2.4

文档作者：马龙帅

博客地址：[***https://www.junmajin***](http://www.junmajinlong.com/)***long.com***

博客地址：[***https://www.cnblogs.com/f-ck-need-u/***](http://www.cnblogs.com/f-ck-need-u/) 更完整详细的Ansible专栏：一步到位玩透***Ansible***

声明：

欢迎善意传播，杜绝恶意利用

本人享有本文档内容的一切法律权利

#### ⽬录

1. [基本配置和使⽤ 5](#_TOC_250091)
   1. [安装Ansible 5](#_TOC_250090)
   2. [配置ansible 7](#_TOC_250089)
      1. [环境配置 7](#_TOC_250088)
      2. [SSH互信配置 8](#_TOC_250087)
      3. [简单测试 9](#_TOC_250086)
   3. [inventory 10](#_TOC_250085)
2. [选项和常⽤模块 12](#_TOC_250084)
   1. [ansible命令解释 12](#_TOC_250083)
   2. [常⽤模块 15](#_TOC_250082)

[shell和command 15](#_TOC_250081)

[复制模块copy 16](#_TOC_250080)

[template模块 17](#_TOC_250079)

[⽂件模块file 18](#_TOC_250078)

[fetch拉取⽂件模块 18](#_TOC_250077)

[rsync模块synchroniz e 19](#_TOC_250076)

[包管理模块yum 19](#_TOC_250075)

[配置yum源模块yum\_repository 20](#_TOC_250074)

[服务管理模块service 20](#_TOC_250073)

[systemd模块 21](#_TOC_250072)

[⽤户管理模块user 21](#_TOC_250071)

[authorz ied\_key模块 22](#_TOC_250070)

[debug模块 22](#_TOC_250069)

[定时任务模块cron 23](#_TOC_250068)

[归档模块archive 24](#_TOC_250067)

[解包模块unarchive 24](#_TOC_250066)

[下载模块get\_url 25](#_TOC_250065)

[wait\_for模块 26](#_TOC_250064)

[script模块 26](#_TOC_250063)

1. [YAML语法和playbook写法 28](#_TOC_250062)
   1. [初步说明 28](#_TOC_250061)

[3.2 列表 28](#_TOC_250060)

[3.3 字典 29](#_TOC_250059)

* 1. [分⾏写 31](#_TOC_250058)
  2. [向模块传递参数 32](#_TOC_250057)
  3. [playbook和play的关系 33](#_TOC_250056)
  4. [playbook中什么时候使⽤引号 34](#_TOC_250055)

1. [.playbook应⽤和⽰例 36](#_TOC_250054)
   1. [yaml简单⽰例 36](#_TOC_250053)
   2. [ansible- playbook命令说明及playbook书写简单⽰例 36](#_TOC_250052)
   3. [playbook的内容 37](#_TOC_250051)
      1. [hosts和remoter\_user 37](#_TOC_250050)
      2. [task list 38](#_TOC_250049)
      3. [notify和handler 40](#_TOC_250048)
      4. [标签tag 41](#_TOC_250047)
   4. [include和roles 41](#_TOC_250046)
      1. [include 41](#_TOC_250045)
      2. [roles 43](#_TOC_250044)
   5. [roles⽰例：批量⾃动化安装 45](#_TOC_250043)
2. [各种变量定义⽅式和变量引⽤ 52](#_TOC_250042)
   1. [ansible facts 52](#_TOC_250041)
   2. [变量引⽤json数据的⽅式 53](#_TOC_250040)
      1. [引⽤json字典数据的⽅式 54](#_TOC_250039)
      2. [引⽤json数组数据的⽅式 54](#_TOC_250038)
      3. [引⽤facts数据 55](#_TOC_250037)
   3. [设置本地facts 56](#_TOC_250036)
   4. [输出和引⽤变量 56](#_TOC_250035)
   5. [注册和定义变量的各种⽅式 57](#_TOC_250034)
      1. [register注册变量 57](#_TOC_250033)
      2. [set\_fact定义变量 57](#_TOC_250032)
      3. [vars定义变量 58](#_TOC_250031)
      4. [vars\_files定义变量 58](#_TOC_250030)
      5. [roles中的变量 58](#_TOC_250029)
      6. [命令⾏传递变量 59](#_TOC_250028)
      7. [借助with\_items叠加变量 59](#_TOC_250027)
      8. [inventory中主机变量和主机组变量 60](#_TOC_250026)
      9. [内置变量 61](#_TOC_250025)

[6 .条件判断和循环 6 5](#_TOC_250024)

[6.1 循环 65](#_TOC_250023)

* + 1. [with\_items迭代列表 65](#_TOC_250022)
    2. [with\_dict迭代字典项 66](#_TOC_250021)
    3. [with\_fileglob迭代⽂件 66](#_TOC_250020)
    4. [with\_lines迭代⾏ 67](#_TOC_250019)
    5. [with\_nested嵌套迭代 67](#_TOC_250018)

[6.2 条件判断 67](#_TOC_250017)

1. [执⾏过程分析、异步模式和速度优化 6 9](#_TOC_250016)
   1. [ansible执⾏过程分析 69](#_TOC_250015)
   2. [ansible并发和异步 71](#_TOC_250014)
   3. [ansible的- t选项妙⽤ 72](#_TOC_250013)
   4. [优化ansible速度 73](#_TOC_250012)
      1. [设置ansible开启ssh长连接 74](#_TOC_250011)
      2. [开启pipelining 74](#_TOC_250010)
      3. [修改ansible执⾏策略 76](#_TOC_250009)
      4. [设置facts缓存 77](#_TOC_250008)
2. [playbook杂项 78](#_TOC_250007)
   1. [指定运⾏play的主机delegate\_to和local\_action 78](#_TOC_250006)
   2. [只运⾏⼀次run\_once 78](#_TOC_250005)
   3. [分批执⾏play 79](#_TOC_250004)
   4. [分批执⾏play时的最⼤失败百分⽐ 80](#_TOC_250003)
   5. [错误处理——忽略错误 80](#_TOC_250002)
   6. [错误处理——⾃定义错误判断条件 80](#_TOC_250001)
3. [playbook⽰例： 编译安装httpd 82](#_TOC_250000)

# 基本配置和使⽤

Ansible是⼀种批量、⾃动部署⼯具，不仅可以批量，还可以⾃动。它主要基于ssh进⾏通信，不要求客户端(被控 制端)安装ansible。

## 安装Ansible

安装⽅法有多种，可以下载源码后编译安装，可以从git上获取资源安装，也可以rpm包安装。rpm安装需要配置

epel源。

**cat <<eof>>/etc/yum.repos.d/my.repo [epel]**

**name=epel baseurl=**[**http://mirrors.aliyun.com/epel/7Server/x86\_64/**](http://mirrors.aliyun.com/epel/7Server/x86_64/) **enable=1**

**gpgcheck=0 eof**

后⾯⼏篇⽂章⽤到的环境。

|  |  |  |  |
| --- | --- | --- | --- |
| **主机描述IP地址** | **主机名** |  | **操作系统** |
| **ansible6\_server1** | **192.168.100.150** | **server1.longshuai.com** | **CentOS 6.6** |
| **ansible6\_node1** | **192.168.100.59** | **node1.longshuai.com** | **CentOS 6.6** |
| **ansible6\_node2** | **192.168.100.60** | **node2.longshuai.com** | **CentOS 6.6** |
| **ansible6\_node3** | **192.168.100.61** | **node3.longshuai.com** | **CentOS 6.6** |
| **ansible7\_server2** | **192.168.100.62** | **server2.longshuai.com** | **CentOS 7.2** |
| **ansible7\_node1** | **192.168.100.63** | **anode1.longshuai.com** | **CentOS 7.2** |
| **ansible7\_node2** | **192.168.100.64** | **anode2.longshuai.com** | **CentOS 7.2** |
| **ansible7\_node3** | **192.168.100.65** | **anode3.longshuai.com** | **CentOS 7.2** |

经测试，CentOS 6上安装ansible 2.3版本有可能会⾮常慢，需要将ansible执⾏的结果使⽤重定向或者- t选项保存到⽂件中，下次执⾏才会快。

**shell> yum -y install ansible**

**/etc/ansible/ansible.cfg**

**/etc/ansible/hosts**

**/etc/ansible/roles**

**/usr/bin/ansible**

**/usr/bin/ansible-2**

**/usr/bin/ansible-2.6**

**/usr/bin/ansible-connection**

**/usr/bin/ansible-console**

**/usr/bin/ansible-console-2**

**/usr/bin/ansible-console-2.6**

**/usr/bin/ansible-doc**

**/usr/bin/ansible-doc-2**

**/usr/bin/ansible-doc-2.6**

**/usr/bin/ansible-galaxy**

**/usr/bin/ansible-galaxy-2**

**/usr/bin/ansible-galaxy-2.6**

**/usr/bin/ansible-playbook**

**/usr/bin/ansible-playbook-2**

**/usr/bin/ansible-playbook-2.6**

**/usr/bin/ansible-pull**

**/usr/bin/ansible-pull-2**

**/usr/bin/ansible-pull-2.6**

**/usr/bin/ansible-vault**

**/usr/bin/ansible-vault-2**

**/usr/bin/ansible-vault-2.6**

使⽤ansible- doc可以列出相关的帮助。

**ansible-doc -h**

**Usage: ansible-doc [options] [module...]**

**Options:**

**-a, --all**

**-h, --help**

**-l, --list**

**Show documentation for all modules**

**show this help message and exit List available modules**

**-M MODULE\_PATH, --module-path=MODULE\_PATH**

**specify path(s) to module library (default=None)**

**-s, --snippet Show playbook snippet for specified module(s)**

**-v, --verbose verbose mode (-vvv for more, -vvvv to enable connection debugging)**

**--version show program's version number and exit**

其中"- l"选项⽤于列出ansible的模块，通常结合grep来筛选。例如找出和yum相关的可⽤模块。

**ansible-doc -l | grep yum**

**yum yum\_repository**

**Manages packages with the `yum' package manager**

**Add or remove YUM repositories**

再使⽤"- s"选项可以获取指定模块的使⽤帮助。例如，获取yum模块的使⽤语法。

**ansible-doc -s yum**

**- name: Manages packages with the `yum' package manager action: yum**

**conf\_file # The remote yum configuration file to use for the transaction. disable\_gpg\_check # Whether to disable the GPG checking of signatures of packages being**

**installed. Has an effect only if state is `present' or `latest'. disablerepo # `Repoid' of repositories to disable for the install/update operation.**

**These repos will not persist beyond the transaction. When specifying multiple repos, separate them with a ",".**

**enablerepo # `Repoid' of repositories to enable for the install/update operation. These repos will not persist beyond the transaction. When specifying multiple repos, separate them with a ",".**

**exclude # Package name(s) to exclude when state=present, or latest installroot # Specifies an alternative installroot, relative to which all packages**

**will be installed.**

**list # Package name to run the equivalent of yum list <package> against. name= # Package name, or package specifier with version, like `name-1.0'.**

**When using state=latest, this can be '\*' which means run: yum -y update. You can also pass a url or a local path to a rpm file(using state=present). To operate on several packages this can accept a comma separated list of packages or (as of 2.0) a list of packages.**

**skip\_broken # Resolve depsolve problems by removing packages that are causing problems from the trans- action.**

**state # Whether to install (`present' or `installed', `latest'), or remove (`absent' or `removed') a package.**

**update\_cache # Force yum to check if cache is out of date and redownload if needed. Has an effect only if state is `present' or `latest'.**

**validate\_certs # This only applies if using a https url as the source of the rpm.**

**e.g. for localinstall. If set to `no', the SSL certificates will not be validated. This should only set to `no' used on personally controlled sites using self-signed certificates as it avoids verifying the source site. Prior to 2.1 the code worked as if this was set to `yes'.**

例如使⽤yum安装unix2dos包。

**ansible 192.168.100.60 -m yum -a "name=unix2dos state=present"**

其中192.168.100.60是被ansible远程控制的机器，即要在此机器上安装unix2dos，下⼀⼩节将说明如何指定待控 制主机。"- m"指定模块名称，"- a"⽤于为模块指定各模块参数，例如name和state。

ansible命令选项和各模块的使⽤⽅法见：Ansible系列(⼆)：选项和常⽤模块。

## 配置ansible

### 环境配置

Ansible配置以ini格式存储配置数据，在Ansible中⼏乎所有配置都可以通过Ansible的Playbook或环境变量来重新赋 值。在运⾏Ansible命令时，命令将会按照以下顺序查找配置⽂件。

**ANSIBLE\_CONFIG ：⾸先，Ansible命令会检查环境变量，及这个环境变量指向的配置⽂件。**

**./ansible.cfg ：其次，将会检查当前⽬录下的ansible.cf g配置⽂件。**

**~/.ansible.cfg ：再次，将会检查当前⽤户home⽬录下的.ansible.cf g配置⽂件。**

**/etc/ansible/ansible.cfg ：最后，将会检查在⽤软件包管理⼯具安装Ansible时⾃动产⽣的配置⽂件。**

1、使⽤化境变量⽅式来配置

⼤多数的Ansible参数可以通过设置带有ANSIBLE\_ 开头的环境变量进⾏配置，参数名称必须都是⼤写字母，如下配置：

**export ANSIBLE\_SUDO\_USER=root**

**设置了环境变量之后， ANSIBLE\_SUDO\_USER 就可以在后续操作中直接引⽤。**

2、设置ansible.cfg配置参数

Ansible有很多配置参数，以下是⼏个默认的配置参数：

**inventory = /root/ansible/hosts library = /usr/share/my\_modules/ forks = 5**

**sudo\_user = root remote\_port = 22 host\_key\_checking = False timeout = 20**

**log\_path = /var/log/ansible.log**

inventory ：该参数表⽰inventory⽂件的位置，资源清单(inventory)就是Ansible需要连接管理的⼀些主机列表。

library ：Ansible的所有操作都使⽤模块来执⾏实现，这个library参数就是指向存放Ansible模块的⽬录。

forks ：设置默认情况下Ansible最多能有多少个进程同时⼯作，默认5个进程并⾏处理。具体需要设置多少个，可以根据控制端性能和被管理节点的数量来确定。

**sudo\_user ：设置默认执⾏命令的⽤户，也可以在playbook中重新设置这个参数。**

**remote\_port ：指定连接被管理节点的管理端⼝，默认是22，除⾮设置了特殊的SSH端⼝，否则不需要修改此参数。host\_key\_checking ：设置是否检查SSH主机的密钥。可以设置为True或False。即ssh的主机再次验证。**

**timeout ：设置SSH连接的超时间隔，单位是秒。**

log\_path ：Ansible默认不记录⽇志，如果想把Ansible系统的输出记录到⽇志⽂件中，需要设置log\_path。需要注意，模块将会调⽤被管节点的(r)syslog来记录，执⾏Ansible的⽤户需要有写⼊⽇志的权限。

### SSH互信配置

将ansible server的ssh公钥分发到各被管节点上。在ansible6\_server1和ansible\_server2上：

**ssh-keygen -t rsa -f ~/.ssh/id\_rsa -N '' ssh-copy-id** [**root@192.168.100.59**](mailto:root@192.168.100.59)

**ssh-copy-id** [**root@192.168.100.60**](mailto:root@192.168.100.60) **ssh-copy-id** [**root@192.168.100.61**](mailto:root@192.168.100.61) **ssh-copy-id** [**root@192.168.100.62**](mailto:root@192.168.100.62) **ssh-copy-id** [**root@192.168.100.63**](mailto:root@192.168.100.63) **ssh-copy-id** [**root@192.168.100.64**](mailto:root@192.168.100.64) **ssh-copy-id** [**root@192.168.100.65**](mailto:root@192.168.100.65) **ssh-copy-id** [**root@192.168.100.150**](mailto:root@192.168.100.150)

也可以使⽤ansible⾃⾝来批量添加密钥到被控节点上。使⽤ansible的authorized\_key模块即可。见后⽂常见模块 介绍部分。

以下是借助expect⼯具实现⾮交互式的ssh- copy- id，免得总是询问远程⽤户的登录密码。

**# 安装expect**

**[root@server2 ~]# yum -y install expect**

**# expect脚本**

**[root@server2 ~]# cat auto\_sshcopyid.exp**

**#!/usr/bin/expect**

**set timeout 10**

**set user\_hostname [lindex $argv 0] set password [lindex $argv 1]**

**spawn ssh-copy-id $user\_hostname expect {**

**"(yes/no)?"**

**{**

**send "yes\n"**

**expect "\*password: " { send "$password\n" }**

**}**

**"\*password: " { send "$password\n" }**

**}**

**expect eof**

**# 批量调⽤expect的shell脚本[root@server2 ~]# cat sshkey.sh #!/bin/bash**

**ip=`echo -n "$(seq -s "," 59 65),150" | xargs -d "," -i echo 192.168.100.{}` password="123456"**

**#user\_host=`awk '{print $3}' /root/.ssh/id\_rsa.pub`**

**for i in $ip;do**

**/root/auto\_sshcopyid.exp root@$i $password &>>/tmp/a.log ssh root@$i "echo $i ok"**

**done**

**# 执⾏shell脚本配置互信**

**[root@server2 ~]# chmod +x /root/{sshkey.sh,auto\_sshcopyid.exp}**

**[root@server2 ~]# ./sshkey.sh**

### 简单测试

向默认的inventory⽂件/etc/ansible/hosts中添加上⼏个被管节点清单。在ansible6\_server1上：

**cat >>/etc/ansible/hosts<<eof 192.168.100.59**

**192.168.100.60**

**192.168.100.61**

**192.168.100.62**

**192.168.100.63**

**192.168.100.64**

**192.168.100.65**

**[centos6] 192.168.100.59**

**192.168.100.60**

**192.168.100.61**

**[centos7] 192.168.100.63**

**192.168.100.64**

**192.168.100.65**

**[centos:children] centos6**

**centos7 eof**

在ansible7\_server2上：

**cat >>/etc/ansible/hosts<<eof 192.168.100.150**

**192.168.100.59**

**192.168.100.60**

**192.168.100.61**

**192.168.100.63**

**192.168.100.64**

**192.168.100.65**

**[centos6] 192.168.100.59**

**192.168.100.60**

**192.168.100.61**

**[centos7] 192.168.100.63**

**192.168.100.64**

**192.168.100.65**

**[centos:children] centos6**

**centos7 eof**

使⽤ping模块测试被管节点。能成功，说明ansible能控制该节点。

**ansible 192.168.100.59 -m ping ansible centos6 -m ping 192.168.100.59 | SUCCESS => {**

**"changed": false, "ping": "pong"**

**}**

* + - 1. **| SUCCESS => {**

**"changed": false, "ping": "pong"**

**}**

* + - 1. **| SUCCESS => {**

**"changed": false, "ping": "pong"**

**}**

如果要指定⾮root⽤户运⾏ansible命令，则加上"- - sudo"或"- s"来提升为sudo\_user配置项所指定⽤户的权限。或者使⽤become提升权限。

**ansible webservers -m ping -u ansible --sudo**

**ansible webservers -m ping -b --become-user=root --become-method=sudo**

## inventory

inventory⽤于定义ansible要管理的主机列表，可以定义单个主机和主机组。上⾯的/etc/ansible/hosts就是默认的inventory。下⾯展⽰了inventory常⽤的定义规则。

**cat -n /etc/ansible/hosts 1 192.168.100.59:22**

1. **192.168.100.60 ansible\_ssh\_pass=123456 ansible\_ssh\_port=22**
2. **[nginx]**

**4 192.168.100.5[7:9]**

1. **[nginx:vars]**
2. **ansible\_ssh\_pass='123456'**
3. **[webservers:children]**
4. **nginx**

第⼀⾏和第⼆⾏单独定义主机，第⼀⾏带上了连接被管节点的端⼝，第⼆⾏带上了单独传递给ssh的参数，分别是 ssh连接时的登录远程⽤户的密码参数和ssh的连接端⼝。

第三⾏和第四⾏定义的是nginx主机组，该组中包含了192.168.100.57到59这3台主机。还⽀持字母的扩展，

如"web[a- d]"。

第五⾏和第六⾏定义了要传递给nginx主机组的变量。若定义为"[all:vars]"或"[\*:vars]"则表⽰传递给所有主机的变 量。

第七和第⼋⾏定义了⼀个新的主机组webservers，改组的组成员有nginx组。

可以指定多个inventory配置⽂件，只需在ansible的配置⽂件如/etc/ansible/ansible.cf g中将inventory指令设置为对应的⽂件或⽬录即可，如果是⽬录，那么此⽬录下的所有⽂件都是inventory⽂件。

inventory⽂件中可以使⽤⼀些内置变量，绝⼤多数ansible的连接和权限变量都可以在此使⽤，见ansible命令解 释。常见的有：

**ansible\_ssh\_host ： ansible使⽤ssh要连接的主机。ansible\_ssh\_port ： ssh的端⼝。默认为22。ansible\_ssh\_user ： ssh登录的⽤户名。默认为root。ansible\_ssh\_pass ： ssh登录远程⽤户时的认证密码。**

**ansible\_ssh\_private\_key\_file ： ssh登录远程⽤户时的认证私钥。(?)**

ansible\_connection ： 使⽤何种模式连接到远程主机。默认值为smart(智能)，表⽰当本地ssh⽀持持久连接(controlpersist)时采⽤ssh连接，否则采⽤python的paramiko ssh连接。

**ansible\_shell\_type ： 指定远程主机执⾏命令时的shell解析器，默认为sh(不是bash，它们是有区别的，也不是全路径)。**

**ansible\_python\_interpreter ： 远程主机上的python解释器路径。默认为/usr/bin/python。**

ansible\_\*\_interpreter ：使⽤什么解释器。例如，sh、bash、awk、sed、expect、ruby等等。其中有⼏个参数可以在配置⽂件ansible.cf g中指定，但指定的指令不太⼀样，以下是对应的配置项：

remote\_port： 对应于ansible\_ssh\_port。remote\_user： 对应于ansible\_ssh\_user。

private\_key\_f ile： 对应于ansible\_ssh\_private\_key\_f ile。

excutable： 对应于ansible\_shell\_type。但有⼀点不⼀样，excutable必须指定全路径，⽽后者只需指定basename。

如果定义了"ansible\_ssh\_host"，那么其前⾯的主机名就称为别名。例如，以下inventory⽂件中nginx就是⼀个别 名，真正连接的对象是192.168.100.65。

**nginx ansible\_ssh\_host=192.168.100.65 ansible\_ssh\_port=22**

当inventory中有任何⼀台有效主机时，ansible就默认隐式地可以使⽤"localhost"作为本机，但inventory中没有任何主机时是不允许使⽤它的，且"all"或"\*"所代表的所有主机也不会包含localhost。例如：

ansible **localhost -i /path/to/inventory\_file -m MODULE -a "ARGS"** ansible **all -i /path/to/inventory\_file -m MODULE -a "ARGS"** ansible **\* -i /path/to/inventory\_file -m MODULE -a "ARGS"**

是ansible中可以使⽤的⼀个变量，该变量代表的是每个主机在inventory中的主机名称。例 如"192.168.100.59"。这是⽬前遇到的第⼀个变量。

**inventory\_hostname**

# 选项和常⽤模块



*本章内容做速查手册*

## ansible命令解释

通过ansible命令执⾏的任务称为ad- hoc命令(任务)，其实它是相对playbook⽽⾔的。通常，命令⾏⽤来实现ansible的批量管理功能，playbook⽤来实现批量⾃动化功能。

【以下为普通选项：】

**-a MODULE\_ARGS**

**--args=MODULE\_ARGS**

传递参数给模块

**--ask-vault-pass**

询问vault的密码

**-B SECONDS**

**--background=SECONDS**

异步后台⽅式执⾏任务，并在指定的秒数后超时，超时会杀掉任务的进程。默认是同步，即保持长连接，它会等待所有执⾏完毕（即阻塞模式）。但有时候是没必要这样的，⽐如某些命令的执⾏时间⽐ssh的超时时间还长。如果不指定超时秒数，将以同步⽅式运⾏任务

**-P POLL\_INTERVAL**

**--poll=POLL\_INTERVAL**

异步模式下轮训任务的时间间隔，默认10秒



**-C**

**--check**

不对远程主机做出⼀些改变，⽽是预测某些可能发⽣的改变



**-D**

**--diff**

当⽂件或模板发⽣了改变，显⽰出不同之处，和- C选项配合使⽤更佳

**-e EXTRA\_VARS**

**--extra-vars=EXTRA\_VARS**

配置额外的配置变量(key=value或者YAML/JSON格式)

**-f FORKS**

**--forks=FORKS**

指定并⾏处理的进程数量，默认为5个



**-h**

**--help**

显⽰帮助信息

**-i INVENTORY**

**--inventory-file=INVENTORY**

指定inventory⽂件，多个⽂件使⽤逗号分隔。默认为/etc/ansible/hosts

**-l SUBSET**

**--limit=SUBSET**

使⽤额外的匹配模式来筛选⽬标主机列表。此处的匹配模式是在已有匹配模式下进⾏的，所以是更严格的筛选。例如指定了主机组的情况下，使⽤- l选项从中只选⼀台主机进⾏控制

**--list-hosts**

不会执⾏任何操作，⽽是列出匹配到的主机列表

**-m MODULE\_NAME**

**--module-name=MODULE\_NAME**

指定要执⾏的模块名，默认的模块为"command"

**-M MODULE\_PATH**

**--module-path=MODULE\_PATH**

指定模块⽬录，默认未设置

**--new-vault-password-file=NEW\_VAULT\_PASSWORD\_FILE**

new vault password f ile f or rekey



**-o**

**--one-line**

简化输出(⼀⾏输出模式)

**--output=OUTPUT\_FILE**

output f ile name f or encrypt or decrypt; use - f or stdout

**--syntax-check**

检查playbook的语法，不会执⾏

**-t TREE**

**--tree=TREE**

记录输出到此⽬录中（测试时以每个host名如IP地址为⽂件名记录，结果记录到对应的⽂件中）。

此选项在ansible巨慢的时候( 如瞬间应该返回的命令还需要10多秒才完成)有奇⽤，或者将ansible的结果重定向到某个⽂件中也能解决，为什么如此，我也不明⽩( 表⾯看来和输出⽅式有关系)，多次亲测有效。

**--vault-password-file=VAULT\_PASSWORD\_FILE**

指定vault密码⽂件



**-v**

**--verbose**

输出详细信息，- vvv和- vvvv会输出更多新

**--version**

显⽰ansible的版本

【以下是连接选项，⽤于控制谁以及如何连接主机：】



**-k**

**--ask-pass**

询问连接时的密码

**--private-key=KEY\_FILE**

**--key-file=KEY\_FILE**

使⽤⽂件来认证SSH连接过程。

**-u REMOTE\_USER**

**--user=REMOTE\_USER**

使⽤指定的⽤户名进⾏连接

**-c CONNECTION**

**--connection=CONNECTION**

连接类型，默认为ssh。paramiko (SSH), ssh, winrm and local. local is mostly usef ul f or crontab or kickstarts.

**-T TIMEOUT, --timeout=TIMEOUT**

连接的超时时间，单位为秒，默认为10

**--ssh-common-args=SSH\_COMMON\_ARGS**

指定传递给sf tp/scp/ssh等⼯具的通⽤额外参数

**--sftp-extra-args=SFTP\_EXTRA\_ARGS**

指定只传递给sf tp的额外参数，如- f

**--scp-extra-args=SCP\_EXTRA\_ARGS**

指定只传递给scp的额外参数，如- l

**--ssh-extra-args=SSH\_EXTRA\_ARGS**

指定只传递给ssh的额外参数，如- R

【以下是权限控制选项：控制在⽬标主机上以什么⾝份和权限运⾏任务：】



**-s**

**--sudo**

为运⾏ansible命令的⽤户提升权限为sudo\_user的权限，此命令已废弃，使⽤become代替

**-U SUDO\_USER**

**--sudo-user=SUDO\_USER**

期望的sudo\_user，默认为root，已废弃，使⽤become替代



**-S**

**--su**

使⽤su的⽅式执⾏操作，已废弃，使⽤become替代

**-R SU\_USER**

**--su-user=SU\_USER**

使⽤此user的su执⾏操作，默认为root，已废弃，使⽤become替代



**-b**

**--become**

使⽤become的⽅式升级权限

**--become-method=BECOME\_METHOD**

指定提升权限的⽅式，可选以下⼏种：sudo/su/pbrun/pf exec/doas/dzdo/ksu/runas值

**--become-user=BECOME\_USER**

要提升为哪个user的权限，默认为root

**--ask-sudo-pass**

询问sudo密码，已废弃，使⽤become替代

**--ask-su-pass**

询问su的密码，已废弃，使⽤become替代



**-K**

**--ask-become-pass**

询问become提升权限时的密码

## 常⽤模块

可以从ansible- doc - l | grep来找出想要的模块。再使⽤ansible- doc - s module\_name来查看此模块的⽤法。官⽅模块列表和说明：https://docs.ansible.com/ansible/latest/modules\_by\_category.html

关于模块的使⽤⽅法，需要注意的是"state"。很多模块都会有该选项，且其值⼏乎都包含有"present"和"absent"， 表⽰肯定和否定的意思。

ansible绝⼤多数模块都天然具有幂等 特性，只有极少数模块如shell和command模块不具备幂等性。所谓的幂等性是指多次执⾏同⼀个操作不会影响最终结果。例如，ansible的yum模块安装rpm包时，如果待安装的包已经安装过 了，则再次或多次执⾏安装操作都不会真正的执⾏下去。再例如，copy模块拷贝⽂件时，如果⽬标主机上已经有了 完全相同的⽂件，则多次执⾏copy模块不会真正的拷贝。ansible具有幂等性的模块在执⾏时，都会⾃动判断是否要执⾏。

### shell和command

默认ansible使⽤的模块是command，即可以执⾏⼀些shell命令。shell和command的⽤法基本⼀样，实际上shell

模块执⾏命令的⽅式是在远程使⽤/bin/sh来执⾏的，如/bin/sh ping。

command不能解析变量(如$HOME)和某些操作符("<", ">", "|", ";"以及"&")，所以明确要使⽤这些不可解析的操作符时，使⽤shell模块来代替command。

**ansible-doc -s shell**

**- name: Execute commands in nodes. action: shell**

**chdir # 在执⾏命令前，先cd到指定的⽬录下**

**creates # ⽤于判断命令是否要执⾏。如果指定的⽂件(可以使⽤通配符)存在，则不执⾏。removes # ⽤于判断命令是否要执⾏。如果指定的⽂件(可以使⽤通配符)不存在，则不执⾏。executable # 不再使⽤默认的/bin/sh解析并执⾏命令，⽽是使⽤此处指定的命令解析。**

**# 例如使⽤expect解析expect脚本。必须为绝对路径。**

在ansible中使⽤shell或command模块⼀定要注意，它们默认不满⾜幂等性，很多操作会重复执⾏，但有些操作是 不允许重复执⾏的。例如mysql的初始化命令mysql\_install\_db，它只能在第⼀次配置的过程中初始化⼀次，其他任 何时候如⾮需要则不允许执⾏。这时候要实现幂等性，可以通过模块的creates和removes选项进⾏判断，但⽆论如何，在执⾏这两个模块的时候都需要考虑要执⾏的命令是否应该实现幂等性。

例如：

**tasks:**

**- shell: touch helloworld.txt creates=/tmp/hello.txt**

但建议，在参数可能产⽣歧义的情况下，使⽤args来传递ansible的参数。如：

**- shell: touch helloworld.txt args:**

**creates: /tmp/hello.txt**

**# You can use shell to run other executables to perform actions inline**

**- name: Run expect to wait for a successful PXE boot via out-of-band CIMC shell: |**

**set timeout 300**

**spawn ssh admin@{{ cimc\_host }}**

**expect "password:"**

**send "{{ cimc\_password }}\n"**

**expect "\n{{ cimc\_name }}" send "connect host\n"**

**expect "pxeboot.n12" send "\n"**

**exit 0 args:**

**executable: /usr/bin/expect delegate\_to: localhost**

### 复制模块copy

**ansible-doc -l | grep copy**

**copy Copies files to remote locations.**

**ec2\_ami\_copy copies AMI between AWS regions, return new image id netapp\_e\_volume\_copy Create volume copy pairs**

**nxos\_file\_copy Copy a file to a remote NXOS device over SCP.**

**unarchive Unpacks an archive after(optionally) copying it from the local machine. vsphere\_copy Copy a file to a vCenter datastore**

**win\_copy Copies files to remote locations on windows hosts. win\_robocopy Synchronizes the contents of two directories using Robocopy.**

使⽤⽅法：

**ansible-doc -s copy**

**- name: Copies files to remote locations. action: copy**

**backup=[yes|no] # 拷贝的同时也创建⼀个包含时间戳信息的备份⽂件，默认为no**

**dest= # ⽬标路径，只能是绝对路径，如果拷贝的⽂件是⽬录，则⽬标路径必须也是⽬录**

**content # 直接以content给定的字符串或变量值作为⽂件内容保存到远程主机上，它会替代src选项**

**directory\_mode # 当对⽬录做递归拷贝时，设置了directory\_mode将会使得只拷贝新建⽂件，**

**# 旧⽂件不会被拷贝。默认未设置**

**follow=[yes|no] # 是否追踪到链接的源⽂件。**

**force=[yes|no] # 设置为yes(默认)时，将覆盖远程同名⽂件。设置为no时，忽略同名⽂件的拷贝。**

**group # 设置远程⽂件的所属组**

**owner # 设置远程⽂件的所有者**

**mode= # 设置远程⽂件的权限。使⽤数值表⽰时不能省略第⼀位，如0644。**

**# 也可以使⽤'u+rwx'或'u=rw,g=r,o=r'等⽅式设置。**

**src= # 拷贝本地源⽂件到远程，可使⽤绝对路径或相对路径。如果路径是⽬录，且⽬录后加了# 斜杠"/"，则只会拷贝⽬录中的内容到远程，如果⽬录后不加斜杠，则拷贝⽬录本⾝和# ⽬录内的内容到远程。**

默认情况下，ansible copy会检查⽂件md5查看是否需要拷贝，相同则不会拷贝，否则会拷贝。如果设置

f orce=yes，则当⽂件md5不同时(即⽂件内容不同)才覆盖拷贝，设置f orce=no时，则只拷贝对⽅没有的⽂件。

**ansible centos -m copy -a "src=/tmp/copy.txt dest=/tmp mode=0770 owner=sshd group=sshd backup=yes" -o -f 6**

**ll /tmp**

**-rwxrwx--- 1 sshd sshd 24 May 28 16:45 copy.txt**

**-rwxrwx--- 1 root root 22 May 28 16:39 copy.txt.10915.2017-05-28@16:45:14~ #这是备份⽂件**

如果拷贝的是⽬录，则⽬标路径必须是⽬录路径。如果使⽤"/"结尾，则拷贝的是⽬录中的⽂件，如果不以斜杠结 尾，则拷贝的是⽬录加⽬录中的⽂件。

**mkdir /tmp/a**

**touch /tmp/a/{1..10}.txt**

**ansible centos -m copy -a "src=/tmp/a dest=/tmp" -o -f 6**

### template模块

template模块⽤法和copy模块⽤法基本⼀致，它主要⽤于复制配置⽂件。

**ansible-doc -s template**

**- name: Templates a file out to a remote server. action: template**

**backup # 拷贝的同时也创建⼀个包含时间戳信息的备份⽂件，默认为no dest= # ⽬标路径**

**force # 设置为yes (默认)时，将覆盖远程同名⽂件。设置为no时，忽略同名⽂件的拷贝group # 设置远程⽂件的所属组**

**owner # 设置远程⽂件的所有者**

**mode # 设置远程⽂件的权限。使⽤数值表⽰时不能省略第⼀位，如0644。**

**# 也可以使⽤'u+rwx' or 'u=rw,g=r,o=r'等⽅式设置**

**src= # ansible控制器上Jinja2格式的模板所在位置，可以是相对或绝对路径validate # 在复制到⽬标主机后但放到⽬标位置之前，执⾏此选项指定的命令。**

**# ⼀般⽤于检查配置⽂件语法，语法正确则保存到⽬标位置。**

**# 如果要引⽤⽬标⽂件名，则使⽤%s，下⾯的⽰例中的%**s**即表⽰⽬标机器上的/etc/nginx/nginx.conf。**

**ansible centos -m template -a "src=/tmp/nginx.conf.j2 dest=/etc/nginx/nginx.conf mode=0770 owner=root group=root ba ckup=yes validate='nginx -t -c %s'" -o -f 6**

虽然template模块可以按需求修改配置⽂件内容来复制模板到被控主机上，但是有⼀种情况它是不能解决的：不同被控节点所需的配置⽂件差异很⼤，并⾮修改⼏个变量就可以满⾜。例如在centos 6和centos 7上通过yum安装的nginx，它们的配置⽂件内容相差⾮常⼤，且centos 6上的nginx的默认就有⼀个/etc/nginx/conf .d/def ault.conf 。如果直接复制同⼀个模板的nginx配置⽂件到centos 6和centos 7上，很可能导致某⼀版本的nginx不能启动。

这时就有必要在复制模板时挑选对应发⾏版的模板⽂件进⾏配对复制，例如要复制到centos 6上的源模板是nginx6.co nf .j2，复制到cent os 7上的源模板是nginx7 .co nf .j2。这种⾏为可以称之为"基于变量选择文件或模



板**"。**

**---**

**- tasks:**

**- name: template file based var**

**template: src=/templates/nginx{{ ansible\_distribution\_major\_version }}.conf.j2 dest=/etc/nginx/nginx.conf va lidate="/usr/sbin/nginx -t -c %s"**



还可以在⽂件内容中指定jinja2的替代变量，在ansible执⾏时⾸先会根据变量内容进⾏渲染，渲染后再执⾏相关模 块。例如，此处的template模块，复制⼀个基于发⾏版本号的yum源配置⽂件。以下是某个repo⽂件模板

base.repo.j2的内容。

**[epel] name=epel**

**baseurl=**[**http://mirrors.aliyun.com/epel/**](http://mirrors.aliyun.com/epel/)**{{ ansible\_distribution\_major\_version }}Server/x86\_64/ enable=1**

**gpgcheck=0**

再复制即可。

**---**

**- tasks:**

**- template: src=my.repo.j2 dest=/etc/yum.repos.d/my.repo**

### ⽂件模块file

管理⽂件、⽬录的属性，也可以创建⽂件或⽬录。

**ansible-doc -s file**

**- name: Sets attributes of files action: file**

**group**

**owner mode path= recurse src state**

**# file/directory的所属组# file/directory的所有者**

**# 修改权限，格式可以是0644、'u+rwx'或'u=rw,g=r,o=r'等# 指定待操作的⽂件，可使⽤别名'dest'或'name'来替代path # (默认no)递归修改⽂件的属性信息，要求state=directory # 创建链接时使⽤，指定链接的源⽂件**

**# directory:如果⽬录不存在则递归创建**

**# file:⽂件不存在时，不会被创建(默认值)**

**# touch:touch由path指定的⽂件，即创建⼀个新⽂件，或修改其mtime和atime # link:修改或创建软链接**

**# hard:修改或创建硬链接**

**# absent:⽬录和其中的⽂件会被递归删除，⽂件或链接将取消链接状态**

需要注意的是，f ile模块可以递归创建⽬录，但是不能在不存在的⽬录中创建⽂件，只能先创建⽬录，再在此⽬录中创建⽂件。

创建⽬录，并递归修改⽬录的属性。

**ansible 192.168.100.63 -m file -a 'path=/tmp/xyz state=directory owner=root group=root mode=0755 recurse=yes'**

修改⽬录中属性。

**ansible 192.168.100.63 -m file -a 'path=/tmp/xyz state=touch mode=0644'**

创建或修改⽂件属性。

**ansible 192.168.100.63 -m file -a 'path=/tmp/xyz/a.txt state=touch mode=0644'**

### fetch拉取⽂件模块

和copy⼯作⽅式类似，只不过是从远程主机将⽂件拉取到本地端，存储时使⽤主机名作为⽬录树，且只能拉取⽂件 不能拉取⽬录。

**ansible-doc -s fetch**

**- name: Fetches a file from remote nodes action: fetch**

**dest= # 本地存储拉取⽂件的⽬录。例如dest=/data，src=/etc/fstab，**

**# 远程主机名host.exp.com，则保存的路径为/data/host.exp.com/etc/fstab。**

**fail\_on\_missing # 当设置为yes时，如果拉取的源⽂件不存在，则此任务失败。默认为no。**

**flat # 改变拉取后的路径存储⽅式。如果设置为yes，且当dest以"/"结尾时，将直接把源⽂件**

**# 的basename存储在dest下。显然，应该考虑多个主机拉取时的⽂件覆盖情况。**

**src= # 远程主机上的源⽂件。只能是⽂件，不⽀持⽬录。在未来的版本中可能会⽀持⽬录递归拉取。**

**validate\_checksum # fetch到⽂件后，检查其md5和源⽂件是否相同。**

**# 存储为/tmp/localhost/etc/fstab**

**ansible localhost -m fetch -a "src=/etc/fstab dest=/tmp"**

**# 存储为/tmp/fstab**

**ansible localhost -m fetch -a "src=/etc/fstab dest=/tmp/ flat=yes"**

**# 存储为/tmp/fstab-localhost**

**ansible localhost -m fetch -a "src=/etc/fstab dest=/tmp/fstab-{{inventory\_hostname}} flat=yes"**

### rsync模块synchronize

synchronize模块⽤于实现rsync的简单版常⽤功能，它⽆法实现完整版的rsync，毕竟rsync功能太多太细致。如果 要使⽤rsync，还是应该使⽤command或shell模块来调⽤rsync命令。

完整的rsync功能见rsync命令中⽂⼿册。

**ansible-doc -s synchronize**

**- name: A wrapper around rsync to make common tasks in your playbooks quick and easy. action: synchronize**

**src= # 指定待传输的源⽂件。可以是相对路径，也可以是绝对路径。**

**dest= # ⽬标路径。可以是绝对路径，也可以是相对路径。**

**mode # 指定推(push)还是拉(pull)的传输模式。**

**# push时，本地为sender端，pull时，远程为sender端。默认为push。**

**archive # 等价于rsync的"-a"选项，即使⽤归档模式。它等价于rsync的"-rtopgDl"选项。值为yes/no。**

**times # 保留mtime属性。**

**group # 保留所属组属性。**

**owner # 保留所有者属性。**

**links # 拷贝链接⽂件⾃⾝。**

**perms # 保留权限属性。**

**recursive # 递归到⽬录中的⽂件。**

**compress # 传输过程中压缩传输。应该总是开启，除⾮遇到问题。即rsync的"-z"选项。**

**copy\_links # 拷贝软链接的⽂件名和其指向的⽂件的内容。即a指向b⽂件时，将在⽬标端⽣成a普通**

**# ⽂件，但此⽂件中的内容是b中的内容。**

**dirs # ⾮递归⽅式传输⽬录。**

**delete # ⽬标端如果⽐源端⽂件多，则删除这些多出来的⽂件，要求recursive=yes。**

**checksum # 等价于"-c"选项，将基于⽂件的checksum来判断是否同步，⽽不是默认的quick check # 算法，该算法基于⽂件⼤⼩和最近的mtime来判断是否要同步。该选项会⼤幅降低效率， # 应谨慎使⽤。注意，它⽆法影响archive，即archive仍会启⽤。**

**existing\_only # receiver端没有的⽂件不同步。但仍会传输，只是临时⽂件重组后不重命名⽽已。partial # 等价于"--partial"选项。默认rsync在传输中断时会删除传输了⼀半的⽂件，指定该选**

**# 项将保留这部分不完整的⽂件，使得下次传输时可以直接从未完成的数据块开始传输。**

**dest\_port # ssh的连接端⼝。**

**rsync\_opts # 指定额外的rsync选项。使⽤数组的⽅式传递这些选项。**

**rsync\_path # 等价于"--rsync-path"选项，⽬的是启动远程rsync。**

**# 例如可以指定[--rsync-path=rsync]，甚⾄[--rsync-path=cd /tmp/c && rsync]。# 当不指定rsync路径时，默认为/usr/bin/rysnc。**

**rsync\_timeout # 指定rsync在多久时间内还没有数据传输就超时退出。**

**verify\_host # 对⽬标主机进⾏ssh的host key验证。**

### 包管理模块yum

**ansible-doc -s yum**

**- name: Manages packages with the `yum' package manager action: yum**

**disable\_gpg\_check # 安装包时禁⽌gpgcheck，仅在state=present或latest时⽣效。**

**disablerepo # 禁⽤指定的repoid，多个repoid使⽤逗号分隔。**

**enablerepo # 明确使⽤该repoid**

**exclude # 排除哪些包不安装，仅在state=present或latest时⽣效。**

**list # 类似于yum list。**

**name= # 指定安装的包名，可带上版本号。多个包可使⽤逗号分隔。**

**state # 状态。('present'、'installed','latest')⽤于安装包，**

**# ('absent'、'removed')⽤于移除已安装包。update\_cache # 强制更新yum的cache。**

name需要配合state来使⽤，如果state指定为present/installed/latest将安装包，其中latest是安装最新包，默认 为present。如果指定为absent/removed则⽤于卸载包。

在ansible中，很多地⽅都会出现present和absent的状态，它们⼀般都表⽰⽬标是否应该存在还是不存在，也就是 要进⾏的动作是创建和删除。

列出和ansible相关的包。安装包。

**ansible centos -m yum -a "list=ansible" -f 6**

**ansible centos -m yum -a "name=dos2unix state=installed" -o -f 6**

安装本地的包，且排除某些包不安装。

**ansible centos -m yum -a "name=/tmp/\*.rpm exclude=\*unix\* state=present"**

卸载包。

**ansible centos -m yum -a "name=dos2unix state=removed" -o -f 6**

### 配置yum源模块yum\_repository

⽤于配置yum源。可以实现⾮常完整的yum仓库配置。但是⼀般只需简单的添加yum源即可。所以，以下是简单版 的⽤法和⽰例。

**ansible-doc -s yum\_repository**

**- name: Add or remove YUM repositories action: yum\_repository**

**baseurl # 地 址**

**mirrorlist # 设置mirrorlist地址**

**description # 描 述 信 息**

**enabled**

**file**

**# 是否启⽤该仓库，默认为yes**

**# 保存此仓库的⽂件，不设置该项的话则默认以name选项中的名称命名，将⾃动以".repo"后缀结尾。**

**gpgcheck # 是否要进⾏gpgcheck**

**name= # 仓库的名称，要保证名称的唯⼀性**

**reposdir # 保存.repo⽂件的⽬录，默认/etc/yum.repos.d/ state # repo⽂件的状态，present/absent，默认present。**

例如：

* **name: Add repository yum\_repository:**

**name: aliyun\_epel description: EPEL YUM repo**

**baseurl:** [**http://mirrors.aliyun.com/epel/7/$basearch/**](http://mirrors.aliyun.com/epel/7/%24basearch/)

* **name: Add multiple repositories into a file yum\_repository:**

**name: epel**

**description: EPEL YUM repo file: sohu\_epel**

**baseurl:** [**http://mirrors.sohu.com/fedora-epel/7/$basearch/**](http://mirrors.sohu.com/fedora-epel/7/%24basearch/) **gpgcheck: no**

**ansible 192.168.100.63 -m yum\_repository -a 'name=aliyun\_epel description="epel repo" baseurl=**[**http://mirrors.aliyun**](http://mirrors.aliyun/)

**.com/epel/7/$basearch/ gpgcheck=no enabled=yes'**

查看⽣成的repo⽂件。

**[root@server2 ~]# ansible 192.168.100.63 -m shell -a "cat /etc/yum.repos.d/aliyun\_epel.repo" 192.168.100.63 | SUCCESS | rc=0 >>**

**[aliyun\_epel]**

**baseurl =** [**http://mirrors.aliyun.com/epel/7/$basearch/**](http://mirrors.aliyun.com/epel/7/%24basearch/) **enabled = 1**

**gpgcheck = 0 name = epel repo**

### 服务管理模块service

**ansible-doc -s service**

**- name: Manage services. action: service**

**enabled # 设置服务为开机⾃启动，默认为no name= # 服务名**

**state # 'started'和'stoped'分别启动和停⽌服务，它们是幂等操作，多次启动或停⽌服务的结果是⼀样的， # 也就是说对于运⾏中的服务不会再执⾏启动操作，同理停⽌也是⼀样。'restarted'总是重启服务，**

**# 'reloaded'总是重读配置⽂件，如果服务是未运⾏状态，则'reloaded会启动服务。# (state和enabled两者⾄少要给⼀个)**

设置httpd开机⾃启动。

**ansible centos -m service -a 'name=httpd enabled=yes' -f 6 -o**

启动httpd服务。

**ansible centos -m service -a 'name=httpd state=started' -f 6 -o**

### systemd模块

管理systemd风格的服务。

**ansible-doc -s systemd**

**- name: Manage services. action: systemd**

**daemon\_reload # 在执⾏所有动作之前，先确定是否要reload⼀次。值为yes/no。**

**enabled # 是否设置开机⾃启动。**

**masked # 是否将此unit做mask(隐藏、掩盖)处理。mask后的unit将⽆法启动。**

**name= # 待操作服务名。可以是name，也可以是name.service。**

**state # 'started'/'stopped'具有幂等性。但restarted和reloaded总是会执⾏。**

### ⽤户管理模块user

同理还有组管理模块group，就不多做说明了。同样，创建⽤户时，默认会创建同名group。

**ansible-doc -s user**

**- name: Manage user accounts action: user**

**name= # 要创建、修改、移除的⽤户名。**

**password # 设置⽤户密码。此处只能使⽤加密密码作为值。**

**system # 设置为yes表⽰创建⼀个系统⽤户，只能⽤于创建，不能⽤于修改已有⽤户为系统⽤户。**

**state # 创建⽤户(present)还是删除⽤户(absent)。默认为present。**

**createhome # 创建家⽬录，或者已有的⽤户但家⽬录不存在也会创建。设置为no则不创建家⽬录。**

**home # 指定要创建的家⽬录路径**

**move\_home # 如果设置为yes，则"home="则表⽰将家⽬录移动到此选项指定的路径下。**

**uid # 设置⽤户的uid**

**group # 设置⽤户的primary group**

**groups # 将⽤户加⼊到辅助组列表中。如果设置"groups="，则会将此⽤户从所有辅助组中移除。**

**shell # 设置⽤户的shell。**

**force # 配合'state=absent'时，等价于'userdel --force'，即强制删除⽤户、家⽬录和邮件列表。remove # 配合'state=absent'时，等价于'userdel --remove'，即删除家⽬录和邮件列表。update\_password # user是幂等模块，"always"将总是修改密码。"on\_create"将只在创建⽤户时设置密码。**

创建系统⽤户，并指定shell。删除⽤户。

**ansible centos -m user -a "name=longshuai system=yes shell=/sbin/nologin"**

**ansible centos -m user -a "name=longshuai state=absent"**

**指定update\_password=always 将总是修改⽤户的密码，不管该⽤户是否已存在。⽽update\_password=on\_create 则只有新创建⽤户时才设置密码，如果⽤户已存在，则不会修改该⽤户的密码。默认值就是always。**

**openssl passwd -1 123456**

**$1$9jwmFoVU$MVz7ywscpoPS5WXC.srcP/**

**ansible centos -m user -a 'name=longshuai3 password="$1$9jwmFoVU$MVz7ywscpoPS5WXC.srcP/" update\_password=always'**

创建⽤户并指定密码，但如果⽤户已存在则不修改密码。

**openssl passwd -1 234567**

**$1$J9Nt0scL$R9Db5Pi1AJ7FQv4Xzia0w/**

**ansible centos -m user -a 'name=longshuai3 password="$1$J9Nt0scL$R9Db5Pi1AJ7FQv4Xzia0w/" update\_password=on\_create'**

### authorzied\_key模块

例如将公钥添加到centos组中的机器中。使⽤- k选项⽤于询问ssh连接的密码。要想使⽤该模块实现⾮交互，需要在inventory⽂件中指定主机或主机组中加

**ansible-doc -s authorized\_key**

**- name: Adds or removes an SSH authorized key action: authorized\_key**

**# 公钥路径，可以是本地⽂件，可以是url地址。**

**# 本地⽂件时使⽤{{ lookup('file', '~/.ssh/id\_rsa.pub') }}， # url使⽤https://github.com/username.keys。**

**manage\_dir # 是否创建或修改⽬标authorized\_keys所在⽬录的所有者和权限。默认为yes。**

**# 使⽤⾃定义的⽬标路径时，必须设置为no或false**

**key=**

**path # authorized\_keys所在的⽬录，默认为家⽬录下的.ssh⽬录中**

**state # present/absent，是否将密钥添加到⽬标authorized\_keys⽂件中。user= # 添加到远程哪个⽤户下的authorized\_keys⽂件**

**ansible centos -m authorized\_key -a "key={{lookup('file','~/.ssh/id\_rsa.pub')}} state=present user=root" -k**

上"ansible\_ssh\_pass='PASSWORD'"参数。例如：

**192.168.100.65 ansible\_ssh\_pass='123456' [centos6]**

**host1 host2 host3**

**[centos6:vars] ansible\_ssh\_pass='123456'**

但这样不会将主机密钥添加到主控制端的known\_hosts⽂件中，因此下次执⾏ansible时会失败。可以在配置⽂件 中做如下设置：

**host\_key\_checking = False**

或者⼲脆使⽤expect⾮交互更⽅便。

### debug模块

⽤于输出⾃定义的信息，类似于echo、print等输出命令。ansible中的debug主要⽤于输出变量值、表达式值，以 及⽤于when条件判断时。使⽤⽅式⾮常简单。

**ansible-doc -s debug**

**- name: Print statements during execution action: debug**

**msg**

**var**

**# 输出⾃定义信息。如果省略，则输出普通字符。**

**# 指定待调试的变量。只能指定变量，不能指定⾃定义信息，且变量不能加{{}}包围，⽽是直接的变量名。**

**verbosity # 控制debug运⾏的调试级别，有效值为⼀个数值N。**

例如：

**ansible centos -m debug -a 'msg="i want to print this messages"' ansible centos -m debug -a 'var=ansible\_eth0.ipv4.address'**

可以输出变量值，不过⼀般使⽤到变量的时候都会在playbook中使⽤debug模块，以下是⼀个⽰例：

**tasks:**

**- name: print any messages**

**debug: msg="you name is {{ name }}"**

### 定时任务模块cron

cron模块⽤于设置定时任务，也⽤于管理定时任务中的环境变量。

**ansible-doc -s cron**

**- name: Manage cron.d and crontab entries. action: cron**

**backup # (yes/no)如果设置了，则会在修改远程cron\_file前备份这些⽂件**

**cron\_file # ⾃定义cron\_file的⽂件名，使⽤相对路径则表⽰在/etc/cron.d中。必须同时指定user选项**

**user # 指定哪个⽤户的crontab将要被修改，默认为root disabled # 禁⽤crontab中的某个job，要求state=present**

**env # (yes/no)设置⼀个环境变量，将添加在crontab的顶端。使⽤name和value定义变量名和值**

**job # 需要执⾏的命令。如果设置了env，则表⽰环境变量的值，此时job="XXXX"等价于value="XXXX"。**

**# 要求state=present**

**minute # 分(0-59, \*, \*/N)，不写时，默认为\* hour # 时(0-23, \*, \*/N)，不写时，默认为\***

**day # ⽇(1-31, \*, \*/N)，不写时，默认为\***

**month # ⽉(1-12, \*, \*/N)，不写时，默认为\***

**weekday # 周(0-6 for Sunday-Saturday, \*)，不写时，默认为\***

**name # 描述crontab任务的字符串。但如果设置的是env，则name为环境变量的名称。要求state=absent # 注意，若未设置name，且state=present，则总会创建⼀个新job条⽬，即使cron\_file中已经存在# 同样的条⽬**

**special\_time # 定时任务的别称，⽤于定义何时运⾏job条⽬。**

**# 有效值有reboot/hourly/daily/weekly/monthly/yearly/annually。**

**state # job或者env的状态是present(默认)还是absent。present⽤于创建，absent⽤于移除**

除了cron模块本⾝可以管理cron的环境变量，另⼀个模块cronvar也可以定义定时任务的环境变量。

**ansible-doc -s cronvar**

**- name: Manage variables in crontabs action: cronvar**

**backup**

**cron\_file state user value**

**# (yes/no)如果设置了，则会在修改远程cron\_file前备份这些⽂件# ⾃定义cron\_file的⽂件名，使⽤相对路径则表⽰在/etc/cron.d中# present⽤于创建变量，absent⽤于移除变量**

**# 指定哪个⽤户的crontab将要被修改，默认为root # 环境变量的值，要求state=present**

例如：创建⼀个job，每2分钟进⾏⼀次时间同步，并且⾃定义cron\_f ile。

**ansible centos7 -m cron -a 'name="ntpdate" job="/usr/sbin/ntpdate ntp1.aliyun.com" cron\_file=ntpdate\_cron user=root minute=\*/2' -o**

验证是否添加正确。

**ansible centos7 -m shell -a 'cat /etc/cron.d/ntpdate\_cron' 192.168.100.65 | SUCCESS | rc=0 >>**

**#Ansible: ntpdate**

**\*/2 \* \* \* \* root /usr/sbin/ntpdate ntp1.aliyun.com**

* + - 1. **| SUCCESS | rc=0 >>**

**#Ansible: ntpdate**

**\*/2 \* \* \* \* root /usr/sbin/ntpdate ntp1.aliyun.com**

* + - 1. **| SUCCESS | rc=0 >>**

**#Ansible: ntpdate**

**\*/2 \* \* \* \* root /usr/sbin/ntpdate ntp1.aliyun.com**

移除⼀个job，要求name必须匹配。如有必要，需要同时指定cron\_f ile和user。

**ansible centos7 -m cron -a 'name="ntpdate" state=absent cron\_file=ntpdate\_cron user=root' -o**

设置crontab环境变量，并定义⼀个job。

**ansible centos7 -m cron -a 'env=yes name=app\_home value=/tmp' -o**

**ansible centos7 -m cron -a 'name="ntpdate" job="/usr/sbin/ntpdate ntp1.aliyun.com" minute=\*/2' -o**

验证设置结果。

**ansible centos7 -m shell -a 'crontab -l' 192.168.100.65 | SUCCESS | rc=0 >>**

**app\_home="/tmp" #Ansible: ntpdate**

**\*/2 \* \* \* \* /usr/sbin/ntpdate ntp1.aliyun.com**

* + - 1. **| SUCCESS | rc=0 >>**

**app\_home="/tmp" #Ansible: ntpdate**

**\*/2 \* \* \* \* /usr/sbin/ntpdate ntp1.aliyun.com**

* + - 1. **| SUCCESS | rc=0 >>**

**app\_home="/tmp" #Ansible: ntpdate**

**\*/2 \* \* \* \* /usr/sbin/ntpdate ntp1.aliyun.com**

### 归档模块archive

⽤于在远端压缩⽂件。当然，前提是在远端主机上要有对应的压缩⼯具。⽀持zip/gz/tar/bz2。

**ansible-doc -s archive**

**- name: Creates a compressed archive of one or more files or trees. action: archive**

**dest # ⽬标归档⽂件名。除⾮path指定要压缩的是单⽂件，否则需要dest选项**

**format # 指定压缩格式，默认为gz格式**

**group # ⽂件/⽬录的所属组**

**owner # ⽂件/⽬录的所有者**

**mode # 设置⽂件/⽬录的的权限，⽀持'0644'或'u+rwx'或'u=rw,g=r,o=r'等格式path= # 要压缩的⽂件，可以是绝对路径，也可以是glob统配的路径，还可以是⽂件列表remove # 压缩后删除源⽂件**

例如：

**# 将⽬录/path/to/foo/压缩为/path/to/foo.tgz**

* **archive:**

**path: /path/to/foo dest: /path/to/foo.tgz**

**# 压缩普通⽂件/path/to/foo为/path/to/foo.gz并删除源⽂件，由于压缩的是单⽂件，所以可以省略dest选项**

* **archive:**

**path: /path/to/foo remove: True**

**# 将单⽂件/path/to/foo压缩为zip格式**

* **archive:**

**path: /path/to/foo format: zip**

**# 将给定的⽂件列表压缩为bz2格式，压缩包路径为/path/file.tar.bz2**

* **archive:**

**path:**

* + **/path/to/foo**
  + **/path/wong/foo dest: /path/file.tar.bz2 format: bz2**

### 解包模块unarchive

默认复制ansible端的归档⽂件到被控主机，然后在被控主机上进⾏解包。如果设置选项remote\_src=yes，则表⽰ 解包被控主机上的归档⽂件。

要求在被控主机上有对应的解包命令。unzip命令⽤于解压".zip"⽂件，gtar(tar包提供)命令⽤于解压".tar"、".tar.gz"、".tar.bz2"和".tar.xz"。

**ansible-doc -s unarchive**

**- name: Unpacks an archive after (optionally) copying it from the local machine. action: unarchive**

**creates # 如果指定的⽂件存在则不执⾏该任务。可⽤于实现幂等性dest= # 远程机器上需要被解包的归档⽂件，要求是绝对路径exclude # 列出解包过程中想要忽略的⽬录和⽂件**

**group # ⽂件/⽬录的所属组**

**owner # ⽂件/⽬录的所有者**

**mode # 设置⽂件/⽬录的的权限，⽀持'0644'或'u+rwx'或'u=rw,g=r,o=r'等格式**

**keep\_newer # 在解包过程中，如果⽬标路径中和包中有同名⽂件，且⽐包中的⽂件更新，则保留新的⽂件**

**list\_files # 设置为true时，将返回归档⽂件中的⽂件列表**

**remote\_src # 设置为yes表⽰远程主机上已有⽬标归档⽂件，即不再从本地复制归档⽂件到远端，直接在远端解包。**

**# 默认为no**

**src= # 如果remote\_src=no,将复制本地归档⽂件到远端，可相对路径也可绝对路径. 如果remote\_src=yes, 将解包远程已存在的归档⽂件**

**如果remote\_src=yes且src中包含了"://",将指挥远程主机从url中下载⽂件并解包**

例如：

**# 复制ansible端的foo.tgz⽂件到远端并解包**

**- unarchive:**

**src: foo.tgz dest: /var/lib/foo**

**# 直接解包远端已经存在的⽂件- unarchive: src: /tmp/foo.zip**

**dest: /usr/local/bin remote\_src: True**

**# 从url上下载压缩包，然后进⾏解压**

**- unarchive:**

**src: https://example.com/example.zip dest: /usr/local/bin**

**remote\_src: True**

### 下载模块get\_url

**ansible-doc -s get\_url**

**- name: Downloads files from HTTP, HTTPS, or FTP to node action: get\_url**

**backup # 下载⽂件时同时创建⼀个名称中包含时间戳的备份⽂件**

**dest= # ⽂件保存路径，必须为绝对路径。**

**# 如果dest是⼀个⽬录，则使⽤url的base name作为⽂件名# 如果dest是⼀个⽬录，则'force'选项不⽣效**

**# 如果dest是⼀个⽬录，则总是会下载⽬标⽂件，但只在已存在的⽂件变化了才会替换旧⽂件**

**force # 如果设置为yes，且dest不是⼀个⽬录时，则总是会下载⽂件，但只在已存在的⽂件变化了才会替换旧⽂件**

**# 如果设置为no(默认)，则只会在⽬录路径下不存在该⽂件时才会进⾏下载。**

**tmp\_dest # 下载时临时存放⽬录，在任务执⾏完成前会删除下载的临时⽂件**

**group # ⽂件/⽬录的所属组**

**owner # ⽂件/⽬录的所有者**

**mode # 设置⽂件/⽬录的的权限，⽀持'0644'或'u+rwx'或'u=rw,g=r,o=r'等格式**

**timeout # 请求url时的超时时间，默认10秒钟**

**url= # 要下载的url路径，(http|https|ftp)://[user[:pass]]@host.domain[:port]/path # 还⽀持file格式的路径，实现复制功能。file:///path/to/file**

注意，dest为⽬录或者f orce=yes时，总是会下载⽂件到临时存放⽬录中，只不过不⼀定会替换旧⽂件。只有f orce=no(默认)且dest是⼀个⽂件时，在⽂件已存在时才不会下载⽂件。

例如：

**# 下载foo.conf，若/etc/foo.conf已存在，则不下载该⽂件**

**get\_url:**

**url:** [**http://example.com/path/file.conf**](http://example.com/path/file.conf) **dest: /etc/foo.conf**

**mode: 0440**

**# 下载foo.conf到/tmp⽬录下，若/tmp/foo.conf已存在则检查⽂件是否相同，不相同则进⾏替换**

**get\_url:**

**url:** [**http://example.com/path/file.conf**](http://example.com/path/file.conf) **dest: /tmp/**

**# 复制⼀个本地⽂件**

**get\_url:**

**url: file:///tmp/afile.txt dest: /tmp/afilecopy.txt**

### wait\_for模块

有些时候任务之间对状态、⽂件、端⼝等资源是有依赖关系的，只有满⾜了前提，任务才会继续。wait\_f or模块就是⽤于判断任务在满⾜什么条件的情况下会继续。主要⽤来判断端⼝是否开启、⽂件是否存在、⽂件中是否存在某些字符串。

**ansible-doc -s wait\_for**

**- name: Waits for a condition before continuing. action: wait\_for**

**delay # 在检查操作进⾏之前等待的秒数**

**host # 等待这个主机处于启动状态，默认为127.0.0.1 port # 等待这个端⼝已经开放**

**path # 这个⽂件是否已经存在**

**search\_regex # 在⽂件中进⾏正则匹配**

**state # present/started/stopped/absent/drained.默认started**

**当检查的是⼀个端⼝时： started:保证端⼝是开放的stopped:保证端⼝是关闭的当检查的是⼀个⽂件时：**

**present/started:在检查到⽂件存在才会继续absent:检查到⽂件被移除后才会继续**

**sleep # 两次检查之间sleep的秒数，默认1秒**

**timeout # 检查的等待超时时间(秒数，默认300)**

例如：

**# 连接上主机后10秒后才检查8000端⼝是否处于开放状态，300秒(默认值)内未开放则超时。**

**- wait\_for:**

**port: 8000**

**delay: 10**

**# 直到/tmp/foo⽂件存在才会继续**

* **wait\_for:**

**path: /tmp/foo**

**# 直到/tmp/foo⽂件中能匹配"completed"字符串才继续**

* **wait\_for:**

**path: /tmp/foo search\_regex: completed**

**# 直到/var/lock/file.lock这个锁⽂件被移除了才继续**

* **wait\_for:**

**path: /var/lock/file.lock state: absent**

**# 直到/proc/3466/status⽂件被移除才继续，可⽤来判断进程是启动还是停⽌，pid⽂件是存在还是被移除等**

* **wait\_for:**

**path: /proc/3466/status state: absent**

### script模块

script模块⽤于控制远程主机执⾏脚本。在执⾏脚本前，ansible会将本地脚本传输到远程主机，然后再执⾏。在执

⾏脚本的时候，其采⽤的是远程主机上的shell环境。

**ansible-doc -s script**

**- name: Runs a local script on a remote node after transferring it action: script**

**chdir # 在远程执⾏脚本前先切换到此⽬录下。**

**creates # 当此⽂件存在时，不执⾏脚本。可⽤于实现幂等性。**

**removes # 当此⽂件不存在时，不执⾏脚本。可⽤于实现幂等性。**

**free\_form= # 本地待执⾏的脚本路径、选项、参数。之所以称为free\_form，是因为它是脚本名+选项+参数。**

例如，将ansible端/tmp/a.sh发送到各被控节点上执⾏，但如果被控节点的/tmp下有hello.txt，则不执⾏。

**---**

**- hosts: centos**

**remote\_user: root tasks:**

**- name: execute /tmp/a.sh,but only /tmp/hello.txt is not yet created script: /tmp/a.sh hello**

**args:**

**creates: /tmp/hello.txt**

# YAML语法和playbook写法

ansible的playbook采⽤yaml语法，它简单地实现了json格式的事件描述。yaml之于json就像markdown之于html⼀样，极度简化了json的书写。在学习ansible playbook之前，很有必要把yaml的语法格式、引⽤⽅式做个梳理。

## 初步说明

以⼀个简单的playbook为例，说明yaml的基本语法。

**---**

**- hosts: 192.168.100.59,192.168.100.65**

**remote\_user: root pre\_tasks:**

**- name: set epel repo for Centos 7 yum\_repository:**

**name: epel7**

**description: epel7 on CentOS 7**

**baseurl:** [**http://mirrors.aliyun.com/epel/7/$basearch/**](http://mirrors.aliyun.com/epel/7/%24basearch/) **gpgcheck: no**

**enabled: True**

**tasks:**

**# install nginx and run it**

* **name: install nginx**

**yum: name=nginx state=installed update\_cache=yes**

* **name: start nginx**

**service: name=nginx state=started**

**post\_tasks:**

* **shell: echo "deploy nginx over" register: ok\_var**
* **debug: msg="{{ ok\_var.stdout }}"**

1. **yaml⽂件以 --- 开头，以表明这是⼀个yaml⽂件，就像xml⽂件在开头使⽤ <?xml version="1.0" encoding="utf-8"?**

> 宣称它是xml⽂件⼀样。但即使没有使⽤ --- 开头，也不会有什么影响。

1. **yaml中使⽤"#"作为注释符，可以注释整⾏，也可以注释⾏内从"#"开始的内容。**
2. **yaml中的字符串通常不⽤加任何引号，即使它包含了某些特殊字符。但有些情况下，必须加引号，最常见的是 在引⽤变量的时候。具体见后⽂。**
3. **关于布尔值的书写格式，即true/f alse的表达⽅式。其实playbook中的布尔值类型⾮常灵活，可分为两种情况：**

模块的参数： 这时布尔值作为字符串被ansible解析。接受yes/on/1/true/no/of f /0/f alse，这时被ansible解析。例如上⾯⽰例中的update\_cache=yes 。

⾮模块的参数： 这时布尔值被yaml解释器解析，完全遵循yaml语法。接受不区分⼤⼩写的

**true/yes/on/y/f alse/no/of f /n。例如上⾯的gpgcheck=no 和enabled=True 。**

建议遵循ansible的官⽅规范，模块的布尔参数采⽤yes/no，⾮模块的布尔参数采⽤True/False。

## 列表

使⽤"- "(减号加⼀个或多个空格)作为列表项，也就是json中的数组。yaml的列表在playbook中极重要，必须得搞

清楚它的写法。例如：

* **zhangsan**
* **lisi**
* **wangwu**

还⽀持内联写法：使⽤中括号。它们等价于json格式的：

**[zhangsan,lisi,wangwu]**

**[**

**"zhangsan",**

**"lisi", "wangwu"**

**]**

再例如：

* **班名: 初中1班**

**⼈数: 35**

**班主任: 隔壁⽼张今天的任务: 扫操场**

* **班名: 初中2班**

**⼈数: 38**

**班主任: 隔壁⽼王今天的任务: 搬桌⼦**

具体在ansible playbook中，列表所描述的是局部环境，它不⼀定要有名称，不⼀定要从同⼀个属性开始，只要使

⽤"- "，它就表⽰圈定⼀个范围，范围内的项都属于该列表。例如：

**---**

**- name: list1**

**hosts: localhost remote\_user: root tasks:**

**# 列表1，同时给了个名称**

**# 指出了hosts是列表1的⼀个对象**

**# 列表1的属性**

**# 还是列表1的属性**

**- hosts: 192.168.100.65 # 列表2，但是没有为列表命名，⽽是直⼊主题**

**remote\_user: root**

**sudo: yes tasks:**

唯⼀要注意的是，每⼀个playbook中必须包含"hosts"和"tasks"项。更严格地说，是每个play的顶级列表必须包含 这两项。就像上⾯的例⼦中，就表⽰该playbook中包含了两个play，每个play的顶级列表都包含了hosts和

tasks。其实绝⼤多数情况下，⼀个playbook中都只定义⼀个play，所以只有⼀个顶级列表项。顶级列表的各项， 其实可以将其看作是ansible- playbook运⾏时的选项。

另外，playbook中某项是⼀个动作、⼀个对象或⼀个实体时，⼀般都定义成列表的形式。见下⽂。

## 字典

官⽅⼿册上这么称呼，其实就是key=value的另⼀种写法。使⽤"冒号+空格"分隔，即 key: value 。它⼀般当作列表项的属性。

例如：

**- 班名: 初中1班**

**⼈数:**

**总数: 35**

**男 : 19**

**⼥ : 16**

**班主任:**

**⼤名: 隔壁⽼张这厮多⼤: 39**

**这厮任教多少年: 15**

**今天的任务: 扫操场**

**- 班名: 初中2班**

**⼈数:**

**总数: 38**

**男 : 19**

**⼥ : 19**

**班主任:**

**⼤名: 隔壁⽼王这厮多⼤: 30**

**喜调戏⼥⽼师: True**

**今天的任务: 搬桌⼦未完成任务怎么办:**

* **继续搬，直到完成**
* **写检讨**

具体到playbook中，⼀般"虚拟性"的内容都可以通过字典的⽅式书写，⽽实体化的、动作性的、对象性的内容则应 该定义为列表形式。

**---**

* **hosts: localhost**

**remote\_user: root tasks:**

* + **name: test1**

**shell: echo /tmp/a.txt register: hi\_var**

* + **debug: var=hi\_var.stdout**
  + **include: /tmp/nginx.yml**
  + **include: /tmp/mysql.yml**
  + **copy:**

**src: /etc/resolv.conf dest: /tmp**

**# 列表1**

**# ⼦列表，下⾯是shell模块，是⼀个动作，所以定义为列表，只不过加了个name**

**# 调⽤模块，这是动作，所以也是列表**

**# 同样是动作，包含⽂件**

**# 调⽤模块，定义为列表。但模块参数是虚拟性内容，应定义为字典⽽⾮列表**

**# 模块参数1 # 模块参数2**

**- hosts: 192.168.100.65**

**remote\_user: root vars:**

**nginx\_port: 80**

**mysql\_port: 3306 vars\_files:**

* **nginx\_port.yml tasks:**
* **name: test2**

**shell: echo /tmp/a.txt register: hi\_var**

* **debug: var=hi\_var.stdout**

**# 列表2**

**# 定义变量，是虚拟性的内容，应定义为字典⽽⾮列表**

**# ⽆法写成key/value格式，且是实体⽂件，因此定义为列表**

**# register是和最近⼀个动作绑定的**

从上⾯⽰例的copy模块可以得出，模块的参数是虚拟性内容，也能使⽤字典的⽅式定义。 字典格式的key/value，也⽀持内联格式写法：使⽤⼤括号。

**{⼤名: 隔壁⽼王,这厮多⼤: 30,喜调戏⼥⽼师: True}**

**{nginx\_port: 80,mysql\_port: 3306}**

这等价于json格式的：

**{**

**"⼤名": "隔壁⽼王", "这厮多⼤": 30,**

**"喜调戏⼥⽼师": "True"**

**}**

**{**

**"nginx\_port": 80,**

**"mysql\_port": 3306**

**}**

再结合其⽗项，于是转换成json格式的内容：

**"班主任": {**

**"⼤名": "隔壁⽼王", "这厮多⼤": 30,**

**"喜调戏⼥⽼师": "True"**

**}**

**"vars": {**

**"nginx\_port": 80,**

**"mysql\_port": 3306**

**}**

再加上列表项(使⽤中括号)，于是：

**[**

**{**

**"hosts": "192.168.100.65",**

**"remote\_user": "root", "vars": {**

**"nginx\_port": 80,**

**"mysql\_port": 3306**

**},**

**"vars\_files": [ "nginx\_port.yml"**

**],**

**"tasks": [**

**{**

**"name": "test2",**

**"shell": "echo /tmp/a.txt", "register": "hi\_var"**

**},**

**{**

**"debug": "var=hi\_var.stdout"**

**}**

**]**

**}**

**]**

## 分⾏写

playbook中有3种⽅式进⾏续⾏。

在"key: "的后⾯使⽤⼤于号。

在"key: "的后⾯使⽤竖线。这种⽅式可以像脚本⼀样写很多⾏语句。多层缩进。

例如，下⾯的3中⽅法。

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo 2 >>/tmp/test.txt creates=/tmp/haha.txt**
  + **shell: >**

**echo 2 >>/tmp/test.txt creates=/tmp/haha.txt**

* + **shell: |**

**echo 2 >>/tmp/test.txt**

**echo 3 >>/tmp/test.txt args:**

**creates: /tmp/haha.txt**

**# ⽐模块shell缩进更多**

**# 在"key: "后使⽤⼤于号**

**# 指定多⾏命令**

## 向模块传递参数

模块的参数⼀般来说是key=value格式的，有3种传递的⽅式：

直接写在模块后，此时要求使⽤"key=value"格式。这是让ansible内部去解析字符串。因为可分⾏写，所以有 多种写法。

写成字典型，即"key: value"。此时要求多层缩进。这是让yaml去解析字典。

使⽤内置属性args，然后多层缩进定义参数列表。这是让ansible明确指定⽤yaml来解析。 例如：

**---**

* **hosts: localhost**

**tasks:**

* + **yum: name=unix2dos state=installed # key=value直接传递**
  + **yum:**

**name: unxi2dos**

**state: installed**

**- yum:**

**args:**

**name: unix2dos state:installed**

**# "key: value"字典格式传递**

**# 使⽤args传递**

但要注意，当模块的参数是free\_form 时，即格式不定，例如shell和command模块指定要执⾏的命令，它⽆法写成key/value格式，此时不能使⽤上⾯的第⼆种⽅式。也就是说，下⾯第⼀个模块是正确的，第⼆个模块是错误的，因 为shell模块的命令"echo haha"是⾃由格式的，⽆法写成key/value格式。

**---**

* **hosts: localhost**

**tasks:**

* + **yum:**

**name: unxi2dos state: installed**

* + **shell:**

**echo haha**

**creates: /tmp/haha.txt**

所以，调⽤⼀个模块的⽅式就有了多种形式。例如：

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo 1 >/tmp/test.txt creates=/tmp/haha.txt**
  + **shell: echo 2 >>/tmp/test.txt creates=/tmp/haha.txt**
  + **shell: echo 3 >>/tmp/test.txt args:**

**creates: /tmp/haha.txt**

* + **shell: >**

**echo 4 >>/tmp/test.txt creates=/tmp/haha.txt**

* + **shell: |**

**echo 5.1 >>/tmp/test.txt**

**echo 5.2 >>/tmp/test.txt args:**

**creates: /tmp/haha.txt**

* + **yum:**

**name: dos2unix state: installed**

## playbook和play的关系

⼀个playbook中可以包含多个play。每个play都⾄少包含有tasks和hosts这两项，还可以包含其他⾮必须项，如vars,vars\_f iles,remote\_user等。tasks中可以通过模块调⽤定义⼀系列的action。只不过，绝⼤多数时候，⼀个playbook都只定义⼀个play。

所以，⼤致关系为：

playbook: [play1,play2,play3]

play: [hosts,tasks,vars,remote\_user...] tasks: [module1,module2,...]

也就是说，每个顶级列表都是⼀个play。例如，下⾯的playbook中包含了两个play。

**---**

**- name: list1**

**hosts: localhost remote\_user: root tasks:**

**- hosts: 192.168.100.65**

**remote\_user: root sudo: yes**

**tasks:**

需要注意，有些时候play中使⽤了role，可能看上去没有tasks，这是因为role本⾝就是整合playbook的，所以没有 也没关系。但没有使⽤role的时候，必须得包含hosts和tasks。例如：

**---**

**- hosts: centos remote\_user: root pre\_tasks:**

**- name: config the yum repo for centos 7 yum\_repository:**

**name: epel description: epel**

**baseurl:** [**http://mirrors.aliyun.com/epel/7/$basearch/**](http://mirrors.aliyun.com/epel/7/%24basearch/) **gpgcheck: no**

**when: ansible\_distribution\_major\_version == "7"**

**- name: config the yum repo for centos 6 yum\_repository:**

**name: epel description: epel**

**baseurl:** [**http://mirrors.aliyun.com/epel/6/$basearch/**](http://mirrors.aliyun.com/epel/6/%24basearch/) **gpgcheck: no**

**when: ansible\_distribution\_major\_version == "6"**

**roles:**

**- nginx**

**post\_tasks:**

* **shell: echo 'deploy nginx/mysql over' register: ok\_var**
* **debug: msg='{{ ok\_var.stdout }}'**

## playbook中什么时候使⽤引号

playbook中定义的都是些列表和字典。绝⼤多数时候，都不需要使⽤引号，但有两个特殊情况需要考虑使⽤引号。

出现⼤括号"{}"。

出现冒号加空格时": "。

⼤括号要使⽤引号包围，是因为不使⽤引号时会被yaml解析成内联字典。例如要使⽤⼤括号引⽤变量的时候，以及 想输出⼤括号符号的时候。

**---**

**- hosts: localhost**

**tasks:**

**- shell: echo "{{inventory\_hostname}}:haha"**

冒号尾随空格时要使⽤引号包围，是因为它会被解析为"key: value"的形式。⽽且包围冒号的引号还更严格。例如下

⾯的debug模块中即使使⽤了引号也是错误的。

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo "{{inventory\_hostname}}:haha" register: hello**
  + **debug: msg="{{hello.stdout}}: heihei"**

因为它把{{...}}当成key，heihei当成value了。因此，必须将整个debug模块的参数都包围起来，显式指定这⼀段是 模块的参数。但这样会和原来的双引号冲突，因此使⽤单引号。

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo "{{inventory\_hostname}}:haha" register: hello**
  + **debug: 'msg="{{hello.stdout}}: heihei"'**

**但是，如果将shell模块中的冒号后也尾随上空格，即写成echo "{{inventory\_hostname}}: haha" ，那么shell模块也会**

报错。因此也要使⽤多个引号，正确的如下：

**---**

* **hosts: localhost**

**tasks:**

* + **shell: 'echo "{{inventory\_hostname}}: haha"' register: hello**
  + **debug: 'msg="{{hello.stdout}}: heihei"'**

# playbook应⽤和⽰例

playbook是ansible实现批量⾃动化最重要的⼿段。在其中可以使⽤变量、引⽤、循环等功能，相⽐ad- hoc⽽⾔， 其功能要强⼤的多。

## yaml简单⽰例

ansible的playbook采⽤yaml语法。以下是⼀个yaml格式的⽂件：

**---**

**# Members in Bob's family name: Bob**

**age: 30 gender: Male wife:**

**name: Alice age: 27 gender: Female**

**children:**

* **name: Jim age: 6 gender: Male**
* **name: Lucy age: 3 gender: Female**

## ansible-playbook命令说明及playbook书写简单⽰例

以下是⼀个简单的playbook⽰例。该⽰例执⾏两个任务，第⼀个任务是执⾏⼀个/bin/date命令，第⼆个任务是复 制/etc/f stab⽂件到⽬标主机上的/tmp下，它们分别使⽤了ansible的command模块和copy模块。

**cat /tmp/test.yaml**

**---**

* **hosts: centos7 tasks:**
  + **name: execute date cmd command: /bin/date**
  + **name: copy fstab to /tmp copy: src=/etc/fstab dest=/tmp**

书写好playbook后，使⽤ansible- playbook命令来执⾏。ansible- playbook命令的选项和ansible命令选项绝⼤部分都相同。但也有其特有的选项。以下是截取出来的帮助信息。

**ansible-playbook --help**

**Usage: ansible-playbook playbook.yml**

**Options:**

**-e EXTRA\_VARS,--extra-vars=EXTRA\_VARS # 设置额外的变量，格式为key/value。-e "key=KEY"，**

**# 如果是⽂件⽅式传⼊变量，则-e "@param\_file"**

**--flush-cache # 清空收集到的fact信息缓存**

**--force-handlers # 即使task执⾏失败，也强制执⾏handlers**

**--list-tags # 列出所有可获取到的tags**

**--list-tasks # 列出所有将要被执⾏的tasks**

**-t TAGS,--tags=TAGS # 以tag的⽅式显式匹配要执⾏哪些tag中的任务**

**--skip-tags=SKIP\_TAGS # 以tag的⽅式忽略某些要执⾏的任务。被此处匹配的tag中的任务都不会执⾏**

**--start-at-task=START\_AT\_TASK # 从此task开始执⾏playbook**

**--step # one-step-at-a-time:在每⼀个任务执⾏前都进⾏交互式确认**

**--syntax-check # 检查playbook语法**

现在执⾏上⾯的test.yaml。

**ansible-playbook /tmp/test.yaml**

**PLAY [centos7] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.64]**

**ok: [192.168.100.65]**

**ok: [192.168.100.63]**

**TASK [execute date cmd] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.63]**

**changed: [192.168.100.64]**

**changed: [192.168.100.65]**

**TASK [copy fstab to /tmp] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.64]**

**changed: [192.168.100.63]**

**changed: [192.168.100.65]**

**PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 192.168.100.63 : ok=3 changed=2 unreachable=0 failed=0 192.168.100.64 : ok=3 changed=2 unreachable=0 failed=0 192.168.100.65 : ok=3 changed=2 unreachable=0 failed=0**

从以上结果中，可以看出：(1)默认情况下，ansible- playbook和ansible是⼀样的，都是同步阻塞模式，需要先在所有主机上执⾏完⼀个任务，才会继续下⼀个任务；(2)在执⾏前会⾃动收集f act信息；(3)从显⽰结果中可以判断出任务是否真的执⾏了，抑或者是因为幂等性⽽没有执⾏。(4)每⼀个play都包含数个task，且都有响应信息play recap。

## playbook的内容

### hosts和remoter\_user

对于playbook中的每⼀个play，使⽤hosts选项可以定义要执⾏这些任务的主机或主机组，还可以使⽤

remote\_user指定在远程主机上执⾏任务的⽤户，实际上remote\_user是ssh连接到被控主机上的⽤户，⾃然⽽然 执⾏命令的⾝份也将是此⽤户。

例如：

**---**

**- hosts: centos6,centos7,192.168.100.59**

**remote\_user: root tasks: XXXXXX**

虽然在hosts处可以使⽤","分隔主机或主机组，但官⽅⼿册上并没有介绍该⽅法。除此之外，有以下⼏种指定主机 和主机组的⽅式：

**all 或\* ：表⽰inventory中的所有主机。**

: ：取并集。例如"host1:host2:group1"表⽰2台主机加⼀个主机组。

:& ：取交集。例如"group1:&group2"表⽰两个主机组中都有的主机。

:! ：排除。例如"group1:!host1"表⽰group1中排除host1主机的剩余主机。

通配符 ：例如"web\*.baidu.com"。

数字范围 ：例如"web[0- 5].baidu.com"。

字母范围 ：例如"web[a- d].baidu.com"。

正则表达式 ：以"~"开头。例如"~web\d\.baidu\.com"。

此外，在ansible命令⾏或ansible- playbook命令⾏中，可以使⽤"- l"选项来限定执⾏任务的主机。例如：

**ansile centos -l host[1:5] -m ping**

表⽰centos主机组中只有host1到host5才执⾏ping模块。

还可以在某个task上单独定义执⾏该task的⾝份，这将覆盖全局的定义。

**---**

**- hosts: centos6,centos7,192.168.100.59**

**remote\_user: root tasks:**

* **name: run a command shell: /bin/date**
* **name: copy a file to /tmp copy: src=/etc/fstab dest=/tmp remote\_user: myuser**

也⽀持权限升级的⽅式。

**---**

**- hosts: centos6,centos7,192.168.100.59**

**remote\_user: yourname tasks:**

* **name: run a command shell: /bin/date**
* **name: copy a file to /tmp copy: src=/etc/fstab dest=/tmp become: yes**

**become\_method: sudo**

**become\_user: root # 此项默认值就是为root，所以可省**

从上⾯的⽰例可以看出remote\_user实际上并不是执⾏任务的绝对⾝份，它只是ssh连接过去的⾝份，只不过没有 指定become的时候，它正好就⽤此⾝份来运⾏任务。

### task list

1. **特性**

每个play都包含⼀个hosts和⼀个tasks，hosts定义的是inventory中待控制的主机，tasks下定义的是⼀系列task 任务列表，⽐如调⽤各个模块。这些task按顺序⼀次执⾏⼀个，直到所有被筛选出来的主机都执⾏了这个task之后 才会移动到下⼀个task上进⾏同样的操作。

需要注意的是，虽然只有被筛选出来的主机会执⾏对应的task，但是所有主机(此处的所有主机表⽰的是，hosts选 项所指定的那些主机)都会收到相同的task指令，所有主机收到指令后，ansible主控端会筛选某些主机，并通过ssh 在远程执⾏任务。也就是说，如果查看ansible-playbook -vvvv 的信息，将会发现临时任务⽂件会通过sf tp发送到所有的被控主机上，但是只有⼀部分被筛选(如果进⾏了筛选)的主机才会ssh过去并远程执⾏命令。

当某⼀台被控主机执⾏某个任务出错或失败时，它将会被移除出任务轮询列表。也就是说，对于某主机来说，某任务执⾏失败，后续的所有任务都不会再去执⾏。当然，这不会影响其他的主机执⾏任务(除⾮主机的任务之间有依赖关系)。

最重要的是，ansible中的task是幂等性的，多次执⾏不会影响那些成功执⾏过的任务。另外幂等性还表现在执⾏失 败后如果修正了playbook再次执⾏，将不会影响那些原本已经执⾏成功的任务，即使是不同主机也不会影响。仅这

⽅⾯⽽⾔，ansible对于排错来说是极其友好的。当然，某些特殊的模块或者特殊定义的task并不⼀定总是幂等的， 例如最简单的，执⾏⼀个command或者shell模块的命令，它会重复执⾏。但也有办法使其变得幂等，以command 和shell模块为例，它们有两个选项：creates和removes，它们分别表⽰被控主机上指定的⽂件存在(不存在)时就不 执⾏命令。

1. **定义task的细节**
   1. **可以为每个task加上name项，也可以多个task依赖于⼀个name。**

例如下⾯的两个例⼦。从两个⽰例中可以看出，两个task其实都是属于⼀个name的，第⼆个task⽆需再使⽤name

命名。

⽰例⼀：

**tasks:**

* **name: do something to initialize mariadb**

**file: path=/mydata/data state=directory owner=mysql group=mysql mode=0755**

* **shell: /usr/bin/mysql\_install\_db --datadir=/mydata/data --user=mysql creates=/mydata/data/ibdata1**

⽰例⼆：

**tasks:**

* **name: echo var passed by nginx shell: echo "{{ hi\_var }}" register: var\_result**
* **debug: msg="{{ var\_result.stdout }}"**

实际上，name只是⼀种描述性语句，它可以定义在任何地⽅。例如，定义在play的顶端。

**---**

**- name: start a play hosts: localhost tasks:**

* 1. **既然是task，那么必然会有其要执⾏的⼀个或多个任务，其本质是加载并执⾏ansible对应的模块。在playbook中，每调⽤的⼀个模块都称为⼀个action。**

例如，定义⼀个确保服务是启动状态的task，有以下三种⽅法传递模块参数：

**tasks:**

**- name: be sure the sshd is running**

**service: name=sshd state=started # ⽅法⼀： 定义为key=value，直接传递参数给模块**

**service:**

**name: sshd state: started**

**# ⽅法⼆： 定义为key: value⽅式**

**service:**

**args:**

**name: sshd state: started**

**# ⽅法三： 使⽤args内置关键字，然后定义为key: value⽅式**

但要注意，ping模块、command和shell模块是不需要 key=value 格式的。对于ping命令，可以直接省略key=value 。对于command和shell，只需要给定命令路径和要接上去的选项或参数即可，且⽆法使⽤上⾯的⽅法⼆。例如下⾯ 定义的是⼀个ntpdate命令，只需给定它的参数即可。

**tasks:**

* **name: execute command ntpdate**

**shell: /usr/sbin/ntpdate ntp1.aliyun.com**

* **name: ping host ping:**

对于command或shell模块来说，有时候要考虑命令的返回状态码。如果要忽略⾮0状态码继续执⾏任务，可以使⽤ 以下两种⽅式：

**tasks:**

**- name: ignore non\_zero return code**

**shell: /usr/sbin/ntpdate ntp1.aliyun.com || /bin/true**

或者

**tasks:**

**- name: another way to ignore the non\_zero return code shell: /usr/sbin/ntpdate ntp1.aliyun.com ignore\_errors: true**

* 1. **如果action的key=value太多，导致内容太长，可以在上⼀⾏的缩进级别基础上继续缩进表⽰续⾏。 例如，下⾯的owner⽐src多缩进了4个空格。**

**tasks:**

**- name: Copy ansible inventory file to client copy: src=/etc/fstab dest=/tmp**

**owner=root group=root mode=0644**

* 1. **在action的value部分，可以引⽤已经定义的变量，可以是已定义好的⾃定义的变量，也可以是内置变量。 变量相关内容见后⽂。**
  2. **使⽤include指令，可以将其他的playbook⽂件包含到此playbook⽂件中。include的⽅式见下⽂。**

### notify和handler

ansible中⼏乎所有的模块都具有幂等性，这意味着被控主机的状态是否发⽣改变是能被捕捉的，即每个任务的changed=true或changed=f alse。ansible在捕捉到changed=true时，可以触发notif y组件(如果定义了该组件)。

notif y是⼀个组件，并⾮⼀个模块，它可以直接定义action，其主要⽬的是调⽤handler。例如：

**tasks:**

* **name: copy template file to remote host**

**template: src=/etc/ansible/nginx.conf.j2 dest=/etc/nginx/nginx.conf notify:**

* + **restart nginx**
  + **test web page**

**copy: src=nginx/index.html.j2 dest=/usr/share/nginx/html/index.html notify:**

* + **restart nginx**

这表⽰当执⾏template模块的任务时，如果捕捉到changed=true，那么就会触发notif y，如果分发的index.html改变了，那么也重启nginx(当然这是没必要的)。notif y下定义了两个待调⽤的handler。handler主要⽤于重启服务或者触发系统重启，除此之外很少使⽤handler。以下是这两个handler的内容：

**handlers:**

* **name: restart nginx**

**service: name=nginx state=restarted**

* **name: test web page**

**shell: curl -I** [**http://192.168.100.10/index.html**](http://192.168.100.10/index.html) **| grep 200 || /bin/false**

handler的定义和tasks的定义完全⼀样，唯⼀需要限定的是handler中task的name必须和notif y中定义的名称相同。

注意，notif y是在执⾏完⼀个play中所有task后被触发的，在⼀个play中也只会被触发⼀次。意味着如果⼀个play中有多个task出现了changed=true，它也只会触发⼀次。例如上⾯的⽰例中，向nginx复制配置⽂件和复制

index.html时如果都发⽣了改变，都会触发重启apache操作。但是只会在执⾏完play后重启⼀次，以避免多余的重

启。

### 标签tag

可以为playbook中的每个任务都打上标签，标签的主要作⽤是可以在ansible- playbook中设置只执⾏哪些被打上tag的任务或忽略被打上tag的任务。

**tasks:**

* **name: make sure apache is running service: name=httpd state=started tags: apache**
* **name: make sure mysql is running service: name=mysqld state=started tags: mysql**

以下是ansible- playbook命令关于tag的选项。

**--list-tags**

**# list all available tags**

**-t TAGS, --tags=TAGS # only run plays and tasks tagged with these values**

**--skip-tags=SKIP\_TAGS # only run plays and tasks whose tags do not match these values**

## include和roles

如果将所有的play都写在⼀个playbook中，很容易导致这个playbook⽂件变得臃肿庞⼤，且不易读。因此，可以将 多个不同任务分别写在不同的playbook中，然后使⽤include将其包含进去即可。⽽role则是整合playbook的⽅

式。⽆论是include还是role，其⽬的都是分割⼤playbook以及复⽤某些细化的play甚⾄是task。

### include

可以将task列表和handlers独⽴写在其他的⽂件中，然后在某个playbook⽂件中使⽤include来包含它们。除此之 外，还可以写独⽴的playbook⽂件，使⽤include来包含这个⽂件。

也即是说，include可以导⼊两种⽂件：导⼊task、导⼊playbook。

1. **⼀种是任务列表式的⽂件(没有tasks或handlers指令)，它只能在tasks或handlers指令的⼦选项处使⽤include包 含。这种⽅式可以传递变量到被包含的⽂件中。**

假设某个task列表⽂件/yaml/a.yaml内容如下：

**---**

**- name: execute ntpdate**

**shell: /usr/sbin/ntpdate ntp1.aliyun.com**

在同⽬录/yaml下有⼀个名为test.yaml的playbook(除了role，playbook中所有相对路径都是基于playbook的)，在此playbook中使⽤include来包含它，如果使⽤相对路径将会包含同⽬录下的⽂件。

**---**

**- hosts: centos7**

**tasks:**

**- include: a.yaml**

可以在include的时候传递变量给对应的⽂件，这样在被包含的⽂件中就可以引⽤该变量的值。

**---**

**- hosts: centos7 tasks:**

**- include: a.yaml sayhi="hello world"**

或者

**---**

**- hosts: centos7 tasks:**

**- include: a.yaml vars:**

**sayhi: "hello world"**

然后可以在被包含的⽂件a.yaml中使⽤该变量。例如：

**---**

* **name: execute ntpdate**

**shell: /usr/sbin/ntpdate ntp1.aliyun.com**

* **name: say hi to world debug: msg="{{ sayhi }}"**

执⾏该test.yaml，将会输出对应的变量值。

**ansible-playbook /yaml/test.yaml**

**PLAY [centos7] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.64]**

**ok: [192.168.100.63]**

**ok: [192.168.100.65]**

**TASK [execute ntpdate] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.64]**

**changed: [192.168.100.65]**

**changed: [192.168.100.63]**

**TASK [sayhi to world] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.63] => {**

**"changed": false, "msg": "hello world"**

**}**

**ok: [192.168.100.65] => {**

**"changed": false, "msg": "hello world"**

**}**

**ok: [192.168.100.64] => {**

**"changed": false, "msg": "hello world"**

**}**

**PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 192.168.100.63 : ok=3 changed=1 unreachable=0 failed=0 192.168.100.64 : ok=3 changed=1 unreachable=0 failed=0 192.168.100.65 : ok=3 changed=1 unreachable=0 failed=0**

1. **另⼀种是include整个playbook⽂件，即include的动作是加载⼀个或多个play，所以写在顶级列表的层次。**

* **name: this is a play at the top level of a file hosts: all**

**remote\_user: root tasks:**

**- name: say hi tags: foo**

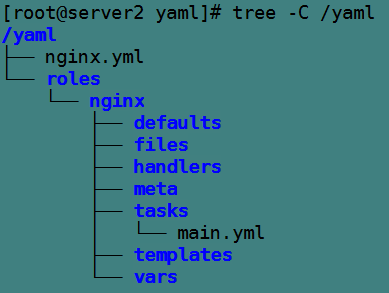
**shell: echo "hi..."**

* **include: load\_balancers.yml sayhi="hello world"**
* **include: webservers.yml**
* **include: dbservers.yml**

**any other operations**

需要说明的是，在ansible 2.4版本中，添加了includes和imports两种导⼊的⽅式，它们对静态和动态导⼊⽀持的更细化，⽽ansible 2.3及以前的include语句已经废弃，但仍可⽤。相关官⽅⼿册地址。

### roles

roles意为⾓⾊，主要⽤于封装playbook实现复⽤性。在ansible中，roles通过⽂件的组织结构来展现。 对于⼀个role，它的⽂件组织结构如下图所⽰。

⾸先需要有⼀个roles⽬录。同时，在roles⽬录所在⽬录中，还要有⼀个playbook⽂件，此处为nginx.yml，nginx.yml⽂件是ansible- playbook需要执⾏的⽂件，在此⽂件中定义了⾓⾊，当执⾏到⾓⾊时，将会到roles中对应的⾓⾊⽬录中寻找相关⽂件。

roles⽬录中的⼦⽬录是即是各个role。例如，此处只有⼀个名为nginx的role，在role⽬录中，有⼏个固定名称的⽬ 录(如果没有则忽略)。在这些⽬录中，还要有⼀些固定名称的⽂件，除了固定名称的⽂件，其他的⽂件可以随意命名。以下是各个⽬录的含义：

tasks⽬录：存放task列表。若role要⽣效，此⽬录必须要有⼀个主task⽂件main.yml，在main.yml中可以使⽤include包含同⽬录(即tasks)中的其他⽂件。

handlers⽬录：存放handlers的⽬录，若要⽣效，则⽂件必须名为main.yml⽂件。

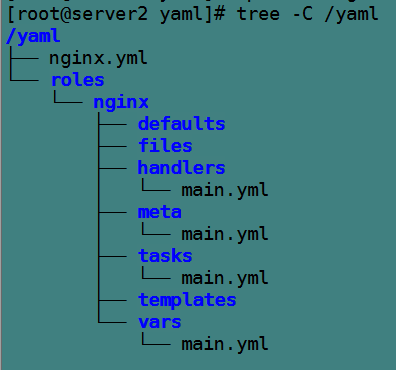
f iles⽬录：在task中执⾏copy或script模块时，如果使⽤的是相对路径，则会到此⽬录中寻找对应的⽂件。

templates⽬录：在task中执⾏template模块时，如果使⽤的是相对路径，则会到此⽬录中寻找对应的模块⽂ 件。

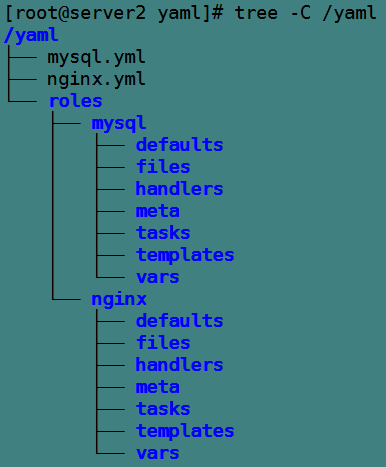
vars⽬录：定义专属于该role的变量，如果要有var⽂件，则必须为main.yml⽂件。

def aults⽬录：定义⾓⾊默认变量，⾓⾊默认变量的优先级最低，会被任意其他层次的同名变量覆盖。如果要有var⽂件，则必须为main.yml⽂件。

meta⽬录：⽤于定义⾓⾊依赖(dependencies)，如果要有⾓⾊依赖关系，则⽂件必须为main.yml。 所以，相对完整的role的⽂件组织结构如下图。



如果是多个role，则在roles同级⽬录下定义多个⼊站(作⽤类似于C语⾔的main函数)⽂件(如上⾯的nginx.yml)，并 在roles⽬录下创建对应的role⽬录即可。



当然，如果不是使⽤相对路径，那么role的⽂件结构就⽆所谓了，但是roles功能开发出来，就是为了解决⽂件混乱 和playbook臃肿问题的。所以如果可以，尽量使⽤推荐的role⽂件结构。

另外，如果role中出现的task、var、handler等和单独定义的对象同名冲突了，则优先执⾏role中的内容。 以下是nginx role的⼊站⽂件nginx.yml的内容。

**---**

**- hosts: centos7 roles:**

**- nginx**

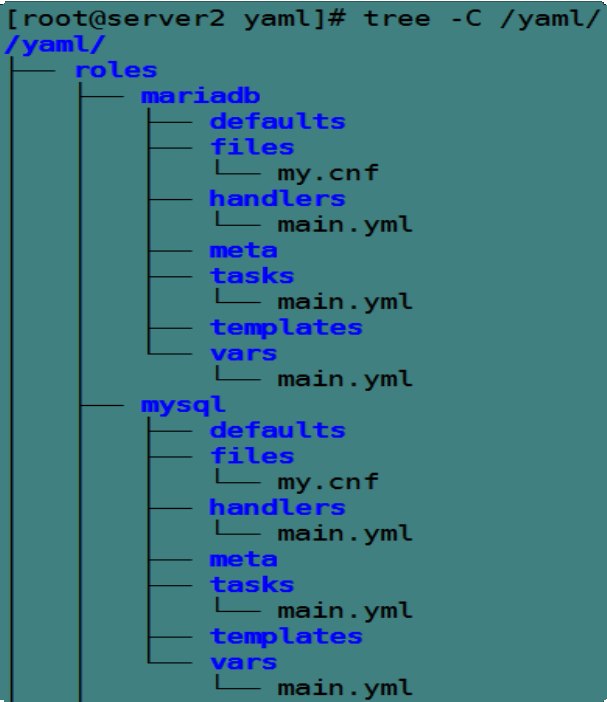
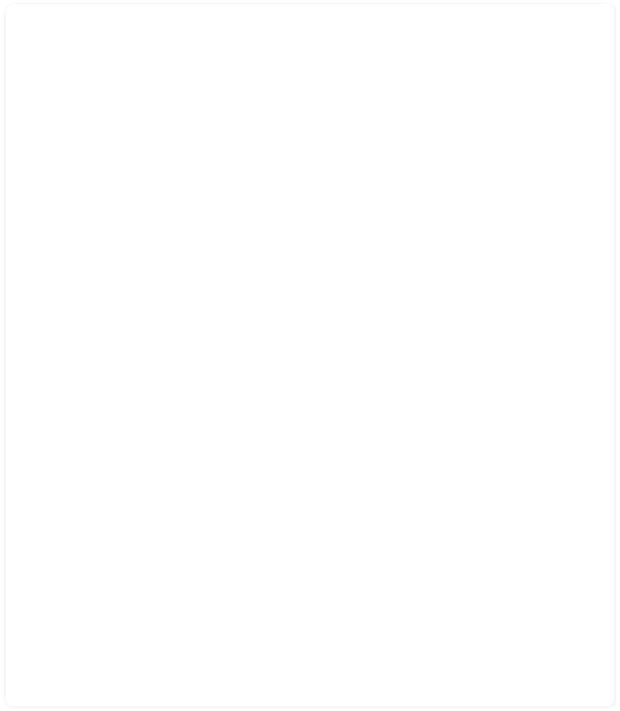
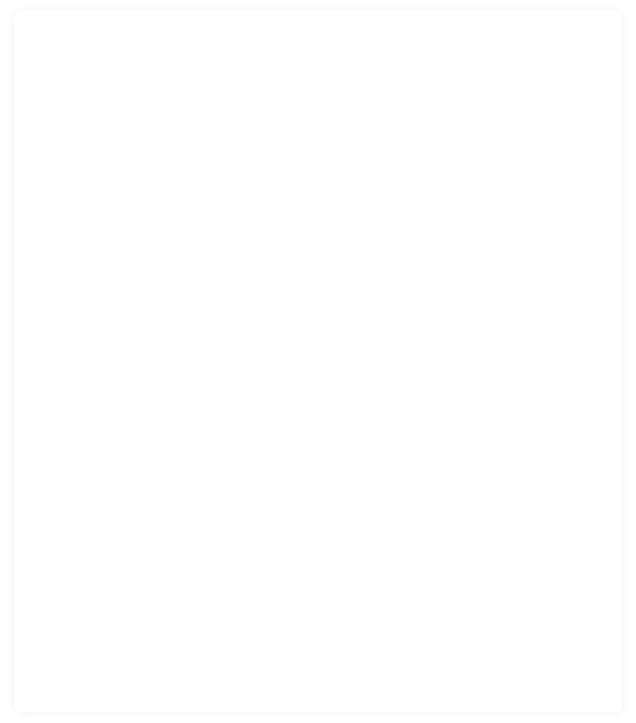
更多更详细的role⽤法以及组织结构，见下⾯的⽰例。

## roles⽰例：批量⾃动化安装

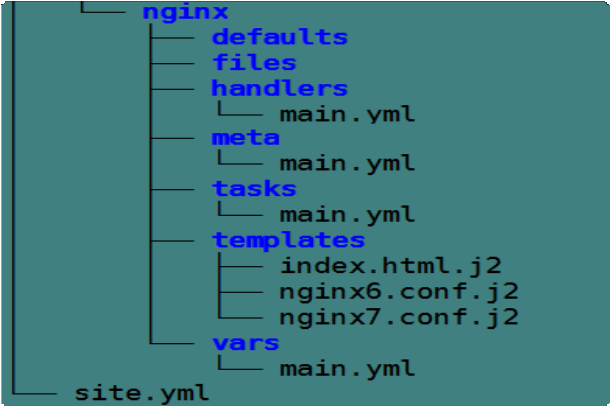
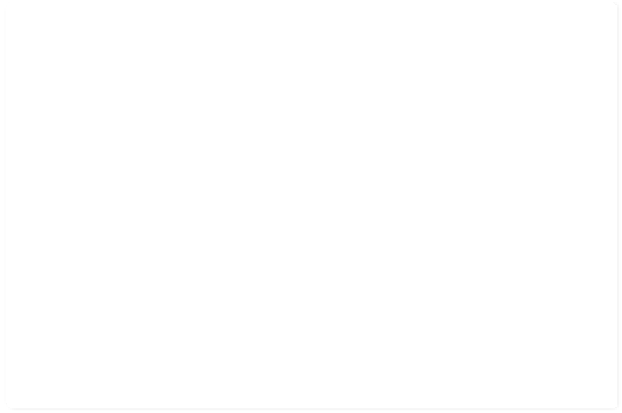
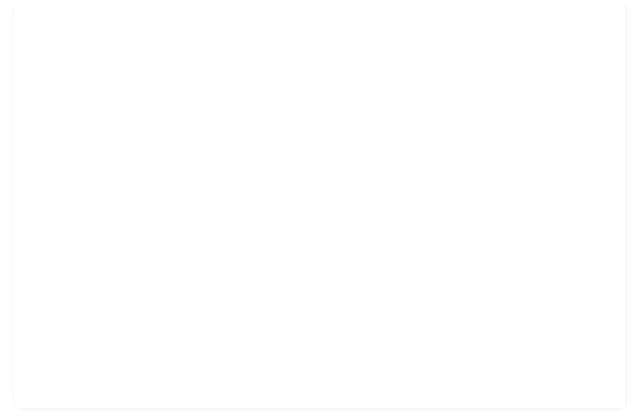
下⾯演⽰的是使⽤role批量⾃动安装nginx和mysql(CentOS 6)或maridb(CentOS 7)的⽰例。由于被控节点有CentOS 6和CentOS 7两种发⾏版的操作系统，因此除了要挑选对应的数据库，还要让nginx的配置⽂件适应各操作系统，因为nginx在这两个版本的系统上配置内容有所不同。在此，nginx、mysql和mariadb是3个role，且让nginx role依赖于mysql或mariadb role。

下⾯的例⼦中，有些地⽅是不太合理或多余的⾏为，但是作为学习⽰例，可以很好的理解roles之间的组织结构和相 关的操作。

⾸先是⽂件的结构。







其中site.yml是⼊站⽂件，⽤于调⽤nginx、mysql和mariadb这3个role，这个⽂件中的内容如下。它有3个作⽤：(1) 在调⽤roles之前，先根据发⾏版配置好yum源(pre\_tasks);(2)调⽤nginx role，此处没有调⽤mysql和mariadb这两个role，因为在nginx role的meta/main.yml⽂件中定义了nginx role依赖于这两个role，所以此处可以不⽤定义;(3) 在执⾏完nginx role之后，输出⼀个提⽰信息(post\_tasks)。

**cat /yaml/site.yml**

**---**

**- hosts: centos remote\_user: root**

**# 根据发⾏版配置好yum源，使⽤when进⾏条件判断**

**pre\_tasks:**

**- name: config the yum repo for centos 7 yum\_repository:**

**name: epel description: epel**

**baseurl:** [**http://mirrors.aliyun.com/epel/7/$basearch/**](http://mirrors.aliyun.com/epel/7/%24basearch/) **gpgcheck: no**

**when: ansible\_distribution\_major\_version == "7"**

**- name: config the yum repo for centos 6 yum\_repository:**

**name: epel description: epel**

**baseurl:** [**http://mirrors.aliyun.com/epel/6/$basearch/**](http://mirrors.aliyun.com/epel/6/%24basearch/) **gpgcheck: no**

**when: ansible\_distribution\_major\_version == "6"**

**roles:**

**- nginx**

**# 输出over消息**

**post\_tasks:**

* **shell: echo 'deploy nginx/mysql over' register: ok\_var**
* **debug: msg='{{ ok\_var.stdout }}'**

以下是nginx role中的各⽂件内容。其中template复制的源⽂件都是从centos 6 nginx和centos 7 nginx上提取的， 只不过是重新命名了⽽已。

**/yaml/roles/nginx/tasks/main.yml**

**---**

* **name: make sure nginx state is installed yum: name=nginx state=installed**
* **name: template nginx.conf**

**# 基于变量赋值配置⽂件模板，检查配置⽂件语法，并在必要的时候触发handler template: src=nginx{{ ansible\_distribution\_major\_version }}.conf.j2**

**dest=/etc/nginx/nginx.conf validate="/usr/sbin/nginx -t -c %s"**

**notify:**

* + **restart nginx**

**# 基于jinja2渲染模板⽂件，且改变时也触发重启操作**

* **name: copy index.html**

**template: src=index.html.j2 dest=/usr/share/nginx/html/index.html notify:**

* + **restart nginx**
* **name: make sure nginx service is running service: name=nginx state=started**

**# 引⽤变量 nginx\_port，在vars/main.yml中定义了**

* **name: make sure port is open**

**wait\_for: port="{{ nginx\_port }}"**

**/yaml/roles/nginx/handlers/main.yml**

**---**

* **name: restart nginx**

**service: name=nginx state=restarted**

**/yaml/roles/nginx/vars/main.yml nginx\_port: 80**

**# 定义nginx依赖于MySQL或mariadb，具体依赖于哪个，是通过条件进⾏判断的，centos 6表⽰依赖于mysql，centos 7表⽰依赖于mariadb # 同时传递了两个值给变量hi\_var，由于是在依赖的时候传递的，所以这两个变量可直接在依赖的role(mysql role或mariadb role)的play book中引⽤**

**/yaml/roles/nginx/meta/main.yml**

**---**

**dependencies:**

* + **{ role: mysql,hi\_var: "hello mysql",when: "ansible\_distribution\_major\_version == '6'" }**
  + **{ role: mariadb,hi\_var: "hello mariadb",when: "ansible\_distribution\_major\_version == '7'" }**

上⾯拷贝了index.html.j2，其内容为：

**shell> cat /yaml/roles/nginx/templates/index.html.j2**

**<h1>hello from {{ ansible\_default\_ipv4.address }}<h1>**

在template执⾏时，它会使⽤jinja2引擎对⽂件中的变量进⾏替换，使得在拷贝到不同主机时，该index.html的内容 是基于远程主机ip的。此处使⽤的变量是收集到的f acts中的变量"ansible\_def ault\_ipv4.address"。

以下是mysql role中各⽂件的内容。注意，MySQL的my.cnf和mariadb的my.cnf默认情况下并不⼀样(mariadb的my.cnf默认多了⼀项配置"!includedir /etc/my.cnf .d"，MySQL需要取消该项)，所以需要分别提供。

**/yaml/roles/mysql/tasks/main.yml**

**---**

* **name: make sure mysql is installed yum: name=mysql-server state=installed**

**# 特别需要注意下⾯的初始化命令，由于执⾏的是shell模块，所以要考虑其幂等性，显然初始化动作是⼀定要实现幂等性的**

* **name: do something to initialize mysql**

**file: path=/mydata/data state=directory owner=mysql group=mysql mode=0755**

* **shell: /usr/bin/mysql\_install\_db --datadir=/mydata/data --user=mysql creates=/mydata/data/ibdata1**
* **name: copy my.cnf**

**copy: src=my.cnf dest=/etc/my.cnf notify:**

**- restart mysql**

* **name: make sure mysql is running service: name=mysqld state=started**
* **name: make sure mysql port is open wait\_for:**

**port: "{{ mysql\_port }}" timeout: 10**

**# 这⾥输出了nginx/meta/main.yml中传递的变量**

* **name: echo var passed by nginx**

**shell: echo "{{ hi\_var }}" register: var\_result**

* **debug: msg="{{ var\_result.stdout }}"**

**/yaml/roles/mysql/handlers/main.yml**

**---**

**- name: restart mysql**

**service: name=mysqld state=restarted**

**/yaml/roles/mysql/vars/main.yml**

**---**

**mysql\_port: 3306**

以下是mariadb role中各⽂件的内容，和mysql⼤体上是⼀致的。注意，MySQL的my.cnf和mariadb的my.cnf默认情况下并不⼀样，所以需要分别提供。

**/yaml/roles/mariadb/tasks/main.yml**

**---**

* **name: make sure mariadb is installed yum: name=mariadb-server state=installed**
* **name: do something to initialize mariadb**

**file: path=/mydata/data state=directory owner=mysql group=mysql mode=0755**

* **shell: /usr/bin/mysql\_install\_db --datadir=/mydata/data --user=mysql creates=/mydata/data/ibdata1**
* **name: copy my.cnf**

**copy: src=my.cnf dest=/etc/my.cnf notify:**

**- restart mariadb**

* **name: make sure mariadb is running service: name=mariadb state=started**
* **name: make sure mariadb port is open wait\_for:**

**port: "{{ mariadb\_port }}" timeout: 10**

* **name: echo var passed by nginx shell: echo "{{ hi\_var }}" register: var\_result**
* **debug: msg="{{ var\_result.stdout }}"**

**/yaml/roles/mariadb/handlers/main.yml**

**---**

* **name: restart mariadb**

**service: name=mariadb state=restarted**

**/yaml/roles/mariadb/vars/main.yml**

**---**

**mariadb\_port: 3306**

以下是两台机器的测试结果，⼀台是centos 7(192.168.100.54)，⼀台是centos 6(192.168.100.70)。

**PLAY [newhosts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.70]**

**ok: [192.168.100.54]**

**TASK [config the yum repo for centos 7] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [config the yum repo for centos 6] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : make sure mysql is installed] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : do something to initialize mysql] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : copy my.cnf] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : make sure mysql is running] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : make sure mysql port is open] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**ok: [192.168.100.70]**

**TASK [mysql : echo var passed by nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [mysql : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.54]**

**ok: [192.168.100.70] => {**

**"changed": false, "msg": "hello mysql"**

**}**

**TASK [mariadb : make sure mariadb is installed] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : do something to initialize mariadb] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : copy my.cnf] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : make sure mariadb is running] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : make sure mariadb port is open] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**ok: [192.168.100.54]**

**TASK [mariadb : echo var passed by nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [mariadb : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* skipping: [192.168.100.70]**

**ok: [192.168.100.54] => {**

**"changed": false, "msg": "hello mysql"**

**}**

**TASK [nginx : make sure nginx state is installed] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [nginx : template nginx.conf] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [nginx : copy index.html] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.70]**

**changed: [192.168.100.54]**

**TASK [nginx : make sure nginx service is running] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [nginx : make sure port is open] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.70]**

**ok: [192.168.100.54]**

**RUNNING HANDLER [mysql : restart mysql] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.70]**

**RUNNING HANDLER [mariadb : restart mariadb] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.54]**

**RUNNING HANDLER [nginx : restart nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.54]**

**TASK [command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* changed: [192.168.100.54]**

**changed: [192.168.100.70]**

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.54] => {**

**"changed": false,**

**"msg": "deploy nginx/mysql over"**

**}**

**ok: [192.168.100.70] => {**

**"changed": false,**

**"msg": "deploy nginx/mysql over"**

**}**

**PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 192.168.100.54 : ok=19 changed=14 unreachable=0 failed=0 192.168.100.70 : ok=18 changed=12 unreachable=0 failed=0**

相信看过上⾯的roles组织⽰例，对roles的⽤法和playbook就有了较深的认识。其实，ansible有⼀个⽹站专门存放 了⼀⼤堆的playbook，算是playbook仓库吧。可以下载下来稍作修改就能使⽤，即使不使⽤，借鉴他们的写法也 是很值得的。地址：ansible galaxy

另外，根据不同标准组织role可能会让playbook写起来更容易，例如上⾯的⽰例中，按照发⾏版来划分role⽐上⾯ 按照安装软件类型划分可能会更简单些。当然，如果在inventory中就划分好centos 6和centos 7也是可以的。哪种更⽅便、复⽤性更好就需要⾃⾏考虑了。

# 各种变量定义⽅式和变量引⽤

## ansible facts

f acts组件是⽤来收集被管理节点信息的，使⽤setup模块可以获取这些信息。

**ansible-doc -s setup**

**- name: Gathers facts about remote hosts**

以下是某次收集的信息⽰例。由于收集的信息项⾮常多，所以截取了部分内容项。

**ansible 192.168.100.64 -m setup 192.168.100.64 | SUCCESS => {**

**"ansible\_facts": { "ansible\_all\_ipv4\_addresses": [**

**"192.168.100.64"**

**],**

**"ansible\_all\_ipv6\_addresses": [ "fe80::20c:29ff:fe03:a452"**

**],**

**"ansible\_apparmor": { "status": "disabled"**

**},**

**"ansible\_architecture": "x86\_64", "ansible\_bios\_date": "07/02/2015", "ansible\_bios\_version": "6.00", "ansible\_cmdline": {**

**"BOOT\_IMAGE": "/vmlinuz-3.10.0-327.el7.x86\_64", "LANG": "en\_US.UTF-8",**

**"biosdevname": "0", "crashkernel": "auto", "net.ifnames": "0", "quiet": true,**

**"ro": true,**

**"root": "UUID=b2a70faf-aea4-4d8e-8be8-c7109ac9c8b8"**

**},**

**........................................**

**"ansible\_default\_ipv6": {}, "ansible\_devices": {**

**"sda": {**

**"holders": [],**

**"host": "SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320**

**SCSI (rev 01)",**

**"model": "VMware Virtual S", "partitions": {**

**"sda1": {**

**"holders": [],**

**"sectors": "512000",**

**"sectorsize": 512,**

**"size": "250.00 MB",**

**"start": "2048",**

**"uuid": "367d6a77-033b-4037-bbcb-416705ead095"**

**},**

**"sda2": {**

**"holders": [], "sectors": "37332992",**

**"sectorsize": 512,**

**"size": "17.80 GB",**

**"start": "514048",**

**"uuid": "b2a70faf-aea4-4d8e-8be8-c7109ac9c8b8"**

**},**

**"sda3": {**

**"holders": [], "sectors": "4096000",**

**"sectorsize": 512,**

**"size": "1.95 GB",**

**"start": "37847040",**

**"uuid": "d505113c-daa6-4c17-8b03-b3551ced2305"**

**}**

**},**

**................................**

**"ansible\_system\_capabilities\_enforced": "True", "ansible\_system\_vendor": "VMware, Inc.", "ansible\_uptime\_seconds": 33022, "ansible\_user\_dir": "/root", "ansible\_user\_gecos": "root", "ansible\_user\_gid": 0,**

**"ansible\_user\_id": "root", "ansible\_user\_shell": "/bin/bash", "ansible\_user\_uid": 0, "ansible\_userspace\_architecture": "x86\_64", "ansible\_userspace\_bits": "64", "ansible\_virtualization\_role": "guest", "ansible\_virtualization\_type": "VMware", "module\_setup": true**

**},**

**"changed": false**

**}**

使⽤f ilter可以筛选指定的f acts信息。例如：

**ansible 192.168.100.64 -m setup -a "filter=changed" 192.168.100.64 | SUCCESS => {**

**"ansible\_facts": {}, "changed": false**

**}**

**ansible localhost -m setup -a "filter=\*ipv4" localhost | SUCCESS => {**

**"ansible\_facts": { "ansible\_default\_ipv4": {**

**"address": "192.168.100.62",**

**"alias": "eth0",**

**"broadcast": "192.168.100.255",**

**"gateway": "192.168.100.2",**

**"interface": "eth0", "macaddress": "00:0c:29:d9:0b:71", "mtu": 1500,**

**"netmask": "255.255.255.0",**

**"network": "192.168.100.0",**

**"type": "ether"**

**}**

**},**

**"changed": false**

**}**

f acts收集的信息是json格式的，其内任⼀项都可以当作变量被直接引⽤(如在playbook、jinja2模板中)引⽤。见下

⽂。

## 变量引⽤json数据的⽅式

在ansible中，任何⼀个模块都会返回json格式的数据，即使是错误信息都是json格式的。

在ansible中，json格式的数据，其内每⼀项都可以通过变量来引⽤它。当然，引⽤的前提是先将其注册为变量。 例如，下⾯的playbook是将shell模块中echo命令的结果注册为变量，并使⽤debug模块输出。

**---**

**- hosts: 192.168.100.65**

**tasks:**

* **shell: echo hello world register: say\_hi**
* **debug: var=say\_hi**

debug输出的结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"say\_hi": {**

**"changed": true,**

**"cmd": "echo hello world", "delta": "0:00:00.002086",**

**"end": "2017-09-20 21:03:40.484507",**

**"rc": 0,**

**"start": "2017-09-20 21:03:40.482421",**

**"stderr": "",**

**"stderr\_lines": [], "stdout": "hello world", "stdout\_lines": [**

**"hello world"**

**]**

**}**

**}**

可以看出，结果是⼀段json格式的数据，最顶端的key为 say\_hi ，其内是⼀⼤段的字典(即使⽤⼤括号包围的)，其中的stdout\_lines 还包含了⼀个json数组，也就是所谓的yaml列表项(即使⽤中括号包围的)。

### 引⽤json字典数据的⽅式

如果想要输出json数据的某⼀字典项，则应该使⽤"key.dict"或"key['dict']"的⽅式引⽤。例如最常见的

stdout项"hello world"是想要输出的项，以下两种⽅式都能引⽤该字典变量。

**---**

**- hosts: 192.168.100.65**

**tasks:**

* **shell: echo hello world register: say\_hi**
* **debug: var=say\_hi.stdout**
* **debug: var=sya\_hi['stdout']**

ansible- playbook的部分输出结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"say\_hi.stdout": "hello world"**

**}**

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"say\_hi['stdout']": "hello world"**

**}**

"key.dict"或"key['dict']"的⽅式都能引⽤，但在dict字符串本⾝就包含"."的时候，应该使⽤中括号的⽅式引⽤。例如：

**anykey['192.168.100.65']**

### 引⽤json数组数据的⽅式

如果想要输出json数据中的某⼀数组项(列表项)，则应该使⽤"key[N]"的⽅式引⽤数组中的第 N项，其中N是数组的index，从0开始计算。如果不使⽤index，则输出的是整个数组列表。

**例如想要输出上⾯的stdout\_lines 中的"hello world"，它是数组stdout\_lines 中第⼀项所以使⽤stdout\_lines[0] 来引**

**⽤，再加上stdout\_lines 上⾯的say\_hi ，于是引⽤⽅式如下：**

**---**

**- hosts: 192.168.100.65**

**tasks:**

* **shell: echo hello world register: say\_hi**
* **debug: var=say\_hi.stdout\_lines[0]**

**由于stdout\_lines 中仅有⼀项，所以即使不使⽤index的⽅式即say\_hi.stdout\_lines 也能得到期望的结果。输出结果如下：**

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"say\_hi.stdout\_lines[0]": "hello world"**

**}**

再看下⾯⼀段json数据。

**"ipv6": [**

**{**

**"address": "fe80::20c:29ff:fe26:1498", "prefix": "64",**

**"scope": "link"**

**}**

**]**

其中key=ipv6，其内有且仅有是⼀个列表项，但该列表内包含了数个字典项。要引⽤列表内的字典，例如上⾯的

address项。应该如下引⽤：

**ipv6[0].address**

### 引⽤facts数据

既然已经了解了json数据中的字典和列表列表项的引⽤⽅式，显然f acts中的⼀⼤堆数据就能引⽤并派上⽤场了。例如以下是⼀段f acts数据。

**shell> ansible localhost -m setup -a "filter=\*eth\*" localhost | SUCCESS => {**

**"ansible\_facts": { "ansible\_eth0": {**

**"active": true, "device": "eth0", "features": {**

**"busy\_poll": "off [fixed]",**

**"fcoe\_mtu": "off [fixed]", "generic\_receive\_offload": "on",**

**.........................**

**},**

**"ipv4": {**

**"address": "192.168.100.62",**

**"broadcast": "192.168.100.255",**

**"netmask": "255.255.255.0",**

**"network": "192.168.100.0"**

**},**

**"macaddress": "00:0c:29:d9:0b:71", "module": "e1000",**

**............................**

**}**

**},**

**"changed": false**

**}**

**显然，f acts数据的顶级key为 ansible\_facts ，在引⽤时应该将其包含在变量表达式中。但⾃动收集的f acts⽐较特殊，它以ansible\_facts 作为key，ansible每次收集后会⾃动将其注册为变量，所以f acts中的数据都可以直接通过变量引⽤，甚⾄连顶级key ansible\_facts 都要省略。**

例如引⽤上⾯的ipv4的地址address项。

**ansible\_eth0.ipv4.address**

⽽不能写成：

**ansible\_facts.ansible\_eth0.ipv4.address**

但其他任意时候，都应该带上所有的key。

## 设置本地facts

在ansible收集f acts时，还会⾃动收集/etc/ansible/f acts.d/\*.f act⽂件内的数据到f acts中，且以 ansible\_local 做为key。⽬前f act⽀持两种类型的⽂件：ini和json。当然，如果f act⽂件的json或ini格式写错了导致⽆法解析，那么肯定也⽆法收集。

例如，在/etc/ansible/f acts.d⽬录下存在⼀个my.f act的⽂件，其内数据如下：

**shell> cat /etc/ansible/facts.d/my.fact**

**{**

**"family": {**

**"father": {**

**"name": "Zhangsan",**

**"age": "39"**

**},**

**"mother": {**

**"name": "Lisi",**

**"age": "35"**

**}**

**}**

**}**

ansible收集f acts后的本地f acts数据如下：

**shell> ansible localhost -m setup -a "filter=ansible\_local" localhost | SUCCESS => {**

**"ansible\_facts": { "ansible\_local": {**

**"my": {**

**"family": {**

**"father": {**

**"age": "39",**

**"name": "Zhangsan"**

**},**

**"mother": {**

**"age": "35",**

**"name": "Lisi"**

**}**

**}**

**}**

**}**

**},**

**"changed": false**

**}**

可见，如果想要引⽤本地⽂件中的某个key，除了带上ansible\_local外，还必须得带上f act⽂件的⽂件名。例如， 引⽤f ather的name。

**ansible\_local.my.family.father.name**

## 输出和引⽤变量

上⽂已经展⽰了⼀种变量的引⽤⽅式：使⽤debug的var参数。debug的另⼀个参数msg也能输出变量，且msg可以 输出⾃定义信息，⽽var参数只能输出变量。

另外，msg和var引⽤参数的⽅式有所不同。例如：

**---**

**- hosts: 192.168.100.65**

**tasks:**

* **debug: 'msg="ipv4 address: {{ansible\_eth0.ipv4.address}}"'**
* **debug: var=ansible\_eth0.ipv4.address**

msg引⽤变量需要加上双⼤括号包围，既然加了⼤括号，为了防⽌被解析为内联字典，还得加引号包围。这⾥使⽤ 了两段引号，因为其内还包括了⼀个": "，加引号可以防⽌它被解析为"key: "的格式。⽽var参数引⽤变量则直接指定变量名。

这就像bash中引⽤变量的⽅式是⼀样的，有些时候需要加上$ ，有些时候不能加$ 。也就是说，当引⽤的是变量的值，就需要加双⼤括号，就像加$ ⼀样，⽽引⽤变量本⾝，则不能加双⼤括号。其实双⼤括号是jinja2中的分隔符。

执⾏的部分结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"msg": "ipv4 address: 192.168.100.65"**

**}**

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"ansible\_eth0.ipv4.address": "192.168.100.65"**

**}**

⼏乎所有地⽅都可以引⽤变量，例如循环、when语句、信息输出语句、template⽂件等等。只不过有些地⽅不能 使⽤双⼤括号，有些地⽅需要使⽤。

## 注册和定义变量的各种⽅式

ansible中定义变量的⽅式有很多种，⼤致有：(1)将模块的执⾏结果注册为变量；(2)直接定义字典类型的变量； (3)role中⽂件内定义变量；(4)命令⾏传递变量；(5)借助with\_items迭代将多个task的结果赋值给⼀个变量； (6)inventory中的主机或主机组变量；(7)内置变量。

### register注册变量

使⽤register选项，可以将当前task的输出结果赋值给⼀个变量。例如，下⾯的⽰例中将echo的结果"haha"赋值给say\_hi变量。注意，模块的输出结果是json格式的，所以，引⽤变量时要指定引⽤的对象。

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo haha register: say\_hi**
  + **debug: var=say\_hi.stdout**

### set\_fact定义变量

set\_f act和register的功能很相似，也是将值赋值给变量。它更像shell中变量的赋值⽅式，可以将某个变量的值赋值给另⼀个变量，也可以将字符串赋值给变量。

例如：

**---**

**- hosts: 192.168.100.65**

**tasks:**

* **shell: echo haha register: say\_hi**
* **set\_fact: var1="{{say\_hi.stdout}}"**
* **set\_fact: var2="your name is"**
* **debug: msg="{{var2}} {{var1}}"**

### vars定义变量

可以在play或task层次使⽤vars定义字典型变量。如果同名，则task层次的变量覆盖play层次的变量。 例如：

**---**

**- hosts: localhost**

**vars:**

**var1: value1 var2: value2**

**tasks:**

**- debug: msg="{{var1}} {{var2}}" vars:**

**var2: value2.2**

输出结果为：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [localhost] => {**

**"msg": "value1 value2.2"**

**}**

### vars\_files定义变量

和vars⼀样，只不过它是将变量以字典格式定义在独⽴的⽂件中，且 vars\_files 不能定义在task层次，只能定义在play层次。

**---**

* **hosts: localhost**

**vars\_files:**

* + **/tmp/var\_file1.yml**
  + **var\_file2.yml tasks:**

**- debug: msg="{{var1}} {{var2}}"**

上⾯var\_f ile2.yml使⽤的是相对路径，基于playbook所在的路径。例如该playbook为/tmp/x.yml，则var\_f ile2.yml也应该在/tmp下。当然，完全可以使⽤绝对路径。

### roles中的变量

由于role是整合playbook的，它有默认的⽂件组织结构。其中有⼀个⽬录vars，其内的main.yml⽤于定义变量。还 有def aults⽬录内的main.yml则是定义role默认变量的，默认变量的优先级最低。

**shell> tree /yaml**

**/yaml**

**├── roles**

**│ └── nginx**

**│ ├── defaults**

**│ └── main.yml**

**│ ├── files**

**│ ├── handlers**

**│ ├── meta**

**│ ├── tasks**

**│ ├── templates**

**│ └── vars**

**│ └── main.yml**

**└── site.yml**

main.yml中变量定义⽅式也是字典格式，例如：

**---**

**mysql\_port: 3306**

### 命令⾏传递变量

**ansible和ansible- playbook命令的"- e"选项都可以传递变量，传递的⽅式有两种： -e key=value 和-e @var\_file 。注意，当key=value⽅式传递变量时，如果变量中包含特殊字符，必须防⽌其被shell解析。**

例如：

**ansible localhost -m shell -a "echo {{say\_hi}}" -e 'say\_hi="hello world"' ansible localhost -m shell -a "echo {{say\_hi}}" -e @/tmp/var\_file1.yml**

其中/tmp/var\_f ile1.yml中的内容如下：

**---**

**say\_hi: hello world**

### 借助with\_items叠加变量

ansible中可以借助with\_items实现列表迭代的功能，作⽤于变量注册的⾏为上，就可以实现将多个结果赋值给同⼀ 个变量。

例如下⾯的playbook中，给出了3个item列表，并在shell模块中通过固定变量"{{item}}"分别迭代，第⼀次迭代的是haha，第⼆次迭代的是heihei，第三次迭代的是hehe，也就实现了3次循环。最后，将结果注册为变量hi\_var。

**---**

* **hosts: localhost**

**remote\_user: root tasks:**

* + **name: test #**

**shell: echo "{{item}}" with\_items:**

* + - **haha**
    - **heihei**
    - **hehe register: hi\_var**
  + **debug: var=hi\_var.results[0].stdout**
  + **debug: var=hi\_var.results[1].stdout**
  + **debug: var=hi\_var.results[2].stdout**

**每次迭代的过程中，调⽤item的模块都会将结果保存在⼀个key为results的数组中。因此，引⽤迭代后注册的变量 时，需要在变量名中加上results，并指定数组名。例如上⾯的 hi\_var.results[N].stdout 。**

还可以使⽤f or循环遍历列表。例如：

**- debug: msg="{% for i in hi\_var.results %} {{i.stdout}} {% endfor %}"**

其实，看⼀下hi\_var的输出就很容易理解了。以下是hi\_var的第⼀个列表的输出。

**"hi\_var": {**

**"changed": true,**

**"msg": "All items completed", "results": [**

**{**

**"\_ansible\_item\_result": true, "\_ansible\_no\_log": false, "\_ansible\_parsed": true, "changed": true,**

**"cmd": "echo \"haha\"",**

**"delta": "0:00:00.001942",**

**"end": "2017-09-21 04:45:57.032946",**

**"invocation": { "module\_args": {**

**"\_raw\_params": "echo \"haha\"", "\_uses\_shell": true,**

**"chdir": null, "creates": null, "executable": null, "removes": null, "warn": true**

**}**

**},**

**"item": "haha", "rc": 0,**

**"start": "2017-09-21 04:45:57.031004",**

**"stderr": "",**

**"stderr\_lines": [],**

**"stdout": "haha", "stdout\_lines": [**

**"haha"**

**]**

**}**

### inventory中主机变量和主机组变量

在inventory⽂件中可以为主机和主机组定义变量，不仅包括内置变量赋值，还包括⾃定义变量赋值。例如以下inventory⽂件。

**192.168.100.65 ansible\_ssh\_port=22 var1=1 [centos7]**

**192.168.100.63**

**192.168.100.64**

**192.168.100.65 var1=2**

**[centos7:vars] var1=2.2 var2=3 [all:vars] var2=4**

**其中** ansible**\_ssh\_port 是主机内置变量，为其赋值22，这类变量是设置类变量，不能被引⽤。此外还在多处为主机**

**192.168.100.65进⾏了赋值。其中 [centos7:vars] 和 [all:vars] 表⽰为主机组赋值，前者是为centos7这个组赋值，**

后者是为所有组赋值。

以下是执⾏语句：

**shell> ansible 192.168.100.65 -i /tmp/hosts -m shell -a 'echo "{{var1}} {{var2}}"' 192.168.100.65 | SUCCESS | rc=0 >>**

**2 3**

从结果可知，主机变量优先级⾼于主机组变量，给定的主机组变量优先级⾼于all特殊组。

除了在inventory⽂件中定义主机、主机组变量，还可以将其定义在host\_vars和group\_vars⽬录下的独⽴的⽂件 中，但要求这些host\_vars或group\_vars这两个⽬录和inventory⽂件或playbook⽂件在同⼀个⽬录下，且变量的

⽂件以对应的主机名或主机组名命名。

例如，inventory⽂件路径为/etc/ansible/hosts，playbook⽂件路径为/tmp/x.yml，则主机192.168.100.65和主机组centos7的变量⽂件路径可以为以下⼏种：

/etc/ansible/host\_vars/192.168.100.65

/etc/ansible/group\_vars/centos7

/tmp/host\_vars/192.168.100.65

/tmp/group\_vars/centos7

以下为⼏个host\_vars和group\_vars⽬录下的⽂件内容。

**shell> cat /etc/ansible/{host\_vars/192.168.100.65,group\_vars/centos7} \**

**/tmp/{host\_vars/192.168.100.65,group\_vars/centos7}**

**var1: 1**

**var2: 2**

**var3: 3**

**var4: 4**

以下为/tmp/x.yml的内容。

**---**

**- hosts: 192.168.100.65**

**tasks:**

**- debug: msg='{{var1}} {{var2}} {{var3}} {{var4}}'**

执⾏结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.65] => {**

**"msg": "1 2 3 4"**

**}**

### 内置变量

ansible除了inventory中内置的⼀堆不可被引⽤的设置类变量，还有⼏个全局都可以引⽤的内置变量，主要有以下

⼏个：

inventory\_hostname、inventory\_hostname\_short、groups、group\_names、hostvars、play\_hosts、inventory\_dir和ansible\_version。

1. ***inventory\_hostname*和*inventory\_hostname\_short***

分表代表的是inventory中被控节点的主机名和主机名的第⼀部分，如果定义的是主机别名，则变量的值也是别名。 例如inventory中centos7主机组定义为如下：

**[centos7] 192.168.100.63**

**host1 ansible\_ssh\_host=192.168.100.64** [**www.host2.com**](http://www.host2.com/) **ansible\_ssh\_host=192.168.100.65**

**分别输出它们的inventory\_hostname 和inventory\_hostname\_short 。**

**shell> ansible centos7 -m debug -a 'msg="{{inventory\_hostname}} & {{inventory\_hostname\_short}}"' 192.168.100.63 | SUCCESS => {**

**"msg": "192.168.100.63 & 192"**

**}**

**host1 | SUCCESS => { "msg": "host1 & host1"**

**}**

[**www.host2.com**](http://www.host2.com/) **| SUCCESS => { "msg": "**[**www.host2.com**](http://www.host2.com/) **& www"**

**}**

1. ***groups*和*group\_names***

group\_names返回的是主机所属主机组，如果该主机在多个组中，则返回多个组，如果它不在组中，则返回ungrouped这个特殊组。

例如，某个inventory⽂件如下：

**192.168.100.60**

**192.168.100.63**

**192.168.100.64**

**192.168.100.65**

**[centos6] 192.168.100.60**

**[centos7] 192.168.100.63**

**host1 ansible\_ssh\_host=192.168.100.64** [**www.host2.com**](http://www.host2.com/) **ansible\_ssh\_host=192.168.100.65 [centos:children]**

**centos6 centos7**

其中100.60定义在centos6和centos中，所以返回这两个组。同理100.63返回centos7和centos。100.64和100.65则返回ungrouped，虽然它们在centos7中都定义了别名，但⾄少将100.64和100.65作为主机名时，它们是 不在任何主机中的。另⼀⽅⾯，host1和[www.host2.com这两个别名主机都返回centos7和centos两个组。](http://www.host2.com这两个别名主机都返回centos7和centos两个组/)

groups变量则是返回其所在inventory⽂件中所有组和其内主机名。注意，该变量对每个控制节点都返回⼀次，所 以返回的内容可能⾮常多。例如，上⾯的inventory中，如果指定被控节点为centos7，则会重复返回3次(因为有3 台被控主机)该inventory⽂件。其中的第三台主机[www.host2.com的返回结果为：](http://www.host2.com的返回结果为/)

[**www.host2.com**](http://www.host2.com/) **| SUCCESS => { "msg": {**

**"all": [**

**"192.168.100.60",**

**"192.168.100.63",**

**"192.168.100.64",**

**"192.168.100.65",**

**"host1", "**[**www.host2.com**](http://www.host2.com/)**"**

**],**

**"centos": [**

**"192.168.100.60",**

**"192.168.100.63",**

**"host1", "**[**www.host2.com**](http://www.host2.com/)**"**

**],**

**"centos6": [**

**"192.168.100.60"**

**],**

**"centos7": [**

**"192.168.100.63",**

**"host1", "**[**www.host2.com**](http://www.host2.com/)**"**

**],**

**"ungrouped": [ "192.168.100.60",**

**"192.168.100.63",**

**"192.168.100.64",**

**"192.168.100.65"**

**]**

**}**

**}**

1. ***hostvars***

该变量⽤于引⽤其他主机上收集的f acts中的数据，或者引⽤其他主机的主机变量、主机组变量。其key为主机名或主机组名。

举个例⼦，假如使⽤ansible部署⼀台php服务器host1，且配置⽂件内需要指向另⼀台数据库服务器host2的ip地址ip2，可以直接在配置⽂件中指定ip2，但也可以在模板配置⽂件中直接引⽤host2收集的f acts数据中的

ansible\_eth0.ipv4.address变量。

例如，centos7主机组中包含了192.168.100.[63:65]共3台主机。playbook内容如下：

**---**

**- hosts: centos7**

**tasks:**

**- debug: msg="{{hostvars['192.168.100.63'].ansible\_eth0.ipv4.address}}"**

执⾏结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.63] => {**

**"msg": "192.168.100.63"**

**}**

**ok: [192.168.100.64] => {**

**"msg": "192.168.100.63"**

**}**

**ok: [192.168.100.65] => {**

**"msg": "192.168.100.63"**

**}**

但注意，在引⽤其他主机facts中数据时，要求被引⽤主机进⾏了facts收集动作，或者有facts缓存。否则都没收集，当然⽆法引⽤其f acts数据。也就是说，当被引⽤主机没有f acts缓存时，ansible的控制节点中必须同时包含引⽤主机和被引⽤主机。

除了引⽤其他主机的facts数据，还可以引⽤其他主机的主机变量和主机组变量，且不要求被引⽤主机有facts数据，因为主机变量和主机组变量是在ansible执⾏任务前加载的。

例如，inventory中格式如下：

**192.168.100.59**

**[centos7] 192.168.100.63 var63=63**

**192.168.100.64**

**192.168.100.65**

**[centos7:vars] var64=64**

playbook内容如下：

**---**

**- hosts: 192.168.100.59**

**tasks:**

**- debug: msg="{{hostvars['192.168.100.63'].var63}} & {{hostvars['192.168.100.65'].var64}}"**

执⾏结果如下：

**TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ok: [192.168.100.59] => {**

**"msg": "63 & 64"**

**}**

1. ***play\_hosts*和*inventory\_dir***

play\_hosts代表的是当前play所涉及inventory内的所有主机名列表。例如，inventory内容为：

**192.168.100.59**

**[centos6] 192.168.100.62**

**192.168.100.63**

**[centos7] 192.168.100.64**

**192.168.100.65**

那么，该inventory内的任意⼀或多台主机作为ansible或ansible- playbook的被控节点时，都会返回整个inventory

内的所有主机名称。

inventory\_dir是所使⽤inventory所在的⽬录。

1. ***ansible\_version***

代表的是ansible软件的版本号。变量返回的内容如下：

**{**

**"full": "2.3.1.0",**

**"major": 2,**

**"minor": 3,**

**"revision": 1,**

**"string": "2.3.1.0"**

**}**

最后，不得不说ansible的变量定义⽅式太丰富了，但是ansible的官⽅⼿册真的恶⼼到吐，太烂了。

# 条件判断和循环

## 循环

ansible中的循环都是借助迭代来实现的。基本都是以"with\_"开头。以下是常见的⼏种循环。

### with\_items迭代列表

ansibel⽀持迭代功能。例如，有⼀⼤堆要输出的命令、⼀⼤堆要安装的软件包、⼀⼤堆要copy的⽂件等等。 例如，要安装⼀堆软件包。

**---**

* **hosts: localhost**

**tasks:**

* + **yum: name="{{item}}" state=installed with\_items:**
    - **pkg1**
    - **pkg2**
    - **pkg3**

它会⼀个⼀个迭代到特殊变量"{{item}}"处。

再例如，指定⼀堆⽂件列表，然后使⽤grep搜索出给定⽂件列表中包含"[www.example.com](http://www.example.com/)"字符串的⽂件：

**---**

* **hosts: localhost**

**tasks:**

* + **shell: grep -Rl "www\.example\.com" "{{item}}" with\_items:**
    - **file1**
    - **file2**
    - **file3**

**register: match\_file**

* + **debug: msg="{% for i in match\_file.results %} {{i.stdout}} {% endfor %}"**

注意，将with\_items迭代后的结果注册为变量时，其注册结果也是列表式的，且其key为"results"。具体的结果⽐ 较长，可以使⽤debug模块的var或msg参数观察match\_f ile变量的结果。

在上⾯，是使⽤f or循环进⾏引⽤的。如果不使⽤f or循环，那么就需要使⽤数组格式。例如，引⽤match\_f ile中的第⼀和第⼆个结果。

* **debug: var=match\_file.results[0].stdout**
* **debug: var=match\_file.results[1].stdout**

显然，不如循环引⽤更好，因为不知道match\_f ile中到底有⼏个匹配⽂件，也就不能确定match\_f ile中的列表数量。

每个列表项中可能都包含⼀个或多个字典，既然with\_items迭代的是列表项，那么肯定也能迭代列表中的各字典。 例如：

**tasks:**

* **command: echo {{ item }} with\_items: [ 0, 2, 4, 6, 8, 10 ] register: num**
* **debug: msg="{% for i in num.results %} {{i.stdout}} {% endfor %}"**

再例如：

**---**

* **hosts: localhost**

**tasks:**

* + **shell: echo "name={{item.name}},age={{item.age}}" with\_items:**
    - **{name: zhangsan,age: 32}**
    - **{name: lisi,age: 33}**
    - **{name: wangwu,age: 35} register: who**
  + **debug: msg="{% for i in who.results %} {{i.stdout}} {% endfor %}"**

### with\_dict迭代字典项

使⽤"with\_dict"可以迭代字典项。迭代时，使⽤"item.key"表⽰字典的key，"item.value"表⽰字典的值。例如：

**---**

**- hosts: localhost**

**tasks:**

**- debug: msg="{{item.key}} & {{item.value}}" with\_dict: { address: 1,netmask: 2,gateway: 3 }**

另⼀种情况，字典是已存储好的。例如ansible f acts中的ansible\_eth0.ipv4，其内容如下：

**"ipv4": {**

**"address": "192.168.100.65",**

**"netmask": "255.255.255.0",**

**"gateway": "192.168.100.2"**

**}**

这种情况下，with\_dict处可以直接指定该字典的key。即：

**---**

**- hosts: localhost**

**tasks:**

**- debug: msg="{{item.key}} & {{item.value}}" with\_dict: ansible\_eth0.ipv4**

再例如，直接引⽤playbook中定义的vars。

**---**

**- hosts: 192.168.100.65**

**gather\_facts: False vars:**

**user:**

**longshuai\_key: name: longshuai gender: Male**

**xiaofang\_key: name: xiaofang gender: Female**

**tasks:**

**- name: print hash loop var**

**debug: msg="{{ item.key }} & {{ item.value.name }} & {{ item.value.gender }}" with\_dict: "{{ user }}"**

### with\_fileglob迭代⽂件

例如，拷贝⼀堆⽤通配符匹配出来的⽂件到各远程主机上。

**---**

* **hosts: centos**

**tasks:**

* + **copy: src="{{item}}" dest=/tmp/ with\_fileglob:**
    - **/tmp/\*.sh**
    - **/tmp/\*.py**

注意，通配符⽆法匹配"/"，因此⽆法递归到⼦⽬录中，也就⽆法迭代⼦⽬录中的⽂件。

### with\_lines迭代⾏

with\_lines很好⽤，可以将命令⾏的输出结果按⾏迭代。例如，f ind⼀堆⽂件出来，copy⾛。

**---**

**- hosts: localhost**

**tasks:**

**- copy: src="{{item}}" dest=/tmp/yaml with\_lines:**

**- find /tmp -type f -name "\*.yml"**

### with\_nested嵌套迭代

嵌套迭代是指多次迭代列表项。例如：

**---**

**- hosts: localhost**

**tasks:**

**- debug: msg="{{item[0]}} & {{item[1]}}" with\_nested:**

**- [a,b]**

**- [1,2,3]**

结果将得到"a & 1"、"a & 2"、"a & 3"、"b & 1"、"b & 2"和"b & 3"共6个结果。

## 条件判断

在ansible中，只有when可以实现条件判断。

**tasks:**

**- name: config the yum repo for centos 6 yum\_repository:**

**name: epel description: epel**

**baseurl:** [**http://mirrors.aliyun.com/epel/6/$basearch/**](http://mirrors.aliyun.com/epel/6/%24basearch/) **gpgcheck: no**

**when: ansible\_distribution\_major\_version == "6"**

注意两点：

when判断的对象是task，所以和task在同⼀列表层次。它的判断结果决定它所在task是否执⾏，⽽不是它下⾯ 的task是否执⾏。

when中引⽤变量的时候不需要加{{ }}符号。此外，还⽀持各种逻辑组合。

**tasks:**

**# 逻辑或**

* **command: /sbin/shutdown -h now**

**when: (ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "6") or (ansible\_distribution == "Debian" and ansible\_distribution\_major\_version == "7")**

**# 逻辑与**

* **command: /sbin/shutdown -t now**

**when:**

* + **ansible\_distribution == "CentOS"**
  + **ansible\_distribution\_major\_version == "6"**

**# 取 反**

* **command: /sbin/shutdown -t now**

**when: not ansible\_distribution == "CentOS"**

还可以直接直接引⽤布尔值的变量。

**---**

**- hosts: localhost**

**vars:**

**epic: False**

**tasks:**

**- debug: msg="This certainly is epic!" when: not epic**

此外，可以使⽤jinja2的def ined来测试变量是否已定义，使⽤undef ined可以取反表⽰未定义。例如：

**tasks:**

* **shell: echo "I've got '{{ foo }}' and am not afraid to use it!" when: foo is defined**
* **fail: msg="Bailing out. this play requires 'bar'" when: bar is undefined**

# 执⾏过程分析、异步模式和速度优化

## ansible执⾏过程分析

使⽤ansible的-vvv 或-vvvv 分析执⾏过程。以下是⼀个启动远程192.168.100.61上httpd任务的执⾏过程分析。其中将不必要的信息都是⽤". "替换了。

**# 读取配置⽂件，然后开始执⾏对应的处理程序。Using /etc/ansible/ansible.cfg as config file META: ran handlers**

**# 第⼀个任务默认都是收集远程主机信息的任务。**

**# 第⼀个收集任务，加载setup模块**

**Using module file /usr/lib/python2.7/site-packages/ansible/modules/system/setup.py**

**# 建⽴连接，获取被控节点当前⽤户的家⽬录，⽤于存放稍后的临时任务⽂件，此处返回值为/root。**

**# 在-vvv的结果中，第⼀⾏属于描述性信息，第⼆⾏为代码执⾏段，第三⾏类似此处的<host\_node>(，，，，，)为上⼀段代码的返回结果。后同**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C ..........................**

**<192.168.100.61> (0, '/root\n', '')**

**# 再次建⽴连接，在远端创建临时任务⽂件的⽬录，临时⽬录由配置⽂件中的remote\_tmp指令控制**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C .......................... '/bin/sh -c '"'"'( umask 77 && mkdir -p "` echo /root/**

**.ansible/tmp/ansible-tmp-1495977564.58-40718671162202 `" && echo ansible-tmp-1495977564.58-40718671162202="` echo / root/.ansible/tmp/ansible-tmp-1495977564.58-40718671162202 `" ) && sleep 0'"'"''**

**<192.168.100.61> (0, 'ansible-tmp-1495977564.58-40718671162202=/root/.ansible/tmp/ansible-tmp-1495977564.58-4071867 1162202\n', '')**

**# 将要执⾏的任务放到临时⽂件中，并使⽤sftp将任务⽂件传输到被控节点上**

**<192.168.100.61> PUT /tmp/tmpY5vJGX TO /root/.ansible/tmp/ansible-tmp-1495977564.58-40718671162202/setup.py**

**<192.168.100.61> SSH: EXEC sftp -b - -C ............. '[192.168.100.61]'**

**<192.168.100.61> (0, 'sftp> put /tmp/tmpY5vJGX /root/.ansible/tmp/ansible-tmp-1495977564.58-40718671162202/setup.py**

**\n', '')**

**# 建⽴连接，设置远程任务⽂件其所有者有可执⾏权限**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C ......................... '/bin/sh -c '"'"'chmod u+x /root/.ansible/tmp/ansible-t mp-1495977564.58-40718671162202/ /root/.ansible/tmp/ansible-tmp-1495977564.58-40718671162202/setup.py && sleep 0'"' "''**

**<192.168.100.61> (0, '', '')**

**# 建⽴连接，执⾏任务，执⾏完成后⽴即删除任务⽂件，并返回收集到的信息给ansible。到此为⽌，setup收集任务结束，关闭共享连接**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C ............ '/bin/sh -c '"'"'/usr/bin/python /root/.ansible/tmp/ansible-tmp-1495 977564.58-40718671162202/setup.py; rm -rf "/root/.ansible/tmp/ansible-tmp-1495977564.58-40718671162202/" > /dev/nul l 2>&1 && sleep 0'"'"''**

**<192.168.100.61> (0, '\r\n{"invocation": {"...........', 'Shared connection to 192.168.100.61 closed.\r\n')**

**# 进⼊下⼀个任务，此处为服务管理任务，所以加载service模块**

**Using module file /usr/lib/python2.7/site-packages/ansible/modules/system/service.py**

**# 建⽴连接，获取被控节点当前⽤户的家⽬录，⽤于存放稍后的临时任务⽂件，此处返回值为/root**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C .........................**

**<192.168.100.61> (0, '/root\n', '')**

**# 建⽴连接，将要执⾏的任务放⼊到临时⽂件中，并传输到远程⽬录**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C .............. '/bin/sh -c '"'"'( umask 77 && mkdir -p "` echo /root/.ansible/tmp**

**/ansible-tmp-1495977564.97-137863382080241 `" && echo ansible-tmp-1495977564.97-137863382080241="` echo /root/.ansi ble/tmp/ansible-tmp-1495977564.97-137863382080241 `" ) && sleep 0'"'"''**

**<192.168.100.61> (0, 'ansible-tmp-1495977564.97-137863382080241=/root/.ansible/tmp/ansible-tmp-1495977564.97-137863 382080241\n', '')**

**<192.168.100.61> PUT /tmp/tmpn5uZhP TO /root/.ansible/tmp/ansible-tmp-1495977564.97-137863382080241/service.py**

**<192.168.100.61> SSH: EXEC sftp -b - -C .............. '[192.168.100.61]'**

**<192.168.100.61> (0, 'sftp> put /tmp/tmpn5uZhP /root/.ansible/tmp/ansible-tmp-1495977564.97-137863382080241/service**

**.py\n', '')**

**# 建⽴连接，设置远程任务⽂件其所有者有可执⾏权限**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C ........................ '/bin/sh -c '"'"'chmod u+x /root/.ansible/tmp/ansible-tm p-1495977564.97-137863382080241/ /root/.ansible/tmp/ansible-tmp-1495977564.97-137863382080241/service.py && sleep 0 '"'"''**

**<192.168.100.61> (0, '', '')**

**# 建⽴连接，执⾏任务，执⾏完成后⽴即删除任务⽂件，并将执⾏的结果返回到ansible端。到此为⽌，service模块任务执⾏结束，关闭共享连接**

**<192.168.100.61> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.61> SSH: EXEC ssh -C .............. '/bin/sh -c '"'"'/usr/bin/python /root/.ansible/tmp/ansible-tmp-14 95977564.97-137863382080241/service.py; rm -rf "/root/.ansible/tmp/ansible-tmp-1495977564.97-137863382080241/" > /d ev/null 2>&1 && sleep 0'"'"''**

**<192.168.100.61> (0, '\r\n{"msg": "............', 'Shared connection to 192.168.100.61 closed.\r\n')**

将上⾯的进⾏总结，执⾏过程将是这样的：

读取配置⽂件

加载inventory⽂件。包括主机变量和主机组变量执⾏第⼀个任务：收集远程被控节点的信息

建⽴连接，获取家⽬录信息

将要执⾏的收集任务放到临时⽂件中

将临时任务⽂件传输到被控节点的临时⽬录中

ssh连接到远端执⾏收集任务删除任务⽂件

将收集信息返回给ansible端关闭连接

执⾏第⼆个任务，此为真正的主任务建⽴连接，获取家⽬录信息

将要执⾏的任务放到临时⽂件中

将临时任务⽂件传输到被控节点的临时⽬录中

ssh连接到远端执⾏任务删除任务⽂件

将执⾏结果返回给ansible端，ansible输出到屏幕或指定⽂件中关闭连接

执⾏第三个任务执⾏第四个任务

如果是多个被控节点，那么将同时在多个节点上并⾏执⾏每⼀个任务，例如同时执⾏信息收集任务。不同节点之间的任务没有先后关系，主要依赖于性能。每⼀个任务执⾏完毕都会⽴即将结果返回给ansible端，所以可以通过ansible端结果的输出顺序和速度判断执⾏完毕的先后顺序。

如果节点数太多，ansible无法一次在所有远程节点上执行任务，那么将先在一部分节点上执行一个任务(每一批节点的数量取决于f ork进程数量)，直到这一批所有节点上该任务完全执行完毕才会接入下一个批节点(数量取决于f ork进程数量)，直到所有节点将该任务都执行完毕，然后重新回到第一批节点开始执行第二个任务。依次类推，直到所有节点执行完所有任务，ansible端才会释放shell。这是默认的同步模式**，也就是说在未执⾏完毕的时**



候，ansible是占⽤当前shell的，任务执⾏完毕后，释放shell了才可以输⼊其他命令做其他动作。

如果是异步模式，假如f ork控制的并发进程数为5，远程控制节点为24个，则ansible⼀开始会将5个节点的任务扔在后台，并每隔⼀段时间去检查这些节点的任务完成情况，当某节点完成不会⽴即返回， ⽽是继续等待直到5个进程都空闲了，才会将这5个节点的结果返回给ansible端，ansible会继续将下⼀批5个节点的任务扔在后台并每隔

⼀段时间进⾏检查，依次类推，直到完成所有任务。

在异步模式下，如果设置的检查时间间隔为0，在将每⼀批节点的任务丢到后台后都会⽴即返回ansible，并⽴即将 下⼀批节点的任务丢到后台，直到所有任务都丢到后台完成后，会返回ansible端，ansible会⽴即释放占⽤的

shell。也就是说，此时ansible是不会管各个节点的任务执⾏情况的，不管执⾏成功还是失败。

因此，在轮训检查时间内，ansible仍然正在运⾏(尽管某批任务已经被放到后台执⾏了)，当前shell进程仍被占⽤处 于睡眠状态，只有指定的检查时间间隔为0，才会尽快将所有任务放到后台并释放shell。

需要注意3点：

1. **.按批(例如每次5台全部完成⼀个任务才进⼊下⼀批的5台)完成任务的模式在ansible 2.0版本之后可以通过修改ansible的执⾏策略来改变(见后⽂)，改变后会变成"前赴后继"的执⾏模式：当⼀个节点执⾏完⼀个任务会⽴即接⼊ 另⼀个节点，不再像默认情况⼀样等待这⼀批中的其他节点完成该任务。**
2. **.上⾯执⾏过程是默认的执⾏过程，如果开启了pipelining加速**ansible**执⾏效率，会省去sf tp到远端的过程。3 .信息收集任务是默认会执⾏的，但是可以设置禁⽤它。**

## ansible并发和异步

上⾯已经对ansible的执⾏过程进⾏了很详细的分析，也解释了同步和异步的模式是如何处理任务的。所以此处简单举⼏个例⼦。

ansible默认只会创建5个进程并发执⾏任务，所以⼀次任务只能同时控制5台机器执⾏。如果有⼤量的机器需要控制，例如20台，ansible执⾏⼀个任务时会先在其中5台上执⾏，执⾏成功后再执⾏下⼀批5台，直到全部机器执⾏完毕。使⽤- f 选项可以指定进程数，指定的进程数量多⼀些，不仅会实现全并发，对异步的轮训poll也会有正⾯影响。

ansible默认是同步阻塞模式，它会等待所有的机器都执⾏完毕才会在前台返回。可以采取异步执⾏模式。

异步模式下，ansible会将节点的任务丢在后台，每台被控制的机器都有⼀个job\_id，ansible会根据这个job\_id去轮 训该机器上任务的执⾏情况，例如某机器上此任务中的某⼀个阶段是否完成，是否进⼊下⼀个阶段等。即使任务早就结束了，但只有轮训检查到任务结束后才认为该job结束。可以指定任务检查的时间间隔，默认是10秒。除⾮指 定任务检查的间隔为0，否则会等待所有任务都完成后，ansible端才会释放占⽤的shell。



如果指定时间间隔为0，则ansible会⽴即返回(⾄少得连接上⽬标主机，任务发布成功之后⽴即返回)，并不会去检查它的任务进度。

**ansible centos -B200 -P 0 -m yum -a "name=dos2unix" -o -f 6**

**192.168.100.61 | SUCCESS => {"ansible\_job\_id": "986026954359.9166", "changed": true, "finished": 0, "results\_file": "/root/.ansible\_async/986026954359.9166", "started": 1}**

* + - 1. **| SUCCESS => {"ansible\_job\_id": "824724696770.9431", "changed": true, "finished": 0, "results\_file": "/root/.ansible\_async/824724696770.9431", "started": 1}**
      2. **| SUCCESS => {"ansible\_job\_id": "276581152579.10006", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/276581152579.10006", "started": 1}**

**192.168.100.64 | SUCCESS => {"ansible\_job\_id": "237326453903.72268", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/237326453903.72268", "started": 1}**

**192.168.100.63 | SUCCESS => {"ansible\_job\_id": "276700021098.73070", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/276700021098.73070", "started": 1}**

**192.168.100.65 | SUCCESS => {"ansible\_job\_id": "877427488272.72032", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/877427488272.72032", "started": 1}**

关于同步、异步以及异步时的并⾏数、轮训间隔对ansible的影响，通过以下⽰例说明： 当有6个节点时，仅就释放shell的速度⽽⾔，以下⼏种写法：

**# 同步模式，⼤约10秒返回。**

**ansible centos -m command -a "sleep 5" -o # 异步模式，分两批执⾏。⼤约10秒返回。**

**ansible centos -B200 -P 1 -m command -a "sleep 5" -o -f 5**

**# 异步模式，和上⼀条命令时间差不多，但每次检查时间长⼀秒，所以可能会稍有延迟。⼤约11-12秒返回。**

**ansible centos -B200 -P 2 -m command -a "sleep 5" -o -f 5 # 异步模式，⼀批就执⾏完，⼤约5-6秒返回。**

**ansible centos -B200 -P 1 -m command -a "sleep 5" -o -f 6 # 异步模式，⼀批就完成，⼤约5-6秒完成。**

**ansible centos -B200 -P 2 -m command -a "sleep 5" -o -f 6**

**# 异步模式，分两批，且检查时间过长。即使只睡眠5秒，但仍需要10秒才能判断该批执⾏结束。所以⼤约20秒返回。**

**ansible centos -B200 -P 10 -m command -a "sleep 5" -o -f 5**

**# 异步模式，⼀批执⾏完成，但检查时间超过睡眠时间，因此⼤约10秒返回。**

**ansible centos -B200 -P 10 -m command -a "sleep 5" -o -f 6**

在异步执⾏任务时，需要注意那些有依赖性的任务。对于那些对资源要求占有排它锁的任务，如yum，不应该将

Poll的间隔设置为0。如果设置为0，很可能会导致资源阻塞。

总结来说,⼤概有以下⼀些场景需要使⽤到ansible的异步特性：

某个task需要运⾏很长的时间,这个task很可能会达到ssh连接的timeout

没有任务是需要等待它才能完成的，即没有任务依赖此任务是否完成的状态需要尽快返回当前shell

当然也有⼀些场景不适合使⽤异步特性：

这个任务是需要运⾏完后才能继续另外的任务的申请排它锁的任务

当然，对于有任务依赖性的任务，也还是可以使⽤异步模式的，只要检查它所依赖的主任务状态已完成就可以。例如，要配置nginx，要求先安装好nginx，在配置nginx之前先检查yum安装的状态。

* **name: 'YUM - fire and forget task' yum: name=nginx state=installed async: 1000**

**poll: 0**

**register: yum\_sleeper**

* **name: 'YUM - check on fire and forget task' async\_status: jid={{ yum\_sleeper.ansible\_job\_id }} register: job\_result**

**until: job\_result.finished retries: 30**

## ansible的-t选项妙⽤

ansible的"- t"或"-- tree"选项是将ansible的执⾏结果按主机名保存在指定⽬录下的⽂件中。

有些时候，ansible执⾏起来的速度会⾮常慢，这种慢体现在即使执⾏的是⼀个⽴即返回的简单命令(如ping模块)， 也会耗时很久，且不是因为ssh连接慢导致的。如果使⽤- t选项，将第⼀次执⾏得到的结果按inventory中定义的主机名保存在⽂件中，下次执⾏到同⼀台主机时速度将会变快很多，即使之后不再加上- t选项，也可以在⼀定时间内保持迅速执⾏。即使执⾏速度正常（如执⾏⼀个Ping命令0.7秒左右），使⽤- t选项也可以在此基础上变得更快。

除了使⽤- t选项，使⽤重定向将结果重定向到某个⽂件中也是⼀样的效果。⾄于为何会如此，我也不知道，是在⽆

意中测试出来的。有必要指出：我在CentOS 6.6上遇到过这样的问题，但并不是总会如此，且在CentOS 7上正常。因此，如果你也出现了这样的问题，可以参考这种偏⽅。

以CentOS 6.6安装的ansible 2.3为例，正常执⾏ansible会⾮常慢，使⽤- t可以解决这个问题。如下。没有使⽤- t时：移除dos2unix包所需时间为13秒多。

**time ansible centos -B200 -P 0 -m yum -a "name=dos2unix state=removed" -o -f 6**

**192.168.100.60 | SUCCESS => {"ansible\_job\_id": "987125400759.10653", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/987125400759.10653", "started": 1}**

**192.168.100.63 | SUCCESS => {"ansible\_job\_id": "735153954362.74074", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/735153954362.74074", "started": 1}**

**192.168.100.61 | SUCCESS => {"ansible\_job\_id": "192721090554.9813", "changed": true, "finished": 0, "results\_file": "/root/.ansible\_async/192721090554.9813", "started": 1}**

**192.168.100.64 | SUCCESS => {"ansible\_job\_id": "494724112239.73269", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/494724112239.73269", "started": 1}**

**192.168.100.59 | SUCCESS => {"ansible\_job\_id": "2259915341.10078", "changed": true, "finished": 0, "results\_file": "/root/.ansible\_async/2259915341.10078", "started": 1}**

**192.168.100.65 | SUCCESS => {"ansible\_job\_id": "755223232484.73025", "changed": true, "finished": 0, "results\_file"**

**: "/root/.ansible\_async/755223232484.73025", "started": 1}**

**real 0m13.746s**

**user 0m1.288s**

**sys 0m1.417s**

使⽤- t选项后：安装dos2unix只需1.9秒左右。

**time ansible centos -B200 -P 0 -m yum -a "name=dos2unix state=installed" -o -f 6 -t /tmp/a**

**real 0m1.933s**

**user 0m0.398s**

**sys 0m0.900s**

之后即使不再使⽤- t选项，对同样的主机进⾏操作，速度也会变得⾮常快。

**time ansible centos -B200 -P 0 -m yum -a "name=dos2unix state=removed" -o -f 6**

**real 0m1.730s**

**user 0m0.892s**

**sys 0m0.572s**

⾄于保存的内容为何？实际上仅仅只是保存了普通的输出内容⽽已。

**ll /tmp/a/ total 24**

**-rw-r--r-- 1 root root 145 May 28 15:54 192.168.100.59**

**-rw-r--r-- 1 root root 145 May 28 15:54 192.168.100.60**

**-rw-r--r-- 1 root root 143 May 28 15:54 192.168.100.61**

**-rw-r--r-- 1 root root 143 May 28 15:54 192.168.100.63**

**-rw-r--r-- 1 root root 145 May 28 15:54 192.168.100.64**

**-rw-r--r-- 1 root root 145 May 28 15:54 192.168.100.65**

**cat /tmp/a/192.168.100.59**

**{"ansible\_job\_id": "659824383578.10145", "changed": true, "finished": 0, "results\_file": "/root/.ansible\_async/6598 24383578.10145", "started": 1}**

## 优化ansible速度

最初，ansible的执⾏效率和saltstack(基于zeromq消息队列的⽅式)相⽐要慢的多的多，特别是被控节点量很⼤的 时候。但是ansible发展到现在，它的效率得到了极⼤的改善。在被控节点不太多的时候，默认的设置已经够快，即 使被控节点数量巨⼤的时候，也可以通过⼀些优化，极⼤的提⾼其执⾏效率。

前⾯"- t"选项也算是⼀种提速⽅式，但算是"bug"式的问题，所以没有通⽤性。

### 设置ansible开启ssh长连接

ansible天然⽀持openssh，默认连接⽅式下，它对ssh的依赖性⾮常强。所以优化ssh连接，在⼀定程度上也在优 化ansible。其中⼀点是开启ssh的长连接，即长时间保持连接状态。

要开启ssh长连接，要求ansible端的openssh版本⾼于或等于5.6。使⽤ ssh -V 可以查看版本号。然后设置ansible

使⽤ssh连接被控端的连接参数，此处修改/etc/ansible/ansible.cf g，在此⽂件中启动下⾯的连接选项，其中ControlPersist=5d是控制ssh连接会话保持时长为5天。

**ssh\_args = -C -o ControlMaster=auto -o ControlPersist=5d**

除此之外直接设置/etc/ssh/ssh\_conf ig(不是sshd\_conf ig，因为ssh命令是客户端命令)中对应的长连接项也是可以的。

开启长连接后，在会话过期前会⼀直建⽴连接，在netstat的结果中会看到ssh连接是⼀直established状态，且会 在当前⽤户家⽬录的".ansible/cp"⽬录下⽣成⼀些socket⽂件，每个会话⼀个⽂件。

例如：执⾏⼀次ad- hoc操作。

**shell> netstat -tnalp**

**Active Internet connections (servers and established)**

**ansible centos -m ping**

查看netstat，发现ssh进程的会话⼀直是established状态。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Proto** | **Recv-Q** | **Send-Q** | **Local Address** | **Foreign Address** | **State** | **PID/Program name** |
| **tcp** | **0** | **0** | **0.0.0.0:22** | **0.0.0.0:\*** | **LISTEN** | **1143/sshd** |
| **tcp** | **0** | **0** | **127.0.0.1:25** | **0.0.0.0:\*** | **LISTEN** | **2265/master** |
| **tcp** | **0** | **0** | **192.168.100.62:58474** | **192.168.100.59:22** | **ESTABLISHED** | **31947/ssh: /root/.a** |
| **tcp** | **0** | **96** | **192.168.100.62:22** | **192.168.100.1:8189** | **ESTABLISHED** | **29869/sshd: root@pt** |
| **tcp** | **0** | **0** | **192.168.100.62:37718** | **192.168.100.64:22** | **ESTABLISHED** | **31961/ssh: /root/.a** |
| **tcp** | **0** | **0** | **192.168.100.62:38894** | **192.168.100.60:22** | **ESTABLISHED** | **31952/ssh: /root/.a** |
| **tcp** | **0** | **0** | **192.168.100.62:48659** | **192.168.100.61:22** | **ESTABLISHED** | **31949/ssh: /root/.a** |
| **tcp** | **0** | **0** | **192.168.100.62:33546** | **192.168.100.65:22** | **ESTABLISHED** | **31992/ssh: /root/.a** |
| **tcp** | **0** | **0** | **192.168.100.62:54824** | **192.168.100.63:22** | **ESTABLISHED** | **31958/ssh: /root/.a** |
| **tcp6** | **0** | **0** | **:::22** | **:::\*** | **LISTEN** | **1143/sshd** |
| **tcp6** | **0** | **0** | **::1:25** | **:::\*** | **LISTEN** | **2265/master** |

且家⽬录下"~/.ansible/cp/"下会⽣成对应的socket⽂件。

**ls -l ~/.ansible/cp/ total 0**

**srw------- 1 root root 0 Jun 3 18:26 5c4a6dce87**

**srw------- 1 root root 0 Jun 3 18:26 bca3850113**

**srw------- 1 root root 0 Jun 3 18:26 c89359d711**

**srw------- 1 root root 0 Jun 3 18:26 cd829456ec**

**srw------- 1 root root 0 Jun 3 18:26 edb7051c84**

**srw------- 1 root root 0 Jun 3 18:26 fe17ac7eed**

### 开启pipelining

pipeline也是openssh的⼀个特性。在ansible执⾏每个任务的流程中，有⼀个过程是将临时任务⽂件put到⼀个ansible端的⼀个临时⽂件中，然后sf tp传输到远端，然后通过ssh连接过去远程执⾏这个任务。如果开启了pipelining，⼀个任务的所有动作都在⼀个ssh会话中完成，也会省去sf tp到远端的过程，它会直接将要执⾏的任务在ssh会话中进⾏。

开启pipelining的⽅式是配置⽂件(如ansible.cf g)中设置pipelining=true，默认是f alse。

**shell> grep '^pipelining' /etc/ansible/ansible.cfg pipelining = True**

但是要注意，如果在ansible中使⽤sudo命令的话(ssh user@host sudo cmd)，需要在被控节点的/etc/sudoers中禁⽤"requiretty"。

之所以要设置/etc/sudoers中的requiretty，是因为ssh远程执⾏命令时，它的环境是⾮登录式⾮交互式shell，默认 不会分配tty，没有tty，ssh的sudo就⽆法关闭密码回显(使⽤"- tt"选项强制SSH分配tty)。所以出于安全考

虑，/etc/sudoers中默认是开启requiretty的，它要求只有拥有tty的⽤户才能使⽤sudo，也就是说ssh连接过去不 允许执⾏sudo。可以通过visudo编辑配置⽂件，注释该选项来禁⽤它。

**grep requiretty /etc/sudoers # Defaults requiretty**

修改设置/etc/sudoers是在被控节点上进⾏的(或者ansible连接过去修改)，其实在ansible端也可以解决sudo的问 题，只需在ansible的ssh参数上加上"- tt"选项即可。

**ssh\_args = -C -o ControlMaster=auto -o ControlPersist=5d -tt**

以下是开启pipelining前ansible执⾏过程，其中将很多不必要的信息使⽤ 来替代了。

**######## 开启pipelining前，执⾏ping模块的过程 ######## Using /etc/ansible/ansible.cfg as config file**

**Loading callback plugin minimal of type stdout, v2.0 from /usr/lib/python2.7/site-packages/ansible/plugins/callback**

**/ init .pyc META: ran handlers**

**Using module file /usr/lib/python2.7/site-packages/ansible/modules/system/ping.py**

**# ⾸先建⽴⼀次ssh连接，获取远端当前⽤户家⽬录，⽤于存放稍后的临时任务⽂件**

**<192.168.100.65> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.65> SSH: EXEC ssh -vvv -C ................ 192.168.100.65 '/bin/sh -c '"'"'echo ~ && sleep 0'"'"''**

**<192.168.100.65> (0, '/root\n', )**

**# 再次建⽴ssh连接，创建临时⽂件⽬录**

**<192.168.100.65> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.65> SSH: EXEC ssh -vvv -C ..................... 192.168.100.65 '/bin/sh -c '"'"'( umask 77 && mkdir -p "......." ) && sleep 0'"'"''**

**<192.168.100.65> (0, 'ansible-tmp-1496489511.13-10633592020239=/root/.ansible/tmp/ansible-tmp-1496489511.13-1063359 2020239\n', ' ')**

**# 将任务放⼊到本地临时⽂件中，然后使⽤sftp传输到远端**

**<192.168.100.65> PUT /tmp/tmp2\_VKGo TO /root/.ansible/tmp/ansible-tmp-1496489511.13-10633592020239/ping.py**

**<192.168.100.65> SSH: EXEC sftp -b - -vvv -C ................. '[192.168.100.65]'**

**<192.168.100.65> (0, 'sftp> put /tmp/tmp2\_VKGo /root/.ansible/tmp/ansible-tmp-1496489511.13-10633592020239/ping.py\ n', ' ')**

**# ⼜⼀次建⽴ssh连接，对任务⽂件进⾏授权**

**<192.168.100.65> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.65> SSH: EXEC ssh -vvv -C ............. 192.168.100.65 '/bin/sh -c '"'"'chmod u+x /ping.py**

**&& sleep 0'"'"''**

**<192.168.100.65> (0, '', '........................')**

**# 最后执⾏任务，完成任务后删除任务⽂件，并返回ansible端信息，注意ssh -tt选项，它强制为ssh会话分配tty，这样可以执⾏sudo命令**

**<192.168.100.65> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.65> SSH: EXEC ssh -vvv -C .............. -tt 192.168.100.65 '/bin/sh -c '"'"'/usr/bin/python .........**

**../ping.py; rm -rf ".........." > /dev/null 2>&1 && sleep 0'"'"''**

**<192.168.100.65> (0, '\r\n{"invocation": {" ')**

**以下是开启pipelining后，ansible执⾏过程。**

**########### 开启pipelining后，执⾏ping模块的过程 ########**

**Using /etc/ansible/ansible.cfg as config file**

**Loading callback plugin minimal of type stdout, v2.0 from /usr/lib/python2.7/site-packages/ansible/plugins/callback**

**/ init pyc**

**META: ran handlers**

**Using module file /usr/lib/python2.7/site-packages/ansible/modules/system/ping.py**

**# 只建⽴⼀次ssh连接，所有动作都在这⼀个ssh连接中完成，由于没有使⽤-tt选项，所以需要在被控主机上禁⽤requiretty选项**

**<192.168.100.65> ESTABLISH SSH CONNECTION FOR USER: None**

**<192.168.100.65> SSH: EXEC ssh -vvv -C ............ 192.168.100.65 '/bin/sh -c '"'"'/usr/bin/python && sleep 0'"'"' '**

**<192.168.100.65> (0, '\n{" ')**

从上⾯的过程对⽐中可以看到，开启pipelining后，每次执⾏任务时都⼤量减少了ssh连接次数(只需要⼀次ssh连 接)，且省去了sf tp传输任务⽂件的过程，因此在管理⼤量节点时能极⼤提升执⾏效率。

### 修改ansible执⾏策略

默认ansible在远程执⾏任务是按批并⾏执⾏的，⼀批控制多少台主机由命令⾏的"- f "或"-- f orks"选项控制。例如， 默认的并⾏进程数是5，如果有20台被控主机，那么只有在每5台全部执⾏完⼀个任务才继续下⼀批的5台执⾏该任务，即使中间某台机器性能较好，完成速度较快，它也会空闲地等待在那，直到所有20台主机都执⾏完该任务才会以同样的⽅式继续下⼀个任务。如下所⽰：

***h1 h2 h3 h4 h5(T1)-->h6 h7 h8 h9 h10(T1)...-->h16 h17 h18 h19 h20(T1)-->h1 h2 h3 h4 h5(T2)-->.....***

在ansible 2.0中，添加了⼀个策略控制选项strategy，默认值为"linear"，即上⾯按批并⾏处理的⽅式。还可以设置strategy的值为"f ree"。

在f ree模式下，ansible会尽可能快的切⼊到下⼀个主机。同样是上⾯的例⼦，⾸先每5台并⾏执⾏⼀个任务，当其中某⼀台机器由于性能较好提前完成了该任务，它不会等待其他4台完成，⽽是会跳出该任务让ansible切⼊到下⼀台机器来执⾏该任务。也就是说，这种模式下，⼀台主机完成⼀个任务后，另⼀台主机会⽴即执⾏任务，它是"前赴

后继"的⽅式。如下所⽰：

***h1 h2 h3 h4 h5(T1)-->h1 h2 h3 h4 h6(T1)-->h1 h3 h4 h6 h7(T1)--> ..-->h17 h18 h19 h20(T1) h1(T2)-->h18***

***h19 h20(T1) h1 h2(T2)-->...***

设置的⽅式如下：

**- hosts: all strategy: free tasks:**

**...**

### 设置facts缓存

ansible或ansible- playbook默认总是先收集f acts信息。在被控主机较少的情况下，收集信息还可以容忍，如果被控主机数量⾮常⼤，收集f acts信息会消耗掉⾮常多时间。

可以设置"gather\_f acts: no"来禁⽌ansible收集f acts信息，但是有时候⼜需要使⽤f acts中的内容，这时候可以设置f acts的缓存。例如，在空闲的时候收集f acts，缓存下来，在需要的时候直接读取缓存进⾏引⽤。

ansible的配置⽂件中可以修改'gathering'的值为'smart'、'implicit'或者'explicit'。smart表⽰默认收集f acts，但f acts已有的情况下不会收集，即使⽤缓存f acts；implicit表⽰默认收集f acts，要禁⽌收集，必须使⽤

**gather\_facts:**

**False ；explicit则表⽰默认不收集，要显式收集，必须使⽤ gather\_facts: Ture 。**

在使⽤f acts缓存时(即设置为smart)，ansible⽀持两种f acts缓存：redis和jsonf ile。例如，以下是/etc/ansible/ansible.cf g中jsonf ile格式的缓存配置⽅法。

**[defaults] gathering = smart**

**fact\_caching\_timeout = 86400 fact\_caching = jsonfile**

**fact\_caching\_connection = /path/to/cachedir**

这⾥设置的缓存过期时间为86400秒，即缓存⼀天。缓存的json⽂件放在/path/to/cachedir⽬录下，各主机的缓存

⽂件以主机名命名。缓存⽂件是⼀个json⽂件，要查看缓存⽂件，如/path/to/cachedir/192.168.100.59中的内容， 使⽤如下语句即可。

**cat /path/to/cachedir/192.168.100.59 | python -m json.tool**

# playbook杂项

实在是因为ansibel的官⽅⽂档太烂太不⼈性化了，⼀⼤堆的知识点乱七⼋糟的分布。所以，借此⽂收集⼀些乱七⼋ 糟的知识点，所以，此⽂会不断更新。

## 指定运⾏play的主机delegate\_to和local\_action

默认情况下，ansible会在指定的hosts的所有主机上运⾏整个play。但有些时候，有些任务只想在某些主机上运

⾏，例如，在haproxy主机上运⾏haproxy相关配置任务，在后端web主机上运⾏nginx或apache相关配置任务。

这时可以使⽤delegate\_to选项来分配任务，且分配的主机对象可以不⽤在hosts列表中，只要它在inventory

中即可。例如：

**---**

**- hosts: 192.168.100.59,192.168.100.150**

**tasks:**

**- shell: echo "{{inventory\_hostname}}" >/tmp/hello.txt delegate\_to: localhost**

上⾯的playbook中指定了两个被控节点192.168.100.59和192.168.100.150，但是它们不会执⾏这个任务，⽽是在ansible的本地执⾏。但注意，这个任务在本地会执⾏两次，因为有两个被控节点，且变

量"inventory\_hostname"的值不是localhost，⽽是被控节点的主机名 。要想让某个任务只运⾏⼀次，可以使

⽤run\_once选项，见下⽂。

如果要将任务分配给多个主机，可以使⽤with\_items的⽅式进⾏迭代。例如：

**---**

**- hosts: 192.168.100.59,192.168.100.150**

**tasks:**

**- shell: echo "{{inventory\_hostname}}" >/tmp/hello.txt delegate\_to: "{{item}}"**

**with\_items:**

**- localhost**

**- 192.168.100.66**

如果是分配给ansible本地端执⾏任务，则还有⼀个更简介的⽅式：使⽤local\_action。例如，下⾯两个task是等价 的。

**tasks:**

* **shell: echo haha >>/tmp/hello.txt delegate\_to: localhost**
* **local\_action: shell echo haha >>/tmp/hello.txt**

实际上，local\_act ion的⽅式是延续了⽼版本ansible的action组件。在以前的版本中，任何⼀个模块都可以使⽤action来指定。例如下⾯两个task是等价的。

**tasks:**

* **shell: echo haha >>/tmp/hello.txt**
* **action: shell echo haha >>/tmp/hello.txt**

## 只运⾏⼀次run\_once

使⽤delegate\_to或local\_action都可以分配任务给指定主机，但如果被控节点有多台时，这些分配后的任务会多次 执⾏。再例如，在备份某个数据库时，由于主从复制结构下的可能会有多台数据库服务器，但实际上只需备份其中

⼀台就够了，这时可以指定run\_once，让任务仅只执⾏⼀次。

**---**

**- hosts: mysql\_rep**

**tasks:**

**- name: backup mysql data shell: mysqldump ......**

**run\_once: True**

既然是只执⾏⼀次任务，那么肯定会挑其中⼀台被控节点。挑选时是按照hosts中的顺序挑选的。

## 分批执⾏play

最典型的案例是：修改负载均衡后端服务器时，要先摘除后端server，再修改后重启，最后加⼊到负载server的后 端列表中。默认情况下，ansible会将后端服务器⼀次性全部摘除(假如hosts指定了所有后端主机)，再依次全部修 改、重启、添加会后端列表。这样就会导致修改后端服务器的过程中⽆法向外提供服务，显然这是不合理的。当然，也可以使⽤hosts限定被控主机列表。

ansible⽀持使⽤serial选项实现分批执⾏play的功能。例如：

**---**

**- hosts: 192.168.100.[59:68]**

**serial: 5 tasks:**

* **task1**
* **task2**
* **task3**

在上⾯的⽰例中，指定了10台被控节点，如果不使⽤serial，那么ansible会先控制10台节点上执⾏task1，执⾏完 后再控制这10台节点执⾏task2，同理task3。但上⾯指定了serial，那么ansible将先控制5台主机执⾏task1、

task2、task3，执⾏完后再控制5台主机执⾏task1、task2、task3，直到所有主机都执⾏结束。

这和ansible的"- - f orks"选项的并⾏不⼀样，"- - f orks"选项指定的并⾏是每个task内分批执⾏，例如默认情况下的"-

- f orks=5"，ansible会先控制5台主机执⾏task1，这5台执⾏结束后再控制另外5台继续执⾏task1，同理task2和task3也如此。

以下⾯的过程来描述serial和"-- f orks"的区别，假如有10台主机0- 9，serial和"- - f orks"都指定为5。

serial的并⾏⽅式：

01234：task1、task2、task3 56789：task1、task2、task3

"- - f orks"的并⾏⽅式:

01234：task1 56789：task1 01234：task2 56789：task2 01234：task3 56789：task3

回到serial。serial的指定⽅式有多种，还⽀持百分⽐、列表的形式。例如：

**---**

**- hosts: host[1:10]**

**serial: 30%**

表⽰每批有hosts的10分之3主机执⾏play。

**---**

**- hosts: host[1:10]**

**serial:**

**- 1**

**- 3**

**- 60%**

表⽰第⼀批只有⼀台主机执⾏play，第2批有3台主机执⾏play，第三批有60%的主机执⾏play。

## 分批执⾏play时的最⼤失败百分⽐

假如所有负载均衡的后端server做为被控节点，且所有被控节点执⾏任务都失败，那么即使分批执⾏play，最终的 结果也依然是⽆法提供服务

ansible也考虑到了这种情况。可以使⽤max\_fail\_percentage 指定最⼤失败百分⽐，例如指定30%，那么分批执⾏时，如果有30%的主机执⾏play失败时，⾃动退出该play(不是退出ansibel，⽽是play)。

**---**

**- hosts: 192.168.100.[59:68]**

**serial: 5**

**max\_fail\_percentage: 30% tasks:**

* **task1**
* **task2**
* **task3**

## 错误处理——忽略错误

Ansible默认会检查命令和模块的返回状态，并进⾏相应的错误处理，默认是遇到错误就中断playbook的执⾏， 这些默认⾏为都是可以改变的。

对于command和shell模块，如果命令返回⾮零状态码，则ansible判定这2个模块执⾏失败，可以通过ignore\_errors忽略返回状态码。如下：

**- name: this will not be counted as a failure command: /bin/false**

**ignore\_errors: yes**

## 错误处理——⾃定义错误判断条件

命令不依赖返回状态码来判定是否执⾏失败，⽽是要查看命令返回内容来决定，⽐如返回内容中包括FAILED字符 串，则判定为失败。⽰例如下：

**- name: command returns FAILED when it fails command: /usr/bin/test\_cmd**

**register: rst**

**failed\_when: "'FAILED' in rst.stderr"**

此外ansible会⾃动判断模块执⾏状态，也就是changed=True/False，但可以⾃定义达到changed=True的条件，

⽰例如下：

* **name: copy in nginx conf**

**template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf**

* **name: validate nginx conf**

**shell: "/usr/local/nginx/sbin/nginx -t" register: command\_result**

**changed\_when: command\_result.stdout.find('successful')**

上⾯的例⼦中，当shell模块的命令返回中有"successf ul"字符串时，则为changed=True状态，下⾯这个设定将永远也不会达到changed=True状态。

**- name: validate nginx conf**

**shell: "/usr/local/nginx/sbin/nginx -t" changed\_when: false**

# playbook⽰例： 编译安装httpd

以下是playbook的内容。它的处理流程是：

1. **先在本地下载apr,apr- util,httpd共3个.tar.gz⽂件。**
2. **解压这3个⽂件。**
3. **安装pcre和pcre- devel依赖包。4. 编译安装apr。**
4. **编译安装apr- util。**
5. **编译安装httpd。**

**---**

* **hosts: all**

**tasks:**

* + **name: download apr,apr-util,httpd get\_url: url="{{item}}" dest=/root/pkg with\_items:**
    - **https://mirrors.tuna.tsinghua.edu.cn/apache/apr/apr-1.6.2.tar.gz**
    - **https://mirrors.tuna.tsinghua.edu.cn/apache/apr/apr-util-1.6.0.tar.gz**
    - **https://mirrors.tuna.tsinghua.edu.cn/apache/httpd/httpd-2.4.27.tar.gz delegate\_to: localhost**

**run\_once: True**

* + **unarchive: src="/root/pkg/{{item}}" dest=/root/ with\_items:**
    - **httpd-2.4.27.tar.gz**
    - **apr-1.6.2.tar.gz**
    - **apr-util-1.6.0.tar.gz tags: unarchive**
  + **name: install pcre and pcre-devel and expat-devel yum: name="{{item}}" state=installed**

**with\_items:**

* + - **pcre**
    - **pcre-devel**
    - **expat-devel**
  + **name: complie apr**

**shell: cd /root/apr-1.6.2 && ./configure --prefix=/usr/local/apr && make && make install**

* + **name: complie apr-util shell: |**

**cd /root/apr-util-1.6.0**

**./configure --prefix=/usr/local/apr-util --with-apr=/usr/local/apr make && make install**

* + **name: complie httpd shell: |**

**cd /root/httpd-2.4.27**

**./configure --prefix=/usr/local/apache --sysconfdir=/etc/apache \**

**--enable-mpms-shared=all \**

**--with-z --with-pcre \**

**--with-apr=/usr/local/apr \**

**--with-apr-util=/usr/local/apr-util \**

**--with-mpm=event make && make install**

编译完成后，还有⼀系列操作，⽐如设置PAT H环境变量、设置man路径、修改配置⽂件、启动httpd等。这些就懒得放进去了。