# Fabric及YAML

|  |  |
| --- | --- |
| 撰稿 | 日期 |
| 张凯 | 2016.11.25 |

目录

[一、YAML概述: 1](#_Toc467157153)

[一）PYYAML基本格式 1](#_Toc467157154)

[二）、使用事项 2](#_Toc467157155)

[二、 fabric 3](#_Toc467157156)

[5、任务参数 4](#_Toc467157157)

[6、环境字典env及运行环境(设置) 5](#_Toc467157158)

[7、直接执行远程命令 5](#_Toc467157159)

[8、fabfile文件及结构使用 6](#_Toc467157160)

[9、连接服务器 6](#_Toc467157161)

[10、标准输出 6](#_Toc467157162)

# 一、YAML概述:

我们使用 YAML 是因为它像 XML 或 JSON 是一种利于人们读写的数据格式. 此外在大多数变成语言中有使用 YAML 的库,  每一个 YAML 文件都是从一个列表开始. 列表中的每一项都是一个键值对, 通常它们被称为一个 “哈希” 或 “字典”. 所以, 我们需要知道如何在 YAML 中编写列表和字典

## 一）PYYAML基本格式

1、YAML 还有一个小的怪癖. 所有的 YAML 文件开始行都应该是 ---. 这是 YAML 格式的一部分, 表明一个文件的开始.

2、列表中的所有***成员***都开始于相同的缩进级别, 并且使用一个 "- " 作为开头(一个横杠和一个空格):

---

# 一个美味水果的列表

- Apple

- Orange

- Strawberry

- Mango

3、一个***字典***是由一个简单的 键: 值 的形式组成(这个冒号后面必须是一个空格):

---

# 一位职工的记录

name: Example Developer

job: Developer

skill: Elite

4、字典也可以使用缩进形式来表示, 如果你喜欢这样的话:

---

# 一位职工的记录

{name: Example Developer, job: Developer, skill: Elite}

5、你可以通过以下格式来指定一个布尔值(true/fase):

---

create\_key: yes

needs\_agent: no

knows\_oop: True

likes\_emacs: TRUE

uses\_cvs: false

6、让我们把目前所学到的 YAML 例子组合在一起，看看效果

---

# 一位职工记录

name: Example Developer

job: Developer

skill: Elite

employed: True

foods:

- Apple

- Orange

- Strawberry

- Mango

languages:

ruby: Elite

python: Elite

dotnet: Lame

## 二）、使用事项

1、使用引号来包裹任何包含冒号的哈希值

foo: somebody said I should put a colon here: so I did 

foo: "somebody said I should put a colon here: so I did" 

2、使用 “{{ var }}” 来引用变量. 如果一个值以 “{” 开头, YAML 将认为它是一个字典, 所以我们必须引用它, 像这样:

foo: "{{ variable }}"

# 二、 fabric

[Fabric](http://docs.fabfile.org/en/1.4.0/index.html)是一个Python库，可以通过SSH在多个host上批量执行任务 可以编写任务脚本，然后通过Fabric在本地就可以使用SSH在大量远程服务器上自动运行。这些功能非常适合应用的自动化部署，使用 Fabric 来写和执行 Python 函数或 task ，以实现与远程服务器的自动化交互。

1、优点：

a 代码可读性强，完全是python语言。  
b 封装了本地、远程操，内置了一些方法。  
c 可以根据需求灵活的定义参数。  
d 角色定义，方便批量管理。

2、常用的配置参数

env.host           -- 主机ip，当然也可以-H参数指定  
env.password       -- 密码，打好通道的请无视

env.roledefs       -- 角色分组，比如：{'web': ['x', 'y'], 'db': ['z']}

fab -H 主机名 -f 文件名 方法

local('pwd')                     -- 执行本地命令  
lcd('/tmp')                      -- 切换本地目录  
cd('/tmp')                       -- 切换远程目录  
run('uname -a')                  -- 执行远程命令，一般会提示叫你输入密码。

sudo('/etc/init.d/nginx start')  -- 执行远程sudo，注意pty选项

3、fabric的使用

fab -l             -- 显示可用的task（命令）  
fab -H             -- 指定host，支持多host逗号分开  
fab -R             -- 指定role，支持多个  
fab -P             -- 并发数，默认是串行  
fab -w             -- warn\_only，默认是碰到异常直接abort退出

fab -f             -- 指定入口文件，fab默认入口文件是：fabfile/fabfile.py

4、通过 **命令行** , 使用 Fabric 来写和执行 Python 函数

* **cat fabfile.py**

**#!/usr/bin/python**

**def** hello**():**

**print(**"Hello world!"**)**

* $ fab hello
* Hello world!
* Done.

## 5、任务参数

和你平时的 Python 编程一样，给任务函数传递参数很有必要``。Fabric 支持 Shell 兼容的参数用法： <任务名>:<参数>, <关键字参数名>=<参数值>,...

* **def** hello**(**name="world"**):**

**print(**"Hello %s!" % name**)**

* fab hello:name=Jeff

Hello Jeff!

Done.

## 6、环境字典env及运行环境(设置)

Fabric 中有一个简单但是必不可少的部分叫做“环境”：它是 Python 字典的子类，既用作设置，也用于任务间数据空间共享

Fabric 的大部分行为可以通过修改 env 变量，例如 env.hosts ,来控制（已经在 入门导览 中见过）。其他经常需要修改的环境变量包括：

* user ：Fabric 在建立 SSH 连接时默认使用本地用户名，必要情况下可以通过修改 env.user 来设置。 [*Execution model*](http://fabric-chs.readthedocs.io/zh_CN/chs/usage/execution.html) 文档中还介绍了如何为每个主机单独设置用户名。
* password ：用来显式设置默认连接或者在需要的时候提供 sudo 密码。如果没有设置密码或密码错误，Fabric 将会提示你输入。
* warn\_only：布尔值，用来设置 Fabric 是否在检测到远程错误时退出。访问 [*Execution model*](http://fabric-chs.readthedocs.io/zh_CN/chs/usage/execution.html) 以了解更多关于此行为的信息。

注意：.host\_string 和 env['host\_string'] 的作用是完全一样

## 7、直接执行远程命令

fab [options] -- [shell command]

$ fab -H system1,system2,system3 -- uname -a

&&

$ fab -H system1,system2,system3 anonymous

## 8、fabfile文件及结构使用

Fabric 能够加载 Python 模块（如： fabfile.py ）和包（如 fabfile/ ），默认情况下，它会根据 Python 包的导入机制加载 fabfile －可以是 fabfile/ 也可以是 fabfile.py

## 9、连接服务器

[**run**](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/operations.html#fabric.operations.run) 和 **[sudo](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/operations.html" \l "fabric.operations.sudo" \o "fabric.operations.sudo)** 这样的操作在连接时都会查看同一处设置： [env.host\_string](http://fabric-chs.readthedocs.io/zh_CN/chs/usage/env.html" \l "host-string) 。其它设置主机列表的机制都用于 fab 命令，和作为 Python 库使用没有关系。如果你想要结合任务 X 和主机列表 Y ，可以使用 [**execute**](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/tasks.html#fabric.tasks.execute) ，就像这样： execute(X, hosts=Y) ，详细介绍请访问 [**execute**](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/tasks.html#fabric.tasks.execute) 文档——手动操作主机列表功能真的很有必要。[**disconnect\_all**](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/network.html#fabric.network.disconnect_all) 断开连接

## 10、标准输出

* **status**：状态信息。包括提示 Fabric 已结束运行、用户是否使用键盘中止操作、或者服务器是否断开了连接。通常来说这些信息都不会很冗长，但是至关重要。
* **aborts**：终止信息。和状态信息一样，只有当 Fabric 做为库使用的时候才可能应该关闭，而且还并不一定。注意，即使该输出集被关闭了，并不能阻止程序退出——你只会得不到任何 Fabric 退出的原因。
* **warnings**：警报信息。通常在预计指定操作失败时会将其关闭，比如说你可能使用 grep 来测试文件中是否有特定文字。如果设置 env.warn\_only 为 True 会导致远程程序执行失败时完全没有警报信息。和 aborts 一样，这项设置本身并不控制警报行为，仅用于是否输出警报信息。
* **running**：输出正在执行的命令或者正在传输的文件名称，比如： [myserver] run:ls /var/www。同时它还输出正在运行的任务名，比如：[myserver] Executing task'foo'。
* **stdout**：本地或远程的 stdout。来自命令行的非错误输出。
* **stderr**：本地或远程的 stderr。比如命令中错误相关的输出。
* **user**：用户生成的输出。比如：fabfile 中使用 **[fastprint](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/utils.html" \l "fabric.utils.fastprint" \o "fabric.utils.fastprint)** 或者 [**puts**](http://fabric-chs.readthedocs.io/zh_CN/chs/api/core/utils.html#fabric.utils.puts) 函数产生的输出。