# 认识多线程

并发编程的意义：并发编程可以使程序执行速度得到极大的提高，或者为设计某些类型的程序提供更简单的模型。

并发编程的几个方面内容：

并发编程令人困惑的一个重要原因是：使用并发时需要解决的问题有多个，而实现并发的方式也有多种，并且这两者之间没有明显的映射关系。

解决办法：我们必须理解所有的这些问题和特例，以便有效的使用并发。

具体办法

* 更快的执行

如果你想让一个程序运行的更快，那么可以将其断开为多个片段，在单独的处理器上运行每个片段。并发是用于多处理器编程的基本工具。当前速度的提高是以多核处理器的形式而不是更快的芯片的形式出现的。为了使程序运行的更快，你必须学习如何利用这些额外的处理器，而这正是并发赋予你的能力。

但是，并发通常是提高运行在单处理器上的程序的性能。

* 改进代码设计

Java的线程机制是抢占式的，这表示调度机制会周期性地中断线程，将上下文切换到另一个线程，从而为每个线程都提供时间片段，使得每个线程都分配到数量合理地时间去驱动它的任务。在协作式系统中，每个任务都会自动的放弃控制，这要求程序员有意识的在每个任务中插入让步语句。协作系统的优势是双重的：上下文切换的开销比抢占式要低廉的多，可以同时执行的线程数量理论上没有限制。当你处理大量的仿真元素时，这是一种理想的解决方案。但是注意，某些协作式系统并未设计为可以在多个处理器之间分配任务，这可能会非常有限。

并发需要付出代价，但这些代价与在程序设计、资源负载均衡以及用户方便方面的改进相比，就显得微不足道。通常，线程能够使我们创建更加松耦合的设计。

## 线程的基本机制

并发编程使得我们可以将程序划分为多个分离的、独立的任务。通过使用多线程机制，这些独立任务中的每一个都将由执行程序来驱动。一个线程就是进程中的一个单一的顺序控制流，因此，单个进程可以拥有多个并发执行的任务，但是你的程序使得每个任务都好像有其自己的CPU一样。其底层机制是切分CPU时间，但我们通常不需要考虑他。

线程模型为编程带来了便利。它简化了在单一程序中同时多个操作的处理。在使用线程时，CPU将轮流给每个任务分配其占用时间。每个任务都觉得自己在一直占用CPU，但事实上CPU时间是划分片段分配给了所有任务（也有可能是运行在多个CPU之上）。线程的一大好处是可以使你从这个层次抽身出来，即代码不需要知道它是运行在一个还是多个CPU上。所以，使用线程机制是一个建立透明的、可扩展的程序的方法，如果程序运行速度太慢，为机器增添一个CPU就很容易的增加程序运行的速度。多个任务、多个线程是使用多个处理器系统的最合理方式。

* 定义任务
* Thread类
* 使用Executor
* 从任务中产生返回值
* 休眠
* 优先级
* 让步
* 后台线程
* 编码的变体
* 加入一个线程
* 捕获异常