

## 一. Ansible 介绍

### 常用的自动化运维工具

#### Puppet

—基于 Ruby 开发，采用 C/S 架构，扩展性强，基于 SSL，远程命令执行相对较弱

#### SaltStack

—基于 Python 开发，采用 C/S 架构，相对 puppet 更轻量级，配置语法使用 YMAL，使得配置脚本更简单

#### Ansible

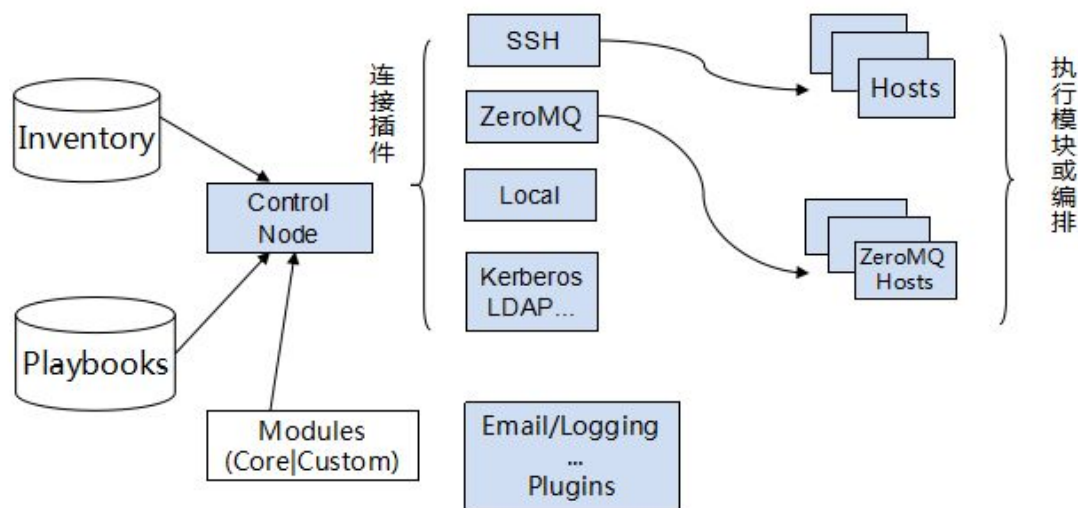
—基于 Python paramiko 开发，分布式，无需客户端，轻量级，配置语法使用 YMAL 及 Jinja2 模板语言，更强的远程命令执行操作

其他 DevOps 请参看：<https://github.com/geekwolf/sa-scripts/blob/master/devops.md>

Ansible 是一个简单的自动化运维管理工具，可以用来自动化部署应用、配置、编排 task(持续交付、无宕机更新等)，采用 paramiko 协议库（fabric 也使用这个），通过 SSH 或者 ZeroMQ 等连接主机，大概每 2 个月发布一个主版本

### Ansible 工作机制

Ansible 在管理节点将 Ansible 模块通过 SSH 协议（或者 Kerberos、LDAP）推送到被管理端执行，执行完之后自动删除，可以使用 SVN 等来管理自定义模块及编排



Ansible工作机制

由上面的图可以看到 Ansible 的组成由 5 个部分组成：

Ansible: 核心

Modules: 包括 Ansible 自带的核心模块及自定义模块

Plugins: 完成模块功能的补充，包括连接插件、邮件插件等

Playbooks: 网上很多翻译为剧本，个人觉得理解为编排更为合理；定义 Ansible 多任务配置文件，有 Ansible 自动执行

Inventory: 定义 Ansible 管理主机的清单

## 二. Ansible 安装

使用环境 Ubuntu 14.04.1 LTS

Control Machine: 192.168.0.4

Managed Nodes0: 192.168.0.5  
192.168.0.6

**注意：**Ansible 默认使用 SSH 协议管理节点

安装要求：

控制服务器：需要安装 Python2.6/2.7

被管理服务器：需要安装 Python2.4 以上版本，若低于 Python2.5 需要安装 python-simplejson; 若启用了 selinux，则需要安装 libselinux-python

方式一：源码安装

```
git clone git://github.com/ansible/ansible.git --recursive
cd ./ansible
source ./hacking/env-setup
sudo easy_install pip
sudo pip install paramiko PyYAML Jinja2 httpplib2
```

Ansible 升级操作：

```
git pull --rebase
git submodule update --init --recursive
```

Git 知识请参考：

<http://blog.csdn.net/huangyabin001/article/details/30100287>

<http://git-scm.com/docs/git-submodule>

执行脚本 env-setup 时会读取默认的主机清单文件/etc/ansible/hosts,主机清单文件路径可以通过环境变量 ANSIBLE\_HOSTS 设置或者执行 ansible 时-i 参数指定  
最后可以执行 sudo make install 将 ansible 安装到 control server

**问题 1：**

```
x86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -fwrapv -Wall -Wstrict-prototypes -fPIC -std=c99 -O3 -fomit-frame-pointer -Isrc/ -I/usr/include/python2.7 -c src/MD2.c -o build/temp.linux-x86_64-2.7/src/MD2.o
```

src/MD2.c:31:20: fatal error: Python.h: No such file or directory

```
#include "Python.h"
```

^

compilation terminated.

error: command 'x86\_64-linux-gnu-gcc' failed with exit status 1

**解决办法：**

```
sudo apt-get install python-dev
```

方式二：yum 和 apt 安装

针对 RedHat 系列（由于 Ansible 可以管理包含 Python2.4+版本的早期启动，故适用 EL5）：  
安装 EPEL 源

```
rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
sudo yum install ansible
```

或者可以使用源码自己打包 rpm 安装

```
git clone git://github.com/ansible/ansible.git
cd ./ansible$ make rpm
sudo rpm -Uvh ~/rpmbuild/ansible-*.noarch.rpm
```

针对 Ubuntu 系列

添加 apt 源

```
vim /etc/apt/source.list
deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
deb-src http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
```

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

注释：旧的 ubuntu 发行版 software-properties-common 包叫 python-software-properties  
也可以通过源码 make deb 生成 deb 包 apt 去安装

通用方法：使用 pip 或者 pipsi 的 python 包管理工具安装  
sudo pip install ansible

```
curl https://raw.githubusercontent.com/mitsuhiko/pipsi/master/get-pipsi.py | \
python
pipsi install ansible
```

### 三. 开始使用

默认情况下，Ansible1.2 之前的版本，需要加参数-c 显式指定；Ansible1.3+版本默认使用 Openssh 来远程连接主机，并开启 ControlPersist 来优化连接速度和认证；如果使用 RHEL6 系作为 Ansible 控制端，由于 OpenSSH 版本太老无法支持 ControlPersist，这时候 Ansible 将会使用高效的 Python 实现的 OpenSSH 即 paramiko；若果想使用原生的 OpenSSH 连接被控端，需要升级 OpenSSH 到 5.6+或者选择 Ansible 的加速模式

针对不支持 ControlPersist 的系统如 RHEL6 可以通过使用 Ansible 加速模式或者 SSH  
pipelining (*pipelining=on*) 进行优化来提高连接速度

Ansible 加速模式支持 RHEL6 控制端和其他受限的环境

加速模式使用方法：

在定义的 playbooks 里面添加 accelerate: true 参数

```

---

- hosts: all
  accelerate: true

tasks:

- name: some task
  command: echo {{ item }}
  with_items:
    - foo
    - bar
    - baz

```

更改 Ansible 用于加速连接的端口，使用参数 `accelerate_port`，端口配置可以设置系统环境变量 `ACCELERATE_PORT` 或者修改 `ansible.cfg` 配置文件，`accelerate_multi_key` 允许使用多把密钥 [accelerate]

`accelerate_port = 5099`

```

---

- hosts: all
  accelerate: true
  # default port is 5099
  accelerate_port: 10000
  accelerate_multi_key = yes

```

注释：Ansible 加速模式支持 `sudo` 操作，但需要注释 `/etc/sudoers` `#Defaults requiretty;` `sudo` 密码还未支持，所以 `sudo` 执行需要 `NOPASSWD`

官方鼓励使用 SSH Keys，也可以通过参数 `--ask-pass` 及 `--ask-sudo-pass` 使用密码

## 尝鲜 Ansible

### 控制端：

```

vim /etc/ansible/hosts
192.168.0.4
192.168.0.5
192.168.0.6

```

配置无密码登陆，密码认证使用 Ansible 的 `-k` 参数

```

root@geekwolf:~# ssh-keygen
root@geekwolf:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.0.4
root@geekwolf:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.0.5

```

```
root@geekwolf:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.0.6
```

如果创建密钥时设置了密码，ansible 不会自动填充私钥密码，可以通过下面的方法自动填充私钥密码：

```
ssh-agent bash

ssh-add ~/.ssh/id_rsa

永久生效,添加代码到.bashrc

if [ -f ~/.agent.env ]; then
    . ~/.agent.env >/dev/null
    if ! kill -s 0 $SSH_AGENT_PID >/dev/null 2>&1; then
        echo "Stale agent file found. Spawning new agent..."
        eval `ssh-agent |tee ~/.agent.env`
        ssh-add
    fielse
    echo "Starting ssh-agent..."
    eval `ssh-agent |tee ~/.agent.env`
    ssh-add
fi
```

测试 ping:

```
root@geekwolf:~# ansible all -m ping
192.168.0.4 | success >> {
  "changed": false,
  "ping": "pong"
}

192.168.0.6 | success >> {
  "changed": false,
  "ping": "pong"
}

192.168.0.5 | success >> {
  "changed": false,
  "ping": "pong"
}
```

其他事例:

以 bruce 身份 ping 所有主机

```
ansible all -m ping -u bruce
```

用 bruce 用户以 root 身份 ping

```
ansible all -m ping -u bruce --sudo
```

用 bruce 用户 sudo 到 batman 用户 ping  
 ansible all -m ping -u bruce --sudo --sudo-user batman

在所有节点上执行命令  
 ansible all -a "/bin/echo hello"

### Ansible 命令参数介绍

Usage: ansible <host-pattern> [options]

Options:

-m MODULE\_NAME, --module-name=MODULE\_NAME 要执行的模块, 默认为 *comm*  
*and*  
 -a MODULE\_ARGS, --args=MODULE\_ARGS 模块的参数  
 -u REMOTE\_USER, --user=REMOTE\_USER ssh 连接的用户名, 默认用 *root*, *ansible.cfg* 中可以配置  
 -k, --ask-pass 提示输入 ssh 登录密码, 当使用密码验证登录的时候用  
 -s, --sudo sudo 运行  
 -U SUDO\_USER, --sudo-user=SUDO\_USER sudo 到哪个用户, 默认为 *root*  
 -K, --ask-sudo-pass 提示输入 sudo 密码, 当不是 *NOPASSWD* 模式时使用  
 -B SECONDS, --background=SECONDS run asynchronously, failing  
 after X seconds(*default=N/A*)  
 -P POLL\_INTERVAL, --poll=POLL\_INTERVAL set the poll interval if using  
 -B (*default=15*)  
 -C, --check 只是测试一下会改变什么内容, 不会真正去执行  
 -c CONNECTION 连接类型(*default=smart*)  
 -f FORKS, --forks=FORKS fork 多少个进程并发处理, 默认 *5*  
 -i INVENTORY, --inventory-file=INVENTORY 指定 *hosts* 文件路径, 默认 *default*  
*=/etc/ansible/hosts*  
 -l SUBSET, --limit=SUBSET 指定一个 *pattern*, 对<*host\_pattern*>已经匹配的主机  
 中再过滤一次  
 --list-hosts 只打印有哪些主机执行这个 *playbook* 文件, 不是实际执行该 *playboo*  
 -M MODULE\_PATH, --module-path=MODULE\_PATH 要执行的模块的路径, 默认  
 为 */usr/share/ansible/*  
 -o, --one-line 压缩输出, 摘要输出  
 --private-key=PRIVATE\_KEY\_FILE 私钥路径  
 -T TIMEOUT, --timeout=TIMEOUT ssh 连接超时时间, 默认 *10* 秒  
 -t TREE, --tree=TREE 日志输出到该目录, 日志文件名会以主机名命名  
 -v, --verbose verbose mode (-vvv for more, -vvvv to enable connection debu  
 gging)

**注释:**

1.在首次连接或者重装系统之后会出现检查 keys 的提示

The authenticity of host '192.168.0.5 (192.168.0.5)' can't be established.

ECDSA key fingerprint is 05:51:e5:c4:d4:66:9b:af:5b:c9:ba:e9:e6:a4:2b:fe.

Are you sure you want to continue connecting (yes/no)?

解决办法:

vim /etc/ansible/ansible.cfg 或者 ~/.ansible.cfg

[defaults]

host\_key\_checking = False

也可以通过设置系统环境变量来禁止这样的提示

export ANSIBLE\_HOST\_KEY\_CHECKING=False

2.在使用 paramiko 模式时, 主机 keys 的检查会很慢

3.默认情况下 Ansible 会记录一些模块的参数等信息到每个被控端的 syslog 日志文件里, 除非在任务或者剧本里设置了 no\_log: True 会不记录日志

**四. 主机清单 Inventory**

Ansible 通过读取默认的主机清单配置/etc/ansible/hosts,可以同时连接到多个远程主机上执行任务,默认路径可以通过修改 ansible.cfg 的 hostfile 参数指定路径  
/etc/ansible/hosts 主机清单配置格式如下

```
mail.example.com[webservers]
Foo.example.com
Bar.example.com
[dbservers]
One.example.com
Two.example.com
three.example.com
badwolf.example.com:5309           指定 SSH 端口 5309
jumper ansible_ssh_port=5555 ansible_ssh_host=192.168.1.50  设置主机别名为 jumper
www[01:50].example.com           支持通配符匹配 www01 www02 ...www50
[databases]
db-[a:f].example.com             支持字母匹配 a b c...f
```

为每个主机指定连接类型和连接用户:

```
[targets]
localhost          ansible_connection=local
other1.example.com  ansible_connection=ssh      ansible_ssh_user=mpdehaan
other2.example.com  ansible_connection=ssh      ansible_ssh_user=mdehaan
```

可以为每个主机单独指定一些变量，这些变量随后可以在 playbooks 中使用：

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

也可以为一个组指定变量，组内每个主机都可以使用该变量：

```
[atlanta]
host1
host2
[atlanta:vars]

ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

组可以包含其他组：

```
[atlanta]
host1
host2
[raleigh]
host2
host3
[southeast:children]
atlanta
raleigh
[southeast:vars]

some_server=foo.southeast.example.com

halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```



[ ] 表示主机的分组名,可以按照功能、系统等进行分类,便于对某些主机或者某一组功能相同的主机进行操作

为 host 和 group 定义一些比较复杂的变量时(如 array、hash),可以用单独文件保存 host 和 group 变量,以 **YAML** 格式书写变量,避免都写在 hosts 文件显得混乱,如 hosts 文件路径为:

```
/etc/ansible/hosts
```

则 host 和 group 变量目录结构:

/etc/ansible/host_vars/all	#host_vars 目录用于存放 host 变量, all 文件对所有主机有效
/etc/ansible/group_vars/all	#group_vars 目录用于存放 group 变量, all 文件对所有组有效
/etc/ansible/host_vars/foosball	#文件 foosball 要和 hosts 里面定义的主机名一样,表示只对 foosball 主机有效
/etc/ansible/group_vars/raleigh	#文件 raleigh 要和 hosts 里面定义的组名一样,表示对 raleigh 组下的所有主机有效

这里/etc/ansible/group\_vars/raleigh 格式如下:

```
---  #YAML 格式要求

ntp_server: acme.example.org           #变量名:变量值

database_server: storage.example.org
```

**注意:** 官方建议将/etc/ansible 目录使用 git/svn 来进行版本控制便于跟踪和修改

hosts 文件支持一些特定指令,上面已经使用了其中几个,所有支持的指令如下:

```
ansible_ssh_host: 指定主机别名对应的真实 IP, 如: 251 ansible_ssh_host=183.60.41.251,
随后连接该主机无须指定完整 IP, 只需指定 251 就行
ansible_ssh_port: 指定连接到这个主机的 ssh 端口, 默认 22
ansible_ssh_user: 连接到该主机的 ssh 用户
ansible_ssh_pass: 连接到该主机的 ssh 密码(连-k 选项都省了), 安全考虑还是建议使用私钥
或在命令行指定-k 选项输入
ansible_sudo_pass: sudo 密码

ansible_sudo_exe(v1.8+的新特性):sudo 命令路径
ansible_connection: 连接类型, 可以是 local、ssh 或 paramiko, ansible1.2 之前默认为 paramiko
ansible_ssh_private_key_file: 私钥文件路径
ansible_shell_type: 目标系统的 shell 类型, 默认为 sh, 如果设置 csh/fish, 那么命令需要遵循它们语法
ansible_python_interpreter: python 解释器路径, 默认是/usr/bin/python, 但是如要连*BSD 系统的话, 就需要该指令修改 python 路径
```

`ansible_*_interpreter`: 这里的“\*”可以是 ruby 或 perl 或其他语言的解释器, 作用和 `ansible_python_interpreter` 类似

例子:

```
some_host      ansible_ssh_port=2222      ansible_ssh_user=manager
aws_host       ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
freebsd_host   ansible_python_interpreter=/usr/local/bin/python
ruby_module_host ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
```

动态获取主机及引用相关参数可以参考, 后续的应用一章会详细介绍

[http://docs.ansible.com/intro\\_dynamic\\_inventory.html](http://docs.ansible.com/intro_dynamic_inventory.html)

<http://rfviamcool.blog.51cto.com/1030776/1416808>

## 五. 通配模式 Patterns

在 Ansible 中, Patterns 意味着要管理哪些机器, 在 playbooks 中, 意味着哪些主机需要应用特定的配置或者过程

命令格式 (下一章介绍 Ansible 的命令具体使用):

`ansible <pattern_goes_here> -m <module_name> -a <arguments>`

比如我们的主机列表配置为:

192.168.0.6

[webservers]

192.168.0.4

[db]

192.168.0.5

`ansible webservers -m service -a "name=httpd state=restarted"`

模式通常用主机组来表示, 上面的命令就代表 webservers 组的所有主机

其他的匹配方式:

表示通配 inventory 中的所有主机

all

\*

也可以指定具有规则特征的主机或者主机名

one.example.com

one.example.com:two.example.com

192.168.1.50

192.168.1.\*

下面的模式, 用来知道一个地址或多个组。组名之间通过冒号隔开, 表示“OR”的意思, 意思是这两个组中的所有主机

webservers

webservers:dbservers

非模式匹配: 表示在 webservers 组不在 phoenix 组的主机

webservers:!phoenix

交集匹配: 表示同时都在 webservers 和 staging 组的主机

webservers:&staging

组合匹配: 在 `webservers` 或者 `dbservers` 组中, 必须还存在于 `staging` 组中, 但是不在 `phoenix` 组中  
`webservers:dbservers:&staging:!phoenix`

在 `ansible-palybook` 命令中, 你也可以使用变量来组成这样的表达式, 但是你必须使用“-e”的选项来指定这个表达式

`webservers:!{{excluded}}:&{{required}}`

你完全不需要使用这些严格的模式去定义组来管理你的机器, 主机名, IP, 组都可以使用通配符去匹配

`*.example.com`

`*.com`

`one*.com:dbservers`

可以匹配一个组的特定编号的主机 (先后顺序 0 到...)

`webservers1[0]` 表示匹配 `webservers1` 组的第 1 个主机

`webservers1[0:25]` 表示匹配 `webservers1` 组的第 1 个到第 25 个主机 (官网文档是“-”表示范围, 测试发现应该使用“-”, 注意不要和匹配多个主机组混淆)

在开头的地方使用 “~”, 表示这是一个正则表达式

`~(web|db).*\.example\.com`

在 `/usr/bin/ansible` 和 `/usr/bin/ansible-playbook` 中, 还可以通过一个参数“-limit”来明确指定排除某些主机或组

`ansible-playbook site.yml --limit datacenter2`

从 Ansible1.2 开始, 如果想要排除一个文件中的主机可以使用“-@”

`ansible-playbook site.yml --limit @retry_hosts.txt`

## 六. Ansible 常用模块的操作

以下是如何使用 `/usr/bin/ansible` 运行一些临时任务的例子, 比如关机、重启服务等并不需要写剧本 `playbooks`, 这时使用 `ansible` 的一行命令或者程序就能方便解决, 对于配置管理及应用部署使用 `playbooks` 将是不错的选择:

### 并行性和 shell 命令

重启 `atlanta` 主机组的所有机器, 每次重启 10 台

`ansible atlanta -a "/sbin/reboot" -f 10`

以 `geekwolf` 用户身份在 `atlanta` 组的所有主机运行 `foo` 命令

`ansible atlanta -a "/usr/bin/foo" -u geekwolf`

以 `geekwolf` 用户身份 `sudo` 执行命令 `foo` (--ask-sudo-pass (-K) 如果有 `sudo` 密码请使用此参数)

`ansible atlanta -a "/usr/bin/foo" -u geekwolf --sudo [--ask-sudo-pass]`

也可以 `sudo` 到其他用户执行命令非 `root`

`ansible atlanta -a "/usr/bin/foo" -u username -U otheruser [--ask-sudo-pass]`

默认情况下, `ansible` 使用的 module 是 `command`, 这个模块并不支持 `shell` 变量和管道等, 若想使用 `shell` 来执行模块, 请使用 `-m` 参数指定 `shell` 模块

使用 shell 模块在远程主机执行命令  
`ansible raleigh -m shell -a 'echo $TERM'`

### 传输文件

拷贝本地的/etc/hosts 文件到 atlanta 主机组所有主机的/tmp/hosts（空目录除外）,如果使用 playbooks 则可以充分利用 template 模块

```
ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

file 模块允许更改文件的用户及权限

```
ansible webserver -m file -a "dest=/srv/foo/a.txt mode=600"
```

```
ansible webserver -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"
```

使用 file 模块创建目录, 类似 `mkdir -p`

```
ansible webserver -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan state=directory"
```

使用 file 模块删除文件或者目录

```
ansible webserver -m file -a "dest=/path/to/c state=absent"
```

### 管理软件包

apt、yum 模块分表用于管理 Ubuntu 系列和 RedHat 系列系统软件包

确保 acme 包已经安装, 但不更新

```
ansible webserver -m apt -a "name=acme state=present"
```

确保安装包到一个特定的版本

```
ansible webserver -m apt -a "name=acme-1.5 state=present"
```

确保一个软件包是最新版本

```
ansible webserver -m apt -a "name=acme state=latest"
```

确保一个软件包没有被安装

```
ansible webserver -m apt -a "name=acme state=absent"
```

**注释:** Ansible 支持很多操作系统的软件包管理, 使用时 `-m` 指定相应的软件包管理工具模块, 如果没有这样的模块, 可以自己定义类似的模块或者使用 `command` 模块来安装软件包

### 用户和用户组

使用 user 模块对于创建新用户和更改、删除已存在用户非常方便

```
ansible all -m user -a "name=foo password=<crypt password here>"
```

```
ansible all -m user -a "name=foo state=absent"
```

生成加密密码方法

```
1.mkpasswd --method=SHA-512
```

```
2.pip install passlib
```

```
python -c "from passlib.hash import sha512_crypt; import getpass; print sha512_crypt.encrypt(getpass.getpass())"
```

其他的模块参数请参考

[http://docs.ansible.com/user\\_module.html#examples](http://docs.ansible.com/user_module.html#examples)

[http://docs.ansible.com/modules\\_by\\_category.html](http://docs.ansible.com/modules_by_category.html)

## 源码部署

Ansible 模块能够通知变更，当代码更新时，可以告诉 Ansible 做一些特定的任务，比如从 git 部署代码然后重启 apache 服务等

```
ansible webserver2 -m git -a "repo=https://github.com/Icinga/icinga2.git    dest=/tmp/myapp  
version=HEAD"
```

## 服务管理

确保 webserver 组所有主机的 httpd 是启动的

```
ansible webserver -m service -a "name=httpd state=started"
```

重启 webserver 组所有主机的 httpd 服务

```
ansible webserver -m service -a "name=httpd state=restarted"
```

确保 webserver 组所有主机的 httpd 是关闭的

```
ansible webserver -m service -a "name=httpd state=stopped"
```

## 后台运行

长时间运行的操作可以放到后台执行，ansible 会检查任务的状态；在主机上执行的同一个任务会分配同一个 job ID

后台执行命令 3600s，-B 表示后台执行的时间

```
ansible all -B 3600 -a "/usr/bin/long_running_operation --do-stuff"
```

检查任务的状态

```
ansible all -m async_status -a "jid=123456789"
```

后台执行命令最大时间是 1800s 即 30 分钟，-P 每 60s 检查下状态默认 15s

```
ansible all -B 1800 -P 60 -a "/usr/bin/long_running_operation --do-stuff"
```

## 搜集系统信息

Facts 在 playbooks 一章中会有详细的描述，这里只通过命令获取所有的系统信息

搜集主机的所有系统信息

```
ansible all -m setup
```

搜集系统信息并以主机名为文件名分别保存在/tmp/facts 目录

```
ansible all -m setup --tree /tmp/facts
```

搜集和内存相关的信息

```
ansible all -m setup -a 'filter=ansible_*_mb'
```

搜集网卡信息

```
ansible all -m setup -a 'filter=ansible_eth[0-2]'
```

详细使用参考

[http://docs.ansible.com/setup\\_module.html](http://docs.ansible.com/setup_module.html)

[http://docs.ansible.com/playbooks\\_variables.html](http://docs.ansible.com/playbooks_variables.html)

## 七. Ansible 配置参数

请参考: [http://docs.ansible.com/intro\\_configuration.html](http://docs.ansible.com/intro_configuration.html)

## 八. Ansible 对 Windows 的支持

### 管理 Windows 测试

从上面的学习中知道 Ansible 管理 Linux 主机是通过 SSH 方式, 从 Ansible1.7 版本开始, 开始支持对 Windows 系统的管理, 使用 powershell 进行远程连接; 前提是需要先在 Ansible 控制端安装 winrm 的 Python 模块和远程主机通信

Ansible 控制端:

```
pip install http://github.com/diyan/pywinrm/archive/master.zip#egg=pywinrm
```

```
vim /etc/ansible/hosts
```

```
[win2]
```

```
192.168.0.8
```

```
vim /etc/ansible/group_vars/win2.yml
```

```
# it is suggested that these be encrypted with ansible-vault:
```

```
# ansible-vault edit group_vars/windows.yml
```

```
ansible_ssh_user: Administrator
```

```
ansible_ssh_pass: 3KRxuSezVdx5
```

```
ansible_ssh_port: 5986
```

```
ansible_connection: winrm
```

**注意:** 如果是使用 git 方式获取安装, 需要更改 ansible 的模块路径参数

```
vim /etc/ansible/ansible.cfg
```

```
library = /root/ansible/lib/ansible/modules
```

```
默认路径是/usr/share/ansible
```

Windows server 2012 R2:

确保系统 powershell 版本在 3.0 以上 (以管理员身份运行 powershell, 执行脚本 upgrade\_to\_ps3.ps1 升级 powershell2.0 到 powershell3.0; 版本号可以通过在 powershell 中执行 \$host.version 命令查看)

[https://github.com/cchurch/ansible/blob/devel/examples/scripts/upgrade\\_to\\_ps3.ps1](https://github.com/cchurch/ansible/blob/devel/examples/scripts/upgrade_to_ps3.ps1)

运行 powershell 脚本自动配置 WinRM

<https://github.com/ansible/ansible/blob/devel/examples/scripts/ConfigureRemotingForAnsible.ps1>

**注意:**

1. 为安全考虑, 需要将端口 5985(HTTP)、5986(HTTPS) 受限给 Ansible 控制端访问

2. 默认情况下 windows server 2012 中 powershell 的脚本执行策略是 remotesigned, windows server 2008 的策略是 Restricted, 需要更改策略后才能执行上面两个脚本

```
get-executionpolicy
```

```
set-executionpolicy remotesigned
```

**问题 1:**

```
root@instance-jb53h7st:~# ansible win -m win_ping
192.168.0.7 | FAILED => Traceback (most recent call last):
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/__init__.py", line 588, in _executor
    exec_rc = self._executor_internal(host, new_stdin)
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/__init__.py", line 767, in _executor_internal
    return self._executor_internal_inner(host, self.module_name, self.module_args,
inject, port, complex_args=complex_args)
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/__init__.py", line 930, in _executor_internal_inner
    conn = self.connector.connect(actual_host, actual_port, actual_user, actual_pass,
actual_transport, actual_private_key_file)
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/connection.py", line 51, in connect
    self.active = conn.connect()
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/connection_plugins/winrm.py", line 132, in connect
    self.protocol = self._winrm_connect()
  File "/usr/local/lib/python2.7/dist-packages/ansible-1.8-
py2.7.egg/ansible/runner/connection_plugins/winrm.py", line 86, in _winrm_connect
    protocol.send_message('')
  File "/usr/local/lib/python2.7/dist-packages/winrm/protocol.py", line 190, in
send_message
    return self.transport.send_message(message)
  File "/usr/local/lib/python2.7/dist-packages/winrm/transport.py", line 82, in
send_message
    response = urlopen(request, timeout=self.timeout)
  File "/usr/lib/python2.7/urllib2.py", line 127, in urlopen
    return _opener.open(url, data, timeout)
  File "/usr/lib/python2.7/urllib2.py", line 404, in open
    response = self._open(req, data)
  File "/usr/lib/python2.7/urllib2.py", line 422, in _open
    '_open', req)
  File "/usr/lib/python2.7/urllib2.py", line 382, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.7/urllib2.py", line 1222, in https_open
    return self.do_open(httplib.HTTPSConnection, req)
  File "/usr/lib/python2.7/urllib2.py", line 1187, in do_open
    r = h.getresponse(buffering=True)
  File "/usr/lib/python2.7/httplib.py", line 1045, in getresponse
    response.begin()
  File "/usr/lib/python2.7/httplib.py", line 409, in begin
    version, status, reason = self._read_status()
  File "/usr/lib/python2.7/httplib.py", line 365, in _read_status
    line = self.fp.readline(_MAXLINE + 1)
  File "/usr/lib/python2.7/socket.py", line 476, in readline
    data = self._sock.recv(self._rbufsize)
```

```
File "/usr/lib/python2.7/ssl.py", line 341, in recv
    return self.read(buflen)
File "/usr/lib/python2.7/ssl.py", line 260, in read
    return self._sslobj.read(len)
SSL error: [Errno 1] _ssl.c:1429: error:14094438:SSL routines:SSL3_READ_BYTES:tlsv1
alert internal error
```

**答案:**

1. 确保 windows 被控端使用了 powershell3.0+以上, 并且 winrm 服务正确配置:

查看 winrm 状态

```
winrm enumerate winrm/config/listener
```

针对 winrm 服务进行配置

```
winrm quickconfig
```

查看 winrm service listener

```
winrm e winrm/config/listener
```

或者在 powershell 中执行

Enable-PSRemoting 进行配置,

Test-Wsman -Computername Geekwolf-PC 检查是否正常

2. Ansible 使用 pywinrm 模块通过 HTTPS 的方式和被控端通信

请确保 ansible\_ssh\_port: 5986 的端口是 https: 5986 而不是 windows 的远程端口 3389

**支持 Windows 的模块**

Ansible 支持的 windows 模块列表:

[http://docs.ansible.com/list\\_of\\_windows\\_modules.html](http://docs.ansible.com/list_of_windows_modules.html)

win\_feature: 安装和卸载功能

win\_get\_url: 从给定的 url 下载文件

win\_group: 添加和删除本地组

win\_msi: 安装和卸载 MSI 文件

win\_ping: windows 版本的 ping 模块

win\_service: 管理 windows 服务

win\_stat: 返回关于 windows 文件的信息

win\_user: 管理本地账号

**注释:** slurp、raw、setup、script、fetch 模块同样支持 windows

例子:

添加用户 bob 及密码

```
root@instance-jb53h7st:~# ansible win2 -m win_user -a "name=bob password=Password12345"
```

```
192.168.0.8 | success >> {
  "changed": true,
  "user_fullname": [
    "bob"
  ],
  "user_name": [
    "bob"
  ],
  "user_path": "WinNT://WORKGROUP/INSTANCE-MPSQMM/bob"
```



```

192.168.0.8 | success >> {
  "ansible_facts": {
    "ansible_distribution": "Microsoft Windows NT 6.3.9600.0",
    "ansible_distribution_version": "6.3.9600.0",
    "ansible_fqdn": "instance-mpsqqmm",
    "ansible_hostname": "INSTANCE-MPSQMM",
    "ansible_interfaces": [
      {
        "default_gateway": "192.168.0.1",
        "dns_domain": "dhcp_domain",
        "interface_index": 12,
        "interface_name": "Red Hat VirtIO Ethernet Adapter"
      }
    ],
    "ansible_ip_addresses": [
      "192.168.0.8",
      "fe80::297b:cc63:8b05:5bab"
    ],
    "ansible_os_family": "Windows",
    "ansible_powershell_version": 4,
    "ansible_system": "Win32NT",
    "ansible_totalmem": 2147483648,
    "ansible_winrm_certificate_expires": "2015-11-18 18:53:26"
  },
  "changed": false
}

```

**注释：**针对 windows 系统的管理我们可能不需要使用 Ansible 提供的甚至自己编写相关的模块（后续会专门讲模块的编写），通常使用 Ansible 的 script 模块来运行 powershell 脚本来解决问题，这对于 windows 管理员更为熟悉：  
比如可以编写下面的 playbook:

```

- hosts: windows
  tasks:
    - script: foo.ps1 --argument --other-argument

```

推送并运行 powershell 脚本

```

- name: test script module
  hosts: windows
  tasks:
    - name: run test script
      script: files/test_script.ps1

```

使用 raw 模块运行原始命令如同在 Linux/Unix 环境中使用 shell 模块、command 模块一样

```

- name: test raw module
  hosts: windows
  tasks:
    - name: run ipconfig
      raw: ipconfig
      register: ipconfig
    - debug: var=ipconfig

```

使用 win\_stat 模块来检查文件或目录是否存在，返回的数据与 Linux 系统返回的数据略有不同

```
- name: test stat module
  hosts: windows
  tasks:
    - name: test stat module on file
      win_stat: path="C:/Windows/win.ini"
      register: stat_file

    - debug: var=stat_file

    - name: check stat_file result
      assert:
        that:
          - "stat_file.stat.exists"
          - "not stat_file.stat.isdir"
          - "stat_file.stat.size > 0"
          - "stat_file.stat.md5"
```

## 九. Playbooks 详解

Playbooks 是 Ansible 管理配置、部署应用和编排的语言，可以使用 Playbooks 来描述你想在远程主机执行的策略或者执行的一组步骤过程等

如果说 Ansible 模块是工作中的工具的话，那么 playbooks 就是方案；推荐阅读 ansible-examples 实例：<https://github.com/ansible/ansible-examples>

Playbooks 采用 YAMAL 语法结构，基本的 YAMAL 语法请参考 <http://docs.ansible.com/YAMLSyntax.html>

### Playbooks 组成

Target section

定义将要执行 playbook 的远程主机组

Variable section

定义 playbook 运行时需要使用的变量

Task section

定义将要在远程主机上执行的任务列表

Handler section

定义 task 执行完成以后需要调用的任务

看下面的例子：

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum: pkg=httpd state=latest
    - name: write the apache config file
      template: src=/srv/httpd.j2 dest=/etc/httpd.conf
      notify:
        - restart apache
```

```
- name: ensure apache is running
  service: name=httpd state=started
handlers:
- name: restart apache
  service: name=httpd state=restarted
```

### 主机和用户

在 playbook 中的每一个 play 都可以选择在哪些机器和以什么用户身份完成，hosts 一行可以是一个主机组或者主机也可以是多个，中间以冒号分隔，可以参考前面讲的通配模式；remote\_user 表示执行的用户账号

```
---
- hosts: webservers
  remote_user: root
```

每一个任务都可以定义一个用户

```
---
- hosts: webservers
  remote_user: root
  tasks:
    - name: test connection
      ping:
        remote_user: yourname
```

在 play 中支持 sudo

```
---
- hosts: webservers
  remote_user: yourname
  sudo: yes
```

在一个任务中支持 sudo

```
---
- hosts: webservers
  remote_user: yourname
  tasks:
    - service: name=nginx state=started
      sudo: yes
```

登陆后 sudo 到其他用户执行

```
---
- hosts: webservers
  remote_user: yourname
  sudo: yes
  sudo_user: postgres
```

**注释：**在使用 sudo\_user 切换到非 root 用户时，Ansible 会将模块参数（非密码选项参数）记录到 /tmp 下的一个临时随机文件，命令执行完后会删除；当 sudo 到 root 或者普通用户登陆时并不记录

### 任务列表

每个任务建议定义一个可读性较强的名字即 name，在执行 playbook 时会输出，tasks 的声明格式，在旧版本使用”action:module options”，建议使用”module:options”的格式

下面以 service 模块为例来定义一个任务，service: key=value 参数，请参看模块的详细介绍  
tasks:

- name: make sure apache is running  
service: name=httpd state=running

command 和 shell 模块不需要增加 key

tasks:

- name: disable selinux  
command: /sbin/setenforce 0

command 和 shell 模块关注命令或者脚本执行后返回值，如果命令成功执行返回值不是 0 的情况下，可以使用以下方法

tasks:

- name: run this command and ignore the result  
shell: /usr/bin/somecommand || /bin/true

或者

tasks:

- name: run this command and ignore the result  
shell: /usr/bin/somecommand  
ignore\_errors: True

如果在任务中参数过长可以回车使用空格缩进

---

- hosts: server  
tasks:
  - name: Copy ansible inventory file to client  
copy: src=/etc/ansible/hosts dest=/tmp/hosts  
owner=root group=root mode=0644

## 变量的使用

### 1.如何创建一个有效的变量名

变量名应该由字母、数组和下划线组成，以字母开头

例如：foo\_port、foo5 就是很好的变量名，而 foo-port、foo port、foo.port、12 都是无效的变量名

### 2.在 inventory 中定义变量

第四章已经讲过关于主机清单中的变量如何定义的内容，请参看第四章

### 3.在 playbook 中如何定义变量

- hosts: webservers  
vars:  
http\_port: 80

### 4.从角色和文件包含中定义变量

参考下面：角色和包含 [http://docs.ansible.com/playbooks\\_roles.html#role-default-variables](http://docs.ansible.com/playbooks_roles.html#role-default-variables)

### 5.如何使用变量：Jinja2

Jinja2 是一个被广泛使用的全功能的 Python 模板引擎，它有完整的 unicode 支持,一个可选的集成沙箱的执行环境,支持 BSD 许可

举一个简单的模板例子：

My amp goes to {{ max\_amp\_value }}

或者在 playbook 中使用

```
template: src=foo.cfg.j2 dest={{ remote_install_path }}/foo.cfg
```

在模板中可以获得一个主机的所有变量，实际上也可以获取其他主机的变量

**注释：**Ansible 允许在模板中使用循环和条件判断，但是在 **playbook** 只使用纯粹的 YAML 语法

## 6.Jinja2 过滤器

变量可以通过**过滤器**修改。过滤器与变量用管道符号（`|`）分割，并且也 可以用圆括号传递可选参数。多个过滤器可以链式调用，前一个过滤器的输出会被作为 后一个过滤器的输入。

例如 `{{ name|striptags|title }}` 会移除 `name` 中的所有 HTML 标签并且改写 为标题样式的大小写格式。过滤器接受带圆括号的参数，如同函数调用。这个例子会 把一个列表用逗号连接起来：`{{ list|join(', ') }}`

Jinja2 Filters 在 Ansible 中并不常用，详细可以参考 Jinja2 官网关于 Filters 的讲解  
<http://docs.jinkan.org/docs/jinja2/templates.html#builtin-filters>

下面列举一些在 Ansible 使用较多的 Filters

格式化数据：

```
{{ ansible_devices | join(',') }}
```

过滤器和条件一起使用：

tasks:

```
- shell: /usr/bin/foo
  register: foo_result
  ignore_errors: True

- debug: msg="it failed"
  when: foo_result|failed

# in most cases you'll want a handler, but if you want to do something right now, this is nice
- debug: msg="it changed"
  when: foo_result|changed

- debug: msg="it succeeded"
  when: foo_result|success

- debug: msg="it was skipped"
  when: foo_result|skipped
```

**注释：**register 关键字的作用是将命令执行的结果保存为变量，结果会因为模块不同而不同，在运行 `ansible-playbook` 时增加 `-v` 参数可以看到 results 可能的值；注册变量就如同通过 `setup` 模块获取 facts 一样

比如当注册变量 `foo_result`

```
- hosts: web_servers
```

tasks:

```
- shell: /usr/bin/foo
  register: foo_result
  ignore_errors: True

- shell: /usr/bin/bar
  when: foo_result.rc == 5
```

```
root@instance-jb53h7st:~/geekwolf# ansible-playbook t1.yml -v
```

```
PLAY [192.168.0.4] *****
```

```
GATHERING FACTS *****
```

```
ok: [192.168.0.4]
```

```
TASK: [shell /bin/echo test] *****
```

```
failed: [192.168.0.4] => {"changed": true, "cmd": "/bin/echo test", "delta": "0:00:00.004034", "end":
"2014-11-28 10:27:37.214545", "rc": 127, "start": "2014-11-28 10:27:37.210511", "warnings": []}
stderr: /bin/sh: 1: /bin/echo: not found
...ignoring
```

```
TASK: [test] *****
```

```
changed: [192.168.0.4] => {"changed": true, "checksum":
"28cc27b7ed8d908fce31134ce21cfe903707e637", "dest": "/tmp/p.txt", "gid": 0, "group": "root", "md5sum":
"e72feb3cab584b809fb751f95f4a9555", "mode": "0644", "owner": "root", "size": 31, "src":
"/root/.ansible/tmp/ansible-tmp-1417141657.23-249511894486070/source", "state": "file", "uid": 0}
```

```
TASK: [debug msg="it failed"] *****
```

```
ok: [192.168.0.4] => {
  "msg": "it failed"
}
```

```
TASK: [debug msg="it changed"] *****
```

```
ok: [192.168.0.4] => {
  "msg": "it changed"
}
```

```
TASK: [debug msg="it succeeded"] *****
```

```
skipping: [192.168.0.4]
```

```
TASK: [debug msg="it was skipped"] *****
```

```
skipping: [192.168.0.4]
```

```
PLAY RECAP *****
```

```
192.168.0.4 : ok=5 changed=2 unreachable=0 failed=0
```

通过上面两个例子我们可以看到 task 的执行输出是和 facts 一样的，我们可以通过 Jinja2 模板获取变量值：

```
{{ foo_result.cmd }}
```

```
{{ foo_result.delta }}
```

或者在 task 中直接使用变量

```
foo_result.rc == 127
```

在 Jinja2 中使用未定义的变量：

```
{{ result.cmd|default(5) }}
```

如果 `result.cmd` 变量未定义，默认返回值 5 或者其他数字字符串 `default('hello world!')`，而在执行该 task 时不会抛错出来

使用 `default(omit)` 自动忽略变量和模块参数：

```
root@instance-jb53h7st:~/geekwolf# cat t.yml
```

```
---
```

```
- hosts: 192.168.0.4
  remote_user: root
```

```
tasks:
```

```
- name: touch files with an optional mode
  file: dest={{ item.path }} state=touch mode={{ item.mode|default(omit) }}
  with_items:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
    mode: "0433"
```

```
root@instance-jb53h7st:~/geekwolf# ansible-playbook t.yml -v
```

```
PLAY [192.168.0.4] *****
```

```
GATHERING FACTS *****
```

```
ok: [192.168.0.4]
```

```
TASK: [touch files] *****
```

```
changed: [192.168.0.4] => (item={'path': '/tmp/foo'}) => {"changed": true, "dest": "/tmp/foo", "gid": 0,
"group": "root", "item": {"path": "/tmp/foo"}, "mode": "0644", "owner": "root", "size": 0, "state": "file",
"uid": 0}
```

```
changed: [192.168.0.4] => (item={'path': '/tmp/bar'}) => {"changed": true, "dest": "/tmp/bar", "gid": 0,
"group": "root", "item": {"path": "/tmp/bar"}, "mode": "0644", "owner": "root", "size": 0, "state": "file",
"uid": 0}
```

```
changed: [192.168.0.4] => (item={'path': '/tmp/baz', 'mode': '0433'}) => {"changed": true, "dest":
"/tmp/baz", "gid": 0, "group": "root", "item": {"mode": "0433", "path": "/tmp/baz"}, "mode": "0433",
"owner": "root", "size": 0, "state": "file", "uid": 0}
```

```
PLAY RECAP *****
```

```
192.168.0.4      : ok=2    changed=1    unreachable=0    failed=0
```

上面的例子执行过程只有最后一个文件 `/tmp/baz` 的权限是 444，其他两个文件为系统默认权限，在执行过程中前两个文件并没有设置 `mode` 变量，`default(omit)` 会自动忽略，从而执行过程中不抛错；不增加 `default(omit)` 过滤的结果：

```
root@instance-jb53h7st:~/geekwolf# cat t.yml
```

```
---
```

```
- hosts: 192.168.0.4
  remote_user: root
```

```
tasks:
```

```
- name: touch files with an optional mode
  file: dest={{ item.path }} state=touch mode={{ item.mode }}
  with_items:
```

```
- path: /tmp/foo
- path: /tmp/bar
- path: /tmp/baz
mode: "0444"
```

```
root@instance-jb53h7st:~/geekwolf# ansible-playbook t.yml -v
```

```
PLAY [192.168.0.4] *****
```

```
GATHERING FACTS *****
ok: [192.168.0.4]
```

```
TASK: [touch files] *****
fatal: [192.168.0.4] => One or more undefined variables: 'dict object' has no attribute 'mode'
```

```
FATAL: all hosts have already failed -- aborting
```

```
PLAY RECAP *****
to retry, use: --limit @/root/t.retry
```

```
192.168.0.4          : ok=1    changed=0    unreachable=1    failed=0
```

列表过滤:

过滤出列表最小值 {{ list1 | min }}

```
root@instance-jb53h7st:~/geekwolf# cat t.yml
```

```
---
```

```
- hosts: 192.168.0.4
  remote_user: root
  vars:
    - test: [1,2,3,4,5]
  tasks:
    - name: vars
      template: src=p.j2 dest=/tmp/p.txt
```

```
root@instance-jb53h7st:~/geekwolf# cat p.j2
```

```
{{test|min}}
```

```
root@instance-jb53h7st:~/geekwolf# cat /tmp/p.txt
```

```
1
```

过滤出列表最大值 {{ [3, 4, 2] | max }}

```
---
```

```
- hosts: 192.168.0.4
  remote_user: root
  vars:
    - test: [3, 4, 2]
  tasks:
    - name: vars
      template: src=p.j2 dest=/tmp/{{test|max}}
```

数据集过滤:

对列表唯一过滤 {{ list1 | unique }}

```
{{[1,2,1,2,4] | unique}}
```

输出结果为:[1,2,4]



对两个列表去重合并 `{{ list1 | union(list2) }}`  
`{{ [1,2,4,2,5] | union([2,5,33,2]) }}`  
输出结果为: `[1,2,4,5,33]`

对两个列表做交集 `{{ list1 | intersect(list2) }}`  
`{{ [1,2,4,2,5] | intersect([2,5,33,2]) }}`  
输出结果为: `[2,5]`

找到两个列表差异部分(在 list1 不在 list2 的差异) `{{ list1 | difference(list2) }}`  
`{{ [1,2,4,2,5] | difference([2,5,33,2]) }}`  
输出结果为: `[1,4]`

找到两个列表都互相不在对方列表的部分 `{{ list1 | symmetric_difference(list2) }}`  
`{{ [1,2,4,2,5] | symmetric_difference([2,5,33,2]) }}`  
输出结果为: `[1,4,33]`

版本过滤:

使用 `version_compare` 比较版本

`{{ ansible_distribution_version | version_compare('12.04', '>=') }}`

如果系统版本大于等于 12.04, 那么返回 True, 否则 False

`version_compare` 支持以下操作符:

`<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne`

随机数过滤:

从列表中随机获取元素

`{{ ['a','b','c','d','e','f'] | random }}`

从 0-59 的整数中随机获取一个数

`{{ 59 | random }}`

从 0-100 中随机获取能被 10 整除的数 (可以理解为 0 10 20 30 40 50 ...100 的随机数)

`{{ 100 | random(step=10) }}`

从 0-100 中随机获取 1 开始步长为 10 的数 (可以理解为 1 11 21 31 41...91 的随机数)

`{{ 100 | random(1, 10) }}`

`{{ 100 | random(start=1, step=10) }}`

随机列表过滤:

给已存在的列表随机排序

`{{ ['a','b','c'] | shuffle }}` => `['c','a','b']`

`{{ ['a','b','c'] | shuffle }}` => `['b','c','a']`

其他有用的过滤器:

列表转换成字符串

`{{ [1,2,4,5,6] | join("-") }}`

输出: `1-2-4-5-6`

获取文件路径最后文件或者目录的名字

`{{ '/etc/asdf/foo.txt' | basename }}`

输出: `foo.txt`

获取文件路径最后一层目录的路径

```
{{ '/root/geekwolf/t.yml' |dirname }}
```

输出: /root/geekwolf

**注释:** 在使用过程中要注意如果是目录后面一定不要忘记加/,否则会认为是文件

Base64 加解密

```
{{ encoded | b64decode }}
{{ decoded | b64encode }}
```

对文件做 sha1 加密

```
{{ filename | sha1 }}
```

布尔值转换[测试发现不需要 bool 转换, 变量复制 True 或者 False 能够识别是布尔值]

```
- debug: msg=test
  when: some_string_value | bool
```

使用 match(需要一个完整的字符串匹配)或者 search (需要部分匹配的字符串) 来正则匹配字符串

---

```
- hosts: 192.168.0.4
  remote_user: root
  vars:
    url: "http://simlinux.com/users/foo/resources/bar"

  tasks:
    - debug: "msg='matched pattern 1'"
      when: url | match("http://simlinux.com/users/*/resources/*")

    - debug: "msg='matched pattern 2'"
      when: url | search("/users/*/resources/*")
```

替换字符串

```
# convert "ansible" to "able"
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}
# convert "foobar" to "bar"
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}
```

## 7.变量文件分离

可以将单个主机或者单个主机组等的变量写到独立的小文件里, 一定程度上降低了分享 playbook 时暴露敏感信息的风险

示例:

---

```
- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml

  tasks:
```

```
- name: this is just a placeholder
  command: /bin/echo foo
```

变量文件就是一个简单的 YAML 格式的字典

```
---
# in the above example, this would be vars/external_vars.yml
somevar: somevalue
password: magic
```

## 8.通过命令行传递变量

通过前面的章节我们了解到可以使用 `var_prompt` 交互方式及 `vars_files` 引用变量文件方式传递变量，下面介绍使用命令行如何传递变量：

```
---
- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'
```

```
tasks:
```

```
- ...
```

比如上面的 playbook，我们可以这样传递 `{{hosts}}`、`{{user}}` 变量：

```
ansible-playbook release.yml --extra-vars "hosts=vipers user=starbuck"
```

在 Ansible1.2 中可以使用 JSON 数据来传递变量

```
--extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
```

在 Ansible1.3 中，可以通过 `@` 导入 JSON 文件来传递变量，也可以使用 YAMAL 格式以命令行或者 `@` 文件的方式导入

```
--extra-vars "@some_file.json"
```

## 9.变量优先级

- \* extra vars(命令中-e)最优先
- \* inventory 主机清单中连接变量(ansible\_ssh\_user 等)
- \* play 中 vars、vars\_files 等
- \* 剩余的在 inventory 中定义的变量
- \* 系统的 facts 变量
- \* 角色定义的默认变量(roles/rolesname/defaults/main.yml)

**注释：**子组会覆盖父组，主机总是覆盖组定义的变量

## 10.变量使用总结

在 playbook 中 action 一行可以使用在 vars 定义的参数

**第一种方法：**定义在 Ansible 中的 `hosts` 和 `groups` 变量中

```
cat /etc/ansible/group_vars/server
```

```
vhost: test
```

```
vim file-book.yml
```

```
---
- hosts: server
  tasks:
```

```
- name: create a virtual host file for {{ vhost }}
  template: src=/etc/hosts dest=/tmp/{{ vhost }}
```

### 第二种方法：在 **playbook** 中指定

```
vim file-book.yml
```

```
---
```

```
- hosts: server
  vars:
    -vhost: test
  tasks:
    - name: create a virtual host file for {{ vhost }}
      template: src=/etc/hosts dest=/tmp/{{ vhost }}
```

### 第三种方法：任意指定变量文件

```
vim /root/variables
```

```
vhost: test
```

```
vim file-book.yml
```

```
---
```

```
- hosts: server
  vars_files:
    - /root/variables
  tasks:
    - name: create a virtual host file for {{ vhost }}
      template: src=/etc/hosts dest=/tmp/{{ vhost }}
```

### 第四种方法：交互方式获取变量值

```
vim file-book.yml
```

```
---
```

```
- hosts: server
  vars_prompt:
    - name: web
      prompt: 'Please input the web server:'
      private: no
  tasks:
    - name: concent of the file test
      template: src=/root/test dest=/tmp/test
```

```
cat /root/test
```

```
{{web}}
```

```

root@instance-jb53h7st:~/geekwolf# ansible-playbook file-book.yml
Please input the web server:: nginx

PLAY [server] *****

GATHERING FACTS *****
ok: [192.168.0.4]
ok: [192.168.0.6]

TASK: [concent of the file test] *****
ok: [192.168.0.6]
ok: [192.168.0.4]

PLAY RECAP *****
192.168.0.4      : ok=2    changed=0    unreachable=0    failed=0
192.168.0.6      : ok=2    changed=0    unreachable=0    failed=0

root@instance-jb53h7st:~/geekwolf# cat /tmp/test
nginx

```

注释: private 设置为 yes 再输入时看不到明文

### 第五种方法: 获取 setup 模块的变量

获取 192.168.0.4 主机的 mac 地址:

```
vim file-book.yml
```

```
---
```

```
- hosts: 192.168.0.4
```

```
  tasks:
```

```
    - name: concent of the file test
```

```
      template: src=/root/test dest=/tmp/test
```

```
cat /root/test
```

```
{{ansible_devices.vda.size}}
```

```
{{ansible_mounts[0].device}}
```

```
{{ ansible_eth0["ipv4"]["address"] }}
```

```
或{{ ansible_eth0.ipv4.address }}
```

默认 playbook 的 setup 模块是开启的, 可以设置 gather\_facts:no 关闭

可以通过在远程主机/etc/ansible/facts.d/目录下创建.fact 结尾的文件, 也可以是 json、ini 或者返回 json 格式数据的可执行文件, 这些将被作为远程主机本地的 facts 执行

示例:

```
/etc/ansible/facts.d/hello.fact
```

```
[test]
```

```
h=hello
```

```
p=world
```

```
root@instance-jb53h7st:/etc/ansible/facts.d# ansible 192.168.0.5 -m setup -a "filter=ansible_local"
```

```

192.168.0.5 | success >> {
  "ansible_facts": {
    "ansible_local": {
      "t": {
        "test": {
          "h": "hello",
          "p": "world"
        }
      }
    }
  }
}

```

```

    },
    "changed": false
}

```

可以通过 `{{ ansible_local.preferences.test.h }}` 方式来使用该变量

下面是创建使用 local facts 的 playbook 示例:

```

- hosts: webservers
  tasks:
    - name: create directory for ansible custom facts
      file: state=directory recurse=yes path=/etc/ansible/facts.d
    - name: install custom impi fact
      copy: src=ipmi.fact dest=/etc/ansible/facts.d
    - name: re-read facts after adding custom fact
      setup: filter=ansible_local

```

获取其他主机 facts:

示例:

```

---
- hosts: test1.simlinux.com:test2.simlinux.com
  remote_user: root
  tasks:
    - name: get other hosts
      shell: "echo {{ hostvars[test2.simlinux.com]['ansible_os_family'] }}"
      when: inventory_hostname == 'test1.simlinux.com'

    - name: get other hosts
      template: src=p.j2 dest=/tmp/p.txt
      when: inventory_hostname == 'test2.simlinux.com'

```

```

cat p.j2
{{ hostvars[test1.simlinux.com]['ansible_distribution'] }}

```

也可以使用 Jinja2 的 if 和 for 语句

```

root@instance-jb53h7st:~/geekwolf# cat t.yml

```

```

---
- hosts: server
  remote_user: root
  tasks:
    - name: get other hosts
      template: src=p.j2 dest=/tmp/p.txt

```

```

root@instance-jb53h7st:~/geekwolf# cat p.j2
{% if 'server' in group_names %}
  {% for host in groups['server'] %}
    {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
  {% endfor %}

{% endif %}

```

```
{% for host in groups['server'] %}
  {{ hostvars[host]['ansible_product_name'] }}
{% endfor %}
```

**注释：**模板中，group\_names 表示 inventory 中所有的主机组名

inventory\_hostname 表示配置在 inventory 清单中的主机名，如果有很长的 FQDN，可以使用 inventory\_hostname\_short，inventory\_dir 表示主机清单 hosts 文件的所在目录，inventory\_file 表示主机清单 hosts 文件绝对路径

play\_hosts  
delegate\_to

## 条件判断与循环

### when 条件判断

示例 1：使用 when 后面使用 Jinja2 表达式

tasks:

```
- name: "shutdown Debian flavored systems"
  command: /sbin/shutdown -t now
  when: ansible_os_family == "Debian"
```

若操作系统是 Debian 时就执行关机操作

示例 2：Jinja2 过滤器同样可以再 when 中使用

tasks:

```
- command: /bin/false
  register: result
  ignore_errors: True
- command: /bin/something
  when: result|failed
- command: /bin/something_else
  when: result|success
- command: /bin/still/something_else
  when: result|skipped
```

示例 3：系统是 RedHat，并且主版本为 6 的系统时执行 shell 模块

tasks:

```
- shell: echo "only on Red Hat 6, derivatives, and later"
  when: ansible_os_family == "RedHat" and ansible_lsb.major_release|int >= 6
```

示例 4：when 与布尔值一起使用做条件判断

vars:

```
epic: true
```

tasks:

```
- shell: echo "This certainly is epic!"
  when: epic
```

tasks:

```
- shell: echo "This certainly isn't epic!"
  when: not epic
```

示例 5: 如果变量没有定义可以使用 Jinja2 中的 defined 来跳过或者抛错误出来

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 not defined

示例 6: when 结合 with\_items 使用

输出 item 大于 5 的数

tasks:

- command: echo {{ item }}  
with\_items: [ 0, 2, 4, 6, 8, 10 ]  
when: item > 5

示例 7: 加载自定义的 facts

首先调用自定义的 facts 搜集模块 site\_facts, 就可以很容易的使用自定义的变量

tasks:

- name: gather site specific fact data  
action: site\_facts
- command: /usr/bin/thingy  
when: my\_custom\_fact\_just\_retrieved\_from\_the\_remote\_system == '1234'

示例 8: 在 roles 和 includes 中使用 when

- include: tasks/sometasks.yml  
when: "reticulating splines' in output"
- hosts: webserver
- roles:
  - { role: debian\_stock\_config, when: ansible\_os\_family == 'Debian' }

示例 9: 引入条件变量

---

- hosts: all  
remote\_user: root  
vars\_files:
  - "vars/common.yml"
  - [ "vars/{{ ansible\_os\_family }}.yml", "vars/os\_defaults.yml" ]
- tasks:
  - name: make sure apache is running  
service: name={{ apache }} state=running

vars/RedHat.yml

apache: httpd

vars/Debian

apache: apache2

如果是 RedHat 系的系统时, 会导入 vars/RedHat.yml 文件, 然后 vars/os\_defaults.yml

示例 10: 基于变量选择文件和模板



```

---
- hosts: 192.168.0.4
  remote_user: root

  tasks:
    - name: INTERFACES | Create Ansible header for /etc/network/interfaces
      template: src={{ item }} dest=/tmp/geekwolf.conf
      with_first_found:
        - files:
            - "{{ansible_distribution}}.conf"
            - "default.conf"

```

## 循环

示例 1: 标准循环 **with\_items**

```

- name: add several users
  user: name={{ item }} state=present groups=wheel
  with_items:
    - testuser1
    - testuser2

```

也可以通过 vars 选项或者 vars\_files 引入变量文件的形式定义使用循环

```

---
- hosts: 192.168.0.4
  remote_user: root
#  vars_files:
#    - somelist.yml
  vars:
    - somelist: [1,2,3,4,5,6]
  tasks:
    - name: lists
      command: echo {{ item }}
      with_items: somelist

```

同样我们在使用 yum 或者 apt 安装多个软件包时也可以使用 with\_items

如果是散列的列表可以这样使用:

```

- name: add several users
  user: name={{ item.name }} state=present groups={{ item.groups }}
  with_items:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }

```

示例 2: 嵌套循环 **with\_nested**

将多个库授权给多个用户 select 权限, 具体 mysql\_user 模块的使用方法参考:

[http://docs.ansible.com/mysql\\_user\\_module.html](http://docs.ansible.com/mysql_user_module.html)

```

---
- hosts: 192.168.0.4
  remote_user: root
  tasks:
    - name: give users select to multiple databases
      mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:select append_privs=yes password=foo
      with_nested:

```

```
- [ 'alice' , 'bob' ]
- [ 'test' , 'test1' ]
```

或者可以通过引用之前定义的变量文件形式使用:

```
- hosts: 192.168.0.4
  remote_user: root
  vars_files:
    - "users.yml"
```

tasks:

```
- name: give users select to multiple databases
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:select append_privs=yes password=foo
  with_nested:
    - users
    - [ 'test' , 'test1' ]
```

示例 3: 遍历字典 **with\_dict**

vim users.yml

```
---
users:
  alice:
    name: Alice
    telephone: 123-456-7890
  bob:
    name: Bob
    telephone: 987-654-3210
```

vim dict\_playbook.yml

```
---
- hosts: 192.168.0.4
  remote_user: root
  vars_files:
    - users.yml

  tasks:

    - name: print name and telephone
      debug: msg="User {{ item.key }} is {{ item.value.name }}({{ item.value.telephone }})"
      with_dict: users
```

示例 4: 使用 **with\_fileglob** 遍历文件 (不递归、对目录无效) **with\_fileglob**

将/root/geekwolf/目录下所有的文件复制到/tmp/fooapp/

```
---
- hosts: 192.168.0.4
  remote_user: root

  tasks:
    #确保目的目录存在
    - file: dest=/tmp/fooapp state=directory
    - copy: src={{ item }} dest=/tmp/fooapp/ owner=root mode=600
      with_fileglob:
        - /root/geekwolf/*
```

示例 5: 使用 **with\_together** 来并行遍历两个列表

```
vim list_test.yml
---
alpha: [ 'a', 'b', 'c', 'd' ]
numbers: [ 1, 2, 3, 4, 5]
假如我们想要['a',1] ['b',2] ...[None,6]可以这样实现:
```

```
vim list_test.yml
---
alpha: ['a','b','c','d']
numbers: [1,2,3,4,5,6]
```

```
vim list_playbook.yml
---
- hosts: 192.168.0.4
  remote_user: root
  vars_files:
    - list_test.yml
  tasks:
    - debug: msg="{{ item.0 }}" and "{{ item.1 }}"
      with_together:
        - alpha
        - numbers
```

#### 示例 6: 循环子元素 **with\_subelements**

创建用户 `alice` 和 `bob`,为 `alice` 配置两把公钥到 `alice` 用户下的 `authorized_keys` 文件, 为 `bob` 配置一把公钥使得两个用户通过客户端私钥登陆, 其中 `onekey.pub` `twokey.pub` `id_rsa.pub` 表示已经创建好的公钥

```
vim subusers.yml
```

```
---
users:
  - name: alice
    authorized:
      - /tmp/alice/onekey.pub
      - /tmp/alice/twokey.pub
  - name: bob
    authorized:
      - /tmp/bob/id_rsa.pub
```

```
vim sub_playbook.yml
---
- hosts: 192.168.0.4
  vars_files:
    - subuser.yml
  tasks:
    - user: name={{ item.name }} state=present generate_ssh_key=yes
      with_items: users
    - authorized_key: "user={{ item.0.name }} key={{ lookup('file', item.1) }}"
      with_subelements:
        - users
        - authorized
```

详细参考:

`authorized_key` 模块: [http://docs.ansible.com/authorized\\_key\\_module.html](http://docs.ansible.com/authorized_key_module.html)

`look_plugins` 插件讲解: <http://rfyamcool.blog.51cto.com/1030776/1441451>

示例 7: 在序列中循环 **with\_sequence**

```
---
- hosts: all

  tasks:

    # 创建用户组
    - group: name=evens state=present
    - group: name=odds state=present

    # 创建格式为 testuser%02x 的 0-32 序列的用户
    - user: name={{ item }} state=present groups=evens
      with_sequence: start=0 end=32 format=testuser%02x

    # 创建目录: 起始位 4,末尾是 16, 步长为 2 命名的目录
    - file: dest=/var/stuff/{{ item }} state=directory
      with_sequence: start=4 end=16 stride=2

    # 简单实用序列的方法: 创建 4 个用户组分表是组 group1 group2 group3 group4
    # create 4 groups
    - group: name=group{{ item }} state=present
      with_sequence: count=4
```

示例 8: 随机选择 **with\_random\_choice**

```
- debug: msg={{ item }}
  with_random_choice:
    - "go through the door"
    - "drink from the goblet"
    - "press the red button"
    - "do nothing"
```

示例 9: until 循环 **until**

```
- action: shell /usr/bin/foo
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

上面的例子 shell 模块会一直执行直到模块结果输出有"all systems go" 或者每个 10s 重试 5 次后结束; 默认是每个 5s 重复 3 次; 执行 playbook 时加上-vv 参数可以查看详细过程

示例 10: 第一次匹配 完成 **with\_first\_found**

```
- name: INTERFACES | Create Ansible header for /etc/network/interfaces
  template: src={{ item }} dest=/etc/foo.conf
  with_first_found:
    - "{{ ansible_virtualization_type }}_foo.conf"
    - "default_foo.conf"
```

配置匹配文件搜索路径:

```
- name: some configuration template
  template: src={{ item }} dest=/etc/file.cfg mode=0444 owner=root group=root
  with_first_found:
    - files:
      - "{{inventory_hostname}}/etc/file.cfg"
    paths:
      - ../../templates.overwrites
      - ../../templates
    - files:
      - etc/file.cfg
    paths:
      - templates
```

示例 11: 打印列表索引 **with\_indexed\_items**

```
---
- hosts: 192.168.0.4

  tasks:
    - name: indexed loop demo
      debug: msg="at array position {{ item.0 }} there is a value {{ item.1 }}"
      with_indexed_items: ['a','b','c','d','e']
```

{{item.0}}表示列表索引, {{item.1}}表示列表对象

示例 12: 整合列表 **with\_flattened**

如果定义了多个列表, 遍历多个列表项

```
---
# file: roles/foo/vars/main.yml
packages_base:
  - ['foo-package', 'bar-package']
packages_apps:
  - ['one-package', 'two-package']
  - ['red-package', 'blue-package']

- name: flattened loop demo
  yum: name={{ item }} state=installed
  with_flattened:
    - packages_base
    - packages_apps
```

## Notify 和 Handlers

在 `playbook` 中, `notify` 行为在每个任务执行的最后会被触发调用 `handlers`, 并且即使 `notify` 多个任务也只是被触发一次; `handlers` 是任务列表和普通任务没有区别, 通过 `notify` 触发使用 `name` 来引用; 在一个 `play` 里面无论对一个 `handlers` 通知调用多少次, 都只是在所有任务执行完后执行一次;

例子: 当文件发生变化时重启 `memcache` 和 `apache`

```
- name: template configuration file
  template: src=template.j2 dest=/etc/foo.conf
  notify:
    - restart memcached
    - restart apache
```

handlers:

- name: restart memcached
  - service: name=memcached state=restarted
- name: restart apache
  - service: name=apache state=restarted

**使用 playbooks 来配置 memcached 示例:**

vim memcache.yml

```
---
- hosts: server
  remote_user: root

  tasks:
    - name: install memcached
      apt: name=memcached state=installed
    - name: set memcached size
      set_fact: memcached_size={{ ansible_memtotal_mb/2 }}
    - name: copy configurations ansible_memtotal_mb/2
      template: src=memcached.j2 dest=/etc/memcached.conf
      notify:
        - restart memcached
    - name: ensure memcached is running
      service: name=memcached state=started
  handlers:
    - name: restart memcached
      service: name=memcached state=restarted
```

vim memcached.j2

```
-p 11311
-l 0.0.0.0
-m {{memcached_size}}
-c 10240
-u memcache
-d
```

文件发生变化时才会通知调用 handlers,在配置文件没变化, memcached 没有启动时这时候就需要确保服务正常启动,在 notify 后面使用 service 模块启动服务

**注释:** 可以使用 gather\_facts:no 关闭 setup 模块的执行, 默认是 yes

## 角色和包含

### 任务包含文件并且复用

比如:

tasks/foo.yml

---

# possibly saved as tasks/foo.yml

- name: placeholder foo
  - command: /bin/foo

```
- name: placeholder bar
  command: /bin/bar
可以在 playbook 中使用 include 包含 foo.yml
tasks:
```

```
- include: tasks/foo.yml
```

include 可以带变量, wp\_user 需要在 wordpress.yml 定义 {{ wp\_user }}

```
tasks:
```

```
- include: wordpress.yml wp_user=timmy
- include: wordpress.yml wp_user=alice
- include: wordpress.yml wp_user=bob
```

Ansible1.4+版本还支持列表和字典

```
tasks:
```

```
- { include: wordpress.yml, wp_user: timmy, ssh_keys: [ 'keys/one.txt', 'keys/two.txt' ] }
```

Ansible1.0+开始, 变量可以通过 include 文件来使用, 也可以通过结构化变量来使用

```
tasks:
```

```
- include: wordpress.yml
  vars:
    wp_user: timmy
    some_list_variable:
      - alpha
      - beta
      - gamma
```

include 在 handlers 中的使用

```
---
# this might be in a file like handlers/handlers.yml
- name: restart apache
  service: name=apache state=restarted
```

```
handlers:
```

```
- include: handlers/handlers.yml
```

一个 playbook 可以 include 另外一个 playbook

```
- 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
- include: webservers.yml
- include: dbservers.yml
```

## 角色

Roles 在 Ansible1.2+版本中支持，是为了更好的组织 playbooks  
举例说明：

```
site.yml
webservers.yml
fooservers.yml
roles/
  common/
    files/
    templates/
    tasks/
    handlers/
    vars/
    defaults/
    meta/
  webservers/
    files/
    templates/
    tasks/
    handlers/
    vars/
    defaults/
    meta/
```

在 playbook 中可以这样使用 roles

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

roles 目录结构说明：

tasks、handlers、vars（只对当前 role 有效）、meta（定义 role 间的直接依赖关系）目录内存在 main.yml 文件时会将对应的任务、处理、变量和 meta 添加到 play  
files 存放文件，ansible 默认会从这里找文件，对应 task 里面的 copy、script 模块  
template 存放模板，对应 task 里面的 template 模块  
tasks 存放任务，include 默认读取这里的任务  
defaults 默认的变量存放位置，使用/defaults/main.yml,相对其他参数变量设置的优先级最低

**注释：**Ansible1.4+以后的版本，可以通过 [roles\\_path](#) 参数配置 roles 路径，多路径使用冒号分隔，可以将常见角色集中存放，指定 roles 路径，这样多个 playbook 可以共用

### 关于角色使用的示例：

#### 1.给角色添加参数

```
---
- hosts: webservers
  roles:
    - common
    - { role: foo_app_instance, dir: '/opt/a', port: 5000 }
    - { role: foo_app_instance, dir: '/opt/b', port: 5001 }
```

#### 2.条件判断是否使用该角色



```
---
```

```
- hosts: webservers
  roles:
    - { role: some_role, when: "ansible_os_family == 'RedHat'" }
```

### 3.使用 tags 标记分配给指定角色

```
---
```

```
- hosts: webservers
  roles:
    - { role: foo, tags: ["bar", "baz"] }
```

### 4.使用 pre\_tasks 和 post\_tasks 分别在 roles 应用的前后执行一些任务

```
---
```

```
- hosts: webservers

  pre_tasks:
    - shell: echo 'hello'

  roles:
    - { role: some_role }

  tasks:
    - shell: echo 'still busy'

  post_tasks:
    - shell: echo 'goodbye'
```

### 关于角色依赖:

角色依赖配置默认在 meta/main.yml 文件, 文件应该在指定的角色之前包含角色依赖和参数列表, 比如下面的例子:

```
---
```

```
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, port: 80 }
  - { role: postgres, dbname: blarg, other_parameter: 12 }
```

角色可以指定绝对路径

```
---
```

```
dependencies:
  - { role: '/path/to/common/roles/foo', x: 1 }
```

角色依赖也可以从版本控制库或者 tar 包获得,使用逗号作为分隔符(分隔路径、版本(tag,commit,branch 等)、可选的角色名字)

```
---
```

```
dependencies:
  - { role: 'git+http://git.example.com/repos/role-foo,v1.1,foo' }
  - { role: '/path/to/tar/file.tgz,,friendly-name' }
```

角色被作为依赖只能执行一次, 如果其他角色也使用同样的依赖将不会被执行, 可以通过 meta/main.yml 文件 allow\_duplicates:yes 来覆盖解决

比如一个叫 car 的角色的依赖是

```
cat car/meta/main.yml
```

```
---
dependencies:
- { role: wheel, n: 1 }
- { role: wheel, n: 2 }
- { role: wheel, n: 3 }
- { role: wheel, n: 4 }
```

我们在 wheel 角色的 meta/main.yml 文件添加

```
---
allow_duplicates: yes
dependencies:
- { role: tire }
- { role: brake }
```

最终的角色执行过程则是:

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

简单来讲, 在一个 task 里面 A 角色依赖 B, B 依赖 C 和 D, C 依赖 D, 那么最后执行的过程如果不在 C 角色的 meta/main.yml 文件增加 allow\_duplicates:yes 就不会在执行 D 了

### 在角色中嵌入模块:

当写了内部模块来帮助配置管理软件和机器, 又想让别人很容易的使用自定义模块的时候可以将模块放在

```
roles/
  my_custom_modules/
    library/
      module1
      module2
```

模块可以被 my\_custom\_modules 角色使用, 也可以被在 my\_custom\_modules 之后的角色调用

```
- hosts: webservers
  roles:
    - my_custom_modules
    - some_other_role_using_my_custom_modules
    - yet_another_role_using_my_custom_modules
```

### 运行 Playbook

```
ansible-playbook playbook.yml -f 10
```

查看模块执行成功与否的详细信息

```
ansible-playbook playbook.yml --verbose
```

查看一个 playbook 中都会对哪些主机产生影响

```
ansible-playbook playbook.yml --list-hosts
```

查看都有哪些任务要执行

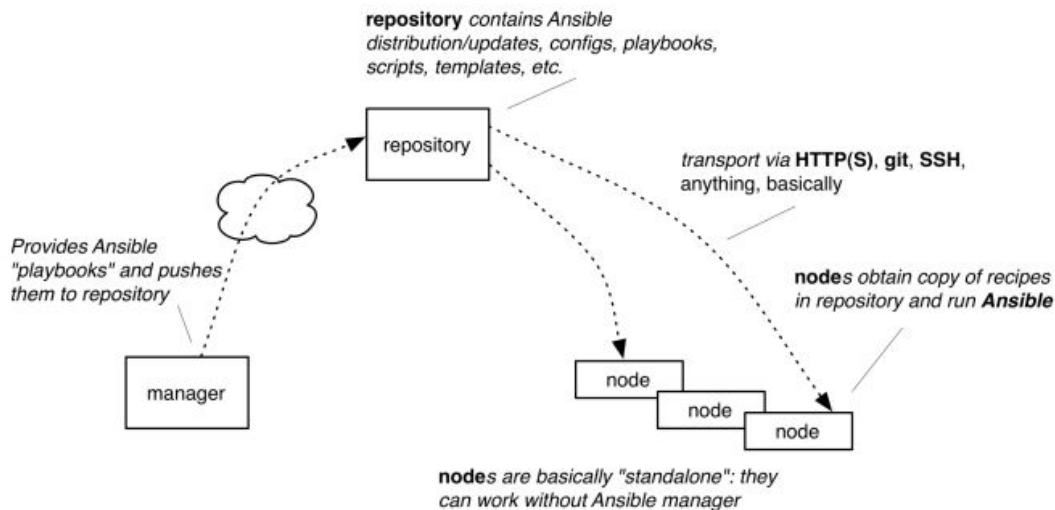
```
ansible-playbook playbook.yml --list-tasks
```

## Ansible 工作模式

前面的学习使用都是基于 Ansible 的 push 模式来工作，即不要在远程主机做任何操作只需要在控制机编排 playbook，push 到远程主机即可完成任务，另外一个工作模式就是 pull

pull 模式适用场景：

1. 被控节点在配置时不可用, 比如自动伸缩的服务池
2. 被控节点较多，控制机资源有限无法满足高线程和低时间的运行效率



Ansible 基于 pull 模式的工作流程：

1. 每台被控端需要安装 Ansible 和 git(svn)
2. 所有的配置及 playbooks 都存放在 git 仓库
3. 被控端的 ansible-pull 计划任务会定期检查给定的 git 的 tag 或者分支
4. ansible-pull 执行特定的 playbooks 即 local.yml 或者 hostname.yml
5. 每次更改 playbooks 只需要提交到 git 即可

现在我们首先使用 push 模式，对被控机做一些设置才能使用 pull 模式：

新系统:Centos6.5 x64 IP:192.168.0.2

1. 设置被控节点支持 pull 模式（之后不需要控制机，或者结合 cobbler 等来节省这一步）

---

```
- hosts: 192.168.0.2
```

```
  user: root
```

```
  vars:
```

```

# schedule is fed directly to cron
schedule: '*/15 * * * *'

# User to run ansible-pull as from cron
cron_user: root

# File that ansible will use for logs
logfile: /var/log/ansible-pull.log

# Directory to where repository will be cloned
workdir: /var/lib/ansible/local

# Repository to check out -- YOU MUST CHANGE THIS
# repo must contain a local.yml file at top level
repo_url: https://github.com/geekwolf/ansible-playbooks.git

tasks:
  - name: Setup EPEL
    action: command rpm -ivh http://mirrors.hustunique.com/epel/6/x86_64/epel-release-6-8.noarch.rpm
    creates=/etc/yum.repos.d/epel.repo

  - name: Install ansible and git
    action: yum name={{ item }} state=latest enablerepo=epel
    with_items:
      - ansible
      - git

  - name: Create local directory to work from
    action: file path={{ workdir }} state=directory owner=root group=root mode=0751

  - name: Copy ansible inventory file to client
    action: copy src=hosts dest=/etc/ansible/hosts
    owner=root group=root mode=0644

  - name: Create crontab entry to clone/pull git repository
    action: template src=templates/etc_cron.d_ansible-pull.j2 dest=/etc/cron.d/ansible-pull owner=root
    group=root mode=0644

  - name: Create logrotate entry for ansible-pull.log
    action: template src=templates/etc_logrotate.d_ansible-pull.j2 dest=/etc/logrotate.d/ansible-pull
    owner=root group=root mode=0644

  - name: ssh keys
    action: shell /usr/bin/ssh-keygen -t rsa -P " " -f /root/.ssh/id_rsa && cat ~/.ssh/id_rsa.pub
    >> ~/.ssh/authorized_keys && /usr/bin/ssh-keyscan 127.0.0.1 github.com >> ~/.ssh/known_hosts

```

**注释:**

上述操作之后被控机将定时同步 git 上的 playbooks 然后执行,可以参考  
<https://github.com/geekwolf/ansible-playbooks.git>; ansible-pull 可以使用参数 -o 来实现只有 playbooks  
 文件发生变化时才执行

**问题:**

msg: github.com has an unknown hostkey. Set accept\_hostkey to True or manually add the hostkey prior to  
 running the git module

**答案:**

```
sudo ssh-keyscan github.com >> /etc/ssh/ssh_known_hosts
```

## 十. Playbooks 高级特性

### 使用 Redis 做 facts 缓存

这个特性是在 Ansible1.8+中才得到支持, 启用 facts 缓存后, 一个主机组中的机器可以引用其他主机组中的变量;目前 Ansible 版本是 1.8.1, facts cache 仍处于 beta, 并未支持 redis 端口密码等

开启 facts 缓存:

```
vim /etc/ansible/ansible.cfg
[defaults]
fact_caching = redis
fact_caching_timeout = 86400 # seconds
```

目前支持 facts cache 的唯一缓存引擎是 Redis

```
apt-get install redis-server
service redis start
pip install redis
```

待续.....

## 十一. 实战

待续.....

### 相关帮助

有疑问就读官方文档: <http://docs.ansible.com/>

Jinja2 中文文档: <http://docs.jinkan.org/docs/jinja2/>

IRC 交流: <https://webchat.freenode.net/> channel: Ansible

Bug 提交: <https://github.com/ansible/ansible/issues>

使用问题交流: <https://groups.google.com/forum/#!forum/ansible-project>

新手推荐学习视频: [http://edu.51cto.com/course/course\\_id-2220.html](http://edu.51cto.com/course/course_id-2220.html)