链滴

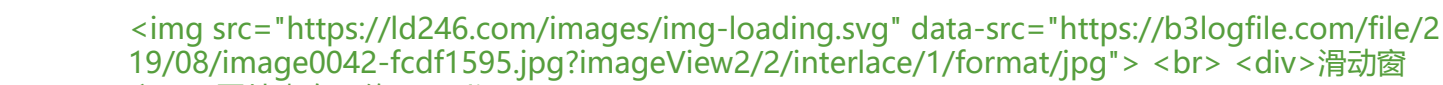
TCP 如何保证可靠性？一篇搞定！

作者：BigBigBigPeach

原文链接：https://ld246.com/article/1565779877331

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180129.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="TCP可靠性">TCP 可靠性</h2>
大部分应用程序在传输层的协议中选择都是 TCP。因为 TCP 能可靠的传输数据，保证数据能正
到达目的地。当然，因为可靠性的保证，导致 TCP 设计非常复杂，传输时间也会略长。
在特殊的应用领域如音频视频领域，一般采用的是 UDP。这是因为音视频对于时间是有精确要
的。假设采用 TCP，消息超时没有 ack，进行消息重发，但是这个时候可能已经过了这一帧数据的播
时间。这样的消息重传是无意义的。另外，像 DNS 查询等收发数据较短的也选择使用 UDP。
<h2 id="TCP可靠性靠什么保证-">TCP 可靠性靠什么保证？</h2>
<h3 id="连接可靠">连接可靠</h3>
上次我们提到了 TCP 连接过程的三次握手四次挥手，参考<a href="https://ld246.com/forwar
?goto=https%3A%2F%2Fwww.deepstack.top%2Farticles%2F2019%2F08%2F09%2F15653434
0177.html" target="_blank" rel="nofollow ugc">谈谈 TCP 的三次握手四次挥手
还有一些异常情况，导致连接失败。比如接收方接到 TCP 传输的包，发现相应端口并不对方开
服务；再比如应用程序突然进程退出了；再比如，网络拥塞严重，重试多次无法正常收发消息。这些
况下，TCP 会主动发送一个叫 RST(reset)的消息，告诉对方，现在无法正常通
，可以直接关闭连接了，自己也会主动关闭相关连接，避免资源一直被占用。
<h3 id="ACK">ACK</h3>
ACK 机制也是我们常见的一种信息确认机制。当消息接收方接收到消息，返回一个对应的 ACK
发送方就知道这个消息已经处理完成。
在连接篇我们也简单提到过 ACK 的信息结构。但 TCP 中的 ACK 需要与滑动窗口相配合。在下面滑
窗口我们再详细说说。
<h3 id="超时重传">超时重传</h3>
消息超时也就是说没有在等待时间内收到对方的 ACK 消息。这个时候，为了保证消息对方能够
确收到，我们需要将这个消息进行重新传输，如果尝试成功，则继续发送接下来的包。若尝试几次均
败，那么 TCP 会强行断开连接，发送 RST 信息。并告知应用程序连接错误。
<h3 id="TCP滑动窗口机制">TCP 滑动窗口机制</h3>



<h4 id="滑动窗口用以解决什么问题">滑动窗口用以解决什么问题</h4>
我们前面提到过 TCP 会将较大数据拆分成一个个小的数据包再进行发送。发送完一个包，等待 A
K，这种模式是最简单的。但是问题也很明显，慢，吞吐量低。所以我们在等待 ACK 的同时，可以继
发送接下来的包。也就是说，滑动窗口就是在发送完一个数据包后，不需要等待 ACK 消息返回，可
发送后面的数据包，解决吞吐量问题。
那么发送多少呢？可以无限制的发送吗？当然不行！因为接收方能不能处理完这些数据我们也不
楚，缓冲一旦溢出，就无法接收新的消息。所以我们需要滑动窗口来告诉我们要发送多少数据才是合
的。
<h4 id="滑动窗口的类别">滑动窗口的类别</h4>

可用窗口。即上面图片所说，窗口的范围内可能有各种状态的包。已经收到 ACK 的包，可用窗
会移动到最新的已收到 ack 的后面，发送后面的包。每次发送完包 或者 收到 ACK，数据包的状态也
改变，可用窗口也会随着更新。
发送窗口。简单说来，这个窗口代表了接收方能够接受的数据量。发送窗口的大小是接受方传给
送方的。发送窗口的更新，是随着接收方应用程序已经将数据读取完成后，清空部分缓冲区，使得可
接收更多数据，这个时候可以把大小更新发送给其接收方（因为 TCP 是全双工通信，所以发送方接
方角色经常互换，描述起来比较费劲。此处接收方指的是向该服务发送数据的发送方...）

<h4 id="发送窗口的数据发送异常处理">发送窗口的数据发送异常处理</h4>
上面我们简单提到了发送窗口接收到 ACK 消息后会根据情况移动。那么有没有可能接收方先接
到后面的数据包，返回了后面数据包的 ACK，导致发送窗口移动后，数据发送异常呢？
其实这个问题是在接收方进行处理的，接收方如果发现当前的数据并不是接着上次的数据包来的

比如收到了 1，2，4，未收到 3，那么接收方会返回 2 的确认信息，而不是 4 的。
再思考下，如果窗口够大，4 后面也传输了很多数据包，那么这些数据包都不能被确认，只能等待 3 如果一个发送窗口内丢失了不少包，那么尝试重新传会比较浪费时间，因为这个重新传输需要等着数的发送方发现超时才会重发。重发的过程中，也有可能之前的发送未确认的一些数据也会超时，那么待超时 & 重发都是比较浪费时间。
选择重发的策略我们可以直接发送丢失包及丢失包后面所有的数据包。这个量比较大的时候必然占用网络资源。那么有没有更好的方式去处理重发呢？毕竟 4 已经收到了，能不能只发丢失的片段呢可以，如果双方都支持 SACK （Selective ACK）的话。那么收到 4 之后，会回一个 SACK 告诉发送方，我已经收到你发的 4 了，但是呢，有一段数据 3 你没给我发过来，需要发~ 这样就解决重发过多的问题了
<h2 id="总结">总结</h2>
TCP 在可靠性上下了很大的功夫，可靠性也增加了设计的复杂度。
保证可靠性的重点就是滑动窗口机制，ACK & 超时重传等机制都在滑动窗口的工作过程中体现
TCP 非常复杂，我们还有很多的细节没提到，比如 TCP 标志位的其他标志、TCP 其他参数选项等，有兴趣的同学也可以自己研究看看。