## TCP协议和UDP协议的区别

**TCP**（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“握手”才能建立起来，简单过程：

①主机A向主机B发出连接请求数据包，这是第一次对话；

②主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包，这是第二次对话；

③主机A再发出一个数据包确认主机B的要求同步，这是第三次对话。

三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。

**TCP三次握手过程**

1 主机A通过向主机B 发送一个含有同步序列号(SYN)的标志位的数据段给主机B ,向主机B 请求建立连接,通过这个数据段,主机A告诉主机B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.

2 主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机A,也告诉主机A两件事:我已经收到你的请求了,你可以传输数据了;你要用哪个序列号作为起始数据段来回应我

3 主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了

这样3次握手就完成了,主机A和主机B 就可以传输数据了.

**3次握手的特点**

没有应用层的数据

SYN这个标志位只有在TCP建立连接时才会被置1

握手完成后SYN标志位被置0

**TCP建立连接要进行3次握手,而断开连接要进行4次**

1 当主机A完成数据传输后,将控制位FIN置1,提出停止TCP连接的请求

2 主机B收到FIN后对其作出响应,确认这一方向上的TCP连接将关闭,将ACK置1

3 由B 端再提出反方向的关闭请求,将FIN置1

4 主机A对主机B的请求进行确认,将ACK置1,双方向的关闭结束.

由TCP的三次握手和四次断开可以看出,TCP使用面向连接的通信方式,大大提高了数据通信的可靠性,使发送数据端和接收端在数据正式传输前就有了交互,为数据正式传输打下了可靠的基础。

**名词解释**

ACK TCP报头的控制位之一,对数据进行确认.确认由目的端发出,用它来告诉发送端这个序列号之前的数据段都收到了.比如,确认号为X,则表示前X-1个数据段都收到了,只有当ACK=1时,确认号才有效,当ACK=0时,确认号无效,这时会要求重传数据,保证数据的完整性.

SYN 同步序列号,TCP建立连接时将这个位置1

FIN 发送端完成发送任务位,当TCP完成数据传输需要断开时,提出断开连接的一方将这位置1

**TCP的包头结构：**

源端口 16位

目标端口 16位

序列号 32位

回应序号 32位

TCP头长度 4位

reserved 6位

控制代码 6位

窗口大小 16位

偏移量 16位

校验和 16位

选项 32位(可选)

这样我们得出了TCP包头的最小长度，为20字节。

**UDP（User Data Protocol，用户数据报协议）**

（1） UDP是一个非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

（2） 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

（3） UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小。

（4） 吞吐量不受拥塞控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。

（5）UDP使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。

（6）UDP是面向报文的。发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付给IP层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

我们经常使用“ping”命令来测试两台主机之间TCP/IP通信是否正常，其实“ping”命令的原理就是向对方主机发送UDP数据包，然后对方主机确认收到数据包，如果数据包是否到达的消息及时反馈回来，那么网络就是通的。

**UDP的包头结构：**

源端口 16位

目的端口 16位

长度 16位

校验和 16位

**小结TCP与UDP的区别：**

1.基于连接与无连接；

2.对系统资源的要求（TCP较多，UDP少）；

3.UDP程序结构较简单；

4.面向字节流模式与数据报模式 ；

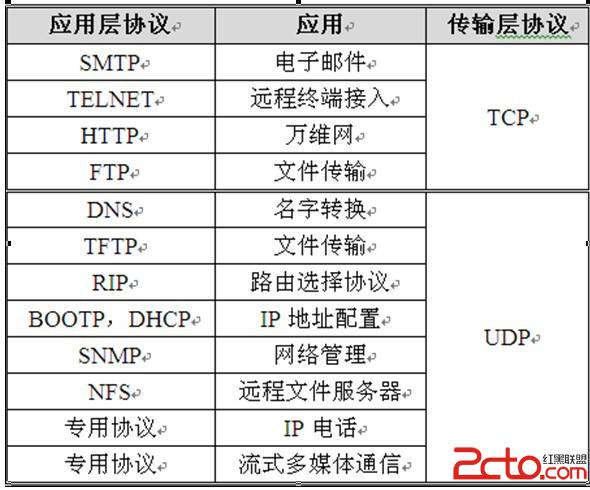
5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

## 面向报文（UDP）和面向字节流（TCP）的区别

**面向报文**的传输方式是应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。因此，应用程序必须选择合适大小的报文。若报文太长，则IP层需要分片，降低效率。若太短，会是IP太小。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。这也就是说，应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。

**面向字节流**的话，虽然应用程序和TCP的交互是一次一个数据块（大小不等），但TCP把应用程序看成是一连串的无结构的字节流。TCP有一个缓冲，当应用程序传送的数据块太长，TCP就可以把它划分短一些再传送。如果应用程序一次只发送一个字节，TCP也可以等待积累有足够多的字节后再构成报文段发送出去。

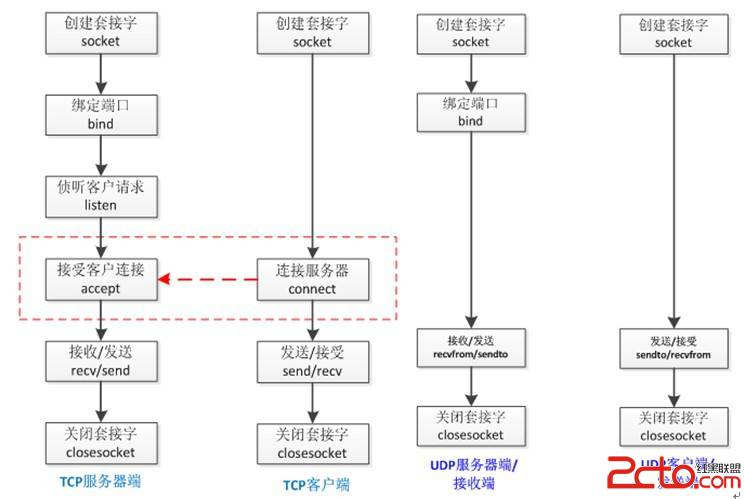
下图是TCP和UDP协议的一些应用。



下图是TCP和UDP协议的比较。



 从程序实现的角度来看，可以用下图来进行描述。



从上图也能清晰的看出，TCP通信需要服务器端侦听listen、接收客户端连接请求accept，等待客户端connect建立连接后才能进行数据包的收发（recv/send）工作。而UDP则服务器和客户端的概念不明显，服务器端即接收端需要绑定端口，等待客户端的数据的到来。后续便可以进行数据的收（recvfrom/sendto）工作。

在前面讲解UDP时，提到了UDP保留了报文的边界，下面我们来谈谈TCP和UDP中报文的边界问题。在默认的阻塞模式下，TCP无边界，UDP有边界。

对于TCP协议，客户端连续发送数据，只要服务端的这个函数的缓冲区足够大，会一次性接收过来，即客户端是分好几次发过来，是有边界的，而服务端却一次性接收过来，所以证明是无边界的；

 而对于UDP协议，客户端连续发送数据，即使服务端的这个函数的缓冲区足够大，也只会一次一次的接收，发送多少次接收多少次，即客户端分几次发送过来，服务端就必须按几次接收，从而证明，这种UDP的通讯模式是有边界的。

TCP无边界，造成对采用TCP协议发送的数据进行接收比较麻烦，在接收的时候易出现**粘包**，即发送方发送的若干包数据到接收方接收时粘成一包。 由于TCP是流协议，对于一个socket的包，如发送 10AAAAABBBBB两次，由于网络原因第一次又分成两次发送， 10AAAAAB和BBBB，如果接包的时候先读取10(包长度)再读入后续数据，当接收得快，发送的慢时，就会出现先接收了 10AAAAAB,会解释错误 ,再接到BBBB10AAAAABBBBB，也解释错误的情况。这就是TCP的粘包。

**粘包出现原因**：在流传输中出现，UDP不会出现粘包，因为它有消息边界

1 发送端需要等缓冲区满才发送出去，造成粘包  
2 接收方不及时接收缓冲区的包，造成多个包接收

在网络传输应用中，通常需要在网络协议之上再自定义一个协议封装一下，简单做法就是在要发送的数据前面再加一个自定义的包头，包头中可以包含数据长度和其它一些信息，接收的时候先收包头，再根据包头中描述的数据长度来接收后面的数据。详细做法是：先接收包头，在包头里指定包体长度来接收。设置包头包尾的检查位（ 比如以0xAA开头，0xCC结束来检查一个包是否完整）。对于TCP来说：

1. 不存在丢包，错包，所以不会出现数据出错 ；

2）如果包头检测错误，即为非法或者请求，直接重置即可。

为了**避免粘包现象**，可采取以下几种措施。

        一、对于发送方引起的粘包现象，用户可通过[编程](http://www.2cto.com/kf)设置来避免，TCP提供了强制数据立即传送的操作指令push，TCP软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；

        二、对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；

         三、由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。

以上提到的三种措施，都有其**不足之处:**

第一种编程设置方法虽然可以避免发送方引起的粘包，但它关闭了优化算法，降低了网络发送效率，影响应用程序的性能， 一般不建议使用。

第二种方法只能减少出现粘包的可能性，但并不能完全避免粘包，当发送频率较高时，或由于网络突发可能使某个时间段数据包到达接收方较快， 接收方还是有可能来不及接收，从而导致粘包。

第三种方法虽然避免了粘包，但应用程序的效率较低，对实时应用的场合不适合。

一般所谓的TCP粘包是在一次接收数据不能完全地体现一个完整的消息数据。TCP通讯为何存在粘包呢？主要原因是TCP是以流的方式来处理数据，再加上网 络上MTU的往往小于在应用处理的消息数据，所以就会引发一次接收的数据无法满足消息的需要，导致粘包的存在。处理粘包的唯一方法就是制定应用层的数据通讯协议，通过协议来规范现有接收的数据是否满足消息数据的需要。在应用中处理粘包的基础方法主要有两种分别是以4节字描述消息大小或以结束符，实际上也有两者相结合的如HTTP,redis的通讯协议等。

## [阻塞、非阻塞、同步、异步浅析(转)](http://www.cnblogs.com/bizhu/archive/2012/05/17/2506197.html)

在网络编程中，我们经常会遇到阻塞、非阻塞、同步、异步这两组概念，许多人很容易混淆，现在就给大伙儿说说我对着两组概念的理解。

先说说阻塞与非阻塞，这主要和程序等待消息时的状态有关

**1、阻塞**

程序会阻塞在某一个函数，而不往下执行，就如挂在那里一样，所有的其他业务也都不执行，为一直等到消息到来才往下执行。

**2、非阻塞**

程序不会阻塞在某一个函数，不等待消息到来，立即返回，往下执行。

举个例子，TCP协议中的send，系统会为其分配一块发送缓存区，假设现在总的缓存 区的大小为1000.，而缓存区里已经有了500个数据，这时调用send,发送1000个字节数据，这时如果在阻塞模式下，send的会先最多的数据放入缓存，直到缓存区满，然后程序就会阻塞在那里，一直等到将所有数据全部发出去为止，而如果在非阻塞模式下，send的会先将最多的数据放入缓存之后，就马上返回，剩余数据下回接着发送，不会卡在send函数中。

同步和异步这两个概念，其实是与消息通知机制有关的。

**1、同步**

发送消息，等待消息处理完后，才往下执行。就如MFC里的SendMessage(),

**2、异步**

发送消息，不等待消息处理完，就往下执行，让后通过特定的接口或者事件，消息通知你事情完成了。如MFC里的PostMessage()

粗略的讲了一下以上的两个概念，估计大家会问，那么阻塞与同步，非阻塞与异步有啥区别，用例说明。

比如我们现在去银行，你可以选择两种方式，取票等待和排队等待，如果你排队等待，你就得等前面所有人都办理完了，才能办理业务，此时如果在等待的过 程中，你啥事都不能干，这时你就处于阻塞状态了，如果你还能一边打电话，一边喝饮料，看书，不过你还得时不时的抬头看看，前面的人还有多少，啥时候轮到 你，免的错过办理，这时你就是处于同步状态。

也就是说，阻塞就是程序挂在那里，其他一切事情都不能做，直到当前事件返回为止，而同步得再原地等待消息，而与此同时，不影响其他业务的执行，体现在程序里就是其他线程的业务处理。

如果你取票等待，那么你只要坐在椅子上，这时你可以做你想做的事情，比如听听歌，看看书，打打电话，而到你时，窗口会自动叫号通知你，这时你就处于非阻塞状态。

如果此时你还觉得不过瘾，想出去溜达溜达，那么你可以和大厅的工作人员说一下，我去哪里，待会到我了，到某某地方通知我一下，然后你就可以出去做你 想做的事情，直到工作人员来叫你为止。这时你就是处于异步状态。两者结合起来就是所谓的异步非阻塞模式，这种由于效率很高，在网络编程里经常被用到。