# 1基础

http://www.ibm.com/developerworks/cn/linux/l-cn-pexpect1/index.html

## 概述

Pexpect 是 Don Libes 的 Expect 语言的一个 Python 实现，是一个用来启动子程序，并使用正则表达式对程序输出做出特定响应，以此实现与其自动交互的 Python 模块。 Pexpect 的使用范围很广，可以用来实现与 ssh、ftp 、telnet 等程序的自动交互；可以用来自动复制软件安装包并在不同机器自动安装；还可以用来实现软件测试中与命令行交互的自动化。

## 下载

Pexpect 可以从 SourceForge 网站下载。 本文介绍的示例使用的是 2.3 版本，如不说明测试环境，默认运行操作系统为 fedora 9 并使用 Python 2.5 。

## 安装

```
download pexpect-2.3.tar.gz
tar zxvf pexpect-2.3.tar.g
cd pexpect-2.3
python setup.py install  (do this as root)
```

## 依赖

- Python 版本 2.4 或者 2.5
- pty module ， pty 是任何 Posix 系统标准库的一部分

由于其依赖 pty module ，所以 Pexpect 还不能在 Windows 的标准 python 环境中执行，如果想在 Windows 平台使用，可以使用在 Windows 中运行 Cygwin 做为替代方案。

## 遵循 MIT 许可证

根据 Wiki 对 MIT License 的介绍“该模块被授权人有权利使用、复制、修改、合并、出版发行、散布、再授权及贩售软件及软件的副本。被授权人可根据程序的需要修改授权条款为适当的内容。在软件和软件的所有副本中都必须包含版权声明和许可声明。”

回页首

# Pexpect 提供的 run() 函数：

**清单 1. run() 的定义**

```
run(command,timeout=-1,withexitstatus=False,events=None,\
    extra_args=None,logfile=None, cwd=None, env=None)
```

函数 run 可以用来运行命令，其作用与 Python os 模块中 system() 函数相似。run() 是通过 Pexpect 类实现的。

如果命令的路径没有完全给出，则 run 会使用 which 命令尝试搜索命令的路径 。

**清单 2. 使用 run()执行 svn 命令**

```
from pexpect import *
run ("svn ci -m 'automatic commit' my_file.py")
```

与 os.system() 不同的是，使用 run() 可以方便地同时获得命令的输出结果与命令的退出状态 。

**清单 3. run() 的返回值**

```
from pexpect import *
(command_output, exitstatus) = run ('ls -l /bin', withexitstatus=1)
```

command_out 中保存的就是 /bin 目录下的内容。

回页首

# Pexpect 提供的 spawn() 类：

## 使用 Pexpect 启动子程序

**清单 4. spawn 的构造函数**

```
class spawn:
    def __init__(self,command,args=[],timeout=30,maxread=2000,\
    searchwindowsize=None, logfile=None, cwd=None, env=None)
```

spawn是Pexpect模块主要的类，用以实现启动子程序，它有丰富的方法与子程序交互从而实现用户对子程序的控制。它主要使用 pty.fork() 生成子进程，并调用 exec() 系列函数执行 command 参数的内容。

可以这样使用：

**清单 5. spawn() 使用示例**

```
child = pexpect.spawn ('/usr/bin/ftp') #执行ftp客户端命令
child = pexpect.spawn ('/usr/bin/ssh user@example.com') #使用ssh登录目标机器
child = pexpect.spawn ('ls -latr /tmp') #显示 /tmp目录内容
```

当子程序需要参数时，还可以使用一个参数的列表：

**清单 6. 参数列表示例**

```
child = pexpect.spawn ('/usr/bin/ftp', [])
child = pexpect.spawn ('/usr/bin/ssh', ['user@example.com'])
```

```
child = pexpect.spawn ('ls', ['-latr', '/tmp'])
```

在构造函数中，maxread 属性指定了 Pexpect 对象试图从 tty 一次读取的最大字节数，它的默认值是2000字节 。

由于需要实现不断匹配子程序输出， searchwindowsize 指定了从输入缓冲区中进行模式匹配的位置，默认从开始匹配。

logfile 参数指定了 Pexpect 产生的日志的记录位置。

例如：

**清单 7. 记录日志**

```
child = pexpect.spawn('some_command')
fout = file('mylog.txt','w')
child.logfile = fout
```

还可以将日志指向标准输出：

**清单 8. 将日志指向标准输出**

```
child = pexpect.spawn('some_command')
child.logfile = sys.stdout
```

如果不需要记录向子程序输入的日志，只记录子程序的输出，可以使用：

**清单 9. 记录输出日志**

```
child = pexpect.spawn('some_command')
child.logfile_send = sys.stdout
```

## 使用 Pexpect 控制子程序

为了控制子程序，等待子程序产生特定输出，做出特定的响应，可以使用 expect 方法。

**清单 10. expect() 定义**

```
expect(self, pattern, timeout=-1, searchwindowsize=None)
```

在参数中： pattern 可以是正则表达式， pexpect.EOF ， pexpect.TIMEOUT ，或者由这些元素组成的列表。需要注意的是，当 pattern 的类型是一个列表时，且子程序输出结果中不止一个被匹配成功，则匹配返回的结果是缓冲区中最先出现的那个元素，或者是列表中最左边的元素。使用 timeout 可以指定等待结果的超时时间 ，该时间以秒为单位。当超过预订时间时， expect 匹配到 pexpect.TIMEOUT。

如果难以估算程序运行的时间，可以使用循环使其多次等待直至等待运行结束：

**清单 11. 使用循环**

```
while True:
    index = child.expect(["suc","fail",pexpect.TIMEOUT])
    if index == 0:
        break
elif index == 1:
        return False
elif index == 2:
        pass        #continue to wait
```

expect() 在执行中可能会抛出两种类型的异常分别是 EOF and TIMEOUF，其中 EOF 通常代表子程序的退出， TIMEOUT 代表在等待目标正则表达式中出现了超时。

**清单 12. 使用并捕获异常**

```
try:
    index = pexpect (['good', 'bad'])
        if index == 0:
            do_something()
        elif index == 1:
            do_something_else()
    except EOF:
        do_some_other_thing()
except TIMEOUT:
    do_something_completely_different()
```

此时可以将这两种异常放入expect等待的目标列表中：

**清单 13. 避免异常**

```
index = p.expect (['good', 'bad', pexpect.EOF, pexpect.TIMEOUT])
if index == 0:
    do_something()
elif index == 1:
    do_something_else()
elif index == 2:
    do_some_other_thing()
elif index == 3:
    do_something_completely_different()
```

expect 不断从读入缓冲区中匹配目标正则表达式，当匹配结束时 pexpect 的 before 成员中保存了缓冲区中匹配成功处之前的内容， pexpect 的 after 成员保存的是缓冲区中与目标正则表达式相匹配的内容。

**清单 14. 打印 before 成员的内容**

```
child = pexpect.spawn('/bin/ls /')
child.expect (pexpect.EOF)
print child.before
```

此时 child.before 保存的就是在根目录下执行 ls 命令的结果。

**清单 15. send 系列函数**

```
send(self, s)
sendline(self, s='')
sendcontrol(self, char)
```

这些方法用来向子程序发送命令，模拟输入命令的行为。 与 send() 不同的是 sendline() 会额外输入一个回车符 ，更加适合用来模拟对子程序进行输入命令的操作。 当需要模拟发送 “Ctrl+c” 的行为时，还可以使用 sendcontrol() 发送控制字符。

**清单 16. 发送 ctrl+c**

```
child.sendcontrol('c')
```

由于 send() 系列函数向子程序发送的命令会在终端显示，所以也会在子程序的输入缓冲区中出现，因此不建议使用 expect 匹配最近一次 sendline() 中包含的字符。否则可能会在造成不希望的匹配结果。

**清单 17. interact() 定义**

```
interact(self, escape_character = chr(29), input_filter = None, output_filter = None)
```

Pexpect还可以调用interact() 让出控制权，用户可以继续当前的会话控制子程序。用户可以敲入特定的退出字符跳出，其默认值为“^]” 。

下面展示一个使用Pexpect和ftp交互的实例。

**清单 18. ftp 交互的实例：**

```
# This connects to the openbsd ftp site and
# downloads the README file.
import pexpect
child = pexpect.spawn ('ftp ftp.openbsd.org')
child.expect ('Name .*: ')
child.sendline ('anonymous')
child.expect ('Password:')
child.sendline ('noah@example.com')
child.expect ('ftp> ')
child.sendline ('cd pub/OpenBSD')
child.expect('ftp> ')
child.sendline ('get README')
child.expect('ftp> ')
child.sendline ('bye')
```

该程序与 ftp 做交互，登录到 ftp.openbsd.org ，当提述输入登录名称和密码时输入默认用户名和密码，当出现 “ftp>” 这一提示符时切换到 pub/OpenBSD 目录并下载 README 这一文件。

以下实例是上述方法的综合应用，用来建立一个到远程服务器的 telnet 连接，并返回保存该连接的 pexpect 对象。

**清单 19. 登录函数：**

```
import re,sys,os
from pexpect import *

def telnet_login(server,user, passwd,shell_prompt= “#|->”):
    """
    @summary: This logs the user into the given server.
    It uses the 'shell_prompt' to try to find the prompt right after login.
    When it finds the prompt it immediately tries to reset the prompt to '#UNIQUEPROMPT#
    more easily matched.
    @return: If Login successfully ,It will return a pexpect object
    @raise exception: RuntimeError will be raised when the cmd telnet failed or the user
    and passwd do not match
    @attention:1. shell_prompt should not include '$',on some server, after sendline
                (passwd)   the pexpect object will read a '$'.
    2.sometimes the server's output before its shell prompt will contain '#' or
     '->'  So the caller should kindly assign the shell prompt
    """
    if not server or not user \
          or not passwd or not shell_prompt:
        raise RuntimeError, "You entered empty parameter for telnet_login "

    child = pexpect.spawn('telnet %s' % server)
    child.logfile_read = sys.stdout
    index = child.expect (['(?i)login:', '(?i)username', '(?i)Unknown host'])
    if index == 2:
        raise RuntimeError, 'unknown machine_name' + server
    child.sendline (user)
    child.expect ('(?i)password:')
    child.logfile_read = None  # To turn off log
    child.sendline (passwd)

    while True:
        index = child.expect([pexpect.TIMEOUT,shell_prompt])
        child.logfile_read = sys.stdout
        if index == 0:
            if re.search('an invalid login', child.before):
                raise RuntimeError, 'You entered an invalid login name or password.'
        elif index == 1:
            break
    child.logfile_read = sys.stdout # To tun on log again
    child.sendline(“PS1=#UNIQUEPROMPT#”)
    #This is very crucial to wait for PS1 has been modified successfully
    #child.expect(“#UNIQUEPROMPT#”)
    child.expect("%s.+%s" % (“#UNIQUEPROMPT#”, “#UNIQUEPROMPT#”))
    return child
```

回页首

## Pxssh 类的使用：

Pxssh 做为 pexpect 的派生类可以用来建立一个 ssh 连接，它相比其基类增加了如下方法：

login() 建立到目标机器的ssh连接 ；

losuckgout() 释放该连接 ；

prompt() 等待提示符，通常用于等待命令执行结束。

下面的示例连接到一个远程服务器，执行命令并打印命令执行结果。

该程序首先接受用户输入用户名和密码，login 函数返回一个 pxssh 对象的链接，然后调用 sendline() 分别输入 “uptime”、“ls” 等命令并打印命令输出结果。

**清单 20. pxssh 示例**

```
import pxssh
import getpass
try:
s = pxssh.pxssh()
    hostname = raw_input('hostname: ')
    username = raw_input('username: ')
    password = getpass.getpass('password: ')
    s.login (hostname, username, password)
    s.sendline ('uptime')  # run a command
    s.prompt()             # match the prompt
    print s.before         # print everything before the propt.
    s.sendline ('ls -l')
    s.prompt()
    print s.before
    s.sendline ('df')
    s.prompt()
    print s.before
    s.logout()
except pxssh.ExceptionPxssh, e:
    print "pxssh failed on login."
    print str(e)
```

回页首

# Pexpect 使用中需要注意的问题：

## spawn() 参数的限制

在使用spawn执行命令时应该注意，Pexpect 并不与 shell 的元字符例如重定向符号 > 、>> 、管道 | ，还有通配符 * 等做交互，所以当想运行一个带有管道的命令时必须另外启动一个 shell ，为了使代码清晰，以下示例使用了参数列表例如：

**清单 21. 启动新的 shell 执行命令**

```
shell_cmd = 'ls -l | grep LOG > log_list.txt'
child = pexpect.spawn('/bin/bash', ['-c', shell_cmd])
child.expect(pexpect.EOF)
```

## 与线程共同工作

Perl 也有 expect 的模块 Expect-1.21，但是 perl 的该模块在某些操作系统例如 fedora 9 或者 AIX 5 中不支持在线程中启动程序执行以下实例试图利用多线同时程登录到两台机器进行操作，不使用线程直接调用时 sub1() 函数可以正常工作，但是使用线程时在 fedora9 和 AIX 5 中都不能正常运行。

**清单 22. perl 使用 expect 由于线程和 expect 共同使用导致不能正常工作的程序**

```
use threads;
use Expect;
$timeout = 5;
my $thr = threads->create(\&sub1(first_server));
my $thr2 = threads->create(\&sub1(second_server));
sub sub1
{
        my $exp = new Expect;
        $exp -> raw_pty(1);
        $exp -> spawn ("telnet",$_[0]) or die "cannot access telnet";
        $exp -> expect ( $timeout, -re=>'[Ll]ogin:' );
        $exp -> send ( "user\n");
        $exp -> expect ( $timeout, -re=>'[Pp]assword:' );
        $exp -> send ( "password\n" );
        $exp -> expect ( $timeout, -re=>" #" );
        $exp -> send ( "date\n" );
        $exp -> expect ( $timeout, -re=>'\w\w\w \w\w\w \d{1,2}
                        \d\d:\d\d:\d\d \w\w\w \d\d\d\d');
        $localtime=$exp->match();
        print "\tThe first server's time is : $localtime\n";
        $exp -> soft_close ();
}
print "This is the main thread!";
$thr->join();
$thr2->join();
```

Pexpect 则没有这样的问题，可以使用多线程并在线程中启动程序运行。但是在某些操作系统如 fedora9 中不可以在线程之间传递 Pexpect 对象。

**清单 使用 Pexpect 在线程中启动控制子程序**

请参见实例

## 对正则表达式的支持

在使用 expect() 时，由于 Pexpect 是不断从缓冲区中匹配，如果想匹配行尾不能使用 “$” ，只能使用 “\r\n”代表一行的结束。 另外其只能得到最小匹配的结果，而不是进行贪婪匹配，例如 child.expect ('.+') 只能匹配到一个字符。

回页首

# 应用实例：

在实际系统管理员的任务中，有时需要同时管理多台机器，这个示例程序被用来自动编译并安装新的内核版本，并重启。它使用多线程，每个线程都建立一个到远程机器的 telnet 连接并执行相关命令。 该示例会使用上文中的登录函数。

**清单 23. 管理多台机器示例**

```
import sys,os
from Login import *
PROMPT = "#UNIQUEPROMPT#"
```

```
class RefreshKernelThreadClass(threading.Thread):
""" The thread to downLoad the kernel and install it on a new server  """
    def __init__(self,server_name,user,passwd):
        threading.Thread.__init__(self)
        self.server_name_ = server_name
        self.user_ = user
        self.passwd_ = passwd
        self.result_ = [] # the result information of the thread

    def run(self):
        self.setName(self.server_name_) # set the name of thread

        try:
            #call the telnet_login to access the server through telnet
            child = telnet_login(self.server_name_,self.user_, self.passwd_)
        except RuntimeError,ex:
            info = "telnet to machine %s failed with reason %s" % (self.server_name_, ex)
self.result_.=(False, self.server_name_+info)
            return self.result_

        child.sendline(' cd ~/Download/dw_test && \
             wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.tar.gz && \
             tar zxvf linux-2.6.28.tar.gz && \
             cd linux-2.6.28 \
             && make mrproper && make allyesconfig and make -j 4 && make modules && \
             make modules install && make install')

        # wail these commands finish
        while True:
            index = child.expect([PROMPT,pexpect.TIMEOUT,pexpect.EOF])
            if index == 0:
                break
            elif index == 1:
                pass
            elif index ==2 :
                self.result_=(False,'Sub process exit abnormally ')
            return False

        # reboot the server
        child.sendline('shutdown -Fr')
        child.expect('\r\n')
        retry_times = 10
        while retry_times > 0:
            index_shutdown = child.expect(["Unmounting the file systems",
                                           pexpect.EOF,
                                           pexpect.TIMEOUT])
            if index_shutdown == 0 or index_shutdown == 1 :
                break
            elif index_shutdown == 2:
                retry_times = retry_times-1
                if retry_times == 0:
                    self.result_=(False,'Cannot shutdown ')
                    return self.result_


def refresh_kernel(linux_server_list,same_user,same_passwd):
    """
    @summary: The function is used to work on different linux servers to download
    the same version linux kernel, conpile them and reboot all these servers
    To keep it simple we use the same user id and password on these servers
    """
    if not type(linux_server_list) == list:
        return (False,"Param %s Error!"%linux_server_list)

    if same_user is None or same_passwd is None or not
    type(same_user)== str or not type(same_passwd) == str:
        return (False,"Param Error!")

    thread_list = []
    # start threads to execute command on the remote servers
    for i in range (len(linux_server_list)):
        thread_list[i] = RefreshKernelThreadClass(linux_server_list[i],
                                                  same_user,same_passwd)
        thread_list[i].start()

    # wait the threads finish
    for i in range (len(linux_server_list)):
        thread_list[i].join()
    # validate the result
    for i in range (len(linux_server_list)):
        if thread_list[0].result_[0] == False:
            return False
        else:
            return True

if __name__ == "__main__":
    refresh_kernel(server_list,"test_user","test_passwd")
```