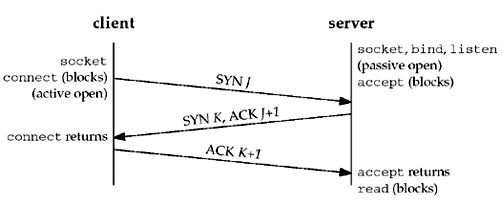
TCP状态转换

# 1 建立连接(CLOSE -> ESTABLISHED)



在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。 第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认； 第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态； 第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。 完成三次握手，客户端与服务器开始传送数据.

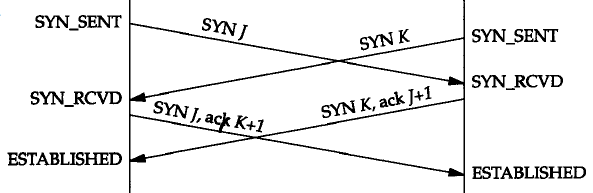
## 1.1 被动打开

服务器在CLOSE状态下执行被动打开时，进入到LISTEN状态，等待可能到来的连接。接收到SYN报文后，发送SYN+ACK报文，然后进入到SYN\_RCVD状态，等待对方回应。当收到ACK报文后进入到ESTABLISHED状态。

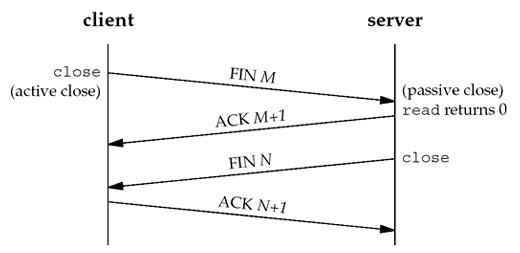
## 1.2 主动打开

客户端在CLOSE状态下执行主动打开时，将发送一个SYN，然后进入到SYN\_SENT状态。接收到SYN+ACK报文后进入到ESTABLISHED状态。

## 1.3 同时打开



# 2 关闭连接(ESTABLISHED -> CLOSE)



由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

　（1） TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送（报文段4）。

　（2） 服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1（报文段5）。和SYN一样，一个FIN将占用一个序号。

　（3） 服务器关闭客户端的连接，发送一个FIN给客户端（报文段6）。

　（4） 客户段发回ACK报文确认，并将确认序号设置为收到序号加1（报文段7）。

## 2.1 主动关闭

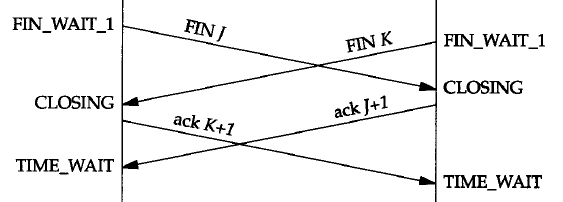
在ESTABLISHED状态时，发送FIN报文，进入到FIN\_WAIT\_1状态，等待对方ACK报文。收到ACK后进入到FIN\_WAIT\_2状态。此时连接的前半部分关闭。当收到对方的FIN报文并发送ACK后进入到TIME\_WAIT状态，2MSL超时后进入CLOSE状态。收到本方应用进程的关闭命令后， TCP 在发送完尚未处理的报文段后，发 FIN ＝ 1 的报文段给对方，且 TCP 不再受理本方应用进程的数据发送。在 FIN 以前发送的数据字节，包括 FIN ，都需要对方确认，否则要重传。注意 FIN 也占一个顺序号。一旦收到对方对 FIN 的确认以及对方的 FIN 报文段，本方 TCP 就对该 FIN 进行确认，在等待一段时间，然后关闭连接。等待是为了防止本方的确认报文丢失，避免对方的重传报文干扰新的连接。

## 2.2 被动关闭

当 TCP 收到对方发来的 FIN 报文时，发 ACK 确认此 FIN 报文，并通知应用进程连接正在关闭，此时状态由ESTABLISHED变为CLOSE\_WAIT状态。应用进程将以关闭命令响应。 TCP 在发送完尚未处理的报文段后，发一个 FIN 报文给对方 TCP ，此后状态变为LAST\_ACK，然后等待对方对 FIN 的确认，收到确认后关闭连接。若对方的确认未及时到达，在等待一段时间后也关闭连接。最终进入到CLOSE状态。

## 2.3 双方同时启动关闭

如果应用程序同时发送FIN，则在发送后会首先进入FIN\_WAIT\_1状态。在收到对端的FIN后，回复一个ACK，会进入CLOSING状态。在收到对端的ACK后，进入TIME\_WAIT状态。2MSL超时后进入CLOSE状态。



# 3三次与四次握手

服务端LISTEN状态下的SOCKET，收到SYN报文的连接请求后，它可以把ACK和SYN（ACK起应答作用，而SYN起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意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。

# 4 TIME\_WAIT状态

1）可靠地实现TCP全双工连接的终止

TCP协议在关闭连接的四次握手过程中，最终的ACK是由主动关闭连接的一端（后面统称A端）发出的，如果这个ACK丢失，对方（后面统称B端）将重发出最终的FIN，因此A端必须维护状态信息（TIME\_WAIT）允许它重发最终的ACK。如果A端不维持TIME\_WAIT状态，而是处于CLOSED 状态，那么A端将响应RST分节，B端收到后将此分节解释成一个错误（在java中会抛出connection reset的SocketException)。

因而，要实现TCP全双工连接的正常终止，必须处理终止过程中四个分节任何一个分节的丢失情况，主动关闭连接的A端必须维持TIME\_WAIT状态 。

2）允许老的重复分节在网络中消逝

TCP分节可能由于路由器异常而“迷途”，在迷途期间，TCP发送端可能因确认超时而重发这个分节，迷途的分节在路由器修复后也会被送到最终目的地，这个迟到的迷途分节到达时可能会引起问题。在关闭“前一个连接”之后，马上又重新建立起一个相同的IP和端口之间的“新连接”，“前一个连接”的迷途重复分组在“前一个连接”终止后到达，而被“新连接”收到了。为了避免这个情况，TCP协议不允许处于TIME\_WAIT状态的连接启动一个新的可用连接，因为TIME\_WAIT状态持续2MSL，就可以保证当成功建立一个新TCP连接的时候，来自旧连接重复分组已经在网络中消逝。

