多任务

- 电脑同时开了很多窗口,可以同时qq， 微信， 听音乐， 等等，对应的就是多任务
- 操作系统都是多任务

- 单核cpu实现多任务的原理：操作系统轮流让各个任务交替执行， 切换速度很快，看起来是同时执行
- 多核cpu实现多任务的原理：真正的多任务，但是由于任务数量远远多于cpu核数，所以操作系统也会自动的把很多任务轮流的调度到每一个cpu上执行

并发和并行

- 并发： 看上去是一起执行， 任务数多于cpu核心数， 一个人吃三个馒头
- 并行： 真正的并发， 任务数小于等于cpu核心数， 三个人吃三个馒头

实现多任务

- 多进程
- 多线程
- 多进程 + 多线程

> 核心目的都是为了提高程序的执行效率

进程

定义

- 是操作系统进行资源分配和调度的基本单位
- 进程是线程的容器

进程间数据不共享

> 操作系统会为每一个进程分配属于自己的内存空间

- 进程是互相独立的，各自运行在自己独立的内存空间上
- 进程之间不共享任何变量

线程

定义

- 是操作系统调度运算的最小单位 ,被包含在进程中,是进程实际的运作单位 一个进程中可以同时并发多个线程, 每条线程执行不同的任务
- 一个进程要执行起来必须有一个线程
- 进程是线程的集合,所以同一个进程里的线程共享进程的内存空间

单进程 (单任务)

```
def work():
    print('我开始了')
    sleep(3)
    print('我结束了')

if __name__ == '__main__':
    for i in range(3):
        work()
```

创建多进程

```
from multiprocessing import Process
import multiprocessing
```

函数式创建并区分主进程和子进程

```
from multiprocessing import Process
from time import sleep, time

def process_func():
    print(os.getpid())
    print('我是子进程')
    sleep(3)
    print ('子进程结束')


print(os.getpid()
st_time = time()
p = Process(target=process_func)
```

```
p1 = Process(target=process_func)
p2 = Process(target=process_func)
p.start()
p1.start()
p2.start()
p.join()
p1.join()
p2.join()
```

其他:

```
is_alive()   判断子进程是否存活
os.getpid()  获取进程号
```

循环创建并验证活性

```
def  work():
    print('子任务开始')
    sleep(3)
    print('子任务结束')

if __name__ == '__main__':
    st_time = time()
    process_list = []

    for i in range(3):
        p = Process(target=work)
        p.start()
        process_list.append(p)

    # 判断子进程是否存活
    print ([i.is_alive() for i in process_list])
    # 需要等待所有子任务都工作起来之后再在环外面阻塞，不然就会变成单任务
    [i.join() for i in process_list]
    # 再次判断是否存活
    print([i.is_alive() for i in process_list])
    end_time = time() - st_time
    print('共耗时:{}'.format(end_time))
    print('主进程结束')
```

创建带有参数的子进程

```
def work(num):
    print('子任务开始:{} 进程号是:{} 编号是:{}'.format(num,os.getpid())
    sleep(3)
    print('子任务结束:{} 进程号是:{}'.format(num,os.getpid()))

if __name__ == '__main__':
    st_time = time()
    process_list = []
```

```
    print('主进程的进程号是:{}'.format(os.getpid()))

    for i in range(3):
        p = Process(target=work, args=(i,))
        # 进程创建完之后不会马上执行,需要使用start启动
        p.start()
        # 在函数里面join会变成单任务, 需要让子任务都先运行起来, 之后再去阻塞
        # p.join()
        process_list.append(p)
    # 判断子进程是否存活
    print ([i.is_alive() for i in process_list])
    # 需要等待所有子任务都工作起来之后在循环外面阻塞, 不然就会变成单任务
    [i.join() for i in process_list]
    # 再次判断子进程是否存活
    print([i.is_alive() for i in process_list])
    end_time = time() - st_time
    print('共耗时:{}'.format(end_time))
```

继承式创建

继承Process类, 并改写父类的run方法, 使用start()启动子进程

```
from multiprocessing import Process
import os

class process_child(Process):
    def run(self):
        print('子任务开始')
        sleep(3)
        print('子任务结束')


if __name__ == '__main__':
    st_time = time()
    p = process_child()
    p1 = process_child()
    p2 = process_child()
    p.start()
    p1.start()
    p2.start()
    p.join()
    p1.join()
    p2.join()
    cost_time = time() - st_time
    print('共耗时:{}'.format(cost_time))
```