一、redis群集

1.Redis cluster介绍

redis-cluster的优势

2.Redis数据分布

3.虚拟槽分区

二、Redis集群原理

1.群集角色工作原理介绍

2.Redis架构细节信息 --了解

3.Redis-cluster选举

4.Rediscluster中服务器节点都使用两个tcp端口号

三、Redis群集部署

1.安装redis (1-6 操作相同)

上传软件包---这里不能用全部会话，会卡死

2.修改主配置文件使其支持群集(1-6 操作相同)

3.重启redis服务并查看6379和16379端口

4.源码安装ruby

5.使用脚本创建群集

6.查看集群状态

7.测试群集

redis多实例部署优化调整---记得给亮哥要

一、Ansible概述

1.Ansible特点

2.Ansible应用

1.使用者 ---运维人员使用

2.Ansible工具集合

3.作用对象

二、Ansbile的搭建

1.实验环境

一、redis群集

我们日常在对于redis的使用中，经常会遇到一些问题

➢高可用问题，如何保证redis的持续高可用性

➢容量问题，单实例redis内存无法无限扩充，达到32G后就进入了64位世界，性能下降

➢并发性能问题，redis 号称单实例10万并发，但也是有尽头的

1.Redis cluster介绍

Redis cluster是redis的分布式解决方案，

在redis3. 0版本中正式推出的采用的是hash slot (hash 槽)，

可以将多个Redis实例整合在一起，形成一个集群，

也就是将数据分散到集群的多台机器上。

有效的解决了reids分布式方面的需求,当遇到单机内存、开发、流量等瓶颈时，

可以采用cluster架构达到负载均衡的目的。

redis-cluster的优势

1. 官方推荐，毋庸置疑。

2. 去中心化，集群最大可增加1000个节点，性能随节点增加而线性扩展

redis集群一般是6台机器，三主三从

3. 管理方便，后续可自行增加或摘除节点，移动分槽等等

4. 简单，易上手。

2.Redis数据分布

分布式数据库主要解决把整个数据集按照分区规则映射到多个节点的问题，

即把数据集划分到多个节点上，

每个节点负责整个数据的一个子集

常见的分区规则有哈希分区和顺序分区。

Rediscluster采用哈希分区规则，因此接下来会讨论哈希规则，

常见的哈希分区有以下几种:

1. 节点取余
2. 一致性哈希分区
3. 虚拟槽分区

redis cluster采用虚拟槽分区

3.虚拟槽分区

虚拟槽分区巧妙地使用了哈希空间，

使用分散度良好的哈希函数把所有的数据映射到一个固定的范围内的整数集合

RedisCluster槽的范围是0-16383。（redis默认带的数据库是16个--0~15）

槽是集群数据管理和迁移的基本的单位。

采用大范围槽的主要目的是为了方便数据的拆分和集群的扩展(同时这也是redis的优点)，

使每个节点负责一定数量的值。

二、Redis集群原理

Rediscluster是一个无中心的结构，其每一个节点都会保存着数据和整个集群的状态。

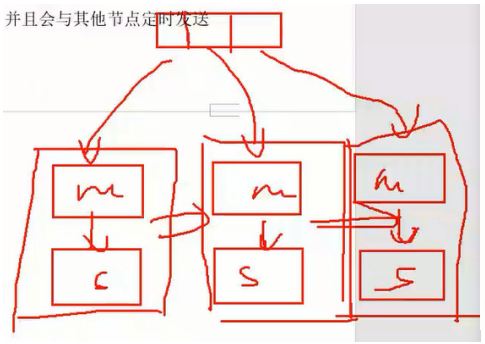
每个节点都会保存其他节点的信息，知道其它节点所负责的槽，

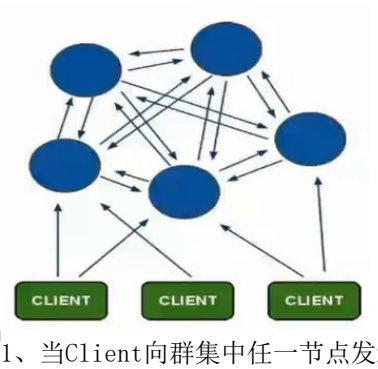
并且会与其他节点定时发送心跳信息能够及时感知整个集群中异常的节点。

类似于raid0，但不完全是

数据分布式存储

Redis群集原理图





1、当Client向群集中任一节点发送与数据库有关命令时

2、接收命令的Redis 节点会计算出命令属于哪个槽，并检查这个槽是否指派了给自己

- 1) 如果键所在的槽正好指派给当前节点，那么节点直接执行这个命令
- 2) 如果键所在的槽没有指派给当前节点，那么节点会向客户端返回一个moved错误，并指引客户端转向(redirect) 正确的节点，在发送希望执行的命令

1.群集角色工作原理介绍

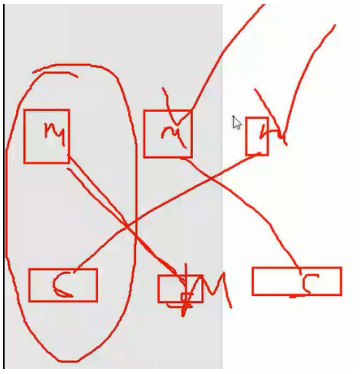
集群角色有Master和Slave。群集中master之间分配slots（槽）[范围: 0-16383]，

一共16384个slot。Slave 向它指定的master同步数据，实现备份，

当其中的一个master无法提供服务时，该master的slave将提升为master，

以保证群集键slot的完整性，

当其中的某一个master和他的slave都失效，导致slot 不完整，
群集将会失效，这时就需要人工处理了
注意：不要把master和slave放在一台机器上



- redis的健康状态检查主要靠ping

Redis群集搭建好以后，群集中的每个节点都会定期地向其他节点发送ping消息，
如果ping消息没有在规定时间内返回pong，
那么发送ping消息的节点就会将其标记为疑似下线(probable fail, PFALL)，
各个节点会通过相互发送消息的方式来交换群集中各个节点的状态信息，
如果在一个集群里面，半数以上的主节点都将某个主节点X报告为疑似下线，
那么这个主节点X将被标记为已下线(FAIL)，
同时会向集群广播一条关于主节点X的FAIL消息，
所有收到这条FAIL消息的节点都会立即将主节点X标记为已下线。

- 对所有机器的健康状态的检查

当群集中需要增加或减少机器时，
我们需要将已经指派给某个节点(源节点:原来slot所在的节点)的槽更改为指派给另一个节点(目标节点:将原来的slot分配给谁)，
并且会将原节点的键值移动到目标节点
Redis群集的从新分片操作是由redis的群集管理软件redis-trib负责执行的，
不支持自动分片，而且还需要自己计算从哪些节点上迁移多大slot，
在重新分片的过程中群集不需要下线，并且原节点和目标节点都可以处理请求

2.Redis架构细节信息 --了解

1.所有的reids彼此互联的机制(ping-pong机制)，
内部使用二进制协议优化传输速度和带宽
2.节点的失效(fail) 在群集中超过半数的主(master) 节点检测失效时才会失效
3.客户端与redis节点直连，不需要中间代理(proxy)层，客户端不需要连接群集所有节点，连接群集中任何几个节点都可以
4. Redis-cluster 把所有的物理及节点映射到[0-16383]slot上，cluster 负责维护
node<->slot<->key 节点--槽--键
redis里面存储的是键值对（key-value）

3.Redis-cluster选举

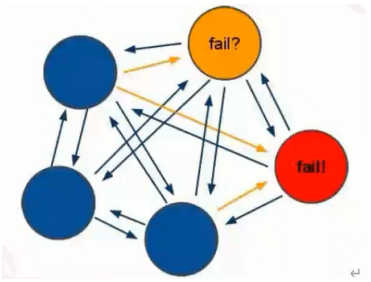
选举过程是群集中所有master参与，
如果半数以上的master节点与当前master节点通信超时(cluster-node-timeout) ，
认为当前的master节点挂掉，出现以下两种情况将认为群集不可用
1.当群集不可用时， 所有对进群的操作都将不可用
2. 收到((error) CLUSTERDBWN The cluster di down) 错误

1）如果群集中任意的master宕机，没有slave直接进入fail状态，

此时之前分配到master上的槽[0-16383 范围中]

在整个群集中将会出现slot不完整进入fail状态

2）如果群集中超过半数的master挂掉，无论是否有slave群集都会进入fail状态



4.Rediscluster中服务器节点都使用两个tcp端口号

1.6379端口号:用于服务于client连接

2.16379端口号:用于群集总线，即使用二进制协议的节点到节点通信通道。

节点使用群集总线进行故障检测，配置更新，故障转移授权等，

如果开启了防火墙需要注意开放两个端口

三、Redis群集部署

实验环境

准备6台CentOS7. 5系列Linux虚拟机，

IP范围为:192.168. 200. 107 -192. 168. 200. 112

1.安装redis (1-6 操作相同)

上传软件包---这里不能用全部会话，会卡死

redis-4.0.10.tar.gz

redis-4.0.1.gem

ruby-2.5.1.tar.gz

```
[root@localhost ~]# tar xf redis-4.0.10.tar.gz -C /usr/src/
[root@localhost ~]# cd /usr/src/redis-4.0.10/ && make && make install
[root@localhost redis-4.0.10]# ./utils/install_server.sh
```

接下来在全部会话敲回车就行了

```
[root@localhost redis-4.0.10]# netstat -lnpt | grep redis
```

2.修改主配置文件使其支持群集(1-6 操作相同)

注意：redis的登录密码在生产环境中一定要配置，没有密码会遭到攻击，注意安全性

```
[root@localhost ~]# vim /etc/redis/6379.conf
70 bind 192.168.200.107      # 自身IP地址
815  cluster-enabled yes        # 去掉行前#启动集群
 823 # cluster-config-file nodes-6379.conf   # 去掉行前# 设置群集配置文件
```

3.重启redis服务并查看6379和16379端口

```
[root@localhost ~]# /etc/init.d/redis_6379 restart
Stopping ...
Redis stopped
Starting Redis server...
[root@localhost ~]# netstat -lnpt | grep 6379
tcp        0      0 192.168.200.107:6379    0.0.0.0:*               LISTEN      7247/redis-server 1
```

```
tcp        0      0 192.168.200.107:16379   0.0.0.0:*               LISTEN      7247/redis-server 1
```

4.源码安装ruby

创建群集需要用到ruby的一一个脚本，

在创建群集前需要先安装ruby运行环境的redis客户端，

该操作只需在其中一台节点上操作即可，下载最新版本的ruby

192.168.200.107：

```
[root@localhost ~]# yum -y install rubygems
[root@localhost ~]# yum -y install gcc gcc-c++ make zlib
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * epel: hkg.mirror.rackspace.com
软件包 gcc-4.8.5-28.el7.x86_64 已安装并且是最新版本
软件包 gcc-c++-4.8.5-28.el7.x86_64 已安装并且是最新版本
软件包 1:make-3.82-23.el7.x86_64 已安装并且是最新版本
软件包 zlib-1.2.7-17.el7.x86_64 已安装并且是最新版本
[root@localhost ruby-2.5.1]# tar xf ruby-2.5.1.tar.gz -C /usr/src/
[root@localhost ruby-2.5.1]# cd /usr/src/ruby-2.5.1/
[root@localhost ruby-2.5.1]# ./configure --prefix=/usr/local/ruby2.5 && make && make install
```

配置文件/etc/profile

```
[root@localhost ruby-2.5.1]# vim /etc/profile
#末尾加入
export RUBY_HOME=/usr/local/ruby2.5        #指定安装路径
export REDIS_HOME=/usr/local/redis
# 注意：这里不用加入路径了，因为安装完redis后可以直接用它的命令的，没必要优化路径了
```

意思是不用这条命令了？

```
export PATH=$RUBY_HOME/bin:$PATH
```

配置redis

source使上面的配置生效

```
[root@localhost ruby-2.5.1]# source /etc/profile
```

gem安装一下redis相关的包

```
[root@localhost ruby-2.5.1]# gem install redis --version 4.0.1
```

查看版本

```
[root@localhost ruby-2.5.1]# ruby -v
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]
```

关防火墙

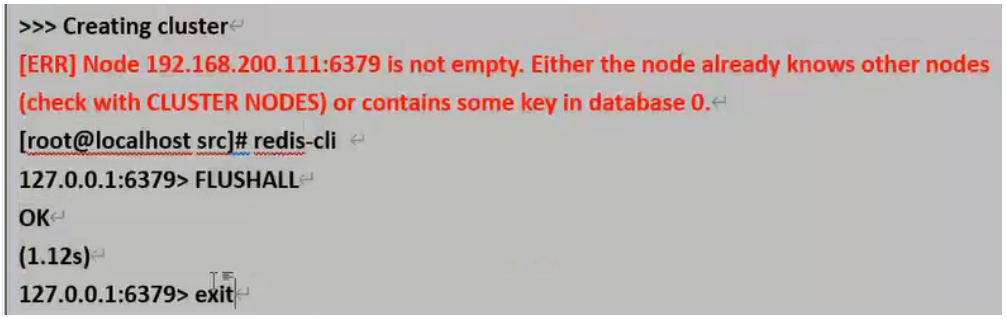
```
[root@localhost ruby-2.5.1]# iptables -F
[root@localhost ruby-2.5.1]# systemctl stop firewalld
[root@localhost ruby-2.5.1]# setenforce 0
setenforce: SELinux is disabled
```

5.使用脚本创建群集

```
[root@localhost src]# ./redis-trib.rb create --replicas 1 192.168.200.107:6379 192.168.200.108:6379
192.168.200.109:6379 192.168.200.110:6379 192.168.200.111:6379 192.168.200.112:6379
Can I set the above configuration? (type 'yes' to accept): yes
```

到这呢，集群就好了

如果有报错的解决方案：



6.查看集群状态

```
[root@localhost ~]# cd /usr/src/redis-4.0.10/src/
[root@localhost src]# ./redis-trib.rb create --replicas 1 192.168.200.107:6379 192.168.200.108:6379
192.168.200.109:6379 192.168.200.110:6379 192.168.200.111:6379 192.168.200.112:6379
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.200.107:6379
192.168.200.108:6379
192.168.200.109:6379
Adding replica 192.168.200.111:6379 to 192.168.200.107:6379
Adding replica 192.168.200.112:6379 to 192.168.200.108:6379
Adding replica 192.168.200.110:6379 to 192.168.200.109:6379
M: 27e5f97914cc42a85778851bbd368f9da3b18e6d 192.168.200.107:6379
   slots:0-5460 (5461 slots) master
M: d53445c20395ad7fb6dba3cec2100aca62685b36 192.168.200.108:6379
   slots:5461-10922 (5462 slots) master
M: 4292e35871049023bb4c8f9391ed55b2f0f0b4df 192.168.200.109:6379
   slots:10923-16383 (5461 slots) master
S: 76d302a898892a70f4a8adb688cfa9b851684396 192.168.200.110:6379
   replicates 4292e35871049023bb4c8f9391ed55b2f0f0b4df
S: 6c924063792be33e7a10e605d2769901235125c4 192.168.200.111:6379
   replicates 27e5f97914cc42a85778851bbd368f9da3b18e6d
S: 9aec8ce62d2330a7a9982e5629f958ff6f66d662 192.168.200.112:6379
   replicates d53445c20395ad7fb6dba3cec2100aca62685b36
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join......
>>> Performing Cluster Check (using node 192.168.200.107:6379)
M: 27e5f97914cc42a85778851bbd368f9da3b18e6d 192.168.200.107:6379
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
```

```
S: 6c924063792be33e7a10e605d2769901235125c4 192.168.200.111:6379
   slots: (0 slots) slave
   replicates 27e5f97914cc42a85778851bbd368f9da3b18e6d
M: d53445c20395ad7fb6dba3cec2100aca62685b36 192.168.200.108:6379
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: 9aec8ce62d2330a7a9982e5629f958ff6f66d662 192.168.200.112:6379
   slots: (0 slots) slave
   replicates d53445c20395ad7fb6dba3cec2100aca62685b36
S: 76d302a898892a70f4a8adb688cfa9b851684396 192.168.200.110:6379
   slots: (0 slots) slave
   replicates 4292e35871049023bb4c8f9391ed55b2f0f0b4df
M: 4292e35871049023bb4c8f9391ed55b2f0f0b4df 192.168.200.109:6379
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@localhost src]# ./redis-trib.rb check 192.168.200.109:6379
随便哪个几点都可以查看
>>> Performing Cluster Check (using node 192.168.200.109:6379)
M: 4292e35871049023bb4c8f9391ed55b2f0f0b4df 192.168.200.109:6379
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
M: d53445c20395ad7fb6dba3cec2100aca62685b36 192.168.200.108:6379
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
M: 27e5f97914cc42a85778851bbd368f9da3b18e6d 192.168.200.107:6379
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
S: 76d302a898892a70f4a8adb688cfa9b851684396 192.168.200.110:6379
   slots: (0 slots) slave
   replicates 4292e35871049023bb4c8f9391ed55b2f0f0b4df
S: 6c924063792be33e7a10e605d2769901235125c4 192.168.200.111:6379
   slots: (0 slots) slave
   replicates 27e5f97914cc42a85778851bbd368f9da3b18e6d
S: 9aec8ce62d2330a7a9982e5629f958ff6f66d662 192.168.200.112:6379
   slots: (0 slots) slave
   replicates d53445c20395ad7fb6dba3cec2100aca62685b36
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
```

[OK] All 16384 slots covered.

7.测试群集

```
[root@localhost src]# redis-cli -h 192.168.200.112 -p 6379       # 从
192.168.200.112:6379> set name sofia
(error) MOVED 5798 192.168.200.108:6379
192.168.200.112:6379> exit
[root@localhost src]# redis-cli -h 192.168.200.108 -p 6379          # 主
192.168.200.108:6379> set name sofia
OK
192.168.200.108:6379> get name
"sofia"
192.168.200.108:6379> exit
```

无数据中心

```
[root@localhost src]# redis-cli -h 192.168.200.107 -p 6379      # slot槽是合理分配的
192.168.200.107:6379> set a aa
(error) MOVED 15495 192.168.200.109:6379
192.168.200.107:6379> exit
[root@localhost src]# redis-cli -h 192.168.200.109 -p 6379
192.168.200.109:6379> set a aa
OK
192.168.200.109:6379> exit
```

到这集群部署好了

redis多实例部署优化调整---记得给亮哥要

生产环境中还是有很多公司在使用的

一、Ansible概述

Ansible是最近非常火的一款开源运维自动化工具，通过Ansible可以实现运维自动化，
提高运维工程师的工作效率，减少人为失误，
Ansible可以通过本身集成的非常丰富的模块实现各种管理任务，
其自带模块数量已超过上千个，更为重要的是，它的操作非常简单，
即使新手也比较容易上手，Ansible 提供的功能却非常丰富，
在运维领域，几乎可以实现任何事情。目前属于RedHat公司产品，
官方地址: https://www.ansible.com/.

1.Ansible特点

Ansible自2012年发布以来，很快在全球流行，其特点表现如下:。

1. Ansible 基于python开发，运维工程师对其二次开发相对较容易;
2. Ansible 拥有丰富的内置模块，基本可以满足一切要求;
3.管理模式非常简单，一条命令可以影响上千台机器;
4.无客户端模式设计，底层基于SSH通信;
5. Ansible 发布后也陆续被AWS， Google Cloud Platfrom，Microsoft Azure，Cisco， HP ，VMware,Twitter等大公司接纳并投入使用;

2.Ansible应用

Ansible没有客户端，也不需要在被管理主机添加任何代理程序，

通过SSH完成底层通信，而SSH在Linux的发型版本中默认已经安装并启用，

而在Windows 系统下则依赖于PowerShell,

Ansible 要求管理端必须是Linux系统，

在管理节点通过应用模块将指令发送到被管理主机上，

并在执行完毕后自动删除产生的临时文件，

根据Ansible使用过程中不同角色，可将其分为三个部分。

1.使用者如何使用Ansible实现自动化运维?

2. Ansible 的工具集，Ansible 可以实现的功能?

3. 作用对象，Ansible可以影响哪些主机?

1.使用者 ---运维人员使用

如图所示: Ansible 使用者可以采用多种方式和Ansible交互，图中我们展示了四种方式

1) CMDB: 类似jumpserver，是一个平台

CMDB 系统存储和管理着企业IT架构中的各种配置信息，

是构建ITL项目核心工具，

运维人员可以组合CMDB和Ansible,

通过CMDB直接下发指令调用Ansible工具完成操作者所希望达到的目标。

2) PUBLIC/PRIVATE 方式， --开发人员使用

Ansible 除了丰富的内置模块外。

同时提供丰富的API语言接口,如PHP，Python, PERL 等多种流行语言，

基于PUBLIC/PRIVATE，Ansible 以API调用的方式运行。

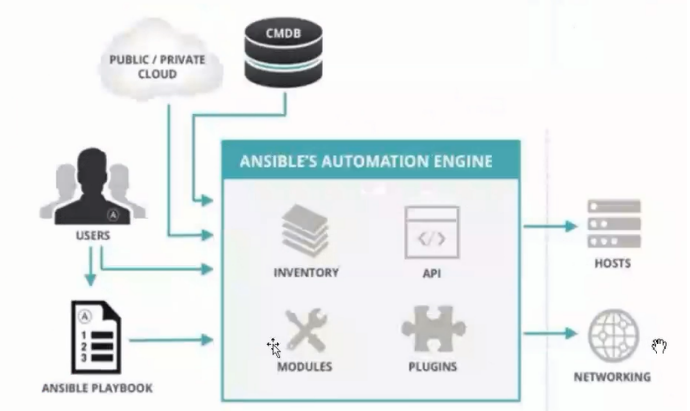
3) Ad-Hoc 命令集，

Users 直接通过Ad-Hoc命令集调用Ansible工具来完成工作

无法实现系列操作

4) Playbooks: 剧本

Users 预先编写好Ansible Playbooks,通过执行Playbooks中预先编排好的任务集按序执行命令



2.Ansible工具集合

Ansible工具集合了Inventory，Moudles, Plugins 和API。

其中，

Inventory

用来管理设备列表，可以通过**分组(不同的业务)**实现，对组的调用直接影响组内所有主机; Moudles

是各种执行模块，几乎所有的管理任务都是通过模块来执行的;

Plugins 提供了各种附加功能的插件;

API

为编程人员提供一个调用接口，可以做Ansible的二次开发具体表现如下。

1）Ansible Playbook：剧本-任务脚本，编排定义Ansible任务集的配置文件，
由Ansible按序依次执行，通常是JSON格式的YML/YAML文件；

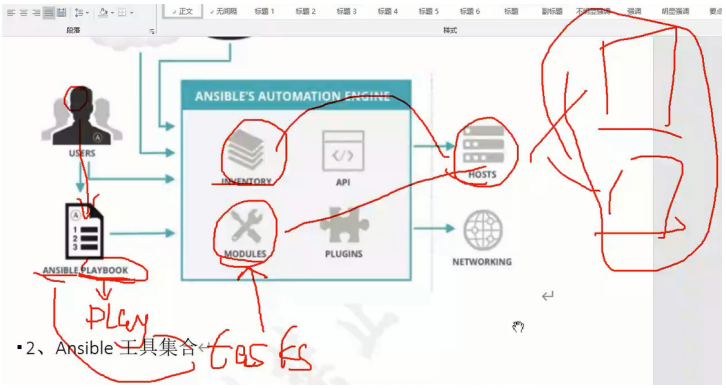
2）Inventory：Ansible 管理主机清单；

3）Moudle：Ansible 执行命令工能模块，多数为内置的核心模块也可以用户自定义；

4）Plugins：模块功能的补充，如连接类型插件，循环插件，变量插件，过滤插件等，该功能不常用；

5）API：提供第三方程序调用的应用程序编程接口；

6）Ansible：该部分图中表示的，组合Inventory，API， Moudles，Plugins 可以理解为Ansible



3.作用对象

Ansible的作用对象不仅仅是Linux 和非Linux 操作系统的主机，
也可以作用与各类PUBLIC/PRIVATE（云平台），商业和非商业设备的网络设施。
使用者使用Ansible或Ansible-playbooks时，
在服务器终端输入Ansible的Ad-Hoc命令集或Playbooks后，
Ansible 会遵循预先定义安排的规则将Playbooks 逐步拆解为Play，
再将Play组织成Ansible可以识别的任务，
随后调用任务涉及的所有模板和插件，
根据Inventory中定义的主机列表
通过SSH将任务集以临时文件或命令的形式传输到远程客户端
执行并返回执行结果，
如果是临时文件则执行完毕后自动删除。

二、Ansbile的搭建

接下来我们来学习Ansible 的安装和部署。Ansible 的安装部署非常简单，
以RPM安装为例，配置好阿里云的yum源直接安装就可以了，
Ansible的管理端只能是Linux,如Redhat,Debian，CentOS，
下 面介绍在CentOS7.x上安装部署Ansible

1.实验环境

操作系统	IP地址	角色	主机名	CPU核心
CentOS7.x	192.168.200.107	Ansible	Ansible	1
CentOS7.x	192.168.200.108	Client	Client1	1
CentOS7.x	192.168.200.109	Client	Client2	2

2.环境准备

```
[root@localhost ~]# iptables -F
[root@localhost ~]# systemctl stop firewalld
[root@localhost ~]# setenforce 0
setenforce: SELinux is disabled
```

远程连接出了问题：

排错：之前做实验时留下的小尾巴

```
[root@localhost ~]# vim ~/.ssh/known_hosts        # 删了里面的记录
```

```
[root@localhost ~]# scp /etc/hosts 192.168.200.108:/etc/
[root@localhost ~]# scp /etc/hosts 192.168.200.109:/etc/
```

3.YUM方式安装ansible (推荐)

而且容易获得最新版的

```
[root@localhost yum.repos.d]# ls
backup  CentOS-Base.repo  epel-release-latest-7.noarch.rpm  epel.repo  epel-testing.repo  httpd-2.4.37.tar.gz  local.repo
[root@localhost ~]# yum -y install ansible
```

ansible底层是由Python开发的

安装方法二：

pip 方式安装 ansible

安装支持包

```
[root@ansible ~]# yum -y install lrzsz vim net-tools gcc gcc-c++ ncurses ncurses-devel unzip zlib-devel zlib openssl-devel openssl
```

源码编译 Python3.5

```
[root@ansible Python-3.5.2]# tar xf Python-3.5.2.tgz -C /usr/src/
[root@ansible Python-3.5.2]# cd /usr/src/Python-3.5.2/
[root@ansible Python-3.5.2]# ./configure --prefix=/usr/local/python/ && make && make install
[root@ansible Python-3.5.2]# ln -s /usr/local/python/bin/python3 /usr/bin/python3
```

```
[root@ansible Python-3.5.2]# which python3
/usr/bin/python3
[root@ansible Python-3.5.2]# python3 -V
Python 3.5.2
```

使用 pip3 安装 ansible

```
[root@ansible Python-3.5.2]# /usr/local/python/bin/pip3 install ansible
```

等待 ansible 安装完毕后，注意基于 pip 安装的没有配置文件，需要自己建立

```
[root@ansible python]# ln -s /usr/local/python/bin/ansible /usr/local/bin/
[root@ansible python]# which ansible
/usr/local/bin/ansible
[root@ansible python]# ansible --version
ansible 2.5.4
  config file = None
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/python/lib/python3.5/site-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 3.5.2 (default, Jun 13 2018, 09:13:32) [GCC 4.8.5 20150623 (Red Hat 4.8.5-11)]
```

4.创建SSH免交互登录

Ansible是通过SSH协议对设备进行管理，而SSH服务包含两种认证方式，

一种是通过密码认证，另一种是通过秘钥对认证

密码方式必须和系统进行交互，而秘钥对是免交互登录

如果希望通过Ansible 自动管理设备应该配置为免交互登录被管理设备

192.168.200.107：

```
[root@localhost ~]# ssh-keygen -t rsa
[root@localhost ~]# ssh-copy-id root@192.168.200.108
[root@localhost ~]# ssh-copy-id root@192.168.200.109
[root@localhost ~]# vim /etc/ansible/hosts
```

末尾添加：

```
[sofia]
192.168.200.108
192.168.200.109
```

到这ansible环境就搭好了

```
[root@localhost ~]# ansible -h
```

查看帮助

192.168.200.107：

```
[root@localhost ~]# ansible sofia -a "df -Th"
    192.168.200.108 | CHANGED | rc=0 >>
    文件系统                类型      容量  已用  可用 已用% 挂载点
    /dev/mapper/centos-root xfs        50G  4.4G   46G    9% /
    devtmpfs                devtmpfs  471M     0  471M    0% /dev
    tmpfs                   tmpfs     488M     0  488M    0% /dev/shm
    tmpfs                   tmpfs     488M  8.0M  480M    2% /run
    tmpfs                   tmpfs     488M     0  488M    0% /sys/fs/cgroup
    /dev/sr0                iso9660   4.2G  4.2G     0  100% /media/cdrom
    /dev/mapper/centos-home xfs        46G   33M   46G    1% /home
    /dev/sda1               xfs      1014M  157M  858M   16% /boot
    tmpfs                   tmpfs      98M     0   98M    0% /run/user/0
    192.168.200.109 | CHANGED | rc=0 >>
    文件系统                类型      容量  已用  可用 已用% 挂载点
    /dev/mapper/centos-root xfs        50G  4.4G   46G    9% /
    devtmpfs                devtmpfs  471M     0  471M    0% /dev
    tmpfs                   tmpfs     488M     0  488M    0% /dev/shm
    tmpfs                   tmpfs     488M  8.0M  480M    2% /run
    tmpfs                   tmpfs     488M     0  488M    0% /sys/fs/cgroup
    /dev/mapper/centos-home xfs        46G   33M   46G    1% /home
    /dev/sda1               xfs      1014M  157M  858M   16% /boot
    /dev/sr0                iso9660   4.2G  4.2G     0  100% /media/cdrom
    tmpfs                   tmpfs      98M     0   98M    0% /run/user/0
```

5.Client端环境准备

Client1:

```
[root@localhost ~]# iptables -F
```

```
[root@localhost ~]# systemctl stop firewalld
[root@localhost ~]# setenforce 0
setenforce: SELinux is disabled
```

server发送公钥给client ---手工的

```
[root@ansible ~]# ssh-copy-id root@192.168.200.108
[root@ansible ~]# ssh-copy-id root@192.168.200.109
```

基于sshpass分发密钥

```
[root@localhost ~]# rpm -q sshpass
sshpass-1.06-2.el7.x86_64
```

sshpass命令使用方法:

1、直接远程连接某主机

```
sshpass -p {密码} ssh {用户名}@{主机IP}
```

2、远程连接指定ssh的端口

```
sshpass -p {密码} ssh -p ${端口} {用户名}@{主机IP}
```

3、从密码文件读取文件内容作为密码去远程连接主机

```
sshpass -f ${密码文本文件} ssh {用户名}@{主机IP}
```

4、从远程主机上拉取文件到本地

```
sshpass -p {密码} scp {用户名}@{主机IP}:${远程主机目录} ${本地主机目录}
```

5、将主机目录文件拷贝至远程主机目录

```
sshpass -p {密码} scp ${本地主机目录} {用户名}@{主机IP}:${远程主机目录}
```

6、远程连接主机并执行命令 µ

```
sshpass -p {密码} ssh -o StrictHostKeyChecking=no {用户名}@{主机IP} 'rm -rf /tmp/test'<
-o StrictHostKeyChecking=no :忽略密码提示
```

```
[root@client1 ~]# rm -rf  .ssh/authorized_keys
[root@localhost ~]#  sshpass -p123456 ssh root@192.168.200.108

[root@client2 ~]# rm -rf .ssh/authorized_keys

[root@localhost ~]#  sshpass -p123456 ssh-copy-id root@192.168.200.108
[root@localhost ~]#  sshpass -p123456 ssh-copy-id root@192.168.200.109
```

这里可以写一个小循环的脚本就可以

```
[root@localhost ~]# ssh root@192.168.200.108

[root@localhost ~]# rm -rf .ssh/know_hosts
```

6、远程连接主机并执行命令

```
sshpass -p {密码} ssh -o StrictHostKeyChecking=no {用户名}@{主机IP} 'rm -rf /tmp/test'
-o StrictHostKeyChecking=no       # 忽略密码提示
```

报错：

```
[root@localhost ~]#  sshpass -p123456 ssh-copy-id root@192.168.200.108
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
```

原因：

没人帮它填交互时的yes

解决办法：

```
[root@localhost ~]#  sshpass -p123456 ssh-copy-id root@192.168.200.108 -o StrictHostKeyChecking=no
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to
install the new keys

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh -o 'StrictHostKeyChecking=no' 'root@192.168.200.108'"
and check to make sure that only the key(s) you wanted were added.
```

除了上面那种加-o StrictHostKeyChecking=no的方式外，还可以用下面的方式：

客户端：

```
[root@localhost ~]# vim /etc/ssh/ssh_config
35 StrictHostKeyChecking no
[root@localhost ~]#  sshpass -p123456 ssh-copy-id root@192.168.200.109
```

ps:

删除密钥的方式：

```
[root@localhost ~]# rm -rf .ssh/know_hosts
[root@client1 ~]# rm -rf  .ssh/authorized_keys
[root@client2 ~]# rm -rf .ssh/authorized_keys
```

三、Ansbile的配置

Inventory是Ansible管理主机信息的配置文件，相当于系统的Hosts文件功能，
默认存放在/etc/ansible/hosts。（ps:这个hosts文件与系统的/etc/hosts文件没有关系）
在hosts 文件中通过分组来组织设备，
Ansible 通过Inventory来定义主机和分组，
通过ansible命令中是用选项-i或者--inventory-file 指定inventory。

1.定义多个组:

```
[root@ansible ~]# vim /etc/ansible/hosts
[sofia]
192.168.200.108
192.168.200.109
[web]
192.168.200.108
[root@ansible ~]# ansible -i /etc/ansible/hosts sofia -m ping
192.168.200.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
```

```
    "changed": false,
    "ping": "pong"
}
192.168.200.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
[root@ansible ~]# ansible -i /etc/ansible/hosts web -m ping
192.168.200.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

注意：如果 /etc/ansible/hosts 文件位置没有改变过的话，可以直接执行，如下

```
[root@ansible ~]# ansible web -m ping
192.168.200.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

2.ansible命令格式：

ansible -i 指定hosts文件位置　组名　-m 指定模块　-a 模块参数

来一波通俗点的理解：

模块：

工具箱（螺丝刀，钳子。。。）

是某一类的管理方式

比如：磁盘管理是一个模块（分区、格式化。。等细节功能）

模块参数：

给某一个工具传一个参数；就是说，工具箱里面是一个锤子，得给它一个钉子，要不然它钉什么呀

---

ansible中还有一种配置方式

不传密钥对也能做，适合于大量主机的时候使用

```
[root@ansible ~]# vim /etc/ansible/hosts
[nokey]
client1 ansible_ssh_host=192.168.200.108 ansible_ssh_port=22 ansible_ssh_user=root
ansible_ssh_pass=123456
```

```
client2 ansible_ssh_host=192.168.200.109 ansible_ssh_port=22 ansible_ssh_user=root
ansible_ssh_pass=123456
```

测试：

```
[root@ansible ~]# ansible nokey -m ping
client1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
client2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

这种方法适用于初期；（这里的初期是指？公司建立初期，还是我刚到公司初期？）

比如公司有很多台机器，我想对它们进行批量管理，因为发密钥对有交互（除非用sshpass），所以可以用这个方法指定，但是注意了，/etc/ansible/hosts 这个文件保管好了，

要不然先通过这个方法先能通信，然后再把密钥发过去，因为直接在这个配置文件中出现密码的话，安全性是个问题。

```
[root@ansible ~]# vim /etc/ansible/hosts
# 将含有密码的组删除
[root@ansible ~]# sshpass -p123456 ssh-copy-id root@192.168.200.108
# 发送密钥
```

以上都是交互式验证的解决方案

Ansible通过将设备列表以分组的方式添加到/etc/ansible/hosts文件来实现对设备的管理，

所以在正式管理之前，

首先要编写hosts文件，hosts 文件中以“[]”包含的部分代表组名，

设备列表支持主机名和ip地址，

默认情况下通过22号端口进行访问(SSH)来管理设备。

若目标主机使用了非默认的SSH端口，

还可以在主机名称后使用冒号夹端口号标明，

以行为单位分隔配置，另外，hosts 文件还支持通配符。同一台主机也可以归属不同的组。

解决known_hosts 的问题，只需要修改ssh的配置文件/etc/ssh/ssh_config 即可

修改ssh配置文件并重启ssh服务

```
[root@ansible ~]# ls .ssh/
authorized_keys  id_rsa          id_rsa.pub       known_hosts
```

配置完成后可以针对hosts定义服务组件进行远程操作，

也可以针对组中的指定的某一个或多个主机操作，

下面介绍

3.如何针对特定的服务器操作

针对crushlinux组中的192.168.200.108主机操作，通--limit 参数限定主机的变更

```
[root@ansible ~]# ansible sofia -m command -a "head -5 /etc/passwd"
192.168.200.109 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
192.168.200.108 | CHANGED | rc=0 >>
```

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[root@ansible ~]# ansible sofia -m command -a "head -5 /etc/passwd" --limit "192.168.200.109"
192.168.200.109 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[root@ansible ~]# ansible 192.168.200.108 -m command -a "head -5 /etc/passwd"
192.168.200.108 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

200网段的主机来执行，注意是/etc/ansible/hosts 文件里面出现过的200网段的主机

```
[root@ansible ~]# ansible 192.168.200.* -m command -a "head -5 /etc/passwd"
192.168.200.109 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
192.168.200.108 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

ansible中默认使用的就是command模块，可以省略不敲，如下

```
[root@ansible ~]# ansible sofia -a "head -5 /etc/passwd"
192.168.200.109 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
192.168.200.108 | CHANGED | rc=0 >>
root:x:0:0:root:/root:/bin/bash
```

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

四、Ansbile命令

Ansible的维护命令大多以ansible开头，在终端输入ansible后连续按两次tab键

会补全所有以ansible字母开头的命令，下 面介绍Ansible的一些常用命令

[root@ansible ~]# ansible

ansible ansible-console 交互式页面 ansible-doc-2.7 ansible-playbook

ansible-pull-2.7

ansible-2 ansible-console-2 ansible-galaxy ansible-playbook-2 ansible-test

ansible-2.7 ansible-console-2.7 ansible-galaxy-2 ansible-playbook-2.7 ansible-

vault

ansible-config ansible-doc 查看帮助 ansible-galaxy-2.7 ansible-pull

ansible-vault-2

ansible-connection ansible-doc-2 ansible-inventory ansible-pull-2 ansible-

vault-2.7

1.ansible

Ansible是生产环境中使用非常频繁的命令之一，主要在以下场景应用

a) 非固化需求

b) 临时一次性操作

c) 二次开发接口调用

因为它是一条一条的命令，不能完成太复杂的东西

非固化需求是指临时性的维护，如查看crushlinux服务器组的磁盘使用情况，

复制一个文件到其他机器等，类似这些没有规律的，临时需要做的任务，我们称为非固化需求，临时一次性操作。

1.语法: ansible <host- pattern> [options]

可用选项如下

➢-v (-verbose):输出详细的执行过程信息，可以得到执行过程所有信息

➢-i PATH(--inventory=PATH):指定inventory信息，默认为/etc/ansible/hosts

➢-f NUM(--forks=NUM):并发线程数，默认为5个线程

➢--private-key=PRIVATE_ KEY_ _FILE 指定秘钥文件

➢-m NAME,-moudle-name=NAME:指定执行时使用的模块

➢-M DIRECTORY (--module path=DIRECTORY):指定模块存放路径默为/usr/share/ansible,

也可以通过ANSIBLE_ LIBRARY 设定默认目录

➢-a ARGUMENTS (--args=ARGUMENTS) :指定模块参数

➢-u USERNAME (--user=USERNAME);指定远程主机USERNAME运行命令

➢-l subset (-limit=SUBSET):限定运行主机

➢-list-hosts:列出符合条件的主机列表，不执行任何命令

[root@ansible ~]# ansible sofia --list-hosts

hosts (2):

192.168.200.108

```
    192.168.200.109
[root@ansible ~]# vim /etc/ansible/hosts
[client]
client1
client2
[root@ansible ~]# ansible all -m ping            # 会执行4遍
192.168.200.108 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
client1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.200.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
client2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

主机和主机组注意事项

主机组范围	解释
all	代表所有主机
webA：webB	可以指定多条主机
主机组范围	解释
all:\!webA	指定all但不包含webB，注意！前需要加转义符号\

```
[root@ansible ~]# ansible client1:192.168.200.109 -m ping
```

```
192.168.200.109 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
client1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
[root@ansible ~]# ansible client:\!client2 -m ping
client1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

2.实验案例

1.检查所有主机是否存活

```
[root@ansible ~]# ansible all -f 4 -m ping
```

-f NUM(--forks=NUM):并发线程数，默认为5个线程

起到加速的作用

其中，192.168.200.112和192.168.200.113（包括client1 和client2）是执行的主机，

SUCCESS表示命令执行成功，

"=>{}""表示返回的结果，

"changed": false,表示没有对主机做出更改，

"ping": "pong"表示执行ping 命令返回的结果。

命令中all关键字是系统默认存在的，代表/etc/ansible/hosts中所有主机，

不需要再hosts中定义关键字all

--list选项列出crushlinux组中所有主机列表信息，

sofia 组中包括192.168.200.108和192.168.200.109两台主机

```
[root@ansible ~]# ansible sofia --list-hosts
  hosts (2):
    192.168.200.108
    192.168.200.109
```

2.批量显示crushlinux组中的磁盘使用情况

[root@ansible ~]# ansible sofia -m command -a "df -Th"

对于-m command 这个模块天生有缺陷，不支持管道和重定向，了解即可

crushlinux需要提前在/etc/ansible/hosts文件中定义组

Ansible的返回结果非常友好，

一般会用三种颜色来表示执行结果:红色，绿色和橘黄色，

其中红色表示执行过程中有异常，

橘黄色表示命令执行后且标有状态变化，

绿色表示执行成功且没有对目标机器做修改

2.Ansible-doc

ansible-doc用来查询ansible模块文档的说明，类似于man命令，针对每个模块都有详细的用法说明及应用案列介绍

[root@ansible ~]# ansible-doc -l   # 查看总帮助

[root@ansible ~]# ansible-doc -s shell   # 查看shell模块的帮助（不加s也行）

[root@ansible ~]# ansible-doc -s raw   # 效率更高一点

前两个借助于Python，而raw是直接用shell执行的，更加原生

查看ping模块的说明信息

[root@ansible ~]# ansible-doc ping

3.ansible- playbook

Ansible-playbpok是日常应用中使用频率最高的命令，

类似于Linux系统中的sh或source命令，用来执行系列任务，

其工作机制是，通过读取编写好的playbook 文件实现集中处理任务

ansible-playbook 命令后跟yml格式的playbook文件，

playbook 文件存放了要执行的任务代码，命令使用方式如下

语法: ansible-playbook playbook.yml

playbook.yml需要之前编译好建议写playbook.yml文件的绝对路径

4.ansible-console --了解即可

ansible-console是ansible 为用户提供的一款交互式工具，类似于Windows中的cmd以及Linux中的shell,用户可以在ansible-console虚拟出来的终端上做像shell一样使用ansible内置的各种命令，这为习惯于使用shell 交互方式的用户提供了良好的体验，在终端输入ansible-console命令后显示如下

```
[root@ansible ~]# ansible-console
Welcome to the ansible console.
Type help or ? to list commands.
root@all (4)[f:5]$ cd sofia
root@sofia (2)[f:5]$ list
192.168.200.108
192.168.200.109
root@sofia (2)[f:5]$
```

五、Ansible模块 --为playbook服务的

1.command模块

Command模块在远程主机执行命令，不支持管道，重定向等Shell 的特性，

常用参数如下(不支持管道，不建议使用)

1.chdir:在远程主机上运行命令前要提前进入的目录

2.creates在命令运行时创建一个文件，如果文件已经存在，则不会创建任务

3. removes: 在命令运行时移除一一个文件，如果文件不存在，则不会执行移除任务

4. executeble: 指明运行命令的shell程序

实验案例

1.在所有主机上运行“Is./”命令，运行前切换到/home目录。操作如下。

准备一下环境:在两台主机上分别创建crushlinux用户，否则/home下是空的查看不到效果

```
[root@ansible ~]# ansible sofia -a "chdir=/tmp ls"
192.168.200.109 | CHANGED | rc=0 >>
ansible_command_payload_PDZ16M
systemd-private-ca471cee72394d609d70c01cab6c1223-cups.service-2tmz7N
vmware-root
192.168.200.108 | CHANGED | rc=0 >>
ansible_command_payload_h7YNwu
systemd-private-b9e1c66f2f58499a9da1b2ec474f8d67-cups.service-UzXBZn
[root@ansible ~]# ansible all -m command -a "chdir=/home ls ./"
192.168.200.109 | CHANGED | rc=0 >>
sofia
client2 | CHANGED | rc=0 >>
sofia
client1 | CHANGED | rc=0 >>
sofia
192.168.200.108 | CHANGED | rc=0 >>
sofia
[root@ansible ~]# ansible sofia -a 'uptime'
192.168.200.108 | CHANGED | rc=0 >>
 08:37:16 up  6:05,  2 users,  load average: 0.00, 0.01, 0.05
192.168.200.109 | CHANGED | rc=0 >>
 00:37:18 up  6:13,  2 users,  load average: 0.00, 0.01, 0.05
```

2.shell模块

Shell模块在远程主机执行命令，相当于调用远程主机的shell进程，

然后在该shell下打开一个子shell运行命令，

和command模块的区别是它支持shell特性，

如管道，重定向等

实验案例

1.重定向

```
[root@ansible ~]# ansible sofia -m shell -a 'echo "sofia" > /tmp/test.txt'
192.168.200.108 | CHANGED | rc=0 >>
192.168.200.109 | CHANGED | rc=0 >>
```

2.测试管道符

过滤client端mac地址　　严格遵守文档格式要不会报错

```
[root@ansible ~]# ansible sofia -m shell -a 'ifconfig ens32 | awk "/ether/{print $2}"'
192.168.200.109 | CHANGED | rc=0 >>
        ether 00:0c:29:af:e0:c4  txqueuelen 1000  (Ethernet)
192.168.200.108 | CHANGED | rc=0 >>
        ether 00:0c:29:cd:9b:a8  txqueuelen 1000  (Ethernet)
```

3.raw模块

最原始的方式运行命令（不依赖python，仅通过ssh实现）

清除yum缓存

```
[root@ansible ~]# ansible sofia -m raw -a "yum clean  all"
192.168.200.108 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
正在清理软件源： sofia
Cleaning up everything
Maybe you want: rm -rf /var/cache/yum, to also free up space taken by orphaned data from disabled or removed repos
Cleaning up list of fastest mirrors
Shared connection to 192.168.200.108 closed.

192.168.200.109 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
正在清理软件源： gabe
Cleaning up everything
Maybe you want: rm -rf /var/cache/yum, to also free up space taken by orphaned data from disabled or removed repos
Cleaning up list of fastest mirrors
Shared connection to 192.168.200.109 closed.
```

建立缓存

```
[root@ansible ~]# ansible all -m raw -a "yum makecache"
192.168.200.108 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Determining fastest mirrors
sofia                                                    | 3.6 kB     00:00
(1/4): sofia/group_gz                                      | 166 kB   00:00
(2/4): sofia/primary_db                                    | 3.1 MB   00:00
(3/4): sofia/filelists_db                                  | 3.1 MB   00:01
(4/4): sofia/other_db                                      | 1.3 MB   00:00
元数据缓存已建立
Shared connection to 192.168.200.108 closed.
```

```
192.168.200.109 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Determining fastest mirrors
gabe                                                     | 3.6 kB     00:00
(1/4): gabe/group_gz                                       | 166 kB   00:00
(2/4): gabe/primary_db                                     | 3.1 MB   00:01
(3/4): gabe/filelists_db                                   | 3.1 MB   00:01
(4/4): gabe/other_db                                       | 1.3 MB   00:00
元数据缓存已建立
Shared connection to 192.168.200.109 closed.
```

yum装nmap包

```
[root@ansible ~]# ansible sofia -m raw -a "yum -y install nmap"
192.168.200.109 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
正在解决依赖关系
--> 正在检查事务
---> 软件包 nmap.x86_64.2.6.40-13.el7 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================
 Package         架构              版本                        源              大小
================================================================================
正在安装:
 nmap            x86_64            2:6.40-13.el7               gabe            3.9 M

事务概要
================================================================================
安装  1 软件包

总下载量：3.9 M
安装大小：16 M
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在安装    : 2:nmap-6.40-13.el7.x86_64                                   1/1
  验证中      : 2:nmap-6.40-13.el7.x86_64                                   1/1

已安装:
```

```
  nmap.x86_64 2:6.40-13.el7

完毕!
Shared connection to 192.168.200.109 closed.

192.168.200.108 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
正在解决依赖关系
--> 正在检查事务
---> 软件包 nmap.x86_64.2.6.40-13.el7 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================
 Package         架构              版本                       源              大小
================================================================================
正在安装:
 nmap            x86_64            2:6.40-13.el7              sofia           3.9 M

事务概要
================================================================================
安装  1 软件包

总下载量：3.9 M
安装大小：16 M
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在安装    : 2:nmap-6.40-13.el7.x86_64                                   1/1
  验证中      : 2:nmap-6.40-13.el7.x86_64                                   1/1

已安装:
  nmap.x86_64 2:6.40-13.el7

完毕!
Shared connection to 192.168.200.108 closed.
```

4.copy模块

copy模块用于复制指定主机文件到远程主机的指定位置，常见参数如下。

1. dest: 指出复制文件的目标目录位置，使用绝对路径。如果源是目录，指目标也要是目录，如果目标文件已经存在会覆盖原有内容
2. src: 指出源文件的路径，可以使用相对路径或绝对路径，支持直接指定目录，如果源是目录则目标也要是目录
3. mode: 指出复制时，目标文件的权限可选
4. owner:指出复制时，目标文件的属主可选
5. group: 指出复制时，目标文件的属组可选
6. content: 指出复制到目标主机上的内容，不能与src一起使用，相当于复制content指明的数据到目标文件中

特别提示:

参数: backup=yes===>意思是， 如果目标路径下，有与我同名但不同内容的文件时，在覆盖前，对目标文件先进行备份。

所有被管理端节点必须安装libselinux python包

```
[root@ansible ~]# rpm -q libselinux-python
libselinux-python-2.5-12.el7.x86_64
[root@client1 ~]# rpm -q libselinux-python
libselinux-python-2.5-12.el7.x86_64
[root@client2 ~]# rpm -q libselinux-python
libselinux-python-2.5-12.el7.x86_64
```

实验案例 ---copy

将crushlinux组中主机的/etc/hosts文件拷贝到/tmp下指定权限为777更该属主为crushlinux更改属组为root

```
[root@ansible ~]# vim /etc/hosts
192.168.200.107 ansible
192.168.200.108 client1
192.168.200.109 client2
192.168.200.200 ppp             # 为区别于其他主机而加的
[root@ansible ~]# ansible sofia -m copy -a "src=/etc/hosts dest=/tmp mode=640 owner=root group=root"
192.168.200.108 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "checksum": "e799d852d834ba545681779e4b0f4c32c337c185",
    "dest": "/tmp/hosts",
    "gid": 0,
    "group": "root",
    "md5sum": "03d38662a3d7411c69974048d6320aa9",
    "mode": "0640",
    "owner": "root",
    "size": 250,
    "src": "/root/.ansible/tmp/ansible-tmp-1589101503.33-1564-86213734581056/source",
    "state": "file",
```

```
    "uid": 0
}
192.168.200.109 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "checksum": "e799d852d834ba545681779e4b0f4c32c337c185",
    "dest": "/tmp/hosts",
    "gid": 0,
    "group": "root",
    "md5sum": "03d38662a3d7411c69974048d6320aa9",
    "mode": "0640",
    "owner": "root",
    "size": 250,
    "src": "/root/.ansible/tmp/ansible-tmp-1589101503.35-1566-211785745285971/source",
    "state": "file",
    "uid": 0
}
```


5.hostname模块

hostname模块用于管理远程主机上的主机名，常用参数如下e

1.name ：

指明主机名

实验案例

更改client1 (192.168.200.112)的主机名为crushlinux

```
[root@ansible ~]# ansible 192.168.200.108 -m hostname -a "name=crushlinux"
192.168.200.108 | CHANGED => {
    "ansible_facts": {
        "ansible_domain": "",
        "ansible_fqdn": "crushlinux",
        "ansible_nodename": "crushlinux",
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "name": "crushlinux"
}
```

6.yum模块

Yum模块基于yum机制，对远程主机管理程序包，常用参数如下。

1. name:程序包的名称，可以带上版本号，如不指定版本号默认安装为最新版本
2. state=present latest absent:指明对程序包执行的操作，pressent 表示安装程序包，latest 表示安装最新版本的程序包，absent 表示卸载程序包
3. disablerepo: 在用yum安装时禁用某个仓库的ID
4. enablerepo:在用yum安装时启用某个参考的IDe

5.conf file:yum运行时的配置文件而不是使用默认的配置文件

6. diable_gpg_check=yes|no: 是否启用完整性校验功能

实验案例

注意实验前要在client端配置yun仓库

管理员只是发送yum命令道被管理端，被管理端要存在可用的yum仓库才可以成功安装

client 端yum安装dhcp

```
[root@ansible ~]# ansible sofia -m yum -a "name=dhcp state=present"
192.168.200.108 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "changes": {
        "installed": [
            "dhcp"
        ]
    },
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: fastestmirror, langpacks\nLoading mirror speeds from cached
hostfile\nResolving Dependencies\n--> 
Running transaction check\n---> Package dhcp.x86_64 12:4.2.5-68.el7.centos will be installed\n--> 
Finished Dependency Resolution\n\nDependencies
Resolved\n\n================================================================================\n
Package      Arch           Version                        Repository
Size\n================================================================================\nInstalling:\n
 dhcp         x86_64         12:4.2.5-68.el7.centos         sofia         513 k\n\nTransaction
Summary\n================================================================================\nInstall
1 Package\n\nTotal download size: 513 k\nInstalled size: 1.4 M\nDownloading packages:\nRunning
transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n
Installing : 12:dhcp-4.2.5-68.el7.centos.x86_64                           1/1 \n  Verifying  :
12:dhcp-4.2.5-68.el7.centos.x86_64                          1/1 \n\nInstalled:\n  dhcp.x86_64
12:4.2.5-68.el7.centos                                         \n\nComplete!\n"    ]
}
192.168.200.109 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "changes": {
        "installed": [
```

```
            "dhcp"
        ]
    },
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: fastestmirror, langpacks\nLoading mirror speeds from cached
hostfile\nResolving Dependencies\n-->
Running transaction check\n---> Package dhcp.x86_64 12:4.2.5-68.el7.centos will be installed\n-->
Finished Dependency Resolution\n\nDependencies
Resolved\n\n================================================================================\n
Package      Arch           Version                        Repository
Size\n================================================================================\nInstalling:\n
 dhcp         x86_64         12:4.2.5-68.el7.centos         gabe         513 k\n\nTransaction
Summary\n================================================================================\nInstall
1 Package\n\nTotal download size: 513 k\nInstalled size: 1.4 M\nDownloading packages:\nRunning
transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n
Installing : 12:dhcp-4.2.5-68.el7.centos.x86_64                              1/1 \n  Verifying  :
12:dhcp-4.2.5-68.el7.centos.x86_64                            1/1 \n\nInstalled:\n  dhcp.x86_64
12:4.2.5-68.el7.centos                                          \n\nComplete!\n"    ]
}
```


```
[root@ansible ~]# ansible sofia -a "yum -y install bind"
[WARNING]: Consider using the yum module rather than running 'yum'.  If you need to use command
because yum isinsufficient you can add 'warn: false' to this command task or set
'command_warnings=False' in ansible.cfg to get rid ofthis message.
192.168.200.108 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
正在解决依赖关系
--> 正在检查事务
---> 软件包 bind.x86_64.32.9.9.4-61.el7 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================
 Package        架构             版本                          源              大小
================================================================================
正在安装:
 bind           x86_64           32:9.9.4-61.el7               sofia           1.8 M

事务概要
================================================================================
```

```
安装  1 软件包

总下载量：1.8 M
安装大小：4.3 M
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  正在安装     : 32:bind-9.9.4-61.el7.x86_64                                 1/1
  验证中       : 32:bind-9.9.4-61.el7.x86_64                                 1/1

已安装:
  bind.x86_64 32:9.9.4-61.el7

完毕!
192.168.200.109 | CHANGED | rc=0 >>
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
正在解决依赖关系
--> 正在检查事务
---> 软件包 bind.x86_64.32.9.9.4-61.el7 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================
 Package         架构              版本                       源              大小
================================================================================
正在安装:
 bind            x86_64            32:9.9.4-61.el7            gabe            1.8 M

事务概要
================================================================================
安装  1 软件包

总下载量：1.8 M
安装大小：4.3 M
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
```

```
  正在安装     : 32:bind-9.9.4-61.el7.x86_64                                 1/1
  验证中       : 32:bind-9.9.4-61.el7.x86_64                                 1/1

已安装:
  bind.x86_64 32:9.9.4-61.el7

完毕!
```

7.service模块

Service模块为用来管理远程主机上的服务的模块，常见参数如下

1. name:被管理的服务名称
2. state=started |stopped lrestarted:动作包含启动关闭或重启
3. enabled=yes I no:表示是否设置该服务开机自启动
4. runlevel: 如果设定了enabled开机自启动，则要定义在哪些运行目标下自启动

实验案例

启动httpd服务并设置为开机自启动

```
    Client准备操作          //如果没有需要进行安装
[root@ansible ~]# ansible sofia -a "rpm -q httpd"
192.168.200.108 | CHANGED | rc=0 >>
httpd-2.4.6-80.el7.centos.x86_64
192.168.200.109 | CHANGED | rc=0 >>
httpd-2.4.6-80.el7.centos.x86_64
[root@ansible ~]# ansible sofia -m service -a "name=httpd state=started"
```

8.User 模块

User模块用于管理远程主机上的用户账户，常见参数如下。

1. name: 必选参数账号名称
2. state=present labsent:创建账号或者删除账号，present 表示创建，absent 表示删除
3. system=yesIno: 是否为系统账号
4. uid: 用户UID
5. group: 用户的基本组
6. groups: 用户的附加组
7. shell: 默认使用的shell
8. home:用户的家目录
9. move_home=yes | no:如果设置的家目录已经存在，是否将已经存在的家目录进行移动
10. password: 用户的密码，建议使用加密后的字符串
11. comment:用户的注释信息
12. remove=yes|no: 当state=absent时，是否删除用户的家目录

实验案例

1.创建用户

```
[root@ansible ~]#  ansible sofia -m user -a 'name=user1 system=yes uid=502 group=root
groups=sshd shell=/sbin/nologin home=/home/user1 password=user1 comment="test user"'
[WARNING]: The input password appears not to have been hashed. The 'password' argument must be
encrypted for this module to work properly.
192.168.200.108 | CHANGED => {
```

```
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "comment": "test user",
    "create_home": true,
    "group": 0,
    "groups": "sshd",
    "home": "/home/********",
    "name": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
    "password": "NOT_LOGGING_PASSWORD",
    "shell": "/sbin/nologin",
    "state": "present",
    "system": true,
    "uid": 502
}
192.168.200.109 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "comment": "test user",
    "create_home": true,
    "group": 0,
    "groups": "sshd",
    "home": "/home/********",
    "name": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
    "password": "NOT_LOGGING_PASSWORD",
    "shell": "/sbin/nologin",
    "state": "present",
    "system": true,
    "uid": 502
}
```


2.删除用户及家目录

```
[root@ansible ~]# ansible sofia -m user -a 'name=user1 state=absent remove=yes'
```


```
192.168.200.108 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "force": false,
    "name": "user1",
    "remove": true,
```

```
    "state": "absent",
    "stderr": "userdel: user1 邮件池 (/var/spool/mail/user1) 未找到\n",
    "stderr_lines": [
        "userdel: user1 邮件池 (/var/spool/mail/user1) 未找到"
    ]
}
192.168.200.109 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "force": false,
    "name": "user1",
    "remove": true,
    "state": "absent",
    "stderr": "userdel: user1 邮件池 (/var/spool/mail/user1) 未找到\n",
    "stderr_lines": [
        "userdel: user1 邮件池 (/var/spool/mail/user1) 未找到"
    ]
}
```

9.script模块

script模块能够实现远程服务器批量运行本地的shell脚本。

远程批量分发并自动部署nginx

所有被管理端需要挂载光盘，并创建本地yum仓库文件

```
[root@ansible ~]# cd /opt/
[root@ansible opt]# rz
[root@ansible opt]# ls
nginx-1.16.0.tar.gz  rh
[root@ansible opt]# vim auto_nginx.sh       # 自动安装nginx脚本
#!/bin/bash
#nginx install shell script
yum -y install gcc gcc-c++ make pcre pcre-devel zlib zlib-devel openssl openssl-devel &>/dev/null
cd /opt/
tar xf nginx-1.16.0.tar.gz -C /usr/src/
cd /usr/src/nginx-1.16.0/
./configure --prefix=/usr/local/nginx --with-http_ssl_module --with-http_stub_status_module
&>/dev/null
make &>/dev/null
make install &>/dev/null
/usr/local/nginx/sbin/nginx
exit 0

[root@ansible opt]# vim fabu.sh
```

```
#!/bin/bash
#批量分布脚本
Group=$1
ansible $Group -m copy -a "src=/opt/ dest=/opt/"
ansible $Group -m script -a "/opt/auto_nginx.sh"
[root@ansible opt]# chmod +x /opt/auto_nginx.sh /opt/fabu.sh
[root@ansible opt]# bash /opt/fabu.sh sofia
[root@ansible opt]# bash /opt/fabu.sh sofia
192.168.200.108 | CHANGED => {
    "changed": true,
    "dest": "/opt/",
    "src": "/opt/"
}
192.168.200.109 | CHANGED => {
    "changed": true,
    "dest": "/opt/",
    "src": "/opt/"
}
192.168.200.108 | CHANGED => {
    "changed": true,
    "rc": 0,
    "stderr": "Shared connection to 192.168.200.108 closed.\r\n",
    "stderr_lines": [
        "Shared connection to 192.168.200.108 closed."
    ],
    "stdout": "",
    "stdout_lines": []
}
```

六、playbook配置文件

1.执行配置文件

Playbook配置文件使用YAML语法，具有简介明了，结构清晰等特点。

Playbook 配置文件类似于shell 脚本，是一个YAML格式的文件，用于保存针对特定需求的任务列表，

前面介绍的ansible命令虽然可以完成各种任务，

但是当配置一系列任务时，逐条输入命令就显得效率非常低下

更有效的方式在playbook 配置中配置所有的任务代码

利用ansible-playbook命令执行该文件，可以实现自动化运维，

YAML 文件的扩展名通常为.yaml或.yml

YAML语法和其他高级语言类似，其结构通过缩进来展示

通过“-”来代表选项

通过冒号“:”来分隔键和值

整个文件以“---”开始并以“...”结束

如下所示

修改hosts文件

```
[root@ansible ~]# vim /etc/ansible/hosts
[sofia]
192.168.200.108
192.168.200.109

[test01]
192.168.200.108

[test02]
192.168.200.109
```

```
[root@ansible ~]# vim /etc/ansible/test.yml
# 创建test.yml文件，文件位置可以随便放，建议放在这个目录下，方便查找
---                                          # 开头格式(可忽略)
- hosts: test01                            # 表示对test01 (192.168.200.108) 的操作
  remote_user: root                       #远端执行用户身份root
  tasks:                                     # 任务列表
    - name: add user                      # 任务名称
      user: name=user2 state=present          # 执行user模块创建用户
      tags:                                       # 创建tag标签
      - testaaa                                  # tag标签为testaaa
    - name: add group                        #  任务名称
      group: name=root system=yes       # 执行group模块创建组
      tags:                                     # 创建tag标签
      -testbbb                                  # tag标签为testbbb
- hosts: test02                                 # 表示对test02 (192.168.200.109) 的操作
  remote_user: root                          #远端执行用户身份root
  tasks:                                    # 任务列表
    - name: copy file                          # 任务名称
      copy: src=/etc/passwd dest=/tmp                    # 执行copy模块复制文件
      tags:                                  # 创建tag标签
      - testccc                             # tag标签为testccc
...                             # 结尾格式(可忽略)
```

所有的“-”和“:”后面均有空格，而且要注意缩进和对齐

Playbook的核心元素包含:

1. hosts: 任务的目标主机，多个主机用冒号分隔，一般调用/etc/ansible/hosts中的分组信息
2. remote user:远程主机上，运行此任务的什么默认为roote
3. tasks: 任务，即定义的具体任务，由模块定义的操作列表
4. handlers: 触发器，类似tasks，只是在特定的条件下才会触发任务。

某任务的状态在运行后为changed时，可通过“notify”通知给相应的handlers进行触发执行。

5. roles: 角色，将hosts剥离出去，由tasks, handlers 等所组成的一种特定的结构集合。

用法

Playbook文件定义的任务需要通过ansible-playbook 命令进行调用并执行，

ansible-playbook命令用法如下

用法: ansible-playbook [option] /PATH/TO/PLAYBOOK.yaml

其中[option]部分的功能包括

1. --syntax-check:检测yaml文件的语法
2. -C (--check):测试，不会改变主机的任何配置
3. -list-hosts:列出yaml文件影响的主机列表
4. -list-tasks:列出yaml文件的任务列表

--标签的作用主要控制任务是怎么执行的

5. -list-tags:列出yaml文件中的标签
6. -t TAGS (-tags=TAGS):表示只执行指定标签的任务。
7. skip-tags=SKIP_ TAGSS:表示除了指定标签任务，执行其他任务
8. -startat-task START _AT:从指定任务开始往下运行

实验案例

1.语法检查

```
[root@ansible ~]# ansible-playbook --syntax-check /etc/ansible/test.yml
playbook: /etc/ansible/test.yml
```

2.预测试

```
[root@ansible ~]# ansible-playbook -C /etc/ansible/test.yml
PLAY [test01] ******************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.200.108]

TASK [add user] ****************************************************************************************************
changed: [192.168.200.108]

TASK [add group] ***************************************************************************************************
ok: [192.168.200.108]

PLAY [test02] ******************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [192.168.200.109]

TASK [copy file] ***************************************************************************************************
changed: [192.168.200.109]

PLAY RECAP *********************************************************************************************************
192.168.200.108            : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.200.109            : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

真正执行

```
[root@ansible ~]# ansible-playbook /etc/ansible/test.yml
```

测试查看：

```
[root@client1 ~]# id user2
uid=1001(user2) gid=1001(user2) 组=1001(user2)


[root@client2 ~]# ls /tmp
  passwd
```

3.列出主机

```
[root@ansible ~]# ansible-playbook --list-hosts /etc/ansible/test.yml

playbook: /etc/ansible/test.yml

  play #1 (test01): test01      TAGS: []
    pattern: [u'test01']
```

```
    hosts (1):
      192.168.200.108

  play #2 (test02): test02      TAGS: []
    pattern: [u'test02']
    hosts (1):
      192.168.200.109
[root@ansible ~]# ansible-playbook --list-tasks /etc/ansible/test.yml

playbook: /etc/ansible/test.yml

  play #1 (test01): test01      TAGS: []
    tasks:
      add user  TAGS: [testaaa]
      add group TAGS: [testbbb]

  play #2 (test02): test02      TAGS: []
    tasks:
      copy file TAGS: [testccc]
[root@ansible ~]# ansible-playbook --list-tags /etc/ansible/test.yml          # 一般很少用

playbook: /etc/ansible/test.yml

  play #1 (test01): test01      TAGS: []
      TASK TAGS: [testaaa, testbbb]

  play #2 (test02): test02      TAGS: []
      TASK TAGS: [testccc]
```

通常情况下会先执行ansible-playbook -C /PATH/TO/PLAYBOOK.yml命令进行测试，

测试没问题后再执行ansible-palybook /PATH/TO/PLAYBOOK.yml命令

执行输出

我们在用playbook 进行ansible模块操作的时候，并没有命令的执行结果输出，

默认被隐藏了。我们可以通过register模块最加输出命令的执行结果

```
[root@ansible ~]# vim /etc/ansible/test.yml
  7       register: print_result        #将之前命令的输出结果保存在变量print_result里
 10       debug: var=print_result        #将变量的值作为debug输出出来
 [root@ansible ~]# ansible-playbook /etc/ansible/test.yml
```

2.触发器

需要触发才能执行的任务，当之前定义在tasks中的任务执行完成后，

若希望在基础上触发其他的任务，这时就需要定义handlers。

例如，当通过ansible的模块对目标主机的配置文件进行修改之后，

如果任务执行成功，可以触发一个触发器，在触发器中定义目标主机的服务重启操作，以使配置文件生效，

handlers 触发器具有以下优点

1.handlers是Ansible提供的条件机制之一，handlers和task很类似，

但是他在被task通知的时候才会触发执行

2. handlers 只会在所有任务执行完成后执行,而且即使被通知了多次，

它也只会执行一次，handlers按照定义的顺序依次执行

handlers触发器的使用示例如下

```
[root@ansible ~]# ansible test01 -m shell -a 'netstat -anpt | grep 80'
192.168.200.108 | CHANGED | rc=0 >>
tcp6       0      0 :::80                   :::*                    LISTEN      22029/httpd
[root@ansible ~]# vim /etc/ansible/httpd.yml
---
- hosts: sofia
  remote_user: root
  tasks:
    - name: change port
      command: sed -i 's/Listen 80/Listen 8080/g' /etc/httpd/conf/httpd.conf
      notify:
        - rehttpd

  handlers:
    - name: rehttpd
      service: name=httpd state=restarted
...
```

检查语法

```
[root@ansible ~]# ansible-playbook --syntax-check /etc/ansible/httpd.yml
playbook: /etc/ansible/httpd.yml
```

预执行

```
[root@ansible ~]# ansible-playbook -C /etc/ansible/httpd.yml
PLAY [sofia] ***********************************************************************************************************

TASK [Gathering Facts] *************************************************************************************************
ok: [192.168.200.109]
ok: [192.168.200.108]

TASK [change port] *****************************************************************************************************
skipping: [192.168.200.109]
skipping: [192.168.200.108]

PLAY RECAP *************************************************************************************************************
192.168.200.108            : ok=1    changed=0    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
192.168.200.109            : ok=1    changed=0    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

执行：

```
[root@ansible ~]# ansible-playbook /etc/ansible/httpd.yml
PLAY [sofia] ***********************************************************************************************************

TASK [Gathering Facts] *************************************************************************************************
ok: [192.168.200.109]
ok: [192.168.200.108]

TASK [change port] *****************************************************************************************************
[WARNING]: Consider using the replace, lineinfile or template module rather than running 'sed'.  If you need to use
command because replace, lineinfile or template is insufficient you can add 'warn: false' to this command task or set
'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [192.168.200.109]
changed: [192.168.200.108]

RUNNING HANDLER [rehttpd] **********************************************************************************************
changed: [192.168.200.109]
changed: [192.168.200.108]

PLAY RECAP *************************************************************************************************************
192.168.200.108            : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.200.109            : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

查看测试：

```
[root@client1 ~]# netstat -lnpt | grep httpd
tcp6       0      0 :::8080                 :::*                    LISTEN      23042/httpd
[root@client2 ~]# netstat -lnpt | grep httpd
tcp6       0      0 :::8080                 :::*                    LISTEN      17997/httpd
```

注释：

```
---                                                    //固定开头格式
- hosts: sofia                        //指定运行主机为test01组
  remote_user: root                 //指定对端运行用户的身份
  tasks:                           //任务列表
    - name: change port              //定义任务名称
      command: sed -i 's/Listen 80/Listen 8080/g' /etc/httpd/conf/httpd.conf         //模块为
command:使用sed命令替换监听端口为8080
      notify:                      //完成任务后调用restart httpd server触发器
        - rehttpd

  handlers:          //配置触发器
    - name: rehttpd                  //指定触发器名字
      service: name=httpd state=restarted               //指定触发条件为重启httpd服务
...                           //结尾句
```

基于Ansible playbook配置zabbix agent端　　---还没讲