异步IO

阻塞IO/非阻塞IO（区别同步/异步IO）

阻塞IO获取全部数据之后再返回，非阻塞IO不带数据立即返回，需要通过轮讯去判断是否完全完成数据的获取。
阻塞IO造成CPU等待浪费，非阻塞IO会让CPU处理状态判断，也是对CPU资源的浪费

轮询

Read：通过反复的调用来判断获取数据完成状态；
Select：通过事件描述符上的事件状态来判断，用一个1024长度的数组来存储判断状态，所有同时可以检查1024个文件描述符；
Poll：采用链表，避免了数组的长度限制，避免不需要的检查；
Epoll：进入轮询时，如果检测到没有IO事件则进入休眠状态，直到事件再次把它唤醒（事件通知、执行回调）

对于应用程序而言，它仍然属于一种同步，因为应用程序仍然需要等待IO完全返回，需要很多等待时间，例如轮询判断状态的时间和CPU休眠时间

理想的非阻塞异步IO

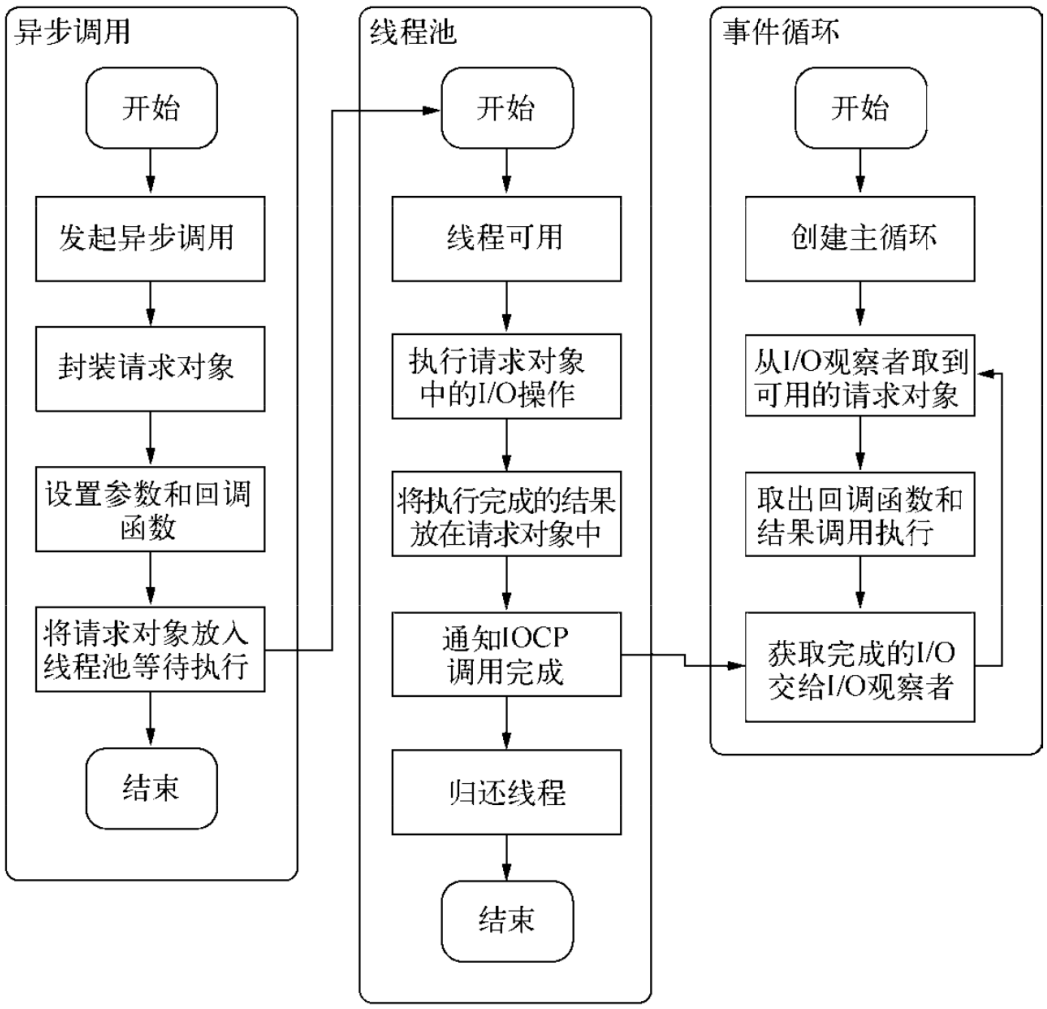
无需等待轮询结果，可以直接处理下一个任务

现实的非阻塞异步IO

部分线程进行阻塞IO或异步IO+轮询，一个线程负责计算，通过线程通信进行数据传递（采用线程池与阻塞IO模拟异步IO）

Node的异步IO

通过事件循环、观察者、封装请求对象、IO线程池实现（例子见57）
libuv做封装层，根据不太平台编译不同的代码到目标文件中



注意：Javascript是单线程执行，但是node本身是多线程，开发者写的代码无法并行，但是所有IO（磁盘IO、网络IO）是可以并行

几种服务器模型

1、同步式
2、每请求/每进程（服务器资源不够）
3、每请求/每线程（Apache）每个线程需要占用服务器内存，内存不够
4、事件驱动（Ngnix[仅限于做web服务器，在处理业务处理方面较为欠缺]、node）无需为每个请求创建一个额外的线程