

```
"changed": false,
"checksum": "93d304488245bb2769752b95e0180607effc69ad",
"dest": "/etc/motd",
"gid": 0,
"group": "root",
"mode": "0644",
"owner": "root",
"path": "/etc/motd",
"secontext": "system_u:object_r:etc_t:s0",
"size": 35,
"state": "file",
"uid": 0
}
serverd.lab.example.com | SUCCESS => {
"ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
},
"changed": false,
"checksum": "93d304488245bb2769752b95e0180607effc69ad",
"dest": "/etc/motd",
"gid": 0,
"group": "root",
"mode": "0644",
"owner": "root",
"path": "/etc/motd",
"secontext": "system_u:object_r:etc_t:s0",
"size": 35,
"state": "file",
"uid": 0
}
servera.lab.example.com | SUCCESS => {
"ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
},
"changed": false,
"checksum": "93d304488245bb2769752b95e0180607effc69ad",
"dest": "/etc/motd",
"gid": 0,
"group": "root",
"mode": "0644",
"owner": "root",
"path": "/etc/motd",
"secontext": "system_u:object_r:etc_t:s0",
"size": 35,
"state": "file",
"uid": 0
}
```

8. 若要以另一种方式确认，请以 all 主机组为目标运行一个临时命令，以使用 command 模块来执行 `cat /etc/motd` 命令。`ansible` 的输出应当为所有主机都显示字符串 `This server is managed by Ansible.`。运行此临时命令不需要特权升级。

```
[student@workstation deploy-review]$ ansible all -m command -a 'cat /etc/motd'
serverb.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

servera.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

serverd.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

serverc.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.
```

9. 在 workstation 上运行 `lab deploy-review grade` 以检查您的工作。

```
[student@workstation deploy-review]$ lab deploy-review grade
```

完成

在 workstation 上，运行 `lab deploy-review finish` 脚本来清理本实验中创建的资源。

```
[student@workstation ~]$ lab deploy-review finish
```

总结

在本章中，您学到了：

- 任何其上装有 Ansible 且能够访问所需的配置文件和 playbook 以管理远程系统（受管主机）的系统都称为控制节点。
- 受管主机在清单中定义。主机模式用于引用清单中定义的受管主机。
- 清单可以是静态文件，或者由程序从外部来源（如目录服务或云管理系统）动态生成。
- Ansible 以一定的优先级顺序在多个位置上寻找其配置文件。它将使用找到的第一个配置文件，并忽略所有其他文件。
- **ansible** 命令用于在受管主机上执行临时命令。
- 临时命令通过使用模块及其参数确定要执行的操作，也可利用 Ansible 的特权升级功能。

章 3

实施 PLAYBOOK

目标

编写简单的 Ansible Playbook 并运行它，以在多个主机上自动执行任务。

培训目标

- 编写基本的 Ansible Playbook，再使用 **ansible-playbook** 命令运行它。
- 编写一个使用多个 play 和各 play 特权升级的 playbook。
- 有效使用 **ansible-doc** 学习如何使用新模块来实施 play 任务。

章节

- 编写和运行 Playbook (及引导式练习)
- 实施多个 Play (及引导式练习)

实验

- 实施 Playbook

编写和运行 PLAYBOOK

培训目标

学完本章节后，您应该能够编写基本的 Ansible Playbook 并使用 **ansible-playbook** 该命令运行它。

ANSIBLE PLAYBOOK 和临时命令

临时命令可以作为一次性命令对一组目标主机运行一项简单的任务。不过，若要真正发挥 Ansible 的力量，需要了解如何使用 playbook 以可轻松重复的方式对一组目标主机执行多项复杂的任务。

play 是针对清单中选定的主机运行的一组有序任务。playbook 是一个文本文件，其中包含由一个或多个按特定顺序运行的 play 组成的列表。

Play 可以让您将一系列冗长而复杂的手动管理任务转变为可轻松重复的例程，并且具有可预测的成功成果。在 playbook 中，您可以将 play 内的任务序列保存为人类可读并可立即运行的形式。根据任务的编写方式，任务本身记录了部署应用或基础架构所需的步骤。

格式化 ANSIBLE PLAYBOOK

为帮助您理解 playbook 的格式，请回顾上一章节中的这条临时命令：

```
[student@workstation ~]$ ansible -m user -a "name=newbie uid=4000 state=present" \
> servera.lab.example.com
```

这可以重新编写为一个单任务 play 并保存在 playbook 中。生成的 playbook 如下方所示：

例 3.1. 简单的 Playbook

```
---
- name: Configure important user consistently
  hosts: servera.lab.example.com
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        state: present
```

Playbook 是以 YAML 格式编写的文本文件，通常使用扩展名 **.yml** 保存。Playbook 使用空格字符缩进来表示其数据结构。YAML 对用于缩进的空格数量没有严格的要求，但有两个基本的规则。

- 处于层次结构中同一级别的数据元素（例如同一列表中的项目）必须具有相同的缩进量。
- 如果项目属于其他项目的子项，其缩进量必须大于父项。

您也可以通过添加空行来提高可读性。

**重要**

只有空格字符可用于缩进；不允许使用制表符。

如果您使用 **vi** 文本编辑器，您可以应用一些设置，以便能更加轻松地编辑 playbook。例如，在 **\$HOME/.vimrc** 文件中添加下面这行后，如果 **vi** 检测到您在编辑 YAML 文件，它将在 **Tab** 键按下时执行一个双空格缩进，并自动缩进后续行。

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

Playbook 开头的一行由三个破折号 (---) 组成，这是文档开始标记。其末尾可能使用三个圆点 (...) 作为文档结束标记，尽管在实践中这通常会省略。

在这两个标记之间，会以一个 play 列表的形式来定义 playbook。YAML 列表中的项目以一个破折号加空格开头。例如，YAML 列表可能显示如下：

```
- apple
- orange
- grape
```

在例 3.1 “简单的 Playbook” 中，--- 后的行以破折号开头，开始列出 play 列表中第一个（和唯一的）play。

Play 本身是一个键值对集合。同一 play 中的键应当使用相同的缩进量。以下示例显示了具有三个键的 YAML 代码片段。前两个键具有简单的值。第三个将含有三个项目的列表作为值。

```
name: just an example
hosts: webservers
tasks:
  - first
  - second
  - third
```

原始示例 play 有三个键：**name**、**hosts** 和 **tasks**，因为这些键都有相同的缩进。

示例 play 的第一行开头是破折号加空格（表示该 play 是列表中的第一项），而后是第一个键，即 **name** 属性。**name** 将一个任意字符串作为标签与该 play 关联。这标识了该 play 的用途。**name** 键是可选的，但建议使用它，因为它有助于记录您的 playbook。这在 playbook 中包含多个 play 时特别有用。

```
- name: Configure important user consistently
```

Play 中的第二个键是 **hosts** 属性，它指定对其运行 play 中的任务的主机。与用于 **ansible** 命令的参数相似，**hosts** 属性将主机模式取为值，如清单中受管主机或组的名称。

```
hosts: servera.lab.example.com
```

Play 中的最后一个键是 **tasks** 属性，其值指定要为该 play 运行的任务的列表。本例中只有一项任务，该任务使用特定参数运行 **user** 模块（以确保用户 **newbie** 存在并且具有 UID 4000）。

```
tasks:
  - name: newbie exists with UID 4000
    user:
      name: newbie
      uid: 4000
      state: present
```

作为 play 中的一部分，**tasks** 属性按顺序实际列出要在受管主机上运行的任务。列表中各项任务本身是一个键值对集合。

在这个示例中，play 中的唯一任务含有两个键：

- **name** 是记录任务用途的可选标签。最好命名您的所有任务，从而帮助记录自动流程中每一步的用途。
- **user** 是要为这个任务运行的模块。其参数作为一组键值对传递，它们是模块的子项（**name**、**uid** 和 **state**）。

下方为另一个含有多项任务的 **tasks** 属性的示例，该示例使用 **service** 模块确保为多个网络服务启用在引导时启动：

```
tasks:
  - name: web server is enabled
    service:
      name: httpd
      enabled: true

  - name: NTP server is enabled
    service:
      name: chronyd
      enabled: true

  - name: Postfix is enabled
    service:
      name: postfix
      enabled: true
```



重要

playbook 中 play 和任务列出的顺序很重要，因为 Ansible 会按照相同的顺序运行它们。

您到目前为止看到的 playbook 是基本的示例；随着本课程的进行，您会看到可以通过 play 和任务达成目标的更为复杂的示例。

运行 PLAYBOOK

ansible-playbook 命令可用于运行 playbook。该命令在控制节点上执行，要运行的 playbook 的名称则作为参数传递。

```
[student@workstation ~]$ ansible-playbook site.yml
```

在运行 playbook 时，将生成输出来显示所执行的 play 和任务。输出中也会报告执行的每一项任务的结果。

以下示例中显示了一个简单 playbook 的内容，后面是运行它的结果。

```
[student@workstation playdemo]$ cat webserver.yml
---
- name: play to setup web server
  hosts: servera.lab.example.com
  tasks:
    - name: latest httpd version installed
      yum:
        name: httpd
        state: latest
  ...output omitted...
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

请注意，在 playbook 运行时，屏幕中会显示每个 play 和任务的 **name** 键的值。（**Gathering Facts** 任务是一项特别的任务，**setup** 模块通常在 play 启动时自动运行这项任务。本课程稍后会对此进行阐述。）对于含有多个 play 和任务的 playbook，设置 **name** 属性后可以更加轻松地监控 playbook 执行的进展。

您也应该会看到 **latest httpd version installed** 任务对 `servera.lab.example.com` 的状态为 **changed**。这表示该任务更改了这台主机上的某一设置，以确保符合其规格要求。在本例中，这表示 httpd 软件包或许没有安装或者不是最新版本。

通常而言，Ansible playbook 中的任务是幂等的，而且能够安全地多次运行 playbook。如果目标受管主机已处于正确的状态，则不应进行任何更改。例如，假定再次运行上例中的 playbook：

```
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=0    unreachable=0    failed=0
```

这一次，所有任务都以状态 **ok** 传递，而且不报告任何更改。

提高输出的详细程度

ansible-playbook 命令提供的默认输出不提供详细的任务执行信息。**ansible-playbook -v** 命令提供了额外的信息，总共有四个级别。

配置 Playbook 执行的输出详细程度

选项	描述
-v	显示任务结果。
-vv	任务结果和任务配置都会显示
-vvv	包含关于与受管主机连接的信息
-vvvv	增加了连接插件相关的额外详细程度选项，包括受管主机上用于执行脚本的用户，以及所执行的脚本

语法验证

在执行 playbook 之前，最好要进行验证，确保其内容的语法正确无误。**ansible-playbook** 命令提供了一个 **--syntax-check** 选项，可用于验证 playbook 的语法。下例演示了一个 playbook 成功通过语法验证。

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
playbook: webserver.yml
```

语法验证失败时，将报告语法错误。输出中也包含语法问题在 playbook 中的大致位置。以下示例演示了一个 playbook 语法验证失败，其中 play 的 **name** 属性后缺少了空格分隔符。

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.
mapping values are not allowed in this context

The error appears to have been in ...output omitted... line 3, column 8, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

- name:play to setup web server
  hosts: servera.lab.example.com
    ^ here
```

执行空运行

您可以使用 **-C** 选项对 playbook 执行空运行。这会使 Ansible 报告在执行该 playbook 时将会发生什么更改，但不会对受管主机进行任何实际的更改。

下例演示了一个 playbook 的空运行，它包含单项任务，可确保在受管主机上安装了最新版本的 httpd 软件包。注意该空运行报告此任务会对受管主机产生的更改。

```
[student@workstation ~]$ ansible-playbook -C webserver.yml  
PLAY [play to setup web server] *****  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [latest httpd version installed] *****  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```



参考文献

ansible-playbook(1) 帮助手册

Playbook 简介 — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

Playbook — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

检查模式（“空运行”）— Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks_checkmode.html

► 指导练习

编写和运行 PLAYBOOK

在本练习中，您将编写和运行一个 Ansible Playbook。

成果

您应能够使用基本的 YAML 语法和 Ansible Playbook 结构编写 playbook，并通过 **ansible-playbook** 命令成功运行它。

在你开始之前

以 **student** 用户身份并使用 **student** 作为密码登录 **workstation**。

在 **workstation** 上，运行 **lab playbook-basic start** 命令。此功能将确保受管主机 **serverc.lab.example.com** 和 **serverd.lab.example.com** 能在网络上访问。它也将确保控制节点上安装了正确的 Ansible 配置文件和清单文件。

```
[student@workstation ~]$ lab playbook-basic start
```

已经在 **workstation** 上为此练习创建了工作目录 **/home/student/playbook-basic**。该目录中已填充了 **ansible.cfg** 配置文件，还填充了 **inventory** 清单文件，后者定义包含上方所列受管主机作为其成员的 **web** 组。

在此目录中，使用文本编辑器创建名为 **site.yml** 的 playbook。此 playbook 中含有一个 play，它应当以 **web** 主机组的成员为目标。该 playbook 应使用任务来确保受管主机上满足下列条件：

- 利用 **yum** 模块确保保存在 **httpd** 软件包。
- 利用 **copy** 模块将本地的 **files/index.html** 文件复制到各受管主机上的 **/var/www/html/index.html**。
- 利用 **service** 模块启动和启用 **httpd** 服务。

您可以使用 **ansible-doc** 命令帮助了解各个模块所需的关键字。

编写了 playbook 后，验证其语法，然后使用 **ansible-playbook** 运行 playbook 来实施配置。

- 1. 更改为 **/home/student/playbook-basic** 目录。

```
[student@workstation ~]$ cd ~/playbook-basic
```

- 2. 使用文本编辑器，创建名为 **/home/student/playbook-basic/site.yml** 的新 playbook。以 **web** 主机组中的主机为目标开始编写一个 play。

2.1. 创建并打开 **~/playbook-basic/site.yml**。文件的第一行应该是三个破折号，表示 playbook 的开头。

```
---
```

- 2.2. 下一行是 play 的开头。它需要以破折号加空格开头，后面接上 play 中的第一个关键字。通过 **name** 关键字，使用任意字符串命名 play，以记录该 play 的用途。

```
---
```

```
- name: Install and start Apache HTTPD
```

- 2.3. 添加一个 **hosts** 关键字值对，以指定在清单的 **web** 主机组中的主机上运行的 play。确保 **hosts** 关键字缩进两个空格，使其与上一行中的 **name** 关键字对齐。

完整的 **site.yml** 文件现在应当显示如下：

```
---
```

```
- name: Install and start Apache HTTPD
  hosts: web
```

► 3. 继续编辑 **/home/student/playbook-basic/site.yml** 文件，再添加 **tasks** 关键字以及说明中为您的 play 指定的三项任务。

- 3.1. 添加 **tasks** 关键字并缩进两个空格（与 **hosts** 关键字对齐），以开始任务列表。您的文件现在应当显示如下：

```
---
```

```
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
```

- 3.2. 添加第一项任务。缩进四个空格，并使用破折号加空格开始任务，然后为任务取一个名称，如 **httpd package is present**。对此任务使用 **yum** 模块。将模块关键字再缩进两个空格；将软件包名称设为 **httpd**，将软件包状态设为 **present**。任务应如下所示：

```
- name: httpd package is present
  yum:
    name: httpd
    state: present
```

- 3.3. 添加第二项任务。使格式与上一任务匹配，再为任务取一个名称，如 **correct index.html is present**。使用 **copy** 模块。模块关键字应将 **src** 键设为 **files/index.html**，将 **dest** 键设为 **/var/www/html/index.html**。任务应如下所示：

```
- name: correct index.html is present
  copy:
    src: files/index.html
    dest: /var/www/html/index.html
```

- 3.4. 添加第三项任务，以启动并启用 **httpd** 服务。使格式与前两项任务匹配，再为任务取一个名称，如 **httpd is started**。对此任务使用 **service** 模块。将服务的 **name**

键设为 `httpd`, `state` 键设为 `started`, 并且 `enabled` 键设为 `true`。任务应如下所示：

```
- name: httpd is started
  service:
    name: httpd
    state: started
    enabled: true
```

3.5. 整个 `site.yml` Ansible Playbook 应当与以下例子相符。确保 play 的关键字、任务列表和各任务关键字的缩进都正确。

```
---
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
    - name: httpd package is present
      yum:
        name: httpd
        state: present

    - name: correct index.html is present
      copy:
        src: files/index.html
        dest: /var/www/html/index.html

    - name: httpd is started
      service:
        name: httpd
        state: started
        enabled: true
```

保存文件并退出文本编辑器。

- 4. 在运行 playbook 之前, 请运行 `ansible-playbook --syntax-check site.yml` 命令以验证其语法是否正确。如果报告任何错误, 请更正后再继续下一步。您应看到类似于下文的输出:

```
[student@workstation playbook-basic]$ ansible-playbook --syntax-check site.yml
playbook: site.yml
```

- 5. 运行 playbook。通读生成的输出, 确保所有任务都已成功完成。

```
[student@workstation playbook-basic]$ ansible-playbook site.yml
PLAY [Install and start Apache HTTPD] ****
TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]
```

```
TASK [httpd package is present] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [correct index.html is present] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [httpd is started] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

PLAY RECAP ****
serverc.lab.example.com      : ok=4      changed=3      unreachable=0      failed=0
serverd.lab.example.com      : ok=4      changed=3      unreachable=0      failed=0
```

- 6. 如果一切都正常，您应能够再次运行 playbook，并且看到所有任务都完成且不对受管主机执行任何更改。

```
[student@workstation playbook-basic]$ ansible-playbook site.yml

PLAY [Install and start Apache HTTPD] **

TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [httpd package is present] ****
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [correct index.html is present] ****
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [httpd is started] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

PLAY RECAP ****
serverc.lab.example.com      : ok=4      changed=0      unreachable=0      failed=0
serverd.lab.example.com      : ok=4      changed=0      unreachable=0      failed=0
```

- 7. 使用 curl 命令验证 serverc 和 serverd 是否都配置为 HTTPD 服务器。

```
[student@workstation playbook-basic]$ curl serverc.lab.example.com
This is a test page.
[student@workstation playbook-basic]$ curl serverd.lab.example.com
This is a test page.
```

完成

在 workstation 上，运行 `lab playbook-basic finish` 脚本来清理本章中创建的资源。

```
[student@workstation ~]$ lab playbook-basic finish
```

本引导式练习到此结束。

实施多个 PLAY

培训目标

学完本节后，您应能够：

- 编写一个使用多个 play 和各 play 特权升级的 playbook。
- 有效使用 **ansible-doc** 学习如何使用新模块来实施 play 任务。

编写多个 PLAY

Playbook 是一个 YAML 文件，含有由一个或多个 play 组成的列表。记住一个 play 按顺序列出了要对清单中的选定主机执行的任务。因此，如果一个 playbook 中含有多个 play，每个 play 可以将其任务应用到单独的一组主机。

在编排可能涉及对不同主机执行不同任务的复杂部署时，这会大有帮助。您可以这样编写 playbook：对一组主机运行一个 play，完成后再对另一组主机运行另一个 play。

编写包含多个 play 的 playbook 非常简单。Playbook 中的各个 play 编写为 playbook 中的顶级列表项。各个 play 是含有常用 play 关键字的列表项。

以下示例显示了含有两个 play 的简单 playbook。第一个 play 针对 `web.example.com` 运行，第二个 play 则针对 `database.example.com` 运行。

```
---
# This is a simple playbook with two plays

- name: first play
  hosts: web.example.com
  tasks:
    - name: first task
      yum:
        name: httpd
        status: present

    - name: second task
      service:
        name: httpd
        enabled: true

- name: second play
  hosts: database.example.com
  tasks:
    - name: first task
      service:
        name: mariadb
        enabled: true
```

PLAY 中的远程用户和特权升级

Play 可以将不同的远程用户或特权升级设置用于 play，取代配置文件中指定的默认设置。这些都在 play 本身中与 **hosts** 或 **tasks** 关键字相同的级别上设置。

用户属性

Playbook 中的任务通常通过与受管主机的网络连接来执行。与临时命令相同，用于任务执行的用户帐户取决于 Ansible 配置文件 **/etc/ansible/ansible.cfg** 中的不同关键字。运行任务的用户可以通过 **remote_user** 关键字来定义。不过，如果启用了特权升级，**become_user** 等其他关键字也会发生作用。

如果用于任务执行的 Ansible 配置中定义的远程用户不合适，可以通过在 play 中使用 **remote_user** 关键字来覆盖。

```
remote_user: remoteuser
```

特权升级属性

也提供额外的关键字，从而在 playbook 内定义特权升级参数。**become** 布尔值关键字可用于启用或禁用特权升级，无论它在 Ansible 配置文件中的定义为何。它可以取 **yes** 或 **true** 值来启用特权升级，或者取 **no** 或 **false** 值来禁用它。

```
become: true
```

如果启用了特权升级，则可以使用 **become_method** 关键字来定义特定 play 期间要使用的特权升级方法。以下示例中指定 **sudo** 用于特权升级。

```
become_method: sudo
```

此外，启用了特权升级时，**become_user** 关键字可定义特定 play 上下文内要用于特权升级的用户帐户。

```
become_user: privileged_user
```

以下示例演示了如何在 play 中使用这些关键字：

```
- name: /etc/hosts is up to date
  hosts: datacenter-west
  remote_user: automation
  become: yes

  tasks:
    - name: server.example.com in /etc/hosts
      lineinfile:
        path: /etc/hosts
        line: '192.0.2.42 server.example.com server'
        state: present
```

查找用于任务的模块

模块文档

Ansible 随附打包的大量模块为管理员提供了许多用于常见管理任务的工具。本课程前面介绍了 Ansible 文档网站 <http://docs.ansible.com>。通过该网站上的模块索引，可以轻松浏览 Ansible 随附的模块列表。例如，适于用户和服务管理的模块可以在 Systems Modules 下找到，而适合数据库管理的模块则可在 Database Modules 下找到。

对于每一个模块，Ansible 文档网站提供了其功能摘要，以及关于如何通过模块的选项来调用各项具体功能的说明。文档还提供了实用的示例，演示各个模块的用法，以及任务中关键字的设置方法。

您已使用过 **ansible-doc** 命令来查找关于本地系统上安装的模块的信息。作为复习，要查看控制节点上可用模块的列表，可运行 **ansible-doc -l** 命令。这将显示模块名称列表以及其功能的概要。

```
[student@workstation modules]$ ansible-doc -l
a10_server           Manage A10 Networks ... devices' server object.
a10_server_axapi3    Manage A10 Networks ... devices
a10_service_group    Manage A10 Networks ... devices' service groups.
a10_virtual_server   Manage A10 Networks ... devices' virtual servers.
...output omitted...
zfs_facts            Gather facts about ZFS datasets.
znode                Create, ... and update znodes using ZooKeeper
zpool_facts          Gather facts about ZFS pools.
zypper               Manage packages on SUSE and openSUSE
zypper_repository    Add and remove Zypper repositories
```

使用 **ansible-doc [module name]** 命令来显示模块的详细文档。与 Ansible 文档网站一样，该命令提供模块功能的概要、其不同选项的详细信息，以及示例。以下示例演示了为 **yum** 模块显示的文档。

```
[student@workstation modules]$ ansible-doc yum
> YUM      (/usr/lib/python3.6/site-packages/ansible/modules/packaging/os/yum.py)

    Installs, upgrade, downgrades, removes, and lists packages and groups with
    the `yum` package manager. This module only works on Python 2. If you require
    Python
        3 support see the [dnf] module.

    * This module is maintained by The Ansible Core Team
    * note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):

- allow_downgrade
    Specify if the named package and version is allowed to downgrade a maybe
    already installed higher version of that package. Note that setting
        allow_downgrade=True can make this module behave in a non-idempotent way.
    The task could end up with a set of packages that does not match the complete
    list of
        specified packages to install (because dependencies between the downgraded
        package and others can cause changes to the packages which were in the earlier
        transaction).
```

```

[Default: no]
type: bool
version_added: 2.4

- autoremove
  If `yes', removes all "leaf" packages from the system that were originally
  installed as dependencies of user-installed packages but which are no longer
  required
    by any such package. Should be used alone or when state is `absent'
    NOTE: This feature requires yum >= 3.4.3 (RHEL/CentOS 7+)
  [Default: no]
  type: bool
  version_added: 2.7

- bugfix
  If set to `yes', and `state=latest' then only installs updates that have
  been marked bugfix related.
  [Default: no]
  version_added: 2.6

- conf_file
  The remote yum configuration file to use for the transaction.
  [Default: (null)]
  version_added: 0.6

- disable_excludes
  Disable the excludes defined in YUM config files.
  If set to `all', disables all excludes.
  If set to `main', disable excludes defined in [main] in yum.conf.
  If set to `repoid', disable excludes defined for given repo id.
  [Default: (null)]
  version_added: 2.7

- disable_gpg_check
  Whether to disable the GPG checking of signatures of packages being
  installed. Has an effect only if state is `present' or `latest'.
  [Default: no]
  type: bool
  version_added: 1.2

- disable_plugin
  `Plugin' name to disable for the install/update operation. The disabled
  plugins will not persist beyond the transaction.
  [Default: (null)]
  version_added: 2.5

- disablerepo
  `Repopid' of repositories to disable for the install/update operation.
  These repos will not persist beyond the transaction. When specifying multiple
  repos,
    separate them with a `","'.
  As of Ansible 2.7, this can alternatively be a list instead of `","'
  separated string
  [Default: (null)]

```

ansible-doc 命令还提供 **-s** 选项，它会生成示例输出，可以充当如何在 playbook 中使用特定模块的示范。此输出可以作为起步模板，包含在实施该模块以执行任务的 playbook 中。输出中包含的注释，提醒管理员各个选项的用法。下例演示了 **yum** 模块的这种输出。

```
[student@workstation ~]$ ansible-doc -s yum
- name: Manages packages with the `yum` package manager
  yum:
    allow_downgrade:          # Specify if the named package ...
    autoremove:               # If `yes`, removes all "leaf" packages ...
    bugfix:                   # If set to `yes`, ...
    conf_file:                # The remote yum configuration file ...
    disable_excludes:          # Disable the excludes ...
    disable_gpg_check:         # Whether to disable the GPG ...
    disable_plugin:            # `Plugin` name to disable ...
    disablerepo:               # `Repol` of repositories ...
    download_only:              # Only download the packages, ...
    enable_plugin:              # `Plugin` name to enable ...
    enablerepo:                # `Repol` of repositories to enable ...
    exclude:                   # Package name(s) to exclude ...
    installroot:                # Specifies an alternative installroot, ...
    list:                      # Package name to run ...
    name:                      # A package name or package specifier ...
    releasever:                # Specifies an alternative release ...
    security:                  # If set to `yes`, ...
    skip_broken:                # Skip packages with ...
    state:                     # Whether to install ... or remove ... a package.
    update_cache:                # Force yum to check if cache ...
    update_only:                 # When using latest, only update ...
    use_backend:                # This module supports `yum` ...
    validate_certs:              # This only applies if using a https url ...
```

模块维护

Ansible 随附了大量模块，它们可用于执行许多任务。上游社区非常活跃，这些模块也可能处于不同的开发阶段。模块的 **ansible-doc** 文档应该会指定上游 Ansible 社区中维护该模块的人，以及该模块的开发状态。这在该模块的 **ansible-doc** 输出末尾的 **METADATA** 部分中指明。

status 字段记录模块的开发状态：

- **stableinterface**: 模块的关键字稳定，将尽力确保不删除关键字或更改其含义。
- **preview**: 模块处于技术预览阶段，可能不稳定，其关键字可能会更改，或者它可能需要本身会受到不兼容更改的库或 Web 服务。
- **deprecated**: 模块已被弃用，未来某一发行版中将不再提供。
- **removed**: 模块已从发行版中移除，但因文档需要存在存根，以帮助之前的用户迁移到新的模块。



注意

stableinterface 状态仅表明模块界面稳定，并不评价模块的代码质量。

supported_by 字段记录上游 Ansible 社区中维护该模块的人。可能的值包括：

- **core**: 由上游“核心”Ansible 开发人员维护，始终随 Ansible 提供。
- **curated**: 模块由社区中的合作伙伴或公司提交并维护。这些模块的维护者必须留意报告的任何问题，或者调取针对该模块提出的请求。在社区维护人员批准了更改后，上游“核心”开发人员审核对策划模块提出的更改。核心提交者也确保因为 Ansible 引擎中的变化而对这些模块造成的问题得到修正。这些模块目前随 Ansible 提供，但是可能会在未来某个时候另外打包。
- **community**: 模块不受到核心上游开发人员、合作伙伴或公司的支持，完全由一般开源社区维护。此类别中的模块仍然完全可用，但对问题的响应速度完全取决于社区。这些模块目前也随 Ansible 提供，但是可能会在未来某个时候另外打包。

上游 Ansible 社区有用于 Ansible 的问题跟踪器及其集成模块，网址为：<https://github.com/ansible/ansible/issues>。

有时候，没有现成的模块可以用于达成您的目标。作为最终用户，您也可以自己编写私有的模块，或者从第三方获取模块。Ansible 会在 **ANSIBLE_LIBRARY** 环境变量指定的位置查找自定义模块；或者，如果未设置此变量，则由当前 Ansible 配置文件中的 **library** 关键字指定。Ansible 也在相对于当前运行的 playbook 的 **./library** 目录中搜索自定义模块。

```
library = /usr/share/my_modules
```

与编写模块相关的信息已超出本课程的范畴。有关编写方法的文档位于 https://docs.ansible.com/ansible/latest/dev_guide/developing_modules.html。

**重要**

使用 **ansible-doc** 命令可以查找和了解如何为您的任务使用模块。

尽管 **command**、**shell** 和 **raw** 模块的用法可能看似简单，但在可能时，应尽量避免在 playbook 中使用它们因为它们可以取用任意命令，因此使用这些模块时很容易写出非幂等的 playbook。

例如，以下使用 **shell** 模块的任务为非幂等。每次运行 play 时，它都会重写 **/etc/resolv.conf**，即使它已经包含了行 **nameserver 192.0.2.1**。

```
- name: Non-idempotent approach with shell module
  shell: echo "nameserver 192.0.2.1" > /etc/resolv.conf
```

可以通过多种方式编写以幂等方式使用 **shell** 模块的任务，而且有时候进行这些更改并使用 **shell** 是最佳的做法。但更快的方案或许是使用 **ansible-doc** 发现 **copy** 模块，再使用它获得所需的效果。

在以下示例中，如果 **/etc/resolv.conf** 文件已包含正确的内容，则不会重写该文件：

```
- name: Idempotent approach with copy module
  copy:
    dest: /etc/resolv.conf
    content: "nameserver 192.0.2.1\n"
```

copy 模块可以测试来了解是否达到了需要的状态，如果已达到，则不进行任何更改。**shell** 模块容许非常大的灵活性，但需要格外小心，从而确保它以幂等方式运行。

幂等的 playbook 可以重复运行，确保系统处于特定的状态，而不会破坏状态已经正确的系统。

PLAYBOOK 语法变化

本章的最后一个部分将探讨您可能遇到的 YAML 或 Ansible Playbook 语法的一些变化。

YAML 注释

注释也可以用于提高可读性。在 YAML 中，编号或井号符号 (#) 右侧的所有内容都是注释。如果注释的左侧有内容，请在该编号符号的前面加一个空格。

```
# This is a YAML comment

some data # This is also a YAML comment
```

YAML 字符串

YAML 中的字符串通常不需要放在引号里，即使字符串中包含空格。字符串可以用双引号或单引号括起。

```
this is a string  
  
'this is another string'  
  
"this is yet another a string"
```

编写多行字符串有两种方式。可以使用竖线 (|) 字符表示要保留字符串中的换行字符。

```
include_newlines: |  
    Example Company  
    123 Main Street  
    Atlanta, GA 30303
```

要编写多行字符串，您也可以使用大于号 (>) 字符来表示换行字符转换成空格并且行内的引导空白将被删除。这种方法通常用于将很长的字符串在空格字符处断行，使它们跨占多行来提高可读性。

```
fold_newlines: >  
    This is an example  
    of a long string,  
    that will become  
    a single sentence once folded.
```

YAML 字典

您已看到过以缩进块的形式编写的键值对集合，如下方所示：

```
name: svcrole  
svcservice: httpd  
svcport: 80
```

字典也可以使用以花括号括起的内联块格式编写，如下方所示：

```
{name: svcrole, svcservice: httpd, svcport: 80}
```

大多数情形中应避免内联块格式，因为其可读性较差。不过，至少有一种情形中会较常使用它。角色的使用将在本课程后面进行讨论。当 playbook 中包含角色列表时，较常使用这种语法，从而更容易区分 play 中包含的角色和传递给角色的变量。

YAML 列表

您也已见到过使用普通单破折号语法编写的列表：

```
hosts:  
- servera  
- serverb  
- serverc
```

列表也有以方括号括起的内联格式，如下方所示：

```
hosts: [servera, serverb, serverc]
```

您应该避免使用此语法，因为它通常更难阅读。

过时的“键=值” Playbook 简写

某些 playbook 可能使用较旧的简写方法，通过将模块的键值对放在与模块名称相同的行上来定义任务。例如，您可能会看到这种语法：

```
tasks:  
  - name: shorthand form  
    service: name=httpd enabled=true state=started
```

通常，您会将相同的任务编写为如下所示：

```
tasks:  
  - name: normal form  
    service:  
      name: httpd  
      enabled: true  
      state: started
```

通常您应避免简写形式，而使用普通形式。

普通形式的行数较多，但更容易操作。任务的关键字垂直堆叠，更容易区分。阅读 play 时，您的眼睛直接向下扫视，左右运动较少。而且，普通语法是原生的 YAML；简写形式不是。如果您使用普通格式而非简写格式，现代文本编辑器中的语法突出显示工具可以更有效地为您提供帮助。

您可能会在文档和他人提供的旧 playbook 中看到这种语法，而且这种语法仍然可以发挥作用。



参考文献

ansible-playbook(1) 和 ansible-doc(1) 帮助手册

Playbook 简介 — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

Playbook — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

开发模块 — Ansible 文档

https://docs.ansible.com/ansible/latest/dev_guide/developing_modules.html

模块支持 — Ansible 文档

https://docs.ansible.com/ansible/latest/user_guide/modules_support.html

YAML 语法 — Ansible 文档

https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

► 指导练习

实施多个 PLAY

在本练习中，您将创建包含多个 play 的 playbook，再用它来在受管主机上执行配置任务。

成果

您将能够构建和执行 playbook，以管理配置并对受管主机进行管理。

在你开始之前

以 student 用户身份并使用 student 作为密码登录 workstation。

在 workstation 上，运行 **lab playbook-multi start** 命令。此功能将确保受管主机 servera.lab.example.com 可在网络上访问。它也将确保控制节点上安装了正确的 Ansible 配置文件和清单文件。

```
[student@workstation ~]$ lab playbook-multi start
```

- 1. workstation 上已经为 Ansible 项目创建工作目录 **/home/student/playbook-multi**。该目录已填充了 **ansible.cfg** 配置文件和 **inventory** 清单文件。此清单文件中已定义了受管主机 **servera.lab.example.com**。创建新 playbook **/home/student/playbook-multi/intranet.yml**，再添加开始第一个 play 所需的行。它应当以受管主机 **servera.lab.example.com** 为目标，并启用特权升级。

1.1. 将目录更改到 **/home/student/playbook-multi** 工作目录。

```
[student@workstation ~]$ cd ~/playbook-multi
[student@workstation playbook-multi]$
```

1.2. 新建 playbook **/home/student/playbook-multi/intranet.yml** 并将它打开，再将由三个破折号组成的一行添加到文件的开头，以注明这是 YAML 文件的开头。

1.3. 将下面这行添加到 **/home/student/playbook-multi/intranet.yml** 文件中，使用名称 **Enable intranet services** 注明 play 的开头。

```
- name: Enable intranet services
```

1.4. 添加下面这行以注明该 play 适用于 **servera.lab.example.com** 受管主机。务必使用两个空格缩进这一行（与上方的 **name** 关键字对齐），以注明这属于第一个 play。

```
hosts: servera.lab.example.com
```