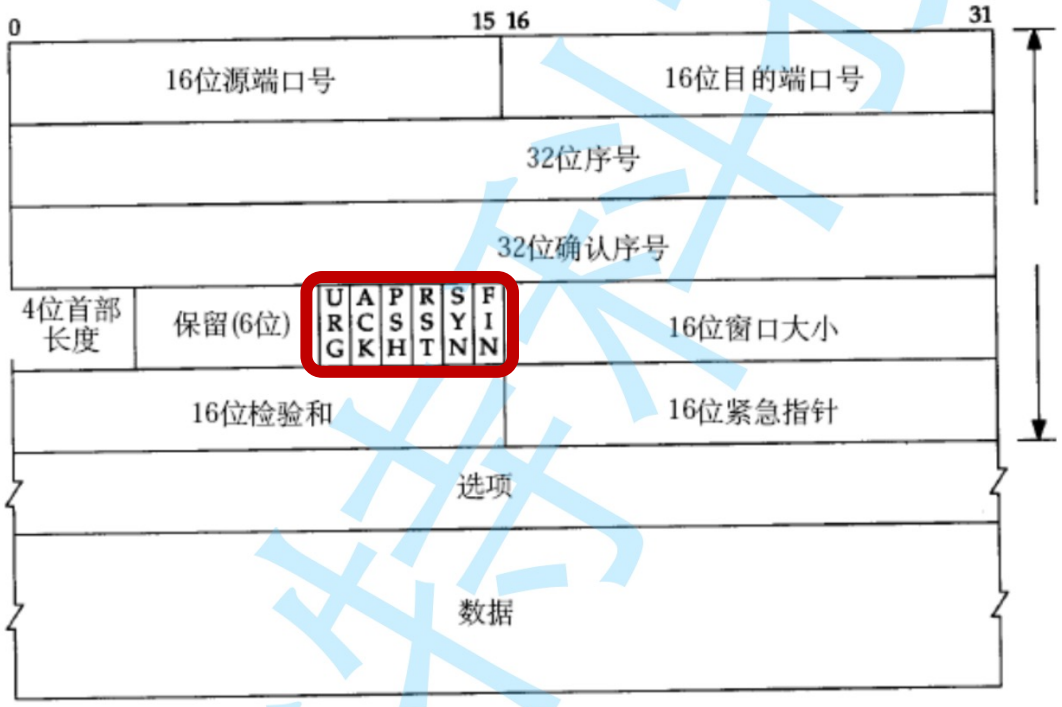
0309_Tcp

16位窗口大小，填的是自己的接收缓冲区的剩余空间大小！！！

下面我们要来谈tcp报头中的6个标记位



6个标记位：
使用一个bit表示某种含义的

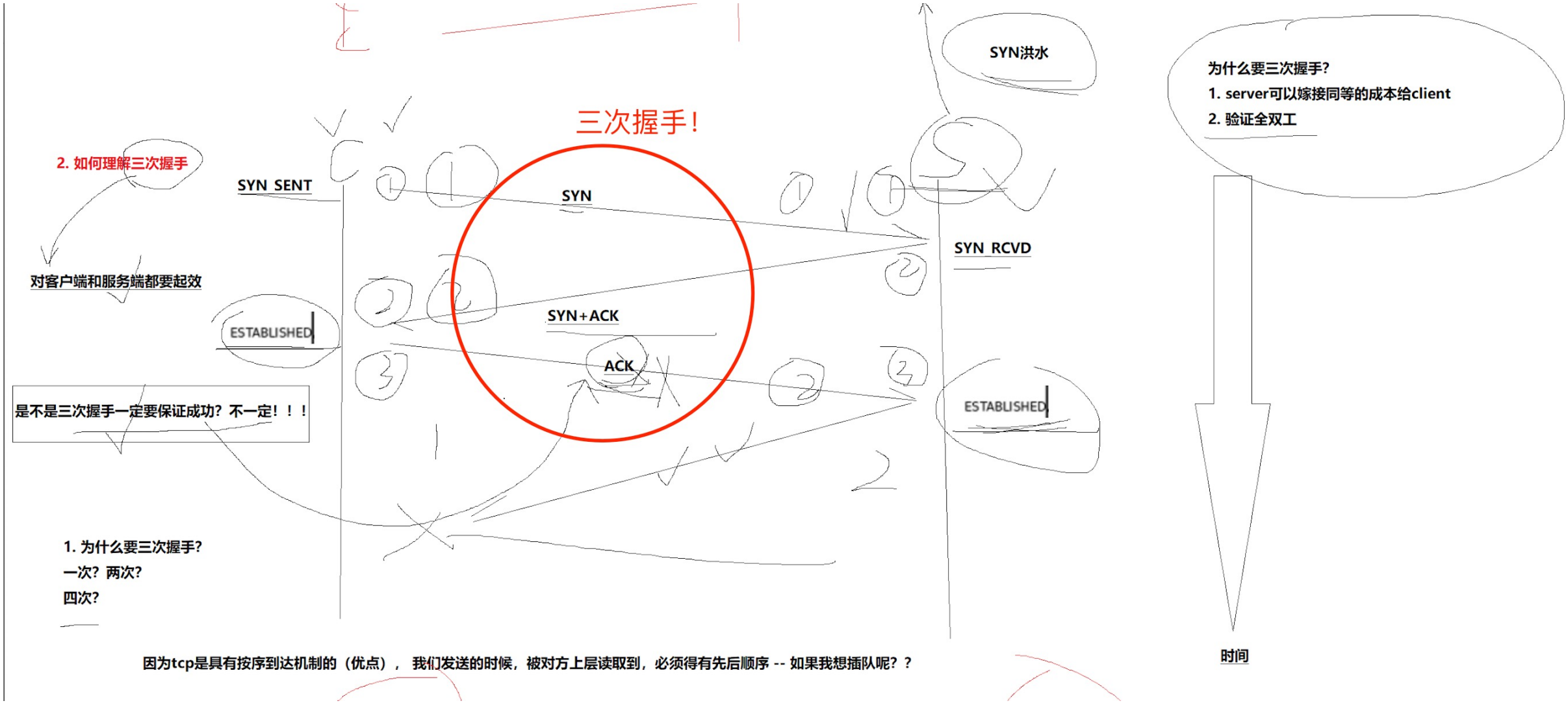
他们是用来标记报文的类型的！

1. SYN：该报文是一个链接请求的报文
2. FIN：该报文是一个断开链接的报文
3. ACK：确认应答标记位，凡事该报文具有应答特征，该标志位都被设置为1
4. 其他：我们先稍微谈下三次握手和四次挥手，再来理解更好

三次握手和四次挥手

因为有大量的client将来可能连接server
所以server一定会存在大量的连接
OS是要管理这些连接的！先组织再描述！
所以连接本质其实就是对某种数据结构的操作
维护连接是有成本的（内存+cpu资源）

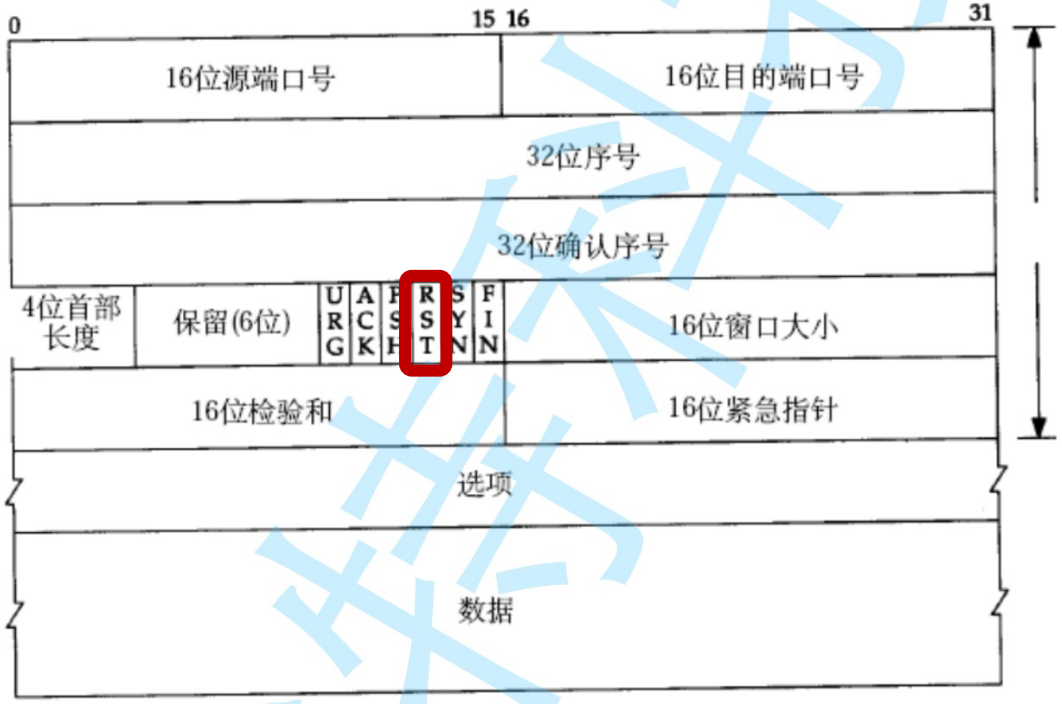
如何理解三次握手？


三次握手！
SYN洪水
2. 如何理解三次握手
SYN_SENT
SYN
SYN_RCVD
对客户端和服务端都要起效
ESTABLISHED
SYN+ACK
ACK
ESTABLISHED
是不是三次握手一定要保证成功？不一定！！！
1. 为什么要三次握手？
一次？两次？
四次？
为什么要三次握手？
1. server可以嫁接同等的成本给client
2. 验证全双工
时间
因为tcp是具有按序到达机制的（优点），我们发送的时候，被对方上层读取到，必须得有先后顺序 -- 如果我想插队呢？？

为什么要三次握手？
一次？两次？四次？

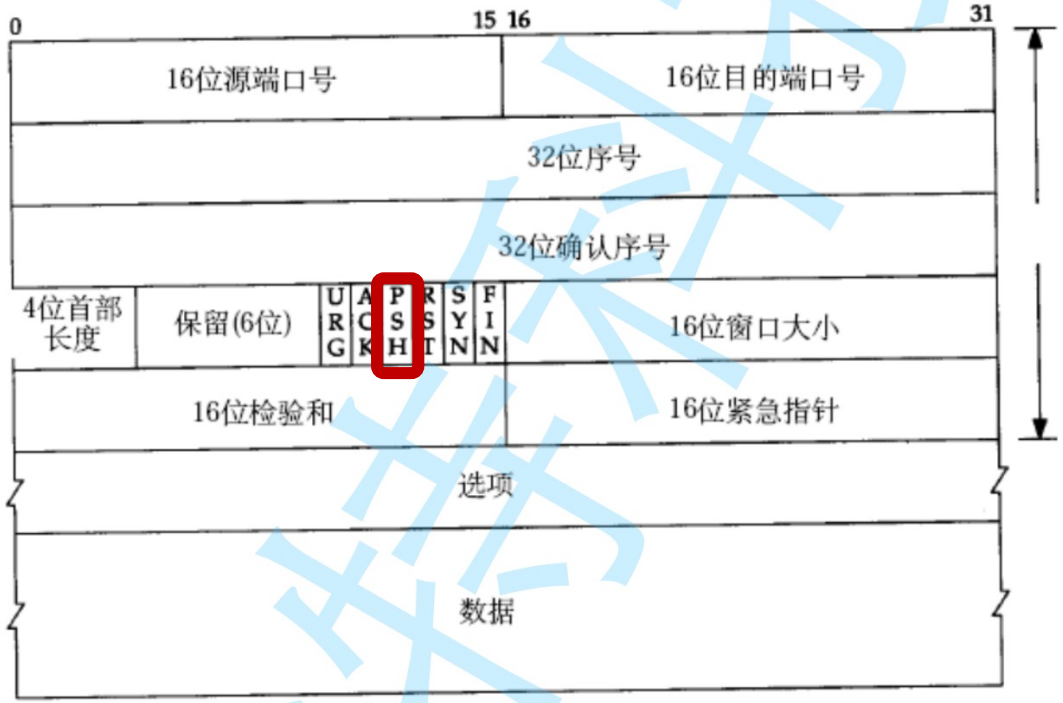
1. server可以嫁接同等的成本给client
2. 验证全双工

如果我们只用一次握手
如果一个黑客，向你的服务器发送大量请求
注意，服务器维护连接是有成本的
所以如果向你的服务器一次发一堆请求，服务器马上就挂掉了
这种情况叫做SYN洪水



现在我们来讲RST标记位

现在假设一种场景：
现在client向server发送三次握手的最后一个ack之后，它就已经认为链接已经建立好了。所以此时他可能会马上向server发送请求，但是此时server可能还没收到那个ack（即server可能还没握手好），server收到client的请求的话，此时server会认为这是一个异常链接，因此此时server就会向client发送一个RST，表示链接重置，表示要重新进行三次握手

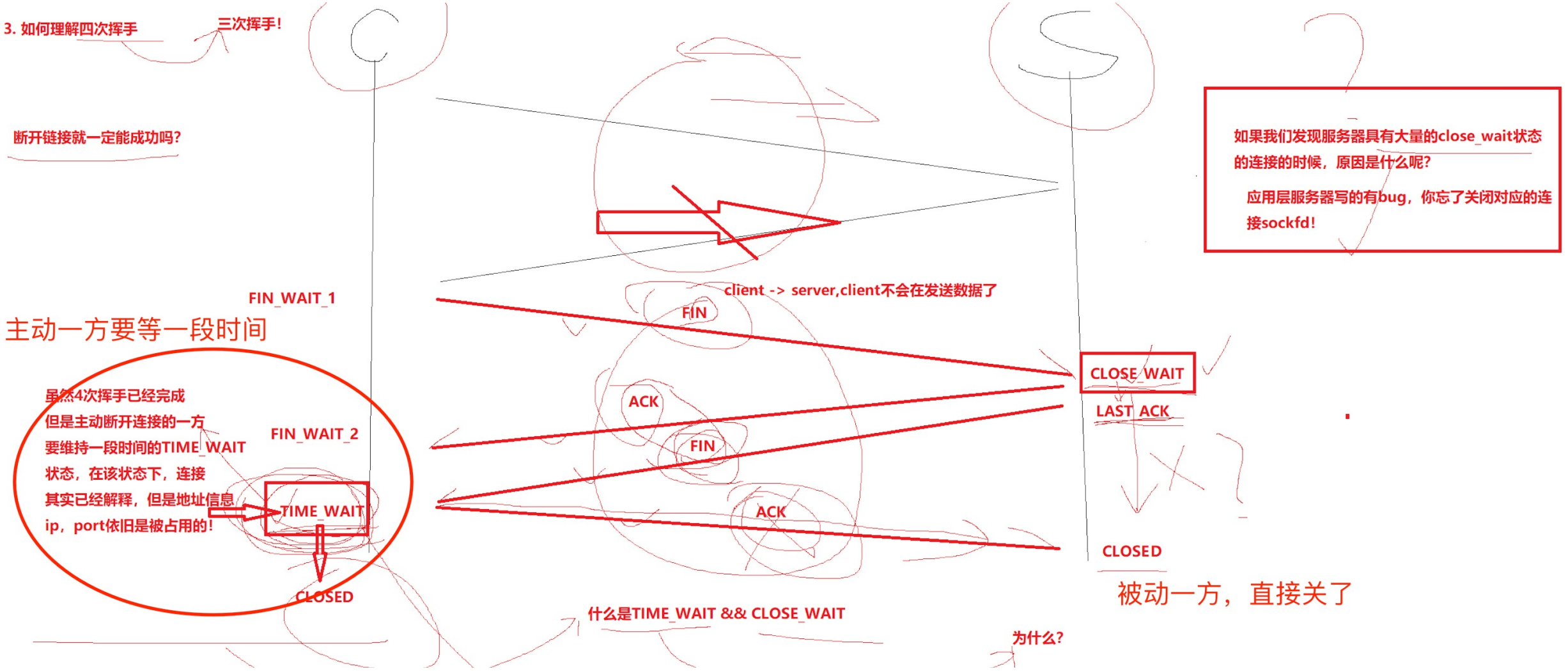


PSH：督促对方尽快把数据进行向上交付！

因为tcp是具有按序到达的机制的（优点）
所以这意味着我们发送的时候，被对方上层读取到，必须得有先后顺序，**但是如果我想插队呢？**

URG：紧急标志位，要配合16位紧急指针使用

四次挥手



如果我们发现服务器具有大量的close_wait状态的连接的时候，原因是什么呢?
应用层服务器写的有bug，忘了关闭对应的sockfd

```cpp
int main()
{
    Sock sock;
    int listen_sock = sock.Socket();
    sock.Bind(listen_sock, 8080);
    sock.Listen(listen_sock);
    while(true)
    {
        std::string client_ip;
        uint16_t client_port;
        int sockfd = sock.Accept(listen_sock, &client_ip, &client_port);
        if(sockfd > 0)
        {
            std::cout << "[" << client_ip << " : " << client_port << "]# " << sockfd << std::endl;
        }
    }
    return 0;
}
```

我们这样写，当我们去调用，
结束之后
不去close

```
(base) [yufc@ALiCentos7:~/Src/BitCodeField/0309]$ ./TcpServer
[NORMAL] [1683776563] create socket success, sock: 3
[NORMAL] [1683776563] init TcpServer Success
[NORMAL] [1683776570] link success, serviceSock: 4 | 43.136.113.128 : 28004238
[43.136.113.128 : 59728]# 4
```

启动服务器，telnet建立连接

```
(base) [yufc@ALiCentos7:~]$ netstat -ntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:43815         127.0.0.1:55134         ESTABLISHED 15217/node
tcp        0      0 127.0.0.1:55134         127.0.0.1:43815         ESTABLISHED -
tcp        0      0 172.31.31.69:57446      100.100.0.5:443         TIME_WAIT   -
tcp        0    188 172.31.31.69:22         120.236.174.205:64922   ESTABLISHED -
tcp        0      0 127.0.0.1:43815         127.0.0.1:55132         ESTABLISHED 15125/node
tcp        0      0 172.31.31.69:50780      100.100.30.26:80        ESTABLISHED -
tcp        0      0 127.0.0.1:55132         127.0.0.1:43815         ESTABLISHED -
tcp        0      0 172.31.31.69:8080       43.136.113.128:59728    ESTABLISHED 16022/./TcpServer
(base) [yufc@ALiCentos7:~]$
```

ESTABLISHED

```
Connection closed by foreign host.
(base) [yufc@VM-12-12-centos:~/Core]$ telnet 47.120.41.234 8080
Trying 47.120.41.234...
Connected to 47.120.41.234.
Escape character is '^]'.
^]

telnet> quit
Connection closed.
(base) [yufc@VM-12-12-centos:~/Core]$
```

客户端主动断开！

```
(base) [yufc@ALiCentos7:~]$ netstat -ntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 172.31.31.69:57452      100.100.0.5:443         TIME_WAIT   -
tcp        0      0 127.0.0.1:43815         127.0.0.1:55134         ESTABLISHED 15217/node
tcp        0      0 127.0.0.1:55134         127.0.0.1:43815         ESTABLISHED -
tcp        0    204 172.31.31.69:22         120.236.174.205:64922   ESTABLISHED -
tcp        0      0 127.0.0.1:43815         127.0.0.1:55132         ESTABLISHED 15125/node
tcp        0      0 172.31.31.69:50780      100.100.30.26:80        ESTABLISHED -
tcp        0      0 127.0.0.1:55132         127.0.0.1:43815         ESTABLISHED -
tcp        1      0 172.31.31.69:8080       43.136.113.128:59728    CLOSE_WAIT  16022/./TcpServer
(base) [yufc@ALiCentos7:~]$
```

此时发现，就是close_wait

所以如果不管sockfd，来了大量链接
就会存在大量close_wait

再次启动服务

```
(base) [yuric@AliCentos7:~]$ netstat -natp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43815         0.0.0.0:*               LISTEN      15125/node
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      16818/./TcpServer
tcp        0      0 127.0.0.1:43815         127.0.0.1:55134         ESTABLISHED 15217/node
tcp        0      0 127.0.0.1:55134         127.0.0.1:43815         ESTABLISHED -
```

telnet进行连接，连接成功后就是ESTABLISHED，这些都没问题

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43815         0.0.0.0:*               LISTEN      15125/node
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      16818/./TcpServer
tcp        0      0 127.0.0.1:43815         127.0.0.1:55134         ESTABLISHED 15217/node
tcp        0      0 172.31.31.69:8080       43.136.113.128:60284    ESTABLISHED 16818/./TcpServer
tcp        0      0 127.0.0.1:55134         127.0.0.1:43815         ESTABLISHED -
```

如果我们直接把服务端关闭，我们会发现会进入一个TIME_WAIT状态

```
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43815         0.0.0.0:*               LISTEN      15125/node
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43815         127.0.0.1:55134         ESTABLISHED 15217/node
tcp        0      0 172.31.31.69:8080       43.136.113.128:60284    TIME_WAIT   -
tcp        0      0 127.0.0.1:55134         127.0.0.1:43815         ESTABLISHED -
tcp        0      0 172.31.31.69:57480      100.100.0.5:443         TIME_WAIT   -
tcp        0    196 172.31.31.69:22         120.236.174.205:64922   ESTABLISHED -
tcp        0      0 127.0.0.1:43815         127.0.0.1:55132         ESTABLISHED 15125/node
```

此时，服务器是无法重启的！
如果用同一个端口号的话
这就是为什么我们经常要等一下，才能能重启服务端，不然会绑定失败！

虽然四次挥手已经完成
但是主动断开链接的一方
要维持一段时间的TIME_WAIT状态！
在该状态下，连接其实已经结束
但是ip，port依旧是被占用的

但是，如果是大型业务，服务器必须具有能够立即重启的能力！怎么办？
设置一个socket的选项！

```
int Socket()
{
    int listen_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_sock < 0)
    {
        logMessage(FATAL, "create socket error, %d: %s", errno, strerror(errno));
        exit(2);
    }
                                设置socket的选项！
    int opt = 1;
    setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof opt);
    logMessage(NORMAL, "create socket success, sock: %d", listen_sock);
    return listen_sock;
}
void Bind(int sock, uint16_t port, std::string ip = "0.0.0.0")
{
```

为什么需要TIME_WAIT?

- [TIME_WAIT -> CLOSED] 客户端要等待一个2MSL(Max Segment Life, 报文最大生存时间)的时间, 才会进入CLOSED状态.

保证历史数据从网络中消散