ansible

1.1 ansible介绍

1.1.1 ansible的作用

- ansible是新出现的自动化运维工具，基于Python开发，集合了众多运维工具（puppet、cfengine、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。
- ansible是基于模块工作的，本身没有批量部署的能力。真正具有批量部署的是ansible所运行的模块，ansible只是提供一种框架。
- ansible架构中，集群只需要2种节点：控制节点和受控节点。利用一台控制主机可以对一群受控主机进行命令的下发，运维管控。只需要控制主机（必须是linux系统）安装ansible即可，受控主机需要配有python的环境（版本至少2.7）。

1.2 环境准备

控制节点 上配置好yum仓库，并在每一台机子上创建用户 student，并赋予student用户的管理员免密权限

主机名	ip地址	安装环境	作用
master	192.168.X.100	图形化	控制节点
slave1	192.168.X.101	文本	受控节点
slave2	192.168.X.102	文本	受控节点
slave3	192.168.X.103	文本	受控节点

注意： 所有操作所需参数参考上方表格

- 修改主机名（防止集群操作时，误操作主机）

  ```
  hostnamectl set-hostname 主机名
  ```

  （因为在集群里，是通过ip地址来确定 主机的位置，通过设置静态ip，保证网络设备的地址的 稳定。）
- 按照上方表格设置静态ip地址
- 主机名和ip地址映射（将主机名和ip地址绑定,从而可以 用主机名来代替ip地址）

```
# vim /etc/hosts
127.0.0.1 localhost localhost.localdomain
localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain
localhost6 localhost6.localdomain6
192.168.182.100 master
192.168.182.101 slave1
192.168.182.102 slave2
192.168.182.103 slave3
```

- 创建普通用户student（所有主机创建）

```
useradd student
passwd student 设置密码
```

- 设置普通用户的sudo权限（）

```
root用户下执行命令visudo(sudo设置免密)

student ALL=(ALL) NOPASSWD: ALL
```

补充：由于ansible是通过ssh来进行脚本的下发，所以需要配置 ssh的免密登录，需要远程登录到哪台机子上，就需要将自己的公钥发送到哪台（ansible也支持密码登录，甚至在主机清单文件中写入ssh密码。）

- ssh免密登录 在普通用户student家目录下执行
  - ssh-keygen(一直回车，生成密钥)
  - ssh-copy-id -i .ssh/id_rsa.pub student@slave1

    (注意：其他受控主机也需要发送密钥，格式：**ssh-copy-id -i 公钥路径 用户名@主机名或ip地址**)

1点30开始上课，声音画面无问题的同学群里发1

```
#scp命令的使用方式
#将windows上的软件发送到linux(利用scp将软件发送到
#红帽系统上，注意发送路径和接收路径)
scp .\sshpass-1.06-3.el8ae.x86_64.rpm root@
#控制节点的ip地址:/root
scp .\ansible-2.9.19-
1.el8ae.noarch.rpm root@控制节点的ip地址:/root
```

- 搭建yum仓库
- 安装ansible（前提控制节点yum仓库搭建完毕）

```
yum install -y ./sshpass-1.06-3.el8ae.x86_64.rpm
yum install -y ./ansible-2.9.19-1.el8ae.noarch.rpm
```

- 安装完毕后切换到普通用户，将/etc/下的ansible目录 复制到当前用户的家目录下

```
su - student
cp -r /etc/ansible ~/
```

  - ansible.cfg：ansible的运行配置文件，默认优先使用当 前工作目录（pwd）下的ansible.cfg，其次使用用户家目 录中的.ansible/ansible.cfg，最后使用/etc/ansible/ansible.cfg
  - hosts：清单文件，配置ansible的控制主机的信息，用来 规划集群
  - roles：ansible的角色文件
- 设置清单文件（作用就是为了更好的规划我们的集群）

```
vim hosts
#编辑清单文件
#第一种方式：直接写主机名或者ip地址
主机名1
主机名2
#第二种方式：通过对主机进行分组，然后对组进行管控
#文件格式如下
[组名1]
主机名1
主机名2
[组名2]
主机名1
主机名3
#针对组来进行分类，然后分好类的组来进行管控
[组名3:children]
组名1
组名2
# 补充清单主机变量 ansible_ssh_pass='密码'
```

- 设置ansible的配置文件（**注意：**要执行ansible命令， 当前目录下一定要有相应的ansible.cfg文件，否则它 会读取/etc/ansible下的配置文件（如果未配置），从 而导致报错）

```
vim ansible.cfg
#文件内容如下（使用普通用户进行远程登录）
[defaults]
inventory=./hosts #指明清单文件的路径
remote_user=student #指明远程登录过去的用户是
谁
# ask_pass=Flase #是否提示输入ssh密钥，如
果配置了公钥则可以配置Flase，如果没有配免密，开启
的话，会询问输入密码
roles_path=./roles #配置角色的指定目录
[privilege_escalation]
become=true #是否提权
become_method=sudo #提权的方式是sudo
become_user=root #提权成root用户
become_ask_pass=false #关闭提权的密码询问提示
信息
如果使用root用户(那么同样要设置root用户的ssh远程
连接的免密登录)
```

```
[defaults]
inventory=./hosts #指明清单文件的路径
remote_user=root #指明远程登录过去的用户是谁
roles_path=./roles #配置角色的指定目录
```

- 查看ansible的版本信息

  ansible --version

- ansible运行命令的方式 以命令行的方式来执行ansible命令,叫做ad-hoc，一般 用作临时的测试使用。
  - 命令格式（默认all代表所有主机）

    ansible 受控主机 -m 模块名 -a 参数

  示例:

  ansible web -m shell -a "ls /root”

  利用shell模块在所有受控节点上创建文件/opt/1.txt ， 5分钟

  完成后提供**ansible web -m shell -a "ls /opt”**的截图

- 列出所有模块：

  ansible-doc -l

- 查看指定模块的信息
  ansible-doc 模块名(主要查看ansible的例子)

题目：

1. 查找file模块中的例子，写一个ad-hoc的命令，删除刚刚创建的 2.txt，完成后截图,25继续

```
[student@master ansible]$ ansible all -m file -a "path=/opt/2.txt state=absent"
192.168.182.102 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "path": "/opt/2.txt",
    "state": "absent"
}
192.168.182.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "path": "/opt/2.txt",
    "state": "absent"
}
192.168.182.103 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "path": "/opt/2.txt",
```

```
    "state": "absent"
}
[student@master ansible]$
[student@master ansible]$ ansible all -m shell -a "ls /opt"
192.168.182.102 | CHANGED | rc=0 >>
1.txt
192.168.182.103 | CHANGED | rc=0 >>
1.txt
192.168.182.101 | CHANGED | rc=0 >>
1.txt
```

1. 在slave1节点上创建目录/opt/dirA，要求文件所属组为 student，完成后截图。flie模块

```
[student@master ansible]$ ansible slave1 -m file -a "path=/opt/dirA
state=directory group=student"
192.168.182.101 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "gid": 1000,
    "group": "student",
    "mode": "0755",
    "owner": "root",
    "path": "/opt/dirA",
    "secontext": "unconfined_u:object_r:usr_t:s0",
    "size": 6,
    "state": "directory",
    "uid": 0
}
[student@master ansible]$ ansible slave1 -m shell -a "ls /opt"
192.168.182.101 | CHANGED | rc=0 >>
1.txt
dirA
[student@master ansible]$ ansible slave1 -m shell -a "ls -l /opt"
192.168.182.101 | CHANGED | rc=0 >>
total 0
-rw-r--r--. 1 root root    0 Aug  2 15:06 1.txt
drwxr-xr-x. 2 root student 6 Aug  2 15:43 dirA
```

- 题目：使用anisble命令的方式，将所有主机的yum仓库搭 建完毕 查看模块帮助 ansible-doc yum_repository

1. 创建挂载点/mnt/packages
2. 然后将/dev/sr0永久挂载到/mnt/packages

同样操作在slave2,slave3都完成一遍，提供lsblk的截图

受控节点改为使用本地源

在受控节点创建目录/etc/yum.repos.d/RockyBak file

然后将所有.repo文件移动到RockyBak目录中 shell

完成后提供ls /etc/yum.repos.d/的截图

yum文件的编写格式	模块的编写格式
[仓库名]	name
name= # 描述信息	description
baseurl= # yum源的路径file://本地路径 http://代表网络协议	baseurl
enabled=1 # 启用仓库	enabled
gpgcheck=1 # 直接关闭使用0	gpgcheck
gpgkey #如果启用gpgcheck还需要配置gpgkey	gpgkey

注意：文件名你可以使用file来指定，也可以不指定文件名，默认已仓库名作为文件名然后加入.repo后缀 25分钟继续

10分钟完成上述题目，使用yum_repository搭建yum仓库（BaseOS和AppStream）

```
[student@master ansible]$ ansible all -m yum_repository -a "name=BaseOS
description=baseos baseurl=file:///mnt/packages/BaseOS enabled=yes gpgcheck=no"

[student@master ansible]$ ansible all -m yum_repository -a "name=AppStream
description=appstream baseurl=file:///mnt/packages/AppStream enabled=yes
gpgcheck=no"
```

完成后请使用yum模块安装一下httpd软件，并截图

```
[student@master ansible]$ ansible all -m yum -a "name=httpd state=present"
```

课程目标：

- 使用shell运行ansible命令
- playbook的三种执行方式
- 常见变量
- playbook常见语法：循环遍历（loop）,判断（when），块（block）
- 使用模块：debug,user,yum

一、shell脚本执行ad-hoc

目的：单纯的的使用ad-hoc的方式来执行命令，就和之前在命令行里执行命令一样，不具有永久性，重用性差

（1）简单的shell脚本的编写

#!/bin/bash

执行的命令（一行代表一条执行的语句）

题目：使用shell脚本写一个查看集群所有受控节点的磁盘状态信息。脚本名lsblk.sh

```
[student@master ansible]$ vim lsblk.sh
[student@master ansible]$ chmod u+x lsblk.sh
[student@master ansible]$ cat lsblk.sh
#!/bin/bash

ansible all -m shell -a "lsblk"
[student@master ansible]$ ./lsblk.sh
192.168.182.103 | CHANGED | rc=0 >>
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0          11:0    1  9.2G  0 rom  /mnt/packages
nvme0n1     259:0    0   20G  0 disk
├─nvme0n1p1 259:1    0    1G  0 part /boot
└─nvme0n1p2 259:2    0   19G  0 part
  ├─rl-root 253:0    0   17G  0 lvm  /
  └─rl-swap 253:1    0    2G  0 lvm  [SWAP]
192.168.182.102 | CHANGED | rc=0 >>
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0          11:0    1  9.2G  0 rom  /mnt/packages
nvme0n1     259:0    0   20G  0 disk
├─nvme0n1p1 259:1    0    1G  0 part /boot
└─nvme0n1p2 259:2    0   19G  0 part
  ├─rl-root 253:0    0   17G  0 lvm  /
  └─rl-swap 253:1    0    2G  0 lvm  [SWAP]
192.168.182.101 | CHANGED | rc=0 >>
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0          11:0    1  9.2G  0 rom  /mnt/packages
nvme0n1     259:0    0   20G  0 disk
├─nvme0n1p1 259:1    0    1G  0 part /boot
└─nvme0n1p2 259:2    0   19G  0 part
  ├─rl-root 253:0    0   17G  0 lvm  /
  └─rl-swap 253:1    0    2G  0 lvm  [SWAP]
```

（2）ansible和shell的区别

ansible有很多现成的模块可以供使用， 降低了使用者的学习成本。

最主要的一个原因：ansible可以根据受控主机的状态来对操作是否执行进行取舍。而shell不管主机的状态，都会执行命令，需要人为进行编辑脚本进行判断，增加复杂度。

二、ansible的playbook

playbook是利用yaml语法来编写的脚本，是一种描述型语言，以键值的方式来描述要执行的操作。

把他当做类似问卷调查的填空就行。把一个个模块当做不同问卷调查的模板，你只需要按照其中的模板进行填空就行。

家庭情况调查	**user**
姓名：	name:
身份证：	uid:

注意：所有执行命令的操作都必须在当前anisble的配置文件目录进行操作

三、编写playbook

vim 文件名.yml（文件的后缀为.yml）

```
---                          # 用三个"-"代表playbook的起始符
- hosts：受控主机名          # 指明哪些主机来执行任务
  tasks:                      # 准备编写任务
      - name: 任务名          # 任务名，描述任务的作用
        模块名：              # 所要执行的模块
        参数名：参数值        # 根据ansible-doc 模块名 中的例子来完成要执行的模块
```

注意：在playbook中，同一项目用减号“-”来表示，不同级别的键值是通过缩进来区分，缩进必须统一（不能tab键和空格键混用），说白了，同一类是用“-”标识，同一类下，不同级别用缩进标识

明白的发1

不明白发2

题目： 创建用户student01,uid为1666

```
[student@master ansible]$ cat student01.yml
---
- hosts: web
  tasks:
    - name: "useradd student01"
      user:
        name: student01
        uid: 1666
```

题目：将studet01用户从所有受控节点删除。（使用playbook的方式，模块名为user）

```
[student@master ansible]$ cat delstudent01.yml
---
- hosts: all
  tasks:
     - name: "del student01"
       user:
         name: student01
         state: absent
         remove: yes
```

state：后面一般会跟三种值	
absent	如果有就删除
present	如果没有，那就创建
latest	一般在yum软件安装出现（更新到最新版本）

四、ansible执行三种方式

注意: 看起来好像多此一举，但是其实它变相的缩小了排错的范围。

（1）检查语法

ansible-playbook --syntax-check 文件名.yml

（2）模拟运行

ansible-playbook -C 文件名.yml

（3）真实运行

ansible-playbook 文件名.yml

四、playbook的排错思路:

（1）在语法检查时报错:

- 查看缩进有没有问题
- playbook中的语法和关键词有没有写错

（2）真实运行时报错

- 模块中的关键词中所需的参数有没有写错。如state：present 写成state: persent("**Unsupported parameters**)
- 判断语句或者循环语句中的逻辑关系有没有理清.
- 事实变量和魔法变量在取用时，格式有没有问题。

注意：ansible的本质就是将控制主机上编写的任务交给受控节点去执行，所以所有编写的ansible脚本都可以转换成我们学过的csa中的命令直接在受控主机上跑一遍，如果觉得自己的playbook没有问题，但是还是报错，那么可以使用该方法去受控主机上排错。查看是不是受控主机本身出了问题。

题目： 使用yum模块，在web组上安装httpd(使用playbook的方式)

五、变量

（1）变量的定义

用一个变量名来存储变量的值，从而通过变量名即可获取到对应的值，

（2）变量命名

以字母开头，并且只能是字母，数字和下划线组成。

在ansible中，变量可以分为2类：

（a）用户自定义变量（变量的定义，需要使用关键词vars）

（b）内置变量（ansible自带，用户可以直接调用）

（3）变量的取值：

```
"{{ 变量名 }}"
```

（4）用户自定义变量

（a）普通变量

```
  vars:
     变量名: 值
```

仿照右边的代码，然后创建用户student02使用变量的方式。

```
---
- hosts: slave1
  vars:
    username: student02
  tasks:
    - name: "msg {{username}}"
      debug:
        msg: "{{username}}"

    - name: "useradd {{username}}"
      user:
        name: "{{username}}"
```

示例：创建playbook，tom.yml,使用变量的方式创建用户tom,密码为1

其中用户名username: tom

```
        password: '1'
```

在所有节点上创建用户tom,并设置密码完成后截图

```
---
- hosts: all
  vars:
    username: tom
    password: '1'
  tasks:
    - name: "useradd {{username}}"
      user:
        name: "{{username}}"
        password: "{{ password | password_hash('sha512')}}"
        # 注意： password中双引号和花括号之间不能出现空格，否则会作为密码一部分导致密码hash
失败。
```

（b）数组（列表）变量

```
 vars:
   变量名:
      - 值1
      - 值2
#数组变量的取值
#(1) 取出单个值
#    数组变量名[值所在的位数]，可以取出列表中某一个位置上的值。
#    例如:取值1，可以写成{{变量名[1]}}
#(2)取出所有值{{数组变量名}}
#(3)搭配循环遍历所有值   在循环中介绍
```

创建一个列表变量，users

```
vars:
  users:
    - user01
    - user02
    - qwe
```

然后该列表变量创建用户user01，user02，密码为qwe（所有值通过变量来取），slave1创建，

3点07继续，完成后截图

```
[student@master ansible]$ cat users.yml
---
- hosts: slave1
  vars:
    users:
      - user01
      - user02
      - qwe
  tasks:
```

```
    - name: "msg users"
      debug:
        msg: "{{users[0]}}---{{users[1]}}---{{users[2]}}"

    - name: "useradd {{users[0]}}"
      user:
        name: "{{users[0]}}"
        password: "{{users[2] | password_hash('sha512')}}"


    - name: "useradd {{users[1]}}"
      user:
        name: "{{users[1]}}"
        password: "{{users[2] | password_hash('sha512')}}"
```

题目：将之前创建的用户删除，然后重新创建用户user01,user02,密码都是2。用户名用列表变量usernames存储，密码单独用普通变量passwd存储（10分钟）受控节点为slave1

（c）字典变量

```
vars:
    变量名A:
       - 变量名1: 值1
         变量名2: 值2
       - 变量名3: 值3
         变量名4: 值4
```

```
---
- hosts: slave1
  vars:
    userlist:
      - name: user01
        password: "1"
      - name: user02
        password: "2"
  tasks:
    - name: "msg userlist"
      debug:
        msg: "{{userlist[1]['password']}}"
```

题目:将上列输出，改成用user模块创建用户，完成后截图

```
[student@master ansible]$
[student@master ansible]$ cat dictvars.yml
---
- hosts: slave1
  vars:
    userlist:
      - name: user01
```

```
        password: "1"
      - name: user02
        password: "2"
  tasks:
    - name: "useradd userlist"
      user:
        name: "{{userlist[0]['name']}}"
        password: "{{userlist[0]['password'] | password_hash('sha512')}}"

    - name: "useradd userlist"
      user:
        name: "{{userlist[1]['name']}}"
        password: "{{userlist[1]['password'] | password_hash('sha512')}}"
```

（5）ansible内置变量

内置变量可以在playbook中直接调用

（a）事实变量

playbook中gather_facts可以设置执行playbook的过程中开启或者关闭事实变量的采集。

gather_facts: no # 默认是开启状态，如果关闭可将值设置成no

- 事实变量主要针对受控主机自身的一些属性值，比如它自己的ip地址，它自己的主机名，它自己的磁盘信息。
- 在命令行中可以使用（**ansible 主机名 -m setup来输出查看有哪些事实变量**）
- 可以使用**ansible 主机名 -m setup > 主机名fact.txt** 将输出保存到本地。
- 然后用vim打开文档，使用查找命令**:\事实变量值**，来反推出**事实变量名。**

例如：我知道slave1的ip地址为192.168.182.101，但是我不知道事实变量对于ip地址的表述方式，那么可以通过ip地址的值来反推查找slave1的ip地址的事实变量名

address　ansible_default_ipv4（一直网上找，找到ansible开头的变量。ansible_facts['default_ipv4'])

```
# 由ip地址192.168.182.101反推出事实变量名是由address---》ansible_default_ipv4（由小范围到大范围）
# 如果要从事实变量表达出事实变量的值，应该是由大范围定义到小范围
# ansible_default_ipv4----->address
# 然后官方推荐将ansible_变成ansible_facts
# 最终结果就是ansible_facts['default_ipv4']['address']
```

打印所有节点的ip地址，完成后截图。

写出根据上述内容获取到size大小的事实变量写法

```
[student@master ansible]$ cat sizeFacts.yml
---
- hosts: all
  tasks:
    - name: "msg size"
      debug:
        msg: " the size of nvme0n1p1 :{{ansible_facts['devices']['nvme0n1']
['partitions']['nvme0n1p1']['size']}}"
```

请输出slave1上可以输出rl-root值的事实变量写法，并编写playbook输出所有节点上的值。

10分钟，10点55继续

1 holders nvme0n1p2 partitions nvme0n1 ansible_devices

```
[student@master ansible]$ cat holdersFacts.yml
---
- hosts: all
  tasks:
    - name: "msg holders['1']"
      debug:
        msg: "{{ansible_facts['hostname']}} : {{ansible_facts['devices']
['nvme0n1']['partitions']['nvme0n1p2']['holders'][1]}}"
```

题目：使用事实变量，输出所有节点的网关地址，使用事实变量，输出sr0的uuid，

- gateway ansible_default_ipv4
- sr0 uuids ansible_device_links

注意： 在变量获取中，如果存在一个值多种获取方式，尽量避免使用列表的形式。

```
---
- hosts: all
  tasks:
    - name: "gateway uuid"
      debug:
        msg: "{{ansible_facts['default_ipv4']['gateway']}} :
{{ansible_facts['device_links']['uuids']['sr0'][0]}}"
```

i）常见事实变量

短主机名	ansible_facts["hostname"]
完全限定的域名	ansible_facts['fqdn']
ipv4地址	ansible_facts["default_ipv4"]["address"]
所有网络接口的名称列表	ansible_facts["interfaces"]

短主机名	ansible_facts["hostname"]
/dev/vda1的磁盘分区大小	ansible_facts["devices"]["vda(磁盘名)"]["partitions"]["vda1（分区名）"]["size"]
DNS服务器列表	ansible_facts["dns"]["namesreves"]
当前运行的内核版本	ansible_facts["kernel"]
当前系统的bios版本	ansible_facts["bios_version"]

```
  ---
  -  hosts: slave1,slave2
     tasks:
        - name: test ip
          debug:
             msg: "{{  ansible_facts["default_ipv4"]["address"] }}"  #使用事实变量查看受控主机本身的ip地址
```

（c）魔法变量

魔法变量记录了整个ansible集群里的一些属性值，比如有多少主机组，组里面一共有哪些主机，整个集群里的配置文件。

所以我们可以通过魔法变量获取到任何一台主机的事实变量值，但事实变量本身只能获取主机自身的值。

在命令行中使用ansible 清单文件中主机名 -m debug -a "var=hostvars",查看所有主机的魔法变量

在命令行中使用ansible slave1 -m debug -a "var=hostvars['slave1']" ， 查看slave1上的魔法变量

```
---
- hosts: all
  tasks:
    - name: "msg slave1 ip hostname"
      debug:
        msg: "{{hostvars['192.168.182.101']['ansible_facts']['default_ipv4']['address']}}  {{hostvars['192.168.182.101']['ansible_facts']['hostname']}}"
```

题目：所有节点上输出 slave2上ip地址和主机名的映射关系,完成后截图。

```
---
- hosts: all
  tasks:
    - name: "msg slave3"
      debug:
        msg: "{{hostvars['192.168.182.103']['ansible_facts']['default_ipv4']['address']}}  {{hostvars['192.168.182.103']['ansible_facts']['hostname']}}"
```

注意： 魔法变量中，想要获取变量值的主机必须在受控节点内。否则会报变量为定义的错误

题目： 请获取web组中，有哪些主机ip

一个指定slave1上的魔法变量信息，所有节点上执行

```
            "192.168.182.101",
            "192.168.182.102"
```

web groups hostvars['192.168.182.101']

事实变量只能显示本主机自己的信息，而魔法变量能显示受控主机集群里任一台的信息

```
-  ---
  - hosts: slave1,slave2
    tasks:
       - name: test
         debug:
           msg: '事实变量:{{ ansible_facts["default_ipv4"]["address"]  }} 魔法变量:
{{  hostvars["slave2"]["ansible_facts"]["default_ipv4"]["address"]  }}'
```

（d）注册变量：register

将任务的运行结果存储到变量中，便于后续任务的调用。(如果某些任务，看不到运行的结果，可用于通过注册变量的方式，将结果用debug模块打印)

```
 ---
 - hosts: slave1
   tasks:
      - name: useradd user03
        user:
          name: user03
        register: result

      - name: msg result
        debug:
          msg: "{{result}}"
```

将返回结果中的用户的家目录打印出来，完成后截图

```
[student@master ansible]$ cat register.yml
---
- hosts: slave1
  tasks:
   - name: "useradd user03"
     user:
       name: user03
     register: result
```

```
  - name: "msg result"
    debug:
      msg: "{{result['home']}}"
```

六、基本语法

6.1 循环遍历（loop）

在ansible中，循环的作用就是为遍历，为了取出数组或字典中的值，每取一个值，就存放在内置变量item中。

格式：

循环和需要循环的任务同一级别，先看循环，然后把每次的循环的值再带入任务中去执行。

```
---
- hosts: slave1
  vars:
    userlist:
      - name: user01
        password: "1"
      - name: user02
        password: "1"
  tasks:
      - name: useradd and passwd
        debug:
           msg: "{{item}}"
        loop:
           "{{userlist}}"
```

利用循环删除用户user01,user02,user03,完成后截图

```
[student@master ansible]$ cat loop.yml
---
 - hosts: slave1
   vars:
     usernames:
       - user01
       - user02
       - user03
   tasks:
     - name: "msg usernames"
       debug:
         msg: "{{item}}"
       loop:
         "{{usernames}}"

     - name: "userdel usernames"
       user:
         name: "{{item}}"
```

```
        state: absent
        remove: yes
      loop:
        "{{usernames}}"
```

习题1：在受控主机slave1上，创建用户user01，user02，密码都是“1”，利用字典变量和循环来完成。

```
[student@master ansible]$ cat loop.yml
---
 - hosts: slave1
   vars:
     userinfo:
        - username: user01
          password: "1"
        - username: user02
          password: "2"
   tasks:
     - name: "msg userinfo"
       debug:
         msg: "{{item['username']}} {{item['password']}}"
       loop:
         "{{userinfo}}"
```

题目：利用循环和字典变量创建用户user01和user02,完成后截图。

6.2 判断（when）

通过when后面的判断条件是否为真，从而决定是否执行对应的任务。符合条件则执行，不符合则跳过，可以搭配loop来使用。

示例条件

操作	示例
等于（值为字符串）	ansible_facts['hostname'] == "slave1"
等于（值为数字）	max_memory == 512
小于	min_memory < 128
大于	min_memory > 256
小于等于	min_memory <= 256
大于等于	min_memory >= 512
不等于	min_memory != 512

操作	示例
变量存在	min_memory is defined
变量不存在	min_memory is not defined
变量值存在，且属于后一个变量的列表值中	ansible_facts["hostname"] in groups["web"]
多条件判断	
and	左边的条件和右边的条件同时成立
or	左边的条件或右边的条件有一个成立即可
更复杂的判断可以使用（）进行组合	条件1 and （条件2 or 条件3）

题目：使用循环来安装软件包httpd，mysql-server，要求如果主机名为slave1,并且属于web组，然后可以安装。

6.3 块处理（block）

用于批量的管理任务(注意缩进)，用法一般有2种：when搭配或者使用block……rescue……always结构使用。

当受控主机为slave1时，写2个任务分别输出完整主机名和分区nvme0n1p1的大小

```
---
- hosts: slave1,slave2
  tasks:
    - block:
      - name: msg hostname
        debug:
          msg: "{{ansible_facts['hostname']}}"


      - name: msg ip
        debug:
          msg: "{{ansible_facts['default_ipv4']['address']}}"
      when: ansible_facts["hostname"] == "slave1"
```

效果等同于

```
---
- hosts: slave1,slave2
```

```
  tasks:
    - name: msg hostname
      debug:
        msg: "{{ansible_facts['hostname']}}"
      when: ansible_facts["hostname"] == "slave1"


    - name: msg ip
      debug:
        msg: "{{ansible_facts['default_ipv4']['address']}}"
      when: ansible_facts["hostname"] == "slave1"
```

6.4 异常处理

利用block块语句，可以结合rescue和always语句来处理错误。

block : 定义要运行的主要任务

rescue: 定义要在block子句中定义的任务失败时运行的任务。

always: 定义始终都独立运行的任务，无论block和rescue子句中定义的任务是成功还是失败。

```
---
- hosts: web
  tasks:
    - block:
       - name: install hosfsd
         yum:
           name: hodfsd
           state: present

       - name: install httpd
         yum:
           name: httpd
           state: present

      rescue:
       - name: msg package
         debug:
           msg: "软件安装失败"

       - name: yum httpd
         yum:
           name: httpd
           state: present
      always:
       - name: install mysql
         yum:
           name: mysql-server
           state: present
```

6.5 忽略失败（ignore_errors）

忽略任务失败使用的是ignore_errors语句，可以防止因为任务失败而导致程序运行的终止。

```
---
- hosts: web
  tasks:
    - name: msg hostname
      debug:
        msdfg: "{{ansible_facts['hostname']}}"
      ignore_errors: yes


    - name: msg ip
      debug:
        msg: "{{ansible_facts['default_ipv4']['address']}}"
```

6.6 触发器(notify.....handlers)

触发器其实就是根据任务运行中的change参数是否发生改变来决定是不是运行，一旦发生改变触发后就会运行handlers里的任务。

注意:notify中可以放入多个触发任务。使用方法如下。

安装软件httpd，如果安装成功那么就启动该软件（systemd），

如果前面步骤错误，那么输出信息提示该软件安装并启动失败，不管怎样都会尝试重启该软件（如果无法重启则忽略）。

```
---
- hosts: slave1
  tasks:
    - name: yum install httpd
      yum:
        name: httpd
        state: present


    - name: restart httpd
      systemd:
        name: httpd
        state: restarted
      notify: msg httpd


  handlers:
    - name: msg httpd
      debug:
        msg: "httpd 已经重启"
```

七、剧本拆分

7.1 变量拆分

变量的拆分本质就是将playbook中的变量，放在外部存储变量文件中，然后使用vars_files关键字将外部变量文件导入。

注意：虽然vars_files从外部文件读取变量，但是外部变量文件的编写也要遵守yml文件的格式

cat userlist.yml

```
userlist:
  - username: user01
    password: "2"
  - username: user02
    password: "2"
```

cat user.yml

```
---
- hosts: slave1
  vars_files:
    - ./userlist.yml
  tasks:
    - name: useradd name
      user:
        name: "{{ item.username }}"
        password: "{{ item.password | password_hash('sha512')}}"
      loop:
        "{{userlist}}"
```

7.2 任务拆分

任务拆分和变量的拆分其实一样，将playbook中的任务，存放在外部的存储文件中，然后通过import_tasks(静态导入)或include_tasks（动态导入）来导入外部文件的任务。

cat import_msg.yml

```
---
- name: import debug
  debug:
    msg: "import tasks is successful"

cat import_msg.yml
---
- name: include debug
  debug:
    msg: "include tasks is successful"
```

cat test.yml

```
---
- hosts: slave1
  tasks:
  - import_tasks: ./import_msg.yml
  - include_tasks: ./include_msg.yml
```

动态任务和静态任务的区别?

可以使用ansible-playbook yml文件 --list-tasks列出yml文件中的任务，可以看到区别。

静态任务，在运行前就已经提前加载好，而动态任务，在运行时才会主动加载。

八、角色

其实无论是项目拆分还是角色都是为了更好的管理playbook，由项目的拆分到角色，其实就是由官方指定了一种框架，围绕着这个框架来编写playbook，将功能细化。

- 角色可以分组内容，从而与他人轻松共享代码。
- 可以编写角色来定义系统类型的基本要素：例如web服务器
- 角色使得较大型项目更容易管理
- 角色可以由不同的管理员并行开发。

ansible-galaxy list命令可以列出本地能找到的角色

8.1 系统角色

- 安装系统角色

  ```
  sudo  yum install rhel-system-roles
  ```

注意：

- 如果忘记了包的具体名字。可以使用yum list | grep role 来过滤出完整的系统包名。
- 系统角色的默认安装位置在/usr/share/ansible/roles/
- 在系统角色的目录下，有对应的使用说明书，README.md
- 为了方便统一管理，建议将安装在默认位置下的角色复制到项目目录下的roles目录。

cp -r /usr/share/ansible/roles/* /home/student/ansible/roles

- 在ansible.cfg配置文件的所在目录下，编辑文档vim timentp.yml

```
- hosts: slave1
  vars:
    timesync_ntp_servers:
      - hostname: ntp.aliyun.com
        iburst: yes
  roles:
    - rhel-system-roles.timesync
```

8.2 第三方角色

galaxy是一个公共的ansible角色资源库，类似yum仓库一样。

- ansible-galaxy search "角色名" --platforms EL (适用于EL级企业linux平台的角色)
- ansible-galaxy info 角色名
- 从ansible-galaxy安装角色
- ansible-galaxy install 角色名 -p 指定安装路径 （可以使用-p来指定安装路径）

使用要求文件安装角色

1. 首先编辑一个yml文件用来存放定义的变量（本质上来讲就是通过变量来指定安装的路径和安装的角色名）

cat cat install_role.yml

```
- src: file:///mnt/roles/community-mysql-2.1.0.tar.gz
  name: mysql

- src: file:///mnt/roles/geerlingguy-php_roles-1.0.0.tar.gz
  name: php
```

注意：

src:首先是匹配的是ansible galaxy中的角色，如果没有则将使用指明角色的url。

name用来覆盖角色的本地名称。

2.使用-r指定role文件的安装位置。-p指定角色的安装位置。

```
ansible-galaxy install -r install_role.yml -p ./
```

8.3 自定义角色

（1）创建角色初始目录

使用命令ansible-galaxy init 新角色名 来创建新角色的目录结构。

角色完成后，可以将没有用到的目录删掉。

角色结构

defaults	默认变量值
files	执行任务时，所需的静态文件
handlers	notify触发任务时，所执行的触发任务
meta	编写角色的相关信息，如作者，许可证，平台以及角色的依赖
templates	存放角色所需的jinja2模板
tasks	要执行的主任务
tests	可以包含用来测试用的清单，yml文件等
vars	定义角色的变量值，此处的变量值优先级较高，用于角色内部的定义
README.md	一般用来解释该角色的使用方法和功能

注意：vars和defaults的区别：你可以这么认为，defaults中的变量更像是初始变量，给变量一个默认值，如果vars有值，那么优先使用vars中的变量值，如果没有，那么就使用defaults中的默认值

（2）定义角色内容

本质上来讲就是角色的拆分，按照不同的功能，将复杂的playbook进行拆分，放到不同的目录下，便于管理和重利用。

题目：在受控主机salve1上安装httpd软件，并使用template模块，将文档index.j2发送到指定受控主机slave1目录/var/www/html,重命名为index.html,重启http服务和防火墙，一旦重启成功，提示信息"httpd,firewalld已经重启"，使用防火墙对http服务放行。

```
---
- hosts: slave1
  vars:
     package: httpd              # 这个可以拆分放到vars目录下
  tasks:
    - name: yum install httpd    #这个拆分后可以放到tasks目录下
      yum:
        name: "{{ package }}"
        state: present

    - name: template file to /var/www/html/
      template:
        src: ./index.j2
        dest: /var/www/html/index.html

    - name: restart {{ package}}
      systemd:
        name: "{{ item }}"
        state: restarted
      loop:
         - httpd
         - firewalld
      notify: msg httpd,firewalld
```

```
    - name: open http port
      firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: yes
  handlers:
    - name: msg httpd,firewalld          #这个拆分后可以放到handlers目录下
      debug:
        msg: "httpd,firewalld已经重启
```

cat ./index.j2注意：

```
 <!DOCTYPE html> <html> <head> <meta  charset="utf-8"> <title>主机名</title>
</head> <body> <h1>主机名:{{ansible_facts['hostname']}}</h1>  <p>ip地址为:
{{ansible_facts['default_ipv4']['address']}}</p>  </body> </html>
```

Jinja2是Python下一个被广泛应用的模版引擎，他的设计思想来源于Djanjo的模板引擎，并扩展了其语法和一系列强大的功能。ansible的模板配置文件就是用jinja2这种模板编程语言写。

jinja2的模板的的使用需要用到template模板，它将jinja2文件从控制节点发送到受控主机上，然后并执行，将生成后的结果文件保存在受控主机上。

jinja2语法：

1. 变量

使用 {{ }} 语法取变量的值

2. 循环

在jinja2使用的是for循环

结构如下：

```
{% for host in groups['all'] %}
{{hosts}}
{{ hostvars[host]["ansible_facts"]["default_ipv4"]["address"] }}  {{
hostvars[host]["ansible_facts"]["hostname"]}}
{% endfor %}
```

利用循环打印db（slave2,slave3）组里面IP地址和主机名

注意点：保证groups中的组成员必须都是开启状态且包含在playbook受控节点中。

3. 判断

(I)单分支判断

```
{% if   ansible_facts['devices']['vdb'] is undefined  %}
{{ansible_facts['hostname']}} : The disk does not exist
{% endif %}
```

(II)多支判断

```
{% if   ansible_facts['hostname'] == 'slave1'  %}
hello,slave1
{% elif ansible_facts['hostname'] == 'slave2' %}
hello,slave2
{% else %}
hello,other slaves
{% endif %}
```

4.注释

{# 注释内容 #}

5.过滤器

当想要输出的值不存在的话，可以利用管道符，显示default("")中的值。 例如 "{{ 变量值 | default('默认值')}}"

（3）在playbook中使用角色

注意：每个角色应该有自己特定的用途，而不是把任何任务都堆在同一个角色中。

建立playbook然后使用roles:来使用角色。

```
---
- hosts:  slave1
  roles:
     - 角色名1
     - 角色名2
```

题目： 将刚刚http软件的安装起动，包括拆分成角色完完整整至少走2遍。

九、模块介绍

9.1 用户管理模块

创建组users（group），然后创建用户user01,要求user01的附加组为users(user),要求创建成功后，输出“user01 had created”注意：同样是在slave1上面，且要求在web组中安装软件组

删除slave1上所有以user开头的用户（要求用到循环遍历和列表变量）

9.1.1 user模块

```
- name: Add the user 'johnd' with a specific uid and a primary >
  user:
    name: johnd
    uid: 1040
    group: admin
#group是指主组，groups是指附加组。
参数补充1：(为用户添加附加组)
    groups: 附加组
    append: yes
参数补充2：
    password: "{{ 密码 | password_hash( 'sha512' )}}""
```

9.1.2 group模块

```
- name: Ensure group "somegroup" exists
  group:
    name: somegroup
    state: present
```

然后web组所有主机安装：httpd，在db组安装软件组：Development tools，然后对slave1上的httpd进行更新，更新完毕后，将 index.j2 文件 该文件发送到slave1的 目录/var/www/html,重命名为index.html,，先放行httpd服务，重启http服务和防火墙，一旦重启成功，提示信息"httpd,firewalld已经重启"

index.j2内容

```
<!DOCTYPE html> <html> <head> <meta  charset="utf-8"> <title>主机名</title>
</head> <body> <h1>主机名:{{ansible_facts['hostname']}}</h1>  <p>ip地址为:
{{ansible_facts['default_ipv4']['address']}}</p>  </body> </html>
```

9.2 软件管理模块

9.2.1 yum_repository模块示例

```
- name: Add multiple repositories into the same file
  yum_repository:
    name: epel #仓库名
    description: EPEL YUM repo #源的描述
    file: external_repos #yum配置的文件名，如果不设置的话，默认以name的值为配置文件名。
    baseurl: https://download.fedoraproject.org/pub/epel/$relea> # yum包的制定路径
    enabled: yes #启用仓库
    gpgcheck: no #启用gpg校验
    gpgkey: gpg校验路径 #指明gpg校验的路径

注意：如果使用shell脚本来执行yum_repository模块，要使用到ad-doc命令，但是参数由冒号改用等号=
```

9.2.2 yum模块示例

示例1：（安装软件组，）

```
- name: install the 'Development tools' package group
  yum:
    name: "@Development tools"
    state: present

示例2：(安装软件)
- name: install one specific version of Apache
  yum:
    name: httpd
    state: present

注意：
* state中有3个选择，present（如果没有安装，那么安装）。latest(更新软件，如果没有安装，自动安装最新版本)。absent（如果已安装，则删除该软件）
```

9.2.3 systemd模块示例

```
示例1：
- name: enable httpd
   systemd:
      name: httpd #启动的软件包
      state: started #启动，关闭是stoppd
      enabled: yes #设置开机自启

firewalld模块示例
- firewalld:
  service: http # 需要放过的服务
    permanent: yes # 永久生效
    state: enabled # 是否启用
    immediate: yes # 立即生效
```

题目：在slave1的/opt目录下创建目录dirA,要求文件所有者为user02,所属组为user03，权限为rwxrwxr_x,然后在dirA中继续创建文件file2。

题目：在slave1中，dirA目录下创建软链接lns1,链接的源文件为/opt/file1

9.3 文件管理模块

9.3.1 file模块示例

```
示例1：创建目录，权限为0775，其中第一个0的位置指的是特殊权限
- name: Create a directory if it does not exist
  file:
    path: /etc/some_directory
```

```
    state: directory
    mode: '0755'
其他参数：
* ower:  文件所有者
* group: 文件所属组

示例2：创建链接
- name: Create a symbolic link
  file:
    src: /file/to/link/to
    dest: /path/to/symlink
    owner: foo
    group: foo
    state: link
```

9.3.2 copy模块示例

将控制节点上的index.j2 复制到slave1上的/opt目录下，在受控节点的index.j2文件内加入hello内容。

```
示例1：（将content后面的内容复制到dest后面的地址上）
- name: Copy using inline content
  copy:
    content: '# This file was moved to /etc/other.conf'
    dest: /etc/mine.conf

- name: 从控制节点复制文件到受控节点上
  copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
    owner: foo
    group: foo
    mode: u=rw,g=r,o=r

注意：copy命令是将控制主机上的文件复制到受控主机上
```

9.3.3 template模块示例

创建hosts.yml的剧本，作用与web主机组上的所有主机，并通过jinja2模板文件hosts.j2生成在slave1上生成/opt/myhosts文件 效果如下：

192.168.122.11 slave1

192.168.122.12 slave2

```
template可以用来处理jinjia2文件。通过对jinjia2语法进行解析，在受控主机上生成配置文件。

- name: Template a file to /etc/files.conf
   template:
     src: /mytemplates/foo.j2
     dest: /etc/file.con
```

9.3.4 lineinfile模块示例

首先将控制节点上的/etc/selinux/config文件复制到受控节点slave1的/opt目录下，然后将slave1上的/opt/config文件内的SELINUX=enforcing改完SELINUX=permissive。

```
# NOTE: Before 2.3, option 'dest', 'destfile' or 'name' was used instead of
'path'
- name: Ensure SELinux is set to enforcing mode
  lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: SELINUX=enforcing
```

通过正则表达式来对指定文件/etc/selinux/config的行进行替换。

9.3.5 get_url模块示例

```
- name: Download file from a file path
  get_url:
    url: file:///tmp/afile.txt
    dest: /tmp/afilecopy.txt
# 从受控主机指定路径复制文件到受控主机路径。
```

9.4 磁盘管理模块

9.4.1 parted模块示例

```
- name: Create a new primary partition with a size of 800MiB
  parted:
    device: /dev/sdb
    number: 1
    state: present
    part_start: 1MiB
    part_end: 801MiB
```

创建分区/dev/sdb2,大小为600MiB格式化成ext4类型，并挂载到/mnt/sdb2目录

9.4.2 lvg模块示例

```
- name: Create a volume group on top of /dev/sda1 with physical extent size =
32MB
  lvg:
    vg: vg.services
    pvs: /dev/sda1
    pesize: 32
```

创建卷组，vg后跟卷组名，pvs：将分区转化成物理卷并加入卷组，多个分区用逗号隔开。
pesize:指定pe的大小，默认单位M

9.4.3 lvol模块示例

```
- name: Create a logical volume of 512g.
  lvol:
    vg: firefly
    lv: test
    size: 512g
```

参数说明：

* vg：卷组名
* lv： 逻辑卷名
* size： 逻辑卷大小

9.4.4 filesystem模块示例

```
- name: Create a ext2 filesystem on /dev/sdb1
  filesystem:
     fstype: ext4
     dev: /dev/sdb1
```

参数说明：

* fstype: 文件系统类型
* dev: 要格式化的目标路径

9.4.5 mount模块示例

```
- name: Mount a volume
  mount:
    path: 挂载点
    src: 设备路径
    state: mounted
    fstype: ext4
```

参数说明：

* path: 挂载点
* src: 存储设备路径
* state: mount（临时挂载），（mounted,永久挂载）