TCP 状态转换

TCP 状态转换在在各种技术面试中出现的概率实在太高，于是今天好好看了一下，感觉这块知识还是很有用的，尤其是对于开发而言，通过 TCP 各种状态来进行故障推断非常有用。

TCP 状态转换单独去看 TCP 状态转换图相信一定是一头雾水，所以我觉得深入理解首先必须对 TCP 知识有一定的了解，至少对 TCP 首部中序号 seq、确认号 ack 以及 SYN 位、ACK 位和 FIN 位的作用非常熟悉。之后通过学习 TCP 连接建立和断开时“三次握手、四次挥手”大体了解整个断开和连接过程。最后结合网络编程构建客户端和服务器数据交流的函数调用具体实例就能比较好把握这个知识点了。

既然是说 TCP 状态转换，首先得知道在 TCP 连接中具体有多少种不同状态，之后我们再结合“三次握手，四次挥手”来具体说明每个状态对应的哪一个阶段。TCP 的状态氛有十一种，分别对应 SYN_SENT，ESTABLISHED，FIN_WAIT_1，FIN_WAIT_2，TIME_WAIT，CLOSING，LISTEN，SYN_RCVD，CLOSE_WAIT，LAST_ACK 和 CLOSED。为了方便接下来的介绍，我们将以最常见的 C/S 模型来表示 TCP 连接释放过程，服务器在特定端口监听，客户端向服务器发起连接请求，传输数据结束后客户端主动向服务器端发起断开连接请求。在这个模型中，无论是连接还是断开，客户端都是主动的一方，服务器端都是被动的一方。在这个模型中，属于客户端一侧的状态有 CLOSED，SYN_SENT，ESTABLISHED，FIN_WAIT_1，FIN_WAIT_2，CLOSING 和 TIME_WAIT。属于服务器端的状态有 CLOSED，LISTEN，SYN_RCVD，CLOSE_WAIT 和 LAST_ACK。

特意将 TCP 状态分为客户机和服务器两大类，这样在之后的讲解中会比较容易些。有过网络编程经验的人都知道客户端程序和服务器端程序具体需要调用哪些函数，二者的区别和联系又在哪里。但是对于没有网络编程经验的人来说，深刻理解还是很有困难的，有必要先显式地区分二者及其可能出现的状态。

首先从最基本的“三次握手”和“四次挥手”介绍起，详细如下图。我们常说 TCP 协议是全双工的有连接的可靠的协议，全双工表示通信双方可以同时发送数据，可靠主要是因为 TCP 中的窗口机制和序号机制，在接下来的学习中会少部分接触到序号的知识。最后要说的就是有连接的，所谓“有连接”就是指在正式传输数据之前需要建立起客户端和服务器端的“确信关系”，确认之后才能相互交流。

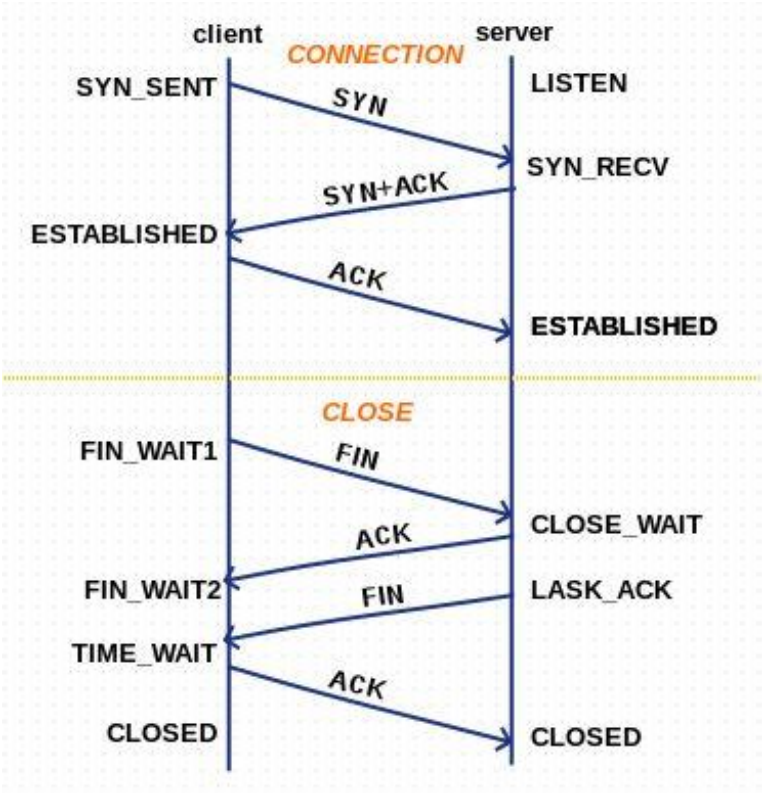

图 1：三次握手、四次挥手

首先是建立连接的过程，客户机向服务器发送连接请求，请求建立起二者的连接。这个连接请求的报文段由客户端发送，如果单单看报文段的TCP首部的话会发现其SYN位为 1，并且序号部分带有一个数字。这里的 SYN 位就是请求连接的位（也可以理解为同步位），并且客户端发送的这个报文段使用了一个序号 seq = x，至于序号的作用是什么，在接下来服务器的回复中会有说明。

接下来，当服务器接收到客户端请求连接的报文段之后立即给客户端发送回一个数据报，数据报中 SYN = 1，ACK = 1，ack = x + 1，seq = y。这里的 SYN 和上面的意思一样，也表示建立连接，不过服务器端发出的 SYN 更贴切的理解应该是接受连接并给客户端的一个确认。这里 ACK 为可以理解为是 ack 请求序号的使能位，只有 ACK 为 1 时候，ack 的结果才是有效的。而 ack 的值则是对之前客户端发送的报文段已经被准确接收的一种确认，客户端发来的 seq 表示发送数据的“首地址”，比如这里是 1000，并且客户端携带了 200 字节的数据，那么服务器端将会接收到字节序为[1000, 1199]的数据，那么接下来下一个字节的字节序就是 1200，也就是 ack 的值。因为这里建立连接时的第一条请求是不包含具体数据的，所以服务器期望得到的下一个字节序就是 x + 1。另外，服务器端依旧需要消耗掉一个序号发给客户端用作下一步确认，这里是 seq = y，至于 y 的值具体是多少并没有要求。

之后客户端对接收到的数据中 seq 的数据进行确认即可。完成此确认工作需要令 ACK 的值为 1 并且 ack = y + 1。这里来回确认的目的就是为了告知对方自己已经准确接收到了你的信息。当这个数据报被服务器接收到时，双方的 TCP 连接就算是正式建立起来了，此时双方可以进行数据传输。

断开连接时依旧是客户机处于主动状态。当客户机想要断开连接，就会发送 FIN 位为 1 的报文段给服务器，同时 ACK 位也为 1，假设 seq = u，ack = v。

之后服务器当收到这个断开连接报文会立即发送 ACK 报文给客户机，对之前的 seq 进行确认，回复的 TCP 首部中 ACK 位为 1，ack = u + 1。

一般这之后，服务器会确认是否还有别的数据需要发送给客户机，所以这里将会有短暂的时间间隔。在确认完之后，服务器会在此发送一个报文段给客户机，其中 FIN 位位被置为 1，且 seq = v。

最后一步操作是客户机对服务器的序号进行最后一次确认，最后一次确认的报文段中 ACK 位为 1，ack = v + 1。

所以，综合之前的过程，会发现连接过程中无论是服务器还是客户机都有两次 SYN 位为 1，ACK 位为 1，虽然它们出现的时机并不相同。同理，断开连接时二者也均有两次 FIN 位为 1 和两次 ACK 位为 1。

那么我们又怎么去理解 TCP 状态呢？

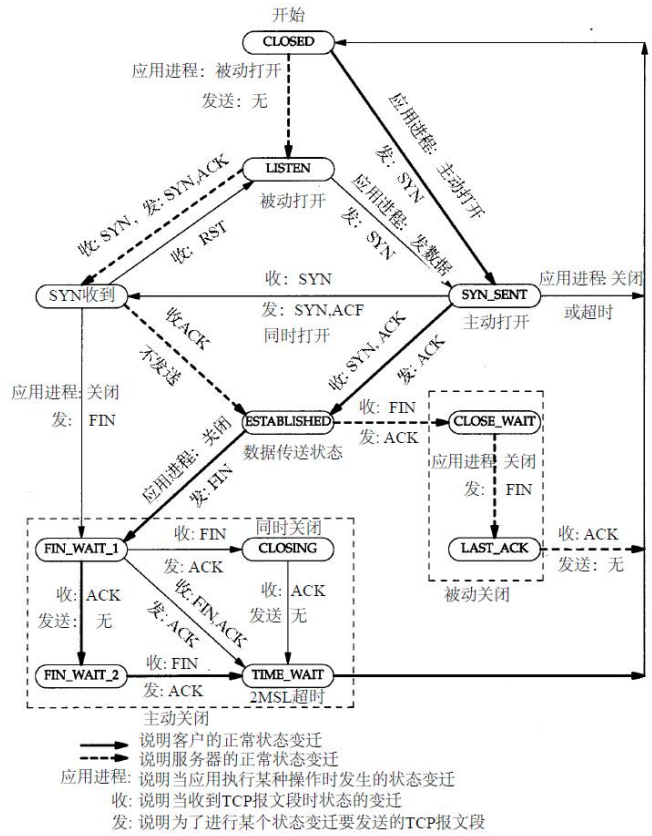

图 2：TCP 状态转换图

首先，TCP 的状态实际上和之前上述每次报文段的发送和接收有密切关系，每一次报文段的发送都会带来两端状态的改变。下面会按顺序描述各个状态。

CLOSED：起初，客户机和服务器都处于关闭状态。

LISTEN：服务器打开，并监听特定端口是否有连接，服务器处于 LISTEN 状态。

SYN_SENT：客户机向服务器发送连接请求，也就是对应三次握手中第一个报文段从客户机发出之后，客户机的状态立即变为 SYN_SENT。

SYN_RCVD：当客户机请求连接的报文被服务器接收，并且服务器发送完反馈的报文之后，服务器的状态变为 SYN_RCVD。

ESTABLISHED：客户机接收到服务器的数据后对收到的数据进行确认，之后变为 ESTABLISHED 状态。服务器也会在收到客户机的确认报文之后变为 ESTABLISHED 状态。

FIN_WAIT_1：当客户机打算断开连接，发送完带有 FIN 的报文之后，立即变为 FIN_WAIT_1 状态。

CLOSE_WAIT：服务器接收到客户机断开连接请求，立即立即向客户机发送确认报文确认已收到，之后自身变为 CLOSE_WAIT 状态。

FIN_WAIT_2：客户机收到服务器的确认之后变为 FIN_WAIT_2 状态。

LAST_ACK：服务器确认是否还有其他数据要发给客户机，没有的话发送带有 FIN 位的报文给客户机，随后变为 LAST_ACK 状态，等待客户机最后一个确认报文。

TIME_WAIT：客户机收到带有 FIN 位的报文并做确认，之后变为 TIME_WAIT 状态。

这里对于客户机而言在 TIME_WAIT 状态需要等待 2MSL，再次之后才能回复到 CLOSED 状态，而对于服务器而言，只有收到了最后一个确认的报文才能重新回到 CLOSED 状态（或者回到 LISTEN 状态）。

有两个问题值得思考：1.为什么握手是三次，挥手却要四次？2.为什么最后客户机在 TIME_WAIT 状态下需要等待 2MSL 才能进入 CLOSED 状态？

第一个问题的答案我们发现客户机的行为在连接和断开时没有什么太大区别，都是作为主动一方先发起连接、断开请求，之后在最后做一下确认。但是服务器端在连接时行为却不相同，在连接过程中，服务器端的 SYN 和 ACK 在同一个报文中发给了客户机，但是断开连接却先发 ACK，再发 FIN。主要原因是连接是一个第一次响应过程，在此之前，二者“毫无瓜葛”，服务器收到请求直接确认收到并响应连接即可。但是断开连接时却不能这样，有可能服务器还有尚需发送给客户机的数据，这时只能先回复 ACK 表示断开连接请求已收到，但具体服务器一侧什么时候断开还需要等数据传输结束，所以 FIN 会稍晚一些发送。

第二个问题的答案和网络的特性有关，因为我们不能保证客户机最后一次确认一定能送达服务器，所以当服务器没能收到最后一次 ACK 确认时会重新发送 FIN 报文请求重新确认，并且保持在 LAST_ACK 状态。所以客户机保持 2MSL 时间是为了防止服务器端因未收到确认而重发 FIN。

上面关于 TCP 状态的介绍已经很详细了，下面结合网络编程中调用的函数来理解这个问题会更容易理解一些。

当客户端一次执行到 connect 函数时，将会发送连接报文给服务器，也就是第一个带有 SYN 的报文，之后进入 SYN_SENT 状态。服务器端执行到 listen 函数之后阻塞自己并进入 LIETEN 状态，当执行完 accept 函数之后进入 ESTABLISHED 状态，之后双方通过套接字传输数据。当执行结束后，客户端调用 close 函数执行断开连接流程，服务器响应并执行到 close 之后继续进入 LISTEN 状态。

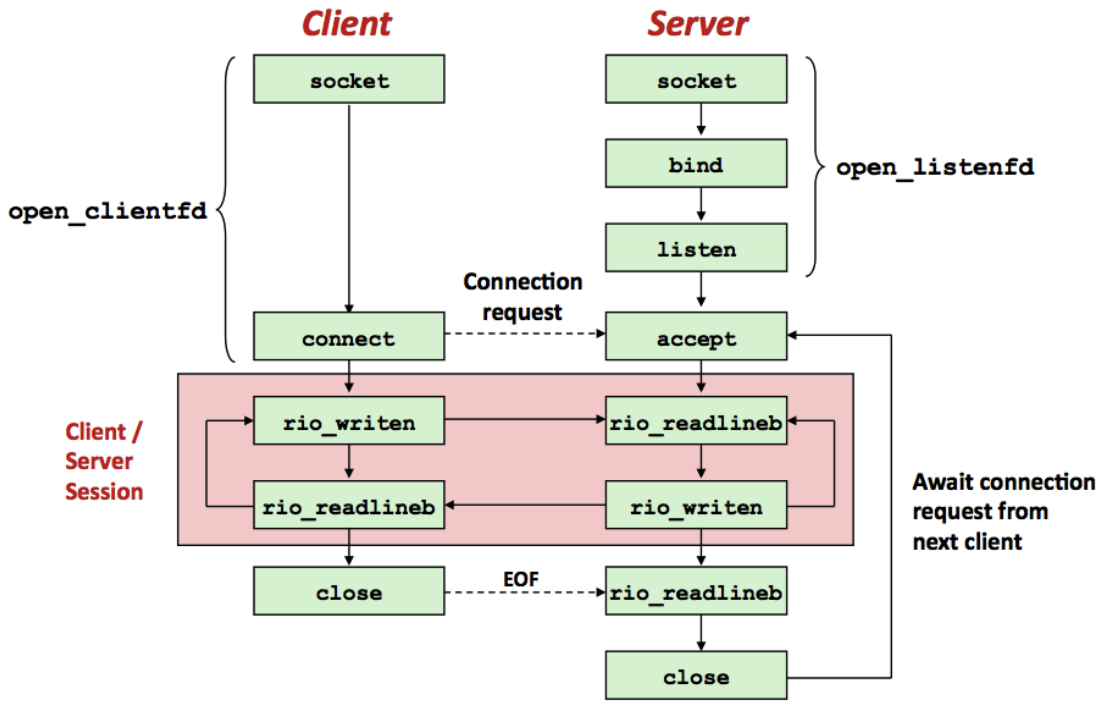

图 3：套接字接口概述

了解 TCP 状态可以帮助我们快速进行网络故障排插，在这里就不展开了，以后有机会会专门发一篇关于网络故障排查的文章。