三次握手与四次挥手

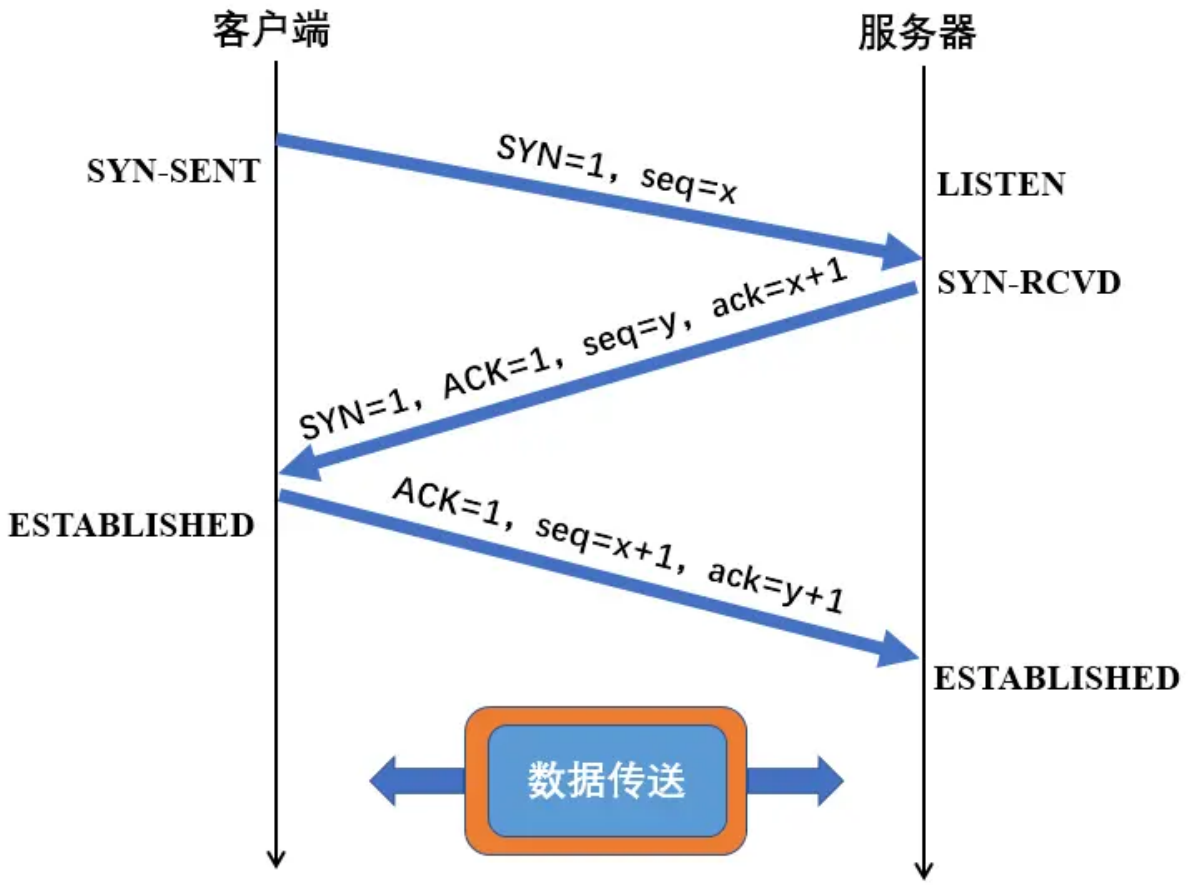
前情回顾

TCP是面向连接的协议。传输连接是用来传送TCP报文的，TCP连接传输的三个阶段分别为：**连接建立、数据传送和连接释放。**

TCP连接的建立采用**客户服务器模式。**主动发起连接建立的应用进程叫做客户，而被动等待连接建立的应用进程叫做服务器。

1.TCP的连接建立——三次握手

TCP建立连接的过程叫做握手，握手需要在客户和服务器之间交换三个TCP报文段，三次握手的过程如下图所示。



(1) **第一次握手**：客户端TCP首先向服务器端TCP发送**连接请求报文段**，这时首部的中的同步位 SYN = 1，同时选择一个初始序号（随机序号） seq = x。此时，客户进程进入**同步已发送**（SYN-SENT）状态

> TCP 规定，SYN报文段（即 SYN = 1的报文段）**不能携带数据**，但是要**消耗掉一个序号**。

> 上文说了，同步SYN控制位是在建立连接时同步序号。当 SYN = 1 而 ACK = 0 表明这是一个连接请求报文段。由于 SYN = 1，所以连接请求报文也叫做 SYN报文段。

(2) **第二次握手**：服务器收到 SYN报文段后，如同意连接，则服务器会为该TCP连接**分配缓存和变量**，并向客户端返回**确认报文段**，在确认报文段中同步位 SYN = 1 和 确认位 ACK = 1，确认号 ack = x + 1，同时也为自己选择一个初始序号 seq = y。这时TCP服务器进程进入**同步收到（SYN-RCVD）状态。**

> 这个确认报文段也**不能携带数据**，同样需要消耗一个序号。

> 如上文所述，这里 ack = x + 1表示服务器已经正确接收了客户进程序号为 x 的报文段，服务器期望客户进程下一个报文段的第一个字节数据的序号为 x + 1。该确认报文段也称为**SYNACK报文段**。

(3) **第三次握手**：客户进程在收到服务器进程的确认报文后，客户端为该TCP连接**分配缓存和变量**，并向服务器端返回一个报文段，这个报文段是对服务器确认报文段进行确认，该报文段中 ACK = 1，确认号 seq = y + 1，而自己序号为 x + 1（即第二次握手服务器确认报文段的确认号）。客户端在发送ACK报文段后进入**已建立连接（ESTABLISHED）状态**，这时TCP连接已经建立。

当服务器收到客户端的确认后，也**进入ESTABLISHED状态。**

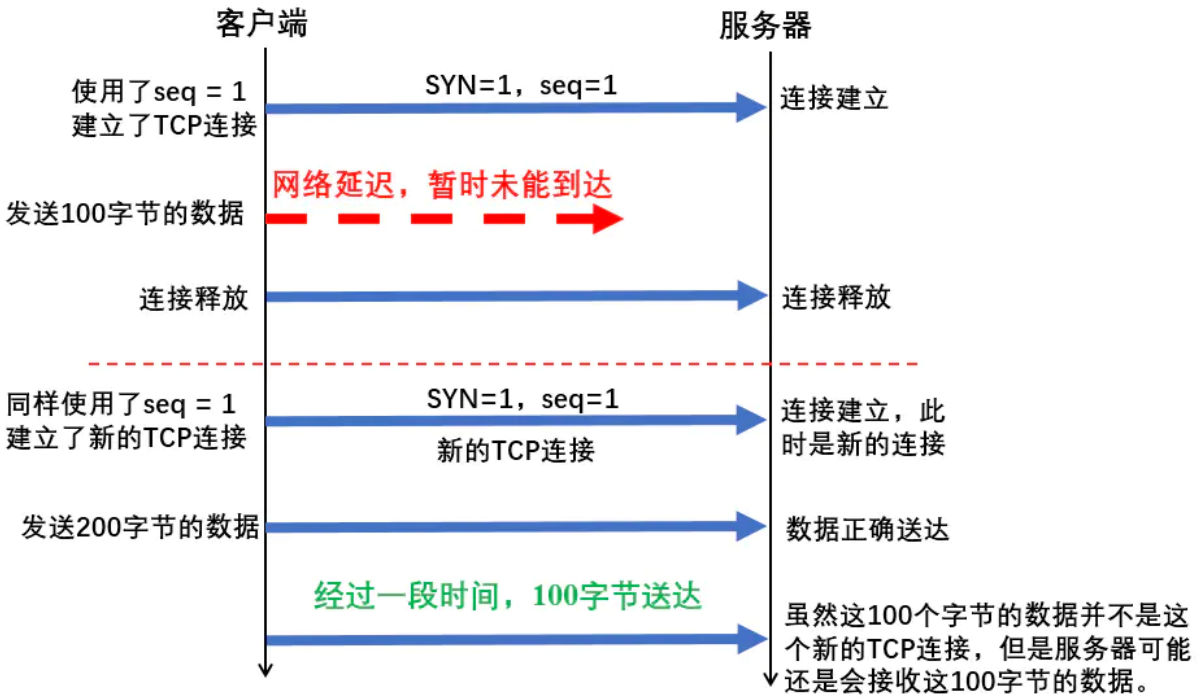
> TCP标准规定，ACK报文段可以携带数据，但是如果不携带数据则不消耗序号，在这种情况下，下一个数据报文段的序号仍是 seq = x + 1。

> 一旦完成上述三次握手，客户服务器就可以相互发送包括数据的报文段了。

2.三次握手的几个问题

2.1 为什么在三次握手的过程中要初始化序列号，为什么要使用随机序号，而不能使用固定的序号?

这样选择序号的目的是为了**防止由于网络路由TCP报文段可能存在延迟抵达与排序混乱的问题，从而而导致某个连接的一方对它作错误的解释。**



> 从上图可以看出，如果客户服务器双方建立连接使用固定的序号，在经过三次握手后建立了一个TCP连接，在传输数据时有网路问题而导致数据未能到达，这个报文段在网络中停留。

> 之后，由于某些原因（如客户端主机故障）导致这个TCP连接被释放，等到客户端主机恢复后，使用相同的序列号重新建立一个连接时，在之后的某个时间段，如果之前由于网络问题的报文段达到服务器端，那么服务器就可能会收下这个报文段，而这个数据是属于之前的旧连接的数据，所以这就会导致数据乱序的问题。

> 由于一个TCP连接是被一对端点所表示的，其中包括2个IP地址和2个端口号构成的4元组，因此即便是同一个连接也会出现不同的实例，如果连接由于某个报文段长时间延迟而关闭，然后又以相同的4元组被重新打开，那么可以相信延迟的报文段又会被视为有效据重新进入新连接的数据流中，这就会导致数据乱序问题。

为了避免上述的问题，**避免连接实例间的序号重叠可以将风险降至最低。**

并且，一个TCP报文段只有同时具备连接的4元组与当前活动窗口的序列号，才会在通信过程中被对方认为是正确的。然而，这也反应了TCP连接的脆弱性：如果选择合适的序列号、IP地址和端口号，那么任何人都能伪造一个TCP报文段，从而打断TCP的正常连接。所以使用初始化序号的方式（通常随机生成序号）使得序列号变得难猜，或者使用加密来避免利用这种缺点被攻击。

所以，**在发送用于建立连接的SYN之前，通信双方都会选择一个初始序列号。初始序列号会随着时间的变化而改变（每4ms加1），这样的目的在于为一个连接的报文段安排序列号，以防止出现与其他连接的序列号重叠的情况。尤其是对于同一连接的两个不同实例而言，新的序列号也不能出现重叠的情况。**

对于上例，如果同一个连接再次建立连接实例后，由于两次使用的是不同的序列号，所以服务器在接收到这个因网络延迟的数据报文段时，发现这个TCP报文段的序号不再新的TCP连接的接收范围之内，就会把这个报文段丢失，也就避免了数据乱序的问题。

所以，可以明白在建立TCP连接时，客户端和服务器端初始化序列号，就避免了上述的问题。前面说过，TCP序号占32位，范围是0~2^32 - 1，并且可以重用。

2.2 为什么客户端第一次握手不能携带数据而第三次握手可以携带数据?

假如 第一次握手可以携带数据的话，如果有人使用伪TCP报文段恶意攻击服务器，那么每次都在第一次握手中的SYN报文中携带大量的数据，因为它不会理会服务器的发送和接收能力是否正常，不断地给服务器重复发送这样携带大量数据的SYN报文，这会导致服务器需要花费大量的时间和内存来接收这些报文数据，这会将导致服务器连接资源和内存消耗殆尽。

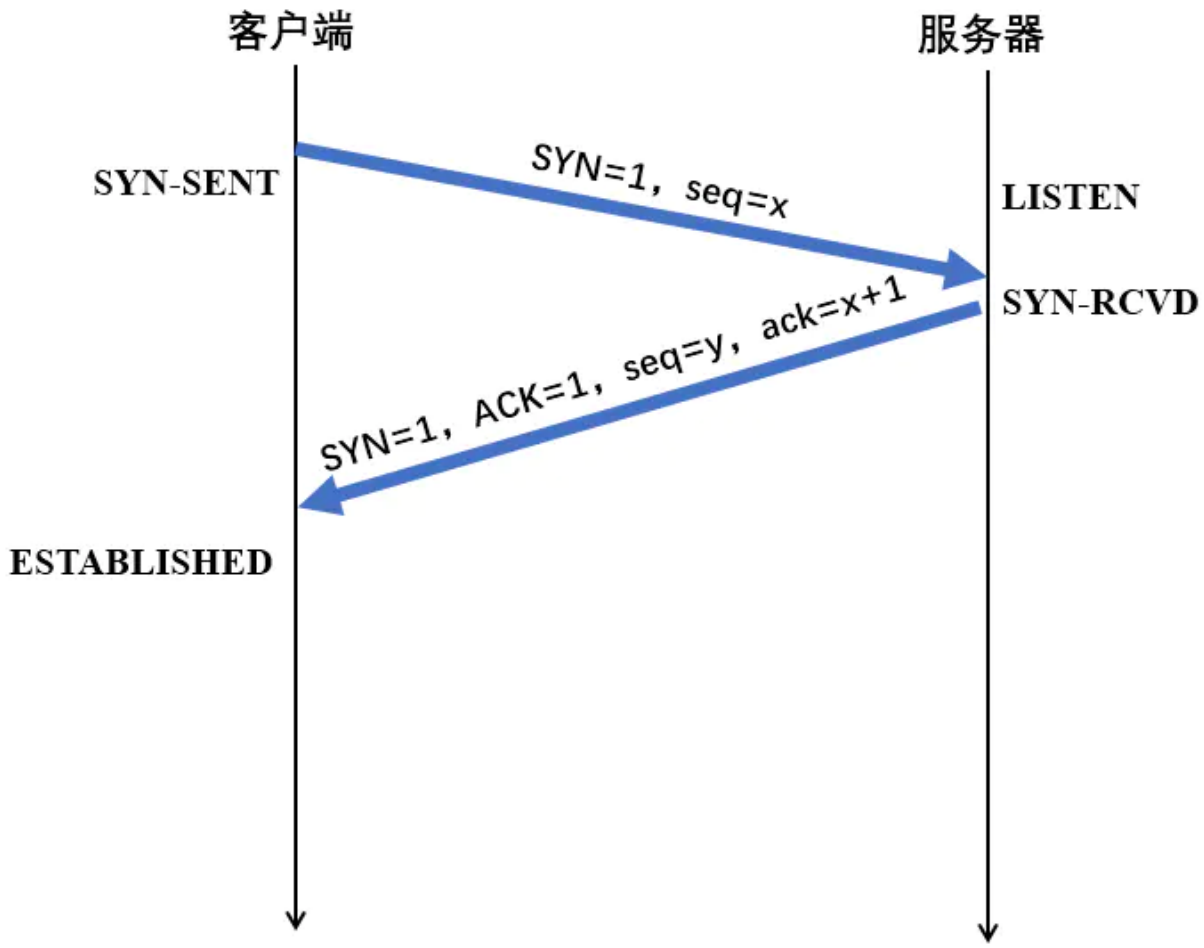
所以，之所以第一次握手不能携带数据，其中的一个原因就是避免让服务器受到攻击。而对于第三次握手，此时客户端已经建立了连接，通过前两次已经知道了服务器的接收正常，并且也知道了服务器的接收能力是多少，所以可以携带数据。

2.3 为什么连接建立需要三次握手，而不是两次握手?

根据前面描述，在第一次握手，客户端向服务发送建立连接请求，第二次握手，服务器同意建立连接，并向客户端返回一个确认报文，至此客户端已经知道了服务器同意建立连接，为什么客户端还需要对服务器的允许连接报文段进行确认?

第三个ACK报文段的目的简单来说主要是为了**实现可靠数据传输。**

三次握手的目的不仅在于让通信双方了解一个连接正在建立，还在于利用数据包的选项来承载特殊的信息，交换初始序列号（Initial Sequence，ISN）。为了实现可靠传输，TCP协议通信双方，都必须维护一个序列号，以标识发送出去的数据报中，哪些是已经被对方收到的。三次握手的过程是通信双方想要告知序列号起始值，并确认已经收到序列号的必经过程。



如上图，在两次握手过程中，通信双方都随机选择了自己的初始段序号，并且第二次握手的时候客户端收到了自己的确认序号，确认了自己的序列号，而服务器端还没有确认自己的序列号，没有收到确认序号， 如果这时候两次握手下就进行数据传递， 序号没有同步，数据就会乱序。即如果只是两次握手，最多只有客户端的起始序列号能被确认，而服务器断的序列号则得不到确认。

> 简单的来说：
>
> 1)客户端在发出第一次报文段后，它既不知道自己的发送能力和服务器方的接收能力是否正常。
>
> 2)在收到服务器端的确认报文后，客户端知道了自己的发送能力正常（因为服务器收到了自己的请求报文并返回了确认报文了）和服务器的发送能力正常。而服务器知道客户端的发送能力没有问题，但是服务器不知道客户端的接收能力和自己的发送能力是否正常。
>
> 如果只有这两次握手，对于服务器来说还不知道它和客户端之间的通信是否可靠。而如果是三次握手时，服务器收到了客户端的确认后，服务器就知道了自己的发送能力和客户端的接收能力也都没有问题，即此时对于通信双方都知道对方及各自的之间的通信是没有问题的。

2.4 SYN 洪泛攻击以及如何解决SYN 洪泛攻击

在三次握手的过程中，服务器为了响应一个受到的SYN报文段，会分配并初始化连接变量和缓存，然后服务器发送一个SYNACK报文段进行响应，并等待客户端的ACK报文段。如果客户不发送ACK来完成该三次握手的第三步，最终（通常在一分多钟之后）服务器将终止该半开连接并回收资源。

这种TCP连接管理协议的特性就会有这样一个漏洞，攻击者发送大量的TCP SYN报文段，而不完成第三次握手的步骤。随着这种SYN报文段的不断到来，服务器不断为这些半开连接分配资源，从而导致服务器连接资源被消耗殆尽。这种攻击就是**SYN泛供攻击。**

为了应对这种攻击，现在有一种有效的防御系统，称为SYN cookie。SYN cookie的工作方式如下：

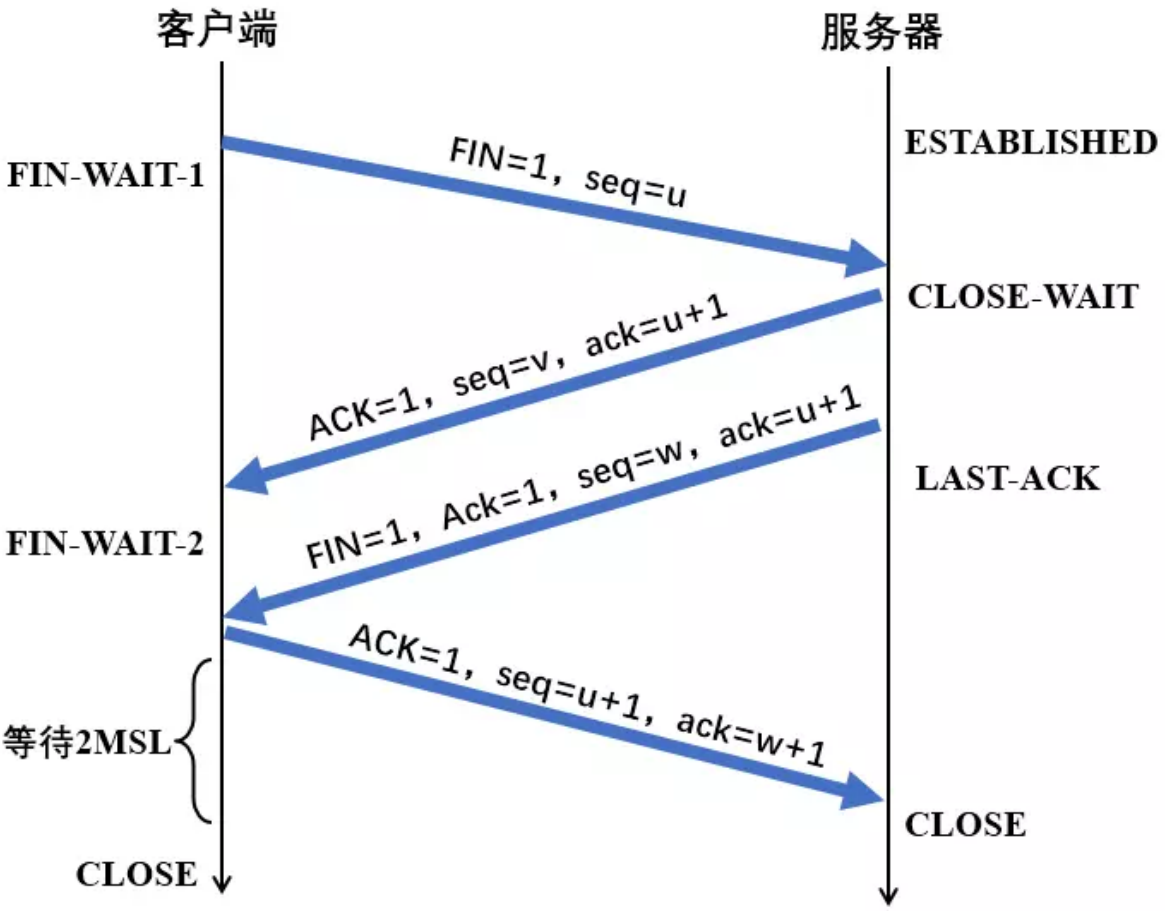
> (1) 当服务器接收到一个SYN报文段时，它并不知道该报文段是来自一个合法的用户，还是这种SYN洪泛攻击的一部分。因为服务器不会为该报文段生成一个半开的连接。相反，服务器生成一个初始TCP序列号，该序列号是SYN报文段的源IP地址和目的IP地址，源端口号和目的端口号以及仅有服务器知道的秘密数的复杂函数（散列函数）。这种精心制作的初始序列号称为为“cookie”。服务器则发送具有这种特殊初始序号的SYNACK报文分组。**服务器并不记忆该cookie或任何对应于SYN的其他状态信息。**

> (2) 如果该客户是合法的，则它将返回一个ACK报文段。当服务器收到该ACK报文段，需要验证该ACK是与前面发送的某个SYN相对应。由于服务器并不维护有关SYN报文段的记忆，所以服务器通过使用SYNACK报文段中的源和目的IP地址与端口号以及秘密数运行相同的散列函数。如果这个函数的结果（cookie值）加1和在客户的ACK报文段中的确认值相同的话，那么服务器就会认为该ACK对应于较早的SYN报文段，因此它是合法的。服务器则会生成一个套接字的全开连接。

> (3) 另一方面，如果客户没有返回一个ACK报文段，说明之前的SYN报文段是洪泛攻击的一部分，但是它并没有对服务器产生危害，因为服务器没有为它分配任何资源。

3.释放连接——四次挥手

连接释放的四次挥手过程如下图所示：



(1) **第一次挥手**：客户端应用进程先向服务器端发出**连接释放报文段**，并停止发送数据，主动关闭连接。其中报文段首部的终止控制为FIN = 1，其序号为seq = u（其值等于前面已传送的数据的最后一个字节的序号加1），此时客户端进入**终止等待1**（FIN-WAIT-1）状态，等待服务器端的确认。

(2) **第二次挥手**：服务器收到连接释放报文段后即发出确认，确认为ACK = 1，确认号为ack = u + 1，序号seq = v（其值是服务器前面已传送过的数据最后一个字节的序号加1），然后服务器就进入了**关闭等待（CLOSE-WAIT）状态。**

> 这时TCP连接处于**半关闭状态**，即客户端到服务器端这个方向的连接就释放了，即客户端没有数据要发送了，但是如果服务器端要发送数据，客户端仍要接收。

> 客户端在收到服务器端的确认后，就进入**终止等待2（FIN-WAIT-2）状态**，等待服务器发出的连接释放报文段。

(3) **第三次挥手**：如果此时服务器没有数据要发送了，此时服务器向客户端发出**连接释放报文段**，其FIN = 1，假设器序号为seq = w（在半关闭状态下服务器可能又发送了一些数据），服务器必须重复上次以发送的确认号ack = u + 1（因为客户端没有向服务器发送过数据，所以确认号和上次一致）。这时，服务器进入**最后确认**（LAST-ACK）状态，等待客户端的确认。

(4) **第四次挥手**：客户端在收到服务器端发出的连接释放报文段后，必须对此发出确认，在确认报文段中将ACK置位1，确认号ack = w + 1，而自己的序号为seq = u + 1。之后客户端进入**时间等待（TIME-WAIT）状态。**在经过**时间等待计时器**设置的时间2MSL后，客户端才进入**关闭（CLOSE）状态**

> 只要服务器收到客户端发出的确认报文就进入关闭（CLOSE）状态。

MSL：最大报文段寿命（Maximum Segment Lifetime)

4.四次挥手的几个问题

4.1 为什么客户端需要等待2MSL时间后，才能进入关闭状态?

这是为了**保证客户端发送的最后一个ACK报文段能够到达服务器端。**

客户端发送的ACK报文段可能丢失，因而使服务器收不到对自己已发送的释放连接报文段的确认。服务器会重传连接释放报文段，重新启动2MSL计时器，最终，客户端和服务器端都能进入CLOSE状态。

> 如果客户端在TIME-CLOSE状态不等待而在发送完ACK报文端后直接释放连接，那么在ACK报文段丢失后就无法收到服务器重传的释放连接报文段，因而也不会再发送一次ACK报文段，这样服务器就不能正常进入CLOSE状态。

4.2 为什么建立连接需要三次握手而释放连接需要四次挥手?

在建立连接时，服务器端处于LISTEN状态时，当收到SYN报文段的建立连接请求后，它可以把ACK报文段和SYN报文段（ACK报文段起确认作用，即确认客户端的连接建立请求；SYN报文段起同步作用）放在一起发送，所以在连接建立时四次握手（即第二次握手时，服务器的ACK报文段和SYN报文段分开发送）可以合并为三次握手。

而在释放连接时需要四次是因为TCP连接的半关闭造成的。由于TCP是全双工的（即数据可在两个方向上同时传递），因此，每个方向都必须要单独进行关闭，这个单方向的关闭就叫半关闭。在关闭连接时，当服务器收到客户端的FIN报文通知时，它仅仅表示客户端没有数据发送服务器了；但服务器未必将所有的数据都全部发送给了客户端，所以服务器端未必马上也要关闭连接，也即服务器端可能还需要发送一些数据给客户端之后，再发送FIN报文给客户端来表示现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的，这也是为什么释放连接时需要交换四次报文了。