浅谈并发编程的几个概念

同步vs异步

同步和异步关注的是消息通信机制，可以理解为一种行为方式。

同步

调用方发出一个请求时，在没有得到被调用方返回结果之前，这个请求不会得到响应，处于等待状态；一旦调用方得到返回结果，请求得到了响应。

简单概括为调用者主动等待被调用方返回结果，在没有返回之前一直“专职”等待。

异步

和同步相反，异步是调用方发出请求之后，不会立刻得到反馈结果，而是被调用方通过状态、通知来告诉调用者，或者通过回调函数来处理这个调用。

简言之：调用者发送请求之后，不会“专职等待”被调用方返回结果。而是当被调用方有了结果后主动通知调用方。

例子：打电话定酒店

阻塞vs非阻塞

阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态，描述的是一种状态。

阻塞

阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。

非阻塞

非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

> 阻塞与非阻塞与是否同步异步无关, 只是对一件事不同角度的理解。

串行vs并行

串行

指的是执行多个任务时，各个任务按照顺序执行，完成一个任务后才能进行下一个任务。

并行

多个任务可以同时执行。

例如：发试卷

并行vs并发

时间上：并行指的是多个事件在同一时刻发生；并发指的是多个事件在同一个时间间隔发生

作用点：并行是作用于多个实体上的多个事件，并发是作用于一个实体上的多个事件；

处理主体个数：并行是多个处理器处理多个任务，并发是一个处理器处理多个任务

I/O密集型 vs 计算密集型

I/O密集型 (I/O-bound)

这类程序一般CPU占用率较低。大部分的状况是 CPU 在等 I/O (硬盘/内存) 的读/写，此时 CPU Loading 不高。这类任务的特点是CPU消耗很少，任务的大部分时间都在等待IO操作完成（因为IO的速度远远低于CPU和内存的速度）。在相对情况下，对于IO密集型任务，任务越多，CPU效率越高。

IO密集型：读取文件，网络访问等

计算密集型 (CPU-bound)

CPU-bound的程序一般而言CPU占用率相当高，而硬盘/内存相对较低。计算密集型任务的特点是要进行大量的计算，消耗CPU资源，比如计算数学计算、对视频进行高清解码等等，全靠CPU的运算能力。

这种计算密集型任务虽然也可以用多任务完成，但是任务越多，花在任务切换的时间就越多，CPU执行任务的效率就越低，所以，要最高效地利用CPU，可以适当使用多进程（核数）。

> 常见的大部分任务都是IO密集型任务，比如Web应用。
>
> 考虑到CPU和IO之间巨大的速度差异以及机器的性能差异，一个任务在执行的过程中大部分时间都在等待IO操作，单进程单线程模型会导致别的任务无法并行执行，因此，我们需要多进程、多线程等模型来支持多任务并发执行。

思考： 如何充分利用机器性能（IO cpu）

并发的三个层次

低阶

这个级别的并发主要是基于操作系统底层，直接使用原子操作来实现的并发。主要针对库编写者而不是针对应用程序开发者的，因为这很容易出错，并且调试起来非常困难。尽管Python并发的实现通常是使用低级操作构建的，但是Python不支持这种并发。

中阶

这个级别的并发几乎所有的语言都支持。 这一层次的并发并不使用显式的原子操作，但是会使用显式的锁。Python提供了对这个层次编程的支持，例如线程，进程，信号量，线程锁定和多处理。锁定等类。通常使用这种级别的并发支持。

高阶

一些现代语言开始支持高级并发。这一层次的并发没有显式的原子操作和显式的锁（锁定和原子操作很可能发生在底层，但我们不必关心它们），python语言中，对中阶实现的过程进行了封装，出现的一些高阶并发框架，如concurrent.futures, asyinco协程, gevent等。