第 3 章 Chapter 3

Ansible 组件介绍

经过前面 2 章的介绍，我们已经熟悉了 Ansible 的一些安装与简单使用。从本章开始我们将全面介绍 Ansible 的各种组件。这些也是我们使用 Ansible 的过程中必须理解的知识点。本章将通过介绍和讲解 Ansible 日常中经常使用的一些组件，使我们能对 Ansible 有一个全面的了解。

3.1 Ansible Inventory

在大规模的配置管理工作中我们需要管理不同业务的不同机器，这些机器的信息都存放在 Ansible 的 Inventory 组件里面。在我们工作中配置部署针对的主机必须先存放在 Inventory 里面，这样才能使用 Ansible 对它进行操作。默认 Ansible 的 Inventory 是一个静态的 INI 格式的文件 /etc/ansible/hosts，当然，还可以通过 ANSIBLE_HOSTS 环境变量指定或者运行 ansible 和 ansible-playbook 的时候用 -i 参数临时设置。

1. 定义主机和主机组

下面我们来看一下如何在默认的 Inventory 文件中定义一些主机和主机组，具体如下：

```
1 172.17.42.101    ansible_ssh_pass='123456'
2 172.17.42.102    ansible_ssh_pass='123456'
```

```
3 [docker]
4 172.17.42.10[1:3]
5 [docker:vars]
6 ansible_ssh_pass='123456'
7 [ansible:children]
8 docker
```

- 第 1 行定义了一个主机是 172.17.42.101，然后使用 Inventory 内置变量定义了 SSH 登录密码。
- 第 2 行定义了一个主机是 172.17.42.102，然后使用 Inventory 内置变量定义了 SSH 登录密码。
- 第 3 行定义了一个组叫 docker。
- 第 4 行定义了 docker 组下面 4 台主机从 172.17.42.101 到 172.17.42.103。
- 第 5 行到第 6 行针对 docker 组使用 Inventory 内置变量定义了 SSH 登录密码。
- 第 7 行到第 8 行定义了一个组叫 ansible，这个组下面包含 docker 组。

ansible_ssh_pass 参数是 Ansible Inventory 内置参数，在本节的最后一小节会进行相关的介绍。Inventory 文件一般用来定义远端主机的认证信息，比如 SSH 登录密码、用户名以及 key 相关信息。当然，Inventory 文件也支持主机或者主机组的便利定义。添加完主机和主机组后我们就可以使用 Ansible 命令针对这些主机进行操作和管理了。下面是分别针对不同的主机和主机组进行 Ansible 的 ping 模块检测，ping 模块是 Ansible 中一个连通性检测的模块。当然，Ansible 还内置大量的其他模块，后续章节我们也会慢慢接触。

```
[root@Master ~]# ansible 172.17.42.101:172.17.42.102 -m ping -o
172.17.42.101 | success >> {"changed": false, "ping": "pong"}

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

[root@Master ~]# ansible docker -m ping -o
172.17.42.101 | success >> {"changed": false, "ping": "pong"}

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

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

[root@Master ~]# ansible ansible -m ping -o
```

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

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

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

2. 多个 Inventory 列表

通过上一小节对 Inventory 的介绍，我们知道 Ansible 默认的 Inventory 文件是一个 INI 的静态文件，其实 Ansible 还支持多个 Inventory 文件，这样我们就可以很方便地管理不同业务或者不同环境中的机器了。如何使用多个 Inventoy 文件呢？

首先需要修改 ansible.cfg 中 hosts 文件的定义，或者使用 ANSIBLE_HOSTS 环境变量定义。这里我们准备一个文件夹，里面将存放多个 Inventory 文件，如以下目录所示：

```
[root@Master ~]# tree inventory/
inventory/
├── docker
└── hosts
```

不同的文件可以存放不同的主机，我们来分别看一下文件的内容：

```
[root@Master ~]# cat inventory/hosts
172.17.42.101   ansible_ssh_pass='123456'
172.17.42.102   ansible_ssh_pass='123456'
[root@Master ~]# cat inventory/docker
[docker]
172.17.42.10[1:3]
[docker:vars]
ansible_ssh_pass='123456'
[ansible:children]
docker
```

最后我们修改了 ansible.cfg 文件中 inventory 的值，这里不再指向一个文件，而是指向一个目录，修改如下：

```
inventory  = /root/inventory/
```

这样我们就可以使用 Ansible 的 list-hosts 参数来进行如下验证：

```
[root@Master ~]# ansible 172.17.42.101:172.17.42.102 --list-hosts
```

```
    172.17.42.101
    172.17.42.102
[root@Master ~]# ansible docker --list-hosts
    172.17.42.101
    172.17.42.102
    172.17.42.103
[root@Master ~]# ansible ansible --list-hosts
    172.17.42.101
    172.17.42.102
    172.17.42.103
```

其实 Ansible 中的多个 Inventory 跟单个文件没什么区别，我们也可以容易定义或者引用多个 Inventory，甚至可以把不同环境的主机或者不同业务的主机放在不同的 Inventory 文件里面，方便日后维护。

3. 动态 Inventory

在实际应用部署中会有大量的主机列表。如果手动维护这些列表将是一个非常繁琐的事情。其实 Ansible 还支持动态的 Inventory，动态 Inventory 就是 Ansible 所有的 Inventory 文件里面的主机列表和变量信息都支持从外部拉取。比如我们可以从 CMDB 系统和 Zabbix 监控系统拉取所有的主机信息，然后使用 Ansible 进行管理。这样一来我们就可以很方便地将 Ansible 与其他运维系统结合起来。关于引用动态 Inventory 的功能配置起来也很简单。我们只需要把 ansible.cfg 文件中 inventory 的定义值改成一个执行脚本即可。这个脚本的内容不受任何编程语言限制，但是这个脚本使用参数时有一定的规范并且对脚本执行的结果也有要求。这个脚本需要支持两个参数：

- --list 或者 -l，这个参数运行后会显示所有的主机以及主机组的信息（JSON 格式）。
- --host 或者 -H，这个参数后面需要指定一个 host，运行结果会返回这台主机的所有信息（包括认证信息、主机变量等），也是 JSON 格式。

下面我们通过一个简单的例子了解动态 Inventory 实现流程。这里编写了一个简单 hosts.py 脚本，代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import argparse
import sys
```

```
import json
def lists():
    r = {}
    h=[ '172.17.42.10' + str(i) for i in range(1,4) ]
    hosts={'hosts': h}
    r['docker'] = hosts
    return json.dumps(r,indent=4)

def hosts(name):
    r = {'ansible_ssh_pass': '123456'}
    cpis=dict(r.items())
    return json.dumps(cpis)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-l', '--list', help='hosts list', action='store_
                        true')
    parser.add_argument('-H', '--host', help='hosts vars')
    args = vars(parser.parse_args())

    if args['list']:
        print lists()
    elif args['host']:
        print hosts(args['host'])
    else:
        parser.print_help()
```

这个脚本定义了两个函数：lists 函数在指定 --list 参数后执行，hosts 函数会在指定 --host 参数后执行。脚本的每个函数都会返回一个 JSON，运行结果如下：

```
[root@Master ~]# python hosts.py  --list
{
    "docker": {
        "hosts": [
            "172.17.42.101",
            "172.17.42.102",
            "172.17.42.103"
        ]
    }
}
[root@Master ~]# python hosts.py  -H 172.17.42.102
{"ansible_ssh_pass": "123456"}
```

这里定义了一个 docker 组且组里定义了 3 台主机，然后定义每台设备的 SSH 密码。ansible_ssh_pass 是 Inventory 内置的参数，下一节会详细解释。最后我们来执行临时指定这个 hosts.py 脚本，没修改 ansible.cfg 文件，运行结果如下：

```
[root@Master ~]# ansible  -i hosts.py 172.17.42.102:172.17.42.103  -m
    ping -o
172.17.42.102 | success >> {"changed": false, "ping": "pong"}

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

[root@Master ~]# ansible  -i hosts.py docker  -m ping -o
172.17.42.101 | success >> {"changed": false, "ping": "pong"}

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

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

本节只是简单地编写了一个 Inventory 脚本，希望读者能理解这个原理，在实际工作中可以根据自己的需求编写相应的脚本。目前官方也有一些关于从 cobbler 和 AWS 获取主机或者主机组相关的 Inventory 脚本，有兴趣的读者可以自己查阅。

4. Inventory 内置参数

这里介绍 Ansible Inventory 内置的一些参数，这些参数在我们实际工作中也会经常使用，我们可以直接在 Inventory 文件中定义它，当然动态的 Inventory 也可以使用它，如表 3-1 所示。

表 3-1　Inventory 内置参数

参数	解释	例子
ansible_ssh_host	定义 host ssh 地址	ansible_ssh_host=192.168.1.117
ansible_ssh_port	定义 hosts ssh 端口	ansible_ssh_port=5000
ansible_ssh_user	定义 hosts ssh 认证用户	ansible_ssh_user=yadmin
ansible_ssh_pass	定义 hosts ssh 认证密码	ansible_ssh_user='123456'
ansible_sudo	定义 hosts sudo 用户	ansible_sudo=yadmin
ansible_sudo_pass	定义 hosts sudo 密码	ansible_sudo_pass='123456'
ansible_sudo_exe	定义 hosts sudo 路径	ansible_sudo_exe=/usr/bin/sudo
ansible_connection	定义 hosts 连接方式	ansible_connection=local
ansible_ssh_private_key_file	定义 hosts 私钥	ansible_ssh_private_key_file=/root/key
ansible_shell_type	定义 hosts shell 类型	ansible_shell_type=zsh

（续）

参数	解释	例子
ansible_python_interpreter	定义 hosts 任务执行 python 路径	ansible_python_interpreter=/usr/bin/python2.6
ansible_*_interpreter	定义 hosts 其他语言解析器路径	ansible_ruby_interpreter=/usr/bin/ruby

3.2　Ansible Ad-Hoc 命令

我们经常会通过命令行的形式使用 Ansible 模块，Ansible 自带很多模块，可以直接使用这些模块。目前 Ansible 已经自带了 259 个模块，我们可以使用 ansible-doc -l 显示所有自带模块，还可以通过 ansible-doc “模块名”，查看模块的介绍以及案例。需要注意的是，如果使用 Ah-hoc 命令，Ansible 的一些插件功能就无法使用，比如 loop facts 功能等。本节就介绍一些日常的 Ad-Hoc 命令。

1. 执行命令

Ansible 命令都是并发执行的，我们可以针对目标主机执行任何命令。默认的并发数目由 ansible.cfg 中的 forks 值来控制。当然，也可以在运行 Ansible 命令的时候通过 -f 指定并发数。如果碰到执行任务时间很长的情况，也可以使用 Ansible 的异步执行功能来执行。下面我们简单地测试一下：

```
[root@Master ~]# ansible docker -m shell -a 'hostname' -o
172.17.42.101 | success | rc=0 | (stdout) 4b461620612a
172.17.42.103 | success | rc=0 | (stdout) 703bb6924049
172.17.42.102 | success | rc=0 | (stdout) 24e575b23394

[root@Master ~]# ansible docker -m shell -a 'uname -r' -f 5 -o
172.17.42.103 | success | rc=0 | (stdout) 3.10.5-3.el6.x86_64
172.17.42.102 | success | rc=0 | (stdout) 3.10.5-3.el6.x86_64
172.17.42.101 | success | rc=0 | (stdout) 3.10.5-3.el6.x86_64
```

使用异步执行功能，-P 0 的情况下会直接返回 job_id，然后针对主机根据 job_id 查询执行结果：

```
[root@Master ~]# ansible docker -B 120 -P 0 -m shell -a 'sleep
    10;hostname' -f 5 -o
background launch...
```

```
172.17.42.103 | success >> {"ansible_job_id": "288872560652.1072", "results_
    file": "/root/.ansible_async/288872560652.1072", "started": 1}

172.17.42.101 | success >> {"ansible_job_id": "288872560652.1096", "results_
    file": "/root/.ansible_async/288872560652.1096", "started": 1}

172.17.42.102 | success >> {"ansible_job_id": "288872560652.1097", "results_
    file": "/root/.ansible_async/288872560652.1097", "started": 1}
```

每台主机会产生不同的 job_id，可以通过 async_status 模块查看异步任务的状态和结果：

```
[root@Master ~ ]# ansible 172.17.42.101 -m async_status -a
    'jid=288872560652.1096'
172.17.42.101 | success >> {
    "ansible_job_id": "288872560652.1096",
    "changed": true,
    "cmd": "sleep 10;hostname",
    "delta": "0:00:10.010299",
    "end": "2015-06-07 12:01:02.553748",
    "finished": 1,
    "rc": 0,
    "start": "2015-06-07 12:00:52.543449",
    "stderr": "",
    "stdout": "4b461620612a",
    "warnings": []
}
```

当 -P 参数大于 0 的时候，Ansible 会自动根据 jod_id 去轮询查询执行结果：

```
[root@Master ~ ]# ansible docker -B 12 -P 1 -m shell -a 'sleep
    5;hostname' -f 5 -o
background launch...

172.17.42.103 | success >> {"ansible_job_id": "9195868444.1102", "results_
    file": "/root/.ansible_async/9195868444.1102", "started": 1}

172.17.42.101 | success >> {"ansible_job_id": "9195868444.1150", "results_
    file": "/root/.ansible_async/9195868444.1150", "started": 1}

172.17.42.102 | success >> {"ansible_job_id": "9195868444.1127", "results_
    file": "/root/.ansible_async/9195868444.1127", "started": 1}
```

```
172.17.42.102 | success >> {"ansible_job_id": "9195868444.1127",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1127", "started": 1}

172.17.42.103 | success >> {"ansible_job_id": "9195868444.1102",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1102", "started": 1}

172.17.42.101 | success >> {"ansible_job_id": "9195868444.1150",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1150", "started": 1}

<job 9195868444.1127> polling on 172.17.42.102, 11s remaining
<job 9195868444.1102> polling on 172.17.42.103, 11s remaining
<job 9195868444.1150> polling on 172.17.42.101, 11s remaining
172.17.42.101 | success >> {"ansible_job_id": "9195868444.1150",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1150", "started": 1}

172.17.42.102 | success >> {"ansible_job_id": "9195868444.1127",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1127", "started": 1}

172.17.42.103 | success >> {"ansible_job_id": "9195868444.1102",
    "changed": false, "finished": 0, "results_file": "/root/.ansible_
    async/9195868444.1102", "started": 1}

<job 9195868444.1127> polling on 172.17.42.102, 10s remaining
<job 9195868444.1102> polling on 172.17.42.103, 10s remaining
<job 9195868444.1150> polling on 172.17.42.101, 10s remaining
172.17.42.101 | success >> {"ansible_job_id": "9195868444.1150",
    "changed": true, "cmd": "sleep 5;hostname", "delta": "0:00:05.008892",
    "end": "2015-06-07 12:04:04.221857", "finished": 1, "rc": 0, "start":
    "2015-06-07 12:03:59.212965", "stderr": "", "stdout": "4b461620612a",
    "warnings": []}

172.17.42.102 | success >> {"ansible_job_id": "9195868444.1127",
    "changed": true, "cmd": "sleep 5;hostname", "delta": "0:00:05.016063",
    "end": "2015-06-07 12:04:04.156382", "finished": 1, "rc": 0, "start":
    "2015-06-07 12:03:59.140319", "stderr": "", "stdout": "24e575b23394",
    "warnings": []}

172.17.42.103 | success >> {"ansible_job_id": "9195868444.1102",
```

```
    "changed": true, "cmd": "sleep 5;hostname", "delta": "0:00:05.017769",
    "end": "2015-06-07 12:04:04.151042", "finished": 1, "rc": 0, "start":
    "2015-06-07 12:03:59.133273", "stderr": "", "stdout": "703bb6924049",
    "warnings": []}

<job 9195868444.1127> finished on 172.17.42.102 => {
    "ansible_job_id": "9195868444.1127",
    "changed": true,
    "cmd": "sleep 5;hostname",
    "delta": "0:00:05.016063",
    "end": "2015-06-07 12:04:04.156382",
    "finished": 1,
    "invocation": {
        "module_args": "jid=9195868444.1127",
        "module_name": "async_status"
    },
    "rc": 0,
    "start": "2015-06-07 12:03:59.140319",
    "stderr": "",
    "stdout": "24e575b23394",
    "warnings": []
}
<job 9195868444.1102> finished on 172.17.42.103 => {
    "ansible_job_id": "9195868444.1102",
    "changed": true,
    "cmd": "sleep 5;hostname",
    "delta": "0:00:05.017769",
    "end": "2015-06-07 12:04:04.151042",
    "finished": 1,
    "invocation": {
        "module_args": "jid=9195868444.1102",
        "module_name": "async_status"
    },
    "rc": 0,
    "start": "2015-06-07 12:03:59.133273",
    "stderr": "",
    "stdout": "703bb6924049",
    "warnings": []
}
<job 9195868444.1150> finished on 172.17.42.101 => {
    "ansible_job_id": "9195868444.1150",
    "changed": true,
    "cmd": "sleep 5;hostname",
    "delta": "0:00:05.008892",
```

```
    "end": "2015-06-07 12:04:04.221857",
    "finished": 1,
    "invocation": {
        "module_args": "jid=9195868444.1150",
        "module_name": "async_status"
    },
    "rc": 0,
    "start": "2015-06-07 12:03:59.212965",
    "stderr": "",
    "stdout": "4b461620612a",
    "warnings": []
}
```

2. 复制文件

我们还可以使用 copy 模块来批量下发文件，文件的变化是通过 MD5 值来判断的，如下所示：

```
[root@Master ~]# ansible docker -m copy -a 'src=hosts.py dest=/root/
    hosts.py  owner=root group=root mode=644 backup=yes' -o

172.17.42.103 | success >> {"changed": true, "checksum": "01800be332e
    6f2ff276e5a5e9c2a33c939c0388a", "dest": "/root/hosts.py", "gid":
    0, "group": "root", "md5sum": "7434e7fe32815562fd18e2e10457bd40",
    "mode": "0644", "owner": "root", "size": 736, "src": "/root/.
    ansible/tmp/ansible-tmp-1433677195.35-100509674775547/source",
    "state": "file", "uid": 0}

172.17.42.101 | success >> {"changed": true, "checksum": "01800be332e
    6f2ff276e5a5e9c2a33c939c0388a", "dest": "/root/hosts.py", "gid":
    0, "group": "root", "md5sum": "7434e7fe32815562fd18e2e10457bd40",
    "mode": "0644", "owner": "root", "size": 736, "src": "/root/.
    ansible/tmp/ansible-tmp-1433677195.43-169279680329648/source",
    "state": "file", "uid": 0}

172.17.42.102 | success >> {"changed": true, "checksum": "01800be
    332e6f2ff276e5a5e9c2a33c939c0388a", "dest": "/root/hosts.py",
    "gid": 0, "group": "root", "md5sum": "7434e7fe32815562fd18e2e104
    57bd40", "mode": "0644", "owner": "root", "size": 736, "src": "/
    root/.ansible/tmp/ansible-tmp-1433677195.36-58360851137130/source",
    "state": "file", "uid": 0}
```

我们再来验证文件下发功能，如下所示：

```
[root@Master ~]# ansible docker -m shell -a 'md5sum /root/hosts.py' -f
    5  -o
172.17.42.101 | success | rc=0 | (stdout) 7434e7fe32815562fd18e2e10457
    bd40  /root/hosts.py
172.17.42.102 | success | rc=0 | (stdout) 7434e7fe32815562fd18e2e10457
    bd40  /root/hosts.py
172.17.42.103 | success | rc=0 | (stdout) 7434e7fe32815562fd18e2e10457
    bd40  /root/hosts.py
```

3. 包和服务管理

Ansible 还支持直接使用 Ad-Hoc 命令来管理包和服务，如下所示：

```
[root@Master ~] ansible docker -m yum -a 'name=httpd state=latest' -f 5 -o

[root@Master ~]# ansible docker -m service -a 'name=httpd state=started
    ' -f 5 -o
172.17.42.101 | success >> {"changed": false, "name": "httpd", "state":
    "started"}
172.17.42.103 | success >> {"changed": false, "name": "httpd", "state":
    "started"}

172.17.42.102 | success >> {"changed": false, "name": "httpd", "state":
    "started"}

[root@Master ~]# ansible docker -m shell -a 'rpm -qa httpd' -f 5 -o
172.17.42.101 | success | rc=0 | (stdout) httpd-2.2.15-39.el6.centos.
    x86_64
172.17.42.102 | success | rc=0 | (stdout) httpd-2.2.15-39.el6.centos.
    x86_64
172.17.42.103 | success | rc=0 | (stdout) httpd-2.2.15-39.el6.centos.
    x86_64
```

验证服务运行情况：

```
[root@Master ~]# ansible docker -m shell -a 'netstat -tpln|grep httpd'
    -f 5
172.17.42.102 | success | rc=0 >>
tcp        0      0 :::80                       :::*
    LISTEN      799/httpd

172.17.42.101 | success | rc=0 >>
tcp        0      0 :::80                       :::*
```

```
    LISTEN      798/httpd

172.17.42.103 | success | rc=0 >>
tcp        0      0 :::80                       :::*
    LISTEN      752/httpd
```

4. 用户管理

首先通过 openssl 命令来生成一个密码，因为 ansible user 的 password 参数需要接受加密后的值，如下所示：

```
[root@Master ~]# echo ansible | openssl passwd -1 -stdin
$1$GPMku7yL$.qu3NC2geUv0J.NvgfCio1
```

然后使用 user 模块批量新建用户：

```
[root@Master ~]# ansible docker -m user -a 'name=shencan
    password="$1$GPMku7yL$.qu3NC2geUv0J.NvgfCio1"' -f 5 -o
172.17.42.103 | success >> {"changed": true, "comment": "",
    "createhome": true, "group": 500, "home": "/home/shencan", "name":
    "shencan", "password": "NOT_LOGGING_PASSWORD", "shell": "/bin/bash",
    "state": "present", "system": false, "uid": 500}

172.17.42.102 | success >> {"changed": true, "comment": "",
    "createhome": true, "group": 500, "home": "/home/shencan", "name":
    "shencan", "password": "NOT_LOGGING_PASSWORD", "shell": "/bin/bash",
    "state": "present", "system": false, "uid": 500}

172.17.42.101 | success >> {"changed": true, "comment": "",
    "createhome": true, "group": 500, "home": "/home/shencan", "name":
    "shencan", "password": "NOT_LOGGING_PASSWORD", "shell": "/bin/bash",
    "state": "present", "system": false, "uid": 500}
```

最后我们通过 SSH 登录来验证前面新建用户是否成功，密码就是前面的 ansible。

```
[root@Master ~]# ssh 172.17.42.103 -l shencan
shencan@172.17.42.103's password:
[shencan@703bb6924049 ~]$ logout
Connection to 172.17.42.103 closed.
```

关于 Ansible 的其他 Ad-Hoc 模块或者模块用法，可以通过 ansible-doc -l 和 ansible-doc 模块名进行查看。

3.3 Ansible playbook

playbook 是 Ansible 进行配置管理的组件，虽然 Ansible 的日常 Ad-Hoc 命令功能很强大，能完成一些基本配置管理工作，但是 Ad-Hoc 命令无法支撑复杂环境的配置管理工作。在我们实际使用 Ansible 的工作中，大部分时间都是在编写 playbook，这是 Ansible 非常重要的组件之一，所以第 4 章将用一整章的篇幅来详细讲解 Ansible 的 playbook。

3.4 Ansible facts

facts 组件是 Ansible 用于采集被管机器设备信息的一个功能，我们可以使用 setup 模块查机器的所有 facts 信息，可以使用 filter 来查看指定信息。整个 facts 信息被包装在一个 JSON 格式的数据结构中，ansible_facts 是最上层的值。下面我们通过实际操作来简单了解 facts 的数据结构：

```
[root@Master ~]# ansible 172.17.42.101 -m setup
172.17.42.101 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.17.0.2",
            "172.17.42.101"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::42:acff:fe11:2",
            "fe80::9093:d8ff:fe7d:f6d4"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
            "KEYBOARDTYPE": "pc",
            "KEYTABLE": "us",
            "LANG": "zh_CN.UTF-8",
            "crashkernel": "auto",
            "quiet": true,
            "rd_LVM_LV": "vg_master/lv_root",
            "rd_NO_DM": true,
```

```
            "rd_NO_LUKS": true,
            "rd_NO_MD": true,
            "rhgb": true,
            "ro": true,
            "root": "/dev/mapper/vg_master-lv_root"
        },
  ---------- 此处省略 N 行  ----------
```

所有的数据格式都是 JSON 格式，facts 还支持查看指定信息，比如下面只查看设备的 ipv4 地址：

```
[root@Master ~]# ansible 172.17.42.101 -m setup -a 'filter=ansible_
    all_ipv4_addresses'
172.17.42.101 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.17.0.2",
            "172.17.42.101"
        ]
    },
    "changed": false
}
```

facts 组件默认已经收集了很多的设备基础信息，这些信息可以在做配置管理的时候进行引用。下一章也会介绍把 facts 信息直接当作 playbook 变量信息进行引用。后面章节也会介绍如何扩展 facts 进行信息收集，可以定制 facts 以便收集我们想要的信息。下面介绍通过 facter 和 ohai 来扩展 facts 信息。

1. 使用 facter 扩展 facts 信息

使用过 Puppet 的读者都熟悉 facter 是 Puppet 里面一个负责收集主机静态信息的组件，Ansible 的 facts 功能也一样。Ansible 的 facts 组件也会判断被控制机器上是否安装有 facter 和 ruby-json 包，如果存在的话，Ansible 的 facts 也会采集 facter 信息。我们来查看以下机器信息：

```
[root@Master ~]# ansible 172.17.42.101 -m shell -a 'rpm -qa ruby-json
    facter'
172.17.42.101 | success | rc=0 >>
facter-1.6.18-8.el6.x86_64
ruby-json-1.4.6-1.el6.x86_64
```

然后运行 facter 模块查看 facter 信息：

```
[root@Master ~]# ansible 172.17.42.101 -m facter
172.17.42.101 | success >> {
    "architecture": "x86_64",
    "changed": false,
    "facterversion": "1.6.18",
    "hardwareisa": "x86_64",
    "hardwaremodel": "x86_64",
    "hostname": "4b461620612a",
    "id": "root",
    "interfaces": "eth0,eth1,lo",
    "ipaddress": "172.17.0.2",
    "ipaddress_eth0": "172.17.0.2",
    "ipaddress_eth1": "172.17.42.101",
    "ipaddress_lo": "127.0.0.1",
    "is_virtual": "true",
    "kernel": "Linux",
    "kernelmajversion": "3.10",
    "kernelrelease": "3.10.5-3.el6.x86_64",
   ------------ 省略 N 行 ------------------
```

当然，如果直接运行 setup 模块也会采集 facter 信息，如下所示：

```
[root@Master ~]# ansible 172.17.42.101 -m setup
172.17.42.101 | success >> {

------------- 省略 N 行 -----------
        "facter_swapfree": "991.48 MB",
        "facter_swapsize": "1024.00 MB",
        "facter_timezone": "UTC",
        "facter_uniqueid": "11ac0200",
        "facter_uptime": "8:41 hours",
        "facter_uptime_days": 0,
        "facter_uptime_hours": 8,
        "facter_uptime_seconds": 31299,
        "facter_virtual": "virtualbox",
        "module_setup": true
    },
    "changed": false
}
```

所有 facter 信息在 ansible_facts 下以 facter_ 开头，这些信息的引用方式跟 Ansible 自带 facts 组件收集的信息引用方式一致。

2. 使用 ohai 扩展 facts 信息

ohai 是 Chef 配置管理工具中检测节点属性的工具，Ansible 的 facts 也支持 ohai 信息的采集。当然需要被管机器上安装 ohai 包。下面介绍 ohai 相关信息的采集：

```
[root@Master ~]# ansible 172.17.42.1 -m shell -a 'gem list|grep ohai'
172.17.42.1 | success | rc=0 >>
ohai (8.4.0)
```

如果主机上没有安装 ohai 包，可以使用 gem 方式进行安装。如果存在 ohai 包，可以直接运行 ohai 模块查看 ohai 属性：

```
[root@Master ~]# ansible 172.17.42.1 -m ohai
172.17.42.1 | success >> {
-------------- 省略 N 行 --------------
    "ohai_time": 1433689885.3133919,
    "os": "linux",
    "os_version": "3.10.5-3.el6.x86_64",
    "platform": "centos",
    "platform_family": "rhel",
    "platform_version": "6.5",
    "root_group": "root",
    "uptime": "9 hours 00 minutes 06 seconds",
    "uptime_seconds": 32406,
    "virtualization": {
        "role": "guest",
        "system": "vbox",
        "systems": {
            "vbox": "guest"
        }
    }
}
```

如果直接运行 setup 模块，也会采集 ohai 信息：

```
[root@Master ~]# ansible 172.17.42.1 -m setup
172.17.42.1 | success >> {
    "ansible_facts": {
```

```
------------ 此处省略 N 行 ---------------
        "ohai_ohai_time": 1433690048.8647399,
        "ohai_os": "linux",
        "ohai_os_version": "3.10.5-3.el6.x86_64",
        "ohai_platform": "centos",
        "ohai_platform_family": "rhel",
        "ohai_platform_version": "6.5",
        "ohai_root_group": "root",
        "ohai_uptime": "9 hours 02 minutes 50 seconds",
        "ohai_uptime_seconds": 32570,
        "ohai_virtualization": {
            "role": "guest",
            "system": "vbox",
            "systems": {
                "vbox": "guest"
            }
        }
    },
    "changed": false
}
```

所有的 ohai 信息在 ansible_facts 下以 ohai_ 开头，这些信息的引用方式跟 Ansible 自带 facts 组件收集的信息引用方式一致。

3.5 Ansible role

Ansible 在 1.2 版本以后就支持了 role。在实际工作中有很多不同业务需要编写很多 playbook 文件，如果时间一久，对这些 playbook 文件很难进行维护，这个时候我们就可以采用 role 的方式管理 playbook。其实 role 只是对我们日常使用的 playbook 的目录结构进行一些规范，与日常的 playbook 没什么区别。下面通过一个案例来介绍 role 相关的目录规范，如下所示：

```
.
├── roles
│   └── nginx
│       ├── files
│       │   └── index.html
│       ├── handlers
│       │   └── main.yaml
```

```
|           ├── tasks
|           |   └── main.yaml
|           ├── templates
|           |   └── nginx.conf.j2
|           └── vars
|               └── main.yaml
└── site.yaml

7 directories, 6 files
```

这里简单了定义了一个 role，它的主要工作就是配置部署 Nginx 服务。role 的所有文件内容都在 nginx 目录下。跟 role 同级别的还有一个 site.yaml 文件，这个文件就是 role 引用的入口文件，文件的名字可以随意定义。files 目录里面存放一些静态文件，handlers 目录里面存放一些 task 的 handler。tasks 目录里面就是我们平常写的 playbook 中的 task。templates 目录里面存放着 jinja2 模板文件。vars 目录下存放着变量文件。下面分别来查看每个文件的内容：

```
[root@Master nginx]# cat site.yaml
---
- hosts: 172.17.42.103
    roles:
        - { role: nginx, version: 1.0.15 }

[root@Master nginx]# cat roles/nginx/tasks/main.yaml
---
- name: Install nginx package
    yum: name=nginx-{{ version }} state=present
- name: Copy nginx.conf Template
    template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf owner=root
        group=root backup=yes mode=0644
    notify: restart nginx
- name: Copy index html
    copy: src=index.html dest=/usr/share/nginx/html/index.html
        owner=root group=root backup=yes mode=0644
- name: make sure nginx service running
    service: name=nginx state=started

[root@Master nginx]# cat roles/nginx/handlers/main.yaml
---
- name: restart nginx
```

```
    service: name=nginx state=restarted

[root@Master nginx]# cat roles/nginx/templates/nginx.conf.j2 |grep '\{{'
worker_processes  {{ ansible_processor_cores }};

[root@Master nginx]# cat roles/nginx/files/index.html
hello kugou
```

上面介绍了 Nginx role 的所有文件。如果以后需要对 role 进行修改或者调整，只需修改相应的文件即可。如果还想把这个 Nginx role 分享给其他朋友，也只需把整个目录分享即可。下面执行这个 role：

```
[root@Master nginx]# ansible-playbook -i /root/hosts site.yaml

PLAY [172.17.42.103] ***************************************************
GATHERING FACTS ********************************************************
ok: [172.17.42.103]

TASK: [nginx | Install nginx package] **********************************
changed: [172.17.42.103]

TASK: [nginx | Copy nginx.conf Template] *******************************
changed: [172.17.42.103]

TASK: [nginx | Copy index html] ****************************************
changed: [172.17.42.103]

TASK: [nginx | make sure nginx service running] ************************
changed: [172.17.42.103]

NOTIFIED: [nginx | restart nginx] **************************************
changed: [172.17.42.103]

PLAY RECAP *************************************************************
172.17.42.103     : ok=6    changed=5    unreachable=0    failed=0
```

role 执行完成后，通过 curl 命令进行 Nginx 服务测试。

```
[root@Master nginx]# curl 172.17.42.103
hello kugou
```

role 文件的内容还是比较简单的，如果我们定义一个 role，然后在写 playbook 的

时候静态文件跟 jinja2 模板文件直接用相对路径就行，Ansible 会自动去相应的目录下寻找相应文件。另外，role 不会关心哪些设备使用它，它只是关于一个功能的集合，只需要编写一个 playbook 去引用即可。比如上面的 site.yaml，文件，它就是一个简单的 playbook 文件，里面有目标 hosts 参数指定目标主机，然后它会引用一个 role 参数去调用我们定义的 role。关于 role 目录的路径还可以使用 ansible.cfg 中的 roles_path 进行指定，也可以引用当前目录下的 roles 目录。

3.6　Ansible Galaxy

Ansible 的 Galaxy 是 Ansible 官方一个分享 role 的功能平台，它的网址是 https://galaxy.ansible.com/list#/roles。可以把你编写的 role 通过 ansible-galaxy 上传到 Galaxy 网站供其他人下载和使用。可以通过 ansible-galaxy 命令很简单地实现 role 的分享和安装。当然 Ansible 也支持直接从 GitHub 上下载 role。在我们使用 ansible-galaxy 命令下载 role 的时候需要了解 role 的运行平台和 Ansible 依赖版本以及相关依赖，等等。日常工作中我们使用 ansible-galaxy install 就可以，默认会安装到 /etc/ansible/roles/ 目录下，其引用跟我们自己写的 role 引用方式一样。

3.7　本章小结

通过本章的学习我们已经了解 Ansible 一些常用的组件，并对 Ansible Ad-Hoc 命令中常用的几个命令进行了演示，我们还介绍了 Ansible playbook、facts、role、Galaxy 等。学习一个软件，首先得了解这个软件的一些常用组件，以及如何使用这个软件。下一章将介绍 Ansible 在做配置管理工作中最重要的一个组件 playbook。