**1、什么是socket**

socket 的原意是“插座”，在计算机通信领域，socket 被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通过 socket 这种约定，一台计算机可以接收其他计算机的数据，也可以向其他计算机发送数据。



把插头插到插座上就能从电网获得电力供应，同样，为了与远程计算机进行数据传输，需要连接到因特网，而 socket 就是用来连接到因特网的工具。

* 1. UNIX/Linux 中的 socket 是什么？

在 UNIX/Linux 系统中，为了统一对各种硬件的操作，简化接口，不同的硬件设备也都被看成一个文件。对这些文件的操作，等同于对磁盘上普通文件的操作。

“UNIX/Linux 中的一切都是文件”，为了表示和区分已经打开的文件，UNIX/Linux 会给每个文件分配一个 ID，被称为文件描述符（File Descriptor）。例如：

通常用 0 来表示标准输入文件（stdin），它对应的硬件设备就是键盘；

通常用 1 来表示标准输出文件（stdout），它对应的硬件设备就是显示器。

UNIX/Linux 程序在执行任何形式的 I/O 操作时，都是在读取或者写入一个文件描述符。一个文件描述符只是一个和打开的文件相关联的整数，它的背后可能是一个硬盘上的普通文件、FIFO、管道、终端、键盘、显示器，甚至是一个网络连接。

请注意，网络连接也是一个文件，它也有文件描述符！。

我们可以通过 socket() 函数来创建一个网络连接，或者说打开一个网络文件，socket() 的返回值就是文件描述符。有了文件描述符，我们就可以使用普通的文件操作函数来传输数据了，例如：

用 read() 读取从远程计算机传来的数据；

用 write() 向远程计算机写入数据。

你看，只要用 socket() 创建了连接，剩下的就是文件操作了，网络编程就简单了。

* 1. Window 系统中的 socket 是什么？

Windows 也有类似“文件描述符”的概念，但通常被称为“文件句柄”。因此，本教程如果涉及 Windows 平台将使用“句柄”，如果涉及 Linux 平台则使用“描述符”。

与 UNIX/Linux 不同的是，Windows 会区分 socket 和文件，Windows 就把 socket 当做一个网络连接来对待，因此需要调用专门针对 socket 而设计的数据传输函数，针对普通文件的输入输出函数就无效了。

socket是用来让一个进程与其他进程互通信息(IPC)，socket接口是TCP/IP网络的API接口函数。

**2、socket的类型**

套接字类型有很多，例如Internet套接字、Unix套接字、X.25套接字、RAW套接字。这里只讲第一种——Internet 套接字，它最具有代表性，也是最经典常用的。后面提及的套接字，都是指的Internet套接字。

在这里，根据数据传输方式将Internet套接字分为流格式套接字和数据报格式套接字。

**2.1 流格式套接字（SOCK\_STREAM）**

流格式套接字（Stream Sockets）也叫“面向连接的套接字”，在代码中使用 SOCK\_STREAM 表示。SOCK\_STREAM 是一种可靠的、双向的通信数据流，数据可以准确无误地到达另一台计算机，如果损坏或丢失，可以重新发送。

SOCK\_STREAM 有以下几个特征：

数据在传输过程中不会消失；

数据是按照顺序传输的；

数据的发送和接收不是同步的（有的教程也称“不存在数据边界”）。

可以将 SOCK\_STREAM 比喻成一条传送带，只要传送带本身没有问题（不会断网），就能保证数据不丢失；同时，较晚传送的数据不会先到达，较早传送的数据不会晚到达，这就保证了数据是按照顺序传递的。

由于使用了 TCP 协议（The Transmission Control Protocol，传输控制协议），TCP 协议会控制你的数据按照顺序到达并且没有错误，流格式套接字可以达到高质量的数据传输。

**“TCP/IP协议”中，TCP 用来确保数据的正确性，IP（Internet Protocol，网络协议）用来控制数据如何从源头到达目的地，也就是常说的“路由”。**

假设传送带传送的是水果，接收者需要凑齐 100 个后才能装袋，但是传送带可能把这 100 个水果分批传送，比如第一批传送 20 个，第二批传送 50 个，第三批传送 30 个。接收者不需要和传送带保持同步，只要根据自己的节奏来装袋即可，不用管传送带传送了几批，也不用每到一批就装袋一次，可以等到凑够了 100 个水果再装袋。

流格式套接字的内部有一个缓冲区（也就是字符数组），通过 socket 传输的数据将保存到这个缓冲区。接收端在收到数据后并不一定立即读取，只要数据不超过缓冲区的容量，接收端有可能在缓冲区被填满以后一次性地读取，也可能分成好几次读取。

也就是说，不管数据分几次传送过来，接收端只需要根据自己的要求读取，不用非得在数据到达时立即读取。传送端有自己的节奏，接收端也有自己的节奏，它们是不一致的。

浏览器所使用的 http 协议就基于面向连接的套接字，因为必须要确保数据准确无误，否则加载的 HTML 将无法解析。

**2.1 数据报格式套接字（SOCK\_DGRAM）**

数据报格式套接字（Datagram Sockets）也叫“无连接的套接字”，在代码中使用 SOCK\_DGRAM 表示。

计算机只管传输数据，不作数据校验，如果数据在传输中损坏，或者没有到达另一台计算机，是没有办法补救的。也就是说，数据错了就错了，无法重传。

因为数据报套接字所做的校验工作少，所以在传输效率方面比流格式套接字要高。

可以将 SOCK\_DGRAM 比喻成高速移动的摩托车快递，它有以下特征：

强调快速传输而非传输顺序；

传输的数据可能丢失也可能损毁；

限制每次传输的数据大小；

数据的发送和接收是同步的（有的教程也称“存在数据边界”）。

众所周知，速度是快递行业的生命。用摩托车发往同一地点的两件包裹无需保证顺序，只要以最快的速度交给客户就行。这种方式存在损坏或丢失的风险，而且包裹大小有一定限制。因此，想要传递大量包裹，就得分配发送。

另外，用两辆摩托车分别发送两件包裹，那么接收者也需要分两次接收，所以“数据的发送和接收是同步的”；换句话说，接收次数应该和发送次数相同。

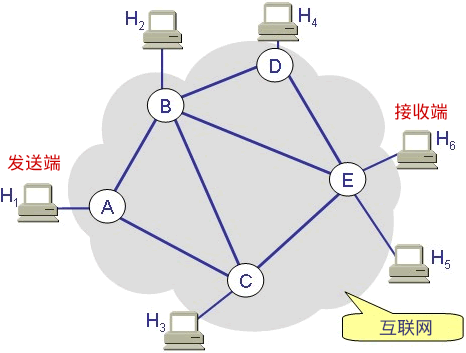
总之，数据报套接字是一种不可靠的、不按顺序传递的、以追求速度为目的的套接字。

数据报套接字也使用 IP 协议作路由，但是它不使用 TCP 协议，而是使用 UDP 协议（User Datagram Protocol，用户数据报协议）。

**3、socket的区别**

流格式套接字（Stream Sockets）就是“面向连接的套接字”，它基于 TCP 协议；数据报格式套接字（Datagram Sockets）就是“无连接的套接字”，它基于 UDP 协议。

面向连接是可靠的通信，无连接也并不是不可靠的通信。不论数据以何传输方式，都是通过整个Internet网络的物理线路传输过去。从字面理解，面向连接好像有一个管道连接发送端和接收端，数据包通过整个管道进行连接；无连接情况下的数据包从发送端到接收端没有固定的线路，但是最终也能达到接收端。



上图是一个简化的互联网模型，H1 ~ H6 表示计算机，A~E 表示路由器，发送端发送的数据必须经过路由器的转发才能到达接收端。假设 H1 要发送若干个数据包给 H6，那么有多条路径可以选择，比如：

路径①：H1 --> A --> C --> E --> H6

路径②：H1 --> A --> B --> E --> H6

路径③：H1 --> A --> B --> D --> E --> H6

路径④：H1 --> A --> B --> C --> E --> H6

路径⑤：H1 --> A --> C --> B --> D --> E --> H6

**3.1 无连接的套接字**

对于无连接的套接字，每个数据包可以选择不同的路径，比如第一个数据包选择路径④，第二个数据包选择路径①，第三个数据包选择路径②……当然，它们也可以选择相同的路径，那也只不过是巧合而已。

每个数据包之间都是独立的，各走各的路，谁也不影响谁，除了迷路的或者发生意外的数据包，最后都能到达 H6。但是，到达的顺序是不确定的，比如：

* 第一个数据包选择了一条比较长的路径（比如路径⑤），第三个数据包选择了一条比较短的路径（比如路径①），虽然第一个数据包很早就出发了，但是走的路比较远，最终还是晚于第三个数据包达到。
* 第一个数据包选择了一条比较短的路径（比如路径①），第三个数据包选择了一条比较长的路径（比如路径⑤），按理说第一个数据包应该先到达，但是非常不幸，第一个数据包走的路比较拥堵，这条路上的数据量非常大，路由器处理得很慢，所以它还是晚于第三个数据包达到了。

还有一些意外情况会发生，比如：

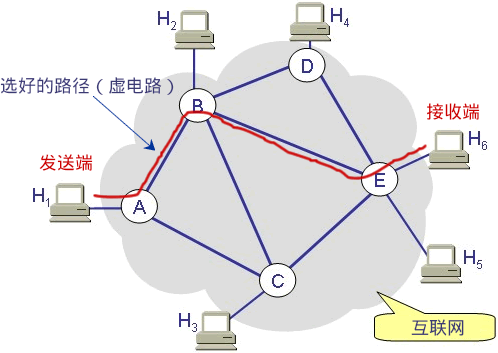
* 第一个数据包选择了路径①，但是路由器C突然断电了，那它就到不了 H6 了。
* 第三个数据包选择了路径②，虽然路不远，但是太拥堵，以至于它等待的时间太长，路由器把它丢弃了。

总之，对于无连接的套接字，数据包在传输过程中会发生各种不测，也会发生各种奇迹。H1 只负责把数据包发出，至于它什么时候到达，先到达还是后到达，有没有成功到达，H1 都不管了；H6 也没有选择的权利，只能被动接收，收到什么算什么，爱用不用。

无连接套接字遵循的是「尽最大努力交付」的原则，就是尽力而为，实在做不到了也没办法。无连接套接字提供的没有质量保证的服务。

**3.2 面向连接的套接字**

面向连接的套接字在正式通信之前要先确定一条路径，没有特殊情况的话，以后就固定地使用这条路径来传递数据包了。当然，路径被破坏的话，比如某个路由器断电了，那么会重新建立路径。



这条路径是由路由器维护的，路径上的所有路由器都要存储该路径的信息（实际上只需要存储上游和下游的两个路由器的位置就行），所以路由器是有开销的。

H1 和 H6 通信完毕后，要断开连接，销毁路径，这个时候路由器也会把之前存储的路径信息擦除。

在很多网络通信教程中，这条预先建立好的路径被称为“虚电路”，就是一条虚拟的通信电路。

为了保证数据包准确、顺序地到达，发送端在发送数据包以后，必须得到接收端的确认才发送下一个数据包；如果数据包发出去了，一段时间以后仍然没有得到接收端的回应，那么发送端会重新再发送一次，直到得到接收端的回应。这样一来，发送端发送的所有数据包都能到达接收端，并且是按照顺序到达的。

面向连接的套接字会比无连接的套接字多出很多数据包，因为发送端每发送一个数据包，接收端就会返回一个数据包。此外，建立连接和断开连接的过程也会传递很多数据包。

不但是数量多了，每个数据包也变大了：除了源端口和目的端口，面向连接的套接字还包括序号、确认信号、数据偏移、控制标志（通常说的 URG、ACK、PSH、RST、SYN、FIN）、窗口、校验和、紧急指针、选项等信息；而无连接的套接字则只包含长度和校验和信息。

有连接的数据包比无连接大很多，这意味着更大的负载和更大的带宽。许多即时聊天软件采用 UDP 协议（无连接套接字），与此有莫大的关系。

**3.3 总结**

两种套接字各有优缺点：

* 无连接套接字传输效率高，但是不可靠，有丢失数据包、捣乱数据的风险；
* 有连接套接字非常可靠，万无一失，但是传输效率低，耗费资源多。

两种套接字的特点决定了它们的应用场景，有些服务对可靠性要求比较高，必须数据包能够完整无误地送达，那就得选择有连接的套接字（TCP 服务），比如 HTTP、FTP 等；而另一些服务，并不需要那么高的可靠性，效率和实时才是它们所关心的，那就可以选择无连接的套接字（UDP 服务），比如 DNS、即时聊天工具等。