协程

jask

09/10/2024

什么是协程

最简单的理解，可以将协程当成一种看起来花里胡哨，并且使用起来也花里胡哨的函数。每个协程在创建时都会指定一个入口函数，这点可以类比线程。协程的本质就是函数和函数运行状态的组合。协程和函数的不同之处是，函数一旦被调用，只能从头开始执行，直到函数执行结束退出，而协程则可以执行到一半就退出（称为 yield），但此时协程并未真正结束，只是暂时让出 CPU 执行权，在后面适当的时机协程可以重新恢复运行（称为 resume），在这段时间里其他的协程可以获得 CPU 并运行，所以协程被描述称为轻量级线程。

协程能够半路 yield、再重新 resume 的关键是协程存储了函数在 yield 时间点的执行状态，这个状态称为协程上下文。协程上下文包含了函数在当前执行状态下的全部 CPU 寄存器的值，这些寄存器值记录了函数栈帧、代码的执行位置等信息，如果将这些寄存器的值重新设置给 CPU，就相当于重新恢复了函数的运行。单线程环境下，协程的 yield 和 resume 一定是同步进行的，一个协程的 yield，必然对应另一个协程的 resume，因为线程不可能没有执行主体。并且，协程的 yield 和 resume 是完全由应用程序来控制的。与线程不同，线程创建之后，线程的运行和调度也是由操作系统自动完成的，但协程创建后，协程的运行和调度都要由应用程序来完成，就和调用函数一样，所以协程也被称为“用户态线程”。

对称协程与非对称协程

对称协程，协程可以不受限制地将控制权交给任何其他协程。任何一个协程都是相互独立且平等的，调度权可以在任意协程之间转移。

非对称协程，是指协程之间存在类似堆栈的调用方被调用方关系。协程出让调度权的目标只能是它的调用者。

有栈协程与无栈协程

有栈协程：用独立的执行栈来保存协程的上下文信息。当协程被挂起时，栈协程会保存当前执行状态（例如函数调用栈、局部变量等），并将控制权交还给调度器。当协程被恢复时，栈协程会将之前保存的执行状态恢复，从上次挂起的地方继续执行。类似于内核态线程的实现，不同协程间切换还是要切换对应的栈上下文，只是不用陷入内核而已。

无栈协程：它不需要独立的执行栈来保存协程的上下文信息，协程的上下文都放到公共内存中，当协程被挂起时，无栈协程会将协程的状态保存在堆上的数据结构中，并将控制权交还给调度器。当协程被恢复时，无栈协程会前保存的状态从堆中取出，并从上次挂起的地方继续执行。协程切换时，使用状态机来切换，就不用切换对应的上，下文了，因为都在堆里的。比有栈协程都要轻量许多。

独立栈与共享栈

独立栈和共享栈都是有栈协程。

共享栈本质就是所有的协程在运行的时候都使用同一个栈空间，每次协程切换时要把自身用的共享栈空间拷贝。减少内存的浪费。

独立栈，也就是每个协程的栈空间都是独立的，固定大小。协程切换时不需要大量拷贝，但是内存空间浪费。

协程的优缺点

经过对协程概念的介绍，想必都对协程的优点有了些许认识，这个问题见仁见智，这里根据我的理解总结两点：提高资源利用率，提高程序并发性能。协程允许开发者编写异步代码，实现非阻塞的并发操作，通过在适当的时候挂起和恢复协程，可以有效地管理多个任务的执行，提高程序的并发性能。与线程相比，协程是轻量级的，它们的创建和上下文切换开销较小，可以同时执行大量的协程，而不会导致系统负载过重，可以在单线程下实现异步，使程序不存在阻塞阶段，充分利用 cpu 资源。简化异步编程逻辑。使用协程可以简化并发编程的复杂性，通过使用适当的协程库或语言特性，可以避免显式的线程同步、锁和互斥量等并发编程的常见问题，用同步的思想就可以编写成异步的程序