

Netmiko 4 新功能使用手册

 原文链接: <https://zhuanlan.zhihu.com/...>

 收藏时间: 2025 年 05 月 11 日

今年（2022 年）3 月 23 日，在 Netmiko 3 的最后一版 Netmiko 3.4.0 发布整整 11 个月后，“Netmiko 之父” [Kirk Byers](#) 正式公布并发行了 Netmiko 4 的初版即 Netmiko 4.0.0，和很多模块的重大更新类似，Netmiko 4.0.0 发布后存在众多 bug，尤其是华为、H3C 等很多国产设备受影响十分严重，很多用户通过 `pip install` 安装 Netmiko 的时候不得不手动指定版本，用回稳定的 3.4.0。不过 Kirk 行动迅速，在 4 月 27 号，6 月 29 号以及上星期 8 月 9 号相继发布了 4.1.0、4.1.1 和 4.1.2 版本，不仅优化了 Netmiko 4 的性能，解决了 4.0.0 众多的 bug，也提供了对更多设备的支持。

1. read_timeout 和 read_timeout_override

Netmiko 的作者 Kirk Byers 曾坦诚，Netmiko 4 诞生之前的所有版本中最困扰他的就是截屏（screen-scraping）问题。所谓截屏问题是指在用户向设备输入一条 `show` 或者 `display` 命令后，Netmiko 无法判断回显内容是否已经完整返回，其实不仅是 Netmiko，任何需要用到截屏功能的模块都会遇到类似的问题，比如 [Paramiko](#) 需要我们用 `time.sleep()` 手动指定休眠时间，完全把问题抛回给用户自己解决。

Netmiko 是 Paramiko 的衍生版本，它帮我们省去了手动指定休眠时间的这一步骤。在 Netmiko4 之前，Netmiko 会用 `delay_factor` 参数（在 `send_command()` 中使用，只对 `send_command()` 里使用的命令有效）或 `global_delay_factor` 参数（在设定登录设备参数的字典中使用，对在该设备上输入的所有命令都有效）配合 `expect_string` 参数来处理这个问题。比如说通过 `send_command()` 在思科交换机里输入 `show ip int brief` 命令，我们可以通过 `expect_string=r'#'` 告诉 Netmiko 去回显内容中抓取最后一个字符，即思科设备的命令提示符 `#`（如果没有指定 `expect_string` 并且 `device_type` 为 `cisco_ios`，则 `expect_string` 的默认值为 `#`），如下图所示：

Python

```
1 device = {
2     "device_type": "hp_comware",
3     "host": "11.0.0.2",
4     "username": "netops",
5     "password": "Admin@1234"
6     "fast_cli": True; #device_type是cisco_ios时, fast_cli=True, 其他设备类型
    时, fast_cli=False
7     "global_delay_factor": 0.1; #延时因子, 默认为1, fast_cli=True时, global_de
    lay_factor=0.1
8 }
```

Python

```
1 with ConnectHandler(**device) as net_connect:
2     output = net_connect.send_command("show ip interface brief",
3                                     expect_string=r"#", #device_type是
    cisco_ios时, expect_string=r"#")
4                                     delay_factor=0.1) #默认值为1, fast_
    cli=True时, delay_factor=0.1
5     print(output)
```

在向设备输入一条命令后，Netmiko 默认**最多等待 100 秒**来从回显内容中抓取 `expect_string` 指定的字符，如果 `fast_cli=True`，则 Netmiko **最多只等待 10 秒钟**，这 10 秒怎么来的？也就是 **100（默认等待时间）乘以 0.1（fast_cli=True 时，delay_factor 和 global_delay_factor 的值）得来的**。如果抓取到了 `expect_string` 指定的回显字符，则 Netmiko 会立即返回回显内容（**注意是立即返回，不是非要等到第 10 秒或第 100 秒才返回**），如果过了 10 秒或 100 秒都还没抓取到 #，则 Netmiko 会返回一个异常。

一般来说 `show ip interface brief` 的回显内容肯定是在 10 秒钟内顺利返回的，但是 `show run` 则不一定，`show tech-support` 就更不可能了，这个时候我们必须手动修改 `fast_cli`，`delay_factor` 或者 `global_delay_factor` 几个参数来调整 Netmiko 等待回显内容的时间。

为了简化这个步骤，Netmiko 4 中特意在 `send_command()` 中引入了 `read_timeout` 参数，`read_timeout` 参数可以让我们直接指定 Netmiko 最多等待多少秒来从回显内容中抓取 `expect_string` 指定的字符，比如说在生产网络里一台思科的 6800 三层交换机上输入 `show tech-support` 后要等 45-50 秒以上才能返回完整的回显内容，则我们可以把 `read_timeout` 参数设为 60，如下图所示：

Python

```
1 with ConnectHandler(**device) as net_connect:
2     output = net_connect.send_command("show tech-support",
3                                     read_timeout=60) # 让netmiko最多等
    待60秒。
```

可以看到 `read_timeout` 最大的好处是大大节省了新手的学习成本（`read_timeout` 默认值为 **10秒**），不用再去学习什么 `fast_cli`, `delay_factor` 和 `global_delay_factor` 这些参数，并且也免去了额外做乘法运算的麻烦。和 `global_delay_factor` 类似，我们也可以在设定登录设备参数的字典中使用 `read_timeout_override` 来**全局修改** `read_timeout` 的值，如下图所示：

Python

```
1 device02 = {
2     "device_type": "cisco_ios",
3     "host": "11.0.0.2",
4     "username": "netops",
5     "password": "Admin@1234",
6     "read_timeout_override": 90; #全局有效
7 }
```

另外，出于兼容性考虑，还可以在字典里将 `delay_factor_compact` 设为 `True`，这样 Netmiko 4 会按照 Netmiko 3 的模式继续使用 `fast_cli`, `delay_factor` 和 `global_delay_factor` 来计算等待时间，如下图所示。

Python

```
1 device02 = {
2     "device_type": "cisco_ios",
3     "host": "11.0.0.2",
4     "username": "netops",
5     "password": "Admin@1234",
6     "read_timeout_override": 90, #全局有效
7     "delay_factor_compact": True # 默认为False，不兼容Netmiko之前的版本
8 }
```

2. `send_multiline()` 和 `send_multiline_timing()`

在《网络工程师的 Python 之路 -- Netmiko 终极指南》实验 8 里曾经提到过如何处理设备交互命令的场景，比如在思科交换机上输入 `del flash0:/test.txt` 这个删除 flash: 下文件的命令后，系统会询问你是否 confirm，如下图所示。

Shell

```
1 Switch#del flash0:/test.txt
2 Delete filename [test.txt]?
3 Delete flash0:/test.txt? [confirm]n
4 Delete of flash0:/test.txt aborted!
5 Switch#
```

或者使用 extended ping 模式后，系统让你输入一系列的和 ping 相关的参数，如下图所示。

Shell

```
1 Switch#ping
2 Protocol [ip]:
3 Target IP address: 192.168.12.2
4 Repeat count [5]:
5 Datagram size [100]:
6 Timeout in seconds [2]:
7 Extended commands [n]:
8 Sweep range of sizes [n]:
9 Type escape sequence to abort.
10 Sending 5, 100-byte ICMP Echos to 192.168.12.2, timeout is 2 seconds:
11 .....
12 Success rate is 0 percent (0/5)
13 Switch#
```

在《网络工程师的 Python 之路 -- Netmiko 终极指南》实验 8 中我们是通过 Netmiko 3 中的 `send_command()` 配合 `expect_string` 来应对这个问题。但是这种做法非常复杂且要写大量代码完成，比如说应对第一个 `del flash0:/test.txt` 的场景时，因为该交互场景只需要我们输入一个参数（是否 confirm），所以代码相对还比较简洁，如下图所示。

Shell

```
1 with ConnectHandler(**Switch01) as connect:
2     print("已经成功登录交换机"+Switch01['host'])
3
4     output = connect.send_command(command_string="delete flash0:/text.txt",
5                                     expect_string=r"Delete flash:/text.txt?",
6                                     strip_prompt=False,
7                                     strip_command=False)
8     output += connect.send_command(command_string="y",
9                                     expect_string=r"#",
10                                    strip_prompt=False,
11                                    strip_command=False)
12 print(output)
```

但是遇到 extended ping 模式这种需要用户输入多个参数的交互场景时，代码量就非常恐怖了，如下图所示。