python子进程模块subprocess详解

作者：jaz
原文链接：https://ld246.com/article/1462524113048
来源网站：链滴

属性
1.Popen.poll()：用于检查子进程是否已经结束。设置并返回returncode属性。
2.Popen.wait()：等待子进程结束。设置并返回returncode属性。
3.Popen.communicate(input=None)：与子进程进行交互。向stdin发送数据，或从stdout和stderr中读取数据。可选参数input指定发送到子进程的参数。Communicate()返回一个元组：(stdoutdata, stderrdata)。注意：如果希望通过进程的stdin向其发送数据，在创建Popen对象的时候，参数stdi必须被设置为PIPE。同样，如果希望从stdout和stderr获取数据，必须将stdout和stderr设置为PIPE
4.Popen.send_signal(signal)：向子进程发送信号。
5.Popen.terminate()：停止(stop)子进程。在windows平台下，该方法将调用Windows API TerminateProcess（）来结束子进程。
6.Popen.kill()：杀死子进程。
7.Popen.stdin：如果在创建Popen对象是，参数stdin被设置为PIPE，Popen.stdin将返回一个件对象用于策子进程发送指令。否则返回None。
8.Popen.stdout：如果在创建Popen对象是，参数stdout被设置为PIPE，Popen.stdout将返回个文件对象用于策子进程发送指令。否则返回None。
9.Popen.stderr：如果在创建Popen对象是，参数stdout被设置为PIPE，Popen.stdout将返回个文件对象用于策子进程发送指令。否则返回None。
10.Popen.pid：获取子进程的进程ID。
11.Popen.returncode：获取进程的返回值。如果进程还没有结束，返回None。
12.subprocess.call(*popenargs, **kwargs)：运行命令。该函数将一直等待到子进程运行结束并返回进程的returncode。文章一开始的例子就演示了call函数。如果子进程不需要进行交互,就可以用该函数来创建。
13.subprocess.check_call(*popenargs, **kwargs)：与subprocess.call(*popenargs, **kwargs功能一样，只是如果子进程返回的returncode不为0的话，将触发CalledProcessError异常。在异常象中，包括进程的returncode信息。
 关于subprocess的安全性：
不像其他的popen函数，不会直接调用/bin/sh来解释命令，也就是说，命令中的每一字符都会被安全地传递到子进程里。
一：用subprocess获取stdout和stderr
第一种方案：
<pre class="brush: py">import subprocess
p = subprocess.Popen(['tail','-10','/tmp/hosts.txt'],stdin=subprocess.PIPE,stdout=subprocess.PIPE,stderr=subprocess.PIPE,shell=False)
stdout,stderr = p.communicate()
print 'stdout : ',stdout
print 'stderr : ',stder </pre>
popen调用的时候会在父进程和子进程建立管道，然后我们可以把子进程的标准输出和错误输出重定向到管道，然后从父进程取出。上面的communicate会一直阻塞，直到子进程跑完。这种方式不能及时获取子程序的stdout和stderr。
第二种方案：
可以获取实时的输出信息
<pre class="brush: py">p = subprocess.Popen("/etc/service/tops-cmos/module/hadoop/test.h", shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
returncode = p.poll()
while returncode is None:
<pre><code class="highlight-chroma">   line = p.stdout.readline()

    returncode = p.poll()

    line = line.strip(

    print line
</code></pre>
print returncode

 </pre>
这里就是把错误输出重定向到PIPE对应的标准输出，也就是说现在stderr都stdout是一起的了，面是一个while，poll回去不断查看子进程是否已经被终止，如果程序没有终止，就一直返回None，是子进程终止了就返货状态码，甚至于调用多次poll都会返回状态码。上面的demo就是可以获取子程的标准输出和标准错误输出。
二：使用subprocess的Popen函数执行系统命令
1、执行shell命令：
Popen函数指定shell=True即可，linux下参数executable将指定程序使用的shell，windows下须指定。
示例1：
在windows下执行cd命令获取当前目录
p2 = Popen('cd',shell=True)
2、执行其他程序
3、指定子进程工作路径：
示例1：
使新建的子进程工作指定的工作目录之下：
<pre class="brush: py">import sys,os,subprocess,commands
from subprocess import Popen,PIPE
p2 = Popen('cd',shell=True,stdout=PIPE,cwd='E:\svnworking')
p2.wait()
print "当前目录:%s" %p2.stdout.read() </pre>

上述命令使用了cwd，该参数指定了子进程工作目录。这个参数很有用，有时涉及到相对路径的候必须如果不指定cwd，则程序可能出错。
示例2：
a.py文件：
p2 = Popen('python c:\b.py',shell=True,stdout=PIPE) #在a.py运行脚本b.py
p2.wait()
print "当前目录:%s" %p2.stdout.read()
b.py文件：
f=open('test.txt','a') #注意这里使用了相对路径
f.close()
当a.py和b.py不在同一个目录的时候，运行a.py肯定报错（找不到指定的文件test.txt）。
原因：因为p2 = Popen('python c:\b.py',shell=True,stdout=PIPE') 创建的子进程与a.py在同目录下工作，而该目录没有test.py。
解决方法：指定cwd参数。
4、获取Popen的返回值及输出
示例：
<pre class="brush: py"># -*- coding: UTF-8 -*-
#执行另外一个脚本
import sys,os,subprocess,commands
from subprocess import Popen,PIPE
p = Popen('python ' + path + '\getCurPath.py', stdout=PIPE, stderr=PIPE)
p.wait()
if(p.returncode == 0):
print "stdout:%s" %p.stdout.read() </pre>

三：subprocess的Popen函数的等待（wait（）方法）
1. Popen对象创建后，主程序不会自动等待子进程完成。
我们必须调用对象的wait()方法，父进程才会等待 (也就是阻塞block)：
    import subprocess

    child = subprocess.Popen(["ping","-c","5","www.google.com"])
    print("parent process")
从运行结果中看到，父进程在开启子进程之后并没有等待child的完成，而是直接运行print。</p

2. 对比等待的情况:
   import subprocess
   child = subprocess.Popen(["ping","-c","5","www.google.com"])
   child.wait()
   print("parent process")
此外，你还可以在父进程中对子进程进行其它操作，比如我们上面例子中的child对象:
child.poll()   # 检查子进程状态
child.kill()   # 终止子进程
child.send_signal()   # 向子进程发送信号
child.terminate()   # 终止子进程
子进程的PID存储在child.pid
四：subprocess的Popen函数的标准输入、标准输出和标准错误
1. 可以在Popen()建立子进程的时候改变标准输入、标准输出和标准错误，
并可以利用subprocess.PIPE将多个子进程的输入和输出连接在一起，构成管道(pipe):
    import subprocess
    child1 = subprocess.Popen(["ls","-l"], stdout=subprocess.PIPE)
    child2 = subprocess.Popen(["wc"], stdin=child1.stdout,stdout=subprocess
PIPE)
    out = child2.communicate()
    print(out)
subprocess.PIPE实际上为文本流提供一个缓存区。
child1的stdout将文本输出到缓存区，随后child2的stdin从该PIPE中将文本读取走。
child2的输出文本也被存放在PIPE中，直到communicate()方法从PIPE中读取出PIPE中的文本。
/p>
要注意的是，communicate()是Popen对象的一个方法，该方法会阻塞父进程，直到子进程完成

2. 还可以利用communicate()方法来使用PIPE给子进程输入:
    import subprocess
    child = subprocess.Popen(["cat"], stdin=subprocess.PIPE)
    child.communicate("vamei")
我们启动子进程之后，cat会等待输入，直到我们用communicate()输入"vamei"。
通过使用subprocess包，我们可以运行外部程序。这极大的拓展了Python的功能。
如果你已经了解了操作系统的某些应用，你可以从Python中直接调用该应用(而不是完全依赖Pyt
on)，
并将应用的结果输出给Python，并让Python继续处理。
shell的功能(比如利用文本流连接各个应用)，就可以在Python中实现。
<pre class="brush: py">

</pre>
