day06

playbook

模块

```
# 在test组中的主机上，安装httpd、php、php-mysqlnd
[root@control ansible]# vim pkg.yml
---
- name: install pkgs
  hosts: test
  tasks:
    - name: install web pkgs
      yum:
        name: httpd,php,php-mysqlnd
        state: present
# 安装多个软件包，还可以写为：
---
- name: install pkgs
  hosts: test
  tasks:
    - name: install web pkgs
      yum:
        name: [httpd,php,php-mysqlnd]
        state: present

# 安装多个软件包，还可以写为：
---
- name: install pkgs
  hosts: test
  tasks:
    - name: install web pkgs
      yum:
        name:
          - httpd
```

```
        - php
        - php-mysqlnd
      state: present

# 根据功能等，可以将一系列软件放到一个组中，安装软件包组，将会把很多软件一起安装上。比如gcc、java等都是开发工具，安装开发工具包组，将会把它们一起安装。
[root@node1 ~]# yum grouplist   # 列出所有的软件包组
[root@node1 ~]# yum groupinstall "Development Tools"
# 如果列出的组名为中文，可以这样进行:
[root@node1 ~]# LANG=C yum grouplist

# 继续编辑pkg.yml，在test组中的主机上安装Development tools组
[root@control ansible]# vim pkg.yml
---
- name: install pkgs
  hosts: test
  tasks:
    - name: install web pkgs
      yum:
        name:
          - httpd
          - php
          - php-mysqlnd
        state: present

    - name: install dev group
      yum:
        name: "@Development Tools"   # @表示后面的名字是组名
```

```
        state: present

[root@control ansible]# ansible-playbook pkg.yml

# 系统升级命令
[root@control ansible]# yum update
# 继续编辑pkg.yml，在test组中的主机上升级所有的包到最新版本
---
- name: install pkgs
  hosts: test
  tasks:
    - name: install web pkgs
      yum:
        name:
          - httpd
          - php
          - php-mysqlnd
        state: present

    - name: install dev group
      yum:
        name: "@Development Tools"
        state: present

    - name: update system    # 相当于yum update命令
      yum:
        name: "*"
        state: latest

[root@control ansible]# ansible-playbook pkg.yml
```

ansible变量

facts变量

- facts翻译过来就是事实。
- facts变量是ansible自带的预定义变量，用于描述被控端软硬件信息。
- facts变量通过setup模块获得。

```
# 通过setup模块查看所有facts变量
[root@control ansible]# ansible test -m setup
```

- facts变量是一个大的由{}构成的键值对字典。在{}中，有很多层级的嵌套。可以通过参数过滤出第一个层级的内容。

```
# 查看所有的IPV4地址，filter是过滤的意思
[root@control ansible]# ansible test -m setup -a
"filter=ansible_all_ipv4_addresses"

# 查看可用内存
[root@control ansible]# ansible test -m setup -a
"filter=ansible_memfree_mb"
```

- 常用的facts变量
  - ansible_all_ipv4_addresses：所有的IPV4地址
  - ansible_bios_version：BIOS版本信息
  - ansible_memtotal_mb：总内存大小
  - ansible_hostname：主机名

- 在playbook中使用变量

```
# 显示远程主机的主机名和内存大小。在ansible中，变量使用{{}}表示
# debug模块用于输出信息，常用的参数是msg，用于输出指定内容
[root@control ansible]# vim debug.yml
---
- name: display host info
  hosts: test
  tasks:
    - name: display hostname and memory
      debug:
        msg: "hostname: {{ansible_hostname}}; mem: {{ansible_memtotal_mb}} MB"

[root@control ansible]# ansible-playbook debug.yml
```

自定义变量

- 引入变量，可以方便Playbook重用。比如装包的playbook，包名使用变量。多次执行playbook，只要改变变量名即可，不用编写新的playbook。
- ansible支持10种以上的变量定义方式。常用的变量来源如下：
  - inventory变量。变量来自于主机清单文件
  - facts变量。

- playbook变量。变量在playbook中定义。
- 变量文件。专门创建用于保存变量的文件。推荐变量写入单独的文件。

```
# 使用inventory变量。
[root@control ansible]# vim hosts
[test]
node1 iname="nb"      # 主机变量定义的方法。iname是自定义名称

[proxy]
node2

[webserver]
node[3:4]

[database]
node5

[cluster:children]
webserver
database

[webserver:vars]        # 组变量定义方法。:vars是固定格式
iname="dachui"

# 通过变量创建用户
[root@control ansible]# vim var1.yml
---
- name: test create user
```

```
  hosts: test
  tasks:
    - name: create user
      user:
        name: "{{iname}}"
        state: present

- name: create user in webserver
  hosts: webserver
  tasks:
    - name: create some users
      user:
        name: "{{iname}}"
        state: present

[root@control ansible]# ansible-playbook var1.yml

# 上述两个play也可以合并为一个，如下:
[root@control ansible]# vim var1.yml
---
- name: test create user
  hosts: test,webserver
  tasks:
    - name: create user
      user:
        name: "{{iname}}"
        state: present


# 在playbook中定义变量
# 在test组中的主机上创建用户jack，他的密码是123456
```

```
[root@control ansible]# vim user_jack.yml
---
- name: create user
  hosts: test
  vars:    # 固定格式，用于声明变量
    username: "jack"    # 此处引号可有可无
    mima: "123456"      # 此处引号是需要的，表示数字字
符
  tasks:
    - name: create some users
      user:
        name: "{{username}}"   # {}出现在开头，必须有引
号
        state: present
        password: "{{mima|password_hash('sha512')}}"

[root@control ansible]# ansible-playbook user_jack.yml


# 将变量定义在文件中
[root@control ansible]# vim vars.yml   # 文件名自定义
---
yonghu: rose
mima: abcd
[root@control ansible]# vim user_rose.yml
---
- name: create user
  hosts: test
  vars_files: vars.yml   # vars_files用于声明变量文件
  tasks:
    - name: create some users
```

```
        user:
          name: "{{yonghu}}"
          state: present
          password: "{{mima|password_hash('sha512')}}"

[root@control ansible]# ansible-playbook user_rose.yml
```

补充模块

firewalld模块

- 用于配置防火墙的模块
- 常用选项：
  - port：声明端口
  - permanent：永久生效，但不会立即生效
  - immediate：立即生效，临时生效
  - state：enabled，放行；disabled拒绝
- 防火墙一般默认拒绝，明确写入允许的服务。
- 有一些服务有名字，有些服务没有名字。但是最终都是基于TCP或UDP的某些端口。比如http服务基于TCP80端口。服务名和端口号对应关系的说明文件是：/etc/services
- 配置服务器的防火墙，一般来说只要配置开放哪些服务或端口即可。没有明确开放的，都默认拒绝。
- 应用
  - 在test组中的主机上安装并启动httpd

- 客户端访问服务器的http服务
- 在test组中的主机上安装并启动firewalld
- 客户端访问服务器的http服务
- 在test组中的主机上开放http服务

```
# 配置httpd服务
[root@control ansible]# vim firewall.yml
---
- name: configure test
  hosts: test
  tasks:
    - name: install httpd pkg
      yum:
        name: httpd
        state: present

    - name: start httpd service
      service:
        name: httpd
        state: started
        enabled: yes

[root@control ansible]# ansible-playbook firewall.yml
[root@control ansible]# curl http://192.168.4.11/  # 可访问

# 安装并启动firewalld
[root@control ansible]# vim firewall.yml
---
- name: configure test
  hosts: test
```

```
  tasks:
    - name: install httpd pkg
      yum:
        name: httpd
        state: present

    - name: start httpd service
      service:
        name: httpd
        state: started
        enabled: yes

    - name: install firewalld pkg
      yum:
        name: firewalld
        state: present

    - name: start firewalld service
      service:
        name: firewalld
        state: started
        enabled: yes

[root@control ansible]# ansible-playbook firewall.yml
[root@control ansible]# curl http://192.168.4.11/  # 被拒绝
curl: (7) Failed to connect to 192.168.4.11 port 80: 没有到主机的路由

# 配置防火墙规则，放行http协议
[root@control ansible]# vim firewall.yml
```

```yaml
---
- name: configure test
  hosts: test
  tasks:
    - name: install httpd pkg
      yum:
        name: httpd
        state: present

    - name: start httpd service
      service:
        name: httpd
        state: started
        enabled: yes

    - name: install firewalld pkg
      yum:
        name: firewalld
        state: present

    - name: start firewalld service
      service:
        name: firewalld
        state: started
        enabled: yes

    - name: set firewalld rules
      firewalld:
        port: 80/tcp
        permanent: yes
        immediate: yes
```

```
        state: enabled

[root@control ansible]# ansible-playbook firewall.yml
[root@control ansible]# curl http://192.168.4.11/  # 可访问
```

template模块

- copy模块可以上传文件，但是文件内容固定
- template模块可以上传具有特定格式的文件（如文件中包含变量）
- 当远程主机接收到文件之后，文件中的变量将会变成具体的值
- template模块上传的文件，使用的语法叫Jinja2。
- 常用选项：
  - src：要上传的文件
  - dest：目标文件路径

```
# 使用template模块将含有变量的文件上传到test组中的主机
[root@control ansible]# vim index.j2
Welcome to {{ansible_hostname}} on
{{ansible_eth0.ipv4.address}}

[root@control ansible]# vim templ.yml
---
- name: upload index
  hosts: test
  tasks:
```

```
    - name: create web index
      template:
        src: index.j2
        dest: /var/www/html/index.html

[root@control ansible]# ansible-playbook templ.yml
[root@control ansible]# curl http://192.168.4.11/
Welcome to node1 on 192.168.4.11
[root@node1 ~]# cat /var/www/html/index.html
Welcome to node1 on 192.168.4.11
```

进阶语法

错误处理

- 当Playbook中包含很多任务时，当某一个任务遇到错误，它将崩溃，终止执行

```
# 在test组中的主机上启动mysqld服务，然后创
建/tmp/service.txt
# 因为目标主机上没有mysqld服务，所以它将崩溃，终止执
行。即，不会创建/tmp/service.txt文件
[root@control ansible]# vim myerr.yml
---
- name: my errors
  hosts: test
  tasks:
    - name: start mysqld service
```

```
      service:
        name: mysqld
        state: started
        enabled: yes

    - name: touch a file
      file:
        path: /tmp/service.txt
        state: touch

# 执行playbook，第1个任务就会失败
[root@control ansible]# ansible-playbook myerr.yml
# 到node1上查看，因为第2个任务没有执行，所以文件不会创建
[root@node1 ~]# ls /tmp/service.txt
ls: cannot access '/tmp/service.txt': No such file or directory
```

- 可以指定某一个任务如果出现错误，则忽略它

```
# 编辑myerr.yml，如果myslqd服务无法启动，则忽略它
[root@control ansible]# vim myerr.yml
---
- name: my errors
  hosts: test
  tasks:
    - name: start mysqld service
      service:
        name: mysqld
        state: started
        enabled: yes
```

```
      ignore_errors: yes

    - name: touch a file
      file:
        path: /tmp/service.txt
        state: touch

[root@control ansible]# ansible-playbook myerr.yml
[root@node1 ~]# ls /tmp/service.txt   # 第2个任务已执行
/tmp/service.txt
```

- 通过全局设置，无论哪个任务出现问题，都要忽略

```
[root@control ansible]# vim myerr.yml
---
- name: my errors
  hosts: test
  ignore_errors: yes
  tasks:
    - name: start mysqld service
      service:
        name: mysqld
        state: started
        enabled: yes

    - name: touch a file
      file:
        path: /tmp/mysql.txt
        state: touch

[root@control ansible]# ansible-playbook myerr.yml
```

```
[root@node1 ~]# ls /tmp/mysql.txt
/tmp/mysql.txt
```

触发执行任务

- 通过handlers定义触发执行的任务
- handlers中定义的任务，不是一定会执行的
- 在tasks中定义的任务，通过notify关键通知handlers中的哪个任务要执行
- 只有tasks中的任务状态是changed才会进行通知。

```
# 创建目录，执行追加命令，在目录中建立文件
[root@control ansible]# vim handle.yml
---
- name: handler tasks
  hosts: test
  tasks:
    - name: create a dir
      file:
        path: /tmp/newdir
        state: directory

    - name: exec shell
      shell: "echo hello >> /tmp/newdir/a.txt"

[root@control ansible]# ansible-playbook handle.yml
[root@node1 ~]# cat /tmp/newdir/a.txt
hello
```

```
# 每次执行playbook，shell命令都会执行一次
[root@control ansible]# ansible-playbook handle.yml
[root@node1 ~]# cat /tmp/newdir/a.txt
hello
hello


# 修改任务执行逻辑，只有第一个任务执行，才会触发执行第2个任务
[root@control ansible]# vim handle2.yml
---
- name: handler tasks
  hosts: test
  tasks:
    - name: create a dir
      file:
        path: /tmp/newdir2
        state: directory
      notify: exec shell

  handlers:
    - name: exec shell
      shell: "echo hello >> /tmp/newdir2/a.txt"

# 第一次运行时，不存在/tmp/newdir2，第1个任务的状态将会是changed，它将触发执行exec shell
[root@control ansible]# ansible-playbook handle2.yml
[root@node1 ~]# cat /tmp/newdir2/a.txt
hello
```

```
# 再次运行时，/tmp/newdir2目录已存在，第1任务的状态将会是ok/success，那么它就不会触发exec shell任务
[root@control ansible]# ansible-playbook handle2.yml
[root@node1 ~]# cat /tmp/newdir2/a.txt
hello
```

when条件

- 只有满足某一条件时，才执行任务
- 常用的操作符：
  - ==：相等
  - !=：不等
  - >：大于
  - <：小于
  - <=：小于等于
  - >=：大于等于
- 多个条件或以使用and或or进行连接
- when表达式中的变量，可以不使用{{}}

```
# 当test组中的主机内存大于2G的时候，才安装mariadb-server
[root@control ansible]# vim when1.yml
---
- name: install mariadb
  hosts: test
  tasks:
```

```
    - name: install mariadb pkg
      yum:
        name: mariadb-server
        state: present
      when: ansible_memtotal_mb>2048

# 如果目标主机没有2GB内存，则不会安装mariadb-server
[root@control ansible]# ansible-playbook when1.yml



# 多条件。系统发行版是RedHat8才执行任务
# /etc/motd中的内容，将会在用户登陆时显示在屏幕上
[root@control ansible]# vim motd
 -------------
< hello world >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

[root@control ansible]# vim when2.yml
---
- name: when condition
  hosts: test
  tasks:
    - name: modify /etc/motd
      copy:
        dest: /etc/motd
```

```
    src: motd
  when: >     # 以下三行合并成一行
    ansible_distribution == "RedHat"
    and
    ansible_distribution_major_version == "8"

[root@control ansible]# ansible-playbook when2.yml
```

cowsay:

```
[root@node1 ~]# yum install -y cowsay-3.04-4.el7.noarch.rpm
[root@node1 ~]# cowsay hello world    # 默认是奶牛形象
[root@node1 ~]# cowsay -l   # 查看可用形象
[root@node1 ~]# cowsay -f sheep hello world

[root@node1 ~]# cowsay -l > ss.txt
[root@node1 ~]# vim ss.txt    # 删除第1行的说明
[root@node1 ~]# for i in $(cat ss.txt)
> do
> echo "--------------$i----------------"
> cowsay -f $i hello
> sleep 3
> echo "-------------------------------"
> done
```