74th

Shell Scriptの代わりにPythonタスクランナーFabric & Invokeを活用する技術

74th　著

Shell Script の代わりに Python タスクランナー Fabric & Invoke を活用する技術

74th　著

2018-10-08 版　　発行

目次

第 1 章

シェルスクリプト、Make、だんだん黒魔術化してきた

どんな環境でも動作するプログラミング言語として、 Shell Script は 2018 年現在も人気である。そしてそれは今後 10 年もそうであろう。しかし、積極的に Shell Script を書きたい！ と思っている人はどれほどいるのだろうか。Shell Script は便利ではあるが、私は以下の点で扱いにくいと考えている。

- 文字列の操作にも cat 、 sed などのコマンドが必要であり、 Shell Script でタスクを処理させるにはそれらのコマンドを使いこなすことも必要である。
- 多くのエディタの場合、型チェックや書式のチェック機能を持っておらず、動作させたときに初めてエラーやロジックのミスに気づく。
- すべてが文字列表現を介してロジックが動作する。変数の中身が数値であることなどを保証しない。
- Windows 上で満足に動作させるには、Windows Subsystem Linux や Cygwin など別のツールが必要になる。
- オブジェクト指向的な扱いは難しい。

Shell Script はそれ自体が直接的な役割を持つことは少なく、一連のコマンドを呼び出すことが役割である事が多い。そのような、コマンドを呼び出したり、タスクをこなす役割で登場したものには、以下のようなものがある。

- Make ソースコードのビルドをためのタスクランナー。ファイル間の依存関係を解決することに特化している。
- Gulp Javascript のためのタスクランナー。プラグインシステムを持ってお

1.1 Fabric をちょっと覗く

Fabric とは、Jeff Forcier 氏 (@bitprophet) が作成した、Python で動作するタスクランナーである。2018 年 5 月に、Version2 がリリースされ、Python2 に加えて Python3 がサポートされた。現在でも活発に開発が行われている（ただし、@bitprophet による仕事がほとんど)。なお、ライセンスは 2 条項 BSD である。

Fabric は、Invoke というタスクランナーを内包している。役割としては以下のように分かれている。

- ローカル環境でのタスクランナー: Invoke
- SSH 接続したサーバ上で実行するタスクランナー: Fabric

ここでは、まず Invoke を簡単に紹介する。Invoke ではタスクを tasks.py という名前のファイルを作成する必要がある。

たとえば、 Google Compute Engine のインスタンスを削除するスクリプトを Shell Script で実装すると、以下のようになる。

リスト 1.1: delete_instance.sh

```
#!/bin/bash
if [ $# -ne 1 ]; then
   echo "delete_instance.sh <NAME>"
   exit 1
fi
gcloud compute instances delete --quiet $1
```

```
./delete_instance.sh hoge_instance
```

一方、Invoke で記述すると以下のようになる。

リスト 1.2: tasks.py

第 2 章

タスクランナー Invoke の機能解説

Fabric は、タスクランナー Invoke を内包しており、ほとんどの機能は Invoke が提供している。Invoke は、ローカルコンピュータ上で実行されるタスクランナーである。

Fabric と Invoke の違いは以下の通りである。

- Fabric: SSH 接続先のコンピュータ上上でタスクを実行する。接続先の環境変数を引き継いでコマンドを実行する。
- Invoke: ローカルコンピュータ上でタスクを実行する。実行環境の環境変数を引き継いでコマンドを実行する。

SSH 接続を要する用途以外では Invoke の機能で基本的に十分である。

2.1 Quick Start

Invoke は pip からインストールできる。

```
PY3 $ pip3 install invoke
PY2 $ pip install invoke
```

タスクを記述するファイル名は、**tasks.py** である必要がある。

tasks.py には、タスク名で関数を定義する。関数には、@task のデコレータを付ける。

リスト 2.1: tasks.py

```
from invoke import task

@task
def build(c):
    print("Building!")
```

このタスクは、下記のコマンドで実行できる。

```
$ invoke build
Building!
```

ここでタスクの関数の第一引数には invoke.Context が格納される。この invoke.Context の run メソッドでコマンドを実行できる。

Golang のビルドを行う例を示す。

リスト 2.2: tasks.py

```
from invoke import task
```