Python

什么是 Python 生成器?

generator，有两种产生生成器对象的方式：一种是列表生成式加括号：

`g1 = (x for x in range(10))`

一种是在函数定义中包含`yield`关键字：

```
def fib(max):
    n, a, b = 0, 0, 1
    while n < max:
        yield b
        a, b = b, a + b
        n = n + 1
    return 'done'

g2 = fib(8)
```

对于generator对象g1和g2，可以通过`next(g1)`不断获得下一个元素的值，如果没有更多的元素，就会报错`StopIteration`

也可以通过for循环获得元素的值。

生成器的好处是不用占用很多内存，只需要在用的时候计算元素的值就行了。

什么是 Python 迭代器?

Python中可以用于for循环的，叫做可迭代`Iterable`，包括list/set/tuple/str/dict等数据结构以及生成器；可以用以下语句判断一个对象是否是可迭代的：

```
from collections import Iterable
isinstance(x, Iterable)
```

迭代器`Iterator`，是指可以被`next()`函数调用并不断返回下一个值，直到`StopIteration`；生成器都是Iterator，而列表等数据结构不是；可以通过以下语句将list变为Iterator：

`iter([1,2,3,4,5])`

生成器都是Iterator，但迭代器不一定是生成器。

list 和 tuple 有什么区别?

- list 长度可变，tuple不可变;
- list 中元素的值可以改变，tuple 不能改变;
- list 支持`append`;`insert`;`remove`;`pop`等方法，tuple 都不支持

Python 中的 list 和 dict 是怎么实现的?

Python 中使用多线程可以达到多核CPU一起使用吗?

Python中有一个被称为Global Interpreter Lock（GIL）的东西，它会确保任何时候你的多个线程中，只有一个被执行。线程的执行速度非常之快，会让你误以为线程是并行执行的，但是实际上都是轮流执行。经过GIL这一道关卡处理，会增加执行的开销。

可以通过多进程实现多核任务。

GIL(Global Interpreter Lock)

全局解释器锁

全局解释器锁(Global Interpreter Lock)是计算机程序设计语言解释器用于同步线程的一种机制，它使得任何时刻仅有一个线程在执行。即便在多核处理器上，使用 GIL 的解释器也只允许同一时间执行一个线程，常见的使用 GIL 的解释器有CPython与Ruby MRI。可以看到GIL并不是Python独有的特性，是解释型语言处理多线程问题的一种机制而非语言特性。

GIL的设计初衷?

▶ 展开

单核时代高效利用CPU, 针对解释器级别的数据安全(不是thread-safe 线程安全)。
首先需要明确的是GIL并不是Python的特性，它是在实现Python解析器(CPython)时所引入的一个概念。当Python虚拟机的线程想要调用C的原生线程需要知道线程的上下文，因为没有办法控制C的原生线程的执行，所以只能把上下文关系传给原生线程，同理获取结果也是线
程在python虚拟机这边等待。那么要执行一次计算操作，就必须让执行程序的线程组串行执行。

为什么要加在解释器,而不是在其他层?

▶ 展开

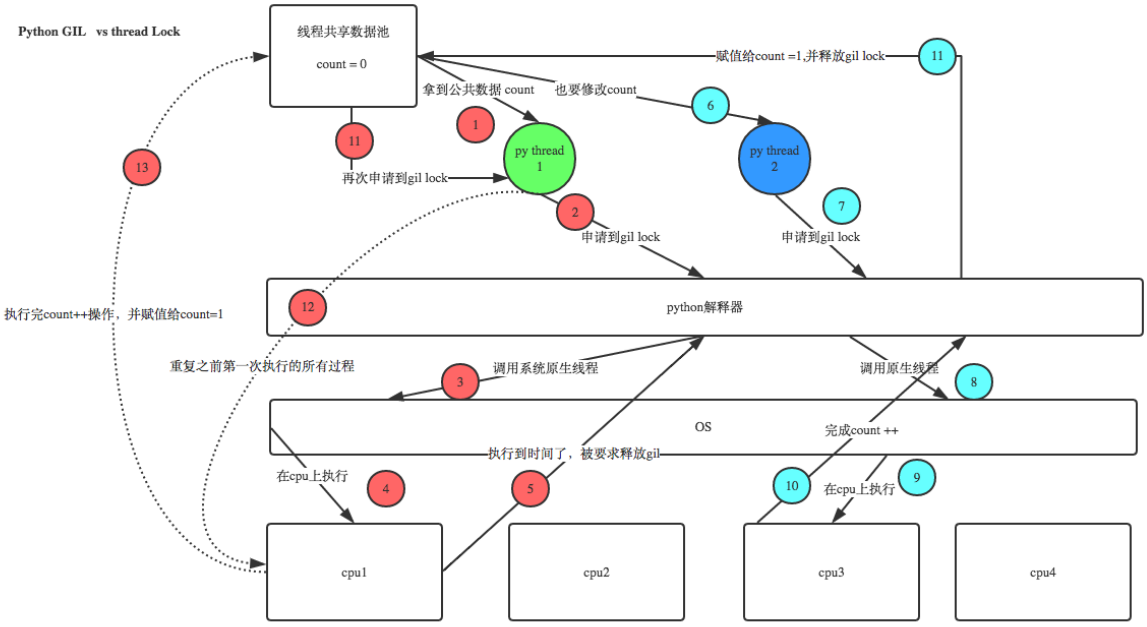
GIL锁加在解释器一层，也就是说Python调用的Cython解释器上加了GIL锁，因为你python调用的所有线程都是原生线程。原生线程是通过C语言提供原生接口，相当于C语言的一个函数。你一调它，你就控制不了了它了，就必须等它给你返回结果。只要已通过python虚拟机
，再往下就不受python控制了，就是C语言自己控制了。加在Python虚拟机以下加不上去，只能加在Python解释器这一层。

GIL的实现是线程不安全?为什么?

▶ 展开

是不安全的，具体情况要分类讨论。

单核情况下:



> 解释:
>
> 1. 到第5步的时候，可能这个时候python正好切换了一次GIL(据说python2.7中，每100条指令会切换一次GIL),执行的时间到了，被要求释放GIL,这个时候thead 1的count=0并没有得到执行，而是挂起状态，count=0这个上下文关系被存到寄存器中.
> 2. 然后到第6步，这个时候thead 2开始执行，然后就变成了count = 1,返回给count，这个时候count=1.
> 3. 然后再回到thead 1，这个时候由于上下文关系，thead 1拿到的寄存器中的count = 0，经过计算，得到count = 1，经过第13步的操作就覆盖了原来的count = 1的值，所以这个时候count依然是count = 1，所以这个数据并没有保护起来。

python2.x和3.x都是在执行IO操作的时候，强制释放GIL，使其他线程有机会执行程序。

Python2.x Python使用计数器ticks计算字节码，当执行100个字节码的时候强制释放GIL，其他线程获取GIL继续执行。ticks可以看作是Python自己的计数器，专门作用于GIL，释放后归零，技术可以调整。

Python3.x Python使用计时器，执行时间达到阈值后，当前线程释放GIL。总体来说比Python3.x对CPU密集型任务更好，但是依然没有解决问题。

多核情况下:

多个CPU情况下，单个CPU释放GIL锁，其他CPU上的线程也会竞争，但是CPU-A可能又马上拿到了GIL，这样其他CPU上的线程只能继续等待，直到重新回到待调度状态。造成多线程在多核CPU情况下，效率反而会下降，出现了大量的资源浪费。

什么是装饰器?

Python 中的垃圾回收机制?

Python垃圾回收机制--完美讲解!

什么是 lambda 表达式?

简单来说，lambda表达式通常是当你需要使用一个函数，但是又不想费脑袋去命名一个函数的时候使用，也就是通常所说的匿名函数。

lambda表达式一般的形式是：关键词lambda后面紧接一个或多个参数，紧接一个冒号“：”，紧接一个表达式

什么是深拷贝和浅拷贝?

赋值（=），就是创建了对象的一个新的引用，修改其中任意一个变量都会影响到另一个。

浅拷贝 copy.copy：创建一个新的对象，但它包含的是对原始对象中包含项的引用（如果用引用的方式修改其中一个对象，另外一个也会修改改变）

深拷贝：创建一个新的对象，并且递归的复制它所包含的对象（修改其中一个，另外一个不会改变）{copy模块的deep.deepcopy()函数}

双等于和 is 有什么区别?

`==`比较的是两个变量的 value，只要值相等就会返回True

`is`比较的是两个变量的 id，即`id(a) == id(b)`，只有两个变量指向同一个对象的时候，才会返回True

但是需要注意的是，比如以下代码：

```
a = 2
b = 2
print(a is b)
```

按照上面的解释，应该会输出False，但是事实上会输出True，这是因为Python中对小数据有缓存机制，-5~256之间的数据都会被缓存。

其它 Python 知识点

类型转换

- list(x)
- str(x)
- set(x)
- int(x)
- tuple(x)

try...except

list

- `lst[a:b]`：左闭右开
- `lst.append(value)`：在末尾添加元素，复杂度O(1)
- `lst.pop()`：弹出列表末尾元素，复杂度O(1)
- `lst.pop(index)`：弹出任意位置元素，将后面的元素前移，复杂度O(n)
- `lst.insert(index, value)`：插入元素，后面的元素后移，复杂度O(n)
- `lst.remove(value)`：移除等于value的第一个元素，后面的元素前移，复杂度O(n)
- `lst.count(value)`：计数值为value的元素个数
- `lst.sort(reverse = False)`：排序，默认升序

参考

- 生成器 - 廖雪峰的官方网站
- Python中的is和==的区别
- 为什么Python多线程无法利用多核
- GIL锁、线程锁(互斥锁)、递归锁(RLock)