目 录

1 Python

1.1 Python简介

Python 是一种简单易学，功能强大的编程语言，它有高效率的高层数据结构，简单而有效地实现了面向对象编程。Python 简洁的语法和对动态输入的支持，再加上解释性语言的本质，使得它在大多数平台上的许多领域都是一个理想的脚本语言，特别适用于快速的应用程序开发。

在 Comware V7 系统上可以采用如下方式使用 Python：

- 通过执行 Python 脚本进行自动化配置系统。
- 进入Python shell，使用Python2.7 版本的命令、标准API或扩展API对设备进行配置。其中，扩展API是Comware对Python进行的扩展，用来方便用户进行系统配置。关于Comware的Python扩展，可以参考“2 Comware扩展Python API”。

1.2 执行Python脚本文件

请在用户视图下执行本命令，执行 Python 脚本文件。

python *filename*

1.3 进入Python shell

请在用户视图下执行本命令，进入 Python shell。

python

1.4 导入Comware包以使用扩展API

用户如需使用扩展 Python API，必须先导入 Comware 包。导入时，可选择导入整个 Comware 包或单个 API。

1.4.1 导入整个Comware包并执行扩展API

1. 配置步骤

(1) 请在用户视图下执行本命令，进入 Python shell。

python

(2) 导入整个 Comware 包。

import comware

(3) 执行扩展 API。

comware.*api*

2. 配置举例

下例采用 API Transfer 将 TFTP 服务器（192.168.1.26）上的文件 test.cfg 下载到设备上。

```
<Sysname> python
```

```
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.Transfer('tftp', '192.168.1.26', 'test.cfg', 'flash:/test.cfg', user='',
password='')
<comware.Transfer object at 0xb7eab0e0>
```

1.4.2 导入单个API函数并执行该函数

1. 配置步骤

(1) 请在用户视图下执行本命令，进入 Python shell。

python

(2) 导入单个 API 函数。

from comware import *api-name*

(3) 执行扩展 API 函数。

api-function

2. 配置举例

下例采用 API Transfer 将 TFTP 服务器（192.168.1.26）上的文件 test.cfg 下载到设备上。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from comware import Transfer
>>> Transfer('tftp', '192.168.1.26', 'test.cfg', 'flash:/test.cfg', user='', password='')
<comware.Transfer object at 0xb7e5e0e0>
```

1.5 退出Python shell

请在 Python shell 下执行本命令，退出 Python shell。

exit()

1.6 Python典型配置举例

1.6.1 Python基础配置举例

1. 组网需求

使用 Python 脚本，下载 main.cfg 和 backup.cfg 两个配置文件到设备上，并设置为下次主用配置文件和备用配置文件。

2. 组网图

图1-1 Python 典型配置举例组网图

3. 配置步骤

在 PC 上使用写字板编辑 Python 脚本文件 test.py，内容如下：

```
#!usr/bin/python
import comware

comware.Transfer('tftp', '192.168.1.26', 'main.cfg', 'flash:/main.cfg')
comware.Transfer('tftp', '192.168.1.26', 'backup.cfg', 'flash:/backup.cfg')
comware.CLI('startup saved-configuration flash:/main.cfg main ;startup saved-configuration flash:/backup.cfg backup')
```

通过 TFTP 将 test.py 文件下载到设备上

```
<Sysname> tftp 192.168.1.26 get test.py
```

执行 Python 脚本文件

```
<Sysname> python flash:/test.py
<Sysname>startup saved-configuration flash:/main.cfg main
Please wait...... Done.
<Sysname>startup saved-configuration flash:/backup.cfg backup
Please wait...... Done.
```

4. 验证结果

使用 **display startup** 命令查看下次启动文件已经变为 main.cfg 和 backup.cfg。

```
<Sysname> display startup
 Current startup saved-configuration file: flash:/startup.cfg
 Next main startup saved-configuration file: flash:/main.cfg
 Next backup startup saved-configuration file: flash:/backup.cfg
```

2 Comware扩展Python API

本文描述在 Comware V7 中提供的扩展 Python API，扩展 Python API 必须遵循标准 Python 语言语法。

2.1 CLI

用来执行 Comware V7 系统的命令并创建 CLI 对象。

【命令】

CLI(*command*='', *do_print*=True)

【参数】

command：表示要下发的命令，缺省为空。CLI 下发命令是从用户视图开始，如果 *command* 中不指定视图，直接输入命令，表示该命令在用户视图下执行；当需要执行其它视图的命令时，需要先输入进视图的命令，再输入具体的配置命令。多条命令之间以空格加分号分隔，如'system-view ;local-user test class manage'。

do_print：表示是否输出执行结果，True 表示输出执行结果，False 表示不输出执行结果。缺省值为 True。

【返回值】

CLI 对象

【使用指导】

需要注意的是，**CLI** 仅支持 Comware 命令，不支持 Linux、Python、Tcl 命令。

【举例】

使用 API CLI 添加本地用户 test。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.CLI('system-view ;local-user test class manage')
```

【结果】

```
<Sysname> system-view
System View: return to User View with Ctrl+Z.
[Sysname] local-user test class manage
New local user added.
<comware.CLI object at 0xb7f680a0>
```

2.2 get_error

用来获取下载文件过程中的错误信息。

【命令】

Transfer.get_error()

【返回值】

下载文件过程中的错误信息，若没有错误信息则返回 None。

【举例】

使用 API Transfer 将 TFTP 服务器（1.1.1.1）上的文件 test.cfg 下载到设备上。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> c = comware.Transfer('tftp', '1.1.1.1', 'test.cfg', 'flash:/test.cfg', user='',
password='')
>>> c.get_error()
```

【结果】

```
“Timeout was reached”
```

2.3 get_output

用来获取命令执行的输出信息。

【命令】

CLI.get_output()

【返回值】

命令执行的输出信息

【举例】

使用 API CLI 添加本地用户，并输出命令行执行结果。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> c = comware.CLI('system-view ;local-user test class manage', False)
>>> c.get_output()
```

【结果】

```
['<Sysname>system-view', 'System View: return to User View with Ctrl+Z.',
'[Sysname]local-user test class manage', 'New local user added.']
```

2.4 get_self_slot

get_self_slot 接口用来获取主用主控板所在的槽位号。（独立运行模式）

get_self_slot 接口用来获取主设备的成员编号。（IRF 模式）

【命令】

get_self_slot()

【返回值】

设备上无主用主控板，返回一个列表对象，值始终为[-1,-1]。（独立运行模式）

返回一个列表对象，格式为：[-1,*slot-number*]，其中 *slot-number* 表示主设备在 IRF 中的成员编号。（IRF 模式）

【举例】

使用 API 获取主设备所在的成员编号。（IRF 模式）

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.get_self_slot()
```

【结果】

```
[-1,0]
```

2.5 get_slot_info

get_slot_info 接口用来获取指定设备的信息。（独立运行模式）

get_slot_info 接口用来获取指定成员设备的信息。（IRF 模式）

【命令】

get_slot_info()

【返回值】

返回一个字典对象，返回值始终为{'Slot': *slot-number*, 'Status': '*status*', 'Chassis': *chassis-number*, 'Role': '*role*', 'Cpu': *CPU-number*}。*slot-number* 表取值固定为 0，*status* 表示成员设备的状态，*chassis-number* 取值固定为 0，*role* 表示设备的角色，*CPU-number* 取值固定为 0。（独立运行模式）

返回一个字典对象，返回值始终为{'Slot': *slot-number*, 'Status': '*status*', 'Chassis': *chassis-number*, 'Role': '*role*', 'Cpu': *CPU-number*}。*slot-number* 表示备在 IRF 中的成员编号，*status* 表示成员设备的状态，*chassis-number* 取值固定为 0，*role* 表示成员设备的角色，*CPU-number* 取值固定为 0。（IRF 模式）

【举例】

使用 API 获取成员编号/槽位号信息。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.get_slot_info(1)
```

【结果】

```
{'Slot': 1, 'Status': 'Normal', 'Chassis': 0, 'Role': 'Master', 'Cpu': 0}
```

2.6 get_slot_range

get_slot_range 接口用来获取当前系统所支持的成员编号范围。

【命令】

get_slot_range()

【返回值】

返回一个字典对象，返回值始终为{'MaxSlot': *max-slot-number*, 'MinSlot': *min-slot-number*}。*max-slot-number* 表示设备支持的最大成员编号，*min-slot-number* 表示设备支持的最小成员编号。

【举例】

使用 API 获取系统成员编号范围。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.get_slot_range()
```

【结果】

{'MaxSlot': 0, 'MinSlot': 0}

2.7 get_standby_slot

get_standby_slot 接口用来获取所有备用主控板所在的槽位号。（独立运行模式）

get_standby_slot 接口用来获取所有从设备的成员编号。（IRF 模式）

【命令】

get_standby_slot()

【返回值】

设备上无备用主控板，返回一个列表对象，返回值始终为[ ]。（独立运行模式）

返回一个列表对象，格式为：[[-1,*slot-number*]]，其中 *slot-number* 表示从设备在 IRF 中的成员编号。如果 IRF 中没有从设备，则返回[ ]；当 IRF 中有多个从设备时，则返回：[[-1,*slot-number1*],[-1,*slot-number2*],……]。（IRF 模式）

【举例】

使用 API 获取从设备所在的成员编号。（IRF 模式）

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
```

```
>>> comware.get_standby_slot()
```

【结果】

```
[[-1, 1], [-1, 2]]
```

2.8 Transfer

用来将指定文件通过指定协议下载到本地。

【命令】

Transfer(*protocol*='', *host*='', *source*='', *dest*='', *vrf*='',*login_timeout*=10, *user*='', *password*='')

【参数】

protocol：表示下载文件时使用的协议。取值为：

- **ftp**：表示使用 FTP 协议传输文件。
- **tftp**：表示使用 TFTP 协议传输文件。
- **http**：表示使用 HTTP 协议传输文件。

host：表示远程服务器的 IP 地址。

source：表示服务器上源文件的名称。

dest：表示保存到本地的目的文件的名称。

vrf：指定目的端所属的 MPLS L3VPN 的 VPN 实例名称，为 1～31 个字符的字符串，区分大小写。如果未指定本参数，则表示目的端位于公网中。

login_timeout：表示下载文件时登录的超时时间，单位为秒，缺省值为 10。

user：表示登录时使用的用户名称。

password：表示登录时使用的用户密码。

【返回值】

Transfer 对象

【举例】

使用 API Transfer 将 TFTP 服务器（192.168.1.26）上的文件 test.cfg 下载到设备上。

```
<Sysname> python
Python 2.7.3 (default)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import comware
>>> comware.Transfer('tftp', '192.168.1.26', 'test.cfg', 'flash:/test.cfg', user='',
password='')
```

【结果】

```
<comware.Transfer object at 0xb7f700e0>
```