03 IO的五种模型是什么

更新时间：2020-07-14 10:19:23

> 立志是事业的大门，工作是登堂入室的旅程。——巴斯德

前言

你好，我是彤哥。

上一节我们一起学习了 Netty 的发展历史，我们提到了 IO、NIO 这些名词。那么，到底什么是 IO 呢？什么又是 NIO 呢？

另外，我们平时又会听到两组很相似的概念：阻塞 / 非阻塞、同步 / 异步。那么，阻塞和非阻塞有什么区别呢？同步和异步又有什么区别呢？很多同学对这两组概念都比较容易混淆，也讲不清楚。

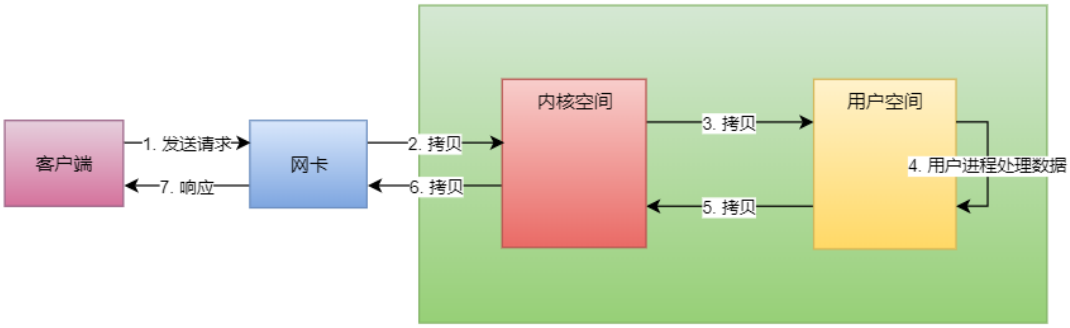
所以，本节就从网络 IO 的角度出发并用生活中常见的案例来白话 IO 的五种模型，以及上面两组概念。

用户空间和内核空间

操作系统的核心是内核，它独立于普通的应用程序，可以访问受保护的内核空间，也有访问底层硬件设备的所有权限。为了保护内核的安全，现在操作系统一般都强制用户进程不能直接操作内核，所以操作系统把内存空间划分成了两个部分：内核空间和用户空间。

这就好比，饭店老板把整个饭店划分成两个部分：大厅和厨房。大厅用于顾客吃饭，厨房用于厨师做饭，厨房的门上面一般还会写着：“厨房重地，闲人免进”，也就是顾客一般不具有直接使用厨房的特性。

所以，当我们使用 TCP 发送数据的时候，需要先将数据从用户空间拷贝到内核空间，再由内核操作将数据从内核空间发送出去；当我们使用 TCP 读取数据的时候，数据先在内核空间准备好，再从内核空间拷贝到用户空间供用户进程使用。



这就好比，当我们在饭店吃饭的时候，先在客厅点好菜，再由服务员把我们的菜单传递进厨房；当厨房做好了菜，再从厨房由服务员传递到客厅一样。

所以，一次 IO 的读取操作分为两个阶段（写入操作类似）：

- 等待内核空间数据准备阶段
- 数据从内核空间拷贝到用户空间

为此，Unix 根据这两个阶段又把 IO 分成了以下五种 IO 模型：

- 阻塞型 IO
- 非阻塞型 IO

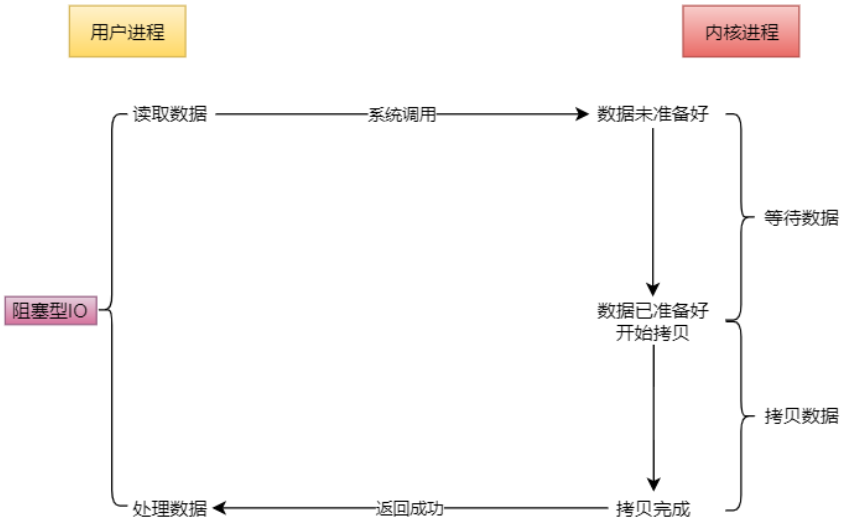
- IO 多路复用
- 信号驱动 IO
- 异步 IO

下面我们一一道来。

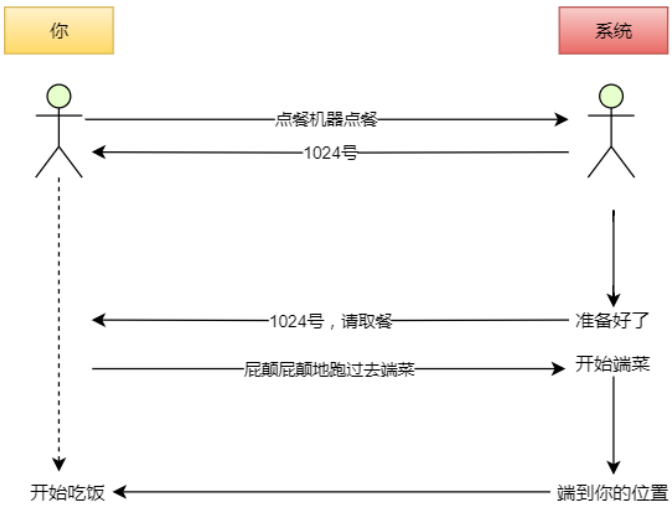
阻塞型 IO

阻塞型 IO，即当用户进程发起请求时，一直阻塞直到数据拷贝到用户空间为止才返回。

阻塞型 IO 在两个阶段是连续阻塞着的，直到数据返回。



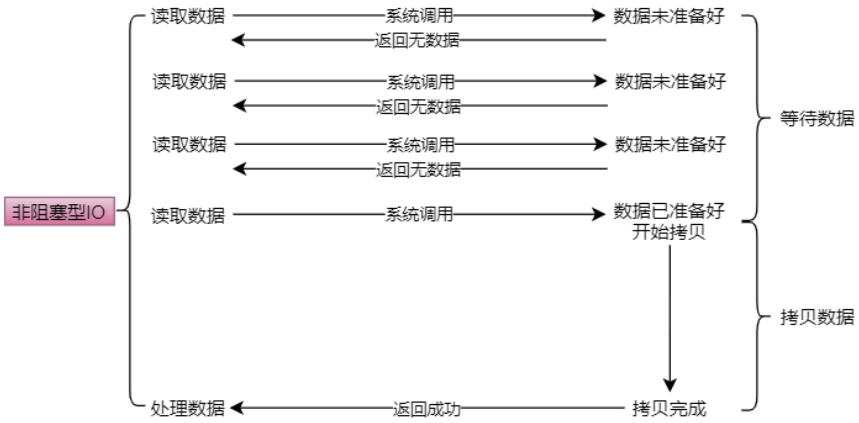
这就好比，你去路边买快餐，这家店比较低级，只有一辆车一个老板。点完餐后，你傻傻地看着老板开始打菜，然后拿给你。整个过程中，你只能看着老板打完菜并拿给你，这两个阶段你都是阻塞的。



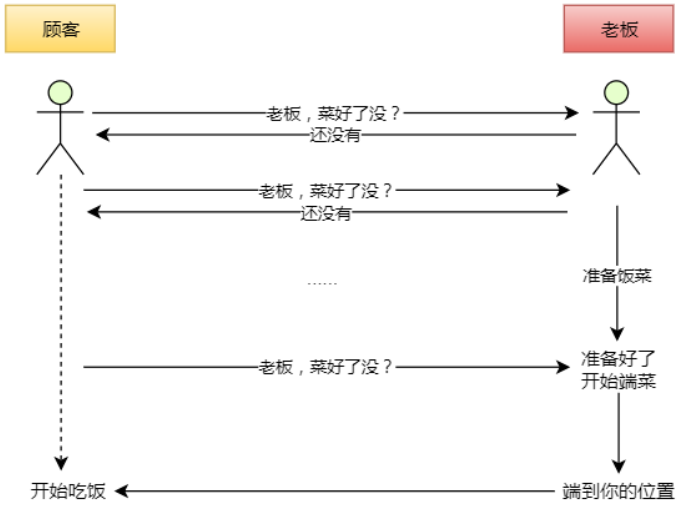
非阻塞型 IO

非阻塞型　IO，用户进程不断询问内核，数据准备好了吗？一直重试，直到内核说数据准备好了，然后把数据从内核空间拷贝到用户空间，返回成功，开始处理数据。

非阻塞型 IO 第一阶段不阻塞，第二阶段阻塞。



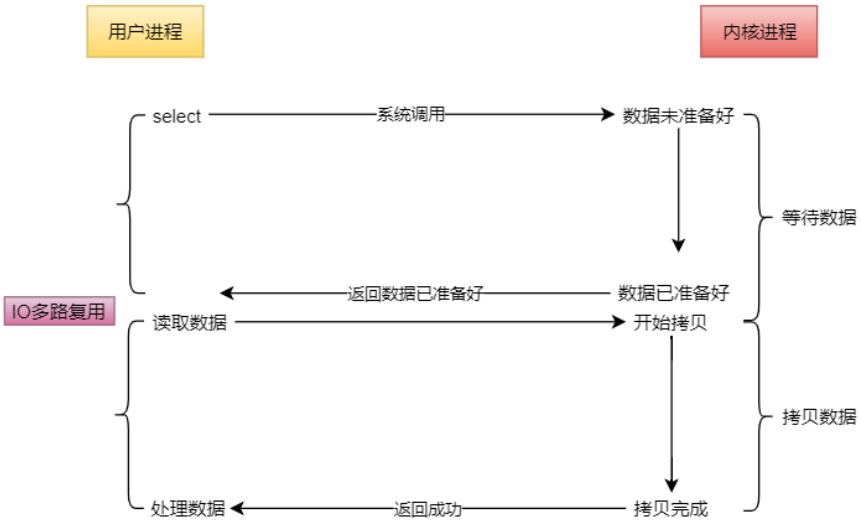
这就好比，你去小炒店，这家店高级一点，有独立的店面。点完餐后，你可以边玩手机边等。隔了一会你跑过去问一下老板“我的菜好了没”，老板说“还没好”；隔一会你又跑过去问了下“我的菜好了没”，老板说“还没有”；几次后，你又说“老板，我的菜好了没”，老板说“来了来了”，然后你看着他把菜端到你面前。整个过程中，询问“菜好了没” 你不用阻塞，老板立即回应你，你可以立即玩手机，但是端菜的时候你是傻傻地看着他端的，这期间你无法玩手机，你是阻塞的。



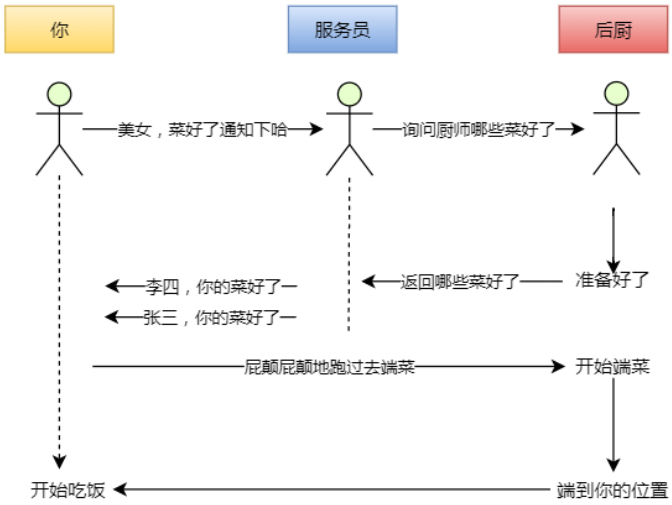
IO 多路复用

IO 多路复用，多个 IO 操作共同使用一个 selector（选择器）去询问哪些 IO 准备好了，selector 负责通知那些数据准备好了的 IO，它们再自己去请求内核数据。

IO 多路复用，第一阶段会阻塞在 selector 上，第二阶段拷贝数据也会阻塞。



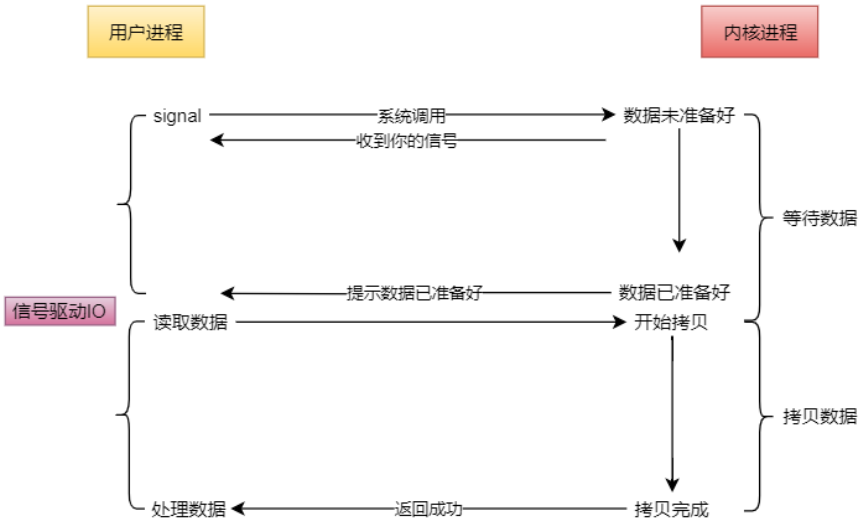
这就好比，你去川菜馆吃饭，这家饭店比较大，人也多，还有个美女服务员。你点完菜后，勾搭了一下美女服务员“美女，我点个辣子鸡丁，好了通知我一下哦”，美女也没搭理你。其它人也是这么勾搭美女的。然后，美女忙得不可开交，隔一会去厨房看一下，哪些菜好了，每次出来，都会喊 “那谁谁谁，你的啥啥菜好了，自己过来端一下。”。整个过程中，美女去厨房看菜是阻塞的，因为没有菜好的时候她还要等一会；你跑过去端菜也是阻塞的。一部分阻塞在美女身上，一部分阻塞在你身上。



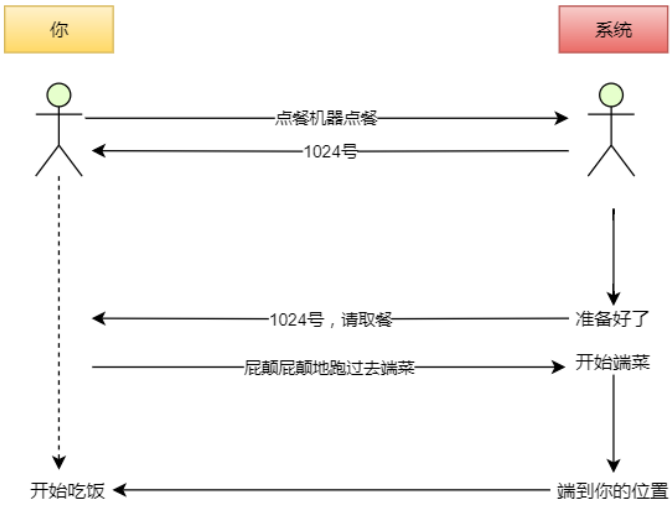
信号驱动 IO

信号驱动　　IO，用户进程发起读取请求之前先注册一个信号给内核说明自己需要什么数据，这个注册请求立即返回，等内核数据准备好了，主动通知用户进程，用户进程再去请求读取数据，此时，需要等待数据从内核空间拷贝到用户空间再返回。

信号驱动，第一阶段不阻塞，第二阶段阻塞。



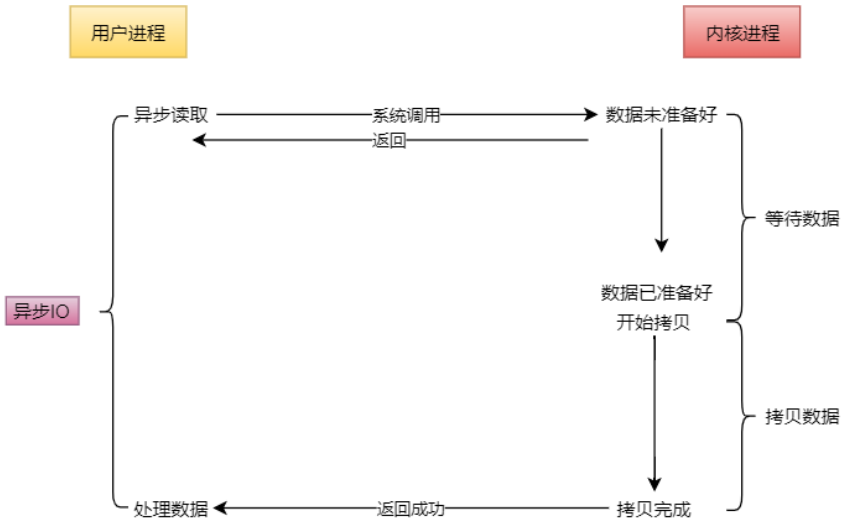
这就好比，你去“金拱门”吃麦当劳一样。你在旁边的机器上点完餐后出来一张小票“1024 号”，然后你边玩手机边等。过了一会，喇叭喊，“1024 号，请取餐。1024 号，请取餐。”，然后，你屁颠屁颠地跑过去取餐。整个过程中，点餐是立即返回的，之后想干啥干啥，不阻塞（也就是说你不用傻等着餐做好）；取餐的过程你需要从柜台端到你的位置上，是阻塞的。



异步 IO

异步 IO，用户进程发起读取请求后立马返回，当数据完全拷贝到用户空间后通知用户直接使用数据。

异步 IO，两个阶段都不阻塞。



这就好比，你去吃“渔粉”。扫码点餐后，你完全不用管，过了一会，一个大妈把饭菜端到你面前，还贴心地说了句“客官，请慢用”，然后你幸福地吃下了这碗 “金汤渔粉”。整个过程中，你既不用傻等着渔粉做好，也不用看着大妈把菜端到你面前或者你自己去端，完全不阻塞，纯异步。所以，这种体验是最好的。

所以，如果把吃饭的过程分成两个部分：“准备饭菜” 和 “端菜”，那么：

1. 如果你傻等着两个阶段完成，就是阻塞 IO；
2. 如果你隔一会询问一下 “菜做好了没”，期间你可以玩手机，但是端菜的时候你傻傻地看着老板端过来，就是非阻塞 IO；
3. 如果你和其他人都委托服务员帮你们隔一会看一下 “菜做好了没”，但是端菜需要自己去端，就是 IO 多路复用；
4. 如果是机器点餐，机器喊话取餐，就是信号驱动 IO；
5. 如果是扫码点餐，自动上餐，就是异步 IO；

阻塞与非阻塞

阻塞，是指调用结果返回之前，当前线程会被挂起，直到调用结果返回。比如，你傻等着端菜结束，你就是阻塞的。

非阻塞，是指不能立即得到结果之前，当前线程不被挂起，而是可以继续做其它的事。比如，你边玩手机边等饭菜准备好，你就是非阻塞的。

简单点，就是阻塞调用你必须挂起傻等着结果返回，非阻塞调用你不关心结果，调用之后你爱干嘛干嘛。

同步与异步

关于同步与异步，我们直接看看 POSIX 中的定义：

> A synchronous I/O operation causes the requesting process to be blocked until that I/O operation completes;
>
> An asynchronous I/O operation does not cause the requesting process to be blocked;

同步，调用者会被阻塞直到 IO 操作完成，调用的结果随着请求的结束而返回。

异步，调用者不会被阻塞，调用的结果不随着请求的结束而返回，而是通过通知或回调函数的形式返回。

> 阻塞 / 非阻塞，更关心的是当前线程是不是被挂起。
>
> 同步 / 异步，更关心的是调用结果是不是随着请求结束而返回。

这里的阻塞是指整个 IO 过程中是否有阻塞，更确切地说是 `recvfrom` 这个系统调用是否会阻塞，在我们的案例中，可以理解为 “端菜” 这个行为对于你来说是不是阻塞的。

所以，阻塞型 IO、非阻塞型、IO 多路复用、信号驱动 IO 都是同步 IO，只有最后一种才是异步 IO。

为什么不选择异步 IO?

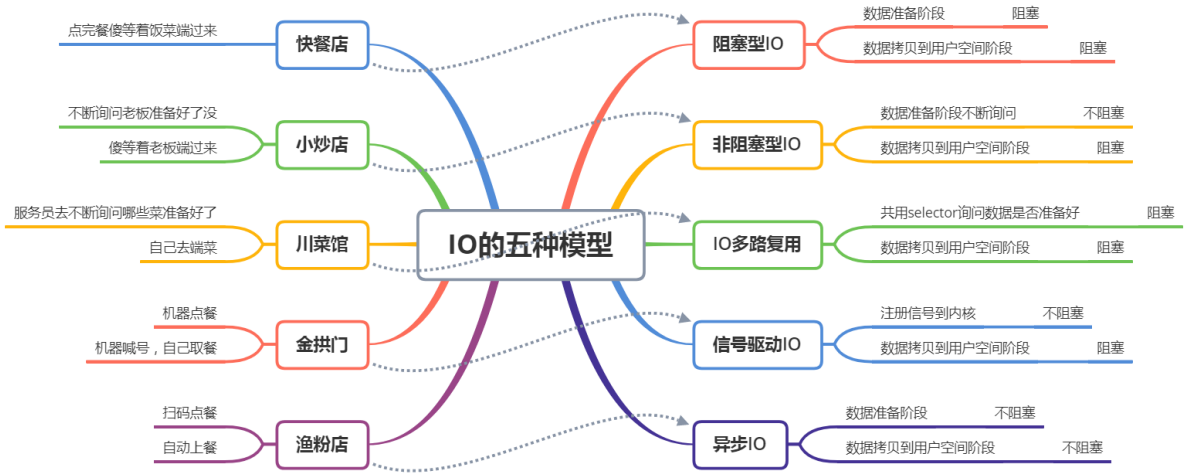
通过上面的分析，异步 IO 才是最牛的 IO 模型，那么，我们为什么不选择异步 IO 呢？

那是因为异步 IO 在 linux 上还不成熟，而我们的服务器通常都是 linux，所以现在大部分框架都不是很支持异步 IO，包括 Netty 之前实现了一版，但是后面给废弃掉了。

后记

本节通过饭店的不同模型来描述了 IO 的五种模型，相信读者们有了清晰的认识和深刻的理解，今天吃饭的时候，你处于哪种模型呢？

思维导图



}

02 Netty是如何发迹的

04 Java中如何使用BIO、NIO、AIO