CTC
Challenging Tomorrow's Changes

明日からはじめる ネットワーク運用自動化

2018/1/25 10:00-11:00JST

ITOCHU Techno-Solutions America, Inc.
NetOpsCoding Committee
Taiji Tsuchiya / 土屋 太二

はじめに

- 本発表に先立ち、
  プログラミング基本部分にフォーカスした内容を
  オンライン発表をさせていただきました。
  - 明日からはじめるネットワーク運用自動化　始動編
    2017年1月9日 10:00 – 12:00 JST
    https://www.janog.gr.jp/meeting/janog41/program/autoprep
  - 上記ページにて発表資料および動画が公開されています。

- Webex上で204名の方にご参加いただきました。
  ご参加していただいた皆様、ありがとうございました！
- 本発表では一部重複する部分がありますが、
  ご容赦いただけると幸いです。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

自己紹介

- 土屋 太二/Taiji Tsuchiya
- Career
  - Solution Engineer @ CTC America (2017.8-Present)
    - 米国進出する日系企業向けのITソリューション提供
    - ハイパースケール企業のネットワーク/インフラ技術の調査
    - アプリケーションソフトウェアのプロトタイプ開発
  - Network Engineer @ BIGLOBE (2011.4-2017.8)
    - DC/バックボーン/Peeringの運用/設計/開発
    - ネットワーク運用自動化システム、SDNシステムの開発
- Community Activities
  - 過去JANOG プログラム委員, 実行委員長など
  - NetOpsCoding 運営委員

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

インフラ運用の自動化への要望
(ネットワークに限らず)

- 機器作業における人為ミスを無くしたい。
- 運用に関連する作業の効率性を高めたい。
  - 例: 機器一台あたりの作業/運用にかかる時間を短縮したい。
  - 例: 技術者一人あたりの運用対象機器を増やしたい。
- 障害発生時に、迅速に原因を発見および復旧したい。
- 障害発生前に、予兆を発見し、危険因子を取り除きたい。
- サービス開発チームからの要求に応じて、
迅速にインフラの準備・更新・廃止を実施したい。
- インフラの正常な状態を維持したい、
もしくは異常発生時に迅速に正常状態を再現したい。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク運用現場から聞こえる声

自動化が必要なのはわかる。
ただ、どのように取り組むべきかわからない...

自動化開発のための知識・スキルが無い ...

ネットワークと開発、両方できる人材がいない...

自動化に対する組織の理解・サポートが足りない...

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

運用自動化の第一歩目:
まずは運用現場からはじめる

運用課題(小)
- 手早く改善効果を出せる作業
- 定型化しやすい作業
- 作業影響が無い作業
- 開発規模が小さいもの

Step.1
- 運用メンバ数名ではじめる
- 影響の少ない作業から自動化
- ソフトウェアを小さく作る
- 社内での実績と協力者を作る

運用課題(中)
- 改善効果が中程度の作業
- 作業影響があるが小~中程度
  もしくは手動復旧できる作業
- 開発規模が中程度のもの

Step.2
- 開発専任メンバーを立てる
  - 運用メンバから抜擢する
  - 他チームから連れてくる
  - 社外から連れてくる
- ソフトウェアを疎結合に作る
- ドキュメントを残す
- 組織の理解と信頼を得る

運用課題(大)
- 改善効果が大きい作業
- 作業影響が大きい作業
- 開発規模が大きいもの
- 継続的に開発が必要なもの

Step.3
- 開発チームを立ち上げる
- 運用性・可用性・スケール
  を考慮したシステムを構築する
- 運用と開発が密に連携しながら
  課題解決に継続的に取り組む
- 組織として、システムと開発体制
  を維持・安定させる

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

本発表のねらい

- ネットワーク運用現場のエンジニア自身が「明日から」自動化開発を始めるための手段を共有。
    - 具体的なプログラミングテクニック
    - 開発に便利なOSSライブラリ・ツール
    - 自動化サンプルコード
        - 本発表に登場するすべてのコードはGithubにて公開しています。 https://github.com/taijiji/NetworkAutomationTutorial
- 本発表での知見を、皆様の職場に持ち帰っていただき、運用改善のための議論をしていただきたいです。
    - 今の運用業務のなかで「自動化」「効率化」「省力化」ができるところはないか？
    - 「どの業務を」「どの手段で」「誰が 」「どこから」着手できそうか？
    - 組織・チームとしてサポートできることはないか？

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク運用自動化の一例

- ネットワーク装置の設定の自動化
  - インタフェース, ACL, BGP, Route Policy, MPLS LSP
- 作業手順書における作成工程の自動化・省力化
- 監視ツール運用の自動化・省力化
- リソース情報管理のDB化・脱Excel化
- ネットワークトポロジー管理の見える化・省力化
- 大量のネットワーク装置へのコマンド同時実行・情報取得
- ネットワーク障害の自動検知/自動復旧
- ネットワーク装置の機能強化
  - CLI、API、Routing機能などの
    自社運用に最適な形にカスタマイズ

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク自動化 失敗パターン by Kirk Byers

1. ハイリスクな問題、難しい問題から手をつける 。
2. All or Nothingなマインドセットで考える(全工程自動化 vs 全手動)
3. すべてを自分自身で再発明しようとする。
4. 理解なしで成功パターン&コードの表面的なコピーで済ませようとする
5. 良いデバッグ方法を学ばない。
6. Over-Engineeringな方法で解決しようとする。
   (不必要なことまで技術で解決しようとする)
7. 小さなスケールから学ばない。
8. 忙しすぎて自動化する時間が無い。
9. コードの再利用の方法を学ばない。 (長期的な観点)
10. 有用な開発者ツールを使わない。 (長期的な観点)
    Git, 文法チェッカー, Unit Test, CIツールなど。

低リスクな問題、
簡単な問題からはじめる！

先人の知恵であるライブラリや
フレームワークを活用！

不必要な工程なのでは？
コードを書くこと無く、
工程そのものを無くせないか?

NANOG71:Network Automation: past, present, and future (Slide, Movie)

参考:「運用自動化」の後に陥る危機

6. まとめ

まとめ: 運用自動化 3つの不都合な真実

- 真実1: 自動化は硬直化
  - 標準化(自動化)は、業務を硬直化させる
  - お役所仕事しかできない低付加価値な運用現場へ
- 真実2: 身近から始めるとキメラ化
  - 「やれるところから自動化」は弊害を生む
  - 保守も変更もできないモノリシックな自動化群の誕生
- 真実3: 自動化の寿命は短い
  - 自動化の寿命は、期待したよりもずっと短い
  - 資産と思って頑張った自動化が、あっという間に負債化(無価値化)

Operation Lab 運用設計ラボ 40

6. まとめ

まとめ: 運用自動化への指針

- 対応1: 自動化は使い捨て
  - 自動化の寿命は短いものと考え、手短に作る
  - 陳腐化したらバッサリ捨てて、次を作る
- 対応2: 外側から自動化
  - 原則として、自動化は可能な限り外側からやる。
  - 自動化には重要なのは、外部影響を把握&制御すること。
- 対応3: 小さく自動化
  - 疎結合・分散型の小さなツールを作る
  - 「ミス低減」と「付加価値増加」の両方を自動化で実現する

Operation Lab 運用設計ラボ 41

運用設計ラボ 波田野 裕一さん発表資料「 運用自動化、不都合な真実 」より引用
https://speakerdeck.com/opelab/20171212-automation

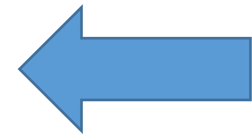
Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

目次

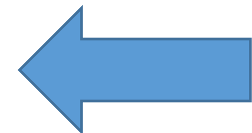
本発表では、プログラミング言語としてPython(3系)を例にします。

- Python 基礎編
  - print
  - 変数 (数値型、文字列型、リスト型、辞書型)
  - if
  - for
  - ファイル入出力
  - 関数
  - クラス
- Python 応用編
  - ライブラリ
  - JSON
  - テンプレートエンジン
  - 正規表現
  - ユニットテスト
- ネットワーク自動化開発に便利なOSSライブラリ/ツールの紹介
- やってみようネットワーク自動化
  - プログラムからルータにログイン、showコマンドを実行
  - プログラムからルータにコンフィグ投入
- 自動化サンプル: BGP Peering作業を自動化する

事前オンライン発表の範囲



JANOG41発表の範囲(P.46以降)



(参考)本発表の理解を深めるツール

- janog41-auto-taiji
- 本発表で紹介するサンプルコードを
  Webブラウザ上で実行することができます。
- ソースコードの変更->実行ができるので、
  試行錯誤しながら理解していくことができます。
- 本ツールは長久 勝 さん (国立情報学研究所クラウド基盤研究開発センター)
  によって開発していただきました。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

なぜPython？

- コードの記述量が少なく、書きやすく読みやすいため
  プログラミング初心者が習得しやすい。
- Webアプリ、サーバサイド、機械学習やデータ解析
  などの分野で、世界中で人気と実績がある。
  (多用途で使えるので、覚えておいて損はしない。)
- ネットワーク自動化系をはじめ、
  多くのライブラリやフレームワークが用意されている。
  (自身でゼロからコードを書く必要が無い。)

(※) 必ずしもPythonを使う必要はなく、
チームメンバー共通で既に使い慣れた
プログラミング言語があればそちらでもOKです。
ただし開発対象の関連ライブラリの有無については
開発スピードに直結するので要事前調査。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Python 2系 vs 3系

- Python2系(最新 2.7.14)と3系(最新 3.6.3)は互換性なし。書式も差異あり。
- Python2系は「2.7」が最後のメジャーリリースであると2010年にアナウンス。
- 一部ライブラリで、Python2系しか対応してないケースあり。(2018年現在では、大半のライブラリは両対応済)
- 一部ライブラリで、Python2系のサポートを2018年頃から終了するアナウンスあり。

Python2系->3系の書き換えは骨が折れるので今から開発をはじめるのであればPython3系がおすすめです。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Python バージョンの調べ方

```
$ python --version

Python 2.7.14
```

デフォルトで動作している
Python バージョン
(OSによって異なる)

```
$ python2 --version

Python 2.7.14
```

```
$ python3 --version

Python 3.6.3
```

OS環境によっては、
"python2"や"python3", "python3.6"
のコマンドが
デフォルト設定されている場合あり。

本発表では、“python3” コマンドを利用します。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Python3.6 インストール

- OSごとのインストール例
  - Windows 10 の場合
    - https://www.python.org/downloads/
  - Mac OS X (macOS) の場合
    - https://qiita.com/ms-rock/items/6e4498a5963f3d9c4a67
  - CentOS 7 の場合
    - https://qiita.com/estaro/items/5a9e4eb8f0902b9977e3
- バージョンがうまく切り替わらない場合は、環境変数PATHの設定変更が必要。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Pythonプログラムの作り方

- プログラム作成
  - 拡張子「.py」をつけたファイルを作成。
  - 入れ子構造は「:」とインデント(スペース4文字 or タブ文字) で表現。
  - 他プログラム言語での、入れ子を示す「{}」や文末の「;」などは使わない。

```
$ vi sample.py

num = 1
if num == 1:
    print(“Hello JANOG!!!”)
```

スペース4文字

- プログラム実行
  - 前述のpythonコマンドで、pythonファイルを指定して実行する。

```
$ python3 sample.py

Hello JANOG!!!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Python 基礎編

- print： 標準出力
- 変数
  - 数値型
  - 文字列型
  - リスト型
  - 辞書型
- if： 条件分岐処理
- for :繰り返し処理
- ファイル入出力
- 関数
- クラス

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

print: 標準出力

プログラム本文

sample_print.py

```
print("Hello JANOG!")
```

プログラム実行結果

プログラム実行結果

```
$ python3 sample_print.py

Hello JANOG!
```

変数 (数値型)

[書式] 変数名 = 数字

プログラム本文

sample_number.py

```
a = 1   # a: int型として生成
b = 1.1 # b: float型として生成
print(a)
print(b)
print('-----')
c = a + b # c: float型として生成
print(c)
```

プログラム実行結果

```
$ python3 sample_number.py

1
1.1
-----
2.1
```

型の指定(C言語でいう「int a;」「float b;」など)は必須ではなく、自動で型が判定されます。(動的型付け)

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

変数(文字列型)

[書式] 変数名 = ”文字列”

プログラム本文

sample_string.py

```
c = “text” # 「‘」「“」のどちらでも可。特別な違いはなし。
d = "テキスト"
#日本語文字列について
#Python3ではデフォルトでUnicode文字列として扱われる。
#Python2ではデフォルトはASCIIであるため「u"テキスト"」と明示する必要あり。

print(c)
print(d)
```

プログラム実行結果

```
$ python3 sample_string.py

text
テキスト
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

変数(リスト型)

[書式]

変数名 = [変数1, 変数2, …]

sample_list.py

```
router_list = ["router_A", "router_B", "router_C"]

# リスト型変数を出力
print(router_list)

# 0から数えて、1番目の値を出力
print(router_list[1])

# 配列の長さを取得(len)
print(len(router_list))
```

プログラム実行結果

```
$ python3 sample_list.py

['router_A', 'router_B', 'router_C']
router_B
3
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

変数(辞書型)

[書式]
変数名 = { キー1:バリュー1, キー2:バリュー2, … }

sample_dictionary.py

```
# 「key」と「value」のペアを変数として定義
router_info = {
    "hostname" : "router_A",
    "os"       : "junos",
    "version"  : "15.1",
}

# ホスト名を表示
print(router_info["hostname"])
# バージョンを表示
print(router_info["version"])
```

プログラム実行結果

```
$ python3 sample_dictionary.py

router_A
15.1
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

if：条件分岐

変数 num の値に応じて、処理を変更する例

sample_if.py

```
num = 2

if num == 1:
    print("num is one")
elif num == 2:
    print("num is two")
else:
    print("num is the others")
```

プログラム実行結果

```
$ python3 sample_if.py

num is two
```

※Pythonには、他プログラミング言語でいうcase文は存在しない。
　すべてif文で表現。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

for: 繰り返し処理

ある処理を、3回分繰り返す例

sample_for_1.py

```
# 回数を指定して繰り返し処理を実施
# range(3) = 0,1,2が格納
for num in range(3):
    print(num)
```

プログラム実行結果

```
$ python3 sample_for_1.py

0
1
2
```

ある処理を、リスト変数の数だけ繰り返す例

sample_for_2.py

```
# リスト型変数を利用して繰り返し処理を実施
router_list = [”router_A”, ”router_B”]

# router_listの値を、順次 router_name に代入
for router_name in router_list:
    print(router_name)
```

プログラム実行結果

```
$ python3 sample_for_2.py

router_A
router_B
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ファイル入出力

ファイル読み込み

[ファイル名: sample_read.txt]

```
Hello JANOG! from sample_read.txt
How are you feeling?
```

ファイル読み込み

sample_file.py

```
# 読み込みモード:"r"
file1 = open("sample_read.txt", "r")
text = file1.read() # ファイル全文を文字列として読み込み
print(text)
file1.close()
```

プログラム実行結果

```
$ python3 sample_file.py

Hello JANOG! from sample_read.txt
How are you feeling?
```

ファイル書き込み

```
# 上書き書き込みモード:"w", 追記書き込みモード:"a"
file2 = open("sample_write.txt", "w")
file2.write("Good!")
file2.close()
```

ファイル書き込み

[ファイル名: sample_write.txt]

```
Good!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

関数とは

- 関数 = 機能をひとまとまりにしたもの。
  - 例: IPv4アドレスの計算をする関数
  - 例: 各ルータOSに応じたコマンドを取得する関数
- プログラムの中で汎用的な処理を関数にして使い回すことで全体のプログラム記述量を減らすことができる。

[書式]

```
def 関数名(引数…)
    関数の処理
    return 戻り値 # 戻り値が無い場合は記述不要


# 関数の呼び出し
関数名()
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

関数 記述例

プログラム本文

sample_function.py

```
# 関数の定義
# ルータ情報を受けとり、
# OSに応じたshow bgp summaryコマンドを返す関数
def get_show_bgp_summary(router_info):
    if router_info["os"] == "junos":
        command = "show bgp summary"
    elif router_info['os'] == 'ios':
        command = "show ip bgp summary"
    else:
        command = “N/A”
    return command

# 辞書型変数
router_info = {
    "hostname"  : "router_A",
    "os"        : "junos"
}
# router_infoを引数として、関数を呼び出し
command = get_show_bgp_summary(router_info)
print(command)
```

プログラム実行結果

```
$ python3 sample_function.py

show bgp summary
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

クラスとは

- クラス = 変数や関数の定義をひとまとまりにした雛形。
- インスタンス = 雛形を元に作成するオブジェクト。
  クラスで定義した機能をもつ。
- 例: ルータクラス
  - 変数: ホスト名, OS名、version名, IPアドレス…
  - 関数: ルータ実機への接続機能、showコマンド実行機能…

[書式]

```
class クラス名:
    クラス変数
    def 関数名(引数…)
        関数の処理
        return 戻り値

変数名 = クラス名() # インスタンスの定義
変数名.クラス変数 # クラス変数の呼び出し
変数名.関数名()    # 関数の呼び出し
```

クラス 記述例

プログラム本文

sample_class.py

```
class Router:
    # 初期化関数: インスタンス生成時に必ず呼ばれる関数。
    def __init__(self, hostname, os, version):
        # インスタンス変数: 各インスタンスで利用する変数。
        self.hostname = hostname
        self.os = os
        self.version = version

    # show bgp summaryコマンドを返す関数。
    def get_show_bgp_summary(self):
        if self.os == "junos":
            command = "show bgp summary"
        elif self.os == 'ios':
            command = "show ip bgp summary"
        else:
            command = 'N/A'
        return command

# Routerクラスのインスタンスを生成
router_A = Router(hostname="router_A", os="junos", version="15.1")
router_B = Router(hostname="router_B", os="ios", version="15.7(3)M")

# それぞれのインスタンスでRouterクラスで定義された関数を呼び出し
print(router_A.hostname)
command = router_A.get_show_bgp_summary()
print(command)

print(router_B.hostname)
command = router_B.get_show_bgp_summary()
print(command)
```

プログラム実行結果

```
$ python3.6 sample_class.py

router_A
show bgp summary
router_B
show ip bgp summary
```

Python 応用編

- ライブラリの利用
- JSONファイルの利用
- テンプレートエンジンの利用
- 正規表現の利用
- ユニットテストの利用

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Pythonライブラリ

第三者によって開発されたプログラムを
再利用するための仕組み。

- 標準ライブラリ
  - デフォルトでインストールされているライブラリ
  - 例: datetime, ipaddress, json

- 外部ライブラリ
  - コミュニティによって作成されたライブラリ
  - パッケージ管理システム「pip」を利用した
    ライブラリのインストールが必要
  - 例: jinja2, django, napalm

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

pip の利用例

外部ライブラリのインストール
(jinja2ライブラリをインストールする例)

```
$ pip3 install jinja2
```

インストール済ライブラリの一覧表示

```
$ pip3 list

Jinja2 (2.9.6)
. . .
```

(※) 応用編として、venvやvirtualenv等を利用することで
アプリ開発環境ごとにpip環境を分ける方法もあります。
(アプリと必要ライブラリを明確にする意図など)
https://docs.python.jp/3/library/venv.html

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

pip のバージョンの調べ方

```
$ pip --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages (python 2.7)
```

デフォルトで動作している
pip バージョン。
対応Pythonバージョンに注目。

```
$ pip2 --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages (python 2.7)
```

```
$ pip3 --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/3.6/lib/python3.6/site-packages (python 3.6)
```

OS環境によっては、
”pip2”や”pip 3”のコマンドが
標準設定されている場合あり。

python3 –m pip コマンドでも
代替可。

(※) 参考: pip インストール方法
Pythonで一番最初に入れるべきパッケージ setuptools と pip
http://www.lifewithpython.com/2012/11/Python-package-setuptools-pip.html

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ライブラリの利用例

datetimeライブラリにより現在時刻を表示

sample_library.py

```
# datetime ライブラリのインポート
import datetime

# datetimeライブラリの、datetimeクラスのnow関数を実行。
now = datetime.datetime.now()
print(now)
```

プログラム実行結果

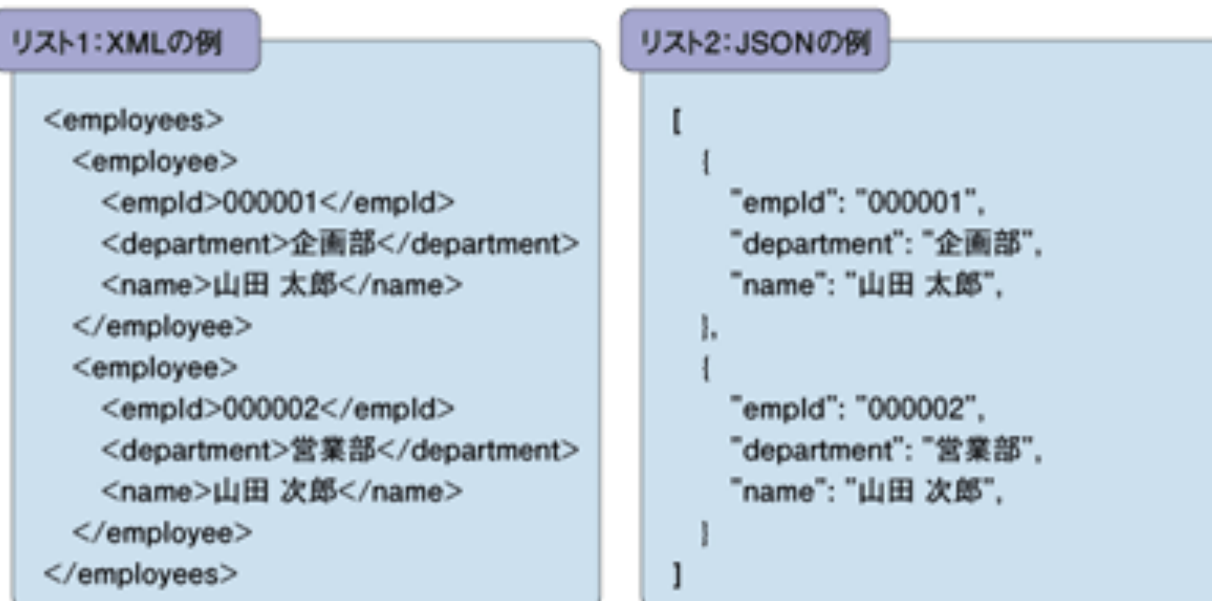
```
$ python3 sample_library.py

2017-11-15 14:25:56.360603
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

JSONとは

データフォーマット(記法)の一つ。
プログラム間のデータやりとりやAPIで多く利用される。
従来のXMLと比較して、人間にもプログラムにも読みやすい利点がある。



参考: JSONってなにもの？
https://thinkit.co.jp/article/70/1

JSONの利用

JSON形式ファイル
(この時点ではただのテキストデータ)

sample_json.json

```
[
  {
   "router_name": "Router_A",
   "ip": "192.168.0.1",
   "os": "junos"
  },
  {
   "router_name": "Router_B",
   "ip": "192.168.0.2",
   "os": "ios"
  },
  {
   "router_name": "Router_C",
   "ip": "192.168.0.3",
   "os": "iosxr"
  }
]
```

Jsonファイルを読み込み、Python変数に変換
(この例では、辞書型変数を格納したリスト型変数 に変換)

sample_json.py

```
import json

file = open("sample_json.json", "r")

# fileオブジェクトからjsonを読み込み、変数に格納
routers_list = json.load(file)

print(routers_list)
print("-----")
print(routers_list[0]["ip"])
```

プログラム実行結果

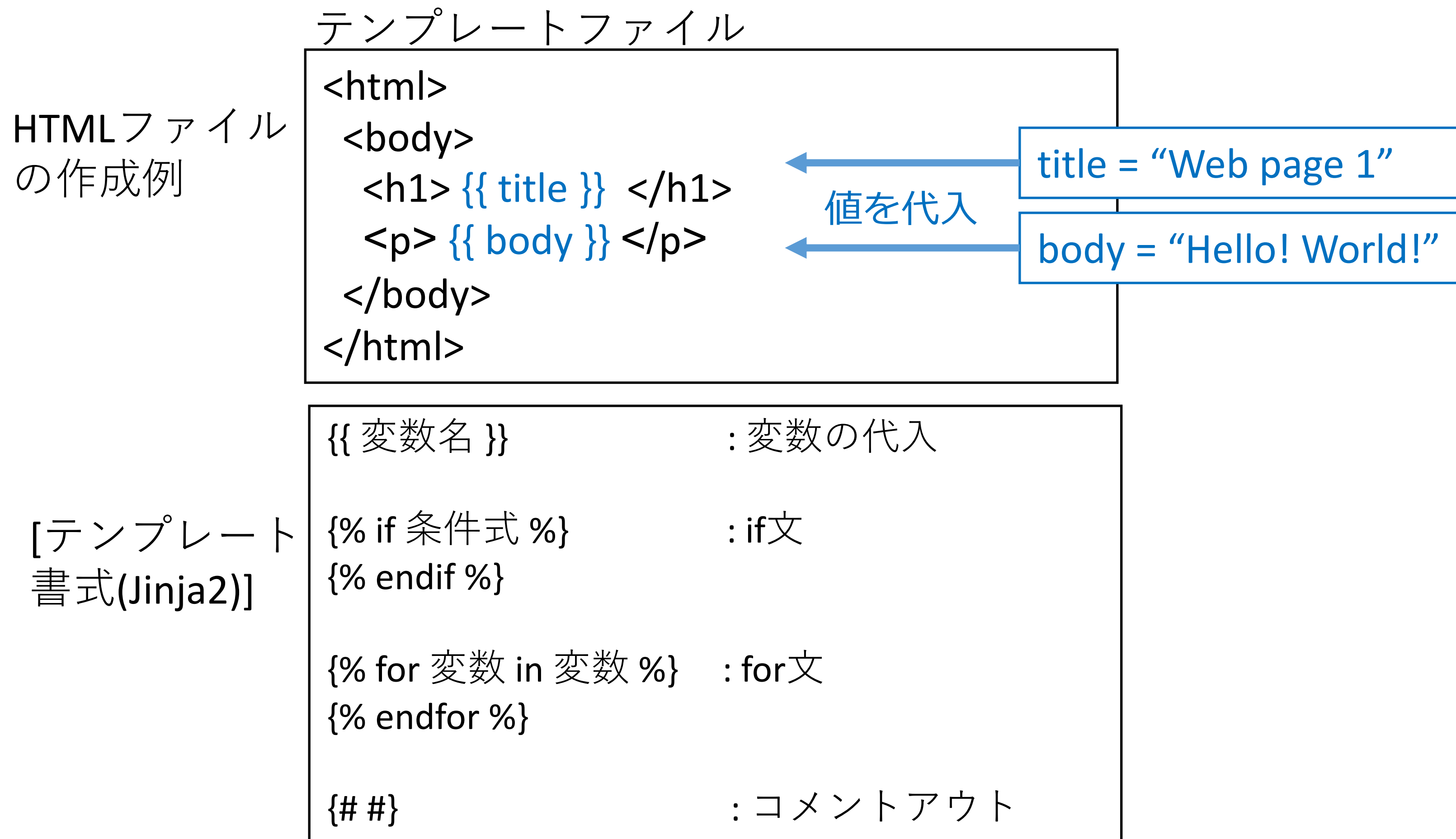
```
$ python3 sample_json.py

[{'router_name': 'Router_A', 'ip': '192.168.0.1', 'os': 'junos'},
{'router_name': 'Router_B', 'ip': '192.168.0.2', 'os': 'ios'},
{'router_name': 'Router_C', 'ip': '192.168.0.3', 'os': 'iosxr'}]
-----
192.168.0.1
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

テンプレートエンジンとは

テンプレートファイル(テキストの雛形)とデータモデルや変数を組み合わせて新たなテキストファイルを自動作成するモジュール。
PythonではテンプレートエンジンとしてJinja2が利用されることが多い。



テンプレートエンジンの利用

テンプレートファイル(Jinja2形式)

sample_template.jinja2

```
interface {{ if_name }}
 description {{ if_description }}
 ip address {{ ip4 }} {{ ip4_subnet }}
 duplex auto
 speed auto
 no shutdown
!
```

プログラム本文

sample_template_1.py

```
import jinja2

# ファイルの読み込み
file = open("sample_template.jinja2", "r")
template_txt = file.read()

# テンプレートオブジェクトの作成
template = \
jinja2.Environment().from_string(template_txt)

# テンプレートに値を代入
config_txt = template.render(
    if_description="To_RouterA",
    if_name="fastethernet 1/1",
    ip4="192.168.0.1",
    ip4_subnet="255.255.255.0")

print(config_txt)
```

プログラム実行結果

```
$ python3 sample_template_1.py

interface fastethernet 1/1
 description To_RouterA
 ip address 192.168.0.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

テンプレートエンジンの利用(複数作成)

テンプレートファイル(Jinja2形式)

sample_template.jinja2

```
interface {{ if_name }}
 description {{ if_description }}
 ip address {{ ip4 }} {{ ip4_subnet }}
 duplex auto
 speed auto
 no shutdown
!
```

sample_template_2.py 抜粋

```
interfaces = [
    {
        "if_description" : "To_RouterA",
        "if_name"        : "fastethernet 1/1",
        "ip4"            : "192.168.0.1",
        "ip4_subnet"     : "255.255.255.0"
    },
    {
        "if_description" : "To_RouterB",
        "if_name"        : "fastethernet 1/2",
        "ip4"            : "192.168.1.1",
        "ip4_subnet"     : "255.255.255.0"
    },
    {
        "if_description" : "To_RouterC",
        "if_name"        : "fastethernet 1/3",
        "ip4"            : "192.168.2.1",
        "ip4_subnet"     : "255.255.255.0"
    }
]

for interface in interfaces:
    config_txt = template.render(interface)
    print(config_txt)
```

プログラム実行結果

```
$ python3 sample_template_2.py

interface fastethernet 1/1
 description To_RouterA
 ip address 192.168.0.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface fastethernet 1/2
 description To_RouterB
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface fastethernet 1/3
 description To_RouterC
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
```

正規表現

パターン文字列を利用して、
文字列の集合の中から、対象の文字列を抽出する方法。

```
text    = “Today is Sunday”

pettern = ”Today is (.+)”
```

「()」：
グループ化

「+」：
直前の正規表現に対して
１回以上の繰り返し

「.」：
改行以外の
任意の文字列1文字

```
match = re.search(pettern, text )
# マッチした行全体を取得
match.group(0) # =>「Today is Sunday」
# マッチしたグループを取得
match.group(1) # =>「Sunday」
```

正規表現の利用

sample_regex.py

```
import re

# 複数行の文字列
# show versionコマンド実行結果を想定
show_version_txt = """
Hostname: vsrx
Model: firefly-perimeter
JUNOS Software Release 12.1X47-D15.4
"""

# 正規表現
# 「()」: グループ化
# 「.」 : 改行以外の任意の文字列1文字
# 「+」 : 直前の正規表現に対して、1回以上の繰り返し
regex = "JUNOS Software Release (.+)"

# show_version_txtの文字列に対して、
# 正規表現による文字列の抽出を実行
match = re.search(regex, show_version_txt )

# マッチした文字文字列全体を表示
print(match.group(0))
print("--------------------")

# マッチした文字列のうちグループ化された部分を抽出
print(match.group(1))
```

プログラム実行結果

```
$ python3 sample_regex.py

JUNOS Software Release 12.1X47-D15.4
--------------------
12.1X47-D15.4
```

ユニットテストとは

- ソフトウェアにおける単体テストを実施するためのモジュール。
- １つの関数に対して「入力値」と「期待する出力値」を定義することで、関数の実装の誤りを検出する。
- テストケースを書くメリット:
  - バグの早期発見が可能。
  - 関数単位で動作確認できるため、プログラム全体を動作させることなく、実装->テスト->修正->テスト->…を素早く実施することが可能。
  - 関数の振る舞いを明記することで、コードの変更・改良が容易になる。第三者もコードの意図を理解しやすくなる。
- テストケースを書くデメリット:
  - テストケースを書くことは、常に工数とのトレードオフの関係にある。
  - 異常系テストケースをどこまで実装すべきかは、プロジェクトの段階や開発対象機能に応じて検討する。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ユニットテスト 記述方法

関数の例

```
def hello(num):
    if num == 1:
        message = “Hello”
    elif num == 2:
        message = “Good bye”
    else:
        message = “No message”

    return message
```

テストケースの一例

```
# hello(1) のときは、
# 戻り値が”Hello”であるべき。
assertEqual(hello(1), “Hello”)
# hello(2) のときは、
# 戻り値が“Good bye”であるべき。
assertEqual(hello(2), “Good bye”)
# hello(-1)のときは、
# 戻り値が”No message”であるべき
assertEqual(hello(-1), “No message”)
# hello(“abc”) のときは、
# 戻り値が”No message”であるべき
assertEqual(hello(“abc”), “No message”)
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ユニットテスト 利用例

テスト対象のプログラム

sample_unittest.py

```
def add(x, y):
    sum = x + y
    return sum
```

ユニットテストのプログラム

tests/test_sample_unittest.py

```
import unittest
import sample_unittest

class TestSample(unittest.TestCase):
    def test_add(self):
        actual = sample_unittest.add(1,2)
        expected = 3
        self.assertEqual(actual, expected)

if __name__ == "__main__":
    unittest.main()
```

ファイル構成

```
├── sample_unittest.py
└── tests
    └── test_sample_unittest.py
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ユニットテスト 実行例

ユニットテストの実行(成功例)

```
python3 -m unittest tests.test_sample_unittest
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

ユニットテストの実行(失敗例)

```
python3 -m unittest tests.test_sample_unittest
F
======================================================================
FAIL: test_add (tests.test_sample_unittest.TestSample)
----------------------------------------------------------------------
Traceback (most recent call last):
  File ”xxx/tests/test_sample_unittest.py", line 12, in test_add
    self.assertEqual(expected, actual)
AssertionError: 3 != 4

----------------------------------------------------------------------
Ran 1 test in 0.000s
```

関数かテストケースのいずれかが間違っている可能性あり

(※) 「python3 –m unittest」と入力することで全テストをまとめて実行することも可能です。
その場合、testディレクトリ内に空ファイル「__init__.py」を設置する必要があります。

その他の開発に関連する便利ツール

- Excelを操作するPythonライブラリ
  - OpenPyXL
- Webアプリケーションを開発するためのPythonフレームワーク
  - Django, Flask
- プログラムのバージョン管理システム
  - Git, GitHub, GitLab
- 継続的インテグレーションシステム
  - Jenkins, CircleCI

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク自動化開発に便利な OSSライブラリ/ツール の紹介

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Tips: 自動化開発のための仮想ルータ環境の準備

- 各ベンダの仮想ルータを利用できるWebサービス
  - NetworkToCode On Demand Labs (有償)
- ローカル環境でGUIで
  仮想ルータ構築+トポロジー作成ができるソフトウェア
  - Cisco CML, VIRL (有償) (Cisco以外の仮想ルータイメージは別途必要)
  - EVE-NG (無償) (仮想ルータイメージは別途必要)
- ローカル環境でCLIで
  仮想ルータ構築ができるソフトウェア
  - Vagrant + Juniper vSRX (無償)
    - (公式Github) Vagrant Junos Guest
    - (個人ブログ) Vagrantでfireflyを動かしたら自動化開発が捗った話
  - Vagrant + Cisco IOSXRv (無償)
    - (公式ブログ) XR toolbox, Part 1 : IOS-XR Vagrant Quick Start
    - (個人ブログ) IOS-XRv Vagrantを試してみた
  - Vagrant + Arista vEOS (無償)
    - (公式ブログ) Using vEOS with Vagrant and VirtualBox

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク設定自動化の手段

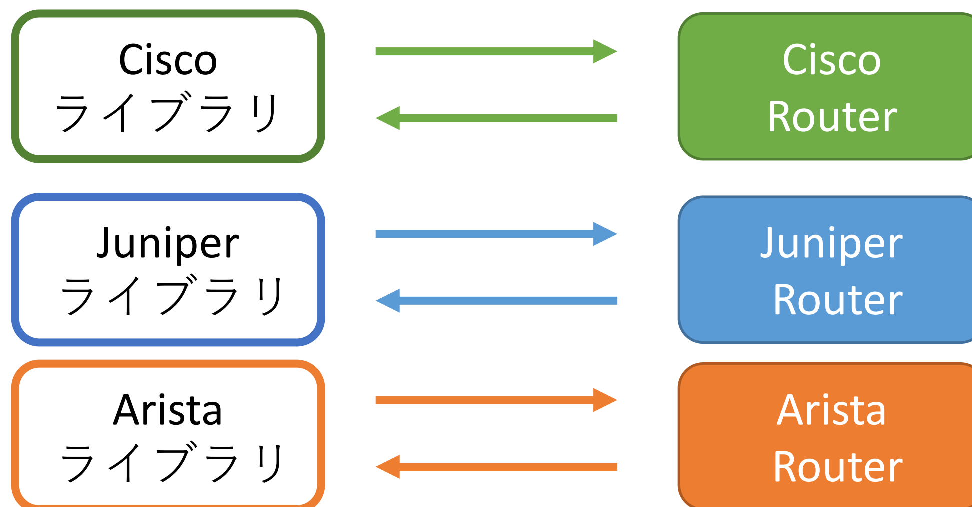
- 装置メーカ提供ライブラリを利用
- SSH/TelnetライブラリでCLIコマンドを送信
- マルチベンダ対応 ライブラリを利用
- マルチベンダ対応 構成管理ツールを利用

自動化しようとしている運用作業における
対象機種/対象機能/ライブラリ有無/開発体制/運用体制
などを加味した上で上記手段から選択する必要があります。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク設定自動化:
装置メーカ公式ライブラリを利用

- 各メーカによって開発されているため、
  機能が豊富。継続的なメンテナンスも実施。
- メーカ・OSによって対象機能や制御方法が大きく異なる。
- 機種を限定した自動化システムであれば非常に有効。
  (例: 自動化のために１機種に統一)



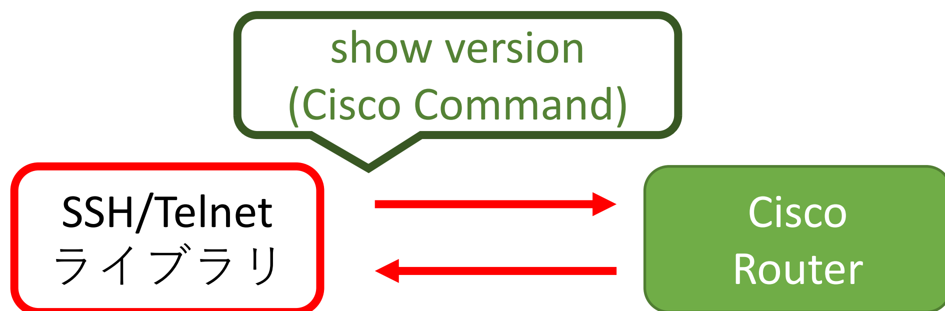
装置メーカ提供ライブラリの一例

Manufacturer	OS	Library	Language	OS version
Cisco	IOS / IOS-XE	One Platform Kit (OnePK)	C, Java, Python	ASR1000: 3.12 -3.17(EoL)
		YANG Development Kit(YDK)	Python, C++	16.5.1 later
	IOS-XR	One Platform Kit (OnePK)	C, Java, Python	ASR900: 5.1.2 - 5.3.3(EoL) ASR9000: 5.1.1 - 5.3.3(EoL)
		YANG Development Kit(YDK)	Python, C++	6.0.0 later
	NX-OS	One Platform Kit (OnePK)	C, Java, Python	Nexus9000: all version Nexus6000: - 7.2(EoL)
		NX-API	Web GUI, HTTP/HTTPS	Nexus9000: all version Nexus7000: 7.2(0)D1(1) later Nesus6000: 7.2(0)N1(1) later
Juniper	JUNOS	PyEZ	Python	11.4 later
		Juniper Extension Toolkit(JET)	Python	16.2 later
Brocade	Network OS	PyNOS	Python	5.0.1 later
Arista	EOS	pyeapi, rbeapi, goeapi	Python, Ruby, Go	4.1.2 later
		EOS SDK	C++, Python	all version

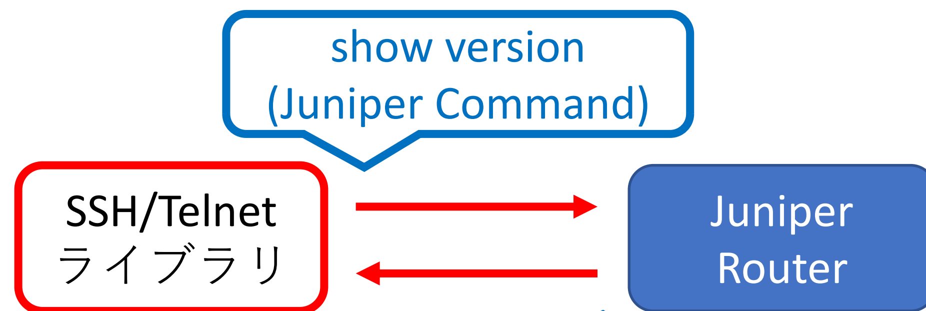
※厳密にはハードウェア機種によって対象OS, versionは異なります。詳細情報は各メーカ担当者にご質問ください。

ネットワーク設定自動化: SSH/TelnetライブラリでCLIコマンドを送信

- 手動設定と同様のコマンドを、SSH/Telnetライブラリ経由で送信。
- CLIを保有するほぼすべての機器に設定可能。
- 機器固有コマンドや正規表現を実装する必要あり。(コード量が多くなる)
- 例: Exscript , pexpect



⇒ 正規表現で「6.2.1.231」を抽出



⇒ 正規表現で「12.1X47-D15.4」を抽出

SSHライブラリ Exscriptで表示コマンド実行

sample_exscript_show.py

```
import Exscript # SSHライブラリ(要:pip3 install exscript)
import re       # 正規表現ライブラリ

username = "user1"
password = "password1"
ip4 = "192.168.33.3"

# SSHセッションの確立
session = Exscript.protocols.SSH2()
session.connect(ip4)

# ルータにログイン
account = Exscript.Account(name=username, password=password)
session.login(account)

# ルータにコマンドを送信、出力結果を取得
session.execute("show version")
result = session.response
print(result)
print("----------")

# 正規表現で情報抽出
pattern = "JUNOS Software Release \[(.+)\]"
match = re.search(pattern, result)
version = match.group(1)
print(version)

# SSHセッションの切断
session.send("exit")
session.close()
```

実行結果

```
$ python3 sample_exscript_show.py

show version
Hostname: vsrx
Model: firefly-perimeter
JUNOS Software Release [12.1X47-D15.4]
----------
12.1X47-D15.4
```

機種固有のコード
(CLIコマンド, 正規表現)

SSHライブラリ Exscriptで設定投入(一部抜粋)

sample_exscript_set.py(抜粋)

```
print("===== Step 1. run show command =====")
session.execute("show configuration interfaces ge-0/0/1")
print(Fore.YELLOW + session.response) # Fore.YELLOW：黄色文字で出力

print("===== Step 2. configure =====")
session.execute("configure")

config_txt = "set interfaces ge-0/0/1 disable"
session.execute(config_txt)
print(Fore.YELLOW + session.response) # 実行結果を黄色文字で出力

print("===== Step 3. commit check =====")
session.execute("show | compare")
print(Fore.YELLOW + session.response)
session.execute("commit check")
print(Fore.YELLOW + session.response)

print("===== Step 4. commit =====")
print("Do you commit? y/n") # ユーザにy or nを質問
choice = input()
if choice == "y":
    session.execute("commit")
    print(session.response)
else:
    session.execute("rollback")
    print(session.response)

session.execute("exit")
print(session.response)

print("===== Step 5. run show command(again) =====")
session.execute("show configuration interfaces ge-0/0/1")
print(Fore.YELLOW + session.response)
```

大半が
機種固有のコード
(CLI, 設定手順)

実行結果

```
$ python3 sample_exscript_set.py

===== Step 1. run show command =====
show configuration interfaces ge-0/0/1
unit 0 {
    family inet {
        address 10.0.1.1/24;
    }
}
===== Step 2. configure =====
set interfaces ge-0/0/1 disable
===== Step 3. commit check =====
show | compare
[edit interfaces ge-0/0/1]
+   disable;
commit check
configuration check succeeds
===== Step 4. commit =====
Do you commit? y/n
y
commit
commit complete
exit
Exiting configuration mode
===== Step 5. run show command(again) =====
show configuration interfaces ge-0/0/1
disable;
unit 0 {
    family inet {
        address 10.0.1.1/24;
    }
}
```

参考: sample_exscript_set.py 全文

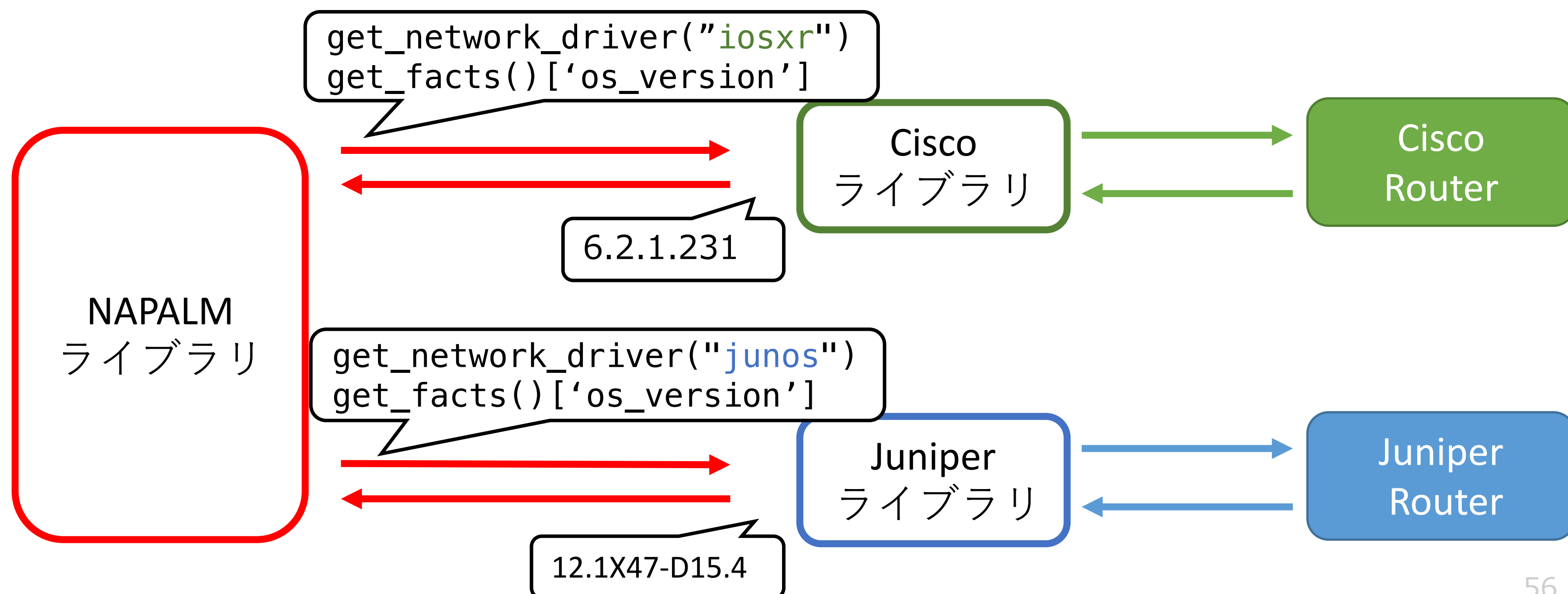
Tips: showコマンド結果を抽出するライブラリ

- TextFSM + NTC-Templates
- 利用事例
  - オペレーション自動化と監視の取り組み(ヤフー 安藤さん)
  - Excel/CSV変換ツールのおかげでshow ip routeのコピペ地獄から解放された話(富士通 岩田さん)

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク設定自動化:
マルチベンダ対応 ライブラリを利用

- 複数ベンダー機器をほぼ同一のプログラムで動かす事が可能。(再利用可能)
- コミュニティ主導によるライブラリ。
  - 主にユーザ企業の有志エンジニアが中心となって開発。
  - 開発コミュニティが小さく、明確なロードマップが無いものも多い。
  - 例: NAPALM , Netmiko
- メーカ製品版のライブラリ(有償)
  - 例: Cisco Network Services Orchestrator, anuta networks NCX



NAPALMとは

- コミュニティ主導で開発されているマルチベンダ対応Pythonライブラリ
  - https://napalm.readthedocs.io/en/latest/index.html
  - https://github.com/napalm-automation/npalm
- 表示機能および設定機能が抽象化されており、ユーザは機種を気にせずに実装することが可能。(ただし設定コンフィグは、既存の機器固有コンフィグを利用)
- 対応機種, 対応機能:
  - Arista EOS 4.15.0F later / Juniper JUNOS 12.1 later / Cisco IOS-XR 5.1.0 later / NXOS 6.1 later / IOS 12.4(20)T later
  - メーカライブラリが無いものは、サードパーティライブラリとして新たに実装。
  - 機種によって、未実装が若干存在。
  - 詳細: http://napalm.readthedocs.io/en/latest/support/index.html#the-transport-argument

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

NAPALMで表示コマンドを実行

sample_napalm_show.py

```
 # NAPALM ライブラリ
 # 要: pip3 install napalm)
import napalm
from pprint import pprint

# JUNOS用インスタンスを生成
driver = napalm.get_network_driver("junos")
device = driver(
    hostname="192.168.33.3",
     username="user1",
     password="password1" )

# コネクションの確立
device.open()

# 基本情報の取得
fact = device.get_facts()
pprint(fact) # pprint: 辞書型変数を見やすく表示

print("----------")

# バージョンの取得(基本情報から抽出)
print(fact["os_version"])
device.close()
```

機種固有のコード
(機種の指定)

実行結果

```
$ python3 sample_napalm_show.py

{'fqdn': 'vsrx',
'hostname': 'vsrx',
'interface_list': [ 'ge-0/0/0',
                    'ge-0/0/1',
                    'ge-0/0/2',
                    . . . ,
                   'vlan'],
'model': 'FIREFLY-PERIMETER',
'os_version': '12.1X47-D15.4',
'serial_number': '83f144ddd4f7',
'uptime': 8309,
'vendor': 'Juniper'}
----------
12.1X47-D15.4
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

NAPALMで設定投入を実行(抜粋)

sample_napalm_set.py(抜粋)

```
・・・

print("===== Step 1. run show command =====")
print("Interface Status: ", end="")
if   device.get_interfaces()["ge-0/0/1"]["is_up"] == True:
    status_before = "UP"
elif device.get_interfaces()["ge-0/0/1"]["is_up"] == False:
    status_before = "DOWN"
print(Fore.YELLOW + status_before)

print("===== Step 2. configure =====")
device.load_merge_candidate(filename="./sample_config_junos.txt")
print("OK")

print("===== Step 3. compare configuration =====")
print(Fore.YELLOW + device.compare_config())

print("===== Step 4. commit =====")
print("Do you commit? y/n")
choice = input()
if choice == "y":
    print("Commit config: ", end="")
    device.commit_config()
    print("OK")
else:
    print("Discard config: ", end="")
    device.discard_config()
    print("OK")

print("===== Step 5. run show command(again) =====")
print("Interface Status: ", end="")
if   device.get_interfaces()["ge-0/0/1"]["is_up"] == True:
    status_after = "UP"
elif device.get_interfaces()["ge-0/0/1"]["is_up"] == False:
    status_after = "DOWN"
print(Fore.YELLOW + status_after)
```

機種固有のコード
(設定コンフィグ)

sample_config_junos.txt

```
interfaces {
   ge-0/0/1 {
      disable;
   }
}
```

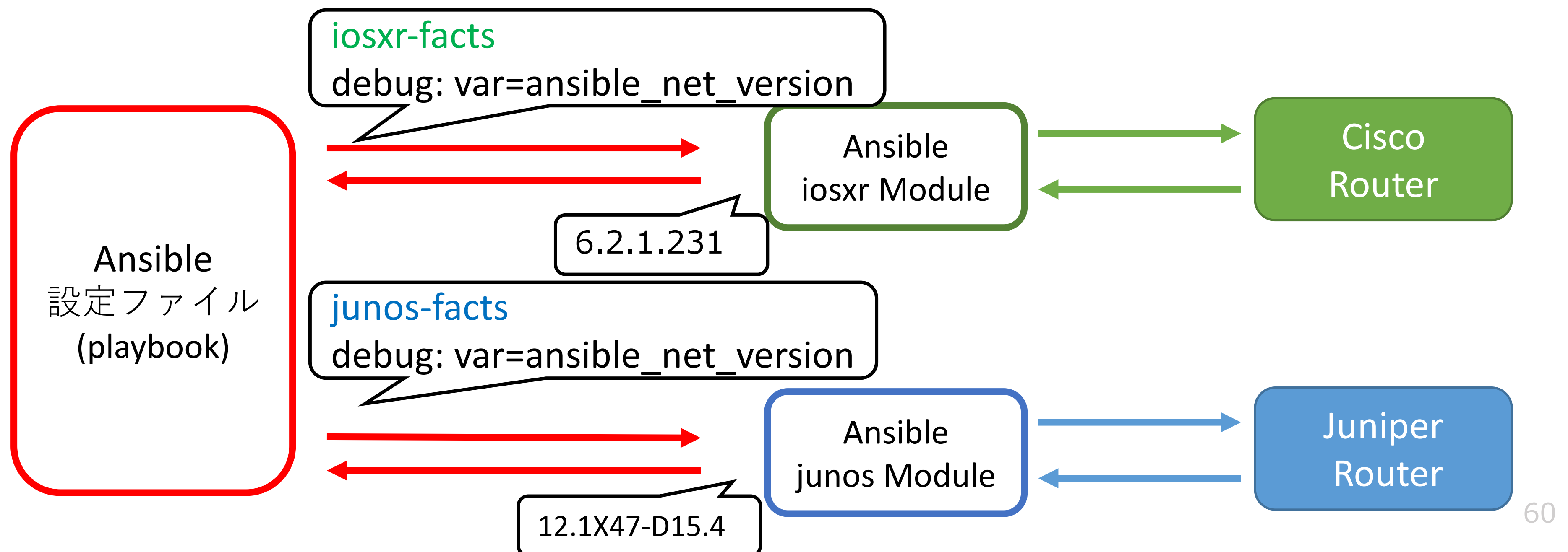
実行結果

```
$ python3 sample_napalm_set.py

===== Step 1. run show command =====
Interface Status: UP
===== Step 2. configure =====
OK
===== Step 3. compare configuration =====
[edit interfaces ge-0/0/1]
+ disable;
===== Step 4. commit =====
Do you commit? y/n
y
Commit config: OK
===== Step 5. run show command(again) =====
Interface Status: DOWN
```

ネットワーク設定自動化:
マルチベンダ対応 構成管理ツールを利用

- コードを書くことなく、自動化を実現することが可能。
  ただし構成管理ツール固有の設定ファイルの記述は必要。
- コミュニティ(主に構成管理ツールメーカおよびNW機器メーカ)による開発が中心で、開発が非常に活発。エンタープライズ版(有償)も存在。
- 自作ツールとは異なり、定められた表示フォーマットに従う必要がある。
- NWメーカ・OSによって機能差分が大きい。実装状況は機種に大きく依存。
- 例: Ansible , SaltStack



Ansibleとは

- サーバ設定自動化のための構成管理ツールとして多く利用され、Network Moduleという形でネットワーク機器への対応も拡大中。
  http://docs.ansible.com/ansible/latest/list_of_network_modules.html
- playbookファイル(YML形式)にて設定内容を記述。
- ネットワーク機器への対応状況:
  - 対応OS
    - IOS, IOSXR, NXOS, JUNOS, EOS, F5, FortiOS, Illumos, Lenovo, Netvisor, Openswitch, Ordnance, Ovs, Panos, Sros, Vyos
  - 対応機能
    - 機種によって実装状況にばらつきあり。
  - Ansible 2.0以降では core module版とGalaxy module版が別々に存在。
- 本発表ではAnisble 2.0 標準モジュールを利用。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Ansibleで表示コマンドを実行(factsで取得)

事前にインストール

```
pip3 install ansible
pip3 install ncclient #対象がjunosの場合
```

インベントリファイル
hosts

```
[vsrx1]
192.168.33.3
[vsrx1:vars]
ansible_ssh_user=user1
ansible_ssh_pass=password1
```

playbookファイル
junos_show_1.yaml

```
- hosts: vsrx1
  connection: local
  gather_facts: no
  tasks:
    - name: Get basic information
      junos_facts:    } 機種固有

    - name: show model name
      debug: var=ansible_net_model

    - name: show serial number
      debug: var=ansible_net_serialnum

    - name: show version
      debug: var=ansible_net_version
```

実行結果≈

```
$ ansible-playbook -i hosts junos_show_1.yaml

PLAY [vsrx1] ************************************

TASK [Get basic information] *********************
ok: [192.168.33.3]

TASK [show model name] ***************************
ok: [192.168.33.3] => {
    "ansible_net_model": "firefly-perimeter"
}

TASK [show serial number] ************************
ok: [192.168.33.3] => {
    "ansible_net_serialnum": "83f144ddd4f7"
}

TASK [show version] ******************************
ok: [192.168.33.3] => {
    "ansible_net_version": null   ← ?
}

PLAY RECAP ***************************************
192.168.33.3   : ok=4 changed=0 unreachable=0 failed=0
```


Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

Ansibleで表示コマンドを実行(CLIで取得)

インベントリファイル hosts

```
[vsrx1]
192.168.33.3
[vsrx1:vars]
ansible_ssh_user=user1
ansible_ssh_pass=password1
```

playbookファイル junos_show_2.yaml

```
- hosts: vsrx1
  connection: local
  gather_facts: no
  tasks:
    - name: send command "show version"
      junos_command:
        commands: show version
      register: result
      # コマンド結果を"result"として保有

    - name: show result
      debug: var=result.stdout_lines
```

機種固有

実行結果

```
$ ansible-playbook -i hosts junos_show_2.yaml

PLAY [vsrx1] ***********************************
TASK [send command "show version"] *****************
ok: [192.168.33.3]

TASK [show result of "show version"] ***************
ok: [192.168.33.3] => {
    "result.stdout_lines": [
        [
            "Hostname: vsrx",
            "Model: firefly-perimeter",
            "JUNOS Software Release [12.1X47-D15.4]"
        ]
    ]
}

PLAY RECAP ****************************************
192.168.33.3 : ok=2 changed=0 unreachable=0 failed=0
```


Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク設定自動化の手段 まとめ

1. 装置メーカ提供ライブラリを利用
   - 機能が豊富。継続的なメンテナンスが実施。
   - メーカ・OSによって対象機能や制御方法が大きく異なる。
   - 対象機種を限定した自動化システムであれば非常に有効。
2. SSH/TelnetライブラリでCLIコマンドを送信
   - CLIを保有するほぼすべての機器に設定可能。
   - 機器固有のコマンドや正規表現を自前で実装する必要あり。コード量が多くなる。
3. マルチベンダ対応 ライブラリを利用
   - 複数ベンダー機器をほぼ同一のプログラムで動かす事が可能。(再利用が可能)
   - コミュニティ(主にユーザ企業の有志のエンジニア)による開発が多いが 開発コミュニティが小さく、明確なロードマップが無いものも多い。
   - メーカ製品ライブラリ(有償)も存在する。
4. マルチベンダ対応 構成管理ツールを利用
   - コード記述ゼロで自動化を実現することが可能。 ただし設定ファイル記述は必要。
   - コミュニティ(主にメーカ企業)による開発が中心で、開発が活発。
   - エンタープライズサポート版(有償)も存在する。
   - 定められた表示フォーマットに従う必要あり。カスタマイズ性が低い。
   - メーカ・OSによって機能差分が大きい。(実装状況はメーカに依存)

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

ネットワーク自動化サンプルコード：NAPALMを使ったBGP Peering作業の自動化

- https://github.com/as2518/napalm-scenario
  - forkして作り直し中 https://github.com/taijiji/napalm-scenario

- 参考資料:
  - NAPALMで作るネットワークオペレーション自動化への道のり
    - 上記ツールの発表資料
  - JSNAPy とPyEZで作る次世代ネットワークオペレーションの可能性
    - NAPALMは使っていないものの、上記アイディアのベースとなったもの

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

まとめ

- 「明日からはじめる」ための
  ネットワーク自動化開発のための手段を共有しました。
  - Python基礎、応用
  - ネットワーク設定自動化のための手段
    - メーカ公式ライブラリ
    - SSHライブラリ
    - マルチベンダ対応 ライブラリ
    - マルチベンダ対応 構成管理ツール

- 自動化の手段を選択する際には、以下の点を念頭において
  議論・検討していただくのが良いかと思います。
  - 「どの作業」を自動化・省力化したいのか。
  - 「どの機種」を自動化したいのか。ライブラリの存在有無。
  - 「誰が」「どの程度」コードを書くか。
  - 「どこから」着手するのがよいか。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.

質疑応答 & 議論

- みなさまから土屋への質問
  - 気になることがあればぜひ。
- 土屋からみなさまへの質問
  - 本発表の項目以外で「この情報があれば社内の自動化開発が進む」といったポイントはありますか？
  - 皆様の会社で「自動化のボトルネック」になっているものはなんでしょうか？
何があれば解決できそうでしょうか？

Challenging Tomorrow's Changes

The corporate logo mark embodies our burning vision “to target more than swift perception of global changes and proper response to market shifts -- aspiring to be a part of inducing those transitions.” Beneath the logo, this desire is expressed in the phrase “Challenging Tomorrow's Changes.”

Copyright © 2018 ITOCHU Techno-Solutions America, Inc.