并发和并行

并发Concurrency：一段时间内发生了很多事情。

并行parallelism：多个事件同时发生，强调同一时刻

想象一下，国庆节前一天晚上等在高速口准备0点上高速的车，是并发还是并行？

进程和线程

由来

早期工作方式

最早计算机很巨大笨重，得把程序转换成某种编码对应的纸带或卡片上的孔（类似2B铅笔涂答题卡），输入设备再读取打孔纸带把程序和数据装入计算机，程序在计算机上运行完，在通过打印机输出。整个过程一个程序从输入到输出在所有环节都是独占资源，实际上输入、输出环节非常慢，而且此时CPU没有计算任务，处于忙等待。再一个，如果下一个程序需要运行，整个过程将再走一遍，这是一个接一个运行的**串行**方式。

计算机处理IO时，会让CPU处于忙等待，能否设计一套控制程序运行的技术，来充分利用CPU资源？

多道处理程序

计算机内存中常驻一个监管程序，把多个程序一并加载到计算机内存中。

例如有A、B两个程序，它们在内存中有各自独立的内存空间。

先运行A，A遇到IO时，让出CPU使用权，监管程序调用B运行，B遇到IO时，让出控制权，监管程序让A运行。

需要注意的是，A、B实际上是**交替**运行，而不是串行执行。

这种控制方式减少了CPU空闲时间，提高了CPU运行效率。

分时系统

随着电子管、晶体管、大规模集成电路技术的突飞猛进，计算机硬件技术进步日新月异，CPU的运算单元和运行频率也越来越高。

可以将CPU运行时间分成非常小的时间片，把时间片分给不同的作业使用。如果某个作业不能完成计算，则暂时中断执行，让出CPU时间给另一个作业执行，等待下一轮轮询到自己。由于计算机运行速度已经很快了，且时间片较短，产生了所有作业在并行的错觉。每个作业也似乎都独占着计算机资源。

分时的好处，使得每个作业在很短的时间内都有执行的机会，就可以和用户终端有很好的及时的交互。

以上所讲，都影响着后来出现的操作系统的并发工作方式。现在CPU已进入多核时代，促使了协程被更多编程语言支持。

协程，不过就是把当年的设计思想真正实现罢了。

```
Go语言推荐使用协程来解决并发。但是其底层利用了多线程、IO多路复用，协程又是要解决多线程的一些弊端，如果不能很好的理解多线程运行模型、IO多路复用模型，就很难理解Goroutine的精髓。
所以，我们的第一步从多线程及阻塞IO模型说起。
```

由于Go语言没有提供线程、进程等简单易用的库，为了让大家能和Goroutine进行对比，这里使用Python语言来讲解、体会多线程的并发。请大家不要把重点放在学习Python语法上，而是去体会并发。

进程和线程概念

我们现在接触到的多是多任务、分时的现代操作系统，其内部包含进程管理模块。

作业就是运行的任务，这些任务被称为进程process。每个进程要占据一块内存空间存放指令、数据等。

进程工作时，如果被阻塞，那么当前进程就只能什么都干不了，怎样才能提高单个进程工作效率，又可以使用进程内存空间呢？线程thread。

一个进程内可以创建很多线程，让不同线程干活，即使是阻塞了一个线程，还有其他线程可以干活。这样提高了进程运行效率，同时进程就变成了一个资源和线程的容器。

在实现了线程的操作系统中，线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一个程序的执行实例就是一个进程。

进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

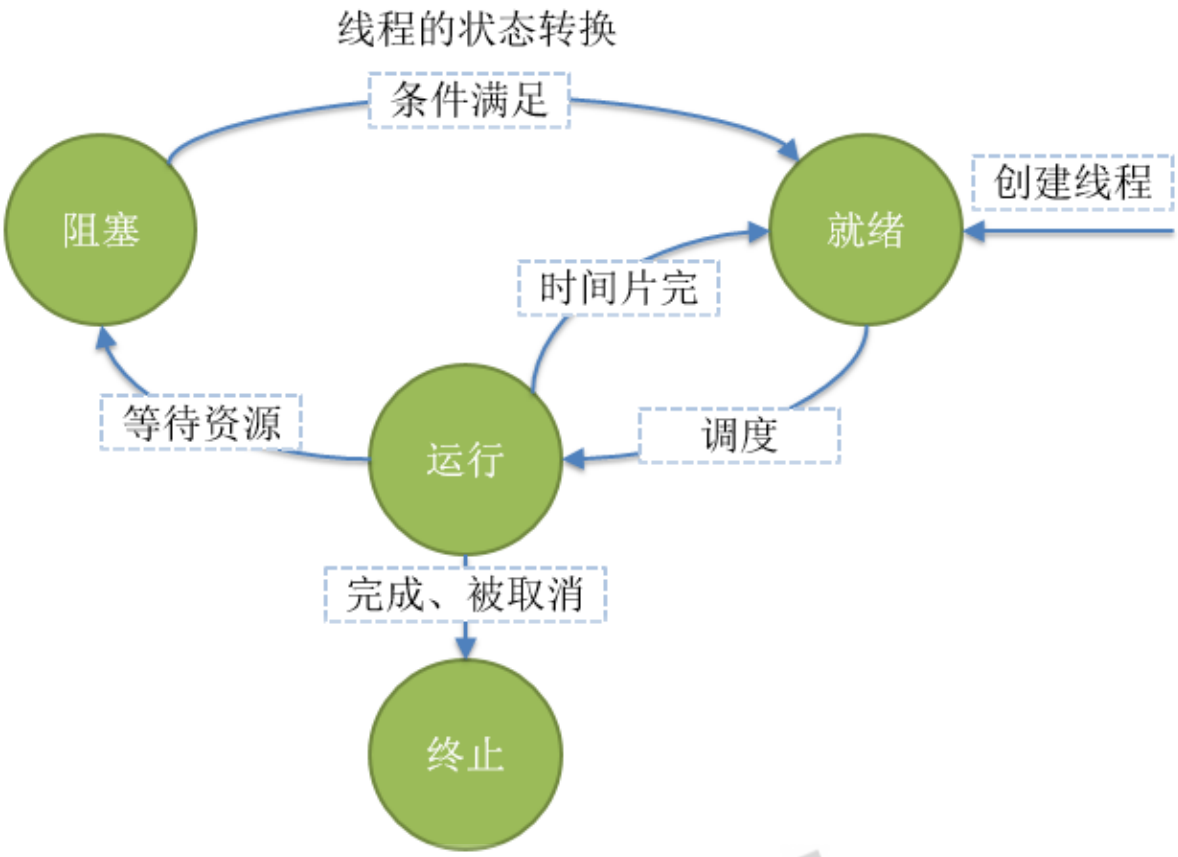
进程和程序的关系：程序是源代码编译后的文件，而这些文件存放在硬盘上。当程序被操作系统加载到内存中，就是进程，进程中存放着指令和数据（资源），它也是线程的容器。

形象化理解

- 现代操作系统提出进程的概念，每一个进程都认为自己独占所有的计算机硬件资源
- 进程就是独立的王国，进程间不可以随便的共享数据
- 线程就是省份，同一个进程内的线程可以共享进程的资源，每一个线程拥有自己独立的堆栈

线程的状态

状态	含义
就绪(Ready)	线程能够运行，但在等待被调度。可能线程刚刚创建启动，或刚刚从阻塞中恢复，或者被其他线程抢占
运行(Running)	线程正在运行
阻塞(Blocked)	线程等待外部事件发生而无法运行，如I/O操作
终止(Terminated)	线程完成，或退出，或被取消



Python中的进程和线程

运行程序会启动一个解释器进程，线程共享一个解释器进程。

Python的线程开发

Python的线程开发使用标准库threading。

进程靠线程执行代码，进程中至少有一个**主线程**，其它线程称为工作线程。 主线程是第一个启动的线程。

父线程：如果线程A中启动了一个线程B，A称为B的父线程。 子线程：B称为A的子线程。

Thread类

```
# 签名
def __init__(self, group=None, target=None, name=None,
                 args=(), kwargs=None, *, daemon=None)
```

参数名	含义
target	线程调用的对象，就是目标函数
name	为线程起个名字
args	为目标函数传递实参，元组
kwargs	为目标函数关键字传参，字典

线程启动

```python
import threading

# 最简单的线程程序
def worker():
    print("I'm working")
    print('Fineshed')

t = threading.Thread(target=worker, name='worker') # 线程对象
# target=worker方式为关键字传参，按名称对应
# Python中还有按照位置对应传参，按照顺序依次对应
t.start() #  启动
```

通过threading.Thread创建一个线程对象，target是目标函数，可以使用name为线程指定名称。 但是线程没有启动，需要调用start方法。

线程之所以执行函数，是因为线程中就是要执行代码的，而最简单的代码封装就是函数，所以还是函数调用。 函数执行完，线程也就退出了。 那么，如果不让线程退出，或者让线程一直工作怎么办呢?

```python
import threading
import time

def worker():
    while True: # for i in range(5):
        time.sleep(1)
        print("I'm working")
    print('Fineshed')

t = threading.Thread(target=worker, name='worker') # 线程对象
t.start() #  启动
```

线程退出

Python没有提供线程退出的方法，线程在下面情况时退出 1、线程函数内语句执行完毕 2、线程函数中抛出未处理的异常

线程的传参

```python
import threading
import time

def worker(x, y):
    s = "{} + {} = {}".format(x, y, x + y)
    print(s, threading.currentThread().ident)

t1 = threading.Thread(target=worker, name='worker', args=(4, 5))
t1.start()
time.sleep(2)
```

线程传参和函数传参没什么区别，本质上就是函数传参。

多线程编程

顾名思义，多个线程，一个进程中如果有多个线程运行，就是多线程，实现一种并发。

想想下面有几个线程运行?

```
import string
import threading
import time

def count():
    c = 1
    while True:
        time.sleep(1)
        print("count = ", c)
        c += 1

def char():
    s = string.ascii_lowercase
    for c in s:
        time.sleep(2)
        print("char = ", c)

t1 = threading.Thread(target=count, name="count")
t2 = threading.Thread(target=char, name="char")
t1.start()
t2.start()
```

3个，count、char、主线程。

注：Python中有一个GIL全局解释器锁，大家初学可以忽略它，它对阻塞性IO其实影响不大。

重点：大家要从例子中找到并发执行的感觉，这对理解包括Goroutine都大有益处。