一文搞定 UDP 和 TCP 高频面试题！

程序员贺同学
腾讯 后端开发工程师
关注他

找工作面试，不管是春招还是秋招，面试官会经常问到 UDP 和 TCP，就拿自己参加的 2019 春招和秋招来说，加起来一共参加了 50 来场大大小小的面试，**几乎每一轮面试，面试官都会问到计算机网络的知识，尤其是 UDP 和 TCP 的知识点。**

面试参加多了，会发现面试官几乎问来问去都是那几个问题，如果我们提前把这些问题搞得明白些，相信，下次参加面试你心里也有一定的自信了，也就能给面试官留下不错的印象，那么今天就来看看，给大家总结其中的核心高频面试题都有哪些，再有面试官问你相关的知识点，看这篇就差不多。

PS：文章有点长，请耐心阅读。

目录：

前言

网络层只把分组发送到目的主机，但是真正通信的并不是主机而是主机中的进程。传输层提供了进程间的逻辑通信，传输层向高层用户屏蔽了下面网络层的核心细节，使应用程序看起来是在两个传输层实体之间有一条端到端的逻辑通信信道。

1、UDP 和 TCP 的特点与区别

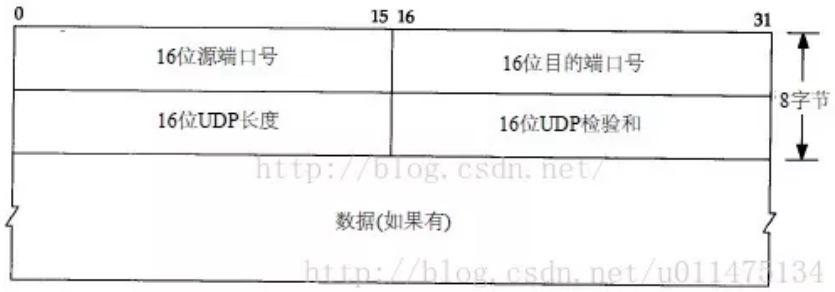
用户数据报协议 UDP（User Datagram Protocol）

是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。

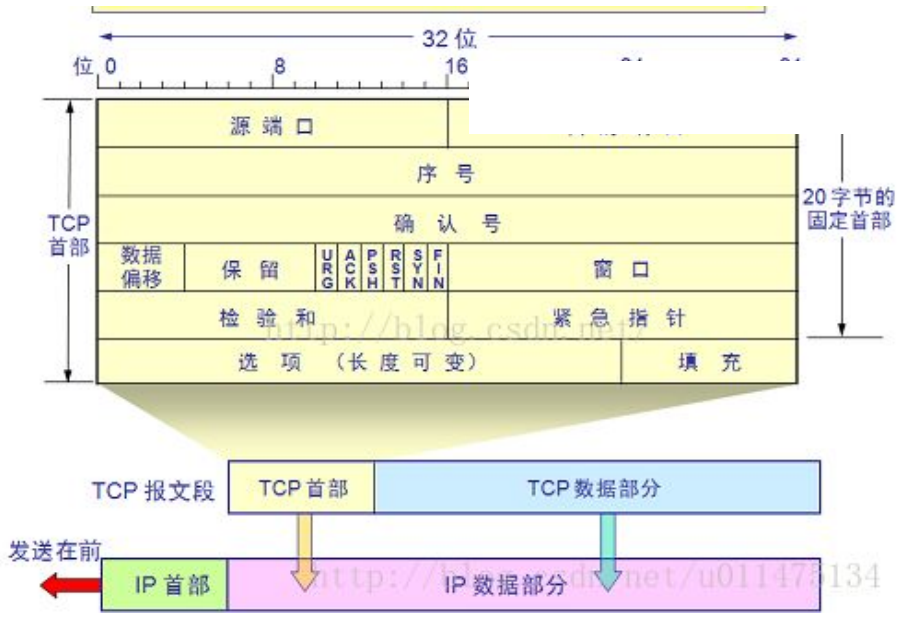
传输控制协议 TCP（Transmission Control Protocol）

是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。

2、UDP 、TCP 首部格式



UDP 首部字段只有 8 个字节，包括源端口、目的端口、长度、检验和。12 字节的伪首部是为了计算检验和临时添加的。



TCP 首部格式比 UDP 复杂。

序号：用于对字节流进行编号，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。

报文段中确认号就为 701。

数据偏移：指的是数据部分距离报文段起始处的偏移量，实际上指的是首部的长度。

控制位：八位从左到右分别是 CWR，ECE，URG，ACK，PSH，RST，SYN，FIN。

CWR：CWR 标志与后面的 ECE 标志都用于 IP 首部的 ECN 字段，ECE 标志为 1 时，则通知对方已将拥塞窗口缩小；

ECE：若其值为 1 则会通知对方，从对方到这边的网络有阻塞。在收到数据包的 IP 首部中 ECN 为 1 时将 TCP 首部中的 ECE 设为 1；

URG：该位设为 1，表示包中有需要紧急处理的数据，对于需要紧急处理的数据，与后面的紧急指针有关；

ACK：该位设为 1，确认应答的字段有效，TCP规定除了最初建立连接时的 SYN 包之外该位必须设为 1；

PSH：该位设为 1，表示需要将收到的数据立刻传给上层应用协议，若设为 0，则先将数据进行缓存；

RST：该位设为 1，表示 TCP 连接出现异常必须强制断开连接；

SYN：用于建立连接，该位设为 1，表示希望建立连接，并在其序列号的字段进行序列号初值设定；

FIN：该位设为 1，表示今后不再有数据发送，希望断开连接。当通信结束希望断开连接时，通信双方的主机之间就可以相互交换 FIN 位置为 1 的 TCP 段。

每个主机又对对方的 FIN 包进行确认应答之后可以断开连接。不过，主机收到 FIN 设置为 1 的 TCP 段之后不必马上回复一个 FIN 包，而是可以等到缓冲区中的所有数据都因为已成功发送而被自动删除之后再发 FIN 包；

窗口：窗口值作为接收方让发送方设置其发送窗口的依据。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

3、什么是 TCP 的三次握手和四次挥手?

TCP 是一种面向连接的单播协议，在发送数据前，通
“连接”，其实是客户端和服务器的内存里保存的一份关于对方的信息，如 IP 地址、端口号等。

TCP 可以看成是一种字节流，它会处理 IP 层或以下的层的丢包、重复以及错误问题。在连接的建立过程中，双方需要交换一些连接的参数。这些参数可以放在 TCP 头部。

TCP 提供了一种可靠、面向连接、字节流、传输层的服务，采用三次握手建立一个连接；采用四次挥手来关闭一个连接。

一个 TCP 连接由一个 4 元组构成，分别是两个 IP 地址和两个端口号。一个TCP连接通常分为三个阶段：启动、数据传输、退出（关闭）。

当 TCP 接收到另一端的数据时，它会发送一个确认，但这个确认不会立即发送，一般会延迟一会（提供网络利用率这部分有讲到）。

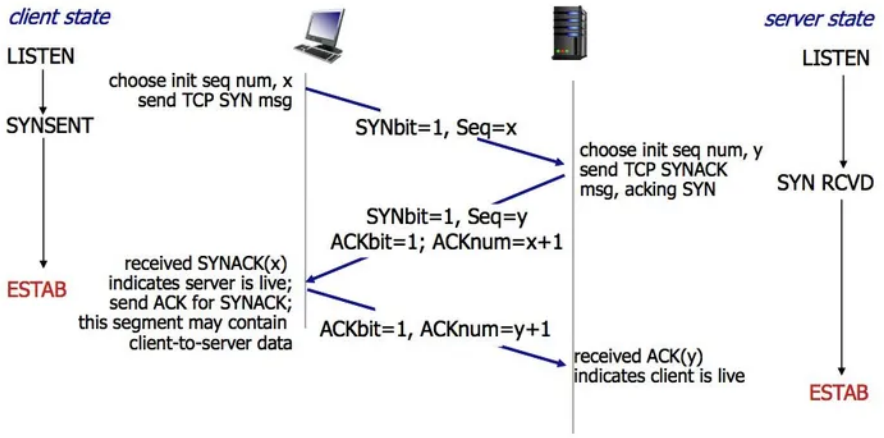
ACK 是累积的，一个确认字节号 N 的 ACK 表示所有直到 N 的字节（不包括 N）已经成功被接收了。这样的好处是如果一个 ACK 丢失，很可能后续的 ACK 就足以确认前面的报文段了。

一个完整的 TCP 连接是双向和对称的，数据可以在两个方向上平等地流动。给上层应用程序提供一种双工服务。一旦建立了一个连接，这个连接的一个方向上的每个 TCP 报文段都包含了相反方向上的报文段的一个 ACK。

另一方面，TCP 是一个字节流协议，绝不会以杂乱的次序给上层程序发送数据。因此 TCP 接收端会被迫先保持大序列号的数据不交给应用程序，直到缺失的小序列号的报文段被填满

4、TCP 的三次握手（为什么三次？）

三次握手：



假设 A 为客户端，B 为服务器端。

首先 B 处于 LISTEN（监听）状态，等待客户的连接请求。

- A 向 B 发送连接请求报文，SYN=1，ACK=0，选择一个初始的序号 x。
- B 收到连接请求报文，如果同意建立连接，则向 A 发送连接确认报文，SYN=1，ACK=1，确认号为 x+1，同时也选择一个初始的序号 y。
- A 收到 B 的连接确认报文后，还要向 B 发出确认，确认号为 y+1，序号为 x+1。

B 收到 A 的确认后，连接建立。

为什么三次?

1、第三次握手是为了防止失效的连接请求到达服务器，让服务器错误打开连接。

2、换个易于理解的视角来看为什么要 3 次握手。

客户端和服务端通信前要进行连接，“3次握手”的作用就是双方都能明确自己和对方的收、发能力是正常的。

第一次握手：客户端发送网络包，服务端收到了。这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的。

第二次握手：服务端发包，客户端收到了。这样客户端就能得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。从客户端的视角来看，我接到了服务端发送过来的响应数据包，说明服务端接收到了我在第一次握手时发送的网络包，并且成功发送了响应数据包，这就说明，服务端的接收、发送能力正常。而另一方面，我收到了服务端的响应数据包，说明我第一次发送的网络包成功到达服务端，这样，我自己的发送和接收能力也是正常的。

第三次握手：客户端发包，服务端收到了。这样服务端就能得出结论：客户端的接收、发送能力，服务端的发送、接收能力是正常的。第一、二次握手后，服务端并不知道客户端的接收能力以及自己的发送能力是否正常。

而在第三次握手时，服务端收到了客户端对第二次握手作的回应。从服务端的角度，我在第二次握手时的响应数据发送出去了，客户端接收到了。所以，我的发送能力是正常的。而客户端的接收能力也是正常的。

每次都是接收到数据包的一方可以得到一些结论，发送的一方其实没有任何头绪。我虽然有发包的动作，但是我怎么知道我有没有发出去，而对方有没有接收到呢？

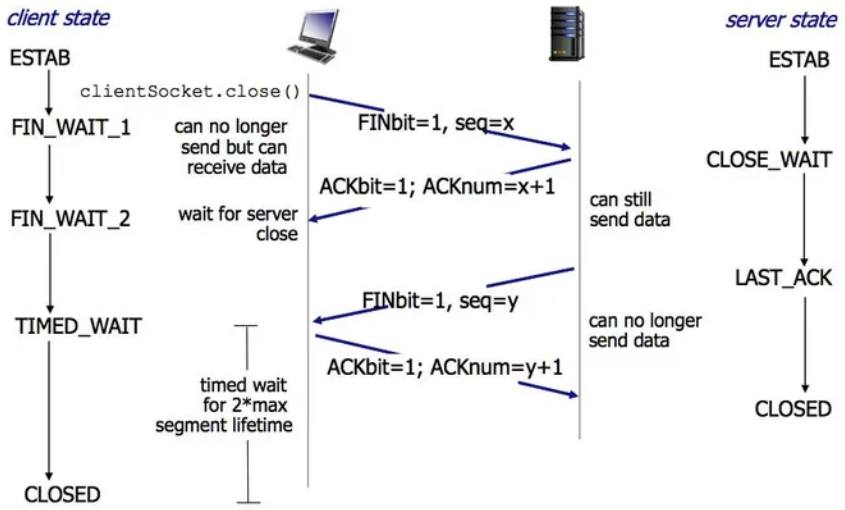
而从上面的过程可以看到，最少是需要三次握手过程的。两次达不到让双方都得出自己、对方的接收、发送能力都正常的结论。

其实每次收到网络包的一方至少是可以得到：对方的发送、我方的接收是正常的。而每一步都是有关联的，下一次的“响应”是由于第一次的“请求”触发，因此每次握手其实是可以得到额外的结论的。

比如第三次握手时，服务端收到数据包，表明看服务端只能得到客户端的发送能力、服务端的接收能力是正常的，但是结合第二次，说明服务端在第二次发送的响应包，客户端接收到了，并且作出了响应，从而得到额外的结论：客户端的接收、服务端的发送是正常的。

5、TCP 的四次挥手（为什么四次？）

四次挥手：



- 客户端发送一个 FIN 段，并包含一个希望接收者看到的自己当前的序列号 K，同时还包含一个 ACK 表示确认对方最近一次发过来的数据。
- 服务端将 K 值加 1 作为 ACK 序号值，表明收到了 端发起了关闭操作，通常这将引起应用程序发起自己的关闭操作。
- 服务端发起自己的 FIN 段，ACK=K+1, Seq=L。
- 客户端确认。进入 TIME-WAIT 状态，等待 2 MSL（最大报文存活时间）后释放连接。ACK=L+1。

为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

1、TCP连接是双向传输的对等的模式，就是说双方都可以同时向对方发送或接收数据。当有一方要关闭连接时，会发送指令告知对方，我要关闭连接了。

2、这时对方会回一个ACK，此时一个方向的连接关闭。但是另一个方向仍然可以继续传输数据，也就是说，服务端收到客户端的 FIN 标志，知道客户端想要断开这次连接了，但是，我服务端，我还想发数据呢？我等到发送完了所有的数据后，会发送一个 FIN 段来关闭此方向上的连接。接收方发送 ACK确认关闭连接。

注意，接收到FIN报文的一方只能回复一个ACK, 它是无法马上返回对方一个FIN报文段的，因为结束数据传输的“指令”是上层应用层给出的，我只是一个“搬运工”，我无法了解“上层的意志”。

接释放报文。

4、因为服务端在 LISTEN 状态下，收到建立连接请求的 SYN 报文后，把 ACK 和 SYN 放在一个报文里发送给客户端。而关闭连接时，当收到对方的 FIN 报文时，仅仅表示对方不再发送数据了但是还能接收数据，己方是否现在关闭发送数据通道，需要上层应用来决定，因此，己方 ACK 和 FIN 一般都会分开发。

TIME_WAIT

客户端接收到服务器端的 FIN 报文后进入此状态，此时并不是直接进入 CLOSED 状态，还需要等待一个时间计时器设置的时间 2MSL。这么做有两个理由：

- 确保最后一个确认报文能够到达。如果 B 没收到 A 发送来的确认报文，那么就会重新发送连接释放请求报文，A 等待一段时间就是为了处理这种情况的发生。
- 等待一段时间是为了让本连接持续时间内所产生的所有报文都从网络中消失，使得下一个新的连接不会出现旧的连接请求报文。

6、TCP 短连接和长连接的区别

短连接：Client 向 Server 发送消息，Server 回应 Client，然后一次读写就完成了，这时候双方任何一个都可以发起 close 操作，不过一般都是 Client 先发起 close 操作。短连接一般只会在 Client/Server 间传递一次读写操作。

短连接的优点：管理起来比较简单，建立存在的连接都是有用的连接，不需要额外的控制手段。

长连接：Client 与 Server 完成一次读写之后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

在长连接的应用场景下，Client 端一般不会主动关闭它们之间的连接，Client 与 Server 之间的连接如果一直不关闭的话，随着客户端连接越来越多，Server 压力也越来越大，这时候 Server 端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可以避免一些恶意连接导致 Server 端服务受损；如果条件再允许可以以客户端为颗粒度，限制每个客户端的最大长连接数，从而避免某个客户端连累后端的服务。

长连接和短连接的产生在于 Client 和 Server 采取的关闭策略，具体的应用场景采用具体的策略。

7、TCP粘包、拆包及解决办法

为什么常说 TCP 有粘包和拆包的问题而不说 UDP？

由前两节可知，UDP 是基于报文发送的，UDP首部采用了 16bit 来指示 UDP 数据报文的长度，因此在应用层能很好的将不同的数据报文区分开，从而避免粘包和拆包的问题。

而 TCP 是基于字节流的，虽然应用层和 TCP 传输层之间的数据交互是大小不等的数据块，但是 TCP 并没有把这些数据块区分边界，仅仅是一连串没有结构的字节流；另外从 TCP 的帧结构也可以看出，在 TCP 的首部没有表示数据长度的字段，基于上面两点，在使用 TCP 传输数据时，才有粘包或者拆包现象发生的可能。

什么是粘包、拆包?

假设 Client 向 Server 连续发送了两个数据包，用 packet1 和 packet2 来表示，那么服务端收到的数据可以分为三种情况，现列举如下：

第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象。

难处理。

第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包要么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理，对于接收端同样是不好处理的。

为什么会发生 TCP 粘包、拆包?

- 要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包。
- 待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包。
- 要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会发生粘包。
- 接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包。

粘包、拆包解决办法

由于 TCP 本身是面向字节流的，无法理解上层的业务数据，所以在底层是无法保证数据包不被拆分和重组的，这个问题只能通过上层的应用协议栈设计来解决，根据业界的主流协议的解决方案，归纳如下：

- **消息定长：**发送端将每个数据包封装为固定长度（
  次接收缓冲区中读取固定长度的数据就自然而然的
- **设置消息边界：**服务端从网络流中按消息边界分离出消息内容。在包尾增加回车换行符进行分割，例如 FTP 协议。
- **将消息分为消息头和消息体：**消息头中包含表示消息总长度（或者消息体长度）的字段。
- 更复杂的应用层协议比如 Netty 中实现的一些协议都对粘包、拆包做了很好的处理。

8、TCP 可靠传输

TCP 使用超时重传来实现可靠传输：如果一个已经发送的报文段在超时时间内没有收到确认，那么就重传这个报文段。

一个报文段从发送再到接收到确认所经过的时间称为往返时间 RTT，加权平均往返时间 RTTs 计算如下：

其中，$0 \leq a < 1$，RTTs 随着 a 的增加更容易受到 RTT 的影响。超时时间 RTO 应该略大于 RTTs，TCP 使用的超时时间计算如下：

其中 RTTd 为偏差的加权平均值。

9、TCP 滑动窗口

窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。

10、TCP 流量控制

流量控制是为了控制发送方发送速率，保证接收方来得及接收。

接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。

实际上，为了避免此问题的产生，发送端主机会时不时的发送一个叫做窗口探测的数据段，此数据段仅包含一个字节来获取最新的窗口大小信息。

11、TCP 拥塞控制

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而拥塞控制是为了降低整个网络的拥塞程度。

TCP 主要通过四个算法来进行拥塞控制：

慢开始、拥塞避免、快重传、快恢复。

发送方需要维护一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口与发送方窗口的区别：拥塞窗口只是一个状态变量，实际决定发送方能发送多少数据的是发送方窗口。

为了便于讨论，做如下假设：

- 接收方有足够大的接收缓存，因此不会发生流量控制；
- 虽然 TCP 的窗口基于字节，但是这里设窗口的大小单位为报文段。

慢开始与拥塞避免

发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...

注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。

如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。

快重传与快恢复

在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。

在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。

在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。

12、提供网络利用率

1、Nagle 算法

发送端即使还有应该发送的数据，但如果这部分数据很少的话，则进行延迟发送的一种处理机制。具体来说，就是仅在下列任意一种条件下才能发送数据。如果两个条件都不满足，那么暂时等待一段时间以后再进行数据发送。

- 已发送的数据都已经收到确认应答。
- 可以发送最大段长度的数据时。

2、延迟确认应答

接收方收到数据之后可以并不立即返回确认应答，而是延迟一段时间的机制。

- 在没有收到 2*最大段长度的数据为止不做确认应答。
- 其他情况下，最大延迟 0.5秒 发送确认应答。
- TCP 文件传输中，大多数是每两个数据段返回一次确认应答。

3、捎带应答

在一个 TCP 包中既发送数据又发送确认应答的一种机制，由此，网络利用率会提高，计算机的负荷也会减轻，但是这种应答必须等到应用处理完数据并将作为回执的数据返回为止。

今天的知识点掌握了吗？不要忘了学而时习之，不亦可乎。

欢迎留言和我交流~

大家如果有补充的，也是可以留言区补充一波哦。

参考：

cnblogs.com/panchanggui...

cnblogs.com/qcrao-2018/...

github.com/CyC2018/CS-N...

最近写的干货：

----------极客时间专栏分享----------

对于计算机网络的学习，我相信大家一开始都会觉得这里面的知识太多太杂，容易摸不着头脑，这里呢，我给大家分享一个极客时间的专栏福利，这是我觉得极客时间上写得最好的一个专栏之一，学起来真的很有趣，而且每一篇后面都会有很多精彩的留言，真的是非常不错的，真诚的推荐给大家，希望大家快乐学习，一起加油！

（极客时间的挺多专栏都写的挺不错哦）。

time.geekbang.org/colum... (二维码自动识别)

----------分割线----------

如果你觉得这篇内容对你挺有启发，我想邀请你帮我三个忙，让更多的人看到这篇文章：

1、**点赞**，让更多的人也能看到这篇内容（收藏不点赞，都是耍流氓 -_-）。

2、**关注我和专栏**，让我们成为长期关系，这样我认真更新的文章你都可以看到。

3、**关注公众号「herongwei」**主要分享技术、计算机基础之类，大学生成长点滴的文章，里面已有 100 多篇原创文章。

----------分割线----------

没想到这篇文章已经破千赞了，谢谢大家的支持。

对于程序员或者未来想从事编程行业的人来说，算法非常重要，算法厉害的人进大厂非常容易，这里小贺送大家一本谷歌大佬+阿里大佬的算法笔记，身边不少朋友通过它加入大厂：

在阅读过程中，有任何问题都可以问小贺，欢迎交流，一起成长！

觉得不错的小伙伴，记得帮我 @herongwei 点个赞哟，笔芯~~

编辑于 2021-08-17 22:50

「不管你赞或不赞，我都会认真写文章」

赞赏

1 人已赞赏

UDP TCP/IP 计算机网络

46 条评论 默认 最新

老实人
衮到收藏夹里吃灰吧
2020-03-23 回复 12

老实人 ‣ **从不卖萌o**
吃了这一年还有下一年
2021-05-25 回复 4

从不卖萌o
吃了一年了已经，哈哈哈哈
2021-05-25 回复 喜欢

专业划水运动员
为什么3次握手，RFC里面说得很清楚，为了交换双方的初始SEQ
2020-03-17 回复 9

JX Wang
交换seq是处于于安全性和tcp segment没有歧义的考量, 真正三次握手的原因 作者这里表述有点不准确, 并不是为了确认双方接发都正常, 而是为了确认双方都知道 “对方愿意建立连接并且对方知道自己愿意建立连接“的这个事实
2020-05-22 回复 9

cg33 ‣ **JX Wang**
「愿意建立连接」有点中文含糊的意味了，感觉理科角度确定一点的表述应该是：发接双方确认连接的有效性，失效连接应不被建立或拒绝。
2020-09-16 回复 8

展开其他 1 条回复 >

顾长风
窗口。。。这翻译的。。 不就是一个队列吗。
2021-03-31 回复 3

问君能歪几层楼
英文就是window。比如信号处理中的window也叫窗口，比如"加窗"
2022-11-21 回复 1

学无止境
总结的不错
2020-02-24 回复 /

程序员贺同学 作者
谢谢
2020-02-24 回复 2

eeechoo
我该如何才能像你一样优秀

2020-04-20 回复 3

程序员贺同学 作者
😔😔
2020-04-20 回复 1

不辣的皮皮
我提个面试题:
udp不稳，tcp太慢

蛋定
mmp
2021-01-14 9

从不卖萌o ▸ **蛋定**
妈卖批?
2021-05-25 回复 1

展开其他 3 条回复 >

爱跑步的鸭脑壳
满满的干货，多看看
2020-03-26 回复 2

钱枭
m
2020-03-25 回复 2

二百八十里
我收藏了，但是😔
2020-03-23 回复 2

星光达米安
但是还是吃灰去了
2020-03-23 回复 2

qwe
假如正常传输，发送方在某个传输轮次发送两个数据包，则接收方返回两个ack还是一个?
2020-05-07 回复 1

一拳超人
如果是挥手时没有在传数据的情况，也需要完成四次挥手吗?
2020-04-17 回复 1

网络巨人
在某一次笔试的时候，我记得有一道题也说过，如果服务器已经没有数据需要传了，第二次第三次可以合并作一次。
2020-08-28 回复 喜欢

杨广
实际抓个包会看到很多三次的
2020-07-24

蓝蓝Coding
活捉大佬
2020-04-11 回复 1

程序员贺同学 作者
嘿嘿嘿，小蓝来了🤭🤭
2020-04-11 回复 1

宇宙警探梁非凡
到我收藏夹呆着吧!
2020-03-27 回复 1

DIFF
这总结 超好 赞赞赞
2020-03-22 回复 1

李清焰
很全面啊！给作者点赞
2020-03-01 回复 1

程序员贺同学 作者
谢谢支持

DarenSu
写的很不错，很强啊，赞
2021-08-09 回复 1

程序员贺同学 作者
感谢老哥支持
2021-08-09 回复 喜欢

明翼
注明出处是否可以转载那?
2020-06-30 回复 喜欢

张杨
看到粘包两个字就不想往下看了.....
2020-06-01 回复 喜欢

xbrin
TCP根本不存在粘包/拆包问题，说明里都写了是面向流的
2020-05-25 回复 喜欢

xbrin ‣ 隔壁小啾
面向业务的属于应用层协议（比如HTTP），与TCP无关
03-04 回复 喜欢

隔壁小啾
面向业务来说的话，是有粘包的这个现象的，毕竟发送端可能希望服务端那边消息按照指定内容大小来接收
03-04 回复 喜欢

菜菜
写的真好🥰
2020-05-21 回复 喜欢

程序员贺同学 作者
感谢支持
2020-05-21 回复 1

点击查看全部评论 >

发布一条带图评论吧

文章被以下专栏收录

编程之路
编程之路的好帮手

推荐阅读

12道UDP和TCP协议的高频面试题

linux服务器开发相关视频解析： 大厂必问面试题：UDP可靠性设计 10道经典面试题的剖析， 技术方向如何决定职业方向c/c++ linux服务器开发免费学习地址： c/c++ linux后台服务器高级架构师不...

linux 发表于linux...

33 张图详解 TCP 和 UDP：打通网络和应用的中间商

网工Fox 发表于图解网络

通信协议下的可靠U协议快在哪呢?

前言：Internet 协议集连接的传输协议，该协数据报协议（UDP，U Datagram Protocol）用程序提供了一种无需可以发送封装的 IP 数

玩转Lin...