协程

协程是我要重点去讲解的一个知识点. 它能够更加高效的利用CPU.

其实, 我们能够高效的利用多线程来完成爬虫其实已经很6了. 但是, 从某种角度讲, 线程的执行效率真的就无敌了么? 我们真的充分的利用CPU资源了么? 非也~ 比如, 我们来看下面这个例子.

我们单独的用一个线程来完成某一个操作. 看看它的效率是否真的能把CPU完全利用起来.

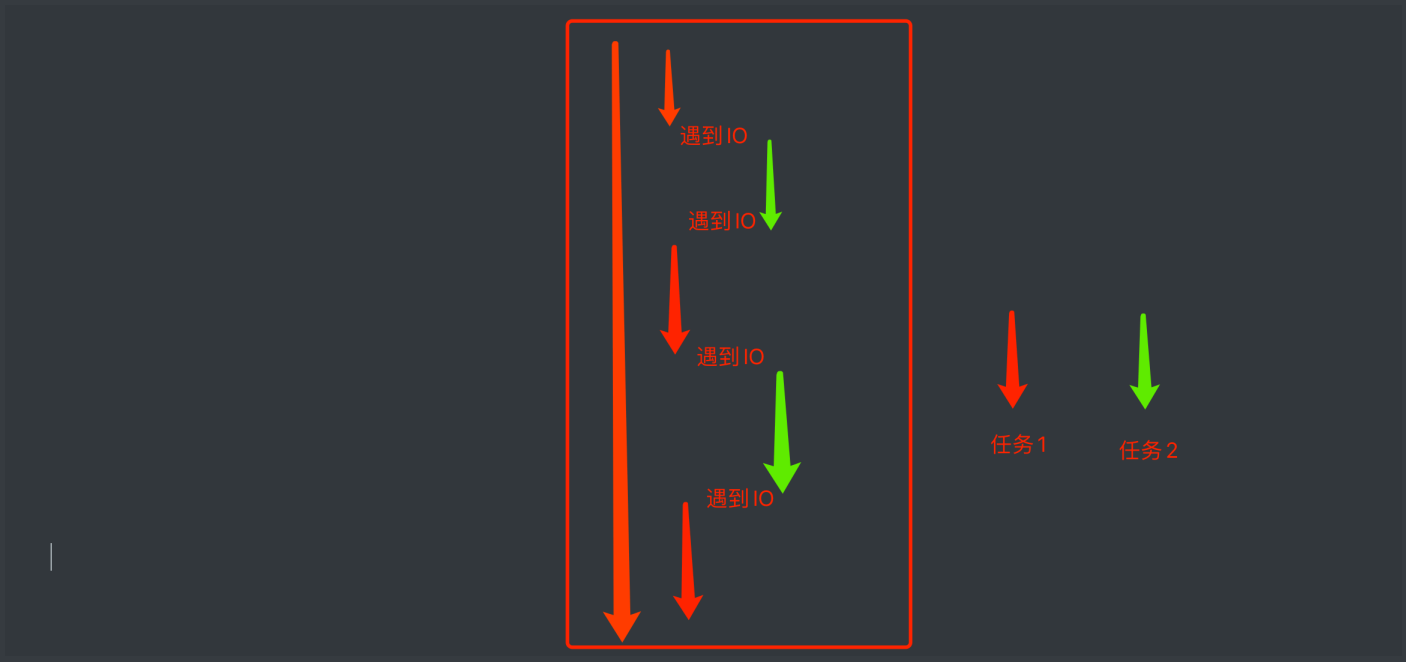
```
import time

def func():
    print("我爱黎明")
    time.sleep(3)
    print("我真的爱黎明")

func()
```

各位请看. 在该程序中, 我们的func()实际在执行的时候至少需要3秒的时间来完成操作. 中间的三秒钟需要让我当前的线程处于阻塞状态. 阻塞状态的线程 CPU是不会来执行你的. 那么此时cpu很可能会

切换到其他程序上去执行. 此时, 对于你来说, CPU其实并没有为你工作(在这三秒内), 那么我们能不能通过某种手段, 让CPU一直为我而工作. 尽量的不要去管其他人.

我们要知道CPU一般抛开执行周期不谈, 如果一个线程遇到了IO操作, CPU就会自动的切换到其他线程进行执行. 那么, 如果我想办法让我的线程遇到了IO操作就挂起, 留下的都是运算操作. 那CPU是不是就会长时间的来照顾我~.

以此为目的, 伟大的程序员就发明了一个新的执行过程. 当线程中遇到了IO操作的时候, 将线程中的任务进行切换, 切换成非 IO操作. 等原来的IO执行完了. 再恢复回原来的任务中.



就形成了这样一种模型, 在程序遇到了IO操作(费时不费力的操作)时, 自动切换到其他任务. 该模型被称为协程.

协程的基本写法: 咱就介绍一种, 也是最好用的一种, 如果各位想看更加详细, 细致的协程推导过程, 可以再等等~~ 未来鄙人会推出更详细的多任务系列教程~.

先上手来一下.

```
async def func():
    print("我是协程")


if __name__ == '__main__':
    # print(func())  # 注意, 此时拿到的是一个协程对象, 和生成器差不多.该函数默认是不会这样执行的

    coroutine = func()
    asyncio.run(coroutine)  # 用asyncio的run来执行协程.
    # lop = asyncio.get_event_loop()
    # lop.run_until_complete(coroutine)   # 这两句顶上面一句
```

效果不明显, 继续加码

```
import time

# await: 当该任务被挂起后,CPU会自动切换到其他任务中
```

```python
async def func1():
    print("func1, start")
    await asyncio.sleep(3)
    print("func1, end")


async def func2():
    print("func2, start")
    await asyncio.sleep(4)
    print("func2, end")


async def func3():
    print("func3, start")
    await asyncio.sleep(2)
    print("func3, end")


if __name__ == '__main__':
    start = time.time()
    tasks = [  # 协程任务列表
        func1(),  # 创建协程任务
        func2(),
        func3()
    ]
    lop = asyncio.get_event_loop()
```

```
30      # 我要执行这个协程任务列表中的所有任务
31      lop.run_until_complete(asyncio.wait(tasks))
    # 我要执行这个协程任务列表中的所有任务
32      print(time.time() - start)
```

妙不妙~~

上面的程序还可以写成这样

```
 1 async def main():
 2     print("start")
 3     # # 添加协程任务
 4     # t1 = asyncio.create_task(func1())
 5     # t2 = asyncio.create_task(func2())
 6     # t3 = asyncio.create_task(func3())
 7     #
 8     # ret1 = await t1
 9     # ret2 = await t2
10     # ret3 = await t3
11
12     tasks = [
13         func1(),
14         func2(),
15         func3()
16     ]
17     # 一次性把所有任务都执行
```

```
    done, pedding = await asyncio.wait(tasks)

    print("end")

if __name__ == '__main__':
    start = time.time()
    asyncio.run(main())
    print(time.time() - start)
```

模拟一下爬虫怎么样~

```
async def download(url):
    print("开始抓取")
    await asyncio.sleep(3)  # 我要开始下载了
    print("下载结束", url)
    return "老子是源码你信么"


async def main():
    urls = [
        "http://www.baidu.com",
        "http://www.h.com",
        "http://luoyonghao.com"
    ]
```

```
14      # 生成任务列表
15      tasks = [download(url) for url in urls]
16      done, pedding = await asyncio.wait(tasks)
17      for d in done:
18          print(d.result())
19
20
21  if __name__ == '__main__':
22      asyncio.run(main())
```