Ansible 自动化运维

版本	日期	修订人	备注
V 0.1	2016-04-27	李培斌	初版
V 0.2	2016-05-04	李培斌	修改

目录大纲

零、Ansible 自动化运维：参考文档

1、Ansible 官方文档指南

```
http://docs.ansible.com/ansible
```

2、Ansible 中文权威指南

```
http://www.ansible.com.cn
```

3、推荐阅读博客

1）云峰

```
http://xiaorui.cc/category/ansible
http://rfyiamcool.blog.51cto.com/1030776/d-51
```

2）灿哥

```
http://www.shencan.net/index.php/category/自动化运维/ansible
```

4、Ansible 官方源版本下载

```
http://releases.ansible.com/ansible
```

5、Ansible-Tower 官方下载

```
http://releases.ansible.com/ansible-tower
```

6、Salt 与 Ansible 全面比较

原文链接：http://jensrantil.github.io/salt-vs-ansible.html

一、Ansible 自动化运维：部署配置

1、原理介绍

1）运维工具

I、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

II、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

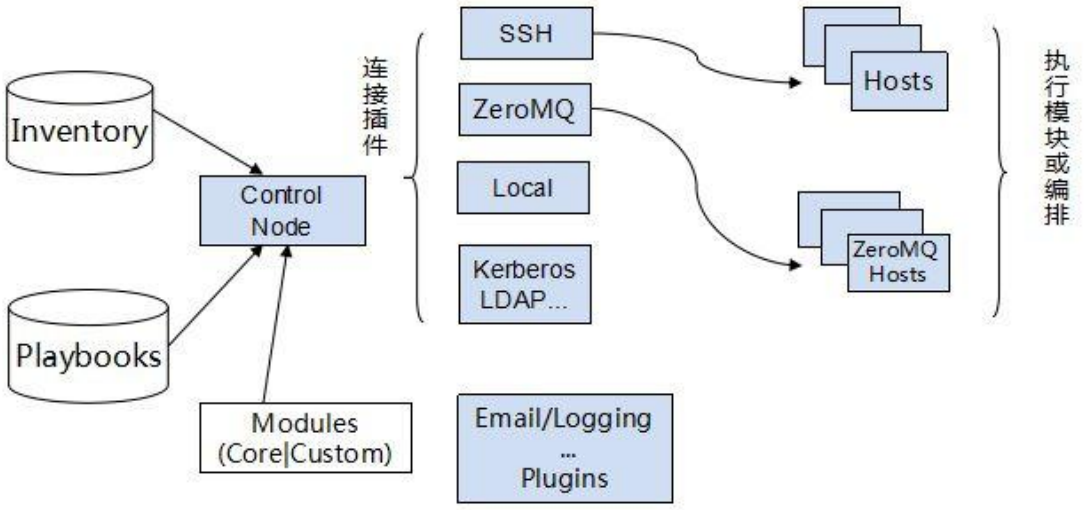
III、Ansible

> 基于 Python paramiko 开发，分布式，无客户端，轻量级，配置语法使用 YMAL 及 jinja 2 模板语言，更强的远程命令执行操作（*1.3 版本后默认为 openssh，但仍兼容支持 paramiko*）。Ansible 最伟大的功能之一是它无需运行任何 daemon 进程和它的远程主机无需要安装客户端即可实现机器互通.且通过标准和安全的 ssh 协议(非私有协议)连接远程主机，众所周知该协议几乎覆盖所有的 linux 服务器做基础管理，安全、稳定、可靠

2）工作机制

I、系统架构

> Ansible 是一个简单的运维管理工具，采用基于 openssh 或者 paramiko 的 ssh 协议（fabric 也使用这个）和 ZeroMQ 插件等连接远程主机，可以用来部署应用、配置、编排 task(持续交付、无宕机更新等)，实现自动化、批量管理服务器和应用服务



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	核心功能，安装 ansible 工具，下面会讲解到如何在 linux 系统上通过多种方式安装部署 ansible 工具
Modules	包括 ansible 自带的核心模块及自定义模块，常用的模块请参考*<< Ansible 自动化运维：常用模块 >>*
Plugins	完成模块功能的补充，包括连接插件、邮件插件、日志插件等
Playbooks	网上很多翻译为剧本或者编排，定义 ansible 任务的配置文件，其它由多个 play 和 task 组成。关于 playbooks 的介绍请参考常用的模块请参考*<< Ansible 自动化运维：剧本（Playbooks） >>*
Inventory	保存要执行模块和编排、剧本任务的远程主机列表,请参考*<< Ansible 自动化运维：主机列表（Inventory） >>*

II、执行流程

Ansible 在首先通过 ssh 协议（或者 ZeroMQ 等插件）连接到远程主机，根据 Ansible 执行命令、或 playbook 中调用的模块、参数、变量、渲染模板等在本地临时目录/tmp 下生成一个可执行的 py 文件，推送到远程主机上的$HOME/.ansible/tmp/ansible-tmp-数字

/xxx 下，然后在远程主机上执行，返回结果给 Ansible，执行完成后自动删除远程主机上的$HOME/.ansible/tmp/ansible-tmp-数字/目录所有文件

2、安装部署

1）pip 安装

```
pip install ansible
```

安装指定版本：

```
pip install ansible==1.9.4
```

2）yum 安装

```
epel 源

# rdhat 5 的 epel

rpm -ivh http://mirrors.sohu.com/fedora-epel/5/x86_64/epel-release-5-4.noarch.rpm

# rhat 6 的 epel

rpm -ivh http://mirrors.sohu.com/fedora-epel/6/x86_64/epel-release-6-8.noarch.rpm

# 安装

yum -y install ansible
```

3）源码安装

```
 (1)、python2.7 安装

# wget https://www.python.org/ftp/python/2.7.8/Python-2.7.8.tgz

# tar xvzf Python-2.7.8.tgz

# cd Python-2.7.8

# ./configure --prefix=/usr/local

# make --jobs=`grep processor/proc/cpuinfo | wc -l`
```

```
# make install

## 将 python 头文件拷贝到标准目录，以避免编译 ansible 时，找不到所需的头文件
# cd /usr/local/include/python2.7
# cp -a ./* /usr/local/include/

## 备份旧版本的 python，并符号链接新版本的 python
# cd /usr/bin
# mv python python2.6
# ln -s /usr/local/bin/python /usr/bin/python

## 修改 yum 脚本，使其指向旧版本的 python，已避免其无法运行
# vim /usr/bin/yum
#!/usr/bin/python  -->  #!/usr/bin/python2.6

(2)、setuptools 模块安装
# wget https://pypi.python.org/packages/source/s/setuptools/setuptools-7.0.tar.gz
# tar xvzf setuptools-7.0.tar.gz
# cd setuptools-7.0
# python setup.py install

(3)、pycrypto 模块安装
# wget https://pypi.python.org/packages/source/p/pycrypto/pycrypto-2.6.1.tar.gz
# tar xvzf pycrypto-2.6.1.tar.gz
# cd pycrypto-2.6.1
# python setup.py install

(4)、PyYAML 模块安装
```

```
# wget http://pyyaml.org/download/libyaml/yaml-0.1.5.tar.gz
# tar xvzf yaml-0.1.5.tar.gz
# cd yaml-0.1.5
# ./configure --prefix=/usr/local
# make --jobs=`grep processor/proc/cpuinfo | wc -l`
# make install

# wget https://pypi.python.org/packages/source/P/PyYAML/PyYAML-3.11.tar.gz
# tar xvzf PyYAML-3.11.tar.gz
# cd PyYAML-3.11
# python setup.py install
```

(5)、jinja2 模块安装

```
# wget https://pypi.python.org/packages/source/M/MarkupSafe/MarkupSafe-0.9.3.tar.gz
# tar xvzf MarkupSafe-0.9.3.tar.gz
# cd MarkupSafe-0.9.3
# python setup.py install

# wget https://pypi.python.org/packages/source/J/Jinja2/Jinja2-2.7.3.tar.gz
# tar xvzf Jinja2-2.7.3.tar.gz
# cd Jinja2-2.7.3
# python setup.py install
```

(6)、paramiko 模块安装

```
# wget https://pypi.python.org/packages/source/e/ecdsa/ecdsa-0.11.tar.gz
# tar xvzf ecdsa-0.11.tar.gz
# cd ecdsa-0.11
# python setup.py install
```

```
# wget https://pypi.python.org/packages/source/p/paramiko/paramiko-1.15.1.tar.gz
# tar xvzf paramiko-1.15.1.tar.gz
# cd paramiko-1.15.1
# python setup.py install

(7)、simplejson 模块安装
# wget https://pypi.python.org/packages/source/s/simplejson/simplejson-3.6.5.tar.gz
# tar xvzf simplejson-3.6.5.tar.gz
# cd simplejson-3.6.5
# python setup.py install

(8)、ansible 安装
# wget https://github.com/ansible/ansible/archive/v1.7.2.tar.gz
# tar xvzf ansible-1.7.2.tar.gz
# cd ansible-1.7.2
# python setup.py install
```

3、远程连接

参考运维部落文章<< Ansible 系列课程-SSH 发展史 >>

最起初，Ansible 只使用 paramiko 实现底层通信功能。但 paramiko 只是 python 的语法的一个第三方库，发展速度远不及 openssh（基于标准 ssh 协议的实现几乎应用遍布开源界），与此同时，paramik 的性能和安全性较 openssh 来比稍逊一筹。从 Ansible1.3 版本开始，默认使用 openssh 连接实现各服务器间通信以支持 controlpersist，在后续发布的新版本中，Ansible 仍继续兼容支持 paramiko ，主要是在诸如 RHEL5/6 等不支持 controlpersist 的系统中封装其为默认通信模块

Ansible 默认使用原生 openssh 来连接远程主机，并开启 controlpersist 指令优化连接速度。如果使用的是 rhel 系统及其衍生系统如 centos，openssh 的版本太旧，不支持 controlpersist（openssh5.6 版本才开始增加 controlpersist 指令，**ssh -V** 可以查看版本号），这时候 ansible 就会选择使用 paramiko 来连接远程主机

大家知道，ansible 的 ssh 连接非常慢，主机原因是在 ansible 的 playbook 里同一个 task 里面是并行的控制多台远程主机.但是每一个 task 都需要和远程主机创建一次 ssh 通道，非常影响效率，下面讲解怎么去做优化 ssh 连接，提高 ansible 的工作效率

1）Ansible 效率优化

I、会话持久：controlpersist

ssh controlpersist 功能允许 ssh 连接在 ssh config 配置的过期时间内长期保持存活，以便使常用命令的执行不必每次都经过最初的握手过程。其实就是持久化 socket，一次验证多次通信.并且只需要修改配置 ssh client 即可，这里相对远程主机来说，ansible 主机就是 ssh client

如上面所说，为了 ansible 的工作效率，可以开启 controlpersist 特性。对于 ssh 版本低于 openssh5.6 的操作系统，如 RHEL 系统及其衍生系统如 CentOS，需要先升级 ssh 到 5.6 以上，以下操作都是在 ansible 服务器上配置

首先安装 dropbear，防止升级失败，可以使用 dropbear 连接到 ansible 服务器

```
$ yum install zlib* gcc make
$ wget https://matt.ucc.asn.au/dropbear/releases/dropbear-2014.66.tar.bz2
$ tar jxf dropbear-2014.66.tar.bz2
$ cd dropbear-2014.66
$ ./configure
$ make && make install
$ mkdir -p /etc/dropbear
```

```
 $ /usr/local/bin/dropbearkey -t dss -f /etc/dropbear/dropbear_dss_host_key

 $ /usr/local/bin/dropbearkey -t rsa -s 4096 -f /etc/dropbear/dropbear_rsa_host_key
```

启动 dropbear

```
/usr/local/sbin/dropbear -p 2222

netstat -luntp|grep 2222

ssh root@ansible_ip -p 2222       ## 验证登录是否正常
```

备份旧版本 ssh

```
openssl version -a

mv /etc/ssh/ /etc/ssh_bak
```

卸载旧版本 ssh

```
rpm -qa | grep openssh && rpm -e `rpm -qa | grep openssh` --nodeps
```

安装新版 ssh

```
$ yum -y install gcc* make openssl openssl-devel perl pam pam-devel

$ wget http://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-6.1p1.tar.gz

$ tar zxf openssh-6.1p1.tar.gz && cd openssh-6.1p1

$ ./configure --prefix=/usr --sysconfdir=/etc/ssh --with-pam --with-zlib --with-md5-passwords

$ make install
```

启动新版本 ssh

```
$ cp /root/openssh-6.1p1/contrib/redhat/sshd.init /etc/init.d/sshd
```

```
$ echo "UseDNS no" >> /etc/ssh/sshd_config
$ chkconfig --add sshd
$ /etc/init.d/sshd restart
$  ssh -V  ## 验证版本号
```

ssh 配置 controlpersist 优化

参考博客 MIKE'S BLOG 文章：<< 利用 ControlPersist 特性自动登陆 SSH 服务器 >>

```
$  cat > /home/{user}/.ssh/config <<EOF   ## {user} ansible 的远程用户
Host *
Compression yes
ServerAliveInterval 60
ServerAliveCountMax 5
ControlMaster auto
ControlPath ~/.ssh/sockets/%r@%h-%p
ControlPersist 4h
EOF
```

验证持久化效果

未持久化 socket 前,需要 2 秒

```
time ansible -i hosts test -m ping
172.20.99.31 | success >> {
    "changed": false,
    "ping": "pong"
}
real    0m2.046s
user    0m0.984s
sys     0m0.593s
```

持久化后,800 毫秒

```
time ansible -i hosts test -m ping
172.20.99.31 | success >> {
    "changed": false,
    "ping": "pong"
}
real    0m0.810s
user    0m0.367s
sys     0m0.246s
```

II、加速模式：accelerate

据官网介绍：加速模式在开启后速度有质的提升，是 ssh 开启 controlpersist 的 2-6 倍，是 paramiko 的 10 倍,尤其对于文件传输功能

加速模式抛弃 ssh 多次连接的方式,通过 ssh 第一次连接初始化后,ansible 推送几个 py 文件到远程主机并运行，让远程主机带着 AES key 的初始化连接信息访问 Ansible 主机特定的端口（默认 5099，这时远程主机类似于 salt 的 agent 一样），主动拉取执行命令和执行脚本文件在本地执行

使用加速模式唯一需要的额外包是 python-keyczar

```
pip install python-keyczar
```

加速模式使用方法：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,如下：

```
echo export ACCELERATE_PORT=5099 >> /etc/profile && source /etc/profile
```

或者修改 ansible.cfg 配置文件

vim /etc/ansible/ansible.cfg

```
[accelerate]
accelerate_port = 5099   # default port is 5099
```

也可以直接在 playbook 中直接应用

```
---
- hosts: all
  accelerate: true
  accelerate_port: 5099   # default port is 5099
  accelerate_multi_key = yes   # 允许使用多把密钥
```

注：ansible 加速模式支持 sudo 操作,但是需要满足以下两点条件：

a、sudoers 文件需要关闭 requiretty 功能

vim /etc/sudoers

```
#Defaults requiretty;
```

b、sudoers 文件配置 NOPASSWD

比如以 ops 用户连接到远程主机，然后 sudo 到 xiaoniu 用户执行某个任务，这时需要在远程主机上配置 ops 免密码 sudo 到 xiaoniu

vim /etc/sudoers

```
ops ALL = NOPASSWD:ALL(xiaoniu) ALL        //ops 就是 ops 用户

# %ops ALL = NOPASSWD:ALL(xiaoniu) ALL    //%ops 是指 ops 组下的所有用户
```

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

III、改进提升：pipelining

Ansible 1.5+版本中的 openssh 有了非常大的改进提升。推送执行脚本等到远程服务器上运行,然后删除它们,但在新版本的代替方案是 openssh 发送执行命令等操作附带在 ssh 连接过程同步实现。该方式的连接只在 1.5+版本有效，且可在 [ssh_connection] 区域增加配置 pipelining=True 开启功能，同时得关闭 sudoers 文件的 requiretty 功能

vim /etc/sudoers

```
#Defaults requiretty;
```

vim /etc/ansible/ansible.cfg

```
[ssh_connection]
pipelining=True
```

2）远程连接验证方式

ansible 支持密码验证和私钥验证，默认是使用私钥验证。

I、密码验证

如果想使用密码验证，则需要为 `ansible` 和 `ansible-playbook` 命令提供`-k, --ask-pass` 参数，连接远程主机时会提示你输入用户密码

II、私匙验证

下面介绍两种方法可以实现批量配置 ansible 与远程主机之间主机信任

a、shell 脚本

```
#!/bin/bash
```

```
# set_ssh_keys.sh
password="123123"     ## 主机的密码，每个主机的密码要求一样
auto_ssh_copy_id() {
    expect -c "set timeout -1;
    spawn ssh-copy-id -i $2;
    expect {
  *(yes/no)* {send -- yes\r;exp_continue;}
        *assword:* {send -- $1\r;exp_continue;}
        eof        {exit 0;}
    }";
}
for i in $(</root/hosts) ##主机 ip 文件，一行一个
do
     auto_ssh_copy_id $password $i
done
```

b、authorized_key 模块

```
ansible-playbook xiaoniu-authorized-key.yml -e "hosts=all user=ops" -k
```

```
---
- name: create user on ansible server as remote_user
  user: name="{{ user }}"
        generate_ssh_key=yes
        ssh_key_type=rsa
        ssh_key_bits=2048
        ssh_key_file=.ssh/id_rsa
        state=present
  delegate_to: localhost
  tags: authorized-key
```

```
- name: create user on remote host
  user: name="{{ user }}" state=present
  tags: authorized-key


- name: copy id_rsa.pub to remote host for authorized trust
  authorized_key: user="{{ user }}" key="{{ lookup("file","/home/"+user+
"/.ssh/id_rsa.pub") }}"
  tags: authorized-key
```

4、第一个命令

1）简单演示

在/etc/ansible/hosts 中添加一个远程主机 `172.20.99.32`，/etc/ansible/目录不存在就自己创建

如果你使用私钥验证，执行如下命令：

```
ansible all -m ping
```

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

如果你使用密码验证，则还需要加上`-k`参数来转换到密码验证模式，默认远程连接使用你所在的当前用户，可以使用`-u`参数指定用什么用户连接

一般建议服务器都设置成私钥验证（配置主机间的相互信任），验证安全性高

安装配置完成后，可以连接到远程主机执行各种各样的操作，ansible 提供了大量的模块用来完成不同的任务，常用的模块请参考*<< Ansible 自动化运维：常用模块 >>*

`ansible-doc -l` 列出所有可用模块

`ansible-doc shell` 查看 `shell` 模块文档

如，利用 `shell` 模块在远程主机执行命令：

ansible all -m shell -a "uname -a"

```
172.20，99.32 | success | rc=0 >>
Linux centos6 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
```

关于更多的模块 http://docs.ansible.com/ansible/list_of_all_modules.html

或者请参考<< Ansible 自动化运维：常用模块 >>

2）ssh 首次连接验证

ansible 第一次连接远程主机时，可能会报以下错误

```
172.20.99.32 | FAILED => Using a SSH password instead of a key is not possible because Host Key checking is enabled and sshpass does not support this.  Please add this host's fingerprint to your known_hosts file to manage this host
```

从上面的输出提示上基本可以了解到由于在本机的~/.ssh/known_hosts 文件中并有 fingerprint key 串，ssh 第一次连接的时候一般会提示输入 yes 进行确认为将 key 字符串加入到~/.ssh/known_hosts 文件

解决办法：

I、StrictHostKeyChecking

ssh 连接时，可以使用-o 参数将 StrictHostKeyChecking 设置为 no，使用 ssh 连接时直接忽略验证，可以解决首次连接时要输入 yes/no 的提示

vim /etc/ansible/ansible.cfg

```
[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s -o StrictHostKeyChecking＝no
```

II、host_key_checking

ansible 跳过 ssh 首次连接验证

vim /etc/ansible/ansible.cfg

```
host_key_checking = False          #  推荐用方法
```

5、命令参数

ansible 命令参数

```
Usage: ansible <host-pattern> [options]
Options:
   # 要执行的模块，默认为 command
  -m MODULE_NAME, --module-name=MODULE_NAME

  # 模块的参数
  -a MODULE_ARGS, --args=MODULE_ARGS
  # ssh 连接的用户名，默认用 root，ansible.cfg 中可以配置
  -u REMOTE_USER, --user=REMOTE_USER

  # ssh 登录密码
  -k, --ask-pass

  # sudo 运行
  -s, --sudo

  # sudo 到哪个用户，默认为 root
  -U SUDO_USER, --sudo-user=SUDO_USER

  # sudo 密码
  -K, --ask-sudo-pass
```

```
-B SECONDS, --background=SECONDS

# set the poll interval if using -B (default=15)
-P POLL_INTERVAL, --poll=POLL_INTERVAL

# 只是测试一下会改变什么内容，不会真正去执行
-C, --check

# 连接类型(default=smart)，首选 openssh,然后是 paramiko
-c CONNECTION

# fork 多少个进程并发处理，默认 5
-f FORKS, --forks=FORKS

 # 指定 hosts 文件路径，默认 default=/etc/ansible/hosts
-i INVENTORY, --inventory-file=INVENTORY

 # 指定一个 pattern，对<host_pattern>已经匹配的主机中再过滤一次
-l SUBSET, --limit=SUBSET

 #只打印有哪些主机会执行这个 playbook 文件，不是实际执行该 playbook
--list-hosts

# 压缩输出，摘要输出
-o, --one-line

# 私钥路径
--private-key=PRIVATE_KEY_FILE

 # ssh 连接超时时间，默认 10 秒
```

```
  -T TIMEOUT, --timeout=TIMEOUT


  #日志输出到该目录，日志文件名会以主机名命名
  -t TREE, --tree=TREE


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

6、ansible.cfg 配置文件

Ansible 默认安装好后有一个配置文件/etc/ansible/ansible.cfg，该配置文件中定义 ansible 的主机的默认配置部分，如默认是否需要输入密码、是否开启 sudo 认证、action_plugins 插件的位置、hosts 主机组的位置、是否开启 log 功能、默认端口、key 文件位置等等，具体如下：

```
[defaults]  \\常规默认配置
hostfile = /etc/ansible/hosts       \\指定默认 hosts 配置的位置
# library_path = /usr/share/my_modules/  \\模块默认存放的位置
remote_tmp = $HOME/.ansible/tmp       \\远程主机缓存放的位置，主机是缓存模块指令、文件、变量参数等
pattern = *                           \\默认匹配所有主机
forks = 5                             \\task 分配给远程主机的并发数
roles_path = /etc/ansible/roles  \\roles 保存位置，默认/etc/ansible/roles
poll_interval = 15                    \\定时检查后台任务状态的时间间隔
sudo_user = root                      \\远程 sudo 用户，默认为 root
#ask_sudo_pass = True                 \\每次执 ansible 命令是否询问 ssh 密码
#ask_pass = True  \\每次执行 ansible 命令时是否询问 sudo 密码
Transport = smart \\连接远程主机的方式，smart 自动选择，其它有 ssh、paramiko
remote_port = 22  \\连接远程主机的端口，默认是 22
module_lang = C
```

```
gathering = implicit
host_key_checking = False  \\关闭第一次使用 ansible 连接客户端是输入命令提示
log_path = /var/log/ansible.log   \\开启 ansible 日志
system_warnings = False    \\关闭运行 ansible 时系统的提示信息，一般为提示升级
# 下面的是一些插件的存放位置
action_plugins     = /usr/share/ansible_plugins/action_plugins
callback_plugins   = /usr/share/ansible_plugins/callback_plugins
connection_plugins = /usr/share/ansible_plugins/connection_plugins
lookup_plugins     = /usr/share/ansible_plugins/lookup_plugins
vars_plugins       = /usr/share/ansible_plugins/vars_plugins
filter_plugins     = /usr/share/ansible_plugins/filter_plugins
# ########################
fact_caching = memory   \\fact 缓存，默认为 mem，另外支持 redis 和 json 方式
[accelerate]  \\ansible 加速模式配置
accelerate_port = 5099      \\加速端口
accelerate_timeout = 30      \\加速超时时间
accelerate_connect_timeout = 5.0   \\加速连接超时时间
accelerate_daemon_timeout = 30      \\


[ssh_connection]                    \\ssh 连接优化
#ssh_args = -o ControlMaster=auto -o ControlPersist=60s  \\ssh 连接参数，可以通过 -o StrictHostKeyChecking=no 跳过首次连接验证的其中一个方法
#pipelining = True                   \\默认为 False
```

二、Ansible 自动化运维：常用模块

ansible 的模块有很多，网罗了系统、网络、虚拟化以及第三方应用等，而且还支持自己开发模块，请参考阅读官网的 ansible 模块相关篇章：[ansible modules]

下面针对性的对一些比较常用的模块进行讲解（以下模块的示例说明都是在 playbook 中应

用）

1、command

默认执行模块，在远程主机执行 shell 指令

参数	必需	默认值	注解
chdir	no		在运行命令前 cd 到这个目录中，相当于 cd xxx && command
creates	no		当文件存在时，该任务跳过不执行
removes	no		当文件不存在时，该任务才执行

示例说明：

```
# 关闭系统
-command: /sbin/shutdown -t now

# 当指定的文件存在时，跳过不执行该命令
-command: /usr/bin/make_database.sh arg1 arg2 creates=/path/to/database

# 当指定不文件存在时，则该命令将会执行
-command: /usr/bin/make_database.sh arg1 arg2 creates=/path/to/database
```

注：这个模块不支持管道和重定向，要支持管道可以用-m shell 或者 -m raw
shell 和 raw 语法一样，它们支持管道和重定向，如：

```
- shell: ifconfig|grep inet|grep -Ev '127.'|cut -d':' -f2|cut -d' ' -f1
- raw: ifconfig|grep inet|grep -Ev '127.'|cut -d':' -f2|cut -d' ' -f1 > /tmp/ip.log
```

2、copy

复制文件、目录等，并对其进行权限修改

参数	必需	默认值	注解
backup	no	no	当复制的文件在远程主机上存在，覆盖前进行备份一次

force	no	yes	强制覆盖
group	no		文件、目录属组，类似 chown 指令
mode	no		文件、目录权限，如 mode=644 或者 mode="u+x，g+x,o+x"类似 chmod 指令
owner	no		文件、目录属主，类似 chown 指令
src	no		复制的源路径
dest	yes		复制到远程主机上目标路径
follow	no	no	当 src 指定的是链接文件，若使用 follow 则会把链接的真实文件复制,否则将仅仅复制 link 文件
directory_mode	no		目录模式

示例说明：

```
# 复制 redis 服务脚本到远程主机机器，force 为强制性复制

- copy: src=/data/ansible/file/redis/redis-server dest=/etc/init.d/ mode=750 owner=root group=root force=yes


# 复制目录，directory_mode 目录模式

- copy: src=/root/nginx dest=/root/nginx mode=750 owner=ops group=ops directory_mode=yes


# 复制前对文件进行备份，backup=yes

- copy: src=/srv/myfiles/foo.conf dest=/etc/foo.conf owner=foo group=foo mode="u+rw,g-wx,o-rwx" backup=yes


# 多个文件的复制，采用循环 with_items

- copy: src={{ item }} dest=/root/

  with_items:

  - /data/a.sh

  - /logs/b.py


# 文件通配符循环 with_fileglob

- copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
```

```
  with_fileglob:
  - /playbooks/files/fooapp/*
```

注：fetch 模块与 copy 相反，是从远程主机复制文件到 ansible 服务器

3、file

文件、文件夹权限的变更，以及文件、文件夹、超级链接类的创立、拷贝、移动、删除操作

参数	必需	默认值	注解
force	no	no	强制覆盖，包括权限、属性、内容
group	no		文件、目录属组，类似 chown 指令
mode	no		文件、目录权限，如 mode=644 或者 mode="u+x，g+x,o+x"类似 chmod 指令
owner	no		文件、目录属主，类似 chown 指令
path	yes		目标文件、目录位置（远程主机），dest,、name 同样适用
recurse	no	no	递归修改文件属性(仅适用于 state=directory)
src	no		源文件、目录位置，适用于 state=link 或者 hard

state	no	file	state= directory # 如果目录不存在将会创建
			state=file # 检查指定的 path 路径是否是文件
			state=link # 创建软链接
			state=hard # 创建硬链接
			state=touch # 创建文件
			state=absent # 删除文件

示例说明：

```
# 创建文件夹,recurse=yes 递归，相当于 mkdir -p
- file: path=/root/soft/ state=directory mode=0755 recurse=yes

# 创建文件
- file: path=/etc/foo.conf state=touch mode="u=rw,g-wx,o-rwx"

# 删除文件
- file: path=/root/soft state=absent
```

```
# 修改文件目录属性
- file: path=/root/soft mode=750 owner=ops group=ops

# 创建文件软件链接
- file: src=/file/link dest=/path/symlink owner=foo group=foo state=link

# 批量创建文件软件链接
- file: src=/tmp/{{ item.src }} dest={{ item.dest }} state=link
  with_items:
       - { src: 'x', dest: 'y' }
       - { src: 'z', dest: 'k' }
```

4、stat

查看文件目录属性，包括创建时间、访问时间、修改时间、存在状态、路径位置、文件大小等等

参数	必需	默认值	注解
path	yes		文件目录的路径

执行：ansible all -m stat -a "path=/data/lipeibin"，返回结果如下：

```
"stat": {
        "atime": 1459386908.4750736,
        "ctime": 1459386908.4740734,
        "dev": 20,
        "exists": true,
        "gid": 99,
        "gr_name": "nobody",
        "inode": 655266,
        "isblk": false,
```

```
        "ischr": false,
        "isdir": true,
        "isfifo": false,
        "isgid": false,
        "islnk": false,
        "isreg": false,
        "issock": false,
        "isuid": false,
        "mode": "0757",
        "mtime": 1459386845.7500744,
        "nlink": 2,
        "path": "/data/lipeibin",
        "pw_name": "nobody",
        "rgrp": true,
        "roth": true,
        "rusr": true,
        "size": 4096,
        "uid": 99,
        "wgrp": false,
        "woth": true,
        "wusr": true,
        "xgrp": true,
        "xoth": true,
        "xusr": true
    }
}
```

可以根据上面文件目录属性输出的结果可以在后面的 task 中引入，如：

```
# 判断一个路径是存在，且是一个目录
- stat: path=/path/to/something
```

```
    register: reg
- debug: msg="Path exists and is a directory"
  when: reg.stat.isdir is defined and p.stat.isdir

# 判断文件属主是否发生改变
- stat: path=/etc/foo.conf
  register: reg
- fail: msg="Oh,file ownership has changed"
  when: re.stat.pw_name != "root"

# 判断文件是否存
- stat: path=/path/to/something
  register: reg
- debug: msg="file doesn't exist"
  when: not reg.stat.exists
- debug: msg="file is exist"
  when: reg.stat.exists
```

5、user

用户管理，包括用户创建、删除，shell 类型、密钥生成以及用户属性设置等

参数	必需	默认值	注解
name	yes		用户名
comment	no		用户名的描述，方便管理
home	no		用户主目录位置
passwd	no		用户密码（不能是明文，需要加密）
group	no		用户属组
expires	no		用户到期时间，使用时间戳表示
force	no		强制性删除 一般用于 state=absent

shell	no		指定 shell 类型:/bin/bash /sbin/nologin(禁止 ssh)
ssh_key_file	no	.ssh/id_rsa	生成用户密钥文件
state	yes	present	state=present 创建用户 state=absent 删除用户
remove	no	no	删除用户时，是否删除用户主目录,默认为 no

示例说明:

```
# 创建用户 lipeibin
- user: name=lipeibin groups=lipeibin  shell=/sbin/nologin state=present


# 删除用户 lipeibin,并删除用户主目录/home/lipeibin
- user: name=lipeibin state=absent remove=yes


#  创建用户 lipeibin,并且设置密码，归属 ops 组
- user: name=lipeibin password=”19890506”| password_hash('sha512') group
=ops


# 设置用户有效时间
- user: name=lipeibin shell=/bin/bash groups=ops expires=1422403387
```

6、 group

用户组管理，包括创建、删除等

参数	必需	默认值	注解
gid	no		创建用户组指定用户组的 id
name	yes		用户组的名字
state	no	present	state=present 创建 state=absent 删除
system	no	no	system=yes 表示创建一个系统用户组

示例说明：

```
# 创建用户组
group: name=lipeibin gid=1024 state=present


# 删除用户组
group: name=lipeibin state=absent
```

7、 yum

rpm 包的安装、更新、删除等等

参数	必需	默认值	注解
name	yes		安装包名字，可以是一个路径或者 url、或者系统工具包名称
enablerepo	no		指定 yum 源
state	no	present	state=present state=installed state=lasest state=absent state=removed 安装(present or installed, latest) 卸载(absent or removed)

示例说明：

```
- name: 安装最新版本的 Apache
  yum: name=httpd state=latest


- name: 删除 Apache
  yum: name=httpd state=absent


- name: 从指定的 yum 源安装最新版本的 Apache
  yum: name=httpd enablerepo=epel state=present
```

```
- name: 安装指定 url 的 RPM 包
yum: name=http://mirrors.sohu.com/fedora-epel/6/x86_64/epel-release-6-8.noarch.rpm state=present


- name: 安装指定版本号的 Apache
  yum: name=httpd-2.2.29-1.4.amzn1 state=present


- name: 更新系统所有的包
  yum: name=* state=latest


- name: 从本地路径上安装指定的 nginx
  yum: name=/usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm state=present


- name: 安装系统开发工具包
  yum: name="@Development tools" state=present
```

8、service

系统服务管理,包括启动、关闭、重启、以及设置开机启动等

参数	必需	默认值	注解
name	yes		进程服务名

state	no		state=started # 服务启动
			state=stoped # 服务关闭
			state=restarted # 服务重启
			state=reload # 服务重载

enabled	no		enabled=yes # 设置服务开机启动

示例说明:

```
# 启动服务,并设置开机启动
- service: name=vsftpd state=started enabled=yes


# 关闭服务
```

```
- service: name=vsftpd state=stopped


# 重启服务
- service: name=vsftpd state=restarted


# 重载服务
- service: name=vsftpd state=reloaded
```

9、cron

计划任务管理，包括新建，删除等

参数	必需	默认值	注解
name	yes		任务描述
mintue	no		分钟： 0-59, *, */2
hour	no		小时：1-12, *, */2
day	no		天：1-31, *, */2
weekday	no		周：0-6 for Sunday-Saturday
month	no		月：1-12, *, */2
user	no	root	执行用户
special_time	no		# 特殊时间需 Reboot # 系统重启的时候执行 yearly # 每年一次 annually # 每年一次 monthly # 每月一次 weekly # 每星期一次 daily # 每天一次 hourly # 每小时一次
job	yes		执行任务命令

state	no	present	state=present # 新增任务
			state=absent # 删除任务

backup	no		yes：修改前对任务文件进行备份

示例说明：

```
# 创建计划任务：0 5,2 * * ls -alh > /dev/null
- cron: name="check dirs" minute="0" hour="5,2" job="ls -alh > /dev/null" state=present

# 删除名为“an old job”计划任务
- cron: name="an old job" state=absent
```

10、get_url

下载，类似于 shell 命令 curl 或者 wget

参数	必需	默认值	注解
url	yes		url 地址
dest	no		下载存放的路径

示例说明：

```
# 下载 ansible 源包
- get_url: url= https://github.com/ansible/ansible/archive/v1.7.2.tar.gz
 dest=/tmp
```

11、unarchive

解压包，支持 zip、tar、jar 等

参数	必需	默认值	注解
src	yes		源包文件路径位置，可以是网络地址
copy	no	yes	copy=no 时，src 文件路径是远程主机上 copy=yes 时，src 文件路径是 ansible 主机上，解压包后上传到远程主机 dest 目录
dest	yes		远程存放解压包的路径位置
mode	no		文件权限，如 mode=644 等
group	no		文件属组
owner	no		文件属主

示例说明：

```
# 解压包，然后上传到远程主机/var/lib/foo 下
- unarchive: src=foo.tgz dest=/var/lib/foo


# 解压一个是存放在远程主机的包，copy=no
- unarchive: src=/tmp/foo.zip dest=/usr/local/bin copy=no


# 从网络上下载一个源包，然后解压上传到远程主机 (2.0 版本)
- unarchive: src=https://example.com/example.zip dest=/usr/local/bin cop
y=no
```

12、synchronize

同步模块，类似于 rsync

参数	必需	默认值	注解
src	yes		源地址路径
dest	yes		目标地址路径
dest_port	no		ssh 连接端口
mode	no	push	mode=push *推送 ansible -> 远程主机* mode=pull *拉取 远程主机 <- ansible*
group	no		文件属组
owner	no		文件属主
archive	no	yes	开启了 recursive(递归), links, perms, times, owner，group 和-D 参数， 设置为 no，那么你将停止很多参数，比如会导致如下目的递归失败，导致无法拉取
delete	no	no	delete=yes 镜像方式同步，使用两边内容一致
compress	no	yes	压缩文件数据在传输
times	no		保持时间属性
perms	no		保持权限
recursive	no		递归

exclude	no		忽略同步文件、目录

示例说明：

```
# 把 ansible 上/some/relative/path 下的文件和目录推送到远程主机上

- synchronize: src=/some/relative/path dest=/some/absolute/path


# 关闭 archive 属性进行推送

- synchronize: src=/some/relative/path dest=/some/absolute/path archive=
no


# 不递归，仅推送目录当下文件

- synchronize: src=some/relative/path dest=/some/absolute/path recursive
=no


# 在 ansible 控制机上保证 src 和 dest 目录内容保持一致

- local_action: synchronize src=/some/relative/path dest=/some/absolute/
path


# 拉取模式，ansible 从远程主机上下载数据到本地路径

- synchronize: mode=pull src=some/relative/path dest=/some/absolute/path


# 镜像同步，ansible src 目录与远程主机 dest 目录保持一致

- synchronize: src=/some/relative/path dest=/some/absolute/path delete=y
es
```

13、filesystem

文件系统创建，磁盘格式化

参数	必需	默认值	注解
dev	yes		目标块设备
fstype	yes		创建文件系统类型 ext2/3/4

force	no		当文件系统已经存在时也允许创建新文件系统
opts	no		传递参数，类似 mkfs 的参数

示例说明：

```
# Create a ext2 filesystem on /dev/sdb1.
- filesystem: fstype=ext2 dev=/dev/sdb1


# Create a ext4 filesystem on /dev/sdb1 and check disk blocks.
- filesystem: fstype=ext4 dev=/dev/sdb1 opts="-cc"
```

14、mount

分区挂载

参数	必需	默认值	注解
name	yes		本地挂载路径，如/mnt/files
src	yes		挂载的设备路径，支持网络路径
fstype	yes		文件系统类型,如 ext2/3/4
opts	no		挂载选项，参考 fstab
state	no	present	present # 挂载，并写入 fstab mounted # 仅挂载 absent/unmounted # 取消挂载

示例说明：

```
# 分区挂载
- mount: name={{ item.name }} src=/dev/{{ item.src }} fstype=ext4 state=
present opts=noatime
  with_items:
       - { name: "/data",src: "sdb1"}
       - { name: "/log", src: "sdb2"}


# nfs 挂载示例
- name: Mount Directory
```

```
  mount: name=/data src=172.20.99.32:/data/. fstype=nfs
         state=present
         opts="rw,bg,rsize=32768,wsize=32768,tcp,hard,intr,noacl,noatim
e,nodiratime"
```

15、lineinfile

搜索匹配替换、插入：若匹配存在的行则替换行，若不存在则插入行

参数	必需	默认值	注解
dest	yes		文件路径位置
insertafter	no	EOF（末行）	接正则表达式；定位到正则匹配所在的行的后一行
insertbefore	no		接正则表达式；定位到正则匹配所在的行的前一行
regexp	no		正则表达搜索匹配内容
line	no		用来替换或者插入的行内容 行存在则替换，行不存在则在文本末尾插入行
mode	no		对文件进行权限变更，如 mode=644,mode=g+x 等
group	no		文件属组
owner	no		文件属主
state	no	present	state=absent 搜索匹配到的行删除
backup	no		backup=yes 修改前对文件进行备份
backrefs	no		正则向后扩展，可用\1 等来表示前面正则匹配内容

示例说明：

```
# 查找以’SELINUX=’开头的行，若存在则替换该行，不存在则在最后插入行（关闭selin
ux）
- lineinfile: dest=/etc/sysconfig/selinux regexp=’^SELINUX=’ line=’SELIN
UX=disabled’ backup=yes


# 配置用户的sudo 权限
```

```
- lineinfile: dest=/etc/sudoers state=present regexp='^%wheel' line='%wh
eel ALL=(ALL) NOPASSWD: ALL'


# 在 Listen 所在的行的后面插入一行’Linster 8080’,插入前备份文件

- lineinfile: dest=/etc/httpd/conf/httpd.conf regexp="^Listen" insertaft
er="^#Listen" line="Listen 8080" owner=apache group=apache mode=0644 bac
kup=yes


# 在’nameserver 8.8.8.8’这行前的一行进行正则匹配"^nameserver 8.8.4.4"，如果
能匹配成功，则以’nameserver 114.114.114.114’内容替换；若匹配不成功，则在‘na
meserver 8.8.8.8’这行前插入’nameserver 114.114.114.114’

- lineinfile: dest=/etc/services regexp="^nameserver 8.8.4.4" insertbefo
re="^nameserver 8.8.8.8$" line="nameserver 114.114.114.114"


# 删除正则匹配成功所在的行

lineinfile: dest=/etc/hosts regexp=’^172.20.99.32 node1’ state=absent


# 正则匹配向后扩展示例

- lineinfile: dest=/opt/jboss-as/bin/standalone.conf regexp='^(.*)Xms(\d
+)m(.*)$' line='\1Xms${xms}m\3' backrefs=yes
```

16、replace

搜索匹配替换，类似于 linux 命令 sed

参数	必需	默认值	注解
dest	yes		文件路径位置
regexp	yes		正则表达搜索匹配内容
replace	yes		用来替换的内容
mode	no		对文件进行权限变更，如 mode=644,mode=g+x 等
group	no		文件属组
owner	no		文件属主
backup	no		backup=yes 修改前对文件进行备份

backrefs	no		正则向后扩展，可用\1 等来表示前面正则匹配内容

示例说明：

```
# 把 old.host.name 替换成 new.host.name

- replace: dest=/etc/hosts regexp='(\s+)old\.host\.name(\s+.*)?$' replac
e='\1new.host.name\2' owner=jdoe group=jdoe mode=644   backup=yes


# 把 Listen 80 和 NameVirtualHost 80 分别替换成 Listen 127.0.0.1:8080 和 Name
VirtualHost 127.0.0.1:8080

    - replace: dest=/etc/apache/ports regexp='^(NameVirtualHost|Listen)\
s+80\s*$' replace='\1 127.0.0.1:8080''
```

17、template

template 使用了 Jinjia2 格式作为文件模版，进行文档内变量的替换的模块。它的每次使用都会被 ansible 标记为”changed”状态，但是 template 只能应用到 playbook

参数	必需	默认值	注解
src	yes		模板源文件的路径
dest	yes		渲染模板生成新文件到远程主机的存放位置
backup	no	no	覆盖前进行备份一次
force	no	yes	强制覆盖
mode	no		文件、目录权限，如 mode=644 或者 mode=“u+x，g+x,o+x”类似 chmod 指令
group	no		文件、目录属组，类似 chown 指令
owner	no		文件、目录属主，类似 chown 指令

示例说明：

```
# 渲染模板，同时备份原来的配置文件

- template: src=/mytemplates/foo.j2 dest=/etc/file.conf owner=bin group=
wheel mode=0644 backkup=yes
```

```
# 渲染模板，且对生成的配置文件修改权限
- template: src=/mytemplates/foo.j2 dest=/etc/file.conf owner=bin group=wheel mode="u=rw,g=r,o=r"
```

19、sysctl

修改 sysctl.conf 配置文件

参数	必需	默认值	注解
name	yes		sysctl.conf 内容是以 kye=value 形式的，name 就是 key，也叫项
value	yes		项值
state	no		对项的操作类型 state=present 修改 state=absent 删除
sysctl_set	no		如果项已经存在，则修改；不存在，则新增
reload	no		reload=yes 重载 sysctl.conf 文件，相当于 sysctl -p

示例说明：

```
# 修改/etc/sysctl.conf 里的 vm.swappiness 值为5
- sysctl: name=vm.swappiness value=5 state=present


# 从/etc/sysctl.conf 里删除 kernel.panic 选项
- sysctl: name=kernel.panic state=absent sysctl_file=/etc/sysctl.conf


# 修改/etc/sysctl.conf 里 net.ipv4.ip_forward 的值为1，但不重新加载 sysctl file 文件
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes


# 修改/etc/sysctl.conf 里 net.ipv4.ip_forward 的值为1，且重新加载 sysctl file 文件
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes state=present reload=yes
```

19、debug

打印输出，特别适合于在 playbook 中打印一个变量结果，看看有没有达到自己预期的效果

示例说明：

```
- debug: msg="system {{ inventory_hostname }} has gateway {{ ansible_def
ault_ipv4.gateway }}"
  when: ansible_default_ipv4.gateway is defined


- name: Display all variables/facts known for a host
  debug: var=hostvars[inventory_hostname] verbosity=4
```

20、wait_for

用于等待到达到某个条件时才继续执行下一个 task

参数	必需	默认值	注解
connect_timeout	no		在下一个事情发生前等待链接的时间，单位是秒，默认是 5 秒
delay	yes		在开始调查之前等待的秒数
host	yes	127.0.0.1	执行这个模块的 host，默认是 127.0.0.1
exclude_hosts	no		list of hosts or ips to ignore when looking for active TCP connections for drained state
path	no		定义文件系统的文件路径存在，则继续
port	no		需要检测的端口号
state	yes	started	判断条件的各种状态，默认是 started 检测一个端口被打开和关闭：started、stopped 检查一条是否活动的连接：drained 检查一个文件或者搜索字符串，如果存在则使用 present or started，若不存则使用 absent
timeout	no	300	wait for 最大等待时间，默认是 300 秒

示例说明：

```
# 先等待 10 秒，然后检测 8000 端口是否被打开;若 300 秒内能检测到，则继续下一个 task,否则超时
- wait_for: port=8000 delay=10


# wait 300 seconds for port 8000 of any IP to close active connections
- wait_for: host=0.0.0.0 port=8000 delay=10 state=drained


# wait 300 seconds for port 8000 of any IP to close active connections
# 排除指定的 ip:10.2.1.2,10.2.1.3
- wait_for: host=0.0.0.0 port=8000 state=drained exclude_hosts=10.2.1.2,
10.2.1.3


# 等待直到文件存在，则继续下一个 task
- wait_for: path=/tmp/foo


# 等待直到文件中存在某一个字符串，则继续下一个 task
- wait_for: path=/tmp/foo search_regex=completed


# 等文件已经被删除后，才继续下一个 task
- wait_for: path=/var/lock/file.lock state=absent


# 等待进程完成
- wait_for: path=/proc/3466/status state=absent
```

三、Ansible 自动化运维：主机列表（Inventory）

ansible 可以连接到多个远程主机执行任务，这些主机信息默认保存在一个文件里面的：/etc/ansible/hosts。同时可以通过配置文件 ansible.cfg 的 `hostfile` 指令修改该默认路径，关于配置文件 ansible.cfg 在*<< Ansible 自动化运维：安装配置 >>*中有讲解

1、简单演示

hosts 文件保存所有远程主机的信息，其中大括号表示一个主机组，一个组可以包含一个或多个主机，当你操作一个组的时候也就表示操作该组下的所有主机：

```
172.20.99.1

172.20.99.2


[webserver]

172.20.99.3:2222          #额外指定连接端口，默认使用 22 端口

172.20.99.4


[dbserver]

172.20.99.5

#为一个主机指定别名

jumper ansible_ssh_port=5555 ansible_ssh_host=172.20.99.50
```

2、通配符：

```
[webservers]

172.20.99.[1:50]      #匹配 172.20.99.1~172.20.99.50


[databases]

db-[a:f].xiaoniu88.com
```

为每个主机指定连接类型和连接用户：

```
[targets]

localhost               ansible_connection=local

other1.xiaoniu88.com    ansible_connection=ssh    ansible_ssh_user=ops

other2.xiaoniu88.com    ansible_connection=ssh    ansible_ssh_user=ops
```

3、定义变量

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=801 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
```

jinja2 的语法格式来使用变量

Ansible 中可以使用 Jinja2 模板引擎来引用定义好的变量

1）template 中使用变量

```
My name is {{ name }}
```

2）playbook 中使用变量

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

4、组包含组

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
```

5、文件定义变量

为 host 和 group 定义一些比较复杂的变量时（如 array、hash），在 hosts 文件中不好实现同时也造成 hosts 文件臃肿不美观，这时可以用单独文件保存 host 和 group 变量，以 YAML 格式书写变量,默认保存目录/etc/ansible/host_vars/和 /etc/ansible/group_vars/下；或者在 playbook 的 play 文件的同级目录。

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
```

6、特定变量指令

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：

```
# 指定主机别名对应的真实 IP，如: s32  ansible_ssh_host=172.20.99.32，随后连接该主机无须指定完整 IP，只需指定 s32 就行

ansible_ssh_host


# 指定连接到这个主机的 ssh 端口，默认 22

ansible_ssh_port


# 连接到该主机的 ssh 用户

ansible_ssh_user


# 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_ssh_pass
```

```
# sudo 密码
ansible_sudo_pass


# 连接类型，可以是 local、ssh 或 paramiko，ansible1.2 之前默认为 paramiko
ansible_connection


#私钥文件路径
ansible_ssh_private_key_file


# 目标系统的 shell 类型，默认为 sh
ansible_shell_type


# python 解释器路径，默认是/usr/bin/python，但是如要要连*BSD 系统的话，就需要该指令修改 python 路径
ansible_python_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
ansible_*_interpreter
```

例子：

```
sz-rjy-share-redis-xiaoniu-31-11  ansible_ssh_port=2200    ansible_ssh_user=ops
sz-rjy-share-redis-xiaoniu-31-12  ansible_ssh_private_key_file=/home/ops/.ssh/id_rsa
```

7、动态 Inventory

Ansible 默认的 inventory 文件/etc/ansible/hosts,如果要增加内容,需要手动编辑添加，比如主机组分类、主机 ip、主机用户、认证密码以及主机变量等等

相对来说 hosts 文件这种形式是***"静态"***的，ansible 还提供一种动态的方法，即 ansible 动态 inventory，这种方法当有新增主机的时候不需要每次都编辑 hosts 文件

ansible 动态 Inventory 是通过调用外部脚本（任何脚本都可以,只要制定返回是 json 格式）生成指定格式的 json，这些 json 就包含了主机相关信息，如主机组、主机、变量等等

Ansible 动态 Inventory 的好处是可以通过这个外部脚本去调度资源管理系统（CMDB）的 API 接口返回获取到主机资源信息。由于 CMDB 资源管理系统能动态对主机资源进行增减和分组，所以相对于 hosts 文件这种主机资源管理方式，它是动态的！当我们执行动态 Inventory 脚本去调度 CMDB 的 API 接口时，都能获取到新增和剔减的主机资源，最后运用到 Ansible 中，很方便！

1）简单演示

由于 Ansible 在接受脚本动态获取主机信息返回的数据是 json 格式,这里我也不从其他系统中取了，通过一段代码打印一段 json 的主机信息来模拟脚本调度 CMDB 返回主机资源的过程,如下：

```
#!/usr/bin/env python
#file: cs_hosts.py
# coding=utf-8
'''这一段代码是模拟数据，实际上要写一段调度 CDMD 的 API 返回主机资源的代码'''
import json
host1ip = ['172.20.99.32','172.20.99.33']        //主机部分必须是一个列表
host2ip = ['172.20.99.34','172.20.99.35']
group1 = 'cs1'
group2 = 'cs2'
# //当主机组里只有 hosts 时（其实还有 vars 和 children,后面会讲解），等效于这个写法：hostdata = {group:host1ip,group2:host2ip}
hostdata = {group1:{"hosts":host1ip},group2:{"hosts":host2ip}}
'''这一段代码是模拟数据，实际上要写一段调度 CDMD 的 API 返回主机资源的代码'''
print json.dumps(hostdata,indent=4)
```

执行脚本：python cs_hosts.py

```
{
```

```
    "cs1": {
        "hosts": [
            "172.20.99.32",
            "172.20.99.33"
        ]
    },
    "cs2": {
        "hosts": [
            "172.20.99.34",
            "172.20.99.35"
        ]
    }
}
```

Ansible 调用脚本:（使用 -i 来指定执行脚本）

```
ansible -i cs_hosts.py cs1 -m shell -a 'uptime'  //cs_hosts.py 必须有执行权限，chmod +x cs_hosts.py
172.20.99.32 | success | rc=0 >>
 23:01pm  up 24 days  8:24,  2 users,  load average: 0.21, 0.35, 0.39
172.20.99.33 | success | rc=0 >>
 23:08pm  up 332 days  5:23,  2 users,  load average: 0.00, 0.01, 0.05
```

脚本返回的主机资源，等同于静态 hosts 文件

```
[cs1]
172.20.99.32
172.20.99.33
[cs2]
172.20.99.34
```

```
172.20.99.35
```

2）复杂示例

在静态主机 hosts 配置文件中，可能会有组变量（vars），组之间的包含（children），这时候要求返回的 json 结构又是不一样的了
如下面这个静态 hosts 文件：

```
[webserver]
host1.xiaoniu88.com
host2.xiaoniu88.com

[webservers:vars]
zoo_server="172.20.50.11:2181"
ntp_server="172.20.50.11"

[webservers:children]
web-9001-user
web-9007-weixin
web-9008-portal

[web-9001-user]
host3.xiaoniu88.com

[web-9001-user:vars]
redis_server="172.20.50.11:26379"

[web-9007-weixin]
host4.xiaoniu88.com

[web-9008-portal]
host5.xiaoniu88.com
```

通过脚本来封装解析成 json（脚本略），如下：

```
{
    "web-9008-portal": {
        "hosts": [
            "host5.xiaoniu88.com"
        ]
    },
    "web-9007-weixin": {
        "hosts": [
            "host4.xiaoniu88.com"
        ]
    },
    "web-9001-user": {
        "hosts": [
            "host3.xiaoniu88.com"
        ],
        "vars": {
            "redis_server": "172.20.50.11:26379"
        }
    },
    "webserver": {
        "hosts": [
            "host1.xiaoniu88.com",
            "host2.xiaoniu88.com"
        ],
        "children": [
            "web-9001-user",
            "web-9007-weixin",
            "web-9008-portal"
```

```
        ],
        "vars": {
            "zoo_server": "172.22.31.101:2181",
            "ntp_server": "172.20.50.11"
        }
    }
}
```

像上面这种复杂的返回格式，一般不会用在ad-hoc环境中，多数会用在ansible-playbook中，因为playbook文件中有时会涉及到vars参数的传参

3）shell脚本

调用的脚本并非一定是py文件，也可以是其他脚本输出的结果，比如是shell，这里做个简单的演示：

```
#!/bin/bash
#file: cs_hosts.sh
cat << EOF
{
    "cs": {
        "hosts": [
            "172.20.99.32",
            "172.20.99.33",
            "172.20.99.34"
        ]
    }
}
EOF
```

Ansible调用动态inventory shell脚本

```
ansible -i cs_hosts.sh cs -m shll -a 'uptime'  //cs_hosts.sh 要有执行权限
```

```
172.20.99.32 | success | rc=0 >>
 00:18am  up 2 days  7:10,  2 users,  load average: 0.00, 0.01, 0.05
172.20.99.33 | success | rc=0 >>
 00:17am  up 2 days  6:32,  2 users,  load average: 0.01, 0.03, 0.05
172.20.99.34 | success | rc=0 >>
 00:11am  up 2 days  9:33,  2 users,  load average: 0.49, 0.42, 0.41
```

四、Ansible 自动化运维：匹配主机（Patterns）

ansible 可以使用多种 `host patterns` 来指定远程主机，用在如下两个地方：

- 命令行 `ansible [host patterns] -m module -a arguments`
- playbook 中通过`- hosts: [host patterns]`来指定要执行该 play 的远程主机

先在 hosts 文件定义几个主机：

```
s12 ansible_ssh_host=172.20.99.12
s13 ansible_ssh_host=172.20.99.13
s14 ansible_ssh_host=172.20.99.14
```

1、all 匹配所有组的主机

```
ansible all -m ping  // 等同于 ansible '*' -m ping
```

2、匹配单个主机或组

在 hosts 文件中添加一个组：

```
[ceshi1]
s12
s13
```

执行命令：

```
[root@opsserver ansible]$ ansible s12 -m ping       #只匹配 12 这台主机
s12 | success >> {
    "changed": false,
    "ping": "pong"
}

[root@opsserver ansible]$ ansible ceshi1 -m ping    #匹配 test1 组内的所有主机
s12 | success >> {
    "changed": false,
    "ping": "pong"
}
s13 | success >> {
    "changed": false,
    "ping": "pong"
}
```

3、：表示匹配多个主机或组

在 hosts 文件中配置两个组：

```
[ceshi1]
s12
s13
[ceshi2]
s13
s14
```

执行命令：

```
[root@opsserver ansible]$ ansible "ceshi1:ceshi2" -m ping              #同时属于多个组的主机只会执行一次
s14 | success >> {
```

```
    "changed": false,
    "ping": "pong"
}
s12 | success >> {
    "changed": false,
    "ping": "pong"
}
s13 | success >> {
    "changed": false,
    "ping": "pong"
}
```

4、！表示排除一个主机组或组

在 hosts 文件中配置两个组：

```
[ceshi1]
s12
s13
s14
[ceshi2]
s13
```

执行命令：

```
[root@opsserver ansible]$ ansible 'ceshi1:!ceshi2' -m ping          #匹配所有在 test1 组却不在 test2 组中的主机
s14 | success >> {
    "changed": false,
    "ping": "pong"
}
s12 | success >> {
    "changed": false,
```

```
    "ping": "pong"
}
```

5、& 表示多个组的交集

在 hosts 文件配置两个组：

```
[ceshi1]
s12
s13
[ceshi2]
s13
s14
```

执行命令：

```
[root@opsserver ansible]$ ansible "ceshi1:&ceshi2" -m ping          #匹配同时在 test1 和 test2 组中的主机
s13 | success >> {
    "changed": false,
    "ping": "pong"
}
```

一个复杂的匹配：

```
webservers:dbservers:&staging:!phoenix
```

首先取出属于 webservers 组和 dbservers 组的所有主机，取出这些主机中同时也属于 staging 组的那部分，然后再去掉不属于 phoenix 组的那些

6、？通配符

```
[root@opsserver ansible]$ ansible 's1?' -m ping          #匹配"s1"开头再接一个任意字符的主机或组
```

```
s12 | success >> {
    "changed": false,
    "ping": "pong"
}
s13 | success >> {
    "changed": false,
    "ping": "pong"
}
s14 | success >> {
    "changed": false,
    "ping": "pong"
}
```

7、~ 表示使用正则表达式

在 hosts 文件配置两个组

```
[ceshi1]
s12
s13
[ceshi2]
s13
s14
```

执行命令：

```
[root@opsserver ansible]$ ansible '~cehsi.*' -m ping        #这里匹配到了ceshi1 和 ceshi2 组
s12 | success >> {
    "changed": false,
    "ping": "pong"
}
s14 | success >> {
```

```
    "changed": false,
    "ping": "pong"
}
s13 | success >> {
    "changed": false,
    "ping": "pong"
}
```

同时要注意，`ansible` 和 `ansible-playbook` 命令还提供了`-l,--limit` 参数，对上面匹配出的结果会再进行一次过滤：

```
$ ansible 's12:s14:s146' -m ping -l 's14*'
s146 | success >> {
    "changed": false,
    "ping": "pong"
}

s14 | success >> {
    "changed": false,
    "ping": "pong"
}
```

五、Ansible 自动化运维：剧本（Playbooks）

playbooks 可以称为是 Ansible 的配置，部署，编排语言。在 playbooks 中，你可以定义远程主机要执行的策略，或者是要执行的某组动作。如：希望远程主机要先安装 httpd,创建一个 vhost,最后启动 httpd 服务。

假如说 Ansible 是一个工具模块的话，那么 playbooks 就是你的设计方案。最基本的，playbooks 可以配置或者部署远程主机；高级些的应用是，可以按照一定的次序执行特定的操作，可以对执行节点的主机进行监控，同时也可监控其负载均衡情况。

Ansible 的 playbook 就如同 salt 的 state，一个 playbook 就是一个 YAML 文件，所以 playbook 文件一般都以.yml 结尾，写 playbook 不需要复杂的 YAML 语法，所以也不用单独去学 YAML 语法。此外 playbook 和模板文件（template 模块）还使用 jinja2 语法语法实现高级功能（后面逐一讲到）

一个 playbook 文件由一个或多个 play 组成，每个 play 定义了在一个或多个远程主机上执行的一系列的 task，task 是按照顺序执行的，且每个 task 会都在指定的所有远程主机上执行，然后再执行下一个 task，其中每个 task 一般就是调用一个 ansible 的模块，如调用 copy 模块复制文件到远程主机或调用 shell 模块执行命令；但是，在执行 task 过程中，某个主机的任务执行失败则退出，结束整个流程（除非设置错误过滤忽 ignore_errors:yes）

每个模块都是等幂的，意思是当你再次运行的时候，他们只做他们必须要做的操作（改变某些文件状态等），以使系统达到所需的状态。这样的话，当你重复运行多次相同的 playbooks 时也是非常安全的

每个任务都应该有一个名称，这个名称会在运行 playbook 的时候输出，如果没有指定名称，那么对主机做的操作行为将会被用来输出到屏幕

1、简单演示

例子：只带有一个 play 的 playbook 文件 apache.yml：

```
--- #任何 playbook 文件(其实就是 yaml 文件)都要以这个开头
- hosts: webservers   #对 webservers 主机组下的所有主机进行操作
  remote_user: ops #连接到远程主机的用户，不是必须，如果没有指定则使用运行的用户
  sudo: yes            #以 sudo 模式运行该 play
  sudo_user: root           #sudo 到哪个用户，默认为 root，如果 sudo 到该用户需要密码，则在执行 ansible-playbook 的时候指定-K 选项来输入 sudo 密码
  vars:                     #为该 play 定义两个变量
    http_port: 80
    max_clients: 200
  tasks:                    #开始定义 tasks
```

```
  - name: httpd install   #这是 task 的名字,会打印到屏幕，如果没有 name,对主机的操作行为将用来输出到屏幕上
    yum: pkg=httpd state=latest
    tags:                 #给该 task 打一个标签
      - apache
  - name: copy the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:           #提供 watch 功能，这里当 apache 配置文件改变时，就调用 handlers 中名为"restart apache"的 task 来重启 apache
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:           #notify 通知这里的 task 执行，谨记：定义在 handlers 下的 task 只有在 notify 触发的时候才会执行
    - name: restart apache
      service: name=httpd state=restarted
```

执行该 playbook 文件：

```
$ ansible-playbook apache.yml
```

运行 playbook 的 yml 文件，ansible 会连接到 webservers 组下的所有远程主机执行上面定义的 tasks，然后返回执行结果。输出也非常容易看懂，通过红黄绿三种颜色标明了不同的执行结果，如下：

```
红色：表示有 task 执行失败
黄色：表示执行了且改变了远程主机状态
绿色：表示执行成功
```

2、patterns

playbook 中的每个 play 首先要定义的就是该 play 要在哪些主机上执行，如上面示例的

```
---
```

```
- hosts: webservers
```

可以指定单个主机、主机组和以：连接的多个主机或主机组，具体见 Ansible 自动化运维：Host Patterns

3、remote_user

remote_user: {{ user }} 表示 ansible 连接到远程主机使用的用户，没有指定则使用运行 ansible-playbook 的用户

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

4、sudo

sudo: yes 表示远程主机要以 sudo 权限执行 tasks，不光可以对整个 play 生效，还可以针对单个 task 生效（remote_user 指令也一样）：

```
---
- hosts: webservers
  remote_user: root
  tasks:
    - shell: cmd
      remote_user: user1
    - service: name=httpd state=started
      sudo: yes            #只 sudo 运行该 task
      sudo_user: user2     #sudo 到 user2 用户，如果 sudo 需要密码记得为 ansible-playbook 命令提供"-K"选项
```

5、vars

定义该 play 的变量，play 下的所有 tasks 都能使用它的变量

```
vars:
    http_port: 80
```

```
    max_clients: 200
tasks:
  - name: copy the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
```

关于变量更多的使用方法，请参考<< Ansible 自动化运维：变量 >>章节

6、tasks

tasks 由一系列 task 组成，多个 task 按照顺序执行，默认所有匹配主机都执行完一个 task 后才会移动到下一个 task 执行，默认如果所有主机都执行某个 task 失败，则 ansible 会中断执行流程并打印错误消息，这里都是说的默认，也就是说这些行为都是可自定义的，以后会讲到这些主题

上面提到过一个 task 其实就是执行一个模块，ansible 中的模块是幂等的(idempotent)，也就是说多次执行同一个 task，只有在状态发生改变后才会真的去执行，通过下面的例子讲解：

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

该 task 用于保证 apache 服务器处于运行状态，如果我多次执行该 task 且 apache 一直处于运行状态的话，则该 task 其实什么都不会做。这样做的好处就是你可以随意执行你的 playbook 文件，不用担心改变远程主机上的内容(除非状态有变化必须要修改)

注：这里要特别提一下 command 和 shell 模块，这两个模块用于在远程主机上执行命令，每次调用这两个模块都会重新执行一遍命令，且返回的状态是 changed，一般来说不会有什么影响，不过 ansible 也提供了 creates 参数来将这两个模块实现等幂 idempotent，具体请参考<< *Ansible 自动化运维：常用模块* >>中的 command 模块相关说明

7、handlers

handlers 是和 notify 指令搭配使用的，作用是当一个 task 执行修改了远程主机状态时就通知(notify)一个 handlers 中的 task 执行，一般用在修改了远程主机的服务的配置文件然后调用 handlers 中的对应 task 去重启该服务，例子如下：

```
tasks:
```

```
  - 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=httpd state=restarted
```

这里只有模板文件 `template.j2` 发生了变化，也就是真正修改了远程主机的 `/etc/foo.conf` 文件后，才会触发 `handlers` 中的两个 task 执行，其他情况 `handlers` 中的 task 都是不会运行的

8、参数分行

如果模块的参数需要使用多个，一行显得太长可以分开多行写：

```
tasks:
  - name: copy facts file to  /etc/ansible/fact.d/
    copy: src=file/get_version.fact dest=/etc/ansible/fact.d/
          owner=root group=root mode=0777
```

9、命令行参数

ansible-playbook 命令参数

```
Usage: ansible-playbook playbook.yml


Options:
## ssh 连接的用户名
  -u REMOTE_USER, --user=REMOTE_USER
```

```
## ssh 登录密码
  -k, --ask-pass

## sudo 运行
  -s, --sudo

##sudo 到哪个用户，默认为root
  -U SUDO_USER, --sudo-user=SUDO_USER

## sudo 密码
  -K, --ask-sudo-pass

## ssh 连接超时，默认 10 秒
  -T TIMEOUT, --timeout=TIMEOUT

## 指定该参数后，执行 playbook 文件不会真正去执行，而是模拟执行一遍，然后输出本次执行会对远程主机造成的修改
  -C, --check
## 连接类型(default=smart)
  -c CONNECTION

## 设置额外的变量如：key=value 形式 或者 YAML or JSON，以空格分隔变量，或用多个-e
  -e EXTRA_VARS, --extra-vars=EXTRA_VARS

## 进程并发处理，默认 5
  -f FORKS, --forks=FORKS

## 指定 hosts 文件路径，默认 default=/etc/ansible/hosts
```

```
  -i INVENTORY, --inventory-file=INVENTORY

## 指定一个pattern，对- hosts:匹配到的主机再过滤一次
  -l SUBSET, --limit=SUBSET

## 只打印有哪些主机会执行这个playbook 文件，不是实际执行该playbook
  --list-hosts

## 列出该playbook 中会被执行的task
  --list-tasks

## 私钥路径
  --private-key=PRIVATE_KEY_FILE

## 同一时间只执行一个task，每个task 执行前都会提示确认一遍
  --step

## 只检测playbook 文件语法是否有问题，不会执行该playbook
  --syntax-check

## 当play 和task 的tag 为该参数指定的值时才执行，多个tag 以逗号分隔
  -t TAGS, --tags=TAGS

## 当play 和task 的tag 不匹配该参数指定的值时，才执行
  --skip-tags=SKIP_TAGS

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

  -v, --verbose
```

六、Ansible 自动化运维：角色与包含（Roles and Include）

你可以将所有东西都放到一个 playbook 文件中，但是随着文件越来越大，你修改起来也越来越麻烦。这时候可以把一些 play、task 或 handler 放到其他文件中，然后通过 include 指令包含进来，这样做完全没问题，但是 ansible 还提供了一个更好的解决方法，也就是 roles，下面分别讲解

1、include 包含

playbook 可以包含其他 playbook 文件、task 文件和 handler 文件

1）包含 task 文件

如果有多个 play 都需要几个相同的 task，在每个 play 中都写一遍这些 task 就不明智了，聪明做法是将这些 task 单独放到一个文件中，格式如下：

```
---
# possibly saved as tasks/foo.yml
- name: placeholder foo
  command: /bin/foo

- name: placeholder bar
  command: /bin/bar
```

然后在需要这些 task 的 play 中通过 include 包含上面的 tasks/foo.yml：

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

还可以向 include 传递变量，如你部署了多个 wordpress 实例，通过向相同的 wordprss.yml 文件传递不同的值来区分实例：

```
tasks:
  - include: wordpress.yml user=lipeibin     #在 foo.yml 可以通过{{ user }} 来使用这些变量
```

```
  - include: wordpress.yml user=liuzhiwei
  - include: wordpress.yml user=qubaoquan
```

如果使用 ansible1.4 及以上版本，include 还可以写成字典格式：

```
tasks:
 - { include: wordpress.yml, user: lipeibin, ssh_keys: [ 'keys/one.txt',
 'keys/two.txt' ] }   #ssh_keys 是一个列表
```

从 ansible1.0 开始，还可以用如下格式向 include 传递变量：

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

2）包含 handler 文件

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

在 play 末尾包含上面的 handler 文件：

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

3）包含 playbook 文件

```
- 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   #这些 playbook 文件中也至少定义了一个 play
- include: webservers.yml
- include: dbservers.yml
```

2、roles 角色

1) roles 组织结构

roles 用来组织 playbook 结构，以多层目录和文件将 playbook 更好的组织在一起，一个经过 roles 组织的 playbook 目录结构如下：

```
webservers.yml
roles/
  webservers/       # 这层目录是role 角色的名字，下面的子目录都不是必须提供的，没有的目录会自动忽略，不会出现问题，所以你可以只有 tasks 子目录也没问题
      files/       # 这个目录是存放的是需要复制到远程主机的文件
      templates/    # 模板文件存放的目录
      tasks/       # 顾名思义，任务目录
      handlers/    # 触发器目录
      vars/        # 定义这个 role 所有 tasks 用到的变量
      meta/        # role 依赖关系定义
```

在 playbook 文件中包含 webservers 这个 role,webservers.yml 内容如下：

```
---
# file: webservers.yml
- hosts: all
  roles:
     - webservers
```

简单演示源码安装 Nginx 服务器，如下：

```
├──── roles
|    └──── nginx
|        ├──── files
|        |    └──── nginx-1.6.1.tar.gz
|        ├──── handlers
|        |    └──── main.yml
|        ├──── tasks
|        |    ├──── basic.yml
|        |    ├──── main.yml
|        |    └──── nginx.yml
|        ├──── templates
|        |    └──── nginx.conf.j2
|        └──── vars
|            └──── main.yml
└──── xiaoniu-nginx.yml
```

注：有些子目录并没有用到 ，如 meta，因为没有其它依赖关 roles 可以不用创建

```
---
# file: xiaoniu-nginx.yml
```

```
- hosts: nginxservers
  remote_user: root
  roles:
     - { role: nginx,tags:nginx }
```

```
---
# file: roles/nginx/vars/main.yml
worker_processes: 4
worker_connections: 65535
keepalive_timeout: 60
```

```
---
# file: roles/nginx/tasks/main.yml
- include: basic.yml
- include: nginx.yml
```

```
---
# file: roles/nginx/tasks/basic.yml
- name: install basic parket for nginx
  yum: name={{ item }} state=latest enablerepo=epel
  with_items:
     - pcre
     - pcre-devel
```

```
---
# file: roles/nginx/tasks/nginx.yml
```

```
- name: unzip nginx parket to remote hosts
  unarchive: src=nginx-1.6.1.tar.gz dest=/tmp


- name: make install nginx
  shell: cd /tmp/nginx-1.6.1/ && ./configure --prefix=/usr/local/nginx --with-http_ssl_module --with-http_stub_status_module &&  make && make install


- name: copy nginx config file to remote hosts
  template: src=nginx.conf.j2 dest=/usr/local/nginx/conf/nginx.conf
  notify:
      - restart nginx service


- name: ensure nginx service is running
  service: name=nginx state=started
```

```
---
# file: roles/nginx/handlers/main.yml
- name: restart nginx service
  service: name=nginx state=restarted
```

2) roles 目录说明

假如有一个 play 包含了一个叫"x"的 role，则：

- /path/roles/x/tasks/main.yml 中的 tasks 都将自动添加到该 play 中
- /path/roles/x/handlers/main.yml 中的 handlers 都将添加到该 play 中
- /path/roles/x/vars/main.yml 中的所有变量都将自动添加到该 play 中
- /path/roles/x/meta/main.yml 中的所有 role 依赖关系都将自动添加到 roles 列表
- /path/roles/x/defaults/main.yml 中为一些默认变量值，具有最低优先权，在没有其他任何地方指定该变量的值时，才会用到默认变量值

- task 中的 copy 模块和 script 模块会自动从/path/roles/x/files 寻找文件，也就是根本不需要指定文件绝对路径或相对路径，如 src=foo.txt 则自动转换为 /path/roles/x/files/foo.txt
- task 中的 template 模块会自动从/path/roles/x/templates/中加载模板文件，无需指定绝对路径或相对路径
- 通过 include 包含文件会自动从/path/roles/x/tasks/中加载文件，无需指定绝对路径或相对路径

注：从 ansible1.4 开始可以在 ansible.cfg 配置文件中通过 roles_path 指令自定义 roles 的路径

vim /etc/ansible/ansible.cfg

```
roles_path=/data/svndata/Code_base/other/svn_project/ansible/atom/roles
```

如同 include 一样，也可以为 role 传递变量，格式如下：

```
---
- hosts: webservers
  roles:
    - common
    - { role: foo_app_instance, dir: '/opt/a',  port: 5000 }     #在foo_app_instance这个role的task文件和模板文件中通过{{ dir }}和{{ port }}来使用变量
    - { role: foo_app_instance, dir: '/opt/b',  port: 5001 }
```

如果一个 play 中，既有 tasks 也有 roles，那么 roles 会先于 tasks 执行，可以通过 pre_tasks 和 post_tasks 指令指定某些 task 先于或晚于 roles 执行：

```
---
- hosts: webservers


  pre_tasks:
    - shell: echo 'hello'      #最先执行
  roles:
    - { role: some_role }      #第二个执行
```

```
tasks:
  - shell: echo 'still busy'

post_tasks:
  - shell: echo 'goodbye'     #最后执行
```

3）roles 依赖关系

可以在一个 role 的 meta/main.yml 中定义该 role 依赖其他的 role，然后调用该 role 的时候，会自动去拉取其他依赖的 role，如一个名为 myapp 的 role 的 meta/main.yml 文件如下：

```
---
dependencies:
  - { role: common, some_parameter: 3 }    #向依赖的 role 传递变量
  - { role: apache, port: 80 }
  - { role: postgres, dbname: blarg, other_parameter: 12 }
```

那么这些 role 的执行顺序为：

```
common
apache
postgres
myapp       #会先把 myapp 依赖的其他所有 role 执行完再执行自己
```

默认一个 role 只能被其他 role 依赖一次，多次依赖不会执行，但是可以通过 allow_duplicates 指令来改变这种行为：
名为 car 的 role 的 meta/main.yml：

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

```
  - { role: wheel, n: 4 }
```

名为 `wheel` 的 role 的 `meta/main.yml`：

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

执行顺序如下：

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

七、Ansible 自动化运维：变量（Variables）

其实 ansible 中变量的用法在其他几篇文章会分别讲解，如下：

1、hosts 文件中定义变量

hosts 文件中定义变量，*<< Ansible 自动化运维：主机列表（Inventory） >>*篇章中有讲解

2、group_vars 和 host_vars

group_vars 和 host_vars 目录下变量文件使用，*<<Ansible 自动化运维：主机列表（Inventory）>>*篇章中有讲解

3、vars_files

vars_files指令导入变量文件，在*<<Ansible 自动化运维：条件判断>>*中的*条件导入*篇章中有讲解

4、setup 模块：facts

setup模块收集主机信息，在*<<Ansible 自动化运维：条件判断>>*中 *setup 模块*篇章中有讲解

5、register 注册变量

register指令注册的变量，该指令两种用法在*<<Ansible 自动化运维：条件判断>>*和*<<Ansible 自动化运维：循环语句（Loops）>>*篇章中有讲解

6、命令行传递变量

```
ansible-playbook release.yml --extra-vars "version=1.23.45
other_variable=foo"
```

也可以向 playbooks 传递主机组和连接的远程用户

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

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

ansible1.2 中，可以以 JSON 格式传递变量

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

ansible1.3 中可以使用 JSON 文件传递变量

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

上面的方式也可以传递一个 YAML 格式的变量文件

7、变量过滤处理

jinja2 过滤器和 ansible 独有的过滤器可以对变量进行某些处理，请参考 jinjia2 模板设计说明文档：*http://jinja.pocoo.org/docs/dev/templates 和 <<Ansible 自动化运维：条件判断>>中的过滤器相关内容*

8、复杂变量定义

```
---
## file:roles/xxx/vars/main.yml
users:
  - name: alice
    authorized:
      - /tmp/alice/onekey.pub
      - /tmp/alice/twokey.pub
    mysql:
        password: mysql-password
        hosts:
          - "%"
          - "127.0.0.1"
          - "::1"
          - "localhost"
        privs:
          - "*.*:SELECT"
          - "DB1.*:ALL"
  - name: bob
    authorized:
      - /tmp/bob/id_rsa.pub
    mysql:
        password: other-mysql-password
```

```
      hosts:
        - "db1"
      privs:
        - "*.*:SELECT"
        - "DB2.*:ALL"
```

类似于 python 的变量定义：

```
users=[
   {'name':'alice',
     'authorized':['/tmp/alice/onekey.pub','/tmp/alice/twokey.pub'],
     'mysql':{'password':'mysql-password',
              'hosts':["%","127.0.0.1","::1","localhost"]},
              'privs':["*.*:SELECT","DB1.*:ALL"]
    },

   {'name':' bob',
    'authorized':['/tmp/bob/id_rsa.pub'],
    'mysql':{'password':'other-mysql-password',
              'hosts':["db1",]},
              'privs':["*.*:SELECT","DB2.*:ALL"]
    }
      ]
```

八、Ansible 自动化运维：条件判断

这节讲如何控制 playbook 的执行流，记得前面说过 playbook 和模板文件中可以使用 `jinja2` 语法么，这节就会大量用到了

1、setup 模块

首先，要讲下 setup 这个模块，作用类似 salt 的 grains，用于获取远程服务器的信息（返回的是 *JSON 格式*的数据集），这些获取到的信息在 template 模块定义的模板文件和 playbook 文件中可以直接使用，该模块获取的结果又叫 facts，包括远程主机的 IP 地址，和操作系统类型，磁盘相关信息等等;facts 会经常在条件语句及模板中使用

1）facts 输出结果打印

```
ansible 172.20.99.4 -m setup
```


```
172.20.99.4 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.99.4"
        ],
        "ansible_all_ipv6_addresses": [],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "04/14/2014",
        "ansible_bios_version": "6.00",
        "ansible_cmdline": {
            "KEYBOARDTYPE": "pc",
            "KEYTABLE": "us",
            "LANG": "zh_CN.UTF-8",
            "crashkernel": "129M@0M",
            "quiet": true,
            "rd_NO_DM": true,
            "rd_NO_LUKS": true,
            "rd_NO_LVM": true,
            "rd_NO_MD": true,
            "rhgb": true,
            "ro": true,
            "root": "UUID=40b85a41-90a3-4a66-a623-6a80814aded7"
        },
        "ansible_date_time": {
```

```
            "date": "2016-05-04",
            "day": "04",
            "epoch": "1462359134",
            "hour": "18",
            "iso8601": "2016-05-04T10:52:14Z",
            "iso8601_micro": "2016-05-04T10:52:14.551369Z",
            "minute": "52",
            "month": "05",
            "second": "14",
            "time": "18:52:14",
            "tz": "CST",
            "tz_offset": "+0800",
            "weekday": "Wednesday",
            "year": "2016"
        },
        "ansible_default_ipv4": {
            "address": "172.20.99.4",
            "alias": "eth0",
            "gateway": "172.20.99.254",
            "interface": "eth0",
            "macaddress": "00:50:56:ac:b5:c8",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "172.20.99.0",
            "type": "ether"
        },
        "ansible_default_ipv6": {},
        "ansible_devices": {
            "sda": {
                "holders": [],
```

```
            "host": "SCSI storage controller: LSI Logic / Symbios Logi
c 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)",
            "model": "Virtual disk",
            "partitions": {
                "sda1": {
                    "sectors": "409600",
                    "sectorsize": 512,
                    "size": "200.00 MB",
                    "start": "2048"
                },
                "sda2": {
                    "sectors": "16777216",
                    "sectorsize": 512,
                    "size": "8.00 GB",
                    "start": "411648"
                },
                "sda3": {
                    "sectors": "66697216",
                    "sectorsize": 512,
                    "size": "31.80 GB",
                    "start": "17188864"
                }
            },
            "removable": "0",
            "rotational": "1",
            "scheduler_mode": "cfq",
            "sectors": "83886080",
            "sectorsize": "512",
            "size": "40.00 GB",
            "support_discard": "0",
```

```
            "vendor": "VMware"
        },
        "sdb": {
            "holders": [],
            "host": "SCSI storage controller: LSI Logic / Symbios Logi
c 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)",
            "model": "Virtual disk",
            "partitions": {
                "sdb1": {
                    "sectors": "41929587",
                    "sectorsize": 512,
                    "size": "19.99 GB",
                    "start": "63"
                }
            },
            "removable": "0",
            "rotational": "1",
            "scheduler_mode": "cfq",
            "sectors": "41943040",
            "sectorsize": "512",
            "size": "20.00 GB",
            "support_discard": "0",
            "vendor": "VMware"
        },
        "sr0": {
            "holders": [],
            "host": "IDE interface: Intel Corporation 82371AB/EB/MB P
IIX4 IDE (rev 01)",
            "model": "VMware IDE CDR10",
            "partitions": {},
```

```
            "removable": "1",
            "rotational": "1",
            "scheduler_mode": "cfq",
            "sectors": "2097151",
            "sectorsize": "512",
            "size": "1024.00 MB",
            "support_discard": "0",
            "vendor": "NECVMWar"
        }
    },
    "ansible_distribution": "CentOS",
    "ansible_distribution_major_version": "6",
    "ansible_distribution_release": "Final",
    "ansible_distribution_version": "6.7",
    "ansible_domain": "",
    "ansible_env": {
        "CVS_RSH": "ssh",
        "G_BROKEN_FILENAMES": "1",
        "HISTFILESIZE": "9999",
        "HISTORY_FILE": "/var/log/.history/history.log",
        "HISTORY_SHELL": "/var/log/.history/.history_shell",
        "HISTORY_VAR": "/var/log/.history/.history_var",
        "HISTSIZE": "3000",
        "HISTTIMEFORMAT": "[ root ] - [ %Y-%m-%d %H:%M:%S ] - ",
        "HOME": "/root",
        "LANG": "C",
        "LC_CTYPE": "C",
        "LESSOPEN": "||/usr/bin/lesspipe.sh %s",
        "LOGIN_IP": "172.20.99.3",
        "LOGIN_TIME": "2016-05-04 18:52",
```

```
            "LOGIN_USER": "root",
            "LOGNAME": "root",
            "MAIL": "/var/mail/root",
            "PATH": "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:
/usr/bin",
            "PROMPT_COMMAND": "{ \\\nTHIS_HIST_ID=`history 1 | awk \"{pri
nt \\\\$1}\"`; \\\nif [ ${THIS_HIST_ID}x != ${LAST_HIST_ID}x ]; then \\\
nEXEC_TIME=`history 1 | awk -F\" - \" \"{print \\\\$2}\" | sed -e \"s/\\
[ //\" -e \"s/ \\]//\"`; \\\nEXEC_COMMAND=`history 1 | awk -F\" - \" \"{p
rint \\\\$3}\"` \\\nEXEC_USER=`id -un`; \\\nSERVERIP=`/sbin/ifconfig | g
rep \"Bcast\" | awk \"{print \\\\$2}\" | awk -F\":\" \"{print \\\\$2}\"
| grep -Ev \"^192.168|127.0.0.1\"`; \\\nEXEC_PWD=`pwd`; \\\necho \"[$LOG
IN_USER] - [$LOGIN_IP] - [$EXEC_USER] - [$EXEC_TIME] - $EXEC_COMMAND\"
 >> $HISTORY_FILE; \\\necho \"SERVERIP=$SERVERIP\" > $HISTORY_VAR; \\\ne
cho \"LOGIN_USER=$LOGIN_USER\" >> $HISTORY_VAR; \\\necho \"LOGIN_IP=$LOG
IN_IP\" >> $HISTORY_VAR; \\\necho \"LOGIN_TIME=$LOGIN_TIME\" >> $HISTORY
_VAR; \\\necho \"EXEC_TIME=$EXEC_TIME\" >> $HISTORY_VAR; \\\necho \"EXEC
_USER=$EXEC_USER\" >> $HISTORY_VAR; \\\necho \"EXEC_PWD=$EXEC_PWD\" >>
$HISTORY_VAR; \\\necho \"EXEC_COMMAND=$EXEC_COMMAND\" >> $HISTORY_VAR; \
\\n/bin/sh $HISTORY_SHELL; \\\nLAST_HIST_ID=$THIS_HIST_ID; \\\nfi; }",
            "PWD": "/root",
            "SHELL": "/bin/bash",
            "SHLVL": "2",
            "SSH_CLIENT": "172.20.99.3 35905 22",
            "SSH_CONNECTION": "172.20.99.3 35905 172.20.99.4 22",
            "SSH_TTY": "/dev/pts/7",
            "TERM": "xterm",
            "USER": "root",
            "_": "/usr/bin/python"
        },
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "172.20.99.4",
```

```
            "netmask": "255.255.255.0",
            "network": "172.20.99.0"
        },
        "macaddress": "00:50:56:ac:b5:c8",
        "module": "vmxnet3",
        "mtu": 1500,
        "promisc": false,
        "type": "ether"
    },
    "ansible_fips": false,
    "ansible_form_factor": "Other",
    "ansible_fqdn": "TMP-OPS-99-4",
    "ansible_hostname": "TMP-OPS-99-4",
    "ansible_interfaces": [
        "lo",
        "eth0"
    ],
    "ansible_kernel": "2.6.32-573.el6.x86_64",
    "ansible_lo": {
        "active": true,
        "device": "lo",
        "ipv4": {
            "address": "127.0.0.1",
            "netmask": "255.0.0.0",
            "network": "127.0.0.0"
        },
        "mtu": 65536,
        "promisc": false,
        "type": "loopback"
    },
```

```
        "ansible_machine": "x86_64",
        "ansible_machine_id": "48e4b91173b91010c86df7d200000009",
        "ansible_memfree_mb": 7075,
        "ansible_memory_mb": {
            "nocache": {
                "free": 7667,
                "used": 205
            },
            "real": {
                "free": 7075,
                "total": 7872,
                "used": 797
            },
            "swap": {
                "cached": 0,
                "free": 8191,
                "total": 8191,
                "used": 0
            }
        },
        "ansible_memtotal_mb": 7872,
        "ansible_mounts": [
            {
                "device": "/dev/sda3",
                "fstype": "ext4",
                "mount": "/",
                "options": "rw",
                "size_available": 29413617664,
                "size_total": 33478791168,
                "uuid": "40b85a41-90a3-4a66-a623-6a80814aded7"
```

```
            },
            {
                "device": "/dev/sda1",
                "fstype": "ext4",
                "mount": "/boot",
                "options": "rw",
                "size_available": 146999296,
                "size_total": 198902784,
                "uuid": "72a3ee29-0985-4ff5-977b-59093d2eb244"
            },
            {
                "device": "/dev/sdb1",
                "fstype": "ext4",
                "mount": "/data",
                "options": "rw",
                "size_available": 19877273600,
                "size_total": 20996739072,
                "uuid": "1ee9b64a-9dc8-4a61-a15c-7a81339dbed0"
            }
        ],
        "ansible_nodename": "TMP-OPS-99-4",
        "ansible_os_family": "RedHat",
        "ansible_pkg_mgr": "yum",
        "ansible_processor": [
            "GenuineIntel",
            "Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz",
            "GenuineIntel",
            "Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz",
            "GenuineIntel",
            "Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz",
```

```
            "GenuineIntel",
            "Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz"
        ],
        "ansible_processor_cores": 2,
        "ansible_processor_count": 2,
        "ansible_processor_threads_per_core": 1,
        "ansible_processor_vcpus": 4,
        "ansible_product_name": "VMware Virtual Platform",
        "ansible_product_serial": "VMware-42 2c 0b 6e bf b6 68 73-3f 44 5
f e7 cf 56 83 60",
        "ansible_product_uuid": "422C0B6E-BFB6-6873-3F44-5FE7CF568360",
        "ansible_product_version": "None",
        "ansible_python_version": "2.6.6",
        "ansible_selinux": {
            "status": "disabled"
        },
        "ansible_ssh_host_key_dsa_public": "AAAAB3NzaC1kc3MAAACBAJCvgNTP
oduSvDM59qwJwBd2kIUObaZ3HEOXGf6piqsI80qSTUvjxmFJxm/BaMiidjV4bFYxR2gd6bM
7YAaNxG8NZGoNrhZpvnA/Z2NjIf/tJknegGsEwPmrcEKBXY04vqE91exuiq/dfMImRU6tRR
SQZoj/wTFRhyDD71iMI34vAAAAFQCHaFYRmkM1bn9PcFYlSG1fVLLKdwAAAIBTZCijcu+rG
+NSelHim4vnERzJflXpNeZhzqxd7mrVkudO1AcjsIp97mgb4+Spj7DpmDTMOVua//4rOjOk
HlMjlPwWWkANGq6qyrG/xOFrzgtFVPwQbW7fUy8WB11e2RQepqUeoG1i6FkiFCbz6WEJ8wL
MN3lBEJdctpOip+s+MQAAAIEAg4TcbnzB9iqwwtYhSxPjcJuGzdgCRrbR3X/naaiDnaITxC
Xpirq+Kuxfb23vnUVXloBplXaPADIkGcdeK52Ywgpu92Ck3krH3WyIvj4+336dHTTsV/Npw
zSdhcwWsGXooGPlsZTrcbd5KVF6rXn4a+KByxPz9T6kpJIu5NthT04=",
        "ansible_ssh_host_key_rsa_public": "AAAAB3NzaC1yc2EAAAABIwAAAQEA
rOga+j4rR3nbS8q7QbQjyxdYaOb3bD5ECJfqxjAKPGoD5f85gCbJZLU3L/mUqWCiHhZiZnZ
plX8UQjvKz1v4M18VvgicRRePJN58vSsgLmuAmCaU6zwiK5F8AMEE5/VRhQGmanV5QlGYrs
eV+4FbpmB2yvEbmwxiOHaOPJntZBKp+JwKfjHV78iLlhnr39kIOjxpLlXrn8EMONQnT9us3
fgCQVmq4hkrFZYK0hOAItaTFIl+vEnls1eKc0UgGIYaW+qDGWOblimaqio3vctezKx7/2V2
qYWxw+jvznMUK6nXxjOpI6h2DIcNDzxi5N4FlAOskuQnNd1d39lSnEO0Qw==",
        "ansible_swapfree_mb": 8191,
        "ansible_swaptotal_mb": 8191,
        "ansible_system": "Linux",
```

```
        "ansible_system_vendor": "VMware, Inc.",
        "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
}
```

2）facts 输出结果引用

playbook 或者模板中可以通过`{{ var }}`来使用这些获取到的信息，假如我有一个模板文件 `os.j2`：

```
{% if ansible_distribution == "FreeBSD" %}
        FreeBSD
{% elif ansible_distribution == "CentOS" %}
        CentOS
{% endif %}
hostname: {{ ansible_hostname }}
```

```
{{ ansible_hostname }}            # 系统的主机名
{{ ansible_eth0.ipv4.address }}  ## ip 地址；等同 {{ ansible_eth0["ipv4"]["address"] }}；其它也是类似
```

```
{{ inventory_hostname }}          ## inventory 主机别名，若没定义别名，则是 ip
{{ ansible_ distribution }}       ## 操作系统的版本
```

注：[] 的里 key 值可以引入变量，如果 ansible_eth0["ipv4"][type]，当 type="address"表示 IP 地址，若 type="gateway"表示网关

3）几个魔法变量

```
{{ play_hosts }}                  ## 在当前 play 范围中可用的一组主机名
{{ group_names }}                 ##当前主机所在所有群组的列表(数组)
{{ groups }}                      ## inventory 中所有群组(主机)的列表.可用于枚举群组中的所有主机
```

演示说明：

```
{% if 'webserver' in group_names %}
{% for host in groups[' webserver '] %}
{{ hostvars[host].ansible_eth0.ipv4.address }}
{% endfor %}
{% endif %}
```

4）使用其它主机 facts

hostvars 可以让你访问其它主机的 fact

```
{{ hostvars['inventory_hostname'].ansible_distribution }}
{{ hostvars['ns1'].ansible_ distribution }}
```

应用实例

```
tasks:
- name: add hosts
```

```
  lineinfile: dest=/etc/hosts line="{{ hostvars[item].ansible_eth0.ipv4.
address }} {{ hostvars[item].ansible_hostname }}"

  with_items: groups['all']

  # with_items: play_hosts


tasks:

- name: add zookeeper's cluster to /etc/hosts

  lineinfile: dest=/etc/hosts line="{{ hostvars[item[1]].ansible_eth0.ip
v4.address }} s{{ item[0]+1 }}.af88.com.cn"

  with_indexed_items: play_hosts
```

5）本地 facts

通常情况下 facts 都是 ansible 的 setup 模块自动获取的。用户也可以以一种更简便的方法来自定义远程主机的 facts，使用 *facts.d* 机制

如果远程主机系统上存在 *etc/ansible/facts.d* 目录，这个目录下的以.fact 为后缀的文件，里面的内容可以是 *JSON* 格式，或者 *ini* 格式书写；或者是一个可以返回 *JSON* 格式数据的*可执行文件*，都可以用来提供本地 facts 信息

I、ini 格式 facts

在远程主机创建一个*/etc/ansible/facts.d/preferences.fact* 的文件，内容如下：

```
[general]

a=1

b=2
```

```
ansible webservers -m setup -a "filter=ansible_local"
```

```
172.20.99.32 | success >> {

"ansible_facts": {

        "ansible_local": {

                "preferences": {

                        "general": {
```

```
                                "a": "1",
                                "b": "2",
                                    }
                                 }
                            }
                    },
"changed": false
}
```

可以在 playbooks 或者模板中这样使用获取到的信息

```
{{ ansible_local.preferences.general.a }}    ## =1
{{ ansible_local.preferences.general.b }}    ## =2
```

II、shell 返回 facts

```
#!/bin/sh
#file: nginx_server_info.fact
worker_processes=`cat /proc/cpuinfo |grep ^processor|wc -l`
if [ $worker_processes -eq 4 ];then
        worker_cpu_affinity="0001 0010 0100 1000"
elif [ $worker_processes -eq 8 ];then
        worker_cpu_affinity="00000001 00000010 00000100 00001000 0001000
0 00100000 01000000 10000000"
fi
cat << EOF
{
"worker_processes":"$worker_processes",
"worker_cpu_affinity":"$worker_cpu_affinity"
}
EOF
```

```
ansible 172.20.99.32, all -m setup -a "filter=ansible_local"
```

注：必须得赋予执行权限 chmod +x /etc/ansible/facts.d/nginx_server_info.fact

```
172.20.99.32 | success >> {
    "ansible_facts": {
        "ansible_local": {
            "nginx_server_info": {
                "worker_cpu_affinity": "00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000",
                "worker_processes": "8"
            }
        }
    },
    "changed": false
}
```

引用：

```
{{ ansible_local.nginx_server_info.worker_processes }}
{{ ansible_local.nginx_server_info.worker_cpu_affinity }}
```

III、python 返回 facts

```
#!/usr/bin/python
#file:zookeeper_cluster.fact

import sys,os,json
hosts="172.20.99.32:172.20.99.33:172.20.99.34"

def return_hosts(hosts):
  hosts_dict={}
```

```
  hosts_dict["ip_list"]=hosts.split(':')
  return json.dumps(hosts_dict,indent=4)

print retrun_hosts(hosts)
```

playbook 中引用

```
---
- hosts: webservers
  tasks:
  - name: create directory for ansible custom facts
    file: state=directory recurse=yes path=/etc/ansible/facts.d
  - name: copy fact file to /etc/ansible/facts.d
    copy: src=zookeeper_cluster.fact dest=/etc/ansible/facts.d mode=777
  - name: reload facts after adding custom fact
    setup: filter=ansible_local
- name: add zookeeper cluster to hosts
  lineinfile: dest=/etc/hosts line="{{ hostvars[item].ansible_eth0.ipv4.
address }} {{ hostvars[item].ansible_hostname }}"
  with_items: ansible_local. zookeeper_cluster.hosts_dict.ip_list
```

ansible_local:

```
"ansible_facts": {
       "ansible_local": {
           " hosts_dict": {
              "ip_list": [
                  "172.20.99.32",
                  "172.20.99.33",
                  "172.20.99.34"
              ]
```

```
            }
        }
    },
    "changed": false
}
```

6）facts 缓存

Ansible 目前支持两种缓存方式： redis 和 json 文件

I、redis

在配置文件 ansible.cfg 中设置如下

```
[defaults]
gathering = smart      # 其值还可以设置为 explicit
fact_caching = redis
fact_caching_timeout = 86400
```

注：目前来说还不支持设置 redis 访问密码和为 redis 设置访问接口的功能。在不久的将来可能会加入支持

II、json

在配置文件 ansible.cfg 中设置如下

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_location = /path/to/cachedir
fact_caching_timeout = 86400    # 单位为秒
```

注：fact_caching_location 指定一个可写的目录，若不存在的话 ansible 会尝试创建它

7）关闭 facts

如果你确信不需要主机的任何 facts 信息，而且对远程主机都了解的很清楚，那么可以将其关闭。远程操作主机较多的时候，关闭 facts 会提升 ansible 的性能。

只需要在 play 中设置如下：

```
- hosts: whatever
  gather_facts: no
```

2、when 语句

when 语句的作用是只有匹配指定条件，才执行 task，when 后面使用 jinja2 的表达方式

1）简单演示

例子：只关闭操作系统为 Debian 的服务器

```
- hosts: all
  tasks:
    - name: "shutdown Debian flavored systems"
      command: /sbin/shutdown -t now          #只有Debian 系统才会执行该 command
      when: ansible_os_family == "Debian"     #这里 ansible_os_family 变量就是通过 setup 模块获取的
```

when 支持 or 和 and 这些多条件表达式，如：

```
    when: ansible_os_family == "RedHat" and ansible_lsb.major_release|int >= 6
```

其它

2）过滤器（filter）

过滤器，ansible 也提供了一些自己的过滤器，也可以使用 jiaja2 模板的过滤器

I、ansible 内置过滤器

示例：

```
## ansible 内置过滤器 failed、success、skipped、changed
  - command: /bin/false
    register: result
    ignore_errors: True          #忽略该 task 的错误
  - command: /bin/cmd1
    when: result|failed          #通过结果判断上一个 task 如果执行失败，则执行该 task
  - command: /bin/cmd2
    when: result|success         #通过结果判断上一个 task 如果执行成功，则执行该 task
  - command: /bin/cmd3
    when: result|changed         #通过结果判断上一个 task 如果执行成功且改变了主机状态，则执行该 task
```

II、jinjia2 模板过滤器

```
tasks:
  - name: get nginx start’s state
    shell: ps aux|grep nginx|grep -v grep|wc -l
    register: cmd_result
  - name: start nginx service
    service: name=nginx state=started
    when: cmd_result.stdout|int == 0   #使用 jinja2 的"int"过滤器


vars:
    nginx_version=1.6.1
tasks:
 - name: update nginx
   yum: name=nginx state=latest
   when: nginx_version.split(‘.’)[0] == ‘1’
```

jinjia2 模板的过滤器还有很多，上面只是展示了其中一个，详细的请参考 *http://jinja.pocoo.org/docs/dev/templates* 和 *<< Ansible 自动化运维：Jinjia2 模板>>* 中的 jinjia2 过滤器相关内容

3）自定义的变量判断过滤

```
vars:
  epic: true
tasks:
  - shell: echo "I've got '{{ epic }}' and am not afraid to use it!"
    when: epic is defined              #变量已定义

  - fail: msg="Bailing out: this play requires 'epic'"
    when: epic is not defined          #变量未定义

  - shell: echo "This certainly is epic!"
    when: epic                         #变量值为真

  - shell: echo "This certainly isn't epic!"
    when: not epic                     #变量值为假
```

4）when 在循环中检查过滤

当 when 用于循环中时，是对列表中的每一项都进行检查：

```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

5）when 在 role 和 include 应用

在 roles 和 include 中使用 when 指令，when 可以用在包含一个 playbook 文件上，且当包含一个 task 文件时，会对 task 文件中的每个 task 都会使用一次 when 判断

```
- hosts: all
  tasks:
    - include: tasks/sometasks.yml
      when: "'reticulating splines' in output"
  roles:
     - { role: debian_stock_config, when: ansible_os_family == 'Debian' }
         #首先判断远程主机是否是Debian，是的话才会导入这个role
```

当执行这个 play 的时候，输出中可能会出现很多的 skipped，这些就是经过 when 的条件判断不符合，跳过执行的输出

6）条件导入

```
- hosts: all
  remote_user: root
  vars_files:    #用于导入变量文件
    - [ "vars/{{ ansible_os_family }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: make sure apache is running
    service: name={{ apache }} state=running
```

根据不同的系统导入不同变量文件，如果是 CentOS 系统则首先导入 vars/CentOS.yml 文件（Debian 系统则会首先导入 vars/Debian.yml），如果该文件不存在则导入 vars/os_defaults.yml 文件，如果两个文件都不存在则生成一个错误，vars/CentOS.yml 文件内容如下：

```
---
# for vars/CentOS.yml
apache: httpd
somethingelse: 42
```

基于变量选择文件或模板

```
- name: template a file
  template: src={{ item }} dest=/etc/myapp/foo.conf
  with_first_found:
    - files:
       - {{ ansible_distribution }}.conf   #CentOS 系统会使用CentOS.conf 文件，Debian 系统会使用Debian.conf 文件
       - default.conf
      paths:
       - /search_location_one/somedir/
       - /opt/other_location/somedir/
```

3、register 注册变量

1）register 作用说明

register 用于注册一个变量，保存 task 任务执行后输出的结果，这个变量的值可以在后面的 task、when 语句或模板文件中使用。如果该指令用在循环中会有不同效果，请看 *<<ansible 自动化运维：Loops>>*中关于 register 的讲解

```
- shell: /bin/pwd
  register: pwd_result
```

此时变量 pwd_result 的结果为：*(注：变量返回的结果是一个json 格式的数据)*

```
{
    'changed': True,
    'end': '2014-02-23 12:02:51.982893',
    'cmd': ['/bin/pwd'],
    'start': '2014-02-23 12:02:51.980191',
    'delta': '0:00:00.002702',
    'stderr': '',
    'rc': 0,                        #这个就是命令返回状态，非0 表示执行失败
    'invocation': {'module_name': 'command', 'module_args': '/bin/pwd'},

    'stdout': '/home/sapser',     #以一个字符串保存命令结果
```

```
    'stdout_lines': ['/home/sapser']    #以列表保存命令结果
}
```

task 中使用该变量：

```
tasks:
- debug: msg="{{pwd_result}}"
  when: pwd_result.rc == 0
```

2）register 与 when 结合应用

可以对 register 注册变量的输出值进行一些处理，若是字符串，可以查找搜索关键词、分割等；若是列表，可以应用于循环中，或者转换成字符串（join 方法：list |join() ）等；然后应用于 when 语句中，做条件表达式判断。

I、字符串

```
# 查找/etc/motd 文件中是否存在“hi”,否存在则echo 一条语句
- shell: cat /etc/motd
  register: motd_contents
- shell: echo "motd contains the word hi"
  when: motd_contents.stdout.find('hi') != -1

# 如果远程主机的hosts 文件不存在，就传一个file 过去
- stat: path=/etc/hosts
  register: hosts_file
- copy: src=/path/to/local/file dest=/path/to/remote/file
  when: hosts_file.stat.exists == false

# 如果当前php 版本为7.0，则执行为php 降级处理
- shell: php --version
  register: php_version
```

```
- shell: yum -y downgrde php*
  when: "'7.0' in php_version.stdout"
```

扩展：对于 register 变量输出的值，可以转化为一个 list（或者已经是一个 list），然后可以在后面任务的"with_items"中使用

"stdout_lines"已经是一个列表了，可以直接在后面 with_items 调用；当然如果你喜欢也可以调用 "变量名.stdout.split()" 来字段切割生成一个列表，如下

II、列表

循环处理 register 注册变量的输出值（值是一个列表）

```
- name: registered variable usage as a with_items list
  hosts: all
  tasks:
      - name: retrieve the list of home directories
        command: ls /home
        register: home_dirs
      - name: add home dirs to the backup spooler
        file: path=/mnt/{{ item }} src=/home/{{ item }} state=link
        with_items: home_dirs.stdout_lines      #等同于 with_items: home_dirs.stdout.split()
```

3）register 与 with_items 结合应用

在循环中使用 register，输出的数据结构会变得不同，注册变量会包含一个 `results` 属性，该属性的值是一个列表，包含该模块本次循环的所有响应数据

例子演示：

```
## 创建用户
---
- name: Create System User
  hosts: all
```

```
  tasks:
  - name: Check The User Exist's State
    shell: grep "^{{ item }}\b" /etc/passwd || /bin/true
    register: reg
    with_items:
    - lipeibin
    - qinlinkai
    changed_when: False


  - name: Create The User That Needs But Don't Exist
      user: name={{ item.item }} state=present
      with_items: reg.results
      when: item.stdout ==""
```

其它示例：

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
- name: Fail if return code is not 0
  debug: msg="The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0
  with_items: echo.results
```

register 注册的 echo 输保存的结果为：

```
{
    "changed": true,
    "msg": "All items completed",
```

```
"results": [
    {
        "changed": true,
        "cmd": "echo \"one\" ",
        "delta": "0:00:00.003110",
        "end": "2016-04-25 12:00:05.187153",
        "invocation": {
            "module_args": "echo \"one\"",
            "module_name": "shell"
        },
        "item": "one",
        "rc": 0,
        "start": "2016-04-25 12:00:05.184043",
        "stderr": "",
        "stdout": "one"
    },
    {
        "changed": true,
        "cmd": "echo \"two\" ",
        "delta": "0:00:00.002920",
        "end": "2016-04-25 12:00:05.245502",
        "invocation": {
            "module_args": "echo \"two\"",
            "module_name": "shell"
        },
        "item": "two",
        "rc": 0,
        "start": "2016-04-25 12:00:05.242582",
        "stderr": "",
        "stdout": "two"
```

```
      }
   ]}
```

九、Ansible 自动化运维：循环语句（Loops）

当遇到一些重复的任务，比如通过 `yum` 模块安装多个包，为每个包写一个 task 的做法低效且冗余，可以在 playbook 中使用循环来解决这类问题。

1、标准循环：with_items

标准循环，最常用到的就是它，`with_items` 可以用于迭代一个列表或字典，通过`{{ item }}`获取每次迭代的值，如通过一条 task 创建多个用户：

```
- name: add several users
  user: name={{ item }} state=present groups=wheel  #创建用户
  with_items:      #这里还可以直接写成 with_items: ['user1', 'user2', 'user3']
     - user1
     - user2
     - user3
```

迭代一个简单字典：

```
- name: add several users
  user: name={{ item.name }} state=present groups={{ item.groups }}
  with_items:
    - { name: 'user1', groups: 'wheel' }
    - { name: 'user2', groups: 'root' }
```

2、嵌套循环：with_nested

```
- name: give users access to multiple databases
```

```
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append_privs=y
es password=foo
  with_nested:
    - [ 'alice', 'bob', 'eve' ]
    - ['clientdb', 'employeedb', 'providerdb']
```

上面例子不容易看懂，我们通过 `debug` 模块写另一个例子测试：

```
---
- hosts: 127.0.0.1
  gather_facts: no
  tasks:
    - debug: msg="{{ item[0] }} - {{ item[1] }}"
      with_nested:
        - ['alice', 'bob']
        - ['clientdb', 'employeedb', 'providerdb']
```

执行：ansible-playbook test.yml

```
PLAY [127.0.0.1] ****************************************************

TASK: [debug msg="{{item[0]}} - {{item[1]}}"] ************************

ok: [127.0.0.1] => (item=['alice', 'clientdb']) => {
    "item": [         #每次循环 item 变量是一个双元素列表
        "alice",
        "clientdb"
    ],
    "msg": "alice - clientdb"}
ok: [127.0.0.1] => (item=['alice', 'employeedb']) => {
    "item": [
        "alice",
        "employeedb"
```

```
    ],
    "msg": "alice - employeedb"}
ok: [127.0.0.1] => (item=['alice', 'providerdb']) => {
    "item": [
        "alice",
        "providerdb"
    ],
    "msg": "alice - providerdb"}
ok: [127.0.0.1] => (item=['bob', 'clientdb']) => {
    "item": [
        "bob",
        "clientdb"
    ],
    "msg": "bob - clientdb"}
ok: [127.0.0.1] => (item=['bob', 'employeedb']) => {
    "item": [
        "bob",
        "employeedb"
    ],
    "msg": "bob - employeedb"}
ok: [127.0.0.1] => (item=['bob', 'providerdb']) => {
    "item": [
        "bob",
        "providerdb"
    ],
    "msg": "bob - providerdb"}

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

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

其实相当于两个 for 循环：

```
for user in [ 'alice', 'bob', 'eve' ]:
    for db in ['clientdb', 'employeedb', 'providerdb']:
        mysql_user: name=user priv=db.*:ALL append_privs=yes password=foo
```

3、字典循环：with_dict

迭代字典，接受一个字典类型的值，`{{ item.key }}`是字典的键，`{{ item.value }}`是字典的值，字典值还可以是一个子字典

在变量文件定义一个复杂的字典变量：

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

playbook 文件 `test.yml` 内容：

```
----
  hosts: 127.0.0.1
  gather_facts: no
  tasks:
    - name: Print phone records
```

```
    debug: msg="User {{ item.key }} is {{ item.value.name }} ({{ item.v
alue.telephone }})"

    with_dict: users
```

执行：ansible-playbook test.yml

```
TASK: [Print phone records] ***************************************

ok: [127.0.0.1] => (item={'value': {'name': 'Bob Bananarama', 'telephone
': '987-654-3210'}, 'key': 'bob'}) => {

    "item": {

        "key": "bob",

        "value": {

            "name": "Bob Bananarama",

            "telephone": "987-654-3210"

        }

    },

    "msg": "User bob is Bob Bananarama (987-654-3210)"}

ok: [127.0.0.1] => (item={'value': {'name': 'Alice Appleworth', 'telepho
ne': '123-456-7890'}, 'key': 'alice'}) => {

    "item": {

        "key": "alice",

        "value": {

            "name": "Alice Appleworth",

            "telephone": "123-456-7890"

        }

    },

    "msg": "User alice is Alice Appleworth (123-456-7890)"}
```

4、目录循环：with_fileglob

匹配指定目录下的所有文件(非递归)，或指定目录下和 pattern 匹配的文件

```
---
- hosts: all
  tasks:
    - name: first ensure our target directory exists
      file: dest=/etc/fooapp state=directory


    - name: copy each file over that matches the given pattern
      copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
      with_fileglob:
        - /playbooks/files/fooapp/*          #这里如果是"*.py"，就只会复制该目录下所有py 文件
```

5、迭代并行：with_together

迭代并行数据集

```
---
- hosts: '172.20.99.32'
  tasks:
  - debug: msg="{{ item.0 }} and {{ item.1 }}"
    with_together:         #相当于: for item in [('a',1), ('b', 2), ('c', 3), ('d', 4)]
      - ['a', 'b', 'c', 'd']
      - [1, 2, 3, 4]
```

6、迭代子元素：with_subelements

```
---
- hosts: 127.0.0.1
  gather_facts: no
  vars:
```

```
  users:            #定义一个字典变量
    - name: alice
      group:
        - ops1
        - ops2
      authorized:    #列表
        - /tmp/alice/onekey.pub
        - /tmp/alice/twokey.pub
    - name: bob
      group:
        - ops1
        - ops2
      authorized:
        - /tmp/bob/id_rsa.pub
tasks:
  - debug: msg="{{ item.0.name }},{{ item.1 }}"
    with_subelements:
      - users
      - authorized
```

执行 ansible-playbook test.yml

```
PLAY [172.20.99.4] ****************************************************

GATHERING FACTS ********************************************************

ok: [172.20.99.4]

TASK: [debug msg="{{ item.0.name }},{{ item.1 }}"] *********************

ok: [172.20.99.32] => (item=({'group': ['ops1', 'ops2'], 'name': 'alice
'}, '/tmp/alice/onekey.pub')) => {

    "item": [

        {
```

```
            "group": [
                "ops1",
                "ops2"
            ],
            "name": "alice"
        },
        "/tmp/alice/onekey.pub"
    ],
    "msg": "alice,/tmp/alice/onekey.pub"
}
ok: [172.20.99.32] => (item=({'group': ['ops1', 'ops2'], 'name': 'alice
'}, '/tmp/alice/twokey.pub')) => {
    "item": [
        {
            "group": [
                "ops1",
                "ops2"
            ],
            "name": "alice"
        },
        "/tmp/alice/twokey.pub"
    ],
    "msg": "alice,/tmp/alice/twokey.pub"
}
ok: [172.20.99.32] => (item=({'group': ['ops3', 'ops4'], 'name': 'bob'},
 '/tmp/bob/id_rsa.pub')) => {
    "item": [
        {
            "group": [
                "ops3",
```

```
                "ops4"
            ],
            "name": "bob"
        },
        "/tmp/bob/id_rsa.pub"
    ],
    "msg": "bob,/tmp/bob/id_rsa.pub"
}
PLAY RECAP *********************************************************************
172.20.99.32: ok=2    changed=0    unreachable=0    failed=0
```

7、序列循环：with_sequence

按照升序生成一个数字序列，可以指定开始(start)、结束(stop)和步长(stride)

```
---
- hosts: all
  tasks:
    - user: name={{ item }} state=present groups=evens
      with_sequence: start=1 end=32 format=testuser%02d          #创建用户testuser01、testuser02、...、testuser32


    - file: dest=/var/stuff/{{ item }} state=directory
      with_sequence: start=0 end=20 stride=5              #stride 表示步长step，也就是间隔


    # a simpler way to use the sequence plugin create 4 groups
    - group: name=group{{ item }} state=present
      with_sequence: count=4
```

8、随机选择：with_random_choice

从列表中随机选择一项

```
- debug: msg={{ item }}
  with_random_choice:                #每次都随机选择一项
     - "go through the door"
     - "drink from the goblet"
     - "press the red button"
     - "do nothing"
```

9、等待循环：until

不停重试，直到某些条件完成

```
- shell: /usr/bin/foo
  register: result          #保存命令结果，和 until 搭配使用时，result 还有个 attempts 属性，表示当前重试次数
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

不停重新执行命令，直到命令输出中出现"all systems go"，或者重试次数达到 5 次间隔 10 秒，默认重试 3 次间隔 5 秒

10、查找文件：with_first_found

查找并返回列表中第一个存在的文件

简略版：

```
- name: INTERFACES | Create Ansible header for /etc/network/interfaces
  template: src={{ item }} dest=/etc/foo.conf
  with_first_found:          #第一个文件不存在，就去找第二个文件
    - "{{ansible_virtualization_type}}_foo.conf"
    - "default_foo.conf"
```

高级版：

```
- 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
```

11、按行迭代：with_lines

按行迭代输出

```
---
- hosts: "172.20.99.32"
  tasks:
  - debug: msg="stdout - {{ item }}"
    with_lines: /bin/echo -e "111\n222\n333"          #按行迭代 echo 命令的
输出
```

通过之前学过的 `with_items` 也能达到同样的效果

```
---
- hosts: "172.20.99.32"
  tasks:
  - shell: /bin/echo -e "111\n222\n333"
```

```
    register: results
  - debug:  msg="stdout - {{ item }}"
    with_items: results.stdout_lines
```

12、索引和值：with_indexed_items

迭代一个列表，获取每一项的索引和值，`{{ item.0 }}`是索引，`{{ item.1 }}`是值

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

13、列表循环：with_flattened

展开嵌套列表并循环，如`[1, [2, [3]]]`展开为`[1, 2, 3]`并循环这个列表

```
---
- hosts: "172.20.99.32"
  vars:
    packages_base:
      - ['foo-package', 'bar-package']
    packages_apps:
      - [['one-package', 'two-package']]
      - [['red-package'], ['blue-package']]
  tasks:
    - name: indexed loop demo
      debug: msg="value - {{ item }}"
      with_flattened:
        - packages_base
        - packages_apps
```

14、register 注册变量

注册一个标量，用来保存命令 task 的输出。在循环中使用 register，数据结构会变得不同，注册变量会包含一个 results 属性，该属性的值是一个列表，包含该模块本次循环的所有响应数据：

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
```

此时 echo 变量的内容为：

```
{
    "changed": true,
    "msg": "All items completed",
    "results": [
        {
            "changed": true,
            "cmd": "echo \"one\" ",
            "delta": "0:00:00.003110",
            "end": "2016-04-25 12:00:05.187153",
            "invocation": {
                "module_args": "echo \"one\"",
                "module_name": "shell"
            },
            "item": "one",
            "rc": 0,
            "start": "2016-04-25 12:00:05.184043",
            "stderr": "",
            "stdout": "one"
```

```
        },
        {
            "changed": true,
            "cmd": "echo \"two\" ",
            "delta": "0:00:00.002920",
            "end": "2016-04-25 12:00:05.245502",
            "invocation": {
                "module_args": "echo \"two\"",
                "module_name": "shell"
            },
            "item": "two",
            "rc": 0,
            "start": "2016-04-25 12:00:05.242582",
            "stderr": "",
            "stdout": "two"
        }
    ]}
```

在随后一个 task 中迭代 `echo.results` 这个列表：

```
- name: Fail if return code is not 0
  debug: msg="The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0
  with_items: echo.results
```

十、Ansible 自动化运维：标签（Tags）

ansible 中可以对 `play`、`role`、`include`、`task` 打一个 tag(标签)，然后：

- 当命令 ansible-playbook 有`-t` 参数时，只会执行`-t` 指定的 tag

- 当命令 ansible-playbook 有`--skip-tags`参数时，则除了`--skip-tags`指定的 tag 外，执行其他所有

1、基本应用

```
---
- hosts: 192.168.0.105
  tags:
    - one              #为一个 play 打 tag
  tasks:
    - name: exec ifconfig
      command: /sbin/ifconfig
      tags:
        - exec_ifconfig            #为一个 task 打 tag
```

2、对 include 语句打 tags

```
---
- hosts: 192.168.0.105
  gather_facts: no
  tasks:
    - include: foo.yml
      tags:      #方式一
        - one
    - include: bar.yml tags=two    #方式二
```

上面两种打 tags 的方式都可以，此时：

```
ansible-playbook test.yml           #执行 foo.yml 和 bar.yml
ansible-playbook test.yml -t one    #只执行 foo.yml
ansible-playbook test.yml -t two    #只执行 bar.yml
```

3、同一对象打打多个 tags

```
---
- hosts: 192.168.0.105
  gather_facts: no
  tasks:
    - include: foo.yml tags=one,two
    - include: bar.yml tags=two
```

可以对同一对象打多个 tags，只要`-t`指定了其中一个 tags，就会执行该对象：

```
ansible-playbook test.yml                #执行 foo.yml 和 bar.yml
ansible-playbook test.yml -t one         #只执行 foo.yml
ansible-playbook test.yml -t two         #因为两个 include 都有 two 这个标签，
所以 foo.yml 和 bar.yml 都会执行
```

4、对 roles 打 tags

```
---
- hosts: 192.168.0.105
  gather_facts: no
  roles:
    - {role: foo, tags: one}
    - {role: bar, tags: [one,two]} #同时对 bar 打两个 tag
```

则：

```
ansible-playbook test.yml          #执行 foo 和 bar 两个 role
ansible-playbook test.yml -t two   #只执行 bar
ansible-playbook test.yml -t one   #两个 role 都有 one 标签，所以 foo 和 bar 都
会执行
```

5、对多个对象打同一个 tags

```
---
- hosts: 192.168.0.105
  gather_facts: no
  tasks:
    - name: exec ifconfig
      command: /sbin/ifconfig
      tags:
        - exec_cmd

    - name: exec ls
      command: /bin/ls
      tags:
        - exec_cmd

    - name: debug_test one
      debug: msg="test1"
      tags:
        - debug_test

    - name: debug_test two
      debug: msg="test2"
      tags:
        - debug_test
```

如上面，可以把执行 command 模块的 task 都打上 `exec_cmd` 标签，则：

```
ansible-playbook test1.yml              #执行所有 tasks
ansible-playbook test1.yml -t exec_cmd   #只执行 tag 为 exec_cmd 的 tasks
```

十一、Ansible 自动化运维：错误处理

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，通常遇到错误就中断 playbook 的执行，这些默认行为都是可以改变的

1、忽略错误

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

`command` 和 `shell` 模块执行的命令如果返回非零状态码则 ansible 判定这两个模块执行失败，可以通过 `ignore_errors: yes` 忽略返回状态码

```
- name: this will not be counted as a failure
  command: shell_cmd || /bin/true
```

2、自定义错误：failed_when

不依靠返回状态码，而是自定义错误判定条件

```
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "command_result.stderr and 'FAILED' in command_result.stderr"
```

这个例子中，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是否有 `FAILED` 字符串，如果包含此字符串则判定命令执行失败。

实际中的应用：

```
---
- hosts: 192.168.0.105
  gather_facts: no
  sudo: yes
```

```
  sudo_user: coc
  tasks:
    - shell: ps -U coc|grep _coc_game|wc -l              #计算指定进程数，如果有三个进程则为启动成功
      register: cmd_result
      failed_when: "'3' not in cmd_result.stdout"       #如果进程数量不为3，则表示启动失败
      failed_when: cmd_result.stdout != 3               #经过测试，用jinja2 语法写也是可以的
```

3、自定义状态：changed_when

ansible 会自动判断模块执行状态，command、shell 及其他模块如果修改了远程主机状态则被判定为 changed 状态，不过可以自己决定达到 changed 状态的条件

```
tasks:
  - shell: /usr/bin/billybass --mode="take me to the river"
    register: bass_result
    changed_when: "bass_result.rc != 2" #命令返回状态码不为2，则为changed状态


  # this will never report 'changed' status
  - shell: wall 'beep'
    changed_when: False                 #永远也不会达到changed 状态
```

十二、Ansible 自动化运维：交互提示输入（Prompts）

1、简单演示

用在执行 playbook 的过程中，提示用户输入，并接收用户的输入

```
---
- hosts: 127.0.0.1
  vars_prompt:
```

```
    name: "what is your name?"
    quest: "what is your quest?"
    favcolor: "what is your favorite color?"
  tasks:
    - debug: msg="name - {{name}}   quest - {{quest}}    favcolor - {{fav
color}}"
```

执行：

```
$ ansible-playbook prompt.yml
what is your favorite color?: : red  #注意问题提示的顺序是反过来的
what is your quest?: : ansible
what is your name?: : lipeibin

PLAY [127.0.0.1] ***********************************************************
TASK: [debug msg="name - lipeibin   quest - ansible    favcolor - red"] *
ok: [127.0.0.1] => {
    "msg": "name - sapser   quest - ansible    favcolor - red"
}
PLAY RECAP *****************************************************************
127.0.0.1                  : ok=2    changed=0    unreachable=0    failed=
0
```

2、设置默认值

```
---
- hosts: 127.0.0.1
  gather_facts: no
  vars_prompt:
    - name: "release_version"                    #变量名
      prompt: "product release version"          #提示
      default: "1.0"                             #默认值
```

```
  tasks:
    - debug: msg="release_version - {{release_version}}"
```

执行：

```
$ ansible-playbook prompt.yml
product release version [1.0]: 1.2


PLAY [127.0.0.1] ***********************************************************
TASK: [debug msg="release_version - 1.2"] ******************************
ok: [127.0.0.1] => {
    "msg": "release_version - 1.2"
}
PLAY RECAP *****************************************************************
127.0.0.1                  : ok=1    changed=0    unreachable=0    failed=
0
```

3、输入隐藏

```
---
- hosts: 127.0.0.1
  gather_facts: no
  vars_prompt:
    - name: "release_version"
      prompt: "product release version"
      default: "1.0"
      private: no
    - name: "passwd"
      prompt: "Enter password"
      private: yes
  tasks:
```

```
    - debug: msg="release_version - {{release_version}}      passwd - {{p
asswd}}"
```

执行：

```
$ ansible-playbook prompt.yml
product release version [1.0]: 2.5
Enter password:

PLAY [127.0.0.1] ***********************************************************
TASK: [debug msg="release_version - 2.5      passwd - 123123"] **********
ok: [127.0.0.1] => {
    "msg": "release_version - 2.5      passwd - 123123"
}

PLAY RECAP *****************************************************************
127.0.0.1                  : ok=1    changed=0    unreachable=0    failed=
0
```

4、加密模块

加密用户的输入，需要 python 第三方模块 passlib

```
vars_prompt:
  - name: "my_password2"
    prompt: "Enter password2"
    private: yes
    encrypt: "md5_crypt"
    confirm: yes
    salt_size: 7
```

Passlib 支持多种加密方式：

- `des_crypt` - DES Crypt
- `bsdi_crypt` - BSDi Crypt

- `bigcrypt` - BigCrypt
- `crypt16` - Crypt16
- `md5_crypt` - MD5 Crypt
- `bcrypt` - BCrypt
- `sha1_crypt` - SHA-1 Crypt
- `sun_md5_crypt` - Sun MD5 Crypt
- `sha256_crypt` - SHA-256 Crypt
- `sha512_crypt` - SHA-512 Crypt
- `apr_md5_crypt` - Apache's MD5-Crypt variant
- `phpass` - PHPass' Portable Hash
- `pbkdf2_digest` - Generic PBKDF2 Hashes
- `cta_pbkdf2_sha1` - Cryptacular's PBKDF2 hash
- `dlitz_pbkdf2_sha1` - Dwayne Litzenberger's PBKDF2 hash
- `scram` - SCRAM Hash
- `bsd_nthash` - FreeBSD's MCF-compatible nthash encoding

十三、Ansible 自动化运维：Lookup 函数

在 playbooks 中可以使用一个名为 `lookup()`的函数，该函数用于 ansible 从外部资源访问数据，根据第一个参数的不同，该函数具有不同的功能，典型的就是读取外部文件内容。注意 `lookup()`只在本地执行，而不是在远程主机上执行

1、file 类型

1）获取文件内容

第一个参数为 `file`，表示获取外部文件内容

```
- hosts: all

  vars:

     contents: "{{ lookup('file', '/etc/foo.txt') }}"      #将值保存到变量中，参数都要引号引起来，不然出错

  tasks:

     - debug: msg="the value of foo.txt is {{ contents }}"


     - debug: msg="the value of foo.txt is {{ lookup('file', '/etc/foo.txt') }}"      #直接使用
```

```
通过 lookup 函数做 ansible 与远程主机之间主机信任关系
- hosts: all
  tasks:
  - authorized_key:
    user=ops
    key="{{ lookup('file', '/home/ops/.ssh/id_rsa.pub') }}"
    #path="/home/ops/.ssh/authorized_keys"
```

2）主机信任配置

```
ansible-playbook xiaoniu-authorized-key.yml -e "hosts=all user=ops" -k
```

```
---
# file: xiaoniu-authorized-key.yml
- hosts: {{ hosts }}
  remote_user: root
  tasks:
  - name: create user on ansible server as remote_user
    user: name="{{ user }}"
          generate_ssh_key=yes
          ssh_key_type=rsa
          ssh_key_bits=2048
          ssh_key_file=.ssh/id_rsa
          state=present
    delegate_to: localhost
    tags: authorized-key

  - name: create user on remote host
    user: name="{{ user }}" state=present
    tags: authorized-key
```

```
  - name: copy id_rsa.pub to remote host for authorized trust

    authorized_key: user="{{ user }}" key="{{ lookup("file","/home/"+use
r+"/.ssh/id_rsa.pub") }}"

    tags: authorized-key
```

2、password 类型

第一个参数为 `password`，表示生成一个随机明文密码，并存储到指定文件中，生成的密码包括大小写字母、数字和`.,:-_`，默认密码长度为 20 个字符，该长度可以通过传递一个额外参数 `length=<length>`修改

```
---
- hosts: 127.0.0.1
  gather_facts: no
  tasks:
    - debug: msg="password - {{ lookup('password', '/tmp/random_pass.txt
 length=10') }}"
```

`cat /tmp/random_pass.txt`

```
ejL.Ho_.mb
```

测试：`ansible-playbook test.yml`

```
PLAY [127.0.0.1] ****************************************************


TASK: [debug msg="password - ejL.Ho_.mb"] ***************************
ok: [127.0.0.1] => {
    "msg": "password - ejL.Ho_.mb"}


PLAY RECAP **********************************************************
127.0.0.1: ok=1    changed=0    unreachable=0    failed=0
```

如果用来保存密码的文件已经存在，则不会往里写入任何数据，且会读取文件已有内容作为密码，如果文件存在且为空，则返回一个空字符串作为密码

除了 `length=<length>`外，从 ansible1.4 开始还加入了 `chars=<chars>`参数，用于自定义生成密码的字符集，而不是默认的大小写字母、数字和`.,:-_`

```
---
- hosts: 127.0.0.1
  gather_facts: no
  tasks:
    #create a random password using only ascii letters:
    - debug: msg="password - {{ lookup('password', '/tmp/passfile1 chars
=ascii_letters') }}"

    #create a random password using only digits:
    - debug: msg="password - {{ lookup('password', '/tmp/passfile2 chars
=digits') }}"

    #create a random password using many different char sets:
    - debug: msg="password - {{ lookup('password', '/tmp/passfile3 chars
=ascii_letters,digits,hexdigits,punctuation,,') }}"   #逗号本身用",,"表示
```

测试：ansible-playbook test.yml

```
PLAY [127.0.0.1] ****************************************************

TASK: [debug msg="password - funEtMBYbqWTUdPlfIGC"] *******************
ok: [127.0.0.1] => {
    "msg": "password - funEtMBYbqWTUdPlfIGC"}

TASK: [debug msg="password - 79223199493177921267"] *******************
ok: [127.0.0.1] => {
    "msg": "password - 79223199493177921267"}
```

```
TASK: [debug msg="password - 0,92YO4R0m6iqg2=4RA8"] ********************
ok: [127.0.0.1] => {
    "msg": "password - 0,92YO4R0m6iqg2=4RA8"}

PLAY RECAP ********************************************************************
127.0.0.1: ok=3    changed=0    unreachable=0    failed=0
```

3、其他类型

```
---
 hosts: all
 tasks:
     - debug: msg="{{ lookup('env','HOME') }} is an environment variable"

     - debug: msg="{{ lookup('pipe','date') }} is the raw result of runni
ng this command"

     - debug: msg="{{ lookup('redis_kv', 'redis://localhost:6379,somekey
') }} is value in Redis for somekey"

     - debug: msg="{{ lookup('dnstxt', 'example.com') }} is a DNS TXT rec
ord for example.com"

    - debug: msg="{{ lookup('template', './some_template.j2') }} is a val
ue from evaluation of this template"
```

十四、Ansible 自动化运维：jinjia2 模板

1、 template 应用示例

jinjia 模板官方文档：http://jinja.pocoo.org/docs/dev/templates

```
users:
   - { username: lipeibin,sex: man }
```

```
  - { username: qinlinkai,sex: wowan }
```

1）for 迭代元素

迭代一个列表或者字典中各个元素，每一次取一个

```
<pre>
{% for user in users %}
    <li>Hello {{ user.username }},Welcome your coming.</li>
{% endif %}
</pre>
```

jinja 的循环不支持break 和 continue 标记,可以对需要迭代的列表使用过滤器来达到与 break 和 continue 相同的目的

```
{% for user in users if not user.sex %}
    Hello {{ user.username }},Welcome your coming.
{% endfor %}
```

2）if 比较判断

比较常见的用法是判断一个变量是否已定义，是否非空，是否为 true

```
<pre>
{% if users %}
<ul>
{% for user in users %}
    <li>{{ user.username }}</li>
{% endfor %}
</ul>
{% endif %}
```

```
</pre>
```

和 python 一样，也可以使用 elif 和 else

```
 <pre>
{% if kenny.sick %}
    Kenny is sick.
{% elif kenny.dead %}
    You killed Kenny! You bastard!
{% else %}
    Kenny looks okay --- so far
{% endif %}
</pre>
```

3）if 做表达式过滤器

if 语句也可以被用来做内联表达式或者 for 语句过滤器

```
{% for i in range((play_hosts|count)+2) if i > 1 %}
server.{{ i-1 }}=s{{ i-1 }}.af88.com.cn:2888:3888
{% endfor %}
```

打印结果

```
server.1=s1.af88.com.cn:2888:3888
server.2=s2.af88.com.cn:2888:3888
server.3=s3.af88.com.cn:2888:3888
```

2、jinjia2 模板过滤器（filter）

1）强制变量定义：mandatory

默认情况下，如果一个变量未被定义， ansible 会报错，但是使用下边的方法可以关闭这个特性

```
{{ variable | mandatory }}
```

2）未定义变量设置默认值：default

jinja2 提供了一个 default 的过滤器，当某个变量未定义的时候会设置一个默认值

```
{{ some_variable | default(5) }}
```

3）忽略未定义变量和参数：default(omit)

```
---
- hosts: webservers
  remote_user: root
  tasks:
  - name: touch files with an optional mode
    file: dest={{item.path}} state=touch mode={{item.mode|default(omi
t)}}
    with_items:
        - { path: /tmp/foo }
        - { path: /tmp/bar }
        - { path: /tmp/baz mode: "0444" }
```

执行输出结果：

```
PLAY [webservers] *************************************************
GATHERING FACTS ***************************************************
ok: [192.168.1.65]
TASK: [touch files with an optional mode] *****************************
changed: [192.168.1.65] => (item={'path': '/tmp/foo'})
changed: [192.168.1.65] => (item={'path': '/tmp/bar'})
```

```
changed: [192.168.1.65] => (item={'path': '/tmp/baz', 'mode': '0444'})
PLAY RECAP *********************************************************
192.168.1.65 : ok=2 changed=1 unreachable=0 failed=0
```

创建前两个文件的时候并没有传递 mode 参数，最后一个文件才会传递 mode=0444 参数

4）列表过滤器

```
{{ list1 | min }}          ##从列表中获取最小值
{{ [3, 4, 2] | max }}      ##从列表中获取最大值
{{ list | join(" ") }}     ##将列表转换成字符串
```

5）集合过滤器

从一个列表中获取一个唯一集合

```
{{ list1 | unique }}
```

两个列表的并集

```
{{ list1 | union(list2) }}
```

两个列表的交集

```
{{ list1 | intersect(list2) }}
```

两个列表的差集，在列表 1 中存在，但不在列表 2 中存在

```
{{ list1 | difference(list2) }}
```

两个列表的对称差（只存在于某一个列表中的元素）

```
{{ list1 | symmetric_difference(list2) }}
```

6）随机数

从一个列表中获取一个随机元素

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

获取一个 0 到 59 之间的数

```
{{ 59 |random}} * * * * root /script/from/cron
```

获取 0 到 100 之间的数，但步长为 10

```
{{ 100 |random(step=10) }} => 70
```

获取从 1 到 100 之间的一个随机数，但步长为 10

```
{{ 100 |random(1, 10) }} => 31
{{ 100 |random(start=1, step=10) }} => 51
```

获取一个列表的随机排序结果

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

7）数字匹配

判断一个变量是否数字

```
{{ myvar | isnan }}
```

8）ip 地址过滤器

判断一个字符串是否是有效 IP

```
{{ myvar | ipaddr }}
```

指定一个特定的 IP 版本

```
{{ myvar | ipv4 }}
{{ myvar | ipv6 }}
```

从 CIDR 中获取 IP 地址

```
{{ '192.0.2.1/24' | ipaddr('address') }}
```

9）hashing 过滤器

获取一个字符串的 sha1 hash

```
{{ 'test1'|hash('sha1') }}
```

获取 MD5 值

```
{{ 'test1'|hash('md5') }}
```

获取 checksum

```
{{ 'test2'|checksum }}
```

10）文件路径扩展

获取文件名

```
{{ path | basename }}
```

获取路径的目录

```
{{ path | dirname }}
```

获取链接 (link) 的真实路径

```
{{ path | realpath }}
```

从指定的起点，获取一个链接的相对路径

```
{{ path | relpath('/etc') }}
```

从一个路径或者文件名中获取名称和扩展名

```
# path == 'nginx.conf'
{{ path | splitext }} -> ('nginx', '.conf')
```

11）正则表达式匹配：match 或 search

```
vars:
   url: "http://example.com/users/foo/resources/bar"
tasks:
- shell: "msg='matched pattern 1'"
  when: url | match("http://example.com/users/.*/resources/.*")
- debug: "msg='matched pattern 2'"
  when: url | search("/users/.*/resources/.*")
```

注：match 是必须匹配整个字符串，search 是搜到指定的子串即可

12）正则表达式替换：regex_replace

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

注：\\1 为后向引用

十五、Ansible 自动化运维：Python API 接口

当业务比较大比较复杂的时候，单纯的使用 Ansible 有时候不会很好的完成相关的运维工作，这个时候就需要开发针对自己业务的一些模块或者 ansible 插件来完成这些工作，ansible 提供了一个 python api 接口用于开发编程，使用 python 开发更为自动化的运维管理系统，这个 python api 接口是 python 模块 ansible 下的一个类，即 runner

1、runner 安装部署

```
wget "https://pypi.python.org/packages/source/p/pip/pip-1.5.4.tar.gz#md5=834b2904f92d46aaa333267fb1c922bb" --no-check-certificate

tar -xzvf pip-1.5.4.tar.gz

cd pip-1.5.4

python setup.py install

pip install runner
```

runner 安装在/usr/lib/python2.6/site-packages/ansible/runner

注：runner 目录下面有个__init__.py 文件，__init__.py 的作用， 相当于 class 中的 def __init__(self):函数，用来初始化模块,把所在目录当作一个 package 处理

2、runner 使用说明

```
#!/usr/bin/python

#coding=utf-8

import ansible.runner

import json

Get_Host_Name = ansible.runner.Runner(

   # host_list="/etc/ansible/xiaorui.py",  ## 可以使用动态 inventory 脚本

   host_list="/etc/ansible/hosts",

   module_name='shell',
```

```
    module_args='hostname',
    pattern='web',
    forks=10 ).run()
print json.dumps(Get_Host_Name,indent=4)
```

上面是一个简单的 ansible python api 的执行的例子，我们可以看到他调用的 runner 模块

run()方法是执行定义好的 runner 对象，并返回类型是字典的执行结果，可以封装解析成 json，如下：

```
{
    "dark": {          # 连接不上的主机的返回结果集合
        "172.20.99.6": {
            "msg": "SSH Error: Permission denied (publickey,password).\n
    while connecting to 172.20.99.6:22\nIt is sometimes useful to re-run t
he command using -vvvv, which prints SSH debug output to help diagnose th
e issue.",
            "failed": true
        }
    },
    "contacted": {     # 可以正常连接的主机的返回结果集合
        "172.20.99.5": {
            "changed": true,
            "end": "2016-04-28 14:19:09.407521",
            ## 标准输出返回的内容
            "stdout": "TMP-OPS-99-5",
            "cmd": [
                "hostname"
            ],
            "start": "2016-04-28 14:19:09.390006",
            "delta": "0:00:00.017515",
            ## 执行成功，错误输出为空
```

```
        "stderr": "",
        ## 返回状态码：0
        "rc": 0,
        "invocation": {
            "module_name": "shell",
            "module_complex_args": {},
            "module_args": "hostname"
        },
        "warnings": []
    },
    "172.20.99.4": {
        "changed": true,
        "end": "2016-04-28 14:19:08.369520",
        ## 执行失败，标准输出为空
        "stdout": "",
        "cmd": [
            "hostname"
        ],
        "start": "2016-04-28 14:19:08.352359",
        "delta": "0:00:00.017161",
        ## 错误输出返回的内容
        "stderr": "/bin/sh: hostname: command not found",
        "rc": 127,
        "invocation": {
            "module_name": "shell",
            "module_complex_args": {},
            "module_args": "hostname"
        },
        "warnings": []
    }
```

```
    }
}
```

这些返回的结果对后面python开发很有用，可以通过数据的字段值来判断执行有没有成功，或者可以获取一些关键数据等等

3、runner 参数讲解

参考：http://xiaorui.cc/2014/05/30/ansible-api 的runner 的用法及源码分析/

```
import absible.runner
class Runner(object):
    def __init__(self,
        ## 这里可以放静态 hosts 文件，也可以放 inventory 的脚本，脚本要 777 权限
        host_list=C.DEFAULT_HOST_LIST,
        ## 这个是 ansible 的路径，一般不用写
        module_path=None,
        ## 模块的名字，模块的位置要选定在/usr/share/absible 下，不然会识别不了
        module_name=C.DEFAULT_MODULE_NAME,
        ## 模块的参数，如“src=/tmp/a dest=/tmp/b”
        module_args=C.DEFAULT_MODULE_ARGS,
        ## 并发线程线
        forks=C.DEFAULT_FORKS,
        ## 超时的时间
        timeout=C.DEFAULT_TIMEOUT,
        ## inventory 的匹配
        pattern=C.DEFAULT_PATTERN,
        ## 远程主机用户的选择
        remote_user=C.DEFAULT_REMOTE_USER,
        ## 远程主机密码的选择
        remote_pass=C.DEFAULT_REMOTE_PASS,
```

```
        ## 远程主机端口的选择
        remote_port=None,
        ## 是否是 sudo
        sudo=False,
        ## sudo 用户名
        sudo_user=C.DEFAULT_SUDO_USER,
        ## sudo 密码
        sudo_pass=C.DEFAULT_SUDO_PASS,
        ## 连接方式 默认是用的 paramiko
        transport=C.DEFAULT_TRANSPORT,
        ## 回调的输出
        callbacks=None,
        ## 如果没有使用密码，可以指定 key
        private_key_file=C.DEFAULT_PRIVATE_KEY_FILE
        ):
       ......
```

4、runner 应用示例

获取各个远程主机的实际可用内存（free + cache + buffers）

```
#!/usr/bin/python
# file: get_freeMemory.py
#coding=utf-8

import ansible.runner
import sys
import json

# 获取远程主机的实际可用内存大小:MB
Get_Hosts_FreeMem = ansible.runner.Runner(
    host_list="/etc/ansible/hosts"
```

```
    pattern="*",
    module_name="shell",
    module_args="/usr/bin/free -m |grep 'buffers/cache' |/bin/awk '{prin
t $3}'",
    forks=10,).run()


# 用json封装一下数据，为了后面print显示结果更直观一些 -- 调试使用
#Get_Hosts_FreeMem = json.dumps(Get_Hosts_FreeMem,indent=4)
#print Get_Hosts_FreeMem+"\n\n"
#Get_Hosts_FreeMem = json.loads(Get_Hosts_FreeMem)
# 能连通的主机
if Get_Hosts_FreeMem["contacted"]:
    for hostname,result in Get_Hosts_FreeMem["contacted"].items():
        # 可以正常获取到内存数据的主机
        if Get_Hosts_FreeMem["contacted"][hostname]["stdout"]:
            print "主机:%s, 实际可用内存:%s" %(hostname,Get_Hosts_FreeMem
["contacted"][hostname]["stdout"])
        # 不能正常获取到内存数据的主机
        elif Get_Hosts_FreeMem["contacted"][hostname]["stderr"]:
            print "主机:%s, 获取内存失败:%s" %(hostname,Get_Hosts_FreeMem
["contacted"][hostname]["stderr"])


# 连接不上主机
if Get_Hosts_FreeMem["dark"]:
    for hostname,result in Get_Hosts_FreeMem["dark"].items():
        print "host:%s, 异常信息显示:%s" %(hostname,Get_Hosts_FreeMem["dar
k"][hostname]["msg"].replace("\n"," "))


# pattern 没有可以匹配的主机
if not Get_Hosts_FreeMem["contacted"] and not Get_Hosts_FreeMem["dark"]:
    print "No hosts found"
```

```
    sys.exit(1)
```

脚本附件：

ansible python api示例.py

执行 python 脚本，结果如下：

```
主机:172.20.99.6, 实际可用内存:171
主机:172.20.99.5, 实际可用内存:171
主机:172.20.99.7, 异常信息显示:SSH Error: Permission denied (publickey,password).while connecting to 172.20.99.7:22 It is sometimes useful to re-run the command using -vvvv, which prints SSH debug output to help diagnose the issue.
```