咕泡学院 VIP 课：分布式系统的基石 TCP/IP 通信协议

课程目标

1. 通信协议在分布式架构中的核心应用
2. 深入了解 TCP/IP 和 UDP/IP 通信协议
3. TCP 流量整形
4. 基于 Java 自身技术实现系统通信
5. 多任务处理及优化
6. 了解什么是 NIO
7. 组播协议 Multicast

网络领域的知识

a) 协议：tcp、udp、multicast

b) IO（BIO、NIO、AIO）

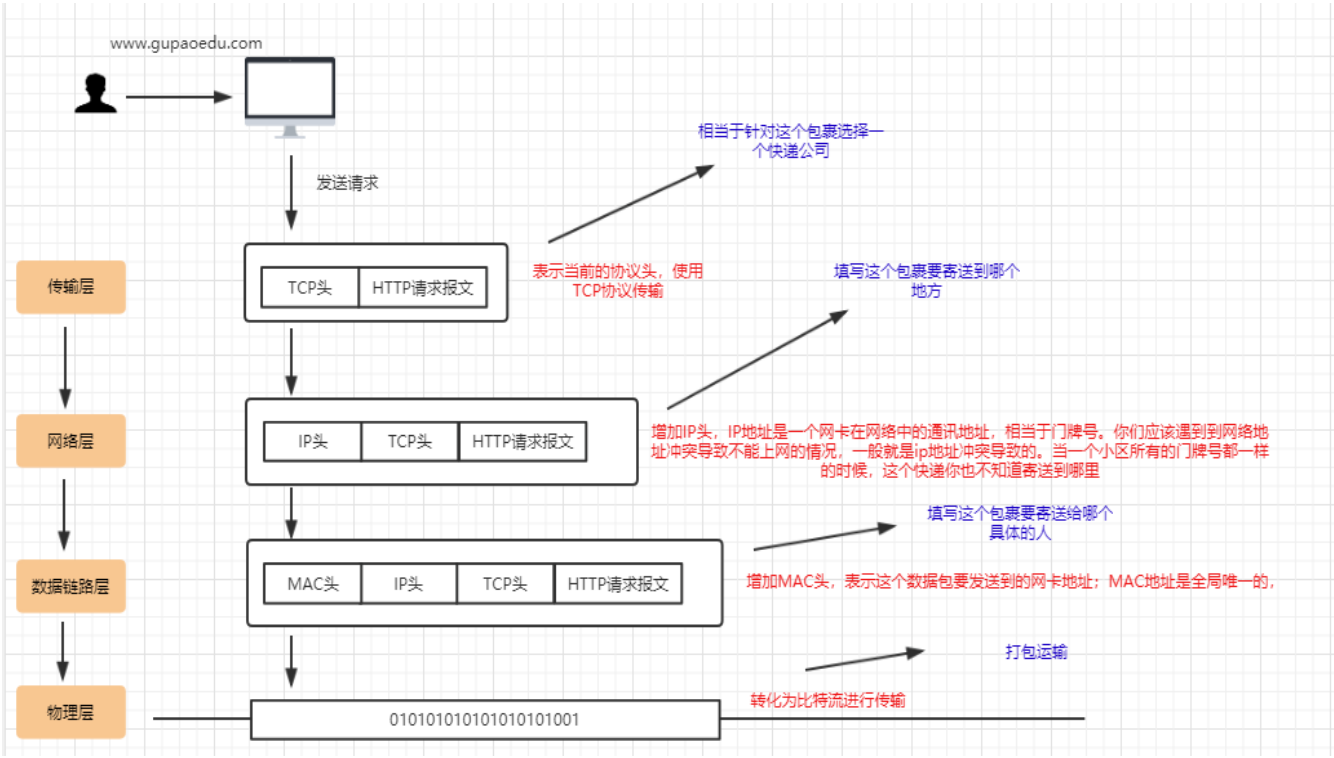
c) Socket

d) NIO(Netty/Mina)

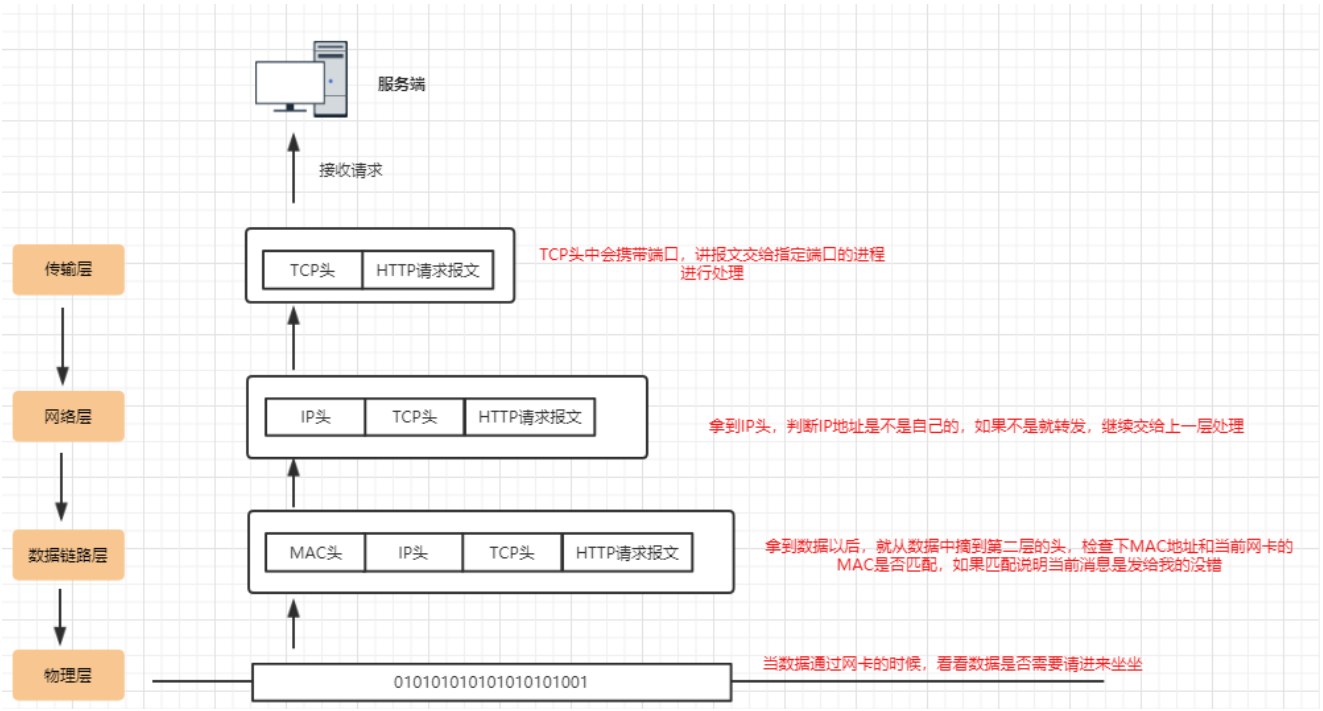
e) 序列化和反序列化

一个 http 请求，在整个网络中的请求过程

当应用程序用ＴＣＰ传送数据时，数据被送入协议栈中，然后逐个通过每一层直到被当作一串比特流送入网络。其中每一层对收到的数据都要增加一些首部信息



当目的主机收到一个以太网数据帧时，数据就开始从协议栈中由底向上升，同时去掉各层协议加上的报文首部。每层协议盒都要去检查报文首部中的协议标识，以确定接收数据的上层协议。这个过程称作分用



为什么有了 MAC 层还要走 IP 层呢?

mac 地址就好像个人的身份证号，人的身份证号和人户口所在的城市，出生的日期有关，但是和人所在的位置没有关系，人是会移动的，知道一个人的身份证号，并不能找到它这个人，mac 地址类似，它是和设备的生产者，批次，日期之类的关联起来，知道一个设备的 mac，并不能在网络中将数据发送给它，除非它和发送方的在同一个网络内。所以要实现机器之间的通信，还需要有 ip 地址的概念，ip 地址表达的是当前机器在网络中的位置，类似于城市名+道路号+门牌号的概念。通过 ip 层的寻址，我们能知道按何种路径在全世界任意两台 Internet 上的的机器间传输数据。

重点了解下 IP 协议和 TCP/UDP 协议

什么是协议

协议相当于两个需要通过网络通信的程序达成的一种约定，它规定了报文的交换方式和包含的意义。比如（HTTP）为了解决在服务器之间传递超文本对象的问题，这些超文本对象在服务器中创建和存储，并由 Web 浏览器进行可视化，完成用户对远程内容的感知和体验

什么是 IP 协议

T C P 和 U D P 是两种最为著名的传输层协议，他们都是使用 I P 作为网络层协议。IP 协议提供了一组数据报文服务，每组分组报文都是由网络独立处理和分发，就像寄送快递包裹一样，为了实现这个功能，每个 IP 报文必须包含一个目的地址的字段；就像我们寄送快递都需要写明收件人信息，但是和我们寄送快递一样，也可能会出现包裹丢失问题，所以 IP 协议只是一个"尽力而为"的协议，在网络传输过程中，可能会发生报文丢失、报文顺序打乱，重复发送的情况。IP 协议层之上的传输层，提供了两种可以选择的协议，TCP、UPD。这两种协议都是建立在 IP 层所提供的服务基础上，根据应用程序的不同需求选择不同方式的传输；

TCP/IP

TCP 协议能够检测和恢复 IP 层提供的主机到主机的通信中可能发生的报文丢失、重复及其他错误。TCP 提供了一个可信赖的字节流通道，这样应用程序就不需要考虑这些问题。同时，TCP 协议是一种面向连接的协议，在使用 TCP 进行通信之前，两个应用程序之间需要建立一个 TCP 连接，而这个连接又涉及到两台电脑需要完成握手消息的交换。

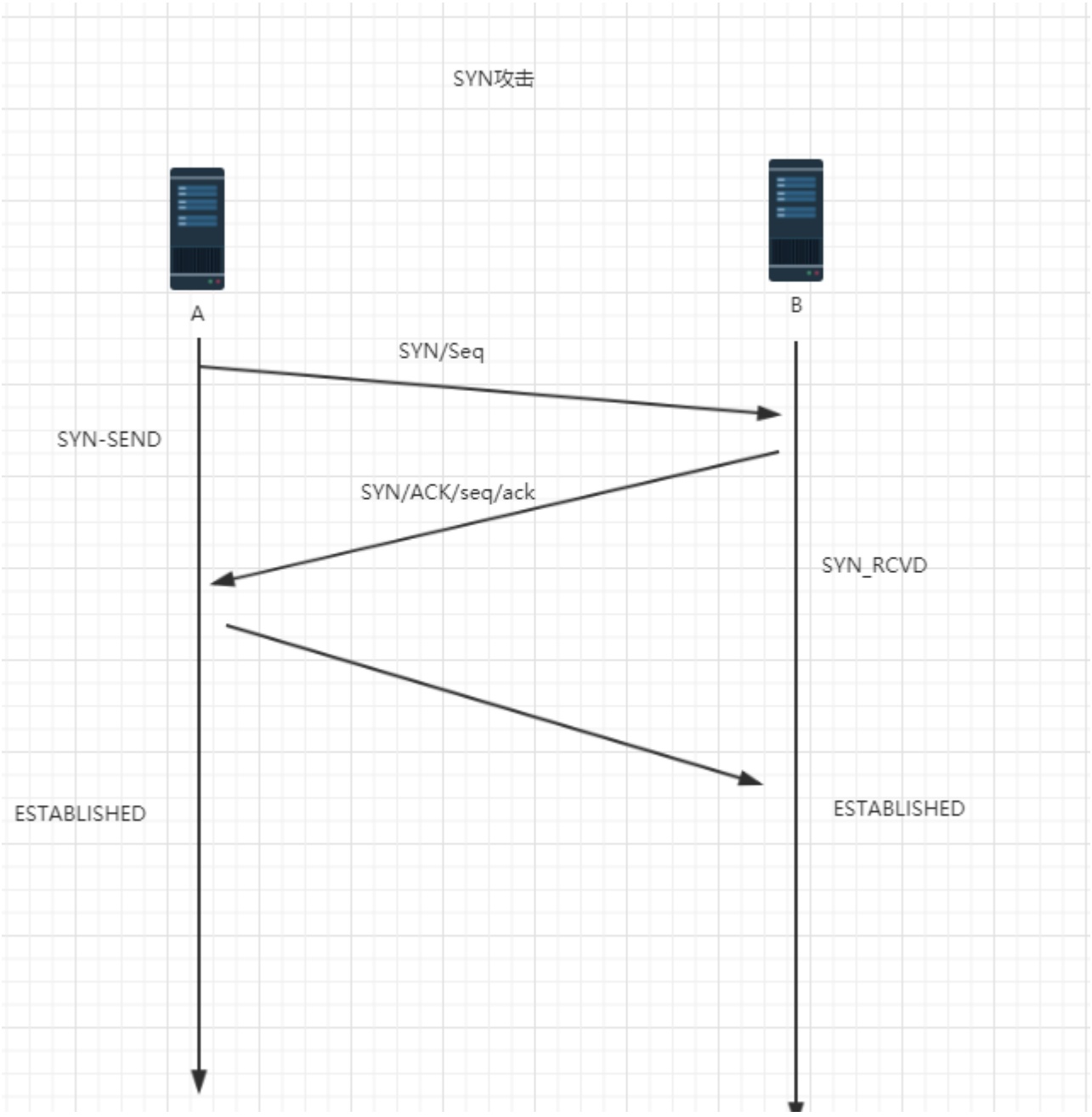
UDP/IP

UDP 协议不会对 IP 层产生的错误进行修复，而是简单的扩展了 IP 协议“尽力而为”的数据报文服务，使他能够在应用程序之间工作，而不是在主机之间工作，因此使用 UDP 协议必须要考虑到报文丢失，顺序混乱的问题

TCP 是如何做到可靠传输的？

建立可靠的链接

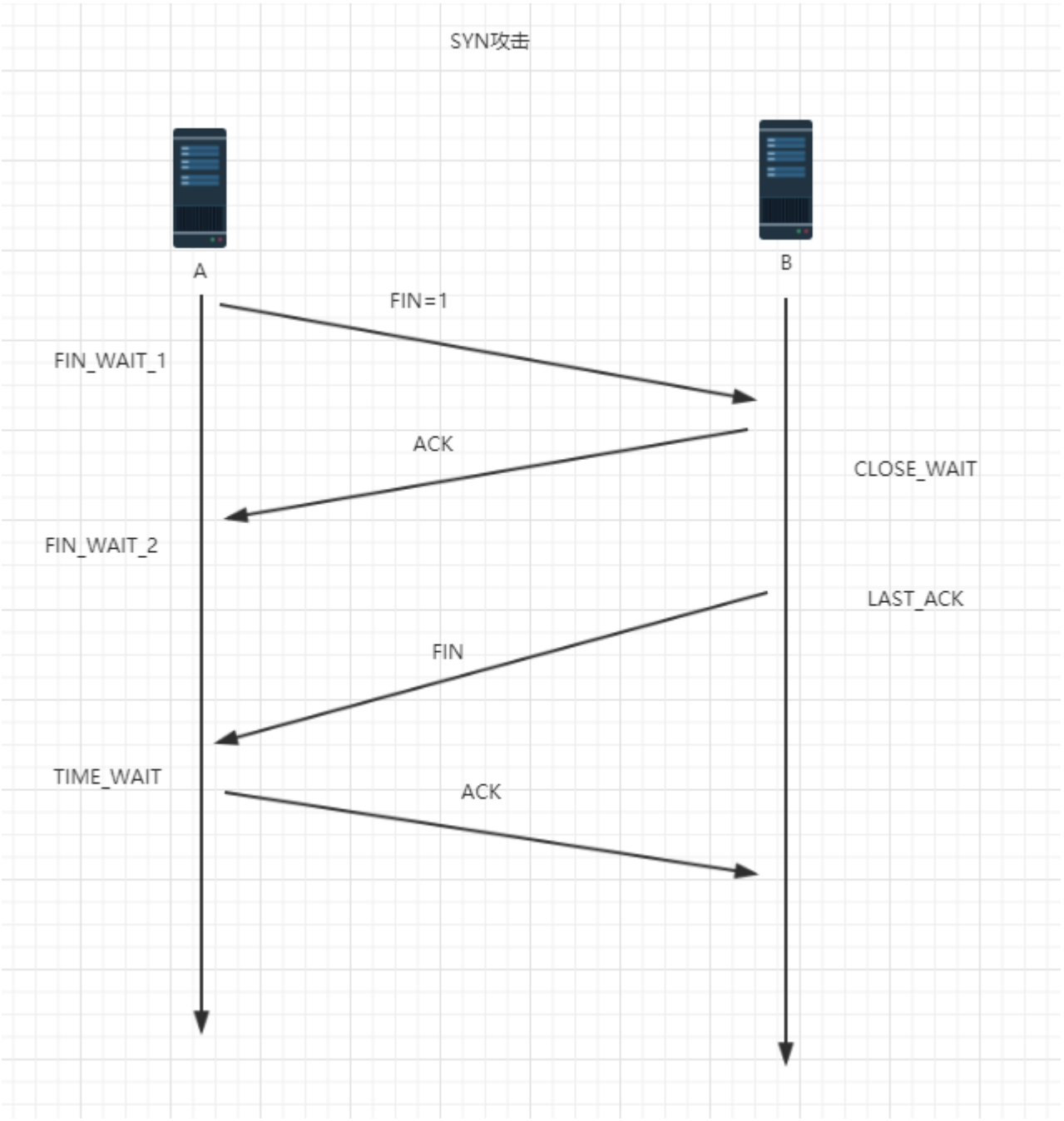
由于 TCP 协议是一种可信的传输协议，所以在传输之前，需要通过三次握手建立一个连接，所谓的三次握手，就是在建立 TCP 链接时，需要客户端和服务端总共发送 3 个包来确认连接的建立



TCP 四次挥手协议

四次挥手表示 TCP 断开连接的时候,需要客户端和服务端总共发送 4 个包以确认连接的断开；客户端或服务器均可主动发起挥手动作(因为 TCP 是一个全双工协议)，在 socket 编程中，任何一方执行 close() 操作即可产生挥手

操作。



tips:为什么连接的时候是三次握手，关闭的时候却是四次握手?

三次握手是因为因为当 Server 端收到 Client 端的 SYN 连接请求报文后，可以直接发送 SYN+ACK 报文。其中 ACK

报文是用来应答的，SYN 报文是用来同步的。但是关闭连接时，当 Server 端收到 FIN 报文时，很可能并不会立即关闭 SOCKET（因为可能还有消息没处理完），所以只能先回复一个 ACK 报文，告诉 Client 端，"你发的 FIN 报文我收到了"。只有等到我 Server 端所有的报文都发送完了，我才能发送 FIN 报文，因此不能一起发送。故需要四步握手。

数据传输过程的流量控制和确认机制

建立可靠连接以后，就开始进行数据传输了。在通信过程中，最重要的是数据包，也就是协议传输的数据。如果数据的传送与接收过程当中出现收方来不及接收的情况，这时就需要对发方进行控制以免数据丢失。利用滑动窗口机制可以很方便的在 TCP 连接上实现对发送方的流量控制。TCP 的窗口单位是字节，不是报文段，发送方的发送窗口不能超过接收方给出的接收窗口的数值。

滑动窗口协议

滑动窗口（Sliding window）是一种流量控制技术。早期的网络通信中，通信双方不会考虑网络的拥挤情况直接发送数据。由于大家不知道网络拥塞状况，同时发送数据，导致中间节点阻塞掉包，谁也发不了数据，所以就有了滑动

窗口机制来解决此问题；发送和接受方都会维护一个数据帧的序列，这个序列被称作窗口

简单解释下，发送和接受方都会维护一个数据帧的序列，这个序列被称作窗口。发送方的窗口大小由接受方确定，目的在于控制发送速度，以免接受方的缓存不够大，而导致溢出，同时控制流量也可以避免网络拥塞。下面图中的 4,5,6 号数据帧已经被发送出去，但是未收到关联的 ACK，7,8,9 帧则是等待发送。可以看出发送端的窗口大小为 6，这是由接受端告知的。此时如果发送端收到 4 号 ACK，则窗口的左边缘向右收缩，窗口的右边缘则向右扩展，此时窗口就向前“滑动了”，即数据帧 10 也可以被发送。

发送窗口

就是发送端允许连续发送的帧的序号表。
发送端可以不等待应答而连续发送的最大帧数称为发送窗口的尺寸。

接收窗口

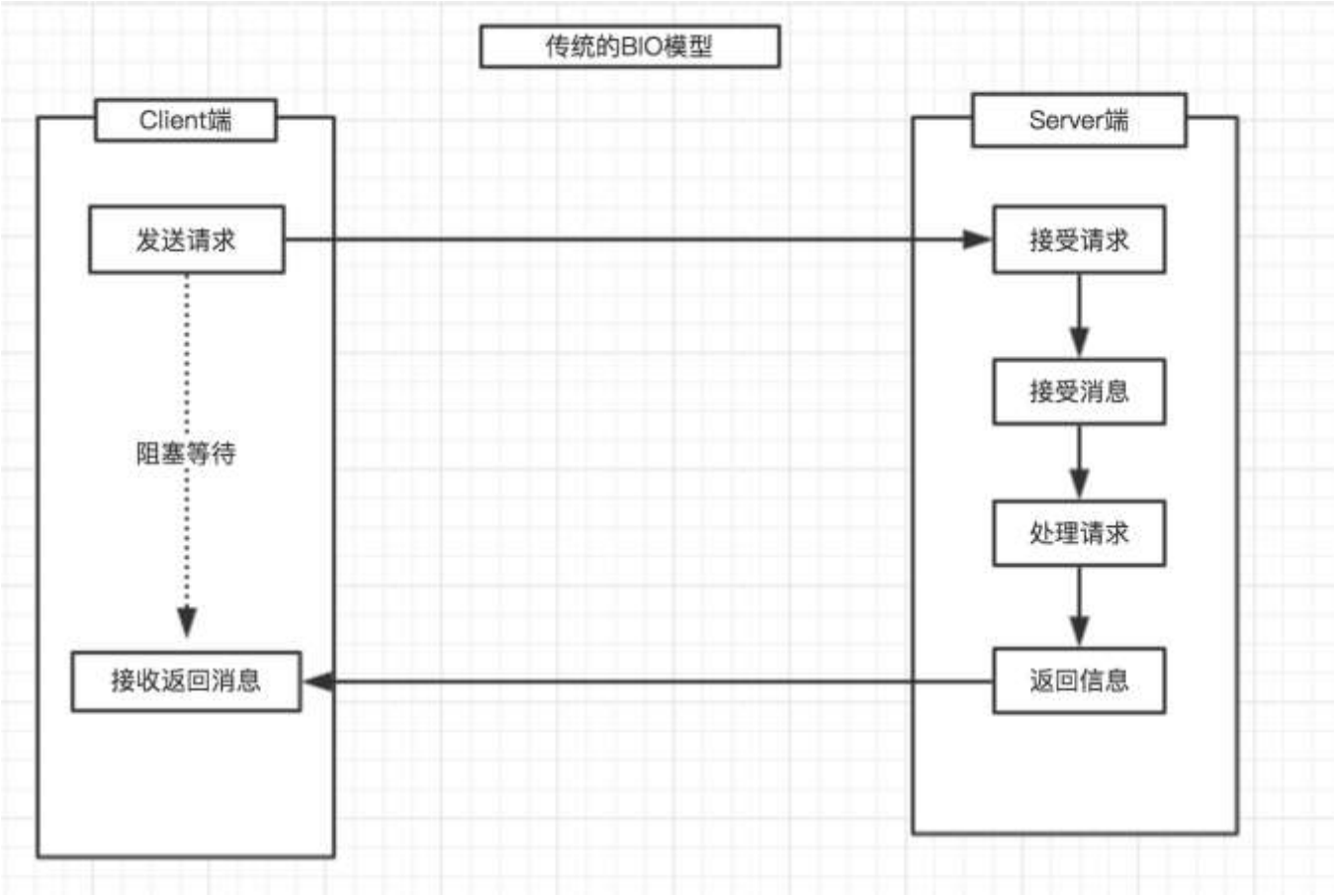
接收方允许接收的帧的序号表，凡落在 接收窗口内的帧，接收方都必须处理，落在接收窗口外的帧被丢弃。
接收方每次允许接收的帧数称为接收窗口的尺寸。
在线滑动窗口演示功能
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html

应用层是如何使用 tcp/udp 进行通信的

基于 socket 和 DatagramSocket 进行通信，请详见代码

通信的性能问题？

正常的通信过程如下(BIO)



我们发现 TCP 响应服务器一次只能处理一个客户端请求，当一个客户端向一个已经被其他客户端占用的服务器发送连接请求时，虽然在连接建立后可以向服务端发送数据，但是在服务端处理完之前的请求之前，却不会对新的客户端做出响应，这种类型的服务器称为“迭代服务器”。迭代服务器是按照顺序处理客户端请求，也就是服务端必须要处理完前一个请求才能对下一个客户端的请求进行响应。但是在实际应用中，我们不能接收这样的处理方式。所以我们需要一种方法可以独立处理每一个连接，并且他们之间不会相互干扰。而 Java 提供的多线程技术刚好满

足这个需求，这个机制使得服务器能够方便处理多个客户端的请求。

如何提高性能

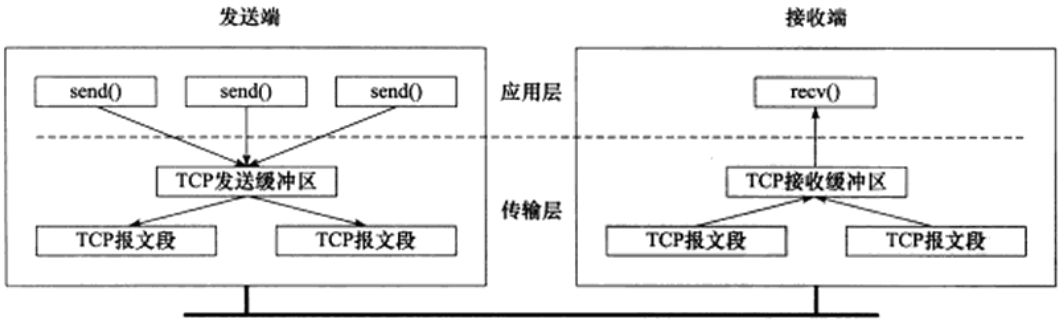
TCP 协议的通信过程

对于 TCP 通信来说，每个 TCP Socket 的内核中都有一个发送缓冲区和一个接收缓冲区，TCP 的全双工的工作模式及 TCP 的滑动窗口就是依赖于这两个独立的 Buffer 和该 Buffer 的填充状态。

接收缓冲区把数据缓存到内核，若应用进程一直没有调用 Socket 的 read 方法进行读取，那么该数据会一直被缓存在接收缓冲区内。不管进程是否读取 Socket，对端发来的数据都会经过内核接收并缓存到 Socket 的内核接收缓冲区。

read 所要做的工作，就是把内核接收缓冲区中的数据复制到应用层用户的 Buffer 里。

进程调用 Socket 的 send 发送数据的时候，一般情况下是将数据从应用层用户的 Buffer 里复制到 Socket 的内核发送缓冲区，然后 send 就会在上层返回。换句话说，send 返回时，数据不一定会被发送到对端。



Socket 的接收缓冲区被 TCP 用来缓存网络上收到的数据，一直保存到应用进程读走为止。如果应用进程一直没有读取，那么 Buffer 满了以后，出现的情况是：通知对端 TCP 协议中的窗口关闭，保证 TCP 接收缓冲区不会移除，保证了 TCP 是可靠传输的。如果对方无视窗口大小发出了超过窗口大小的数据，那么接收方会把这些数据丢弃。

如何使用非阻塞提高性能？

非阻塞要解决的就是 I/O 线程与 Socket 解耦的问题，因此，它引入了事件机制来达到解耦的目的。我们可以认为 NIO 底层中存在一个 I/O 调度线程，它不断的扫描每个 Socket 的缓冲区，当发现写入缓冲区为空的时候，它会产生一个 Socket 可写事件，此时程序就可以把数据写入到 Socket 中。如果一次写不完，就等待下一次的可写事件通知；反之，当发现缓冲区里有数据的时候，它会产生一个 Socket 可读事件，程序收到这个通知事件就可以从 Socket 读取数据了。

关于 NIO

实际上基于上面讲的传统的 BIO 模型，一个请求一个线程的方式，如果要涉及到上千个客户端访问时，会产生很多的问题，比如扩展性、系统资源开销等等。所以我们需要一种方法来轮询一组客户端，来查找哪个连接需要提供服务，这个就是我们讲的“NIO”；

缓冲区

在 NIO 中，所有数据都是用缓冲区处理，在读取数据的时候，它是直接读到缓冲区中，在写如数据的时候，也是写到缓冲区。任何时候访问 NIO 中的数据，都是通过缓冲区进行的操作

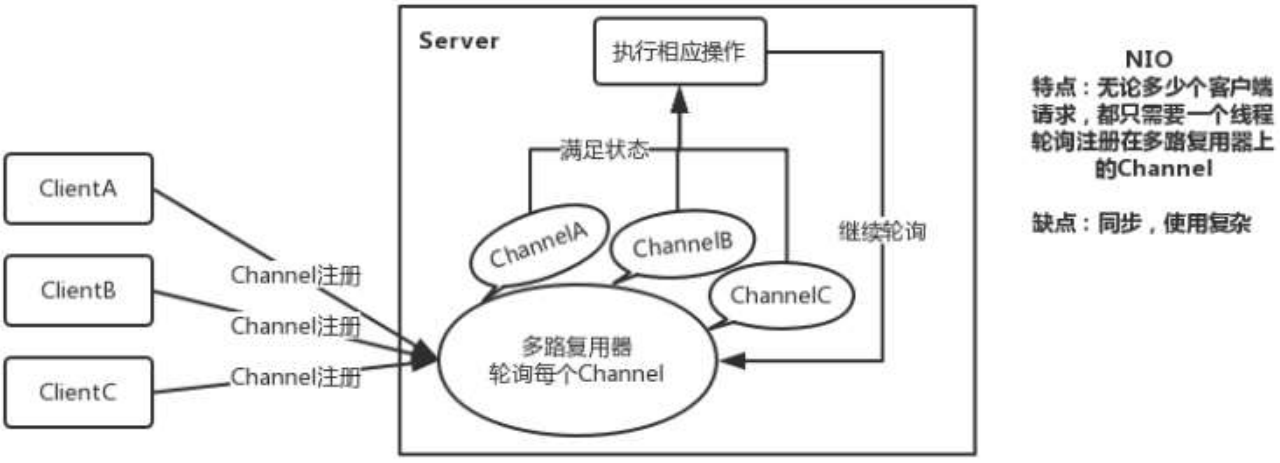
通道

Channel 通道，就像一个自来水管一样，可以通过它读取和写入数据，Channel 是全双工的，所以数据是双向流动。

多路复用

多路复用器 Selector，是 NIO 的基础，多路复用器提供选择已经就绪的任务的能力，简单来说，Selector 会不断轮询注册上的 Channel，如果某个 Channel 上面有新的 TCP 连接接入、读、写事件，这个 Channel 就处于就绪状态，

会被 Selector 轮询出来，然后通过 SelectionKey 可以获取就绪的 Channel 进行 I/O 操作；一个多路复用器可以同时轮询多个 Channel。通过这个机制可以接入成千上万的客户端。



组播协议 Multicast

对于某些信息，多个接受者都可能感兴趣的时候，那么我们应该怎么解决呢？我们可以向每个接受者单播一个数据副本，但是如果这样的话，效率会低；而且同样的数据发送多次，浪费带宽。

解决方案是，我们可以把复制数据包的工作交给网络来做，而不是由发送者负责。这样无论是多少客户端，都没问题

有两种分发类型，广播（broadcast）和多播（multicast）；

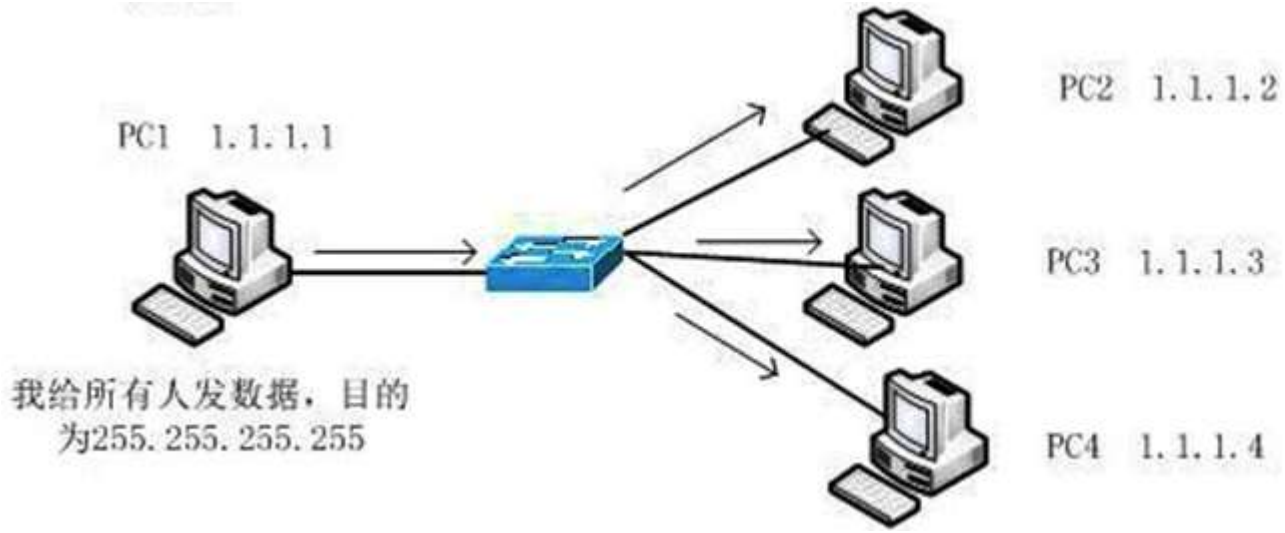
广播：网络中的所有主机都会接收到一份数据副本

多播：消息只发送给一个多播地址，网络只是将数据分发给哪些想要接收发送到该多播地址的数据的主机。

总的来说，要实现这个功能，只有 UDP 是最合适的

广播

广播是主机向子网内所有主机发送消息，子网内所有主机都能收到来自某台主机的广播信息，属于点对所有点的通信。广播意味着网络向子网每一个主机都投递一份数据包，不论这些主机是否乐意接收该数据包;



多播

多播是主机向一组主机发送信息，存在于某个组的所有主机都可以接收到消息，属于点对多点的通信。