Python subprocess 模块总结[windows]

subprocess模块用于替换os.system旧模块，在python父进程中创建子进程，以批处理命令形式执行。

**p=subprocess.call(‘cmd’,stdout=None;stderr=None，shell=True)**

没有p.wait()，父进程直接等待子进程p完成，p为返回值[正常返回为0，异常返回则>0，异常退出<0]，p没有returncode；

cmd可以为字符串也可以为序列或文件，为序列时，第一项为批处理命令，后面各项为参数；

shell=True建议打开以确保命令正确执行，因为就算cmd为不含空格的字符串，不设置都可能报错。不过，shell=True存在极大的**命令注入**风险。可以通过shlex对命令分割后执行。

stdout/stderr为输出流控制参数，可选为None、subprocess.PIPE管道及文件对象等，stdout不能是标准输出STDOUT,stderr可以[因为stdout只能通过流传递，貌似没必要这样，毕竟错误是要打印出来看的]。subprocess.PIPE通过管道将输出重定向到缓冲区[或许是为了另一个子进程可以从缓冲区通过stdin读取p的输出]，导致子进程运行过程中，日志不会直接打印。官方文档不建议指定stdout参数，因为该方法没有类型Popen的communicate**进程通信**方法，极易导致子**进程死锁[**死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象]。

**p=subprocess.run(‘cmd’,stdout=None;stderr=None，shell=True)**

run()方法在py3.5才引入，与call()大致类似但更加成熟。subprocess.run返回CompletedProcess对象：

p.returncode:子进程返回值

p.args：cmd参数

p.stdout:获取子进程输出

p.stderr:获取子进程错误日志

因为父进程会等待直到子进程执行完毕，子进程执行过程中，输出会首先输入缓存区。stdout=None时，输出与具体执行命令[按条]对应，命令执行完，即输出[直接输出文本日志]，命令执行中，输出先存入缓存区；stdout=subprocess.PIPE时，只有在子进程执行完毕后，才打印日志，并且PIPE为字节流，直接打印不友好，可以先解码，再打印：print(p.stdout.decode(‘utf-8’)),含中文解码为’gbk’。

**p=subprocess.check\_call(‘cmd’,stdout=None;stderr=None，shell=True)**

父进程等待子进程完成，p只有在cmd得到正确执行且返回值为0的情况下，才会得到正确赋值0，如果cmd执行错误或者cmd返回非零，p都将得不到赋值，产生未定义行为。因此，不要将subprocess.check\_call赋值给p。使用该方法最好的方式就是try…except:

***try:***

***subprocess.check\_call()***

***except subprocess.CalledProcessError as ERR :***

***print（ERR.returncode）***

***print(‘exitcode not zero)***

建议不指定stdout，直接打印输出，避免进程死锁。如果返回值不为0，则抛出subprocess.CalledProcessError，执行except语句内容。

**p=subprocess.check\_output(‘cmd’,stdout=None;stderr=None，shell=True)**

父进程等待子进程完成，p只有在cmd得到正确执行且返回值为0的情况下，才会得到正确赋值0，如果cmd执行错误或者cmd返回非零，p都将得不到赋值，产生未定义行为。因此，不要将subprocess.check\_call赋值给p。使用该方法最好的方式就是try…except:

***try:***

***subprocess.check\_output()***

***except subprocess.CalledProcessError as ERR :***

***print（ERR.returncode）***

***print（ERR.stdout）***

***print(‘exitcode not zero)***

子进程运行完毕不会输入日志，不允许指定stdout，因为subprocess.CalledProcessError对象有stdout属性，用于日志输出。

**p=subprocess.Popen(‘cmd’, bufsize=0, executable=None, stdin=None, stdout=None, stderr=None, preexec\_fn=None, close\_fds=False, shell=False, cwd=None, env=None, universal\_newlines=False, startupinfo=None, creationflags=0)**

最具定制性的子进程生成方法，包括以下方法和属性：

p.wait() # 父进程等待直到子进程结束

p.poll() # 检查子进程状态，结束返回0，运行中返回None

p.kill() # 杀死子进程

p.terminate() # 终止子进程

p.send\_signal() # 向子进程发送信号，进程通信

p.communicate() # 获取管道缓冲区输出和错误信息，进程通信

p.pid # 子进程pid

p.stdout # 标准输出，可打印控制台，输出到管道和文件

p.stderr # 标准错误，可打印控制台，输出到管道和文件

p.stdin # 标准输入

universal\_newlines=True时，p.stdout.readline()为str，而非bytes，无需解码即可输出。

stdout,stderr=p.communicate(stdin)实现进程通信