

```

        ],
    },
    "webservers": {
        "hosts": [
            "web1.lab.example.com",
            "web2.lab.example.com"
        ]
    }
}

```

如果要自己编写动态清单脚本，可以通过“Ansible 开发人员指南”中的部署动态清单来源 [[http://docs.ansible.com/ansible/dev\\_guide/developing\\_inventory.html](http://docs.ansible.com/ansible/dev_guide/developing_inventory.html)] 获得更详细的信息。以下是一个简略概要。

脚本以适当的解释器行（例如，**`#!/usr/bin/python`**）开头并且可以执行，以便 Ansible 能够运行它。

在传递 **--list** 选项时，脚本必须显示清单中所有主机和组的 JSON 编码散列/字典。

在最简单的形式中，组可以是一个受管主机列表。在这个清单脚本的 JSON 编码输出示例中，**webservers** 是一个主机组，该组内含 **web1.lab.example.com** 和 **web2.lab.example.com** 受管主机。**databases** 主机组的成员有 **db1.lab.example.com** 和 **db2.lab.example.com** 主机。

```
[student@workstation ~]$ ./inventoryscript --list
{
    "webservers" : [ "web1.lab.example.com", "web2.lab.example.com" ],
    "databases" : [ "db1.lab.example.com", "db2.lab.example.com" ]
}
```

此外，每个组的值可以是 JSON 散列/字典，含有由每一受管主机、任何子组和可能设置的任何组变量组成的列表。下一示例显示了一个比较复杂的动态清单的 JSON 编码输出。**boston** 组具有两个子组（**backup** 和 **ipa**）、自己的三个受管主机，以及一个组变量集合（**example\_host: false**）。

```
{
    "webservers" : [
        "web1.demo.example.com",
        "web2.demo.example.com"
    ],
    "boston" : {
        "children" : [
            "backup",
            "ipa"
        ],
        "vars" : {
            "example_host" : false
        },
        "hosts" : [
            "server1.demo.example.com",
            "server2.demo.example.com",
            "server3.demo.example.com"
        ]
    },
}
```

```

"backup" : [
    "server4.demo.example.com"
],
"ipa" : [
    "server5.demo.example.com"
],
"_meta" : {
    "hostvars" : {
        "server5.demo.example.com": {
            "ntpserver": "ntp.demo.example.com",
            "dnsserver": "dns.demo.example.com"
        }
    }
}
}

```

该脚本也支持 **--host managed-host** 选项。此选项必须显示由与该主机关联的变量组成的 JSON 散列/字典，或者空白的 JSON 散列/字典。

```
[student@workstation ~]$ ./inventoryscript --host server5.demo.example.com
{
    "ntpserver" : "ntp.demo.example.com",
    "dnsserver" : "dns.demo.example.com"
}
```



### 注意

通过 **--host hostname** 选项调用时，该脚本必须显示指定主机的变量的 JSON 散列/字典。如果不提供任何变量，则可能显示空白的 JSON 散列或字典。

另外，如果 **--list** 选项返回名为 **\_meta** 的顶级元素，则可以在一次脚本调用中返回所有主机变量，从而提升脚本性能。这时，不会进行 **--host** 调用。

有关详细信息，请参见 部署动态清单来源 [[http://docs.ansible.com/ansible/developing\\_inventory.html](http://docs.ansible.com/ansible/developing_inventory.html)]。

## 管理多个清单

Ansible 支持在同一运行中使用多个清单。如果清单的位置是一个目录（不论是由 **-i** 选项设置的、是 **inventory** 参数的值，还是以某种其他方式设置的），将组合该目录中包含的所有清单文件（不论是静态还是动态）来确定清单。该目录中的可执行文件将用于检索动态清单，其他文件则被用作静态清单。

清单文件不应依赖于其他清单文件或脚本来解析。例如，如果静态清单文件指定某一个组应当是另一个组的子级，则它也需要具有该组的占位符条目，即使该组的所有成员都来自动态清单。请思考以下示例中的 **cloud-east** 组：

```
[cloud-east]  
  
[servers]  
test.demo.example.com  
  
[servers:children]  
cloud-east
```

这可以确保不论清单文件以什么顺序解析，它们在内部都一致。



### 注意

清单文件的解析顺序不是由文档指定的。目前，如果存在多个清单文件，它们会按照字母顺序进行解析。如果一个清单源依赖于其他清单来源的信息，则它们的加载顺序可能会确定清单文件是按预期工作还是引发错误。因此，务必要确保所有文件都自相一致，从而避免意外的错误。

Ansible 会忽略清单目录中以特定后缀结尾的文件。这可以通过在 Ansible 配置文件中的 **inventory\_ignore\_extensions** 指令来控制。有关更多信息，请参阅 Ansible 文档。



### 参考文献

#### 使用动态清单：Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/intro\\_dynamic\\_inventory.html](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html)

#### 开发动态清单：Ansible 文档

[https://docs.ansible.com/ansible/latest/dev\\_guide/developing\\_inventory.html](https://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html)

## ► 指导练习

# 管理动态清单

在本练习中，您将安装可动态生成清单主机列表的自定义脚本。

## 成果

您将能够安装和使用现有的动态清单脚本。

## 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 projects-inventory start** 命令。它将确保 workstation 上是否装有 Ansible，同时创建供本练习使用的 **/home/student/projects-inventory** 工作目录。

```
[student@workstation ~]$ lab projects-inventory start
```

- 1. 在 workstation 上，更改到本练习的工作目录 **/home/student/projects-inventory**。查看 **ansible.cfg** Ansible 配置文件的内容。该配置文件将清单位置设置为 **inventory**。

```
[defaults]
inventory = inventory
```

- 2. 创建 **/home/student/projects-inventory/inventory** 目录。

```
[student@workstation projects-inventory]$ mkdir inventory
```

- 3. 从 <http://materials.example.com/labs/projects-inventory/> 将 **inventorya.py**、**inventoryw.py** 和 **hosts** 文件下载到您的 **/home/student/projects-inventory/inventory** 目录。两个以 **.py** 结尾的文件都是生成动态清单的脚本，第三个文件则是静态清单。

- **inventorya.py** 脚本提供 **webservers** 组，该组含有 **servera.lab.example.com** 主机。
- **inventoryw.py** 脚本提供 **workstation.lab.example.com** 主机。
- **hosts** 静态清单文件定义 **servers** 组，它是 **webservers** 组的父组。

```
[student@workstation projects-inventory]$ wget \
> http://materials.example.com/labs/projects-inventory/inventorya.py \
> -O inventory/inventorya.py
[student@workstation projects-inventory]$ wget \
> http://materials.example.com/labs/projects-inventory/inventoryw.py \
> -O inventory/inventoryw.py
```

```
> -O inventory/inventoryw.py  
[student@workstation projects-inventory]$ wget \  
> http://materials.example.com/labs/projects-inventory/hosts \  
> -O inventory/hosts
```

- 4. 使用 **ansible** 命令并将 **inventorya.py** 脚本用作清单，列出与 **webservers** 组关联的受管主机。这将引发与 **inventorya.py** 权限相关的错误。

```
[student@workstation projects-inventory]$ ansible -i inventory/inventorya.py \  
> webservers --list-hosts  
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/  
inventorya.py with script plugin: problem running /home/student/projects-  
inventory/inventory/inventorya.py --list ([Errno 13] Permission denied)  
  
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/  
inventorya.py with ini plugin: /home/student/projects-inventory/inventory/  
inventorya.py:3: Expected key=value host variable assignment, got: subprocess  
  
[WARNING]: Unable to parse /home/student/projects-inventory/inventory/  
inventorya.py as an inventory source  
  
[WARNING]: No inventory was parsed, only implicit localhost is available  
  
[WARNING]: provided hosts list is empty, only localhost is available. Note that  
the implicit localhost does not match 'all'  
  
[WARNING]: Could not match supplied host pattern, ignoring: webservers  
  
hosts (0):
```

- 5. 检查 **inventorya.py** 脚本的当前权限，将它们更改为 755。

```
[student@workstation projects-inventory]$ ls -la inventory/inventorya.py  
-rw-rw-r--. 1 student student 639 Apr 29 14:20 inventory/inventorya.py  
[student@workstation projects-inventory]$ chmod 755 inventory/inventorya.py
```

- 6. 将 **inventoryw.py** 脚本的权限更改为 755。

```
[student@workstation projects-inventory]$ chmod 755 inventory/inventoryw.py
```

- 7. 利用 **--list** 参数，检查 **inventorya.py** 脚本的当前输出。将显示与 **webservers** 组关联的主机。

```
[student@workstation projects-inventory]$ inventory/inventorya.py --list  
{"webservers": {"hosts": ["servera.lab.example.com"], "vars": { }}}
```

- 8. 利用 `--list` 参数，检查 `inventoryw.py` 脚本的当前输出。将显示 `workstation.lab.example.com` 主机。

```
[student@workstation projects-inventory]$ inventory/inventoryw.py --list
{"all": {"hosts": ["workstation.lab.example.com"], "vars": {}}}
```

- 9. 检查 `/home/student/projects-inventory/inventory/hosts` 文件中的 `servers` 组定义。动态清单中的 `webservers` 组配置为 `servers` 组的子级。

```
[student@workstation projects-inventory]$ cat inventory/hosts
[servers:children]
webservers
```

- 10. 运行下列命令，验证 `webservers` 组中的主机列表。这将引发关于 `webservers` 组尚未定义的错误。

```
[student@workstation projects-inventory]$ ansible webservers --list-hosts
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/hosts
with yaml plugin: Syntax Error while loading YAML. found unexpected ':'
The error appears to have been in '/home/student/projects-inventory/inventory/hosts':
line 1, column 9, but may be elsewhere in the file depending on the exact syntax
problem. The offending line appears to be: [servers:children] ^ here

[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/hosts
with ini plugin: /home/student/projects-inventory/inventory/hosts:2: Section
[servers:children] includes undefined group: webservers

[WARNING]: Unable to parse /home/student/projects-inventory/inventory/hosts as an
inventory source

hosts (1):
servera.lab.example.com
```

- 11. 为避免此问题，静态清单中必须含有一个定义空的 `webservers` 主机组的占位符条目。务必要使静态清单定义所引用的任何主机组，因为它可能会从外部来源动态消失，从而导致此错误。

编辑 `/home/student/projects-inventory/inventory/hosts` 文件，使它包含以下内容：

```
[webservers]
[servers:children]
webservers
```



### 重要

如果提供主机组的动态清单脚本的命名方式使得它排在引用它的静态脚本的前面，则您将不会看到此错误。不过，如果主机组从动态清单中消失，而您没有占位符，静态清单将会引用不存在的主机组，其错误就会破坏清单的解析。

- 12. 重新运行下列命令，验证 **webservers** 组中的主机列表。它应当正常运行，没有错误。

```
[student@workstation projects-inventory]$ ansible webservers --list-hosts  
hosts (1):  
    servera.lab.example.com
```

## 完成

在 **workstation** 上，运行 **lab projects-inventory finish** 命令来清理本练习。

```
[student@workstation ~]$ lab projects-inventory finish
```

本引导式练习到此结束。

# 配置并行

---

## 培训目标

学完本节后，您将能够调整 Ansible 向受管主机开放的同时连接数量，以及 Ansible 如何通过 play 任务处理受管主机组。

## 使用分叉在 ANSIBLE 中配置并行

当 Ansible 处理 playbook 时，会按顺序运行每个 play。确定 play 的主机列表之后，Ansible 将按顺序运行每个任务。通常，所有主机必须在任何主机在 play 中启动下一个任务之前成功完成任务。

理论上，Ansible 可以同时连接到 play 中的所有主机以执行每项任务。这非常适用于小型主机列表。但如果该 play 以数百台主机为目标，则可能会给控制节点带来沉重负担。

Ansible 所进行的最大同时连接数由 Ansible 配置文件中的 **forks** 参数控制。默认情况下设为 **5**，这可通过以下方式之一来验证。

```
[student@demo ~]$ grep forks ansible.cfg
forks      = 5

[student@demo ~]$ ansible-config dump |grep -i forks
DEFAULT_FORKS(default) = 5

[student@demo ~]$ ansible-config list |grep -i forks
DEFAULT_FORKS:
description: Maximum number of forks Ansible will use to execute tasks on target
- {name: ANSIBLE_FORKS}
- {key: forks, section: defaults}
name: Number of task forks
```

例如，假设 Ansible 控制节点配置了五个分叉的默认值，并且 play 具有十个受管主机。Ansible 将在前五个受管主机上执行 play 中的第一个任务，然后在其他五个受管主机上对第一个任务执行第二轮。在所有受管主机上执行第一个任务后，Ansible 将继续一次在五个主机的组中的所有受管主机上执行下一个任务。Ansible 将依次对每个任务执行此操作，直到 play 结束。

**forks** 的默认值设置得非常保守。如果您的控制节点正在管理 Linux 主机，则大多数任务将在受管主机上运行，并且控制节点的负载较少。在这种情况下，您通常可以将 **forks** 的值设置得更高，可能接近 100，然后性能就会提高。

如果您的 playbook 在控制节点上运行很多代码，则应明智地提高分叉限值。如果使用 Ansible 管理网络路由器和交换机，则大多数模块在控制节点上运行而不在网络设备上运行。由于这会增加控制节点上的负载，因此其支持分叉数量增加的能力将显著低于仅管理 Linux 主机的控制节点。

您可以从命令行覆盖 Ansible 配置文件中 **forks** 的默认设置。**ansible** 和 **ansible-playbook** 命令均提供 **-f** 或 **--forks** 选项以指定要使用的分叉数量。

## 管理滚动更新

通常，当 Ansible 运行 play 时，它会确保所有受管主机在启动任何主机进行下一个任务之前已完成每个任务。在所有受管主机完成所有任务后，将运行任何通知的处理程序。

但是，在所有主机上运行所有任务可能会导致意外行为。例如，如果 play 更新负载平衡 Web 服务器集群，则可能需要在进行更新时让每个 Web 服务器停止服务。如果所有服务器都在同一个 play 中更新，则它们可能全部同时停止服务。

避免此问题的一种方法是使用 `serial` 关键字，通过 play 批量运行主机。在下一批次启动之前，每批主机将在整个 play 中运行。

在下面的示例中，Ansible 一次在两个受管主机上执行 play，直至所有受管主机都已更新。Ansible 首先在前两个受管主机上执行 play 中的任务。如果这两个主机中的任何一个或两个都通知了处理程序，则 Ansible 将根据这两个主机的需要运行处理程序。在这两个受管主机上执行完 play 时，Ansible 会在接下来的两个受管主机上重复该过程。Ansible 继续以这种方式运行 play，直到所有受管主机都已更新。

```
---
- name: Rolling update
  hosts: webservers
  serial: 2
  tasks:
    - name: latest apache httpd package is installed
      yum:
        name: httpd
        state: latest
      notify: restart apache

  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

假设上一个示例中的 `webservers` 组包含五个 Web 服务器，它们位于负载均衡器后面。将 `serial` 参数设置为 `2` 后，play 一次将运行两台 Web 服务器。因此，五台 Web 服务器中的大多数服务器将始终可用。

相反，如果不使用 `serial` 关键字，将同时在五台 Web 服务器上执行 play 和生成的处理程序。这可能会导致服务中断，因为 Web 服务将在所有 Web 服务器上同时重新启动。



### 重要

出于某些目的，每批主机算作在主机子集上运行的完整 play。这意味着，如果整个批处理失败，play 就会失败，这将导致整个 playbook 运行失败。

在设置了 `serial: 2` 的上一个场景中，如果出现问题并且处理的前两个主机的 play 失败，则 playbook 将中止，其余三个主机将不会通过 play 运行。这是一个有用的功能，因为只有一部分服务器会不可用，使服务降级而不是中断。

`serial` 关键字也可以指定为百分比。此百分比应用于 play 中的主机总数，以确定滚动更新批处理大小。无论百分比为何，每一工序的主机数始终为 1 或以上。



### 参考文献

滚动更新批处理大小 — 委派、滚动更新和本地操作 — Ansible 文档

[http://docs.ansible.com/ansible/playbooks\\_delegation.html#rolling-update-batch-size](http://docs.ansible.com/ansible/playbooks_delegation.html#rolling-update-batch-size)

**Ansible 性能调优（兼顾乐趣与利益）**

<https://www.ansible.com/blog/ansible-performance-tuning>

## ▶ 指导练习

# 配置并行

在本练习中，您将探索不同的串行和分叉指令对 Ansible 如何处理 play 的影响。

## 成果

您将能够跨多个受管主机调整 playbook 的并行和串行执行。

## 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 projects-parallelism start** 命令。该脚本将检查 workstation 上已安装 Ansible，然后创建供实验环境使用的目录结构和关联文件。

```
[student@workstation ~]$ lab projects-parallelism start
```

- ▶ 1. 更改为 **/home/student/projects-parallelism** 目录。检查项目目录的内容以熟悉项目文件。

- 1.1. 检查 **ansible.cfg** 文件的内容。请注意，清单文件设置为 **inventory**。另请注意，**forks** 参数设置为 **4**。

```
[defaults]
inventory=inventory
remote_user=devops
forks=4
...output omitted...
```

- 1.2. 检查 **inventory** 文件的内容。请注意，它包含主机组 **webservers**，其中包含四个主机。

```
[webservers]
servera.lab.example.com
serverb.lab.example.com
serverc.lab.example.com
serverd.lab.example.com
```

- 1.3. 检查 **playbook.yml** 文件的内容。该 playbook 将在 **webservers** 主机组上执行，确保最新的 httpd 软件包已经安装，并且 **httpd** 服务已启用并已启动。

```
---
- name: Update web server
  hosts: webservers
  tasks:
```

```

- name: Latest httpd package installed
  yum:
    name: httpd
    state: latest
  notify:
    - Restart httpd

handlers:
- name: Restart httpd
  service:
    name: httpd
    enabled: yes
    state: restarted

```

- 1.4. 最后，检查 **remove\_apache.yml** 文件的内容。该 playbook 将在 **webservers** 主机组上执行，确保 **httpd** 服务已禁用并已停止，然后确保没有安装 **httpd** 软件包。

```

---
- hosts: webservers
  tasks:
    - service:
        name: httpd
        enabled: no
        state: stopped
    - yum:
        name: httpd
        state: absent

```

- 2. 执行 **playbook.yml** playbook，使用 **time** 命令确定 playbook 运行所需的时间。观察 playbook 的运行情况。注意 Ansible 如何同时在所有四台主机上执行每项任务。

```

[student@workstation projects-parallelism]$ time ansible-playbook playbook.yml
PLAY [Update apache] ****

```

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

```

...output omitted...

```

real	0m22.701s
user	0m23.275s
sys	0m2.637s

- 3. 执行 **remove\_apache.yml** playbook 以停止和禁用 **httpd** 服务并删除 **httpd** 软件包。

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- 4. 在 **ansible.cfg** 中，将 **forks** 参数的值更改为 **1**。

```
[defaults]
inventory=inventory
remote_user=devops
forks=1
...output omitted...
```

- ▶ 5. 重新执行 **playbook.yml** playbook，使用 **time** 命令确定 playbook 运行所需的时间。观察 playbook 的运行情况。请注意，这次 Ansible 将一次在一台主机上执行各项任务。另请注意减少分叉数量如何导致了 playbook 执行时间比以前更长。

```
[student@workstation projects-parallelism]$ time ansible-playbook playbook.yml

PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]
...output omitted...

real    0m37.853s
user    0m22.414s
sys     0m4.749s
```

- ▶ 6. 执行 **remove\_apache.yml** playbook 以停止和禁用 httpd 服务并删除 httpd 软件包。

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- ▶ 7. 将 **ansible.cfg** 中的 **forks** 参数值设为 **2**。

```
[defaults]
inventory=inventory
remote_user=devops
forks=2
...output omitted...
```

- ▶ 8. 向 **playbook.yml** playbook 中的 play 添加下面的 **serial** 参数，以便该 play 一次只能在两台主机上执行。Playbook 的开头应如下所示：

```
---
- name: Update web server
  hosts: webservers
  serial: 2
```

- 9. 重新执行 **playbook.yml** playbook。观察 playbook 的运行情况。注意 Ansible 如何只在两台主机上执行整个 play，再在剩余两台主机上重新执行此 play。

```
[student@workstation projects-parallelism]$ ansible-playbook playbook.yml

PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]

TASK [Latest version of apache installed] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

...output omitted...

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

TASK [Latest version of apache installed] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

...output omitted...
```

- 10. 执行 **remove\_apache.yml** playbook 以停止和禁用 httpd 服务并删除 httpd 软件包。

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- 11. 将 **ansible.cfg** 中的 **forks** 参数值重新设为 4。

```
[defaults]
inventory=inventory
remote_user=devops
forks=4
...output omitted...
```

- 12. 在 **playbook.yml** playbook 中将 **serial** 参数设置为 3。Playbook 的开头应如下所示：

```
---
- name: Update web server
  hosts: webservers
  serial: 3
```

- 13. 重新执行 **playbook.yml** playbook。观察 playbook 的运行情况。注意 Ansible 如何只在三台主机上执行整个 play，然后在剩余一台主机上重新执行此 play。

```
[student@workstation projects-parallelism]$ ansible-playbook playbook.yml

PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]
ok: [serverc.lab.example.com]

TASK [Latest version of apache installed] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverc.lab.example.com]

...output omitted...

PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]

TASK [Latest version of apache installed] ****
changed: [serverd.lab.example.com]

...output omitted...
```

## 完成

在 workstation 上，运行 **lab projects-parallelism finish** 命令来清理本练习。

```
[student@workstation ~]$ lab projects-parallelism finish
```

本引导式练习到此结束。

# 包含和导入文件

## 培训目标

学完本节后，您应能够通过无条件或基于条件测试导入或包含来自外部文件的其他 playbook 或任务来管理大型 playbook。

## 管理大型 PLAYBOOK

如果 playbook 很长或很复杂，您可以将其分成较小的文件以便于管理。您可采用模块化方式将多个 playbook 组合为一个主要 playbook，或者将文件中的任务列表插入 play。这样可以更轻松地在不同项目中重用 play 或任务序列。

## 包含或导入文件

Ansible 可以使用两种操作将内容带入 playbook。您可以包含内容，也可以导入内容。

包含内容是一个动态操作。在 playbook 运行期间，Ansible 会在内容到达时处理所包含的内容。

导入内容是一个静态操作。在运行开始之前，Ansible 在最初解析 playbook 时预处理导入的内容。

## 导入 PLAYBOOK

`import_playbook` 指令允许您将包含 play 列表的外部文件导入 playbook。换句话说，您可以把一个或者多个额外 playbook 导入到主 playbook 中。

由于导入的内容是一个完整的 playbook，因此 `import_playbook` 功能只能在 playbook 的顶层使用，不能在 play 内使用。如果导入多个 playbook，则将按顺序导入并运行它们。

导入两个额外 playbook 的主 playbook 的简单示例如下所示：

```
- name: Prepare the web server
  import_playbook: web.yml

- name: Prepare the database server
  import_playbook: db.yml
```

您还可以使用导入的 playbook 在主 playbook 中交替 play。

```
- name: Play 1
  hosts: localhost
  tasks:
    - debug:
        msg: Play 1

- name: Import Playbook
  import_playbook: play2.yml
```

在前面的例子中，**Play 1** 首先运行，然后运行从 `play2.yml` playbook 中导入的 play。

## 导入和包含任务

您可以将任务文件中的任务列表导入或包含在 play 中。任务文件是包含一个任务平面列表的文件：

```
[admin@node ~]$ cat webserver_tasks.yml
- name: Installs the httpd package
  yum:
    name: httpd
    state: latest

- name: Starts the httpd service
  service:
    name: httpd
    state: started
```

### 导入任务文件

您可以使用 `import_tasks` 功能将任务文件静态导入 playbook 内的 play 中。导入任务文件时，在解析该 playbook 时将直接插入该文件中的任务。Playbook 中的 `import_tasks` 的位置控制插入任务的位置以及运行多个导入的顺序。

```
---
- name: Install web server
  hosts: webservers
  tasks:
    - import_tasks: webserver_tasks.yml
```

导入任务文件时，在解析该 playbook 时将直接插入该文件中的任务。由于 `import_tasks` 在解析 playbook 时静态导入任务，因此对其工作方式有一些影响。

- 使用 `import_tasks` 功能时，导入时设置的 `when` 等条件语句将应用于导入的每个任务。
- 您无法将循环用于 `import_tasks` 功能。
- 如果使用变量来指定要导入的文件的名称，那么您将无法使用主机或组清单变量。

### 包含任务文件

您也可以使用 `include_tasks` 功能将任务文件动态导入 playbook 内的 play 中。

```
---
- name: Install web server
  hosts: webservers
  tasks:
    - include_tasks: webserver_tasks.yml
```

在 play 运行并且这部分 play 到达之前，`include_tasks` 功能不会处理 playbook 中的内容。Playbook 内容的处理顺序会影响包含任务功能的工作方式。

- 使用 `include_tasks` 功能时，包含时设置的 `when` 等条件语句将确定任务是否包含在 play 中。
- 如果运行 `ansible-playbook --list-tasks` 以列出 playbook 中的任务，则不会显示已包含任务文件中的任务。将显示包含任务文件的任务。相比之下，`import_tasks` 功能不会列出导入任务文件的任务，而列出已导入任务文件中的各个任务。

- 您不能使用 **ansible-playbook --start-at-task** 从已包含任务文件中的任务开始执行 playbook。
- 您不能使用 **notify** 语句触发已包含任务文件中的处理程序名称。您可以在包含整个任务文件的主 playbook 中触发处理程序，在这种情况下，已包含文件中的所有任务都将运行。

**注意**

有关在使用条件时 **import\_tasks** 和 **include\_tasks** 之间行为差异的更详细讨论，可参阅“Ansible 用户指南”中的“条件” [[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_conditionals.html#applying-when-to-roles-imports-and-includes](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#applying-when-to-roles-imports-and-includes)]。

## 任务文件的用例

请参考下面的示例，在这些情景中将任务组作为与 playbook 独立的外部文件来管理或许有所帮助：

- 如果新服务器需要全面配置，则管理员可以创建不同的任务集合，分别用于创建用户、安装软件包、配置服务、配置特权、设置对共享文件系统的访问权限、强化服务器、安装安全更新，以及安装监控代理等。每一任务集合可通过单独的自包含任务文件进行管理。
- 如果服务器由开发人员、系统管理员和数据库管理员统一管理，则每个组织可以编写自己的任务文件，再由系统经理进行审核和集成。
- 如果服务器要求特定的配置，它可以整合为按照某一条件来执行的一组任务。换句话说，仅在满足特定标准时才包含任务。
- 如果一组服务器需要运行某一项/组任务，则它/它们可以仅在属于特定主机组的服务器上运行。

## 管理任务文件

您可以创建专门用于任务文件的目录，并将所有任务文件保存在该目录中。然后您的 playbook 就可以从该目录包含或导入任务文件。这便能够构建复杂的 playbook，同时简化其结构和组件的管理。

## 为外部 PLAY 和任务定义变量

使用 Ansible 的导入和包含功能将外部文件中的 play 或任务合并到 playbook 中极大地增强了在 Ansible 环境中重用任务和 playbook 的能力。为了最大限度地提高重用可能性，这些任务和 play 文件应尽可能通用。变量可用于参数化 play 和任务元素，以扩大任务和 play 的应用范围。

例如，以下任务文件将安装 Web 服务所需的软件包，然后启用并启动必要的服务。

```
---
- name: Install the httpd package
  yum:
    name: httpd
    state: latest
- name: Start the httpd service
  service:
    name: httpd
    enabled: true
    state: started
```

如果如下例所示对软件包和服务元素进行参数化，则任务文件也可用于安装和管理其他软件及其服务，而不仅仅用于 Web 服务。

```

---
- name: Install the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: Start the {{ service }} service
  service:
    name: "{{ service }}"
    enabled: true
    state: started

```

随后，在将任务文件合并到一个 playbook 中时，定义用于执行该任务的变量，如下所示：

```

...output omitted...
tasks:
- name: Import task file and set variables
  import_tasks: task.yml
vars:
  package: httpd
  service: service

```

Ansible 使传递的变量可用于从外部文件导入的任务。

您可以使用相同的技术使 play 文件更具可重用性。将 play 文件合并到 playbook 中时，传递变量以用于执行该 play，如下所示：

```

...output omitted...
- name: Import play file and set the variable
  import_playbook: play.yml
  vars:
    package: mariadb

```



### 重要

早期版本的 Ansible 使用 `include` 功能包含 playbook 和任务文件，具体取决于上下文。出于多种原因，此功能将被弃用。

在 Ansible 2.0 之前，`include` 像静态导入一样操作。在 Ansible 2.0 中，它改为动态操作，但这造成了一些限制。在 Ansible 2.1 中，`include` 可根据任务设置进行动态或静态操作，而这令人困惑并且易于出错。在确保 `include` 在所有上下文中都能正常工作方面也存在问题。

因此，在 Ansible 2.4 中，`include` 被 `include_tasks`、`import_tasks` 和 `import_playbook` 等新的指令取代。您可能会在旧 playbook 中找到 `include` 的示例，但应避免在新 playbook 中使用它。



### 参考文献

#### 包含和导入 — Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_reuse\\_includes.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_includes.html)

#### 创建可重用的 Playbook — Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_reuse.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse.html)

#### 条件 — Ansible 文档

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html)

## ► 指导练习

# 包含和导入文件

在本练习中，您将在顶级 Ansible Playbook 中包含并导入 playbook 和任务。

## 成果

您将能够在 playbook 中包含任务和 playbook 文件。

## 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 projects-file start** 命令。该脚本创建工作目录 /home/student/projects-file 以及相关的项目文件。

```
[student@workstation ~]$ lab projects-file start
```

- 1. 以 student 用户身份，在 workstation 上更改到 /home/student/projects-file 目录。

```
[student@workstation ~]$ cd ~/projects-file
[student@workstation projects-file]$
```

- 2. 检查 tasks 子目录中的三个文件的内容。

- 2.1. 检查 **tasks/environment.yml** 文件的内容。该文件包含用于软件包安装和服务管理的任务。

```
[student@workstation projects-file]$ cat tasks/environment.yml
---
- name: Install the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: Start the {{ service }} service
  service:
    name: "{{ service }}"
    enabled: true
    state: started
```

- 2.2. 检查 **tasks/firewall.yml** 文件的内容。该文件包含用于防火墙软件安装、管理和配置的任务。

```
[student@workstation projects-file]$ cat tasks/firewall.yml
---
- name: Install the firewall
```

```

yum:
  name: "{{ firewall_pkg }}"
  state: latest

- name: Start the firewall
  service:
    name: "{{ firewall_svc }}"
    enabled: true
    state: started

- name: Open the port for {{ rule }}
  firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  loop: "{{ rule }}"

```

2.3. 检查 **tasks/placeholder.yml** 文件的内容。此文件包含用于填充占位符 Web 内容文件的任务。

```

[student@workstation projects-file]$ cat tasks/placeholder.yml
---
- name: Create placeholder file
  copy:
    content: "{{ ansible_facts['fqdn'] }} has been customized using Ansible.\n"
    dest: "{{ file }}"

```

► 3. 检查 **plays** 子目录中的 **test.yml** 文件的内容。此文件包含一个测试与 Web 服务的连接的 play。

```

---
- name: Test web service
  hosts: localhost
  become: no
  tasks:
    - name: connect to internet web server
      uri:
        url: "{{ url }}"
        status_code: 200

```

► 4. 创建名为 **playbook.yml** 的 playbook。使用名称 **Configure web server** 定义第一个 play。应对 **inventory** 文件中定义的 **servera.lab.example.com** 受管主机执行此 play。该文件的开头应如下所示：

```

---
- name: Configure web server
  hosts: servera.lab.example.com

```

► 5. 在 **playbook.yml** playbook 中，用三组任务定义任务部分。包括 **tasks/environment.yml** 任务文件中的第一组任务。定义所需的变量，用于安装 httpd 软件包以

及启用和启动 `httpd` 服务。导入 `tasks/firewall.yml` 任务文件中的第二组任务。定义所需的变量，用于安装 `firewalld` 软件包，以启用和启动 `firewalld` 服务，以及允许 `http` 连接。从 `tasks/placeholder.yml` 任务文件导入第三组任务。

- 通过将以下条目添加到 `playbook.yml` playbook，在第一个 play 中创建任务部分。

```
tasks:
```

- 使用 `include_tasks` 功能包括 `tasks/environment.yml` 中的第一组任务。将 `package` 和 `service` 变量设置为 `httpd`。将 `svc_state` 变量设置为 `started`。

```
- name: Include the environment task file and set the variables
  include_tasks: tasks/environment.yml
  vars:
    package: httpd
    service: httpd
    when: ansible_facts['os_family'] == 'RedHat'
```

- 使用 `import_tasks` 功能从 `tasks/firewall.yml` 导入第二组任务。将 `firewall_pkg` 和 `firewall_svc` 变量设置为 `firewalld`。将 `rule` 变量设置为 `httpd`。

```
- name: Import the firewall task file and set the variables
  import_tasks: tasks/firewall.yml
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    rule:
      - http
      - https
```

- 使用 `import_tasks` 功能从 `tasks/placeholder.yml` 导入最后一组任务。将 `file` 变量设置为 `/var/www/html/index.html`。

```
- name: Import the placeholder task file and set the variable
  import_tasks: tasks/placeholder.yml
  vars:
    file: /var/www/html/index.html
```

## ► 6. 使用 `plays/test.yml` playbook 的内容，添加第二个也是最后一个 play 到 `playbook.yml` playbook。

- 添加第二个 play 到 `playbook.yml` playbook 中以验证 Web 服务器安装。从 `plays/test.yml` 导入该 play。将 `url` 变量设置为 `http://servera.lab.example.com`。

```
- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

6.2. 更改完成后，您的 playbook 应如下方所示：

```
---
- name: Configure web server
  hosts: servera.lab.example.com

  tasks:
    - name: Include the environment task file and set the variables
      include_tasks: tasks/environment.yml
      vars:
        package: httpd
        service: httpd
      when: ansible_facts['os_family'] == 'RedHat'

    - name: Import the firewall task file and set the variables
      import_tasks: tasks/firewall.yml
      vars:
        firewall_pkg: firewalld
        firewall_svc: firewalld
        rule:
          - http
          - https

    - name: Import the placeholder task file and set the variable
      import_tasks: tasks/placeholder.yml
      vars:
        file: /var/www/html/index.html

    - name: Import test play file and set the variable
      import_playbook: plays/test.yml
      vars:
        url: 'http://servera.lab.example.com'
```

6.3. 保存对 **playbook.yml** playbook 的更改。

- 7. 运行该 playbook 前，通过运行 **ansible-playbook --syntax-check** 来验证其语法是否正确。如果报告错误，请更正后再继续下一步。

```
[student@workstation projects-file]$ ansible-playbook playbook.yml --syntax-check
```

```
playbook: playbook.yml
```

- 8. 执行 **playbook.yml** playbook。该 playbook 的输出显示了任务和 play 文件的导入。

```
[student@workstation projects-file]$ ansible-playbook playbook.yml
```

```
PLAY [Configure web server] ****
```

```
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
```

```
TASK [Install the httpd package] ****
changed: [servera.lab.example.com]
```

```
TASK [Start the httpd service] ****
changed: [servera.lab.example.com]

TASK [Install the firewall] ****
ok: [servera.lab.example.com]

TASK [Start the firewall] ****
ok: [servera.lab.example.com]

TASK [Open the port for http] ****
changed: [servera.lab.example.com]

TASK [Create placeholder file] ****
changed: [servera.lab.example.com]

PLAY [Test web service] ****

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

TASK [connect to internet web server] ****
ok: [localhost]

PLAY RECAP ****
localhost : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com : ok=8    changed=4    unreachable=0    failed=0
```

## 完成

在 workstation 上，运行 **lab projects-file finish** 脚本来清理本练习。

```
[student@workstation ~]$ lab projects-file finish
```

本引导式练习到此结束。