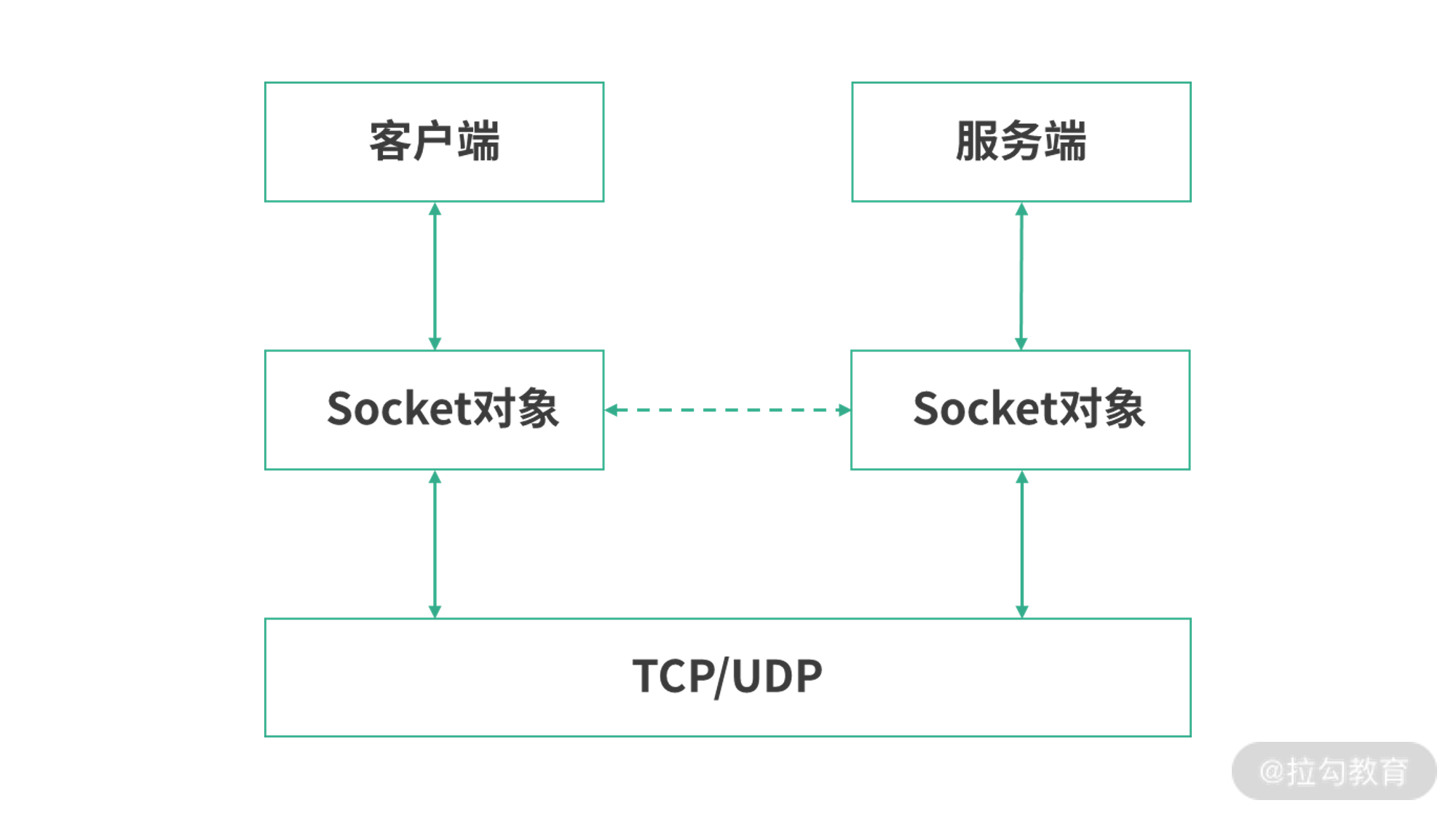
## **Socket是什么？**

Socket是一种编程模型，从编程角度来看，客户端数据发送给在客户端侧的Socket对象，然后客户端侧的Socket对象将数据发送给服务端侧的Socket对象。Socket对象负责提供数据通信能力，并处理底层的TCP/UDP 连接。对服务端而言，每一个客户端接入，就会形成一个和客户端对应的Socket对象。如果服务器要读取客户端发送的信息，或者向客户端发送信息，就会需要通过这个客户端Socket对象



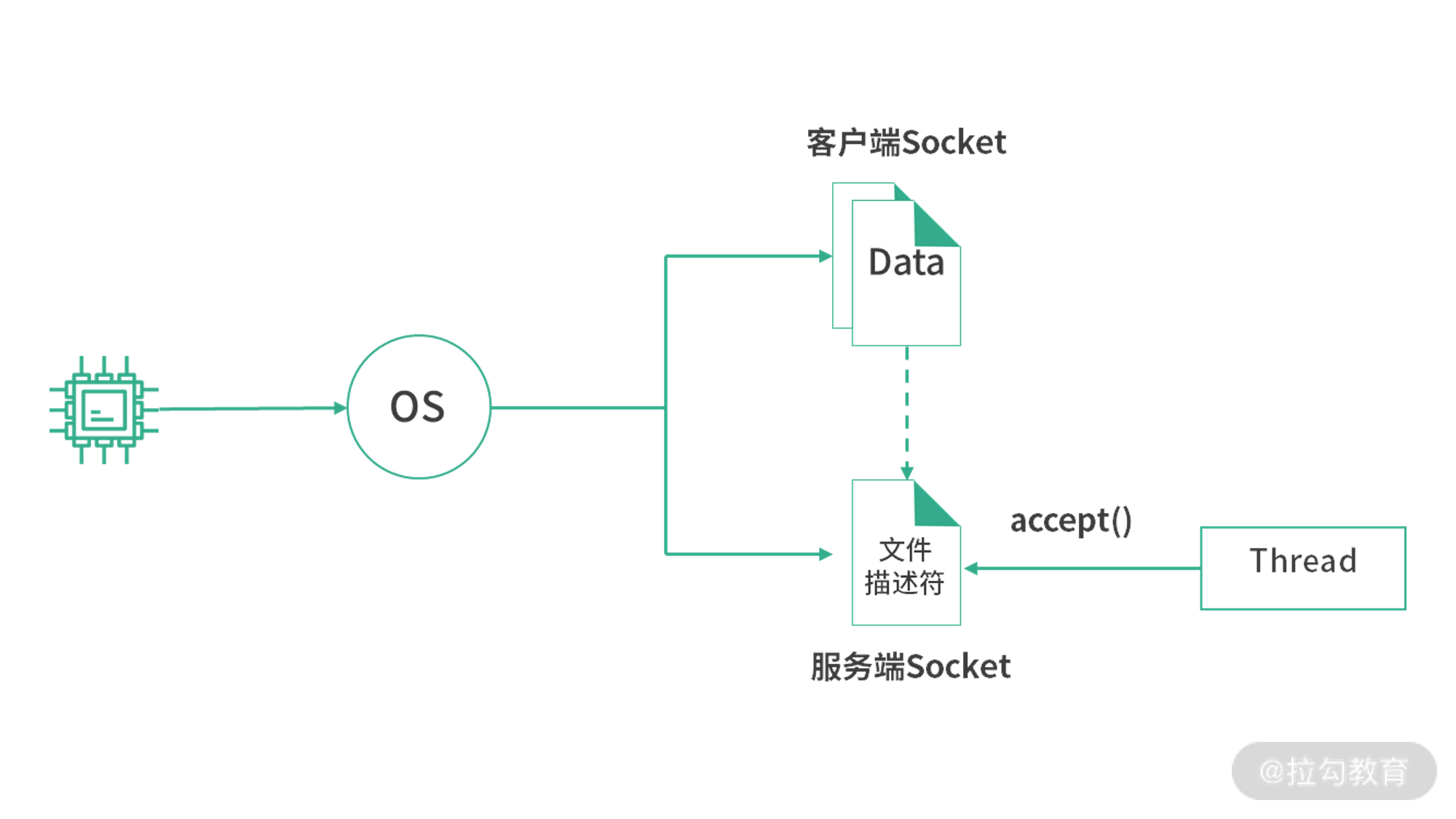
从另一个角度去分析，Socket还是一种文件，准确的所说是一种双向管道文件。管道文件会将一个程序的输出导向另一个程序的输入。双向管道文件连接的程序是对等的，都可以作为输入输出。

var serverSocket = new ServerSocket();

serverSocket.bind(new InetSocketAddress(80));

上述代码创建了一个服务端的Socket对象，如果从管道文件的层面可以理解问这是一个文件，它里面存储了所有客户端Socket文件的文件描述符。

当一个客户端连接到服务的时候，操作系统就会创建一个客户端Socket文件。然后，操作系统将这个文件的文件描述符写入服务端程序创建的服务端Socket文件中。服务端Socket文件，是一个管道文件。如果读取这个文件的内容，就相当于从管道中取出了一个客户端文件描述符。



如上图所示，服务端Socket文件相当于一个客户端Socekt的目录，线程可以通过accept操作每次拿走一个客户端文件描述符。拿到文件描述符就相当于拿到了和客户端进行通信的接口。

当线程想要读取客户端传输来的数据时，就从客户端socket文件中读取数据；当线程想要发送数据到客户端时，就向客户端Socket文件中写入数据。客户端Socket是一个双向管道，操作系统将客户端传来的数据写入管道，也将线程写入管道的数据发送到客户端。

既然Socket可以双向传送，那么是两个单向管道拼凑在一起实现的吗？这取决于操作系统。Linux中的管道是单向的。因此Socket文件是一种区别于操作系统管道的单独实现。

****总结一下，Socket首先是文件，存储的是数据。对服务端而言，分成服务端Socket文件和客户端Socket文件。服务端文件存储的是客户端文件描述符；客户端Socket文件存储的是传输数据。读取客户端Socket文件就是读取客户端发来的数据；写入客户端文件就是向客户端发送数据。对一个客户端而言，Socket文件存储的是发送给（或接收）服务端的数据****

****综上，Socket首先是文件，在文件的基础上又封装了一段程序，这段程序提供了API负责最终的数据传输。****

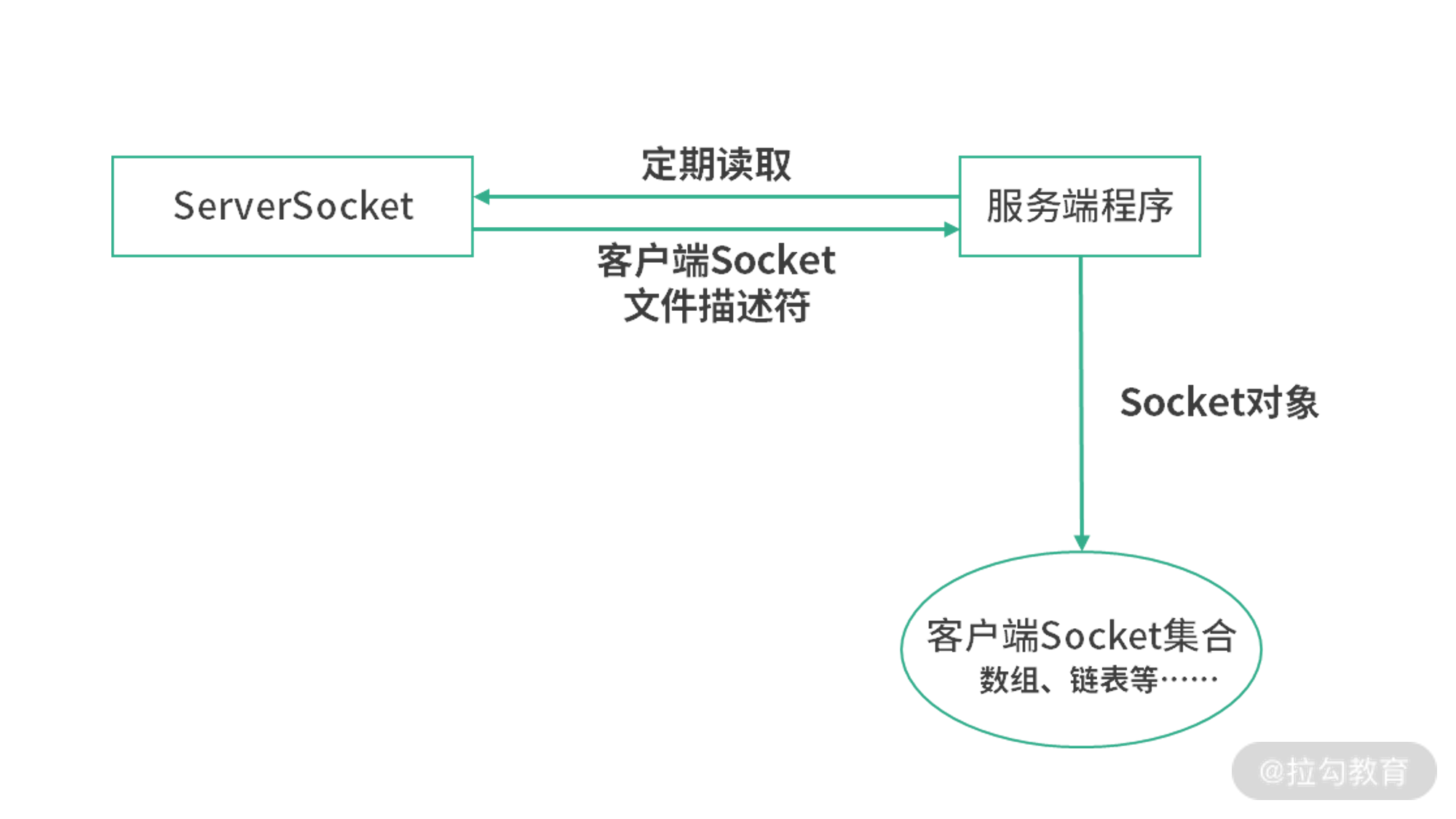
## **服务端Socket的绑定**

为了区别应用，对于一个服务端Socket文件，我们要设置它的监听端口，比如Nginx监听80端口、Node监听3000端口、SSH监听22端口、Tomcat监听8080端口。端口的监听不能重复，不然客户端连接进来创建客户端Socket文件，文件描述符就不知道写入哪个服务端Socket了。这样操作系统就会把连接到不同端口的客户端分类。将客户端Socket文件描述符存到对应不同端口的服务端Socket文件中。

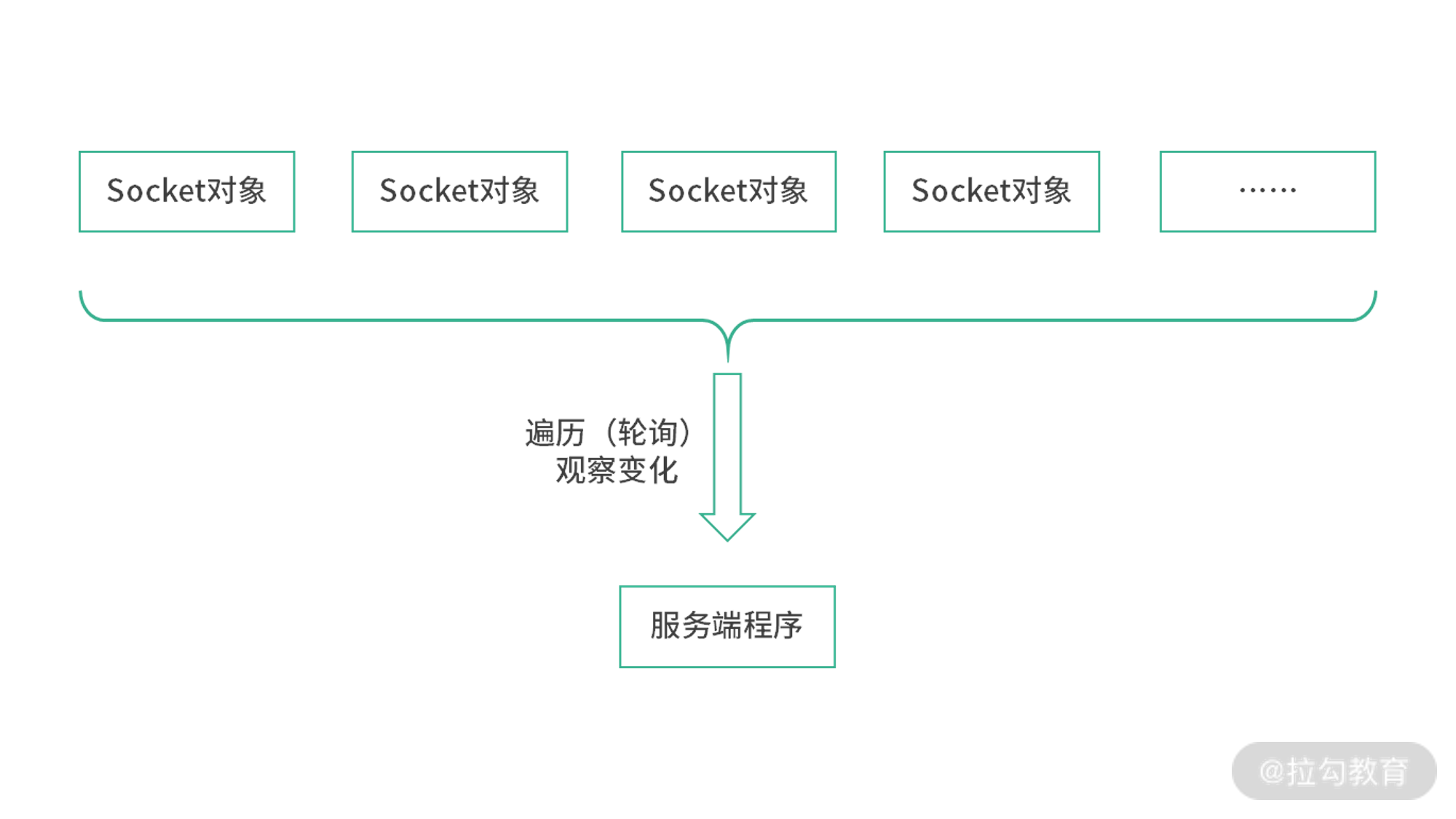
因此，服务端监听端口的本质是将服务端Socket文件和端口绑定，这个操作也称为bind。有时候不仅仅要绑定端口，还要绑定IP地址。因为有时候我们只允许指定IP访问我们的服务器程序。

## **扫描和监听**

对于服务端程序，可以定期扫描服务端文件的变更，来了解有哪些客户端想要连接进来。如果在服务端Socket文件中读取杜鳌一个客户端文件描述符，就可以将这个文件描述符实例化成一个Socket对象。



之后，服务端可以将这个Socket对象加入到一个集合，通过定期遍历所有客户端Socket对象，查找背后Socket文件的状态，从而确定是否有新的数据从客户端传输过来。



上述过程通过一个线程就可以响应多个客户端连接，也被称作****I/O多路复用技术****

## **响应式**

在I/O多路复用技术中，服务端程序需要维护一个Socket的集合，然后定期遍历这个集合。这样的做法在客户端Socket较少的情况下没有问题，但是，如果接入的客户端Socket较多，比如达到上万，每次轮训的开销就会很大。

从程序设计来看，像这样主动遍历，比如遍历一个Socket集合看看有没有发生写入称为命令式编程。这样的程序设计好像在执行一条条命令一样，程序主动的查看每个Socket的状态。

命令式会让负责下命令的程序负载过重。例如，在高并发场景下，上述讨论中循环遍历Socket集合的线程会因为负担过重导致系统吞吐量下降。

与命令式相反的是响应式。响应式的程序当中，每一个参与者有独立的思考方式。就好像拥有独立的人格，可以自己针对不同的环境出发不同的行为。

从响应式的角度看Socket编程，应该是有某个观察者会观察到Socket文件状态的变化，从而通知处理线程响应。线程不再需要遍历Socket集合，而是等待观察者程序的通知。

而最合适的观察者其实是操作系统本身，因为只有操作系统非常清楚每一个Socket文件的状态。原因是对Socket文件读写都要经过操作系统。在实现这个模型的时候，有几件事要注意。

线程需要高速中间的观察者自己要观察什么，或者说在什么情况下响应。比如具体到哪个Socket发生什么变化，是读写还是其他事件，这一步称为注册

中间的观察者需要实现一个高效的数据结构（通常是基于红黑树的二叉搜索树）。这是因为中间观察者不仅仅是服务某个线程，而是服务与很多线程。当一个Socket文件发生变化时，中间观察者要立刻知道究竟是哪个线程需要这个信息，而不是将所有线程都遍历一遍。

## **总结**

Socket即是一种编程模型或者说一端程序，同时也是一个文件，一个双向管道文件。Socket API是在Socket文件基础上进行一层封装，而Socket文件是操作系统提供支持的一种文件格式。

在服务端有两种Socket文件，每个客户端接入后会生成一个客户端Socket文件，客户端文件的文件描述符会存入服务端Socket文件。通过这种方式，一个线程可以通过读取服务端Socket文件中的内容拿到所有客户端Socket。这样一个线程就可以负责响应所有客户端的I/O，这个技术称为I/O多路复用

主动式的I/O多路复用对负责I/O的线程压力过大。因此，通常会涉及一个高线的中间数据结构作为I/O事件的观察者。线程通过订阅事件被动响应，这就是响应式模型。在Socket编程中，最适合提供这种中间数据结构的就是操作系统内核。事实上epoll模型也是在操作系统的内核中提供了红黑树结构。