首页

Python创建线程（2种方式）详解

Python 中，有关线程开发的部分被单独封装到了模块中，和线程相关的模块有以下 2 个：

- _thread：是 Python 3 以前版本中 thread 模块的重命名，此模块仅提供了低级别的、原始的线程支持，以及一个简单的锁。功能比较有限。正如它的名字所暗示的（以 _ 开头），一般不建议使用 thread 模块；
- threading：Python 3 之后的线程模块，提供了功能丰富的多线程支持，推荐使用。

本节就以 threading 模块为例进行讲解。Python 主要通过两种方式来创建线程：

1. 使用 threading 模块中 Thread 类的构造器创建线程。即直接对类 threading.Thread 进行实例化创建线程，并调用实例化对象的 start() 方法启动线程。
2. 继承 threading 模块中的 Thread 类创建线程类。即用 threading.Thread 派生出一个新的子类，将新建类实例化创建线程，并调用其 start() 方法启动线程。

调用Thread类的构造器创建线程

Thread 类提供了如下的 __init__() 构造器，可以用来创建线程：

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

此构造方法中，以上所有参数都是可选参数，即可以使用，也可以忽略。其中各个参数的含义如下：

- group：指定所创建的线程隶属于哪个线程组（此参数尚未实现，无需调用）；
- target：指定所创建的线程要调度的目标方法（最常用）；
- args：以元组的方式，为 target 指定的方法传递参数；
- kwargs：以字典的方式，为 target 指定的方法传递参数；
- daemon：指定所创建的线程是否为后代线程。

> 这些参数，初学者只需记住 target、args、kwargs 这 3 个参数的功能即可。

下面程序演示了如何使用 Thread 类的构造方法创建一个线程：

```
01.  import threading
02.  #定义线程要调用的方法，*add可接收多个以非关键字方式传入的参数
03.  def action(*add):
04.      for arc in add:
05.          #调用 getName() 方法获取当前执行该程序的线程名
06.          print(threading.current_thread().getName() +" "+ arc)
07.  #定义为线程方法传入的参数
08.  my_tuple = ("http://c.biancheng.net/python/",\
09.              "http://c.biancheng.net/shell/",\
10.              "http://c.biancheng.net/java/")
11.  #创建线程
12.  thread = threading.Thread(target = action,args =my_tuple)
```

> 有关 Thread 类提供的和线程有关的方法，可阅读Python Thread手册，由于不是本节重点，这里不再进行详细介绍。

由此就创建好了一个线程。但是线程需要手动启动才能运行，threading 模块提供了 start() 方法用来启动线程。因此在上面程序的基础上，添加如下语句：

```
01.  thread.start()
```

再次执行程序，其输出结果为：

```
Thread-1 http://c.biancheng.net/python/
Thread-1 http://c.biancheng.net/shell/
Thread-1 http://c.biancheng.net/java/
```

可以看到，新创建的 thread 线程（线程名为 Thread-1）执行了 action() 函数。

> 默认情况下，主线程的名字为 MainThread，用户启动的多个线程的名字依次为 Thread-1、Thread-2、Thread-3、...、Thread-n 等。

为了使 thread 线程的作用更加明显，可以继续在上面程序的基础上添加如下代码，让主线程和新创建线程同时工作：

```
01.  for i in range(5):
02.      print(threading.current_thread().getName())
```

再次执行程序，其输出结果为：

```
MainThreadThread-1 http://c.biancheng.net/python/

MainThreadThread-1 http://c.biancheng.net/shell/

MainThreadThread-1 http://c.biancheng.net/java/

MainThread
MainThread
```

可以看到，当前程序中有 2 个线程，分别为主线程 MainThread 和子线程 Thread-1，它们以并发方式执行，即 Thread-1 执行一段时间，然后 MainThread 执行一段时间。通过轮流获得 CPU 执行一段时间的方式，程序的执行在多个线程之间切换，从而给用户一种错觉，即多个线程似乎同时在执行。

> 如果程序中不显式创建任何线程，则所有程序的执行，都将由主线程 MainThread 完成，程序就只能按照顺序依次执行。

继承Thread类创建线程类

通过继承 Thread 类，我们可以自定义一个线程类，从而实例化该类对象，获得子线程。

需要注意的是，在创建 Thread 类的子类时，必须重写从父类继承得到的 run() 方法。因为该方法即为要创建的子线程执行的方法，其功能如同第一种创建方法中的 action() 自定义函数。

下面程序，演示了如何通过继承 Thread 类创建并启动一个线程：

```
01.  import threading
02.
03.  #创建子线程类，继承自 Thread 类
04.  class my_Thread(threading.Thread):
05.      def __init__(self,add):
06.          threading.Thread.__init__(self)
07.          self.add = add
08.      # 重写run()方法
09.      def run(self):
10.           for arc in self.add:
11.              #调用 getName() 方法获取当前执行该程序的线程名
```

```
            print(threading.current_thread().getName() +" "+ arc)

#定义为 run() 方法传入的参数
my_tuple = ("http://c.biancheng.net/python/",\
            "http://c.biancheng.net/shell/",\
            "http://c.biancheng.net/java/")
#创建子线程
mythread = my_Thread(my_tuple)
#启动子线程
mythread.start()
#主线程执行此循环
for i in range(5):
    print(threading.current_thread().getName())
```

程序执行结果为：

```
MainThreadThread-1 http://c.biancheng.net/python/

MainThreadThread-1 http://c.biancheng.net/shell/

MainThreadThread-1 http://c.biancheng.net/java/

MainThread
MainThread
```

此程序中，子线程 Thread-1 执行的是 run() 方法中的代码，而 MainThread 执行的是主程序中的代码，它们以快速轮换 CPU 的方式在执行。