



Python API

- [Python API の概要 \(1 ページ\)](#)
- [Python の使用 \(2 ページ\)](#)

Python API の概要

Cisco NX-OS リリース 9.3(5) 以降、Python 3 はサポート対象です：Python 2.7 は引き続きサポートされます。新しいスクリプトには、**python3** コマンドを使用することをお勧めします。

Cisco Nexus 3500 プラットフォーム スイッチは、インタラクティブ モードと非インタラクティブ（スクリプト）モードの両方で Python v2.7.11 [と v3.7.3 (and v3.7.3)] をサポートし、ゲストシェルで使用できます。

Python は簡単に習得できる強力なプログラミング言語です。効率的で高水準なデータ構造を持ち、オブジェクト指向プログラミングに対してシンプルで効果的なアプローチを取っています。Python は、簡潔な構文、動的な型指定、インタープリタ型という特長を持っており、ほとんどのプラットフォームのさまざまな分野でスクリプティングと高速アプリケーション開発が可能な理想的な言語です。

Python インタープリタと広範な標準規格ライブラリが Python Web サイトで送信元形式またはバイナリ形式で自由に利用できます：

<http://www.python.org/>

また、このサイトには、サードパーティが無償で提供している多数の Python モジュール、プログラム、ツールのディストリビューションとそれらへのリンク、さらに追加のドキュメンテーションが掲載されています。

Python スクリプト機能は、さまざまなタスクを実行するためにデバイスのコマンドライン インターフェイス（CLI）Power On Auto Provisioning（POAP）または Embedded Event Manager（EEM）アクションへのプログラムによるアクセスを提供します。Python は Bash シェルからアクセスできます。

Python インタープリタは Cisco NX-OS ソフトウェアで使用できます。

Cisco Nexus デバイスでの Python の使用については、<https://developer.cisco.com/site/nx-os/docs/apis/python/> の URL にある Cisco Nexus 9000 Series Python

SDK User Guide and API Referenceを参照してください。[この情報は、Cisco Nexus 3000 シリーズ デバイスにも適用されます。（This information applies also to the Cisco Nexus 3000 Series devices.）]

Python の使用

ここでは、Python スクリプトの作成と実行の方法について説明します。

Cisco Python パッケージ

Cisco NX-OS は、インターフェイス、VLAN、VRF、ACL、ルートなど、多くのコア ネットワーク デバイス モジュールへのアクセスを可能にする Cisco Python パッケージを提供します。 **help()** コマンドを入力すると、Cisco Python パッケージの詳細を表示できます。モジュール内のクラスとメソッドに関する追加情報を取得するには、特定のモジュールに対して **help** コマンドを実行します。たとえば、**help(cisco.interface)** は、`cisco.interface` モジュールのプロパティを表示します。

その他の Python モジュールについては、https://devhub.cisco.com/artifactory/open-nxos/9.2.1/x86_64/ から `python-modules-nxos RPM` (`python-modules-nxos-1.0.0-9.2.1.lib32_x86.rpm`) をインストールできます。RPM のインストール手順については、「機能 RPM の管理」セクションを参照してください。

次の例は、Cisco Python パッケージに関する情報を表示する方法を示します。

```
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
    cisco

FILE
    /isan/python/scripts/cisco/__init__.py

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

CLASSES
    __builtin__.object
        cisco.cisco_secret.CiscoSecret
```

```
cisco.interface.Interface
cisco.key.Key
```

次に、Python 3 用の Cisco Python パッケージに関する情報を表示する方法の例を示します。

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
cisco

PACKAGE CONTENTS
acl
bgp
buffer_depth_monitor
check_port_discards
cisco_secret
feature
historys
interface
ipaddress
key
line_parser
mac_address_table
md5sum
nxcli
nxos_cli
ospf
routemap
routes
section_parser
ssh
system
tacacs
transfer
vlan
vrf

CLASSES
builtins.dict(builtins.object)
cisco.history.History
builtins.object
cisco.cisco_secret.CiscoSecret
cisco.interface.Interface
cisco.key.Key
```

CLI コマンド API の使用

Python プログラミング言語は、CLI コマンドを実行できる 3 つの API を使用します。API は Python CLI モジュールから利用できます。

これらの API については、次の表で説明します。*** from cli import** コマンドを使用して API を有効にする必要があります。これらの API の引数は CLI コマンドの文字列です。Python インタープリタ経由で CLI コマンドを実行するには、次の API のいずれかの引数文字列として CLI コマンドを入力します。

表 1: CLI コマンド API

API	説明
cli() 例: <pre>string = cli ("cli-command")</pre>	制御文字または特殊文字を含む CLI コマンドの未処理の出力を返します。 (注) インタラクティブな Python インタープリタは、制御文字または特殊文字を「エスケープ」して出力します。キャリッジリターンは '\n' として出力され、結果が読みにくい場合があります。 clip() API は、判読性が高い結果を出力します。
clid() 例: <pre>json_string = clid ("cli-command")</pre>	cli-command コマンドに XML サポートが存在する場合は、JSON 出力を返します。それ以外の場合は、例外がスローされます。 (注) この API は、 show コマンドの出力の検索時に使用すると便利な場合があります。
clip() 例: <pre>clip ("cli-command")</pre>	CLI コマンドの出力を直接 stdout に出力し、Python には何も返されません。 (注) <pre>clip ("cli-command")</pre> と同等です (is equivalent to) <pre>r=cli("cli-command") print r</pre>

2 つ以上のコマンドを個別に実行すると、その状態は 1 つのコマンドから後続のコマンドまで持続しません。

次の例では、最初のコマンドの状態が 2 番目のコマンドで持続しないため、2 番目のコマンドが失敗します。

```
>>> cli("conf t")
>>> cli("interface eth4/1")
```

2 つ以上のコマンドを同時に実行すると、その状態は 1 つのコマンドから後続のコマンドまで持続します。

次の例では、2 番目と 3 番目のコマンドの状態が持続するため、2 番目のコマンドは成功しています。

```
>>> cli("conf t ; interface eth4/1 ; shut")
```



(注) 例に示すように、コマンドは「;」で区切られます。セミコロン(;)は、単一のブランク文字で囲む必要があります。

CLI からの Python インタープリタの呼び出し

次に、CLI から Python 2 を呼び出す方法を表示します:



(注) Python インタープリタのプロンプトは「>>>」または「...」で表示されます。



重要 Python 2.7 はサポート終了です。将来の Cisco NX-OS ソフトウェアは Python 2.7 のサポートを廃止します。新しいスクリプトの代わりに **python3** を使用することをお勧めします。新しいシェルを使用するように **python3** を入力します。

```
switch# python
switch# python
```

```
Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.
```

```
Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print intf['interface']
...
mgmt0
loopback1
>>>
```

次に、CLI から Python 3 を呼び出す方法を表示します:

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
```

```
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print(intf['interface'])
...
mgmt0
loopback1
>>>
```

表示フォーマット

次に、Python API を使用したさまざまな表示フォーマットを示します:

例 1:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> clip('where detail')
mode:
username:          admin
vdc:               switch
routing-context vrf: default
```

例 2:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
' mode:          \n username:          admin\n vdc:
  switch\n routing-context vrf: default\n'
>>>
```

例 3:

```
>>> r = cli('where detail')
>>> print(r)
mode:
username: admin
vdc: switch
routing-context vrf: default

>>>
```

例 4:

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...     print("%30s - %s" % (k,out[k]))
...
header_str - Cisco Nexus Operating System (NX-OS) Software
```

```

TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2020, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.
bios_ver_str - 07.67
kickstart_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
nxos_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
bios_cmpl_time - 01/29/2020
kick_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
nxos_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
kick_cmpl_time - 5/10/2020 21:00:00
nxos_cmpl_time - 5/10/2020 21:00:00
kick_tmstamp - 05/12/2020 07:08:44
nxos_tmstamp - 05/12/2020 07:08:44
chassis_id - Nexus9000 93180YC-EX chassis
cpu_name - Intel(R) Xeon(R) CPU @ 1.80GHz
memory - 24632252
mem_type - kB
proc_board_id - FDO22280FFK
host_name - switch
bootflash_size - 53298520
kern_uptm_days - 0
kern_uptm_hrs - 0
kern_uptm_mins - 19
kern_uptm_secs - 34
rr_usecs - 641967
rr_ctime - Tue May 12 09:52:28 2020
rr_reason - Reset Requested by CLI command reload
rr_sys_ver - 9.4(1)
rr_service - None
plugins - Core Plugin, Ethernet Plugin
manufacturer - Cisco Systems, Inc.
>>>

```

非インタラクティブ Python

Python スクリプト名を引数として Python CLI コマンドで使用することで、Python スクリプトを非インタラクティブモードで実行できます。Python スクリプトは、ブートフラッシュまたは揮発性スキームの下に配置する必要があります。Python CLI コマンドでは、Python スクリプトの最大 32 個のコマンドライン引数を使用できます。

Cisco Nexus 3500 プラットフォーム スイッチは、Python スクリプトを実行するための送信元 CLI コマンドもサポートしています。bootflash:scripts ディレクトリは、ソース CLI コマンドのデフォルトのスクリプト ディレクトリです。

この例では、最初にスクリプトを表示してから実行します。保存は、任意のファイルをブートフラッシュに持ってくるようなものです。

```
switch# show file bootflash:scripts/deltaCounters.py
#!/isan/bin/python3
from cli import *
import sys, time
ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'
out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print ('row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast')
print ('=====')
print (' %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc))
print ('=====')
i = 0
while (i < count):
    time.sleep(delay)
    out = json.loads(clid(cmd))
    rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
    rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
    rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
    txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
    txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
    txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
    i += 1
    print ('%-3d %8d %8d %8d %8d %8d' % (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew - rxbc, txucNew - txuc, txmcNew - txmc, txbcNew - txbc))

switch# python bootflash:scripts/deltaCounters.py mgmt0 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
=====
          291          8233          1767          185          57          2
=====
1           1           4           1           1           0           0
2           2           5           1           2           0           0
3           3           9           1           3           0           0
4           4          12           1           4           0           0
5           5          17           1           5           0           0
switch#
```

次の例は、**source** コマンドがコマンドライン引数を指定する方法を表示しています。この例では、**policy-map** は **cgrep python** スクリプトへの引数です。この例は、**source** コマンドがパイプ演算子（「|」）の後に続くことも表示しています。

```
switch# show running-config | source sys/cgrep policy-map

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
```



```
policy-map type qos pfc-tor-port
```

Embedded Event Manager でのスクリプトの実行

Cisco Nexus 3500 プラットフォーム スイッチ上の組み込みイベント マネージャ（EEM）ポリシーは、Python スクリプトをサポートします。

次の例は、EEM アクションとして Python スクリプトを実行する方法を示しています。

- アクション コマンドを使用することで、EEM アプレットに Python スクリプトを含めることができます。

```
switch# show running-config eem

!Command: show running-config eem
!Running configuration last done at: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

version 9.3(5) Bios:version 07.67
event manager applet a1
  event cli match "show clock"
  action 1 cli python bootflash:pydate.py

switch# show file logflash:vdc_1/event_archive_1 | last 33

eem_event_time:06/25/2020,15:34:24 event_type:cli event_id:24 slot:active(1) vdc
:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
stty: standard input: Inappropriate ioctl for device
Executing the following commands succeeded:
  python bootflash:pydate.py
Completed executing policy a1
Event Id:24 event type:10241 handling completed
```

- **show file logflash:event_archive_1** コマンドを実行して、ログファイル内のイベントによってトリガーされたアクションを検索できます。

```
switch# show file logflash:event_archive_1 | last 33

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
Python

2011-05-01 19:40:28.644891
Executing the following commands succeeded:
  python bootflash:pydate.py

PC_VSH_CMD_TLV(7679) with q
```

Cisco NX-OS ネットワーク インターフェイスとの Python 統合

Cisco Nexus 3500 プラットフォーム スイッチでは、Python が基盤となる Cisco NX-OS ネットワーク インターフェイスと統合されています。cisco.vrf.set_global_vrf () API を介してコンテキストを設定することにより、ある仮想ルーティング コンテキストから別の仮想ルーティング コンテキストに切り替えることができます。

次の例は、デバイスの管理インターフェイスを介して HTML ドキュメントを取得する方法を示しています。目的の仮想ルーティング コンテキストに切り替えることにより、帯域内インターフェイスを介して外部エンティティへの接続を確立することもできます。

```
switch# python
```

```
Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.
```

```
Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000
>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set_global_vrf in module cisco.vrf:
set_global_vrf(vrf)
Sets the global vrf. Any new sockets that are created (using socket.socket)
will automatically get set to this vrf (including sockets used by other
python libraries).
Arguments:
vrf: VRF name (string) or the VRF ID (int).
Returns: Nothing
>>>
```

Python による Cisco NX-OS セキュリティ

Cisco NX-OS 情報技術は、ソフトウェアの Cisco NX-OS サンドボックス レイヤおよび CLI ロールベース アクセス コントロール (RBAC) によって保護されます。

Cisco NX-OS network-admin または dev-ops ロールに関連付けられているすべてのユーザは、特権ユーザです。カスタム ロールで Python へのアクセスが許可されているユーザは、非特権ユーザと見なされます。非特権ユーザは、ファイル システム、ゲスト シェル、Bash コマンドなどの Cisco NX-OS 情報技術へのアクセスが制限されています。特権ユーザは、Cisco NX-OS のすべての情報技術へのアクセスが向上します。

セキュリティとユーザー権限の例

•

Scheduler でのスクリプトの実行例

.

翻訳について

このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。