TCP是指传输控制协议（Transmission Control Protocal），是面向连接的协议，为用户进程提供可靠的全双工字节流，TCP套接字是一种流套接字。TCP关心确认、超时、重传等细节，广泛应用于互联网应用程序中。

**1、TCP的特点**

TCP提供客户与服务器之间的连接，TCP客户先与指定服务器建立一个连接，再跨越该连接与服务器进行交换数据，之后终止连接。

1. 可靠性

当TCP向另一端发送数据时，要求对端返回一个确认。如果没有收到确认回复，TCP就自动重传数据并等待更长时间以接收确认回复。在多次重传失败后放弃并断开连接，根据具体实现可能等待1~10分钟。

1. 有序性

TCP通过给每个字节关联一个序列号对发送的数据进行排序。例如，应用发送2048字节数据到TCP套接字，用两个分节发送，第一个分节包含序号1~1024，第二个分节包含序列号1025~2048。若分节数据不是按序到达，接收端TCP会先根据序列号排序再传递给应用程序。又如因为网络通信拥挤对端认为一个分节丢失重传但这个分节并没有丢失，可以根据序列号判定数据重复而丢弃重复数据。

1. 流量控制

TCP总是对端在任何时刻能够从对端接受多少字节的数据，也就是通知窗口。该窗口指示当前接受缓冲区可用的空间量，确保发送端发送的数据不会使接收缓冲区溢出。这个窗口是动态变化的：接收来自发送端的数据后窗口大小就减小，接收端应用从缓冲区读取数据后窗口大小就增大。窗口大小也可能减小到0，当接收区缓冲区已满，导致必须等待应用程序从缓冲区读取数据时，才能再从对端接收数据。

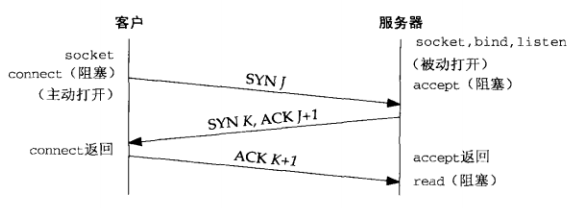
1. 全双工

在给定的一个连接上，应用可以在任何时刻在进出两个方向上即发送数据又接收数据。因此，TCP必须为每个数据流方向跟踪诸如序列号、通告窗口大小等状态信息。建立一个全连接双工后，可通过shutdown函数关闭读或者写使其变成单工连接。

**2、TCP连接的建立和终止**

理解connect()、accept()和close()三个函数，了解TCP连接如何建立和终止，了解TCP的状态转换图。

**2.1 连接建立**



1. 服务器准备好接收外来的连接。通常服务器调用socket、bind和listen函数来完成，成为被动打开
2. 客户端通过调用connect发起主动打开，客户端TCP会发送一个SYN（同步）分节，告诉服务器将在（待建立）连接中发送数据的初始序列号J。通常SYN分节不携带数据，其所在的IP数据报文中只包含IP首部、TCP首部和可能有的TCP选项。
3. 服务器必须确认（ACK）客户的SYN，同时也发送一个SYN，它含有服务器将在同一连接中发送的数据的初始序列号K。服务器在单个分节中发送SYN和对客户的SYN的ACK。
4. 客户必须确认服务器的SYN。

这种交换至少需要(2)(3)(4)三个步骤的分节数据，因此称之为TCP的**三次握手**。**每个SYN的ACK中的确认号，就是该SYN的上一次序列号加1**，例如（2）中客户端发给服务端的SYN的序列号为J，接着（3）中服务端ACK的序号就是J+1。同时，服务端回复ACK的同时发送SYN且序列号为K，接着（4）客户端回复ACK的序列号就是K+1。

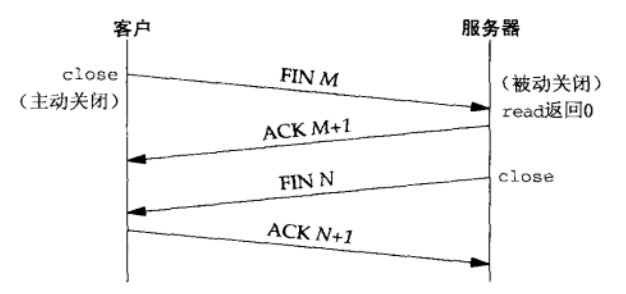
**2.2 TCP选项**

每一个SYN可以含有多个TCP选项。常用的TCP选项有：

1. MSS选项。发送端使用该选项通知对端它的最大分节大小，即MSS，也就是它在本连接的每个TCP分节中愿意接受的最大数据量。发送端TCP使用接收端的MSS值作为所发送分节的最大大小。对应套接字中的选项为TCP\_MAXSEG。
2. 窗口规模选项。TCP连接任何一段能通知对端的最大窗口是65535，因为在TCP首部中对应的字段占16位。对于高速线路或长延迟路径要求更大窗口获取尽可能大的吞吐量。这个新选项在指定TCP首部中通知窗口必须扩大（左移）的位数（0~14），因此提供最大窗口接近1GB（65535\*2^14）。使用规模选项的前提是两个端系统都支持这个选项。对应套接字中的选项为TCP\_REVBUF
3. 时间错选项。这个新选项对应高速网络是必要的，可以防止由于路由问题造成的数据损坏。我们不用考虑这个问题。

**2.3 TCP连接终止**

TPC建立一个连接需要3个分节，终止一个连接则需要4个分节，成为**“四次挥手”。**



1. 某个客户端首先调用close函数执行主动关闭，会发送一个FIN分节，表示数据发送完毕。
2. 服务器接收这个对端的FIN分节执行被动关闭（read返回0）。这个FIN由TCP确认。它的接收也作为一个文件结束符传递给接收端应用进程，也意味着在这个连接上不会有额外的数据可以接收。但是，服务端仍可以向客户端发送数据，称为**半关闭**状态。
3. 一段时间后，接收到这个文件结束符的应用进程调用close()关闭套接字，导致服务端TCP也会发送一个FIN分节给客户。
4. 客户端收到这个最终的FIN会确认ACK给服务端。

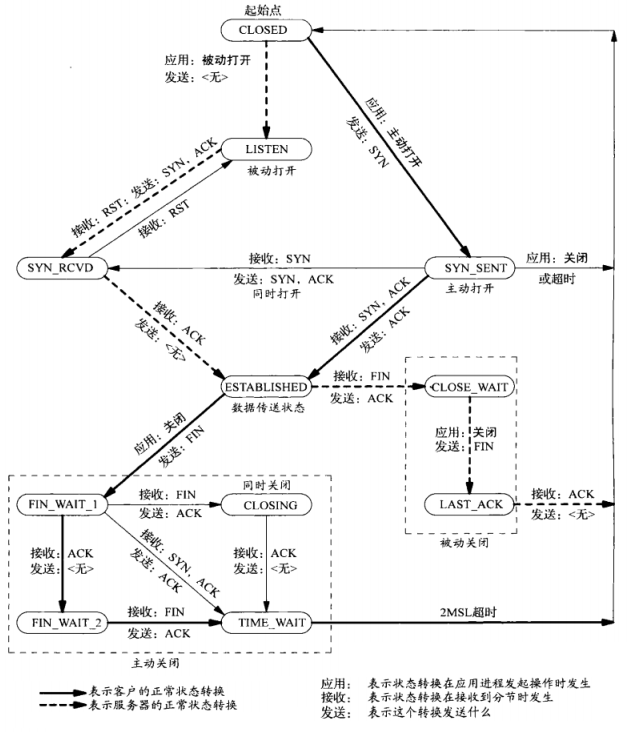
说通常有4个分节是因为第一个FIN可能和用户数据一起发送，(2)(3)中的2个分节都是由服务端发送，可能合并成一个分节。类似SYN，一个FIN也占据一个字节的序列号空间，因此每个FIN的ACK去人号就是这个FIN的序号加1。

这里的连接终止的情况，是客户端执行主动关闭，实际上，无论是客户还是服务端，任何一端都可以执行主动关闭。通常情况是客户端执行主动关闭（HTTP/1.0是由服务端主动关闭）。

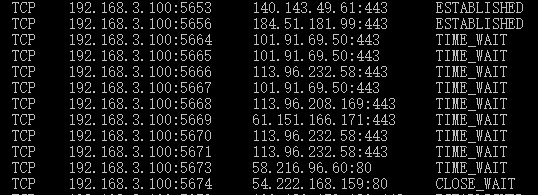
**2.4 TCP状态转换图**

TCP连接的建立和终止的操作可以用状态转换图来说明。TCP规定了一个连接的11种状态，并规定了如何基于当前状态下接受分节从一个状态切换到另一个状态的规则。例如，某个进程在CLOSED状态下执行主动打开，TCP将发送一个SYN，且新的状态是SYN\_SEND。如果这个TCP接着接收到一个带ACK的SYN，它将发送一个ACK，且新的状态是ESTABLISHED。这个最终状态是绝大多数数据传送发生的状态。

自ESTABLISHED状态引出两个箭头处理连接的终止。如果进程在接收到一个FIN之前调用close(主动关闭)，那就转换到FIN\_WAIT\_1状态。如果在ESTABLISHED状态期间接收到一个FIN（被动关闭），那就转换到CLOSE\_WAIT状态。



使用netstat可以查看连接状态，例如



**2.5 TIME\_WAIT状态**

在上面的状态转换图中，**当一个TCP连接建立后，发起主动关闭的一端，收到对方FIN分节回复ACK确认后，会进入到TIME\_WAIT状态**，并在这种状态下停留时间时最长分节生命期（maximum segment lifetime，MSL）的两倍，记为2MSL。

在实际应用中，高并发情况下有很多短连接，这是正常的。但是，每一个处于TIME\_WAIT状态的TCP连接都会占用一个本地端口，而本地端口上限是65535（16bit），可能会造成新建TCP连接时出现address already in use的异常。

可能存在的解决方案有很多。应用上，在HTTP请求头部设置connnect为close，服务端就会在处理完请求后主动关闭TCP连接；现在的浏览器中，一般设置为keep-alive，延长其使用周期。客户端编码时，可以设置允许TIME\_WAIT状态的socket重用；缩短TIME\_WAIT的时间，如1 MSL(2分钟)。

当然，TIME\_WAIT的存在也是有必要的：

1. 可靠的实现TCP全双工连接的终止。四次挥手关闭TCP 连接过程中，最后的ACK是由主动关闭的一端发出的，如果这个 ACK 丢失，则对方会重发 FIN 请求，因此，在主动关闭连接的一端，需要维护一个TIME\_WAIT状态，处理对方重发的 FIN 请求；
2. 处理延迟到达的报文。由于路由器可能抖动，TCP 报文会延迟到达，为避免被误认为是新TCP连接的数据，则需要在允许新创建 TCP 连接之前，保持一个不可用的状态，等待所有延迟报文的消失，一般设置为 2MSL，解决延迟达到的 TCP 报文问题；