

Threading

查看有多少线程

```
len(threading.enumerate(1))
```

threading 什么时候创建

```
[<_MainThread(MainThread, started 140103364720384)>]
[<_MainThread(MainThread, started 140103364720384)>]
----test1--0---
[<_MainThread(MainThread, started 140103364720384)>, <Thread(Thread-1, started 14010333545728)>]
----test1--1---
----test1--2---
----test1--3---
----test1--4---
python@ubuntu:~/Desktop/python就业班/python-05$ cat 05-验证创建线程以及运行时间.py
import threading
import time

def test1():
    for i in range(5):
        print("----test1--%d--- %i)" % (i, i))
        time.sleep(1)

def main():
    # 在调用Thread之前先打印当前线程信息
    print(threading.enumerate())
    t1 = threading.Thread(target=test1)

    # 在调用Thread之后打印
    print(threading.enumerate())

    t1.start()

    # 在调用start之后打印
    print(threading.enumerate())
```

当调用Thread的时候，不会创建线程
当调用Thread创建出来的实例对象的start方法地时候才会创建线程以及让这个线程开始运行

Threading 什么时候结束

线程调用的函数执行完成之后线程结束

使用类作为多线程的执行函数

```
#coding=utf-8
import threading
import time

class MyThread(threading.Thread):
    def run(self):
        for i in range(3):
            time.sleep(1)
            msg = "I'm " + self.name + " @ " + str(i) #name属性中保存的是当前线程的名字
            print(msg)

if __name__ == '__main__':
    t = MyThread()
    t.start()
```

Threading 中传参数

```
3
4 def test1(temp):
5     temp.append(33)
6     print("----in test1 temp=%s----" % str(temp))
7
8
9 def test2(temp):
10    print("----in test2 temp=%s----" % str(temp))
11
12
13 g_nums = [11, 22]
14
15 def main():
16    # target指定将来 这个线程去哪个函数执行代码
17    # args指定将来调用 函数的时候 传递什么数据去
18    t1 = threading.Thread(target=test1, args=(g_nums,))
19    t2 = threading.Thread(target=test2, args=(g_nums,))
20
```

注意 args 中必须是元组

Global

```
1
2
3 num = 100
4 nums = [11, 22]
5
6 def test():
7     global num
8     num += 100
9
10
11 def test2():
12     nums.append(33)
13     num2 += [100, 200]
14
15 print(num)
16 print(nums)
17
18 test()
19 test2()
20
21 print(num)
22 print(nums)
23
```

num 100
nums [11, 22]
[11, 22]
[11, 22] + [100, 200]
[11, 22, 100, 200]

在一个函数中 对全局变量进行修改的时候, 到底是否需要使用global进行说明
要看 是否对 全局变量的执行指向进行了修改。
如果修改了执行, 即让全局变量指向了一个新的地方, 那么必须使用global
如果, 仅仅是修改了 指向的空间中的数据, 此时不用必须使用global

锁

```
# 创建锁
mutex = threading.Lock()

# 锁定
mutex.acquire()

# 释放
mutex.release()
```