# 第二十二章 Ansible二次开发

本节所讲内容：

22.1 Ansible 关于yaml的讲解

22.2 Ansible playbook 剧本

22.3 Ansible api 介绍

22.4 Ansible api的案例使用

## 22.1 Ansible 关于yaml的讲解

YAML 语言（发音 /ˈjæməl/ ）的设计目标，就是方便人类读写。它实质上是一种通用的数据串行化格式。

它的基本语法规则如下。

大小写敏感

使用缩进表示层级关系

缩进时不允许使用Tab键，只允许使用空格。

缩进的空格数目不重要，只要相同层级的元素左侧对齐即可

YAML 支持的数据结构有三种。

对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes） / 字典（dictionary）

数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）

纯量（scalars）：单个的、不可再分的值

Yaml缩进类似Python，数据结构类似java

对象

name: persion #键值对中间有一个空格

数组

- python  
- php  
- linux

纯量

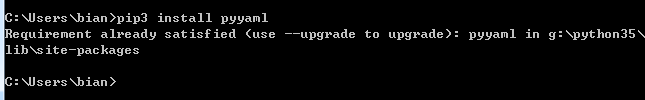
12.3 *#浮点*3 *#整数*true *#布尔值*~ *#空*abc *#字符串*

我们在工作当中，我们采用的是混合的配置

*#混合*xuegod:   
 - python  
 - linux  
 - php  
 - web  
teacher:   
 - while  
 - for  
 - cd  
 - mk  
description:   
 name: while  
 age: 12

Python 操作yaml

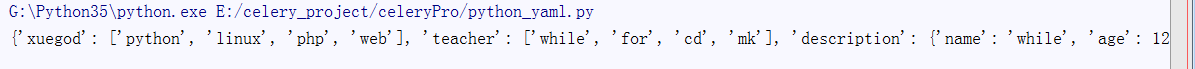
安装Python yaml模块

pip3 install pyyaml 

读yaml文件

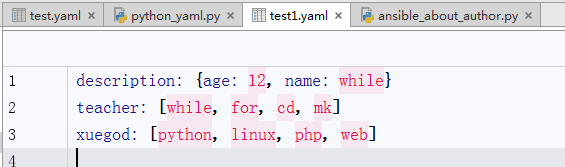
import yaml  
with open("test.yaml","rb") as f:  
 data = yaml.load(f)  
print(data)

结果如下(读取出来的是json)



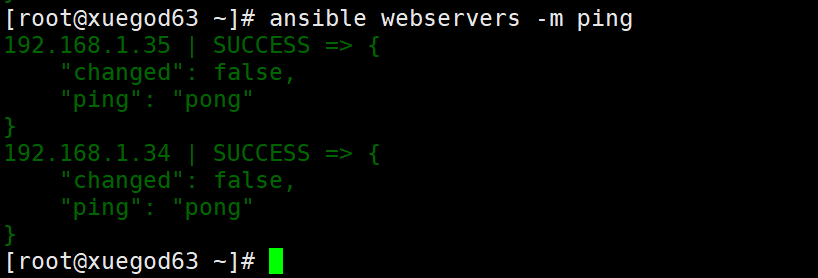
写yaml

with open("test1.yaml","w") as f:  
 yaml.dump(content,f)



## 22.2 Ansible playbook 剧本

首先，我们保证，我们的ansible出于畅通状态



假如批量执行的是单条命令，那么我们上面的操作就很优秀，但是如果我们是要进行一个复杂的操作，不如：lamp的安装，这样的操作方式还是有些麻烦，所以，我们引入了剧本(palybook)模式：

Playbook是一个不同于使用ansible命令行执行方式的模式，功能更强大更灵活。

playbooks使用步骤：

定义剧本

- name： task description #任务描述信息

module\_name: module\_args #需要使用的模块名字： 模块参数

执行剧本

ansible-playbook 执行 命令

实战-使用Playbook批量部署多台LAMP环境

Playbook常用文件夹作用：

files：存放需要同步到异地服务器的源码文件及配置文件；

handlers：当服务的配置文件发生变化时需要进行的操作，比如：重启服务，重新加载配置文件； ['hændləz]

meta：角色定义，可留空；  ['metə] 元

tasks：需要进行的执行的任务；  #任务

templates：用于执行lamp安装的模板文件，一般为脚本；  ['templɪts] 模板

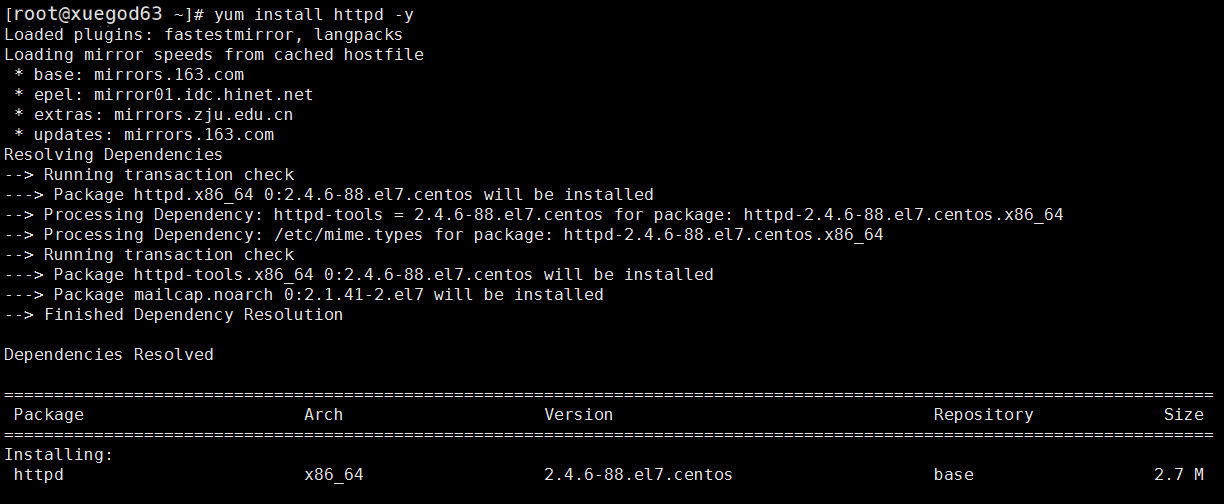
vars：本次安装定义的变量

开始剧本

首先，我们可以在ansible服务器上安装LAMP环境，然后，再将配置文件通过ansible拷贝到远程主机上

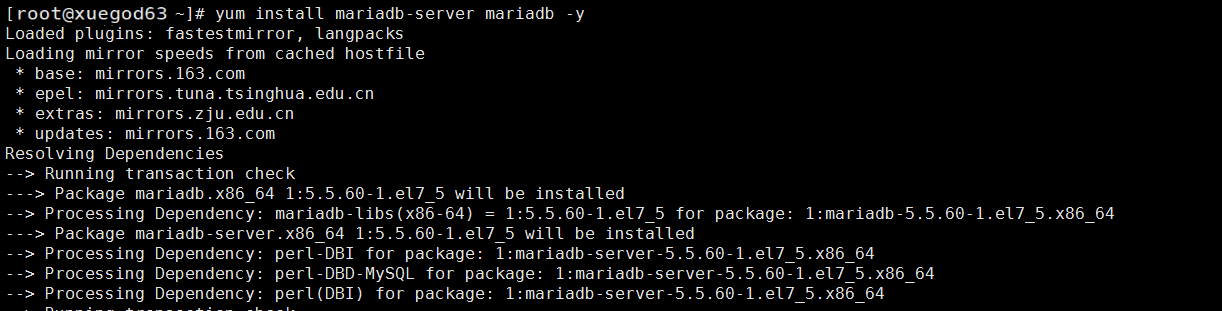
1. 安装apache服务器

yum install httpd



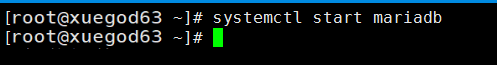
1. 安装maria（mysql）

yum install mariadb-server mariadb



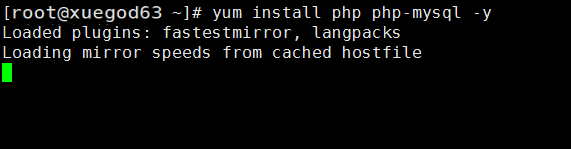
重启数据库

systemctl start mariadb



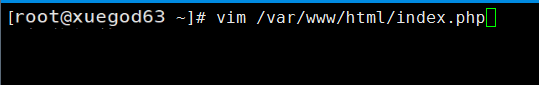
1. 安装PHP和php-mysql模块

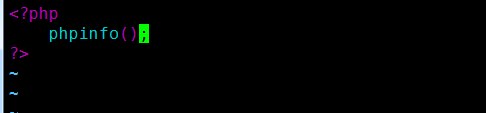
yum install php php-mysql -y



提供php测试页面

vim /var/www/html/index.php





<?php

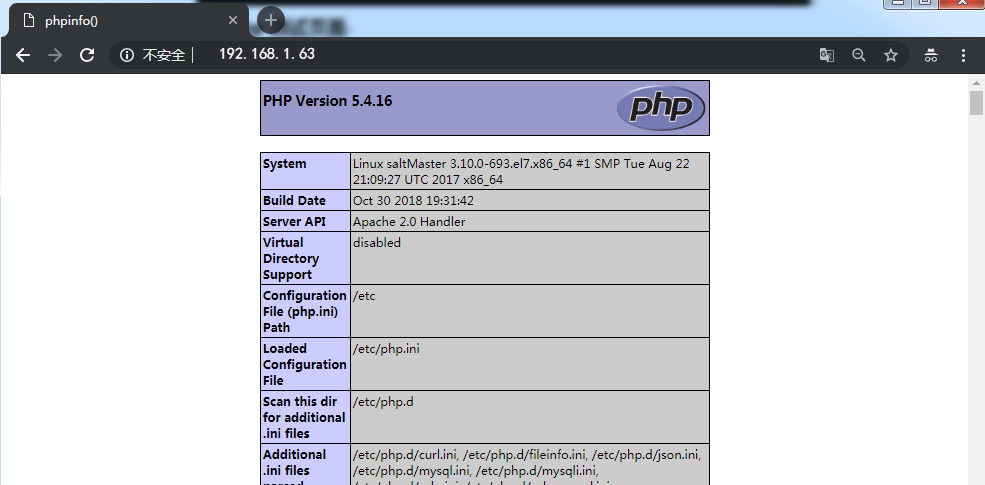
phpinfo();

?>

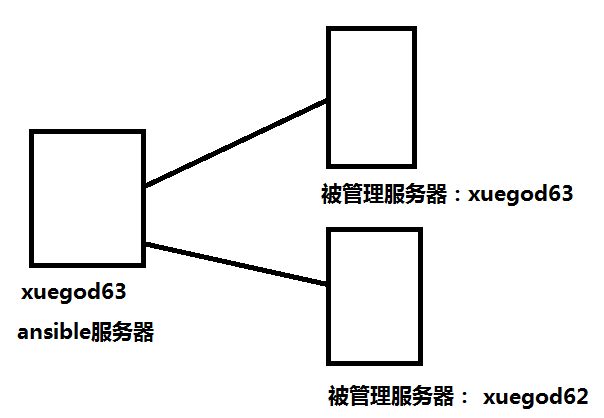
重启http服务器，然后查看效果

systemctl restart httpd（注意关闭防火墙）

C:\Users\Administrator\Desktop\011.png011



实战：使用playbook创建一个LAMP构建的任务



创建相关文件

mkdir -pv /etc/ansible/lamp/roles/{prepare,httpd,mysql,php}/{tasks,files,templates,vars,meta,default,handlers}

我们将上面搭建成功的LAMP环境的httpd和MySQL的配置文件拷贝到对应目录下

[root@xuegod63 ~]# cd /etc/ansible/

[root@xuegod63 ansible]# cp /etc/httpd/conf/httpd.conf lamp/roles/httpd/files/

[root@xuegod63 ansible]# cp /etc/my.cnf lamp/roles/mysql/files/

写prepare角色的playbooks（tasks）

[root@xuegod63 tasks] cd /etc/ansible/lamp/roles/prepare/tasks

[root@xuegod63 tasks]# vim main.yml

内容如下：

- name: delete yum config

shell: rm -rf /etc/yum.repos.d/\*

- name: provide yumrepo file

shell: wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

- name: clean the yum repo

shell: yum clean all

- name: clean the iptables

shell: iptables -F

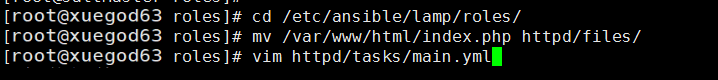


构建httpd的任务

[root@xuegod63 roles]# cd /etc/ansible/lamp/roles/

[root@xuegod63 roles]# mv /var/www/html/index.php httpd/files/

[root@xuegod63 roles]# vim httpd/tasks/main.yml



- name: webserver install

yum: name=httpd state=present

- name: provide test page

copy: src=index.php dest=/var/www/html

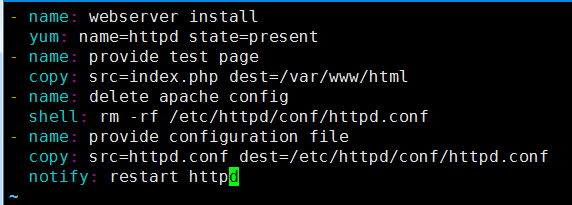
- name: delete apache config

shell: rm -rf /etc/httpd/conf/httpd.conf

- name: provide configuration file

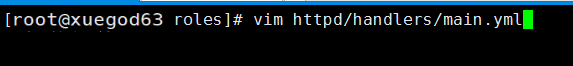
copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf

notify: restart httpd



构建httpd的handlers

[root@xuegod63 roles]# vim httpd/handlers/main.yml



- name: restart httpd

service: name=httpd enabled=yes state=restarted

- name: install the mysql

yum: name=mariadb-server state=present

- name: mkdir date directory

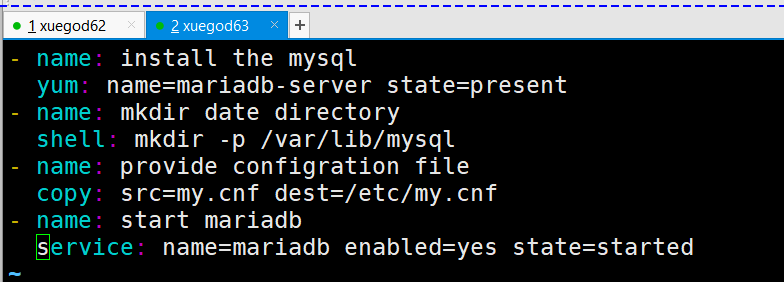
shell: mkdir -p /var/lib/mysql

- name: provide configration file

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

- name: start mariadb

service: name=mariadb enabled=yes state=started



构建php任务

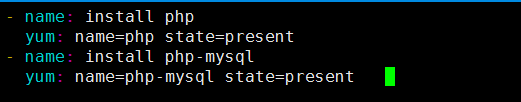
[root@xuegod63 roles]# vim php/tasks/main.yml

- name: install php

yum: name=php state=present

- name: install php-mysql

yum: name=php-mysql state=present



定义整个的任务

[root@xuegod63 roles]# pwd

/etc/ansible/lamp/roles

[root@xuegod63 roles]# vim site.yml

- name: LAMP build

remote\_user: root

hosts: webservers

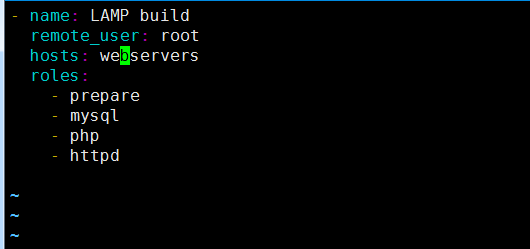
roles:

- prepare

- mysql

- php

- httpd



免密：1 ssh-keygen 回车\*3

2 ssh-copy-id [root@192.168.\*.\*](mailto:root@192.168.*.*)

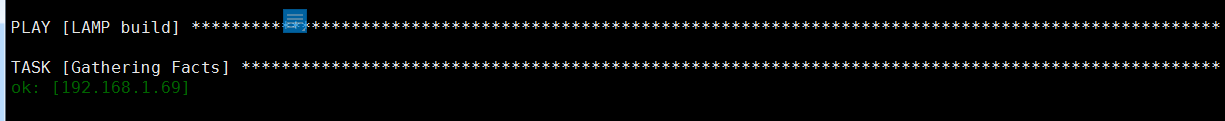
断开ssh链接方法

法1：Ctrl+D

法2：输入 logout

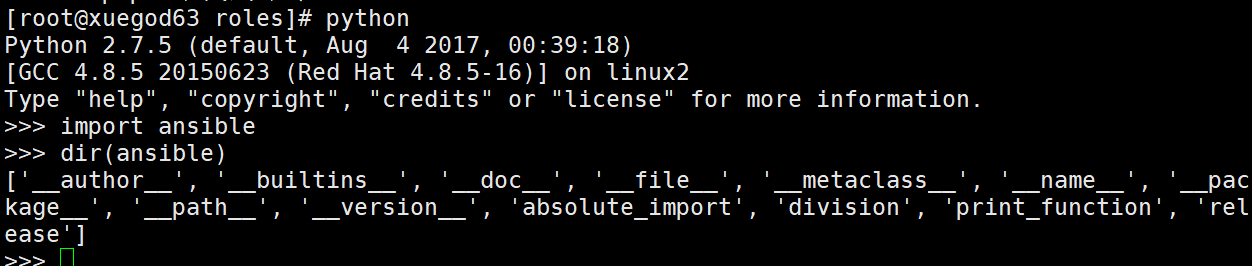
执行剧本查看结果

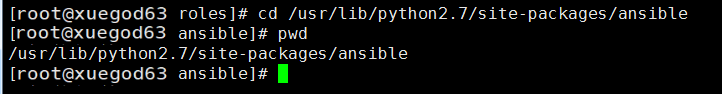
执行命令ansible-playbook -i /etc/ansible/hosts /etc/ansible/lamp/roles/site.yml



## 22.3 Ansible api 介绍

Ansible是由Python编写的一个批量操作服务器的工具，已经实现了大部分的批量操作的功能，但是在运维的过程当中，难免会遇到需要自己定义，描述的需求。所以我们要进行二次开发，与之前zabbix二次开发不同的是，ansible的二次开发的思路不在api接口方向，而是在我们安装部署ansible之后，会自动的有一个ansible的模块安装在Python当中，提供给我们进行二次开发使用。





在ansible2.0之前，ansible api 二次开发是相当简单的一个过程。用官方的描述来说就是so simple，但是but在2.0之后，如果没有相当的Python基础，那么搞定ansible不是那么容易的事情。

ansible2.0更贴近于ansible cli的常用命令执行方式，不同于上一版本只能发送单个命令或playbook；而更推荐用户在调用ansibleAPI的时候，将playbook的每个task拆分出来，获取每个task的结果。能够跟灵活处理在执行批量作业过程中的各种反馈。

1、将执行操作的队列模型，包含各类环境参数设置，归结到“ansible.executor.task\_queue\_manager”类中

2、将执行过程中的各个task的设置，或者说playbook中的编排内容，归结到“ansible.playbook.play”中

上述两个东西，几乎囊括了可以在执行过程中设置的所有参数，足够灵活，但是也让人抓狂，相当于需要自己写一个1.9版本中的runner。

参数描述

inventory --> 由ansible.inventory模块创建，用于导入inventory文件 #清单

variable\_manager --> 由ansible.vars模块创建，用于存储各类变量信息

loader --> 由ansible.parsing.dataloader模块创建，用于数据解析

options --> 存放各类配置信息的数据字典

passwords --> 登录密码，可设置加密信息

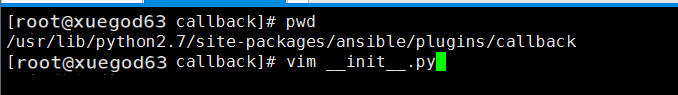
stdout\_callback --> 回调函数

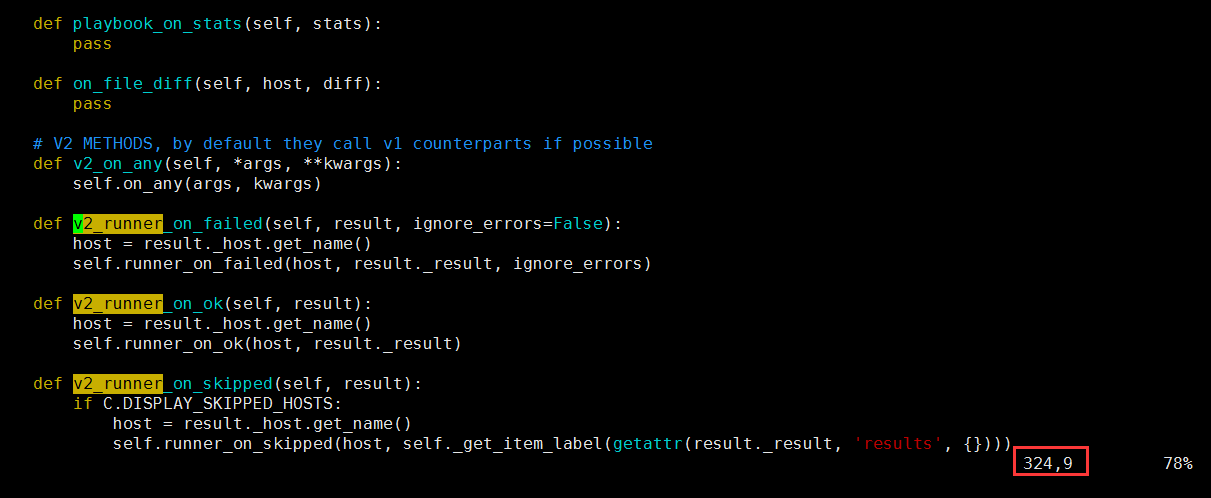
ansible.playbook.play

ansible.playbook是一个原生模块，既用于CLI（命令行界面）也用于API。从源码可以看出来：

ansible.playbook.play（ansible/playbook/play.py）

## 22.4 Ansible api的案例使用





Ansible 2.0api执行 ls

*#! /usr/bin/python  
#coding:utf-8*import json  
import shutil  
from collections import namedtuple  
*# DataLoader用于解析yaml/json/ini文件*from ansible.parsing.dataloader import DataLoader  
*# VariableManager用于分析ansible用到的变量*from ansible.vars.manager import VariableManager  
*# InventoryManager用于分析主机文件*from ansible.inventory.manager import InventoryManager  
from ansible.playbook.play import Play  
*# task\_queue\_manager管理任务队列*from ansible.executor.task\_queue\_manager import TaskQueueManager  
from ansible.plugins.callback import CallbackBase  
class ResultCallback(CallbackBase):  
 def v2\_runner\_on\_ok(self,result,\*\*kwargs):  
 host = result.\_host  
 print(json.dumps({host.name: result.\_result},indent=4))  
*#创建元组，将选项加入，如：connection：连接，module\_path：模块路径，forks：进程数量等  
#Options = namedtuple('Options',['contention','module\_path','forks','become','become\_method','become\_user','check','diff'])  
# 创建具体的实例对象  
# connection有三个选择local/ssh/smart  
# local表示在本机执行，ssh表示通过ssh协议执行，smart表示自动选择  
#options = Options(contention = 'local',module\_path = ['/to/mymodules'],forks=10,become=None,become\_method=None,become\_user=None,check=False,diff=False)  
#负责查找和读取YAML、JSON和INI文件*loader = DataLoader()  
*# 用于存储加密密码、远程连接密码等*passwords = dict(vault\_pass='secret')  
  
results\_callback = ResultCallback()  
*# 声明被ansible管理的主机有哪些，可以把各主机用逗号分开形成字符串  
# 也可以使用主机清单文件路径，将路径放到列表中*inventory = InventoryManager(loader=loader,sources="/etc/ansible/hosts")  
*#变量管理器负责合并所有不同的源，从而为每个上下文提供变量的统一视图。*variable\_manager = VariableManager(loader=loader,inventory=inventory)  
*#脚本执行时屏幕显示的结果结构及信息*play\_source = dict(   
 name = "Ansible Play",*# Play名称* hosts = "localhost", *# 在哪些主机上执行命令* gather\_facts = "no",*# 不收集主机信息  
 # 以下是执行的命令* tasks =[dict(action=dict(module='shell', args='ls'), register='shell\_out.stdout'),]  
)  
*#上面导入的对象，play\_source执行的任务有哪些，变量到的分析*play = Play.load(play\_source,variable\_manager=variable\_manager,loader=loader)  
  
tqm=None  
try:  
 *#tqm是taskQueueManager任务管理器生成的实例* tqm= TaskQueueManager(  
 inventory=inventory, *#主机清单* variable\_manager=variable\_manager, *#参数管理* loader=loader, *#json等语法分析  
 # options=options,#选项* passwords=passwords,*#密码* stdout\_callback=results\_callback,  
 )  
 result=tqm.run(play)*# tqm实例中的run方法开始执行play中的任务* print(result)  
except Exception as e:  
 print e  
finally:  
 if tqm is not None:  
 print(tqm.cleanup())*#清理*

效果图



总结：

22.1 Ansible 关于yaml的讲解

22.2 Ansible playbook 剧本

22.3 Ansible api 介绍

22.4 Ansible api的案例使用