2实例分析

http://www.ibm.com/developerworks/cn/linux/l-cn-pexpect2/

通过本系列第一部分《探索 Pexpect，第 1 部分：剖析 Pexpect 》（请参阅参考资料）的介绍，相信大家已经对 Pexpect 的用法已经有了比较全面的了解，知道 Pexpect 是个纯 Python 语言实现的模块，使用其可以轻松方便的实现与 ssh、ftp、passwd 和 telnet 等程序的自动交互，但是读者的理解还可能只是停留在理论基础上，本文将从实际例子入手具体介绍 Pexpect 的使用场景和使用心得体验，实例中的代码读者都可以直接拿来使用，相信会对大家产生比较大的帮助。 以下是本文所要介绍的所有 Pexpect 例子标题：

回页首

例 1：ftp 的使用

本例实现了如下功能：ftp 登录到 develperWorks.ibm.com 主机上，并用二进制传输模式下载一个名叫 rmall的文件。

清单 1. ftp 的例子代码

```
#!/usr/bin/env python

import pexpect
# 即将 ftp 所要登录的远程主机的域名
ipAddress = 'develperWorks.ibm.com'
# 登录用户名
loginName = 'root'
# 用户名密码
loginPassword = 'passw0rd'

# 拼凑 ftp 命令
cmd = 'ftp ' + ipAddress
# 利用 ftp 命令作为 spawn 类构造函数的参数，生成一个 spawn 类的对象
child = pexpect.spawn(cmd)
# 期望具有提示输入用户名的字符出现
index = child.expect(["(?i)name", "(?i)Unknown host", pexpect.EOF, pexpect.TIMEOUT])
# 匹配到了 "(?i)name"，表明接下来要输入用户名
if ( index == 0 ):
    # 发送登录用户名 + 换行符给子程序.
    child.sendline(loginName)
    # 期望 "(?i)password" 具有提示输入密码的字符出现.
    index = child.expect(["(?i)password", pexpect.EOF, pexpect.TIMEOUT])
    # 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出.
    if (index != 0):
        print "ftp login failed"
        child.close(force=True)
    # 匹配到了密码提示符，发送密码 + 换行符给子程序.
    child.sendline(loginPassword)
    # 期望登录成功后，提示符 "ftp>" 字符出现.
    index = child.expect( ['ftp>', 'Login incorrect', 'Service not available',
    pexpect.EOF, pexpect.TIMEOUT])
    # 匹配到了 'ftp>'，登录成功.
    if (index == 0):
        print 'Congratulations! ftp login correct!'
        # 发送 'bin'+ 换行符给子程序，表示接下来使用二进制模式来传输文件.
        child.sendline("bin")
        print 'getting a file...'
        # 向子程序发送下载文件 rmall 的命令.
        child.sendline("get rmall")
        # 期望下载成功后，出现 'Transfer complete.*ftp>'，其实下载成功后，
        # 会出现以下类似于以下的提示信息:
        #    200 PORT command successful.
```

```
    #   150 Opening data connection for rmall (548 bytes).
    #   226 Transfer complete.
    #   548 bytes received in 0.00019 seconds (2.8e+03 Kbytes/s)
    # 所以直接用正则表达式 '.*' 将 'Transfer complete' 和提示符 'ftp>' 之间的字符全省去.
    index = child.expect( ['Transfer complete.*ftp>', pexpect.EOF, pexpect.TIMEOUT] )
    # 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出.
    if (index != 0):
        print "failed to get the file"
        child.close(force=True)
    # 匹配到了 'Transfer complete.*ftp>'，表明下载文件成功，打印成功信息，并输入 'bye'，结束 ftp session.
    print 'successfully received the file'
    child.sendline("bye")
# 用户名或密码不对，会先出现 'Login incorrect'，然后仍会出现 'ftp>'，但是 pexpect 是最小匹配，不是贪婪匹配,
# 所以如果用户名或密码不对，会匹配到 'Login incorrect'，而不是 'ftp>'，然后程序打印提示信息并退出.
elif (index == 1):
    print "You entered an invalid login name or password. Program quits!"
    child.close(force=True)
# 匹配到了 'Service not available'，一般表明 421 Service not available, remote server has
# closed connection，程序打印提示信息并退出.
# 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出.
else:
    print "ftp login failed! index = " + index
    child.close(force=True)


# 匹配到了 "(?i)Unknown host"，表示 server 地址不对，程序打印提示信息并退出
elif index == 1 :
    print "ftp login failed, due to unknown host"
    child.close(force=True)
# 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出
else:
    print "ftp login failed, due to TIMEOUT or EOF"
    child.close(force=True)
```

注：

- 运行后，输出结果为：

```
Congratulations! ftp login correct!
getting a file...
successfully received the file
```

- 本例 expect 函数中的 pattern 使用了 List，并包含了 pexpect.EOF和pexpect.TIMEOUT，这样出现了超时或者 EOF，不会抛出 expection 。（关于 expect() 函数的具体使用，请参阅参考资料）
- 如果程序运行中间出现了错误，如用户名密码错误，超时或者 EOF，远程 server 连接不上，都会使用 c hild.close(force=True) 关掉 ftp 子程序。调用 close 可以用来关闭与子程序的 connection 连接，如果你不仅想关闭与子程序的连接，还想确保子程序是真的被 terminate 终止了，设置参数 force=True，其最终会调用 c hild.kill(signal.SIGKILL) 来杀掉子程序。

回页首

例 2：记录 log

本例实现了如下功能：运行一个命令，并将该命令的运行输出结果记录到 log 文件中 ./command.py [-a] [-c command] {logfilename} -c 后接的是要运行的命令的名字，默认是“ls -l”； logfilename 是记录命令运行结果的 log 文件名，默认是“command.log”；指定 -a 表示命令的输出结果会附加在 logfilename 后，如果 logfilename 之前已经存在的话。

清单 2. 记录 log 的例子代码

```
#!/usr/bin/env python
"""
This run a user specified command and log its result.

./command.py [-a] [-c command] {logfilename}

logfilename : This is the name of the log file. Default is command.log.
-a : Append to log file. Default is to overwrite log file.
-c : spawn command. Default is the command 'ls -l'.

Example:

This will execute the command 'pwd' and append to the log named my_session.log:

./command.py -a -c 'pwd' my_session.log

"""
import os, sys, getopt
import traceback
import pexpect

# 如果程序中间出错，打印提示信息后退出
def exit_with_usage():
    print globals()['__doc__']
    os._exit(1)

def main():
    ########################################################################
    # Parse the options, arguments, get ready, etc.
    ########################################################################
    try:
        optlist, args = getopt.getopt(sys.argv[1:], 'h?ac:', ['help','h','?'])
    # 如果指定的参数不是' -a ' , ' -h ' , ' -c ' , ' -? ' , ' --help ' ,
    # --h '或' --? '时，会抛出 exception,
```

```
    # 这里 catch 住，然后打印出 exception 的信息，并输出 usage 提示信息.
    except Exception, e:
        print str(e)
        exit_with_usage()
    options = dict(optlist)
    # 最多只能指定一个 logfile，否则出错.
    if len(args) > 1:
        exit_with_usage()
    # 如果指定的是 '-h','--h','-?','--?' 或 '--help'，只输出 usage 提示信息.
    if [elem for elem in options if elem in ['-h','--h','-?','--?','--help']]:
        print "Help:"
        exit_with_usage()
    # 获取 logfile 的名字.
    if len(args) == 1:
        script_filename = args[0]
    else:
    # 如果用户没指定，默认 logfile 的名字是 command.log
        script_filename = "command.log"
    # 如果用户指定了参数 -a，如果之前该 logfile 存在，那么接下来的内容会附加在原先内容之后,
    # 如果之前没有该 logfile，新建一个文件，并且接下来将内容写入到该文件中.
    if '-a' in options:
        fout = open (script_filename, "ab")
    else:
    # 如果用户没指定参数 -a，默认按照用户指定 logfile 文件名新建一个文件，然后将接下来将内容写入到该文件中.
        fout = open (script_filename, "wb")
    # 如果用户指定了 -c 参数，那么运行用户指定的命令.
    if '-c' in options:
        command = options['-c']
    # 如果用户没有指定 -c 参数，那么默认运行命令'ls – l'
    else:
        command = "ls -l"

    # logfile 文件的 title
    fout.write ('==========Log Tile: IBM developerWorks China==========\n')

    # 为接下来的运行命令生成一个 pexpect 的 spawn 类子程序的对象.
    p = pexpect.spawn(command)
    # 将之前 open 的 file 对象指定为 spawn 类子程序对象的 log 文件.
    p.logfile = fout
    # 命令运行完后，expect EOF 出现，这时会将 spawn 类子程序对象的输出写入到 log 文件.
    p.expect(pexpect.EOF)
    #open 完文件，使用完毕后，需关闭该文件.
    fout.close()
    return 0

if __name__ == "__main__":
    try:
        main()
    except SystemExit, e:
        raise e
    except Exception, e:
        print "ERROR"
        print str(e)
        traceback.print_exc()
        os._exit(1)
```

注：

- 运行：`./command.py -a -c who cmd.log`

```
运行结束后，cmd.log 的内容为：
IBM developerWorks China
Root   :0    2009-05-12 22:40
Root   pts/1    2009-05-12 22:40 (:0.0)
Root   pts/2    2009-07-05 18:55 (9.77.180.94)
```

- **logfile：**

只能通过 spawn 类的构造函数指定。在 spawn 类的构造函数通过参数指定 logfile 时，表示开启或关闭 logging 。所有的子程序的 input 和 output 都会被 copy 到指定的 logfile 中。设置 logfile 为 None 表示停止 logging，默认就是停止 logging 。设置 logfile 为 sys.stdout，会将所有东西 echo 到标准输出。

- **logfile_read和logfile_send：**

logfile_read：只用来记录 python 主程序接收到 child 子程序的输出，有的时候你不想看到写给 child 的所有东西，只希望看到 child 发回来的东西。 logfile_send：只用来记录 python 主程序发送给 child 子程序的输入 logfile、logfile_read 和 logfile_send 何时被写入呢？ logfile、logfile_read 和 logfile_send 会在每次写 write 和 send 操作后被 flush 。

  - 调用 `send` 后，才会往 `logfile` 和 `logfile_send` 中写入，`sendline/sendcontrol/sendoff/write/writeline` 最终都会调用 `send`，所以 `sendline` 后 `logfile` 中一定有内容了，只要此时 `logfile` 没有被 `close` 。
  - 调用 `read_nonblocking` 后，才会往 `logfile` 和 `logfile_read` 中写入，`expect_loop` 会调用 `read_nonblocking`，而 `expect_exact` 和 `expect_list` 都会调用 `expect_loop`，`expect` 会调用 `expect_list`，所以 `expect` 后 `logfile` 中一定有内容了，只要此时 `logfile` 没有被 `close` 。

如果调用的函数最终都没有调用 send 或 read_nonblocking，那么 logfile 虽然被分配指定了一个 file，但其最终结果是：内容为空。见下例：

清单 3. log 内容为空的例子代码

```
import pexpect
p = pexpect.spawn( ' ls -l ' )
fout = open ('log.txt', "w")
p.logfile = fout
fout.close()
```

运行该脚本后，你会发现其实 log.txt 是空的，没有记录 ls -l 命令的内容，原因是没有调用 send 或 read_nonblocking，真正的内容没有被 flush 到 log 中。如果在 fout.close() 之前加上 p.expect(pexpect.EOF)，log.txt 才会有 ls -l 命令的内容。

回页首

例 3：ssh 的使用

本例实现了如下功能：ssh 登录到某个用户指定的主机上，运行某个用户指定的命令，并输出该命令的结果。

清单 4. ssh 的例子代码

```
#!/usr/bin/env python

"""
This runs a command on a remote host using SSH. At the prompts enter hostname,
user, password and the command.
"""

import pexpect
import getpass, os

#user: ssh 主机的用户名
#host：ssh 主机的域名
#password：ssh 主机的密码
#command：即将在远端 ssh 主机上运行的命令
def ssh_command (user, host, password, command):
    """
    This runs a command on the remote host. This could also be done with the
    pxssh class, but this demonstrates what that class does at a simpler level.
    This returns a pexpect.spawn object. This handles the case when you try to
    connect to a new host and ssh asks you if you want to accept the public key
    fingerprint and continue connecting.
    """
    ssh_newkey = 'Are you sure you want to continue connecting'
    # 为 ssh 命令生成一个 spawn 类的子程序对象.
    child = pexpect.spawn('ssh -l %s %s %s'%(user, host, command))
    i = child.expect([pexpect.TIMEOUT, ssh_newkey, 'password: '])
    # 如果登录超时，打印出错信息，并退出.
    if i == 0: # Timeout
        print 'ERROR!'
        print 'SSH could not login. Here is what SSH said:'
        print child.before, child.after
        return None
    # 如果 ssh 没有 public key，接受它.
    if i == 1: # SSH does not have the public key. Just accept it.
        child.sendline ('yes')
        child.expect ('password: ')
        i = child.expect([pexpect.TIMEOUT, 'password: '])
        if i == 0: # Timeout
        print 'ERROR!'
        print 'SSH could not login. Here is what SSH said:'
        print child.before, child.after
        return None
    # 输入密码.
    child.sendline(password)
    return child

def main ():
    # 获得用户指定 ssh 主机域名.
    host = raw_input('Hostname: ')
    # 获得用户指定 ssh 主机用户名.
    user = raw_input('User: ')
    # 获得用户指定 ssh 主机密码.
    password = getpass.getpass()
    # 获得用户指定 ssh 主机上即将运行的命令.
    command = raw_input('Enter the command: ')
    child = ssh_command (user, host, password, command)
    # 匹配 pexpect.EOF
    child.expect(pexpect.EOF)
    # 输出命令结果.
    print child.before

if __name__ == '__main__':
    try:
        main()
    except Exception, e:
        print str(e)
        traceback.print_exc()
        os._exit(1)
```

注：

- 运行后，输出结果为：

```
Hostname: developerWorks.ibm.com
User: root
Password:
Enter the command: ls -l
```

```
total 60
drwxr-xr-x   2 root   system   512 Jun 14 2006  .dt
drwxrwxr-x   3 root   system   512 Sep 23 2008  .java
-rwx------   1 root   system   1855 Jun 14 2006  .kshrc
-rwx------   1 root   system   806 Sep 16 2008  .profile
-rwx------   1 root   system   60 Jun 14 2006  .rhosts
drwx------   2 root   system   512 Jan 18 2007  .ssh
drwxr-x---   2 root   system   512 Apr 15 00:04 223002
-rwxr-xr-x   1 root   system   120 Jan 16 2007  drcron.sh
-rwx------   1 root   system   10419 Jun 14 2006  firewall
drwxr-x---   2 root   system   512 Oct 25 2007  jre
-rw-------   1 root   system   3203 Apr 04 2008  mbox
-rw-r--r--   1 root   system   0 Jun 14 2006  pt1
-rw-r--r--   1 root   system   0 Jun 14 2006  pt2
```

- 使用了 getpass.getpass() 来获得用户输入的密码，与 raw_input 不同的是，getpass.getpass() 不会将用户输入的密码字符串 echo 回显到 stdout 上。（更多 python 相关技术，请参阅参考资料）

回页首

例 4：pxssh 的使用

本例实现了如下功能：使用 pexpect 自带的 pxssh 模块实现 ssh 登录到某个用户指定的主机上，运行命令’ uptime ’和’ ls -l ’，并输出该命令的结果。

清单 5. 使用 pxssh 的例子代码

```
#!/usr/bin/env python
import pxssh
import getpass
try:
    # 调用构造函数，创建一个 pxssh 类的对象.
    s = pxssh.pxssh()
    # 获得用户指定 ssh 主机域名.
    hostname = raw_input('hostname: ')
    # 获得用户指定 ssh 主机用户名.
    username = raw_input('username: ')
    # 获得用户指定 ssh 主机密码.
    password = getpass.getpass('password: ')
    # 利用 pxssh 类的 login 方法进行 ssh 登录，原始 prompt 为'$' , '#或'>'
    s.login (hostname, username, password, original_prompt='[$#>]')
    # 发送命令 'uptime'
    s.sendline ('uptime')
    # 匹配 prompt
    s.prompt()
    # 将 prompt 前所有内容打印出，即命令 'uptime' 的执行结果.
    print s.before
    # 发送命令 ' ls -l '
    s.sendline ('ls -l')
    # 匹配 prompt
    s.prompt()
    # 将 prompt 前所有内容打印出，即命令 ' ls -l ' 的执行结果.
    print s.before
    # 退出 ssh session
    s.logout()
except pxssh.ExceptionPxssh, e:
    print "pxssh failed on login."
    print str(e)
```

- 运行后，输出结果为：

```
hostname: develperWorks.ibm.com
username: root
password:
uptime
02:19AM   up 292 days,  12:16,  2 users,  load average: 0.01, 0.02, 0.01

ls -l
total 60
drwxr-xr-x   2 root   system   512 Jun 14 2006  .dt
drwxrwxr-x   3 root   system   512 Sep 23 2008  .java
-rwx------   1 root   system   1855 Jun 14 2006  .kshrc
-rwx------   1 root   system   806 Sep 16 2008  .profile
-rwx------   1 root   system   60 Jun 14 2006  .rhosts
drwx------   2 root   system   512 Jan 18 2007  .ssh
drwxr-x---   2 root   system   512 Apr 15 00:04 223002
-rwxr-xr-x   1 root   system   120 Jan 16 2007  drcron.sh
-rwx------   1 root   system   10419 Jun 14 2006  firewall
drwxr-x---   2 root   system   512 Oct 25 2007  jre
-rw-------   1 root   system   3203 Apr 04 2008  mbox
-rw-r--r--   1 root   system   0 Jun 14 2006  pt1
-rw-r--r--   1 root   system   0 Jun 14 2006  pt2
```

- pxssh 是 pexpect 中 spawn 类的子类，增加了 login, logout 和 prompt 几个方法，使用其可以轻松实现 ssh 连接，而不用自己调用相对复杂的 pexpect 的方法来实现。 pxssh 做了很多 tricky 的东西来处理 ssh login 过程中所可能遇到的各种情况。比如：如果这个 session 是第一次 login，pxssh 会自动接受远程整数 remote certificate ；如果你已经设置了公钥认证，pxssh 将不会再等待 password 的提示符。（更多 ssh 相关知识，请参阅参考资料） pxssh 使用 shell 的提示符来同步远程主机的输出，为了使程序更加稳定，pxssh 还可以设置 prompt 为更加唯一的字符串，而不仅仅是“ $ ”和“ # ”。
- **login 方法**

```
login (self,server,username,password='',terminal_type='ansi',
    iginal_prompt=r"[#$]",login_timeout=10,port=None,auto_prompt_reset=True):
```

使用原始 original_prompt 来找到 login 后的提示符（这里默认 original_prompt 是“$”或“#”，但是有时候可能也是别的 prompt，这时就需要在 login 时手动指定这个特殊的 prompt，见上例，有可能是“ > ”），如果找到了，立马使用更容易匹配的字符串来重置该原始提示符（这是由 pxssh 自己自动做的，通过命令 "PS1='[PEXPECT]\$ '" 重置原始提示符，然后每次 expect 匹配 \[PEXPECT\][\$\#]）。原始提示符是很容易被混淆和胡弄的，为了阻止错误匹配，最好根据特定的系统，指定更加精确的原始提示符，例如 "Message Of The Day" 。 有些情况是不允许重置原始提示符的，这时就要设置 auto_prompt_reset 为 False 。而且此时需要手动设置 PROMPT 域为某个正则表达式来 match 接下来要出现的新提示符，因为 prompt() 函数默认是 expect 被重置过的 PROMPT 的。

- **prompt方法**

prompt (self, timeout=20):
匹配新提示符（不是 original_prompt）。注：这只是匹配提示符，不能匹配别的 string，如果要匹配特殊 string，需直接使用父类 spawn 的 expect 方法。 prompt 方法相当于是 expect 方法的一个快捷方法。如果auto_prompt_reset 为 False，这时需要手动设置 PROMPT 域为某个正则表达式来 match 接下来要出现的 prompt，因为 prompt() 函数默认是 expect 被重置过的 PROMPT 的。

- **logout方法**

logout (self):
发送'exit'给远程 ssh 主机，如果有 stopped jobs，会发送'exit'两次。

回页首

例 5：telnet 的使用

本例实现了如下功能：telnet 登录到某远程主机上，输入命令“ls -l”后，将子程序的执行权交还给用户，用户可以与生成的 telnet 子程序进行交互。

清单 6. telnet 的例子代码

```
#!/usr/bin/env python
import pexpect

# 即将 telnet 所要登录的远程主机的域名
ipAddress = 'develperWorks.ibm.com'
# 登录用户名
loginName = 'root'
# 用户名密码
loginPassword = 'passw0rd'
# 提示符，可能是' $ ' , ' # '或' > '
loginprompt = '[$#>]'

# 拼凑 telnet 命令
cmd = 'telnet ' + ipAddress
# 为 telnet 生成 spawn 类子程序
child = pexpect.spawn(cmd)
# 期待'login'字符串出现，从而接下来可以输入用户名
index = child.expect(["login", "(?i)Unknown host", pexpect.EOF, pexpect.TIMEOUT])
if ( index == 0 ):
    # 匹配'login'字符串成功，输入用户名.
    child.sendline(loginName)
    # 期待 "[pP]assword" 出现.
    index = child.expect(["[pP]assword", pexpect.EOF, pexpect.TIMEOUT])
    # 匹配 "[pP]assword" 字符串成功，输入密码.
    child.sendline(loginPassword)
    # 期待提示符出现.
    child.expect(loginprompt)
    if (index == 0):
        # 匹配提示符成功，输入执行命令 'ls -l'
        child.sendline('ls -l')
        # 立马匹配 'ls -l'，目的是为了清除刚刚被 echo 回显的命令.
        child.expect('ls -l')
        # 期待提示符出现.
        child.expect(loginprompt)
        # 将 'ls -l' 的命令结果输出.
        print child.before
        print "Script recording started. Type ^] (ASCII 29) to escape from the script
            shell."
        # 将 telnet 子程序的执行权交给用户.
        child.interact()
        print 'Left interactve mode.'
    else:
        # 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出.
        print "telnet login failed, due to TIMEOUT or EOF"
        child.close(force=True)
else:
    # 匹配到了 pexpect.EOF 或 pexpect.TIMEOUT，表示超时或者 EOF，程序打印提示信息并退出.
    print "telnet login failed, due to TIMEOUT or EOF"
    child.close(force=True)
```

- 运行后，输出结果为：

```
total 60
drwxr-xr-x   2 root     system          512 Jun 14 2006  .dt
drwxrwxr-x   3 root     system          512 Sep 23 2008  .java
-rwx------   1 root     system         1855 Jun 14 2006  .kshrc
-rwx------   1 root     system          806 Sep 16 2008  .profile
-rwx------   1 root     system           60 Jun 14 2006  .rhosts
drwx------   2 root     system          512 Jan 18 2007  .ssh
drwxr-x---   2 root     system          512 Apr 15 00:04 223002
```

```
-rwxr-xr-x   1 root     system          120 Jan 16 2007  drcron.sh
-rwx------   1 root     system        10419 Jun 14 2006  firewall
drwxr-x---   2 root     system          512 Oct 25 2007  jre
-rw-------   1 root     system         3203 Apr 04 2008  mbox
-rw-r--r--   1 root     system            0 Jun 14 2006  pt1
-rw-r--r--   1 root     system            0 Jun 14 2006  pt2
essni2
Script recording started. Type ^] (ASCII 29) to escape from the script shell.
此时程序会 block 住，等待用户的输入，比如用户输入' pwd '，输出/home/root
接下来用户敲入 ctrl+] 结束子程序
```

- **interact方法**

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

通常一个 python 主程序通过 pexpect.spawn 启动一个子程序，一旦该子程序启动后，python 主程序就可以通过 child.expect 和 child.send/child.sendline 来和子程序通话，python 主程序运行结束后，子程序也就死了。比如 python 主程序通过 pexpect.spawn 启动了一个 telnet 子程序，在进行完一系列的 telnet 上的命令操作后，python 主程序运行结束了，那么该 telnet session（telnet 子程序）也会自动退出。但是如果调用 child.interact，那么该子程序（python 主程序通过 pexpect.spawn 衍生成的）就可以在运行到 child.interact 时，将子程序的控制权交给了终端用户（the human at the keyboard），用户可以通过键盘的输入来和子程序进行命令交互，管理子程序的生杀大权，用户的键盘输入 stdin 会被传给子程序，而且子程序的 stdout 和 stderr 输出也会被打印出来到终端。 默认 ctrl +] 退出 interact() 模式，把子程序的执行权重新交给 python 主程序。参数 escape_character 指定了交互模式的退出字符，例如 child.interact(chr(26)) 接下来就会变成 ctrl + z 退出 interact() 模式。

回页首

pexpect 使用 tips

调试 pexpect 程序的 tips

- 获得 `pexpect.spawn` 对象的字符串 `value`值，将会给 `debug` 提供很多有用信息。

清单 7. 打印 pexpect.spawn 对象的字符串 value 值的例子代码

```
try:
    i = child.expect ([pattern1, pattern2, pattern3, etc])
except:
    print "Exception was thrown"
    print "debug information:"
    print str(child)
```

- 将子程序的 `input` 和 `output` 打 `log` 到文件中或者直接打 `log` 到屏幕上也非常有用

清单 8. 记录 log 的例子代码

```
# 打开 loggging 功能并将结果输出到屏幕上
child = pexpect.spawn (foo)
child.logfile = sys.stdout
```

pexpect 不会解释 shell 中的元字符

- `pexpect` 不会解释 `shell` 的元字符，如重定向 `redirect`，管道 `pipe`，和通配符 `wildcards`(" > " , " | "和" * "等) 如果想用的话，必须得重新启动一个新 `shell`（在 `spawn` 的参数 `command` 中是不会解释他们的，视其为 `command string` 的一个普通字符）

清单 9. 重新启动一个 shell 来规避 pexpect 对元字符的不解释

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

如果想在 spawn 出来的新子程序中使用重定向 redirect，管道 pipe，和通配符 wildcards(" > " , " | "和" * "等)，好像没有好的方法，只能不使用这些字符，先利用 expect 匹配命令提示符，从而在 before 中可以拿到之前命令的结果，然后在分析 before 的内容达到使用重定向 redirect, 管道 pipe, 和通配符 wildcards 的目的。

EOF 异常和 TIMEOUT 异常

- TIMEOUT 异常

如果子程序没有在指定的时间内生成任何 output，那么 expect() 和 read() 都会产生 TIMEOUT 异常。超时默认是 30s，可以在 expect() 和 spawn 构造函数初始化时指定为其它时间，如：

```
child.expect('password:', timeout=120) # 等待 120s
```

如果你想让 expect() 和 read() 忽略超时限制，即无限期阻塞住直到有 output 产生，设置 timeout 参数为 None。

清单 10. 忽略 timeout 超时限制的例子代码

```
child = pexpect.spawn( "telnet developerWorks.ibm.com" )
child.expect( "login", timeout=None )
```

- EOF 异常

可能会有两种 EOF 异常被抛出，但是他们除了显示的信息不同，其实本质上是相同的。为了实用的目的，不需要区分它们，他们只是给了些关于你的 python 程序到底运行在哪个平台上的额外信息，这两个显示信息是：

```
End Of File (EOF) in read(). Exception style platform.
End Of File (EOF) in read(). Empty string style platform.
```

有些 UNIX 平台，当你读取一个处于 EOF 状态的文件描述符时，会抛出异常，其他 UNIX 平台，却只会静静地返回一个空字符串来表明该文件已经达到了状态。

使用 run() 来替代某些的 spawn 的使用

pexpect 模块除了提供 spawn 类以外，还提供了 run() 函数，使用其可以取代一些 spawn 的使用，而且更加简单明了。

清单 11. 使用 run() 来替代 spawn 的使用的例子代码

```
# 使用 spawn 的例子
from pexpect import *
child = spawn('scp foo myname@host.example.com:.')
child.expect ('(?i)password')
child.sendline (mypassword)
# 以上功能，相当于以下 run 函数：
from pexpect import *
run ('scp foo myname@host.example.com:.', events={'(?i)password': mypassword})
```

- run (command, timeout=-1, withexitstatus=False, events=None, extra_args=None, logfile=None, cwd=None, env=None):
  - command：执行一个命令，然后返回结果，run() 可以替换 os.system()（更多 os.system() 知识，请参阅参考资料），因为 os.system() 得不到命令输出的结果
  - 返回的 output 是个字符串，STDERR 也会包括在 output 中，如果全路径没有被指定，那么 path 会被 search
  - timeout：单位 s 秒，每隔 timeout 生成一个 pexpect.TIMEOUT 异常
  - 每行之间被 CR/LF (\\r\\n) 相隔，即使在 Unix 平台上也是 CR/LF，因为 Pexpect 子程序是伪 tty 设备
  - withexitstatus：设置为 True，则返回一个 tuple，里面包括 (command_output, exitstatus)，如果其为 False，那么只是仅仅返回 command_output
  - events：是个 dictionary，里面存放 {pattern:response} 。无论什么时候 pattern 在命令的结果中出现了，会出现以下动作：
    - 发送相应的 response String 。如果需要回车符“ Enter ”的话，“ \\n ”也必须得出现在 response 字符串中。
    - response 同样也可以是个回调函数，不过该回调函数有特殊要求，即它的参数必须是个 dictionary，该 dictionary 的内容是：包含所有在 run() 中定义的局部变量，从而提供了方法可以访问 run() 函数中 spawn 生成的子程序和 run() 中定义的其他局部变量，其中 event_count, child, 和 extra_args 最有用。回调函数可能返回 True，从而阻止当前 run() 继续执行，否则 run() 会继续执行直到下一个 event 。回调函数也可能返回一个字符串，然后被发送给子程序。 'extra_args' 不是直接被 run() 使用，它只是提供了一个方法可以通过 run() 来将数据传入到回调函数中（其实是通过 run() 定义的局部变量 dictionary 来传）

清单 12. 其它一些使用 run() 的例子代码

```
# 在 local 机器上启动 apache 的 daemon
from pexpect import *
run ("/usr/local/apache/bin/apachectl start")
# 使用 SVN check in 文件
from pexpect import *
run ("svn ci -m 'automatic commit' my_file.py")
# 运行一个命令并捕获 exit status
from pexpect import *
command_output, exitstatus) = run ('ls -l /bin', withexitstatus=1)
# 运行 SSH，并在远程机器上执行' ls -l '，如果 pattern '(?i)password' 被匹配住，密码 'secret'
# 将会被发送出去
run ("ssh username@machine.example.com 'ls -l'", events={'(?i)password':'secret\\n'})
# 启动 mencoder 来 rip 一个 video，同样每 5s 钟显示进度记号
from pexpect import *
def print_ticks(d):
    print d['event_count']
run ("mencoder dvd://1 -o video.avi -oac copy -ovc copy", events={TIMEOUT:print_ticks})
```

expect_exact() 的使用

expect_exact(self, pattern_list, timeout = -1, searchwindowsize = -1); expect_exact() 与 expect() 类似，但是 pattern_list 只能是字符串或者是一个字符串的 list，不能是正则表达式，其匹配速度会快于 expect()，原因有两个：一是字符串的 search 比正则表达式

的匹配要快，另一个则是可以限制只从输入缓冲的结尾来寻找匹配的字符串。还有当你觉得每次要 escape 正则表达式中的特殊字符为普通字符时很烦，那么你也可以使用 expect_exact() 来取代 expect()。

清单 13. expect_exact() 的例子代码

```
import pexpect
child = pexpect.spawn('ls -l')
child.expect_exact('pexpect.txt')
print child.after
```

expect() 中正则表达式的使用 tips

expect() 中的正则表达式不是贪婪匹配 greedy match，而是最小匹配，即只匹配缓冲区中最早出现的第一个字符串。 因为是依次读取一个字符的 stream 流来判断是否和正则表达式所表达的模式匹配，所以如果参数 pattern 是个 list，而且不止一次匹配，那么缓冲区中最早出现的第一个匹配的字符串才算数。

清单 14. expect() 的最小匹配例子代码

```
# 如果输入是 'foobar'
index = p.expect (['bar', 'foo', 'foobar'])
#index 返回是 1 ('foo') 而不是 2 ('foobar')，即使 'foobar' 是个更好的匹配。原因是输入是个流 stream，
# 当收到 foo 时，第 1 个 pattern ('foo') 就被匹配了，不会等到' bar '的出现了，所以返回 1
```

- “$”不起任何作用，匹配一行的结束 (end of line)，必须得匹配 CR/LF

正则表达式中，'$'可以匹配一行的结束（具体'$'正则表达式的使用，请参阅参考资料），但是 pexpect 从子程序中一次只读取一个字符，而且每个字符都好像是一行的结束一样，pexpect 不能在子程序的输出流去预测。匹配一行结束的方法必须是匹配 "\r\n" (CR/LF) 。即使是 Unix 系统，也是匹配 "\r\n" (CR/LF)，因为 pexpect 使用一个 Pseudo-TTY 设备与子程序通话，所以当子程序输出 "\n" 你仍然会在 python 主程序中看到 "\r\n" 。原因是 TTY 设备更像 windows 操作系统，每一行结束都有个 "\r\n" (CR/LF) 的组合，当你从 TTY 设备去解释一个 Unix 的命令时，你会发现真正的输出是 "\r\n" (CR/LF)，一个 Unix 命令只会写入一个 linefeed (\n)，但是 TTY 设备驱动会将其转换成 "\r\n" (CR/LF) 。

清单 15. 匹配一行结束 1

```
child.expect ('\r\n')
```

如果你只是想跳过一个新行，直接 expect('\n') 就可以了，但是如果你想在一行的结束匹配一个具体的 pattern 时，就必须精确的寻找 (\r)，见下例：

清单 16. 匹配一行结束 2

```
# 成功在一行结束前匹配一个单词
child.expect ('\w+\r\n')
# 以下两种情况都会失败
child.expect ('\w+\n')
child.expect ('\w+$')
```

这个问题其实不只是 pexpect 会有，如果你在一个 stream 流上实施正则表达式匹配时，都会遇到此问题。正则表达式需要预测，stream 流中很难预测，因为生成这个流的进程可能还没有结束，所以你很难知道是否该进程是暂时性的暂停还是已经彻底结束。

- 当 `'.'` 和 `'*'` 出现在最后时

child.expect ('.+'); 因为是最小匹配，所以只会返回一个字符，而不是一个整个一行（虽然 pexpect 设置了 re.DOTALL，会匹配一个新行。 child.expect ('.*'); 每次匹配都会成功，但是总是没有字符返回，因为 '*' 表明前面的字符可以出现 0 次 , 在 pexpect 中，一般来说，任何 '*' 都会尽量少的匹配。

isalive() 的使用 tips

- `isalive(self)`

测试子程序是否还在运行。这个方法是非阻塞的，如果子程序被终止了，那么该方法会去读取子程序的 exitstatus 或 signalstatus 这两个域。返回 True 表明子程序好像是在运行，返回 False 表示不再运行。当平台是 Solaris 时，可能需要几秒钟才能得到正确的状态。 当子程序退出后立马执行 isalive() 有时可能会返回 1 (True)，这是一个 race condition，原因是子程序已经关闭了其文件描述符，但是在 isalive() 执行前还没有完全的退出。增加一个小小的延时会对 isalive() 的结果有效性有帮助。

清单 17. isalive() 的例子代码

```
# 以下程序有时会返回 1 (True)
child = pexpect.spawn('ls')
child.expect(pexpect.EOF)
print child.isalive()
# 但是如果在 isalive() 之前加个小延时，就会一直返回 0 (False)
child = pexpect.spawn('ls')
child.expect(pexpect.EOF)
time.sleep(0.1)    # 之前要 import time，单位是秒 s
print child.isalive()
```

delaybeforesend 的使用 tips

spawn 类的域 delaybeforesend 可以帮助克服一些古怪的行为。比如，经典的是，当一个用户使用 expect() 期待 "Password:" 提示符时，如果匹配，立马 sendline() 发送密码给子程序，但是这个用户会看到他们的密码被 echo back 回显回来了。这是因为，通常许多应用程序都会在打印出 "Password:" 提示符后，立马关掉 stdin 的 echo，但是如果你发送密码过快，在程序关掉 stdin 的 echo

之前就发送密码出去了，那么该密码就会被 echo 出来。

清单 18. delaybeforesend 的例子代码

```
child.expect ('[pP]assword:')
child.sendline (my_password)
# 在 expect 之后，某些应用程序，如 SSH，会做如下动作：
#1. SSH 打印 "password:" 提示符给用户
#2. SSH 关闭 echo.
#3. SSH 等待用户输入密码
# 但是现在第二条语句 sendline 可能会发生在 1 和 2 之间，即在 SSH 关掉 echo 之前输入了 password 给子程序，从
# 而在 stdout，该 password 被 echo 回显出来，出现了 security 的问题
# 所以此时可以通过设置 delaybeforesend 来在将数据写（发送）给子程序之前增加一点点的小延时，因为该问题经
# 常出现，所以默认就 sleep 50ms. 许多 linux 机器必须需要 0.03s 以上的 delay
self.delaybeforesend = 0.05 # 单位秒
```