subprocess执行系统命令

```
subprocess模块可以以阻塞或非阻塞的形式执行操作系统命令.
```

In [2]:
```
import subprocess

print(dir(subprocess))
```

```
['CalledProcessError', 'CompletedProcess', 'DEVNULL', 'PIPE', 'Popen', 'STDOUT', 'SubprocessError', 'TimeoutExpired', '_PIPE_BUF', '_PLATFORM_DEFAULT_CLOSE_FDS', '_P
openSelector', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_active', '_args_from_interprete
r_flags', '_cleanup', '_mswindows', '_optim_args_from_interpreter_flags', '_posixsubprocess', '_time', 'builtins', 'call', 'check_call', 'check_output', 'errno', 'ge
toutput', 'getstatusoutput', 'io', 'list2cmdline', 'os', 'run', 'select', 'selectors', 'signal', 'sys', 'threading', 'time', 'warnings']
```

```
#常用方法.

#1.call() 以阻塞的方式执行系统命令, call返回值是命令的退出码, 调用者通过这个状态码来判断命令是否执行成功.

help(subprocess.call)

call(*popenargs, **kwargs)
    Run command with arguments.  Wait for command to complete, then
    return the returncode attribute.

    The arguments are the same as for the Popen constructor.  Example:

    retcode = call(["ls", "-l"])
(END)
```

In [ ]:
```
code = subprocess.call('ping -c 4 127.0.0.1', shell=True)
print(code)
```

In [8]:
```
#2. check_call() 函数与call()类似, 当执行命令出错时会抛出一个错误CalledProcessError
```

In [9]:
```
x = subprocess.check_call('ls /tmp', shell=True)
print(x)

y = subprocess.check_call('ls1 /tmp', shell=True)
print(y)
```

```
0

---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-9-92de18757548> in <module>
      2 print(x)
      3
----> 4 y = subprocess.check_call('ls1 /tmp', shell=True)
      5 print(y)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/subprocess.py in check_call(*popenargs, **kwargs)
    289         if cmd is None:
    290             cmd = popenargs[0]
--> 291         raise CalledProcessError(retcode, cmd)
    292     return 0
    293

CalledProcessError: Command 'ls1 /tmp' returned non-zero exit status 127.
```

In [10]:
```
# 3. check_output 补获执行命令的结果, 当执行命令出错时 抛出 CalledProcessError
x = subprocess.check_output('ls /tmp', shell=True)
print(x)
```

```
b'AlTest1.err\nAlTest1.out\nUniAccessAgentOnlyOneInstance\nadobegc.log\ncom.adobe.acrobat.rna.RdrCefBrowserLock.DC\ncom.adobe.reader.rna.0.1f5.DC\ncom.adobe.reader.r
na.352.1f5\ncom.apple.launchd.b8EoahYgL6\ncom.apple.launchd.x9rmgK2Sbv\ncvcd\ndavinci-gm-sec-policy\ndavinci-gm-user-config-mgr\ndvc-gui-app-mark-liushuo\nlv-agent-c
onfig-sec-pol-arc-lock-mark\nlv-agent-filetrlex2-arc-lock-mark\npowerlog\n'
```

In [11]:
```
x = subprocess.check_output('ls1 /tmp', shell=True)
```

```
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-11-7baa02961758> in <module>
----> 1 x = subprocess.check_output('ls1 /tmp', shell=True)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/subprocess.py in check_output(timeout, *popenargs, **kwargs)
    334
    335     return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
--> 336                **kwargs).stdout
    337
    338

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/subprocess.py in run(input, timeout, check, *popenargs, **kwargs)
    416         if check and retcode:
    417             raise CalledProcessError(retcode, process.args,
--> 418                                      output=stdout, stderr=stderr)
    419     return CompletedProcess(process.args, retcode, stdout, stderr)
    420

CalledProcessError: Command 'ls1 /tmp' returned non-zero exit status 127.
```

```
#3.Popen() 该类用于在一个新的进程中执行一个子程序, 上面介绍的这些函数都是基于subprocess.Popen类实现的, 通过使用这些被封装后的高级函数可以很方面的完成一些常见的需求. 由于subprocess模块底层的
进程创建和管理是由Popen类来处理的, 因此, 当我们无法通过上面哪些高级函数来实现一些不太常见的功能时就可以通过subprocess.Popen类提供的灵活的api来完成.
```

In [13]:
```
print(dir(subprocess.Popen))
```

```
['__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__', '__gt__', '_
_hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof
__', '__str__', '__subclasshook__', '__weakref__', '_check_timeout', '_child_created', '_communicate', '_execute_child', '_get_devnull', '_get_handles', '_handle_exi
tstatus', '_internal_poll', '_remaining_time', '_save_input', '_stdin_write', '_translate_newlines', '_try_wait', 'communicate', 'kill', 'poll', 'send_signal', 'term
inate', 'wait']
```

In [15]:
```
#3.1 Popen() 可以以非阻塞的形式执行系统命令
subprocess.Popen('sleep 3; echo 1', shell=True)
print("main process")
```

```
main process  系统命令执行后, 主进程继续往下执行. 并没有等执行系统命令的子进程返回后才执行, 所以是非阻塞的

bash-3.2$ PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.155 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.141 ms
```

In [19]:
```
#3.2 处理输出, subprocess.PIPE 通道
x = subprocess.Popen('ls /tmp', shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
y = subprocess.Popen('ls1 /tmp', shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
print(x.stdout.read())
print("##" * 10)
print(x.stderr.read())
print("##" * 10)
print(y.stdout.read())
print("##" * 10)
print(y.stderr.read())
```

```
b'AlTest1.err\nAlTest1.out\nUniAccessAgentOnlyOneInstance\nadobegc.log\ncom.adobe.acrobat.rna.RdrCefBrowserLock.DC\ncom.adobe.reader.r
na.352.1f5\ncom.apple.launchd.b8EoahYgL6\ncom.apple.launchd.x9rmgK2Sbv\ncvcd\ndavinci-gm-sec-policy\ndavinci-gm-user-config-mgr\ndvc-gui-app-mark-liushuo\nlv-agent-c
onfig-sec-pol-arc-lock-mark\nlv-agent-filetrlex2-arc-lock-mark\npowerlog\n'
####################
b''
####################
b''
####################
b'/bin/sh: ls1: command not found\n'
```

```
3.3 Popen() 类可调用的方法

Popen.pool() 用于检查子进程 (命令) 是否已经执行结束, 没结束返回None, 结束后返回状态码.
Popen.wait() 等待子进程结束, 如果在指定秒数没有结束 则返回TimeExpire异常, python2中与python3中不一样
Popen.terminate()  #停止该进程
Popen.kill() #杀死该进程
```

In [3]:
```
import subprocess
l = subprocess.Popen('sleep 4; echo 1', shell=True)
print(l.poll())
l.wait()
print(l.poll(), 'sub process finished')
```

```
None
0 sub process finished
```

In [4]:
```
help(l.wait)
```

```
Help on method wait in module subprocess:

wait(timeout=None, endtime=None) method of subprocess.Popen instance
    Wait for child process to terminate.  Returns returncode
    attribute.
```

In [5]:
```
import subprocess
l = subprocess.Popen('sleep 4; echo 1', shell=True)
print(l.poll())
l.wait(timeout=2)
print(l.poll(), 'sub process finished')
```

```
None

---------------------------------------------------------------------------
TimeoutExpired                            Traceback (most recent call last)
<ipython-input-5-7c6c0500f20c> in <module>
      2 l = subprocess.Popen('sleep 4; echo 1', shell=True)
      3 print(l.poll())
----> 4 l.wait(timeout=2)
      5 print(l.poll(), 'sub process finished')

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/subprocess.py in wait(self, timeout, endtime)
   1426                     remaining = self._remaining_time(endtime)
   1427                     if remaining <= 0:
-> 1428                         raise TimeoutExpired(self.args, timeout)
   1429                     delay = min(delay * 2, remaining, .05)
   1430                     time.sleep(delay)

TimeoutExpired: Command 'sleep 4; echo 1' timed out after 2 seconds
```

In [6]:
```
help(l.terminate)
```

```
Help on method terminate in module subprocess:

terminate() method of subprocess.Popen instance
    Terminate the process with SIGTERM
```

In [7]:
```
help(l.kill)
```

```
Help on method kill in module subprocess:

kill() method of subprocess.Popen instance
    Kill the process with SIGKILL
```

示例代码：执行操作系统命令设置超时时间
在日常工作中调用操作系统命令可能会出现长时间不返回结果的情况, 如果我们不想持续等下去, 或者只想等待指定的时间 如果超过这个时间就不等待了
可以按照下面方式来写代码

In [16]:
```
class TimeoutError(Exception):
    pass

import subprocess
import time
def os_system(cmd, timeout):
    beg_time = int(time.time())
    s = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    while 1:
        stop = s.poll()
        if stop is not None:
            break
        end_time = int(time.time())
        if (end_time - beg_time) >= timeout:
            s.terminate()
            raise TimeoutError("cmd exe expired!")
        time.sleep(0.1)
    result = s.stdout.read()
    return result

print(os_system('sleep 3; ls /tmp', 2))
```

```
---------------------------------------------------------------------------
TimeoutError                              Traceback (most recent call last)
<ipython-input-16-26e9ce5838f0> in <module>
     19     return result
     20 
---> 21 print(os_system('sleep 3; ls /tmp', 2))

<ipython-input-16-26e9ce5838f0> in os_system(cmd, timeout)
     14         if (end_time - beg_time) >= timeout:
     15             s.terminate()
---> 16             raise TimeoutError("cmd exe expired!")
     17         time.sleep(0.1)
     18     result = s.stdout.read()

TimeoutError: cmd exe expired!
```

获取执行脚本的实时输出，通过Python调用系统中的Shell脚本，如果希望能够实时看到脚本的输出信息，可以使用如下方式。
示例代码：

In [10]:
```
import subprocess
import sys

def cmd_realtime_output(cmd):
    output = ''
    p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    while p.poll() is None:
        line =str(p.stdout.readline(), encoding="utf-8") #python3
        #line = p.stdout.readline() #python2
        line = line.strip() + "\n"
        output += line
        sys.stdout.write(line)
    if p.returncode == 0:
        result = {"exite_code":0, "message": output}
        return result
    else:
        result = {"exit_code": 1, "message": "CMD execute failed!"}
        return result

if __name__ == '__main__':
    cmd_realtime_output("ping www.baidu.com -t 3")
```

```
PING www.a.shifen.com (14.215.177.38): 56 data bytes
64 bytes from 14.215.177.38: icmp_seq=0 ttl=53 time=11.101 ms
64 bytes from 14.215.177.38: icmp_seq=1 ttl=53 time=24.842 ms
64 bytes from 14.215.177.38: icmp_seq=2 ttl=53 time=14.630 ms

--- www.a.shifen.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 11.101/16.858/24.842/5.827 ms
```

#3.4 管道连接

多个命令可以连接为一个管线，分别创建多个Popen实例，把他们的输入和输出串联在一起，一个Popen实例的stdout可以成为另一个Popen实例的输入

In [36]:
```
import subprocess
ls = subprocess.Popen('ls /tmp', shell=True, stdout=subprocess.PIPE)
grep = subprocess.Popen("grep com", shell=True, stdin=ls.stdout, stdout=subprocess.PIPE)
print(grep.stdout.read())
```

```
b'com.adobe.acrobat.rna.RdrCefBrowserLock.DC\ncom.adobe.reader.rna.0.1f5.DC\ncom.adobe.reader.rna.352.1f5\ncom.apple.launchd.b8EoahYgL6\ncom.apple.launchd.x9rmgK2Sbv
\n'
```