# 第十八章实现流程



> 本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

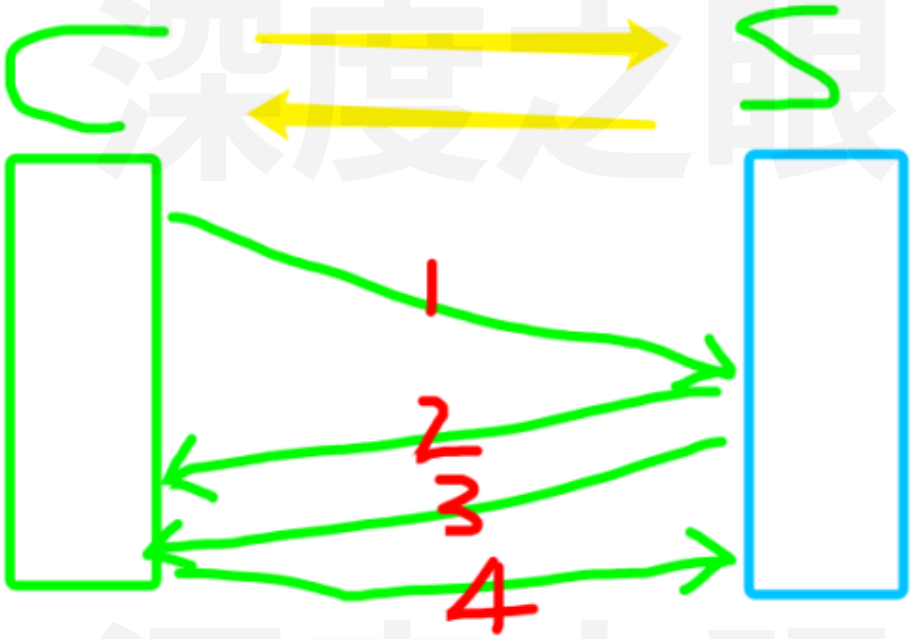
## 一 TCP协议数据传输

### 1. TCP协议三次握手建连接

在上面的章节，我们了解了网络通信的基本原理，接下来我们来学习的是传输层的协议，传输层的协议有两个，分别是TCP协议和UDP协议，我们先来学习TCP协议。

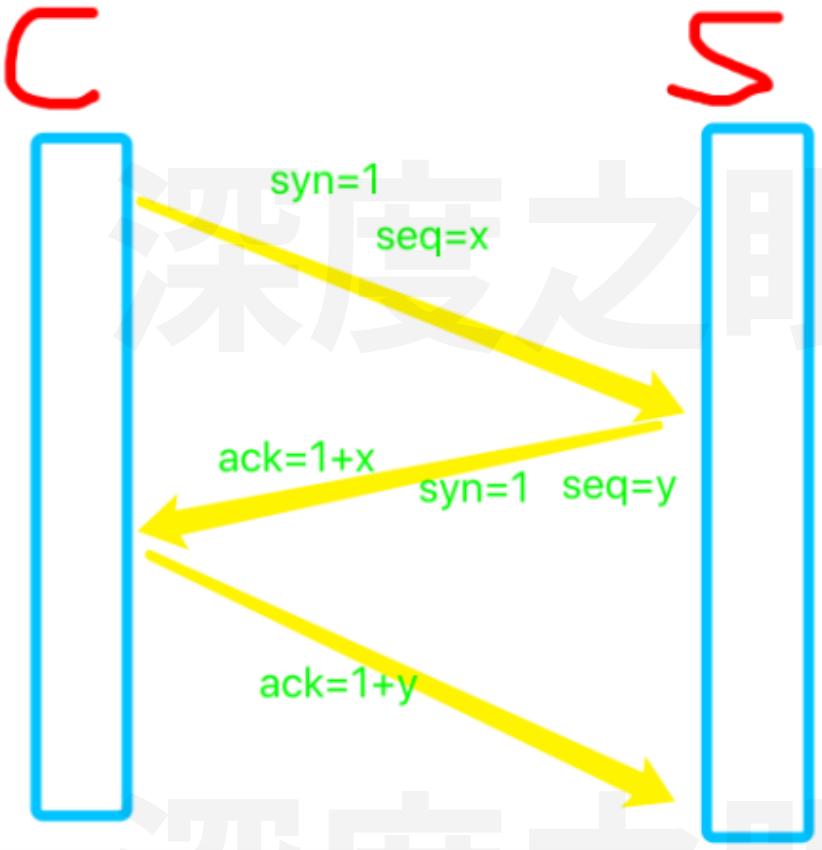
传输层是负责传输数据的，它的数据来自应用层，应用层产生了数据，把数据交给操作系统，也就是首先给了给传输层，让他来负责对外通信传输。应用层的数据由应用层软件来控制，传输层和传输层以下的数据由操作系统来控制。完整的数据传输过程就是客户端或者服务端应用层产生数据，把这个数据给它的操作系统，操作系统会把数据按照传输层，网络层，数据链路层和物理层的顺序发送给目标机器，目标机器再反过来把数据依次解包最后给到目标机器的应用层软件。通过前面章节的学习，相信你已经清楚了这个流程，那么这个一系列的流程看似很复杂，就像是一条蜿蜒曲折的通道，走到头，最后才完成了通信。那么是不是每次通信传输数据的时候都要完整的走一遍这个流程呢?

答案当然是否定的，我们以客户端机器C和服务端机器S为例，最好建立好一条C的传输层到S的传输层的通路和一条S到C的通路（注意：必须是两条路），以后再需要传输数据就沿着这两条通路走就可以了。所以，我们接下里就要在双方的传输层建立一个双向的通路。




> 这就像是两个人谈恋爱一样，客户端和服务端需要建立两条黄色的通路，比如，我们把詹姆斯看作客户端，把韦德看作服务端。
>
> 詹姆斯对韦德说：“我喜欢你”，这是红色数字第一条线
> 韦德说：“嗯，我知道了”，算是给詹姆斯一个回应，这是红色的第二条线，至此，从左向右的第一条通路建立完成；
> 然后韦德接着又说：“其实，我也喜欢你”，这是红色的第三条线
> 最后詹姆斯也回给韦德一个相应：“嗯嗯，我知道了，小德德，那咱俩就在一块吧”。
>
> 至此从右向左的第二条黄色通路也建立完成，这才有了我们在NBA看到的“詹伟连线”，这也就是传输层的双向通路的建立过程。

这就是TCP协议的通讯过程，我们一般会把发一次消息称之为握手一次， 你会发现要建好这个双向通路经过了4次握手，但是，其实第二次和第三次握手可以合并成一次，服务端再给客户端相应的同时，也可以给客户端发一个连接请求，所以建立连接是经历了三次握手，就是TCP协议的三次握手建连接。以后在想要发送数据就不需要找对方在哪里，只要基于这个已经建立好的连接就可以了。




刚才我们通过比喻的说法描述了一下建立连接的过程，那么在这过程中会发送请求信息和确认信息，由于与服务端相连接的客户端并不只有一个，所以还要确保响应的信息是基于刚才的连接请求，所以在客户端在发送请求的时候会有一个syn=1，这是请求信息，seq = x，这个可以理解为是该客户端的请求编号，号码我们用x 表示，服务端在做相应的时候回一个ack=1，这就是表示确认收到请求了，但是如何表示收到的是刚才这个客户端的请求呢？就是对应请求编号给一个响应：ack= 1 + x，那么服务端还要再给客户端发一个请求，同理：syn= 1，seq = y，客户端响应：ack= 1 + y。

TCP协议在发送数据前，通信双方必须在彼此间建立一条连接。所谓的“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。

## 2. TCP协议存在的漏洞

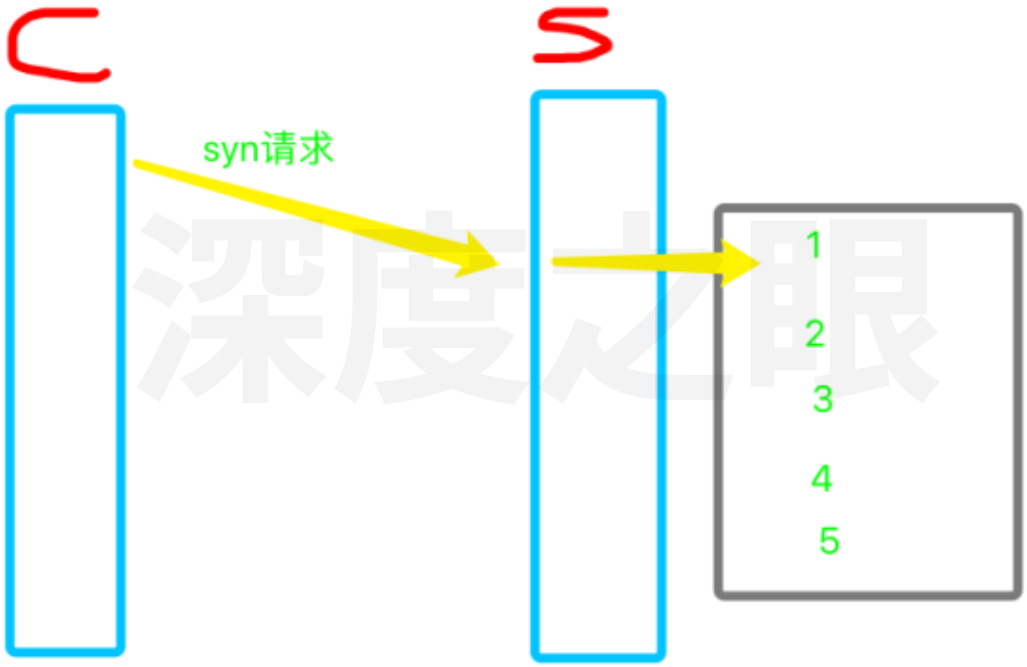
通过TCP协议发送数据需要客户端要先发送请求，服务端再给对应的回一个请求，然后就按照上面的过程建立双向连接。服务端可以和一台客户端建立连接，也就可以和多台甚至是无数台客户端建立连接， 建立连接就要占用资源，在应用程序和操作系统中做一个选择，他所占用的资源肯定是操作系统的资源， 连接是在传输层和传输层以下，是由操作系统来维持的，所以占用操作系统的资源。

TCP协议就是客户端给服务端发一个请求，服务端就要回给客户端一个响应，所以TCP协议又被人们称为“好人协议”，无论是谁发送请求，他都会给一个回应， 这个漏洞是设计层面的漏洞，这么设计可以大大降低网络的压力，但是这种设计无论好坏都会给客户端一个回应，这就给了一些不法分子可乘之机， 有一种攻击叫做syn flood（俗称syn洪水攻击） ，一些假的客户端模拟大量的syn请求向服务器机器发包，发完一次包服务器就会给他回一个，但是他是假的ip，不会等着接收服务器的响应，服务器会响应回一次他没有响应，服务端还会再给他回一次， 服务端默认是第一次隔5秒回一次响应，第二次是3秒，后面是2秒，1秒，当然作为服务器开发，Linux操作系统可以优化这个默认的时间，但问题是假的客户端无论你怎么优化他都收不到响应，那么他就会一直占用系统资源。与此同时，真正的客户端机器也要发送syn请求，但是有大量的假的请求进来，正常的请求就有可能备阻塞（就像是你的绞肉机里面被人恶意的塞了很多大白菜进去，堵在了里面，你再放肉进去肯定要先堵在外面）。其实syn洪水攻击，我们可以通过调整Linux操作系统的内核参数来改变，这里我们做一个了解就可以了。

syn洪水攻击是由黑客发起的，但是就算是没有黑客，当有大量的真正的客户端发送请求的时候（绞肉机只放肉也有可能会堵住），也有可能使服务端瘫痪，新浪微博号称能承受八个星轨（一个星轨指的是一位明星出轨），却承受不住一对明星结婚，淘宝天猫双十一在成交高峰时同样也会服务器瘫痪，中国最牛逼的网站12306没有之一，论技术水平是世界级的， 它的并发量即使是facebook也难以望其项背，技术水平是一方面，资金的实力也是必不可少的。

## 3. 半连接池

服务端机器肯定不能任由大量的连接请求把自己玩废了，机器本身就应该具备一定的保护机制。对于服务端来说，来一个syn请求就立马接收，再来一个syn请求又立马接收，如果真的是这样，会导致内存被无限的占用。所以，即使是正常的客户端机器来连接的时候也应该做一个限制，这个限制叫做backlog（中文叫半连接池），来一个syn请求先放在这个半连接池中，池中的请求数量是固定的，当达到这个数量后再来的请求会挡在池的外面，同时处理的连接请求只有池中的固定的数量，这也是服务端机器操作系统所能够承受的数量。

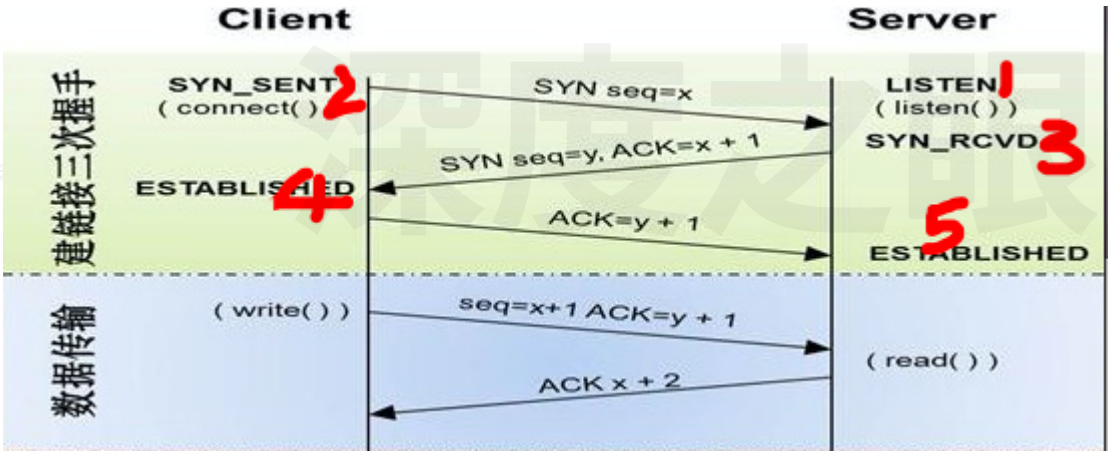



如图所示，灰色的框框就表示半连接池，服务端机器操作系统会在自己的可承受范围内，从半连接池去取出一个连接来做回应。与此同时，半连接池中的连接请求就会少一个，这时就会有新的连接请求进来。如果服务端机器内存大，那么半连接池中所能承载的连接请求数量就会多。其实半连接池中装的连接请求并没有多少，但是服务端响应的速度非常快，这样就达到了一个缓冲 的目的。

类似这样的缓冲我们生活中随处可见，你们过年回家取登机牌办理行李托运的时候（这个是经济舱，头等舱我没座过，不清楚是怎么样的），后面往往会排了很长的队伍，但是在柜台办理的业务的人一般只有那么几个，虽然柜台那里还有空间，但是保安不让你过去了，一方面是为了保护乘客的隐私，另外一方面就是为了控制好秩序，能够让服务台高效的工作。

刚才我们讲了一个半连接池的概念，为什么会叫这个名字呢？因为从半连接池仅仅是syn请求发送给了服务端，并没有完成全部的连接过程。

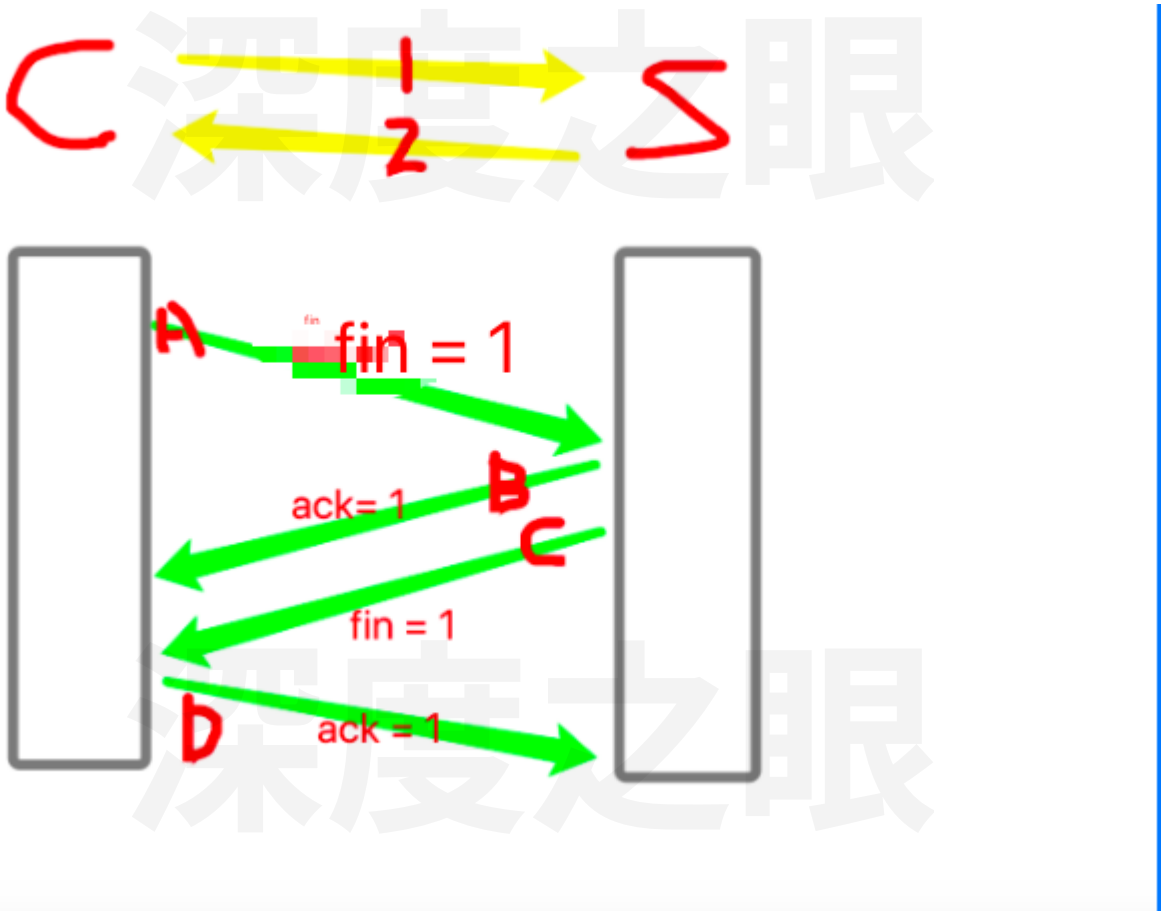
## 4. 连接请求状态



三次握手建连接，每个位置都会有一个相应的状态。首先服务端会进入一个LISTEN状态，这个就是服务端的监听状态，等待客户端发送连接请求，客户端只要一发送连接请求，就会进入SYN_SENT 状态，接下里服务端会进入SYN_RCVD状态，再接下来客户端就会进入ESTABLISHED状态，表示连接建立成功，这指的是客户端到服务端的连接这条通路建立成功，后面客户端再回一个相应，第五条线也就是服务端到客户端的通路建立成功。如果服务端机器同时服务于大量的客户端机器，这叫做并发（并发是我们下一阶段课程要学习的内容），如果你的服务端机器在遭受syn洪水攻击的时候，服务端会出现大量的SYN_RCVD状态（服务端有1，3，5三个状态，客户端有2，4两个状态），除此之外，服务端机器会迅速的在1，3，5中切换，你是看不到SYN_RCVD状态的。

## 5. 四次挥手断连接

我们建好连接是为了发送数据，那么我们这个连接也是占用操作系统资源的，当这个连接不再使用的时候，我们的操作系统就需要回收这个连接。我们要发送数据的时候建立连接，那么什么回收连接呢?
很简单，肯定是数据发送完了才才回收连接。 我们的连接有两条，分别是客户端到服务端的连接和服务端到客户端的连接，这两条连接建立的时候是一条一条来建立的，断开连接肯定也是一条一条断开的，客户端要给服务端发送数据，服务端也要给客户端发送数据，主动发送数据的那一方先发完了，就会断开这个连接。

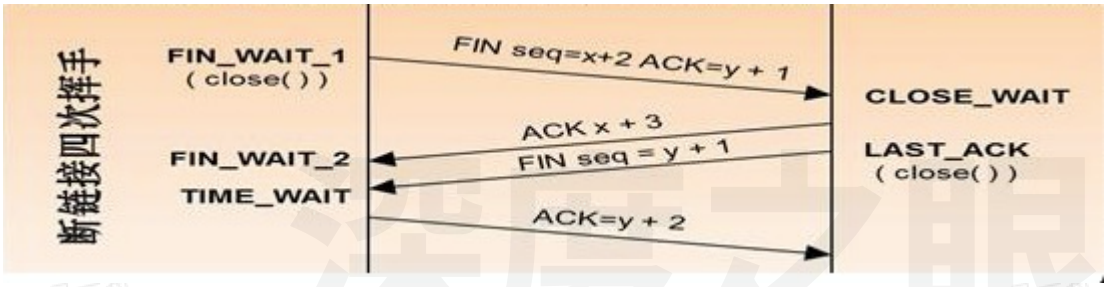



客户端与服务端已经建立好1，2两条通路，当客户端的数据发送完了就会向服务端发一个断连接的请求A，信息是fin= 1（seq当然也有，否则服务端不知道断开哪一个连接，我这里没画下面也一样），服务端收到这个请求之后，回给服务端回一个相应B，ack=1，表示收到了这个断连接的请求，至此客户端到服务

端的连接通路1 断开了；当服务端向客户端发送的数据也发完了，就会给客户端发送一个断连接请求C，Fin = 1， 然后客户端给一个相应D，ack= 1，至此服务端到客户端的连接通路2断开了。
这里面和建立连接的时候不同的是B，C这两步不能合并成一步，因为已经建立好了连接，还在发送数据的时候连接是不能断开的，所以断连接需要四步，这就是四次挥手断连接。

> 其实这个和谈恋爱也是一样的，刚开始的两个人一拍即合就在一块了，但是到了分手的时候往往没有那么简单。

> 韦德和詹姆斯说：“我要退役了，以后咱俩就不要在搞什么詹伟连线了吧”
> 詹姆斯说：“我知道了，过两年你在退役吧，我还差一个总冠军没给你”
> 韦德说：“我知道了，那我再等你一个赛季，不过以后nba赛场上，再无詹伟连线了”
> 詹姆斯说：(一个赛季之后)“这个总冠军给不了你了，你老了，我也不再年轻了，咱们还是分手吧”
> 韦德：“好”




建立连接的时候是一般是客户端发起的，而断开连接的时候一般是都服务端先发起的，因为服务端同时服务于很多用户，他特别想要断开连接服务于其他的用户。所以，FIN_WAIT_1这个状态是服务端先进入，断连接请求给客户端，然后客户端进入CLOSE_WAIT状态，客户端发一个相应给服务端，服务端会进入FIN_WAIT_2的状态，等客户端数据发送完成，客户端也会发一个断连接请求给服务端，服务端收到这个请求之后就会进去TIME_WAIT状态，然后再发一个相应给客户端，至此连接断开成功。
对于服务端来说，上面的三种状态中最有可能出现停滞的是TIME_WAIT状态，因为他会等待客户端发送了完了数据才会进入这个状态。所以，如果服务端机器出现了大量的TIME_WAIT状态，这说明服务器正在处于高并发的状态，你需要及时的监听机器的内存是否够用（主要是内存 ），CPU的使用率是多少，如果发现内存的占用达到70%以上 ，你就应该提高警惕，如果任由并发继续下去，服务端的机器就有可能瘫痪。

## 6. TCP协议是可靠协议

我们建立好了连接，中间还有传输过程，TCP协议又被称作“可靠协议”，怎么保证数据传输的可靠呢？比如你发送了一个快递，当接收快递的这一方收到了这个快递，会给他点一个电话或者发一个短信，这就能确保物流传输确实传送到了指定的地址，TCP协议也是一样的，客户端发送一个数据，服务端收到了会 回给客户端一个相应。


## 二 UDP协议数据传输

UDP数据传输比较简单，他叫“不可靠传输协议”，他不建立连接，也是基于ip和端口发送，只要知道对方的ip和端口就会对应的发，每次发都是这么发，根本就不会等对方确认，他就发过去就完事了，就像咱们有些同学打卡一样，这叫“不可靠打卡”，不管自己发的内容是什么，别人能不能看的懂，只要发过去，自己的活就干完了，那么这样的话这个效率肯定会比其他同学高。同理，UDP传输的效率会比TCP协议高，因为他不需要确认信息，不需要考虑接收的这一方是什么情况 ，就给你这么一直发。所以：

```
TCP协议的好处是数据安全，但是效率低
UDP协议的好处是效率高，但是数据不安全
```

那么我们应该使用TCP协议还是UDP协议呢?
对于不同的情况，我们的选择是不一样的，比如你的qq里面的聊天信息，你说了两句：“你好啊，我能跟你扯扯犊子吗”类似这样的废话，丢了也无关紧要，早期的qq聊天中就会经常出现丢包的现象，包括现在也还会有。但是如果涉及到转账，你转了1000块钱给烧烤店老板，他能不能收到全凭缘分，这样估计你就要被烤了，对于这种情况，我们就一定要用TCP协议去做了。所以，为了确保数据安全，我们用的比较多还是TCP，一些与查询相关的少量数据信息会使用UDP协议。