IO 模型

作者：ZhangVincent
原文链接：https://ld246.com/article/1569811466358
来源网站：链滴

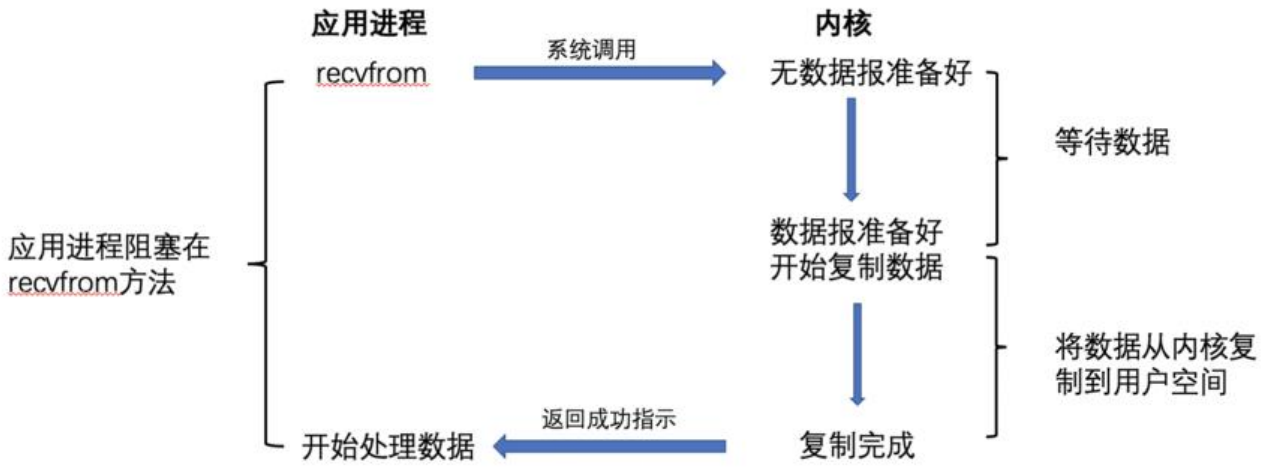
简介

当用户进程发出一次读数据请求时，数据并非直接从网络或者磁盘到达用户空间，而是经过了系统内的中转，即数据先由网络或者磁盘到达系统内核空间（数据等待阶段），然后再从系统内核空间复制用户空间（数据复制阶段）。这两阶段中，为了更好的协调CPU和外设之间的工作，逐渐发展出了各种O模型。

IO模型的分类

在《UNIX网络编程》中，将IO模型一共分为五种：阻塞式、非阻塞式、IO多路复用、信号驱动和异。接下来，我们将逐个介绍。

阻塞式IO

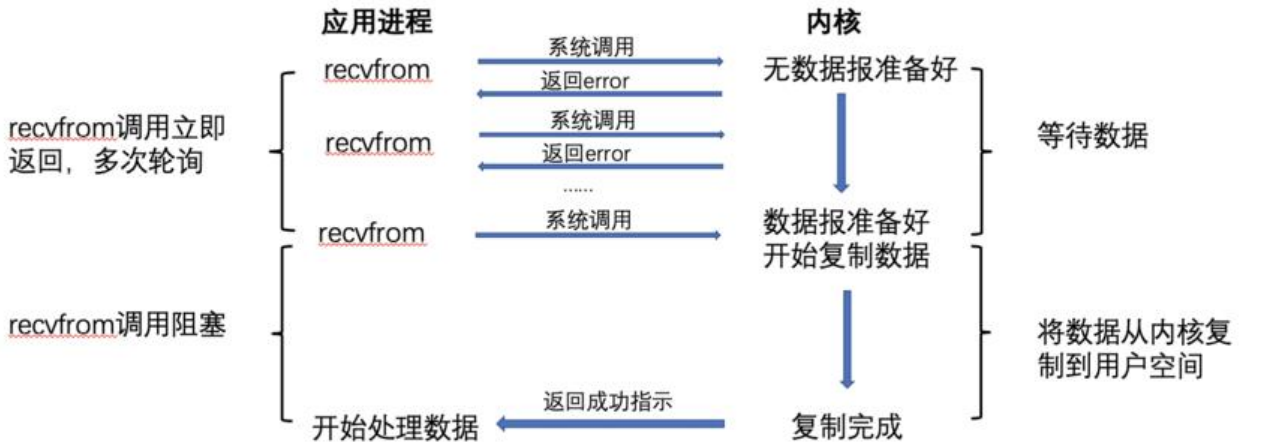


在阻塞式IO中，一个读操作的流程往往如图所示。我们可以看到，用户进程在进行recv系统调用后会阻塞，直到数据已经被复制到了用户空间。几乎所有的程序员第一次接触到的网络编程都是从listen()send()、recv() 等接口开始的，这些接口都是阻塞型的。

如果用单进程/单线程去处理所有请求的话，则线程/进程很有可能被阻塞在了对某个网络连接的IO操中，从而来不及处理另外的网络连接。因此后端业务系统往往在每个网络连接创建时给该连接创建一单独的线程。在小并发量的情况下，这种一个连接对应一个线程的模式，确实可以很好的处理网络IO作。然而，由于每个线程都会占用一部分内存，一旦系统并发量变大，一个连接对应一个线程的模式能会导致系统内存不足；另外，大并发量情况下，一个连接对应一个线程的模式会频繁的创建线程和繁的线程调度，都会严重的拖累系统性能，从而导致系统不可用。

总的来看，阻塞式IO只适用于小并发量的系统，不适合海量并发的场景。

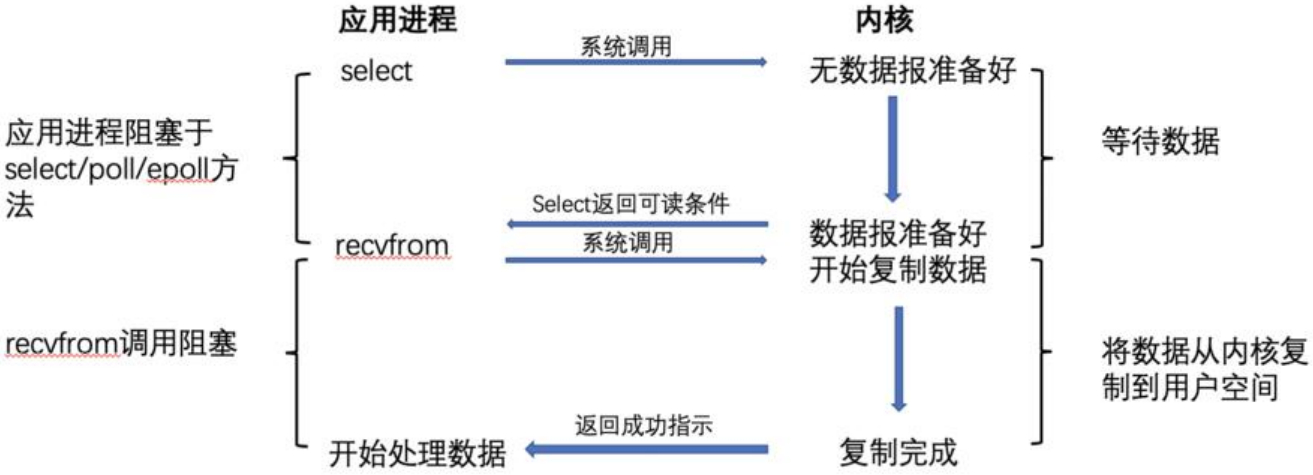
非阻塞式IO



同样以读操作为例，非阻塞式IO的流程如图所示。非阻塞式IO和阻塞式IO的不同，主要体现在数据等阶段：非阻塞式IO中，用户进程进行了读操作的系统调用后，如果内核中的数据没有准备好的话，系不会将用户进程阻塞，而是直接返回一个表示数据未准备好的状态码。从用户进程来看，进程发出一读请求后并不需要等待 ，而是立马得到一个结果。用户进程通过该结果即可知道数据并没有准备好，而后续会再次发起读请求。一旦系统内核中的数据准备好，用户进程再次发起读请求时，系统会将用进程阻塞，直到数据从内核复制到用户空间完毕。

我们不难看出，在非阻塞式IO模型情况下，为了读取到数据，用户进程需要主动地不停的发起读请求统调用，从而一直处于忙等待的状态。

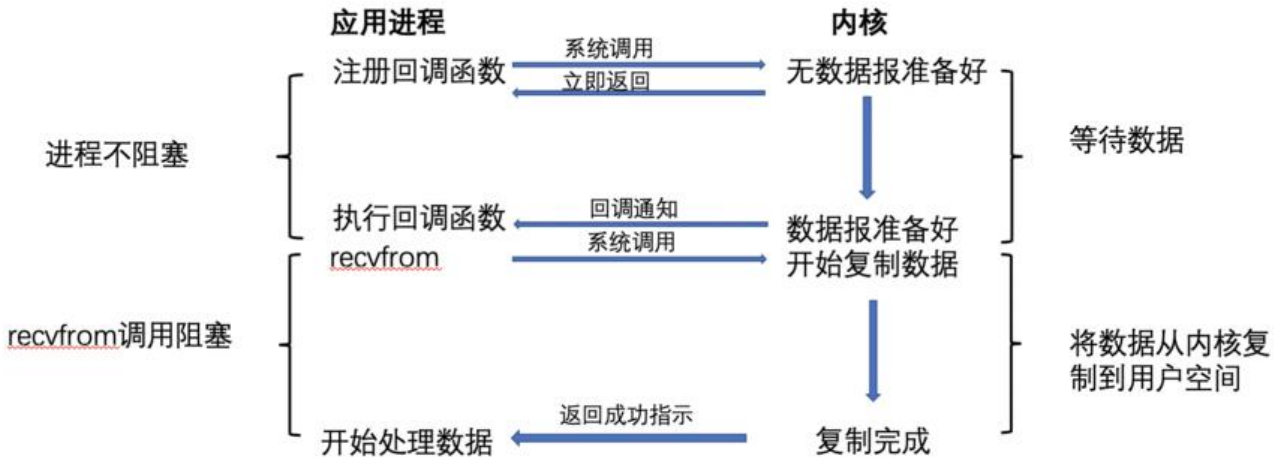
IO多路复用



从IO多路复用的流程中可知，与阻塞式IO类似，用户进程会阻塞在数据准备和数据复制两个环节。但阻塞式IO不同，在IO多路复用中，一个用户进程通过select、poll、epoll系统调用，可以监视多个网连接，一旦某个或某几个连接可读（数据已经到达系统内核）时，select、poll、epoll系统调用会立返回。用户可以通过select、poll、epoll系统调用的参数获取到可读、可写的连接的文件描述符。通连接的文件描述符，即从连接读取数据或者写入数据。

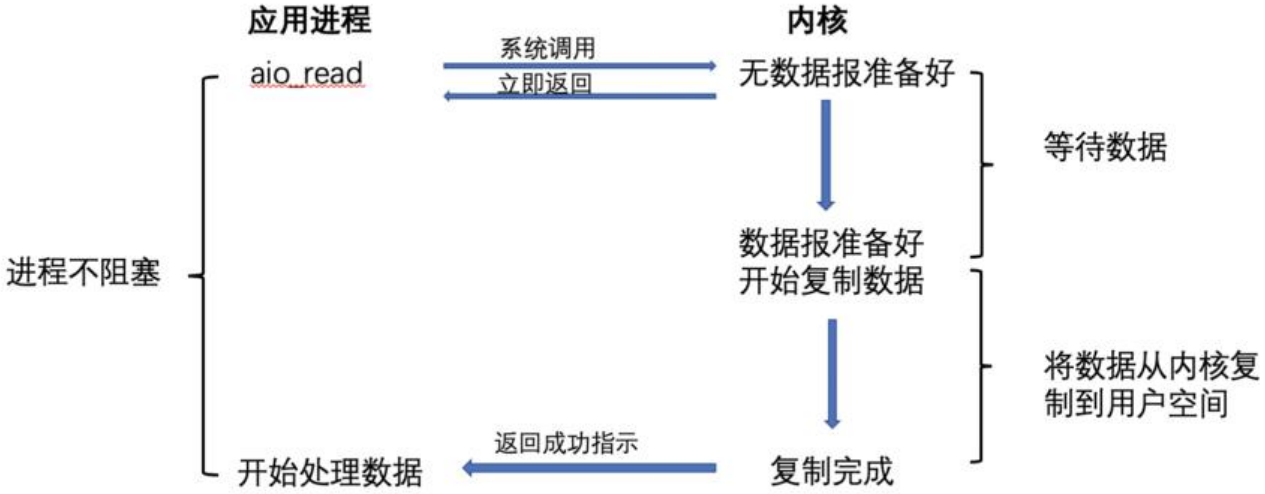
利用IO多路复用的特性，我们可以用java的NIO构建Reactor模式的应用程序。这部分内容可以参考《从Java NIO 到 Reactor模式》

信号驱动IO



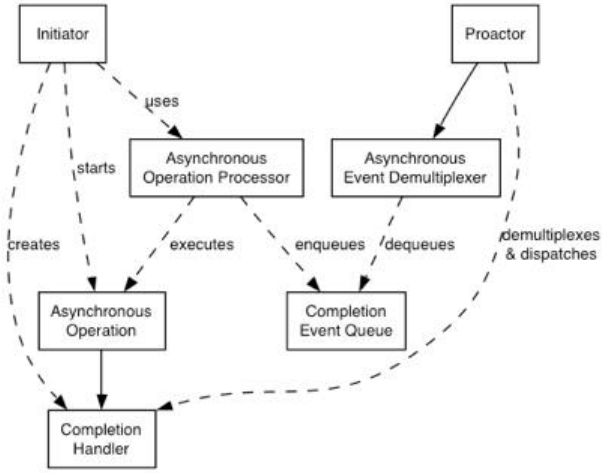
信号驱动式IO就是指进程预先向内核注册一个信号处理函数，然后用户进程返回不阻塞。当内核数据绪时会发送一个信号给进程，用户进程便在信号处理函数中调用IO读取数据。图中可以看出，数据从核拷贝到用户进程的过程中，用户进程还是会阻塞的。

异步IO



在异步IO中，用户进程发起read操作之后，立刻就可以开始去做其它的事。而从kernel的角度，当它到一个read请求后，会立刻返回，不会对用户进程产生任何block。同时，kernel会等待数据准备完，然后将数据拷贝到用户内存，当这一切都完成之后，kernel会给用户进程发送一个signal，告诉它read操作完成了。

下图是基于异步IO构建的Proactor模式的应用示意图。在Proactor中，有一个Initiator模块负责初始系统，一个异步操作处理器（往往由系统内核提供）负责执行异步操作（如读、写）。异步操作完成，异步操作处理器将完成事件或者失败事件放入完成事件队列。Proactor模块通过监视器监听完成事队列并获取到事件，从而调用Completion Handler进行后续的处理。



同步/异步、阻塞/非阻塞的区别

很多情况下，人们经常会把同步/异步IO的概念与阻塞/非阻塞IO的概念混同，即简单的将同步IO等同阻塞IO，将异步IO等同于非阻塞IO。其实本质上这两种概念还是有区别的。简单来讲，阻塞/非阻塞的是进程发起操作时，进程本身所处的状态；而同步/异步则指的是操作完成的通知机制。比如在读作中，如果是用户进程主动的去了解到数据已被读取，则为同步读；如果是由系统内核通过回调或者知的方式告知用户进程操作已完成，则属于异步读。由此可见，同步/异步、阻塞/非阻塞分别描述的两个不同维度的事情。

总结

本文主要介绍了阻塞式、非阻塞式、IO多路复用、信号驱动式和异步式等五种IO模型，并对同步/异、阻塞/非阻塞的概念做了一个区分。其中，IO多路复用是Reactor模式的基础，而Reactor模式被应到了当前很多开源高性能IO组件中（如Netty、Kafka、Nginx等）。了解IO模型，是构建高性能IO件、开发网络应用的一个充分前提。