拔掉网线后， 原本的 TCP 连接还存在吗？

程序员的那些事 2022-02-27 03:55

小林coding

专注图解计算机基础，让天下没有难懂的八股文！刷题网站：xiaolincoding.com

今天，聊一个有趣的问题：**拔掉网线几秒，再插回去，原本的 TCP 连接还存在吗？**

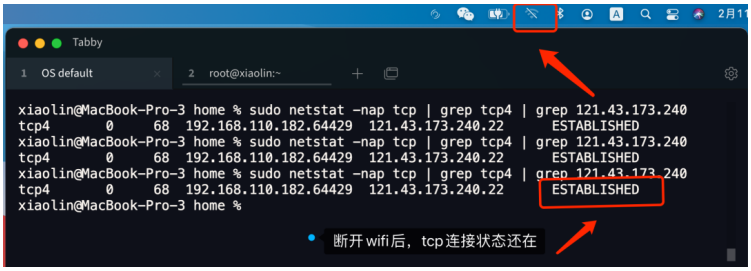
可能有的同学会说，网线都被拔掉了，那说明物理层被断开了，那在上层的传输层理应也会断开，所以原本的 TCP 连接就不会存在了。就好像， 我们拨打有线电话的时候，如果某一方的电话线被拔了，那么本次通话就彻底断了。

真的是这样吗？

上面这个逻辑就有问题。问题在于，错误地认为拔掉网线这个动作会影响传输层，事实上并不会影响。

实际上，TCP 连接在 Linux 内核中是一个名为 `struct socket` 的结构体，该结构体的内容包含 TCP 连接的状态等信息。当拔掉网线的时候，操作系统并不会变更该结构体的任何内容，所以 TCP 连接的状态也不会发生改变。

我在我的电脑上做了个小实验，我用 ssh 终端连接了我的云服务器，然后我通过断开 wifi 的方式来模拟拔掉网线的场景，此时查看 TCP 连接的状态没有发生变化，还是处于 ESTABLISHED 状态。



通过上面这个实验结果，我们知道了，拔掉网线这个动作并不会影响 TCP 连接的状态。

接下来，要看拔掉网线后，双方做了什么动作。

针对这个问题，要分场景来讨论：

- 拔掉网线后，有数据传输；
- 拔掉网线后，没有数据传输。

拔掉网线后，有数据传输

在客户端拔掉网线后，服务端向客户端发送的数据报文会得不到任何的响应，在等待一定时长后，服务端就会触发**超时重传**机制，重传未得到响应的数据报文。

如果在服务端重传报文的过程中，客户端刚好把网线插回去了，由于拔掉网线并不会改变客户端的 TCP 连接状态，并且还是处于 ESTABLISHED 状态，所以这时客户端是可以正常接收服务端发来的数据报文的，然后客户端就会回 ACK 响应报文。

此时，客户端和服务端的 TCP 连接依然存在，就感觉什么事情都没有发生。

但是，**如果在服务端重传报文的过程中，客户端一直没有将网线插回去**，服务端超时重传报文的次数达到一定阈值后，内核就会判定出该 TCP 有问题，然后通过 Socket 接口告诉应用程序该 TCP 连接出问题了，于是服务端的 TCP 连接就会断开。

而等客户端插回网线后，如果客户端向服务端发送了数据，由于服务端已经没有与客户端相同四元组的 TCP 连接了，因此服务端内核就会回复 RST 报文，客户端收到后就会释放该 TCP 连接。

此时，客户端和服务端的 TCP 连接都已经断开了。

> 那 TCP 的数据报文具体重传几次呢？

在 Linux 系统中，提供了一个叫 tcp_retries2 配置项，默认值是 15，如下图：

```
[root@xiaolin ~]#
[root@xiaolin ~]# cat /proc/sys/net/ipv4/tcp_retries2
15
[root@xiaolin ~]# _
```

这个内核参数是控制，在 TCP 连接建立的情况下，超时重传的最大次数。

不过 tcp_retries2 设置了 15 次，并不代表 TCP 超时重传了 15 次才会通知应用程序终止该 TCP 连接，内核还会基于「最大超时时间」来判定。

每一轮的超时时间都是倍数增长的，比如第一次触发超时重传是在 2s 后，第二次则是在 4s 后，第三次则是 8s 后，以此类推。

Retransmission #	RTO (ms)	Time before a timeout	
1	200	0.2 secs	0.0 mins
2	400	0.6 secs	0.0 mins
3	800	1.4 secs	0.0 mins
4	1600	3.0 secs	0.1 mins
5	3200	6.2 secs	0.1 mins
6	6400	12.6 secs	0.2 mins
7	12800	25.4 secs	0.4 mins
8	25600	51.0 secs	0.9 mins
9	51200	102.2 secs	1.7 mins
10	102400	204.6 secs	3.4 mins
11	120000	324.6 secs	5.4 mins
12	120000	444.6 secs	7.4 mins
13	120000	564.6 secs	9.4 mins
14	120000	684.6 secs	11.4 mins
15	120000	804.6 secs	13.4 mins
16	120000	924.6 secs	15.4 mins

内核会根据 tcp_retries2 设置的值，计算出一个最大超时时间。

在重传报文且一直没有收到对方响应的情况时，先达到「最大重传次数」或者「最大超时时间」这两个的其中一个条件后，就会停止重传，然后就会断开 TCP 连接。

拔掉网线后，没有数据传输

针对拔掉网线后，没有数据传输的场景，还得看是否开启了 TCP keepalive 机制 （TCP 保活机制）。

如果**没有开启** TCP keepalive 机制，在客户端拔掉网线后，并且双方都没有进行数据传输，那么客户端和服务端的 TCP 连接将会一直保持存在。

而如果**开启**了 TCP keepalive 机制，在客户端拔掉网线后，即使双方都没有进行数据传输，在持续一段时间后，TCP 就会发送探测报文：

- 如果**对端是正常工作**的。当 TCP 保活的探测报文发送给对端，对端会正常响应，这样 **TCP 保活时间会被重置**，等待下一个 TCP 保活时间的到来。
- 如果**对端主机崩溃，或对端由于其他原因导致报文不可达**。当 TCP 保活的探测报文发送给对端后，石沉大海，没有响应，连续几次，达到保活探测次数后，**TCP 会报告该 TCP 连接已经死亡**。

所以，TCP 保活机制可以在双方没有数据交互的情况，通过探测报文，来确定对方的 TCP 连接是否存活。

> TCP keepalive 机制具体是怎么样的？

这个机制的原理是这样的：

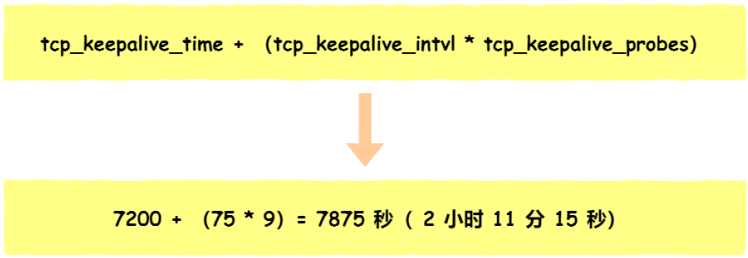
定义一个时间段，在这个时间段内，如果没有任何连接相关的活动，TCP 保活机制会开始作用，每隔一个时间间隔，发送一个探测报文，该探测报文包含的数据非常少，如果连续几个探测报文都没有得到响应，则认为当前的 TCP 连接已经死亡，系统内核将错误信息通知给上层应用程序。

在 Linux 内核有对应的参数可以设置保活时间、保活探测的次数、保活探测的时间间隔，以下都为默认值：

```
net.ipv4.tcp_keepalive_time=7200
net.ipv4.tcp_keepalive_intvl=75
net.ipv4.tcp_keepalive_probes=9
```

- tcp_keepalive_time=7200：表示保活时间是 7200 秒（2小时），也就 2 小时内如果没有任何连接相关的活动，则会启动保活机制；
- tcp_keepalive_intvl=75：表示每次检测间隔 75 秒；
- tcp_keepalive_probes=9：表示检测 9 次无响应，认为对方是不可达的，从而中断本次的连接。

也就是说在 Linux 系统中，最少需要经过 2 小时 11 分 15 秒才可以发现一个「死亡」连接。



注意，应用程序若想使用 TCP 保活机制，需要通过 socket 接口设置 `SO_KEEPALIVE` 选项才能够生效，如果没有设置，那么就无法使用 TCP 保活机制。

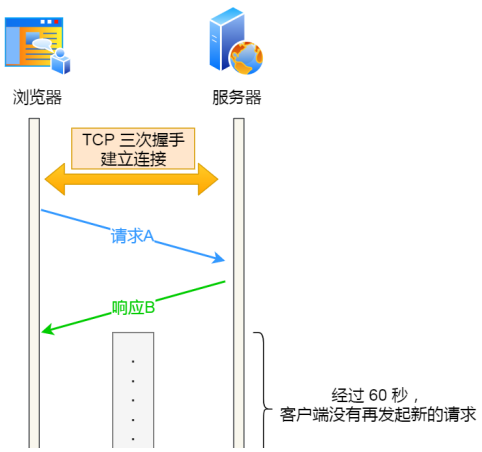
> TCP keepalive 机制探测的时间也太长了吧?

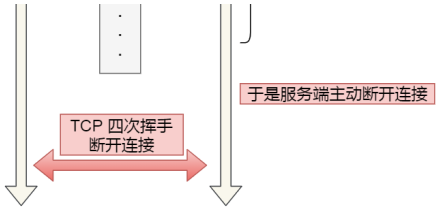
对的，是有点长。

TCP keepalive 是 **TCP 层（内核态）** 实现的，它是给所有基于 TCP 传输协议的程序一个兜底的方案。

实际上，我们应用层可以自己实现一套探测机制，可以在较短的时间内，探测到对方是否存活。

比如，web 服务软件一般都会提供 `keepalive_timeout` 参数，用来指定 HTTP 长连接的超时时间。如果设置了 HTTP 长连接的超时时间是 60 秒，web 服务软件就会**启动一个定时器**，如果客户端在发完一个 HTTP 请求后，在 60 秒内都没有再发起新的请求，**定时器的时间一到，就会触发回调函数来释放该连接。**





总结

客户端拔掉网线后，并不会直接影响 TCP 连接状态。所以，拔掉网线后，TCP 连接是否还会存在，关键要看拔掉网线之后，有没有进行数据传输。

有数据传输的情况：

- 在客户端拔掉网线后，如果服务端发送了数据报文，那么在服务端重传次数没有达到最大值之前，客户端就插回了网线，那么双方原本的 TCP 连接还是能正常存在，就好像什么事情都没有发生。

- 在客户端拔掉网线后，如果服务端发送了数据报文，在客户端插回网线之前，服务端重传次数达到了最大值时，服务端就会断开 TCP 连接。等到客户端插回网线后，向服务端发送了数据，因为服务端已经断开了与客户端相同四元组的 TCP 连接，所以就会回 RST 报文，客户端收到后就会断开 TCP 连接。至此， 双方的 TCP 连接都断开了。

没有数据传输的情况：

- 如果双方都没有开启 TCP keepalive 机制，那么在客户端拔掉网线后，如果客户端一直不插回网线，那么客户端和服务端的 TCP 连接状态将会一直保持存在。
- 如果双方都开启了 TCP keepalive 机制，那么在客户端拔掉网线后，如果客户端一直不插回网线，TCP keepalive 机制会探测到对方的 TCP 连接没有存活，于是就会断开 TCP 连接。而如果在 TCP 探测期间，客户端插回了网线，那么双方原本的 TCP 连接还是能正常存在。

除了客户端拔掉网线的场景，还有客户端「宕机和杀死进程」的两种场景。

第一个场景，客户端宕机这件事跟拔掉网线是一样无法被服务端感知的，所以如果在没有数据传输，并且没有开启 TCP keepalive 机制时，**服务端的 TCP 连接将会一直处于 ESTABLISHED 连接状态**，直到服务端重启进程。

所以，我们可以得知一个点。在没有使用 TCP 保活机制，且双方不传输数据的情况下，一方的 TCP 连接处在 ESTABLISHED 状态时，并不代表另一方的 TCP 连接还一定是正常的。

第二个场景，杀死客户端的进程后，客户端的内核就会向服务端发送 FIN 报文，**与客户端进行四次挥手**。

所以，即使没有开启 TCP keepalive，且双方也没有数据交互的情况下，如果其中一方的进程发生了崩溃，这个过程操作系统是可以感知得到的，于是就会发送 FIN 报文给对方，然后与对方进行 TCP 四次挥手。

- EOF -

推荐阅读 — 点击标题可跳转

1、元宇宙不是 PPT，已经发展到这个地步了！

2、“阿里味” PUA 编程语言火上 GitHub 热榜

3、一些著名的软件都用什么语言编写？