Cisco Nexus 3000 シリーズ NX-OS Python API リファレンス ガイド リリース 5.0(3)U3(2)

2012 年 4 月 27 日

Text Part Number: OL-26601-02-J

【注意】シスコ製品をご使用になる前に、安全上の注意（www.cisco.com/jp/go/safety_warning/）をご確認ください。

本書は、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。

また、契約等の記述については、弊社販売パートナー、または、弊社担当者にご確認ください。

Cisco Nexus 3000 シリーズ NX-OS Python API リファレンス ガイド リリース 5.0(3)U3(2)

CONTENTS

はじめに v

対象読者 v

表記法 v

関連資料 vi

マニュアルの入手方法およびテクニカル サポート vii

CHAPTER 1 概要 1-1

Python API の概要 1-1

Python のインストール 1-2

サードパーティ製の純粋な Python パッケージのインストール 1-2

Python の使用 1-2

Python シェルの開始 1-3

スクリプトの実行 1-3

スクリプトへのパラメータの引き渡し 1-3

Embedded Event Manager のサポート 1-3

CHAPTER 2 アプリケーション プログラミング インターフェイス（API）関数 2-1

Routes() 2-1

show_arp_table() 2-2

show_vsh_routes() 2-3

show_hw_routes() 2-3

verify_routes() 2-4

verify_arp_table() 2-5

CheckPortDiscards() 2-6

クラス BufferDepthMonitor(CLI) 2-7

get_total_instant_usage() 2-7

get_remaining_instant_usage() 2-8

get_max_cell_usage() 2-8

get_switch_cell_count() 2-9

transfer() 2-10

CLI() 2-10

get_output() 2-11

rerun() 2-12
History() 2-12
get_history() 2-13
clear_history() 2-13

はじめに

このマニュアルでは、Cisco Nexus 3000 シリーズ スイッチの Python アプリケーション プログラミング インターフェイス（API）をリストし、説明します。

この章は、次の内容で構成されています。

- 「対象読者」（P.v）
- 「表記法」（P.v）
- 「関連資料」（P.vi）
- 「マニュアルの入手方法およびテクニカル サポート」（P.vii）

対象読者

このマニュアルは Cisco Nexus 3000 シリーズ スイッチの設置と管理を担当するシステム管理者を対象としています。

表記法

コマンドの説明では、次の表記法を使用しています。

表記法	説明
太字	コマンドおよびキーワードは太字で示しています。
イタリック体	ユーザが値を指定する引数は、イタリック体で示しています。
{ }	波カッコの中の要素は、必須の選択要素です。
[ ]	角カッコの中の要素は、省略可能です。
x | y | z	いずれか 1 つを選択する要素は、縦線で区切って示されます。
string	引用符を付けない一組の文字。string の前後には引用符を使用しません。引用符を使用すると、その引用符も含めて string とみなされます。

出力例では、次の表記法を使用しています。

`screen フォント`	デバイスが表示するターミナル セッションおよび情報は、screen フォントで示しています。
太字の `screen` フォント	ユーザが入力しなければならない情報は、太字の screen フォントで示しています。

イタリック体の screen フォント	ユーザが値を指定する引数は、イタリック体の screen フォントで示しています。
< >	パスワードのように出力されない文字は、山カッコ（<>）で囲んで示しています。
[ ]	システム プロンプトに対するデフォルトの応答は、角カッコで囲んで示しています。
!、#	コードの先頭に感嘆符（!）またはポンド記号（#）がある場合には、コメント行であることを示します。

このマニュアルでは、次の表記法を使用しています。

（注） 「*注釈*」を意味します。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。

注意 「*要注意*」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。

関連資料

Cisco Nexus 3000 シリーズ スイッチのドキュメンテーションは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/tsd_products_support_series_home.html

ドキュメンテーション セットは次のカテゴリに分けられます。

リリース ノート

リリース ノートは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/prod_release_notes_list.html

インストレーション ガイドおよびアップグレード ガイド

インストレーション ガイドとアップグレード ガイドは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/prod_installation_guides_list.html

コマンド リファレンス

コマンド リファレンスは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/prod_command_reference_list.html

テクニカル リファレンス

テクニカル リファレンスは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/prod_technical_reference_list.html

コンフィギュレーション

コンフィギュレーション ガイドは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/products_installation_and_configuration_guides_list.html

エラー メッセージおよびシステム メッセージ

システム メッセージ リファレンス ガイドは、次の URL で入手できます。

http://www.cisco.com/en/US/products/ps11541/products_system_message_guides_list.html

マニュアルの入手方法およびテクニカル サポート

マニュアルの入手方法、テクニカル サポート、その他の有用な情報について、次の URL で、毎月更新される『*What's New in Cisco Product Documentation*』を参照してください。シスコの新規および改訂版の技術マニュアルの一覧も示されています。

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

『*What's New in Cisco Product Documentation*』は RSS フィードとして購読できます。また、リーダーアプリケーションを使用してコンテンツがデスクトップに直接配信されるように設定することもできます。RSS フィードは無料のサービスです。シスコは現在、RSS バージョン 2.0 をサポートしています。

CHAPTER 1

概要

この章では、Cisco Nexus 3000 シリーズ スイッチで Python アプリケーション プログラミング インターフェイス（API）サポート機能を使用するための概要と、必要なインストール情報を示します。

この章は、次の内容で構成されています。

- 「Python API の概要」（P.1-1）
- 「Python のインストール」（P.1-2）
- 「サードパーティ製の純粋な Python パッケージのインストール」（P.1-2）
- 「Python の使用」（P.1-2）

Python API の概要

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、動的な型付け、およびインタープリタ型という性質によって、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なアプリケーション開発を実現する理想的な言語です。

Python のインタープリタと広範な標準ライブラリは、Python Web サイトから、主要なプラットフォームに対応したソース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、および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。

Cisco Nexus 3000 シリーズ スイッチは、Python v2.7.2 で使用できるすべての機能をサポートします。

Cisco Nexus 3000 シリーズ スイッチの Python スクリプティング機能では、次の作業を行うことができます。

- スイッチ起動時に設定を確認するためのスクリプトを実行する。
- 設定をバックアップする。
- バッファ使用状況の特性をモニタし、対応することで、予防的な輻輳管理を行う。
- 電源投入時の自動プロビジョニングまたは EEM モジュールとの統合。
- 一定の時間間隔でジョブを実行する（ポートの自動定義など）。
- スイッチのコマンドライン インターフェイス（CLI）にプログラムからアクセスしてさまざまなタスクを実行する。

Python のインストール

Python インタープリタは Cisco NX-OS ソフトウェアにデフォルトで搭載されています。Python は **python** コマンドを入力することで起動できます。また、**import cisco.py** コマンドで cisco.py モジュールをインポートすることで、Cisco NX-OS API にアクセスするスクリプトを作成できます。

サードパーティ製の純粋な Python パッケージのインストール

mypkg.tgz をサーバにコピーすることで、サードパーティ製の純粋な Python パッケージをインストールできます。サードパーティ製パッケージを抽出してインストールするには、次の手順を実行します。

- **copy scp://user@server/path/to/mypkg.tgz bootflash:mypkg.tgz vrf management** コマンドを実行して、tar ファイルのセキュア コピーを行います。
- **tar extract bootflash:mypkg.tgz** コマンドを使用して、mypkg.tgz ファイルを展開します。
- **move bootflash:mypkg-1.2/* bootflash:** コマンドを使用して、抽出したファイルをブートフラッシュに移動します。
- **python setup.py instal** コマンドを使用して、パッケージをインストールできます。
- コピーしたファイルをブートフラッシュから削除します。
- サードパーティ製パッケージは、スクリプトまたは Python シェルで使用できます。

```
switch# python
>>> import mypkg
```

（注） 将来のリリースでは、**easy_install** コマンドを使用して、サードパーティ製パッケージをインストールできるようになります。

Python の使用

（注） SPEP8 コーディングのガイドラインに準拠するために、Cisco Python API の名前がいくつか変更されました。
既存のスクリプトを新しい API に変換するために、次の例を使用できます。

```
% cat sed.in
s/showArpTable/show_arp_table/g
s/showVshRoutes/show_vsh_routes/g
s/showHwRoutes/show_hw_routes/g
s/verifyRoutes/verify_routes/g
s/verifyArpTable/verify_arp_table/g
s/getTotalInstantUsage/get_total_instant_usage/g
s/GetRemainingInstantUsage/get_remaining_instant_usage/g
s/getMaxCellUsage/get_max_cell_usage/g
s/getSwitchCellCount/get_switch_cell_count/g

% sed -f sed.in < python_script_with_old_API_names.py > python_script_with_old_API_names.py
```

ここでは、パラメータを渡すことによって Python スクリプトを作成し、実行する方法について説明します。内容は次のとおりです。

- 「Python シェルの開始」（P.1-3）
- 「スクリプトの実行」（P.1-3）
- 「スクリプトへのパラメータの引き渡し」（P.1-3）
- 「Embedded Event Manager のサポート」（P.1-3）

Python シェルの開始

Python シェルは、パラメータを指定せずに **python** コマンドを使用することで開始できます。

```
switch# python
Python 2.7.2 (default, Oct 11 2011, 13:55:49)
[GCC 3.4.3 (MontaVista 3.4.3-25.0.143.0800417 2008-02-22)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NX-OS lib!
>>> print 'helo world!'
helo world!
>>>exit()
switch#
```

スクリプトの実行

Python スクリプトは、**python** <*filename*> コマンドを使用して実行することができます。

```
switch# python test.py
['/bootflash/test.py']
doing 0/1
doing 0/2
doing 1/2
switch#
```

スクリプトへのパラメータの引き渡し

Python スクリプトは、**python** <*filename*> [*arg1, arg2, arg3,.....*] コマンドを使用して実行することができます。

```
switch# python test.py foo bar 1 2
['/bootflash/test.py', 'foo', 'bar', '1', '2']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Embedded Event Manager のサポート

Embedded Event Manager（EEM）は、イベントに基づいた Python スクリプトの呼び出しをサポートします。**variable $** コマンドを使用して、イベントの syslog を Python スクリプトに渡すことができます。

次の例は、IF_DOWN イベントに対して Python スクリプトを呼び出す EEM を示しています。

EEM の設定：

```
switch(config)# event manager applet if-mon
switch(config-applet)# event syslog pattern "*IF_DOWN.*"
Configuration accepted successfully
switch(config-applet)# action 1.0 cli python if-mon.py eth1/1 $command
switch(config-applet)# end
```

Python スクリプト:

```
If-mon.py

import re
import sys

def findIf ():
    x = re.compile ('[Ee]thernet\d+\/\d+')
    for a in sys.argv[1:]:
        if x.match (a):
            print a
            return a
    return None

print 'Starting my script.. args:'
print sys.argv
intf = findIf ()
if not intf:
    intf = 'eth1/1'

print 'Detected shut on interface %s' % intf

from cisco import *
s, o = cli ('show run int %s' % intf)
print ('----------------------\n%s\n' % o)
print 'Restoring interface %s' % intf
s, o = cfg_if (intf, desc='++dont shut++', state='up')
print ('----------------------\n%s\n' % o)
s, o = cli ('show run int %s' % intf)
print ('sh ver\n----------------------\n%s\n' % o)

print ('\nbye\n')
```

CHAPTER 2

アプリケーション プログラミング インターフェイス（API）関数

この章では、次の Python アプリケーション プログラミング インターフェイス（API）関数について説明します。この章は、次の内容で構成されています。

- 「Routes()」（P.2-1）
- 「show_arp_table()」（P.2-2）
- 「show_vsh_routes()」（P.2-3）
- 「show_hw_routes()」（P.2-3）
- 「verify_routes()」（P.2-4）
- 「verify_arp_table()」（P.2-5）
- 「CheckPortDiscards()」（P.2-6）
- 「クラス BufferDepthMonitor(CLI)」（P.2-7）
- 「get_total_instant_usage()」（P.2-7）
- 「get_remaining_instant_usage()」（P.2-8）
- 「get_max_cell_usage()」（P.2-8）
- 「get_switch_cell_count()」（P.2-9）
- 「transfer()」（P.2-10）
- 「CLI()」（P.2-10）
- 「get_output()」（P.2-11）
- 「rerun()」（P.2-12）
- 「History()」（P.2-12）
- 「get_history()」（P.2-13）
- 「clear_history()」（P.2-13）

Routes()

概要

```
Routes() - Class Object
```

構文
```
Routes()
```

説明
Routes クラスのオブジェクトをインスタンス化します。

パラメータ
なし。

戻り値
Routes クラスのオブジェクト。

例
```
rObj = Routes()
```

show_arp_table()

概要
```
show_arp_table()
```

構文
```
Routes.show_arp_table()
```

説明
show ip arp コマンドを実行し、出力を返します。

パラメータ
なし。

戻り値
スイッチの ARP テーブル エントリを返します。

例
```
routeObj = Routes()
data = routeObj.show_arp_table().get_output()
```

出力例
```
Flags: D - Static Adjacencies attached to down interface

IP ARP Table for context default
Total number of entries: 4
Address         Age       MAC Address     Interface
50.1.201.2      00:02:10  547f.ee40.5a7c  Vlan201
50.1.1.10       00:07:53  547f.ee62.f801  Ethernet1/34
50.1.2.10       00:08:31  547f.ee62.f801  Ethernet1/35
50.1.3.10       00:08:31  547f.ee62.f801  Ethernet1/35.1

<cisco.CLI object at 0xb7c1462c>
```

show_vsh_routes()

概要
```
show_vsh_routes()
```

構文
```
Routes.show_vsh_routes()
```

説明
show ip fib route を実行し、出力を返します。

パラメータ
なし。

戻り値
ソフトウェア ルート エントリを返します。

例
```
routeObj = Routes()
data = routeObj.show_vsh_routes().get_output()
```

出力例
```
IPv4 routes for table default/base

------------------+------------------+----------------------
Prefix            | Next-hop         | Interface
------------------+------------------+----------------------
0.0.0.0/32          Drop               Null0
50.1.1.0/24         Attached           Ethernet1/34
50.1.1.0/32         Drop               Null0
50.1.1.10/32        50.1.1.10          Ethernet1/34
50.1.1.100/32       Receive            sup-eth1
50.1.1.255/32       Attached           Ethernet1/34
50.1.2.0/24         Attached           Ethernet1/35
50.1.2.0/32         Drop               Null0
50.1.2.10/32        50.1.2.10          Ethernet1/35
50.1.2.100/32       Receive            sup-eth1
50.1.2.255/32       Attached           Ethernet1/35
50.1.3.0/24         Attached           Ethernet1/35.1
50.1.3.0/32         Drop               Null0
50.1.3.10/32        50.1.3.10          Ethernet1/35.1
50.1.3.100/32       Receive            sup-eth1
50.1.3.255/32       Attached           Ethernet1/35.1

<cisco.CLI object at 0xb7b0a6ac>
```

show_hw_routes()

概要
```
show_hw_routes()
```

構文
```
Routes.show_hw_routes()
```

説明
ハードウェア ルートを計算し、出力を返します。

パラメータ
なし。

戻り値
ハードウェア ルート エントリを返します。

例
```
routeObj = Routes()
data = routeObj.show_hw_routes()
```

出力例
```
------------------+------------------+----------------------
Prefix            | Next-hop         | Interface
------------------+------------------+----------------------
50.1.1.100/32       Receive            sup-eth1
50.1.2.100/32       Receive            sup-eth1
50.1.201.1/32       Receive            sup-eth1
0.0.0.0/32          Drop               Null0
50.1.3.0/32         Drop               Null0
50.1.201.0/32       Drop               Null0
50.1.2.255/32       Attached           sup-hi
50.1.1.255/32       Attached           sup-hi
60.1.1.0/32         Drop               Null0
50.1.3.255/32       Attached           sup-hi
50.1.201.255/32     Attached           sup-hi
255.255.255.255/32  Receive            sup-eth1
```

verify_routes()

概要
```
verify_routes()
```

構文
```
Routes.verify_routes()
```

説明
ソフトウェアとハードウェアのルートを確認します。

パラメータ
なし。

戻り値
ハードウェアとソフトウェアの間で一致したルートおよび一致しないルートの数を返します。

例
```
routeObj = Routes()
found,nfound = routeObj.verify_routes()
```

出力例

```
Routes verified and found:  26

Routes not found:
50.1.205.0/24       3
51.1.1.0/24         3
51.1.2.0/24         4
51.1.3.0/24         6
100.1.1.0/24        7
100.1.2.0/24        7
100.1.3.0/24        7
101.1.1.0/24        7
101.1.2.0/24        7
101.1.3.0/24        7
120.1.1.0/24        7
```

verify_arp_table()

概要

```
verify_arp_table()
```

構文

```
Routes.verify_arp_table()
```

説明

ソフトウェアとハードウェアの ARP テーブル エントリを確認します。

パラメータ

なし。

戻り値

ハードウェアとソフトウェアの間で一致した ARP テーブル エントリおよび一致しなかった ARP テーブル エントリの数を返します。

例

```
routeObj = Routes()
found,notfound = routeObj.verify_arp_table()
```

出力例

```
Flags: D - Static Adjacencies attached to down interface

IP ARP Table for context default
Total number of entries: 4
Address         Age       MAC Address     Interface
50.1.201.2      00:02:31  547f.ee40.5a7c  Vlan201
50.1.1.10       00:08:15  547f.ee62.f801  Ethernet1/34
50.1.2.10       00:08:53  547f.ee62.f801  Ethernet1/35
50.1.3.10       00:08:53  547f.ee62.f801  Ethernet1/35.1

mac address:54:7f:ee:40:5a:7c
Arp entry for 50.1.201.2 547f.ee40.5a7c Vlan201 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.1.10 547f.ee62.f801 Ethernet1/34 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.2.10 547f.ee62.f801 Ethernet1/35 found in HW
```

```
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.3.10 547f.ee62.f801 Ethernet1/35.1 found in HW
```

CheckPortDiscards()

概要

```
CheckPortDiscards(<port>)
```

構文

```
CheckPortDiscards('ethernet1/1')
```

説明

特定のポートでの入力廃棄を確認します。廃棄が 0 を超える場合は、broadcom から廃棄の理由を照会して印刷します。

パラメータ

port

戻り値

なし。

例

```
c = CheckPortDiscards('eth1/1')
```

出力例

```
Ethernet1/1 is up
  Hardware: 100/1000/10000 Ethernet, address: 547f.ee57.dd28 (bia 547f.ee57.dd28)
  MTU 1500 bytes, BW 10000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA
  Port mode is trunk
  full-duplex, 10 Gb/s, media type is 10G
  Beacon is turned off
  Input flow-control is off, output flow-control is off
  Rate mode is dedicated
  Switchport monitor is off
  EtherType is 0x8100
  Last link flapped 00:42:16
  Last clearing of "show interface" counters never
  30 seconds input rate 5016 bits/sec, 627 bytes/sec, 6 packets/sec
  30 seconds output rate 3232 bits/sec, 404 bytes/sec, 5 packets/sec
  Load-Interval #2: 5 minute (300 seconds)
    input rate 4.69 Kbps, 7 pps; output rate 2.82 Kbps, 4 pps
  RX
    297 unicast packets  20588 multicast packets  5 broadcast packets
    20890 input packets  1848701 bytes
    0 jumbo packets  0 storm suppression packets
    0 giants  0 input error  0 short frame  0 overrun   0 underrun
    0 watchdog  0 if down drop
    0 input with dribble  0 input discard(includes ACL drops)
    0 Rx pause
  TX
    262 unicast packets  16151 multicast packets  5 broadcast packets
    16418 output packets  1407200 bytes
    0 jumbo packets
```

```
    0 output errors  0 collision  0 deferred  0 late collision
    0 lost carrier  0 no carrier  0 babble
    0 Tx pause
  2 interface resets

zero discards
```

クラス BufferDepthMonitor(CLI)

概要

```
BufferDepthMonitor()
```

構文

```
monitorObj = BufferDepthMonitor()
```

説明

このクラスは、**show hardware internal buffer info pkt-stats** コマンドの出力を分析し、バッファの使用量を返すメソッドにアクセスできるようにします。

パラメータ

なし。

戻り値

このクラスのオブジェクト インスタンス。

例

```
b = BufferDepthMonitor()
```

出力例

```
|-----------------------------------------------------------------------------------------
|

                          Total Instant Usage             0
                          Remaining Instant Usage     46080
                          Max Cell Usage                 19
                          Switch Cell Count           46080
|-----------------------------------------------------------------------------------------
|
<cisco.BufferDepthMonitor object at 0xb7c1462c>
```

get_total_instant_usage()

概要

```
get_total_instant_usage()
```

構文

```
monitorObj = BufferDepthMonitor()
totUsage = monitorObj.get_total_instant_usage()
```

説明

show hardware internal buffer info pkt-stats コマンドの出力からその瞬間の合計使用量を返すメソッド。

パラメータ

なし。

戻り値

その瞬間の合計使用量を返します。

例

```
b = BufferDepthMonitor()
b.get_total_instant_usage()
```

出力例

```
0
```

get_remaining_instant_usage()

概要

```
get_remaining_instant_usage()
```

構文

```
monitorObj = BufferDepthMonitor()
remUsage = monitorObj.get_remaining_instant_usage()
```

説明

show hardware internal buffer info pkt-stats コマンドの出力からその瞬間の残りの使用量を返すメソッド。

パラメータ

なし。

戻り値

その瞬間の合計使用量を返します。

例

```
b = BufferDepthMonitor()
b.get_remaining_instant_usage()
```

出力例

```
46080
```

get_max_cell_usage()

概要

```
get_max_cell_usage()
```

構文

```
monitorObj = BufferDepthMonitor()
cellUsage = monitorObj.get_max_cell_usage()
```

説明

show hardware internal buffer info pkt-stats コマンドの出力からセル使用量を返すメソッド。

パラメータ

なし。

戻り値

その瞬間の合計使用量を返します。

例

```
b = BufferDepthMonitor()
b.get_max_cell_usage()
```

出力例

```
19
```

get_switch_cell_count()

概要

```
get_switch_cell_count()
```

構文

```
monitorObj = BufferDepthMonitor()
cellCount = monitorObj.get_switch_cell_count()
```

説明

show hardware internal buffer info pkt-stats コマンドの出力からセル カウントの使用量を返すメソッド。

パラメータ

なし。

戻り値

その瞬間の合計使用量を返します。

例

```
b = BufferDepthMonitor()
b.get_switch_cell_count()
```

出力例

```
46080
```

transfer()

概要
```
transfer()
```

構文
```
transfer (<protocol>, <host>, <source>, <dest>, <vrf>, <login_timeout>, <user>,
<password>)
```

説明
<source> で指定されたファイルを <host> から <protocol> を使用して <dest> に記載されているパスに転送する API。**プロトコルには、scp、tftp、ftp、または sftp を使用できます。**

パラメータ
protocol、host、source、dest、vrf、login_timeout、user、password。

戻り値
転送が成功した場合は True を返します。

例
scp を使用する転送:
```
c = transfer("scp", "10.193.190.100", "/tftpboot/transfer_test_image",
"transfer_test_image", user="scpUser", password="scpPasswd")
```

sftp を使用する転送:
```
c = transfer("sftp", "10.193.190.100", "/tftpboot/transfer_test_image",
"transfer_test_image", user="sftpUser", password="sftpPasswd")
```

tftp を使用する転送:
```
c = transfer("tftp", "10.193.190.100", "/transfer_test_image", "transfer_test_image",
user="", password="")
```

ftp を使用する転送:
```
c = transfer("ftp", "10.193.190.51", "golden/home/su-ash/transfer_test_image",
"transfer_test_image", user="ftpUser", password="ftpPasswd")
```

CLI()

概要
```
CLI() - Class Object
```

構文
```
CLI (<command>, <do_print>)
```

説明
<command> で指定された CLI コマンドで CLI クラスのオブジェクトをインスタンス化します。
<do_print> を False に設定するとコマンドの出力が印刷されず、True（デフォルト）に設定すると出力が印刷されます。

パラメータ
command、do_print

戻り値
CLI クラスのオブジェクト。

例
```
c = CLI ('show runn inter eth1/1')
```

出力例
```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:33:24 2012

version 5.0(3)U3(1)

interface Ethernet1/1
  switchport mode trunk
  udld enable
  channel-group 12

<cisco.CLI object at 0xb7ae948c>
```

get_output()

概要
```
get_output()
```

構文
```
CLI.get_output()
```

説明
CLI コマンドの出力を返します。

パラメータ
なし。

戻り値
CLI コマンドの出力。

例
```
c = CLI ('show runn inter eth1/1')
c.get_output()
```

出力例
```
['', '!Command: show running-config interface Ethernet1/1', '!Time: Mon Feb 27 14:36:10
2012', '', 'version 5.0(3)U3(1)', '', 'interface Ethernet1/1', '  switchport mode trunk',
'  udld enable', '  channel-group 12', '', '']
```

rerun()

概要
```
rerun()
```

構文
```
CLI.rerun()
```

説明
コマンドを再実行します。

パラメータ
なし。

戻り値
なし。

例
```
c = CLI ('show runn inter eth1/1')
c.rerun()
```

出力例
```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:37:05 2012

version 5.0(3)U3(1)

interface Ethernet1/1
  switchport mode trunk
  udld enable
  channel-group 12
```

History()

概要
History() - クラス オブジェクト

構文
History()

説明
History クラスのオブジェクトをインスタンス化します。

パラメータ
なし。

戻り値
History クラスのオブジェクト。

例

```
a = History()
```

get_history()

概要

```
get_history()
```

構文

```
History.get_history()
```

説明

これまでに実行された CLI コマンドの履歴を取得します。

パラメータ

なし。

戻り値

実行されたコマンドの履歴を返します。

例

```
a = History()
a.get_history()
```

clear_history()

概要

```
clear_history()
```

構文

```
History.clear_history()
```

説明

履歴をクリアします。

パラメータ

なし。

戻り値

なし。

例

```
a = History()
a.clear_history()
```

clear_history()