Jinja2

1. Jinja2 使用一个名为 Environment 的中心对象。这个类的实例用于存储配置、全局对象，并用于从文件系统或其它位置加载模板。配置 Jinja2 为你的应用加载文档的最简单方式看起来大概是这样：

```
from jinja2 import Environment, PackageLoader
env = Environment(loader=PackageLoader('yourapplication', 'templates'))
```

调用 get_template() 方法从这个环境中加载模板，并会返回已加载的 Template：

```
template = env.get_template('mytemplate.html')
```

用若干变量来渲染它，调用 render() 方法：

```
print template.render(the='variables', go='here')
```

- 这个是 Python 的代码。

2. Jinja2 模板的默认编码为 utf-8。
3. unicode 是一种字符集，unicode 码就是直接使用 unicode 字符集来保存数据；而 utf-8 和 gbk 等是一种 unicode 编码规则，也就是使用 unicode 字符集的变种来保存数据。
4. 分隔符有两种：{% ... %} 和 {{ ... }}。前者用于执行语句，后者用于打印变量。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
  <title>My Webpage</title>
</head>
<body>
  <ul id="navigation">
  {% for item in navigation %}
    <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
  {% endfor %}
  </ul>

  <h1>My Webpage</h1>
  {{ a_variable }}
</body>
</html>
```

5. 点（.）和下标语法[]都可以用来来访问变量的属性，两者的效果是一样的：

```
{{ foo.bar }}
{{ foo['bar'] }}
```

6. 变量可以使用管道符号（|）连接过滤器，对其进行修改。

```
{{ name|striptags|title }}
```

移除 name 中的所有 HTML 标签并且改写为标题样式的大小写格式。

7. 测试器用于测试一个变量，使用 is 关键字以及测试器名来对一个变量进行测试。例如，要得出一个值是否定义过，你可以用 name is defined ，这会根据 name 是否定义返回 true 或 false 。

测试器也可以接受参数。如果测试器只接受一个参数，可以省掉括号。例如， 下面的两个表达式做同样的事情：

```
{% if loop.index is divisibleby 3 %}
{% if loop.index is divisibleby(3) %}
```

详情见文档中的内置测试器清单。

8. JinJa 的注释语法为 {# ... #}。

```
{# note: disabled template because we no longer use this
   {% for user in users %}
      ...
   {% endfor %}
#}
```

9. 默认配置中，模板引擎不会对空白做进一步修改。使用“-”来移除换行符：

```
{% for item in seq -%}
   {{ item }}
{%- endfor %}
```

这会产出中间不带空白的所有元素。如果 seq 是 1 到 9 的数字的列表，输出会是 123456789 。

- 注意：标签和减号之间不能有空白。也就是上面例子的减号和百分号间不能有空白。

10. 以#开始的语句和{%%}形式的语句是一样的 ， 下面的两个例子是等价的：

```
<ul>
# for item in seq
   <li>{{ item }}</li>
# endfor
</ul>

<ul>
{% for item in seq %}
   <li>{{ item }}</li>
{% endfor %}
</ul>
```

11. if、for 等语句块以冒号结尾：

```
# for item in seq:
   ...
# endfor
```

12. 圆括号、花括号或方括号可以跨越多行：

```
<ul>
# for href, caption in [('index.html', 'Index'),
                        ('about.html', 'About')]:
   <li><a href="{{ href }}">{{ caption }}</a></li>
# endfor
</ul>
```

13. ##为行注释前缀，行中所有## 之后的内容（不包括换行符）会被忽略。

14. 模板继承允许你构建一个包含你站点共同元素的基本模板“骨架”，并定义子模板可以覆盖的块。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   {% block head %}
   <link rel="stylesheet" href="style.css" />
   <title>{% block title %}{% endblock %} - My Webpage</title>
   {% endblock %}
</head>
<body>
   <div id="content">{% block content %}{% endblock %}</div>
   <div id="footer">
      {% block footer %}
      &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.
      {% endblock %}
   </div>
</body>
```

子模板可以替换掉父模板 block 标签内的部分，从而继承非 block 部分内容。

15. block 头标签和尾标签之间的内容可以被覆盖。

16. 使用 extends 关键字来声明该模板继承自另一个模板：

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
```

```
  {{ super() }}
  <style type="text/css">
    .important { color: #336699; }
  </style>
{% endblock %}
{% block content %}
  <h1>Index</h1>
  <p class="important">
   Welcome on my awesome homepage.
  </p>
{% endblock %}
```

这个模板继承自 base.html。

17. extends 关键字中可以使用斜线来访问特定目录的模板：

```
{% extends "layout/default.html" %}
```

18. Jinja2 中的 self 变量和 Python 中的 self 是差不多的。

```
<title>{% block title %}{% endblock %}</title>
<h1>{{ self.title() }}</h1>
{% block body %}{% endblock %}
```

19. 可以调用 super 来渲染父级块的内容。这会返回父级块的结果：

```
{% block sidebar %}
  <h3>Table Of Contents</h3>
  ...
  {{ super() }}
{% endblock %}
```

20. Jinja2 允许定义一个块名来改善可读性：

```
{% block sidebar %}
  {% block inner_sidebar %}
    ...
  {% endblock inner_sidebar %}
{% endblock sidebar %}
```

21. 在嵌套块中，默认不允许访问块外作用域中的变量：

```
{% for item in seq %}
  <li>{% block loop_item %}{{ item }}{% endblock %}</li>
{% endfor %}
```

在这个例子中，item 变量输出为空，因为块不允许访问块外作用域的变量。

22.在嵌套块中，使用 scoped 关键字，令块可以访问块外作用域的变量：

```
{% for item in seq %}
  <li>{% block loop_item scoped %}{{ item }}{% endblock %}</li>
{% endfor %}
```

此时会输出 item。

23.转义字符（避免跨站 XXS 攻击）：

(1) 手动转义：通过用管道传递到过滤器|e 来实现。

```
{{ user.username|e }}
```

(2) 自动转义

24.控制结构清单：

(1) For：遍历序列中的每项。

```
<h1>Members</h1>
<ul>
{% for user in users %}
  <li>{{ user.username|e }}</li>
{% endfor %}
</ul>
```

在一个 for 循环块中你可以访问这些特殊的变量：

变量	描述
loop.index	当前循环迭代的次数（从 1 开始）
loop.index0	当前循环迭代的次数（从 0 开始）
loop.revindex	到循环结束需要迭代的次数（从 1 开始）
loop.revindex0	到循环结束需要迭代的次数（从 0 开始）
loop.first	如果是第一次迭代，为 True 。
loop.last	如果是最后一次迭代，为 True 。
loop.length	序列中的项目数。
loop.cycle	在一串序列间期取值的辅助函数。

(2) Jinja 中的 if 语句类似 Python 中的 if 语句。在最简单的形式中，你可以测试一个变量是否未定义，为空或 false：

```
{% if users %}
<ul>
{% for user in users %}
```

```
  <li>{{ user.username|e }}</li>
{% endfor %}
</ul>
{% endif %}
```

(3) macro 关键字用于定义宏。宏类似常规编程语言中的函数，用于把常用行为作为可重用的函数，取代手动重复的工作。

```
{% macro input(name, value='', type='text', size=20) -%}
  <input type="{{ type }}" name="{{ name }}" value="{{
    value|e }}" size="{{ size }}">
{%- endmacro %}
```

宏的调用方式像函数一样：

```
<p>{{ input('username') }}</p>
<p>{{ input('password', type='password') }}</p>
```

如果宏在不同的模板中定义，你需要首先使用 import 。

25. 与 Python 中不同，模板中的循环内不能 break 或 continue 。但你可以在迭代中过滤序列来跳过项目。下面的例子中跳过了所有隐藏的用户：

```
{% for user in users if not user.hidden %}
  <li>{{ user.username|e }}</li>
{% endfor %}
```

26. 在宏内部，你可以访问三个特殊的变量：

(1) varargs：如果有多于宏接受的参数个数的位置参数被传入，它们会作为列表的值保存在 varargs 变量上。

(2) kwargs：同 varargs ，但只针对关键字参数。所有未使用的关键字参数会存储在这个特殊变量中。

(3) caller：如果宏通过 call 标签调用，调用者会作为可调用的宏被存储在这个变量中。

27. 可以使用特殊的 call 块来调用宏，这里跟上面直接调用宏名作用应该是相同的。

```
{% macro render_dialog(title, class='dialog') -%}
  <div class="{{ class }}">
    <h2>{{ title }}</h2>
    <div class="contents">
      {{ caller() }}
    </div>
  </div>
{%- endmacro %}

{% call render_dialog('Hello World') %}
```

```
  This is a simple dialog rendered by using a macro and
  a call block.
{% endcall %}
```

28.过滤器段允许你在一块模板数据上应用常规 Jinja2 过滤器。只需要把代码用 filter 节包裹起来：

```
{% filter upper %}
  This text becomes uppercase
{% endfilter %}
```

29.为变量赋值使用 set 标签，可以为多个变量赋值：

```
{% set navigation = [('index.html', 'Index'), ('about.html', 'About')] %}
{% set key, value = call_something() %}
```

30.extends 标签用于从另一个模板继承。

31.include 语句用于包含一个模板，并在当前命名空间中返回那个文件的内容渲染结果：

```
{% include 'header.html' %}
  Body
{% include 'footer.html' %}
```

32.导入宏最简单灵活的方式是把整个模块导入为一个变量。这样你可以访问属性：

```
{% macro input(name, value='', type='text') -%}
  <input type="{{ type }}" value="{{ value|e }}" name="{{ name }}">
{%- endmacro %}

{%- macro textarea(name, value='', rows=10, cols=40) -%}
  <textarea name="{{ name }}" rows="{{ rows }}" cols="{{ cols
    }}">{{ value|e }}</textarea>
{%- endmacro %}
```

导入宏：

```
{% import 'forms.html' as forms %}
<dl>
  <dt>Username</dt>
  <dd>{{ forms.input('username') }}</dd>
  <dt>Password</dt>
  <dd>{{ forms.input('password', type='password') }}</dd>
</dl>
<p>{{ forms.textarea('comment') }}</p>
```

33.jinja 是一种 python 模板语言，使用的是 python 的控制语句。

Ansible 自动化运维：技术最佳实践

第 2 章　Ansible 安装与配置

1. ansible 是简单的自动化 IT 工具。这个工具的目标有这么几项：让我们自动化部署 APP；自动化管理配置项；自动化的持续交付；自动化的（AWS）云服务管理。所有的这几个目标本质上来说都是在一个台或者几台服务器上，执行一系列的命令而已。
2. fabric 和 ansible 有什么差别呢？简单来说 fabric 像是一个工具箱，提供了很多好用的工具，用来在 Remote 执行命令，而 Ansible 则是提供了一套简单的流程，你要按照它的流程来做，就能轻松完成任务。这就像是库和框架的关系一样。当然，它们之间也是有共同点的——都是基于 paramiko 开发的。
3. playbook 定义 Ansible 任务的脚本，可以将多个任务定义在一个脚本中，由 Ansible 自动执行，playbook 执行支持多个任务，可以由控制主机运行多个任务，同时对多台远程主机进行管理。
4. playbook 是 Ansible 的配置、部署和编排语言，可以描述一个你想要的远程系统执行策略，或一组步骤的一般过程。
5. Ansible 系统由控制主机对被管节点的操作方式可分为两类，即 ad-hoc 和 playbook：

(1) ad-hoc 模式使用单个模块，支持批量执行单条命令。ad-hoc 直接通过 ansible 命令执行操作：

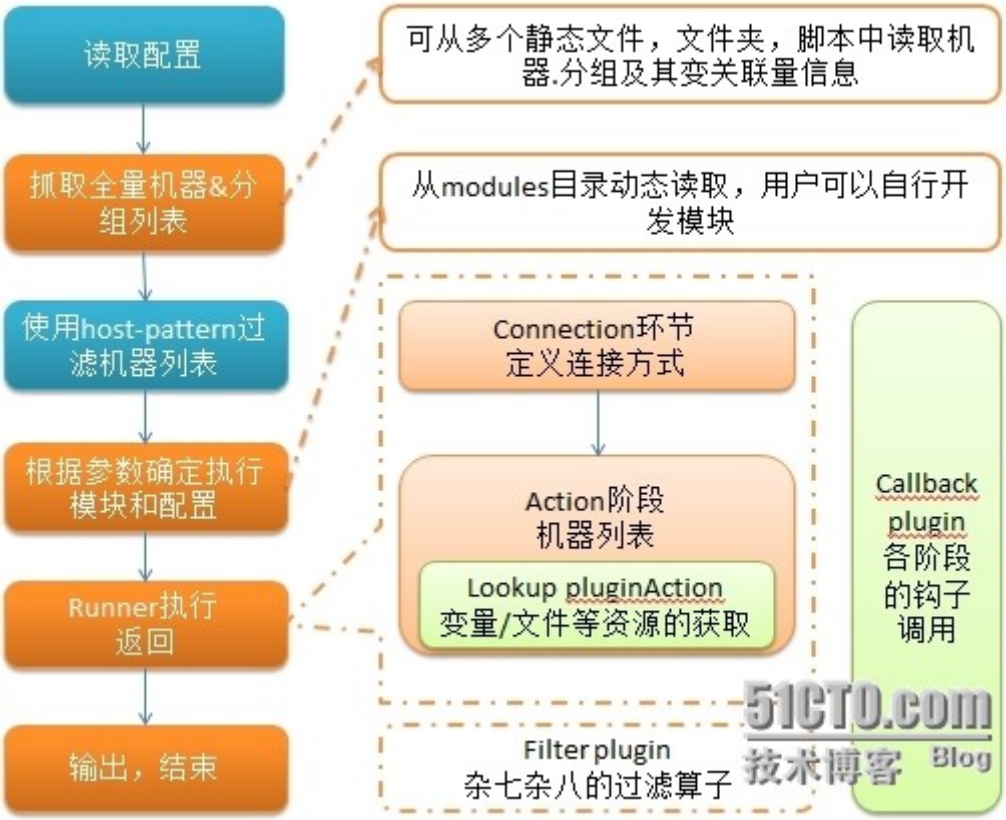
```
ansible myservers -m copy -a "src=/opt/app/bin/transfer.tar dest=~/"
```

使用 copy 命令将本地的 transfer.tar 文件复制到 myservers 组的主机中。

(2) Playbook 模式是 Ansible 主要管理方式，可以简单地把 playbook 理解为通过组合多条 ad-hoc 操作的配置文件。playbook 通过 ansible-playbook 命令执行 playbook 脚本。

```
$ ansible-playbook playbook.yml -f 10
```

-f 表示线程数。



6. 如果想通过 sudo 去执行命令，如下：

```
$ ansible atlanta -a "/usr/bin/foo" -u username --sudo [--ask-sudo-pass]
```

--sudo [--ask-sudo-pass]选项指通过密码验证的方式使用 SSH。

7. 如果被管节点上启用了 SElinux，需要安装 libselinux-python，这样才可使用 Ansible 中与 copy/file/template 相关的函数。可以通过 Ansible 的 yum 模块在需要的托管节点上安装 libselinux-python。
8. 如果你通过操作系统软件包管理工具或 pip 安装，那么你在/etc/ansible 目录下应该已经有了 ansible.cfg 配置文件；如果你是通过 GitHub 仓库安装的，在你复制的仓库中 examples 目录下可以找到 ansible.cfg，你可以把它拷贝到/etc/ansible 目录下。
9. 设置 ansible.cfg 配置参数：

- inventory——这个参数表示资源清单 inventory 文件的位置，资源清单就是一些 Ansible 需要连接管理的主机列表。

```
inventory=/etc/ansible/hosts
```

- library——Ansible 的操作动作，无论是本地或远程，都使用一小段代码来执行，这小段代码称为模块，这个 library 参数就是指向存放 Ansible 模块的目录。配置实例如下：

```
library=/usr/share/ansible
```

Ansible 支持多个目录方式，只要用冒号（：）隔开就可以，同时也会检查当前执行

playbook 位置下的./library 目录。

- forks——设置默认情况下 Ansible 最多能有多少个进程同时工作，默认设置最多 5 个进程并行处理。具体需要设置多少个，可以根据控制主机的性能和被管节点的数量来确定，可能是 50 或 100，默认值 5 是非常保守的设置。配置实例如下：

```
forks=5
```

- sudo_user——这是设置默认执行命令的用户，也可以在 playbook 中重新设置这个参数。配置实例如下：

```
sudo_user=root
```

- remote_port——这是指定连接被管节点的管理端口，默认是 22。除非设置了特殊的 SSH 端口，不然这个参数一般是不需要修改的。配置实例如下：
- host_key_checking——这是设置是否检查 SSH 主机的密钥。可以设置为 True 或 False。
- timeout——这是设置 SSH 连接的超时间隔，单位是秒。
- log_path——Ansible 系统默认是不记录日志的，如果想把 Ansible 系统的输出记录到日志文件中，需要设置 log_path 来指定一个存储 Ansible 日志的文件。

10. 为了避免 ansible 发指令时输入目标主机密码，可以通过证书签名达到 SSH 无密码访问，推荐使用 ssh-keygen 与 ssh-copy-id 来实现快速证书的生成及公钥下发，其中 ssh-keygen 生成一对密钥，使用 ssh-copy-id 来下发公钥。

(1) 在本地主机上执行 ssh-keygen -t rsa，有询问按回车，将在/root/.ssh 下生成一对密钥，其中 id_rsa 为私钥，id_rsa.pub 为公钥。

(2) 使用 ssh-copy-id 拷贝公钥到远程主机上：

```
#ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.1.21
```

(3) 密钥分发后，需要验证一下 SSH 无密码配置是否成功，只需运行 ssh root@192.168.1.111，如果直接进入被管节点 root 账号提示符，即出现[root@web1~]#，则说明配置成功。

11. /etc/ansible/hosts 文件的格式：

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
```

```
two.example.com
three.example.com
```

方括号[]中是组名，用于对系统进行分类，便于对不同系统进行个别的管理。一个系统可以属于不同的组，比如一台服务器可以同时属于 webserver 组 和 dbserver 组。这时属于两个组的变量都可以为这台主机所用。

12. 可以在 host 中指定 SSH 使用的端口：

```
badwolf.example.com:5309
```

13. 假设你有一些静态 IP 地址，希望设置一些别名，但不是在系统的 host 文件中设置，又或者你是通过隧道在连接，那么可以设置如下：

```
jumper ansible_ssh_port=5555 ansible_ssh_host=192.168.1.50
```

在这个例子中，通过 “jumper” 别名，会连接 192.168.1.50:5555。这是通过 inventory 文件的特性功能设置的变量。 一般而言，这不是设置变量的最好方式。

14. ansible 的 ping 模块用于测试主机是否能 ping 通：

```
$ ansible test -m ping
```

测试是否能 ping 通 test 主机。

15. Absible 主机与组的配置文件默认是/etc/ansible/hosts 文件，格式为 ini。如要添加两台主机 IP，同时定义两个 IP 到 webservers 组，更新内容如下：

```
[ /etc/ansible/hosts ]
#green.example.com
#blue.example.com
192 . 168 . 1.21
192 . 168 . 1.22
[webservers]
#alpha.example.org
#beta.example.org
192 . 168 . 1.21
192 . 168 . 1.22
```

然后，用 ping 模块对单台主机进行 ping 操作：

```
$ ansible 192.168.1.111 -m ping
```

测试是否能 ping 通 192.168.1.111。

```
$ ansible webservers -m ping
```

测试是否能 ping 通 webservers 组的主机。

- 注意：这里测试时在控制主机与被管节点之间配置了 SSH 证书信任。如果没有用证书认证，则需要在执行 Ansible 命令时添加-k 参数，在提示“SSH password：”时输入 root（默认）账号密码。实际生产环境中，大多数更倾向于使用 Linux 普通用户账户进行连接并通过 sudo 命令实现 root 权限。

16. 相似的主机可以使用中括号简写：

```
[webservers]
www[01:50].example.com
```

这里表示从 01 到 50，也可简写为 1:50。：

```
[databases]
db-[a:f].example.com
```

字母范围的简写模式。

17. 主机后接等号表达式，表示为特定主机定义变量，这些变量在 playbooks 中使用：

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

18. 使用[组名:vars]的形式定义属于整个组的变量：

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

19. 在 Ansible 1.9.2 版本中有 8 个主要的 Ansible 管理工具，每个管理工具都是一系列的模块、参数支持。对于 Ansible 每个工具，都可以简单地在命令后面加上-h 或—help 直接获取帮助。

20. ansible-doc 命令可以查看 module 的用法：

```
ansible-doc module_name
```

如查看 yum 模块的用法：

```
ansible-doc yum
```

第 3 章　Ansible 组件介绍

1. Ansible 中的临时命令的执行是通过 Ad-Hoc 来完成，能够快速执行，而且不需要保存执行的命令，例如：

```
ansible -i ～/hosts all -m command -a ‘who’ -u root
```

主要参数如下：

- -m：要执行的模块，默认为 command

- -a：模块的参数
- -u：ssh 连接的用户名，默认用 root，ansible.cfg 中可以配置
- -k：提示输入 ssh 登录密码。当使用密码验证的时候用
- -s：sudo 运行
- -U：sudo 到那个用户，默认为 root
- -K：提示输入 sudo 密码，当不是 NOPASSWD 模式时使用
- -f：fork 多少个进程并发处理，默认为 5 个
- -i：指定 hosts 文件路径，默认 default=/etc/ansible/hosts
- -I：指定 pattern，对<host_pattern>已匹配的主机中再过滤一次
- -T： ssh 连接超时时间，默认 10 秒
- -t：日志输出到该目录，日志文件名以主机名命名

2. 定义主机和主机组

```
172.17.42.101 ansible_ssh_pass='123456'
172.17.42.102 ansible_ssh_pass='123456'

[docker]
172.17.42.10[1:3]                    #表示目标主机为 172.17.42.101 到 172.17.42.103

[docker:vars]
ansible_ssh_pass='123456'

[ansible:children]
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。

3. inventory 的值可以指向一个目录时，表示 ansible 使用多个 Inventory 文件，所有的 Inventory 文件都在目录里。
4. 所有定义的主机与组规则都在/etc/ansible/hosts 文件中，主机可以用域名、ip、别名进行标识，其中 webservers、dbservers 为组名，紧跟着的主机为其成员：

```
mail.example.com
192.168.1.21:2135

[webservers]
foo.example.com
bar.example.com
192 . 168 . 1.22

[dbservers]
one.example.com
two.example.com
three.example.com
192 . 168 . 1.23
```

当然，也可以用别名来描述一台主机：

```
jumper ansible_ssh_port=22 ansible_ssh_host=192.168.1.50
```

jumper 是定义的别名， ansible_ssh_port 为主机 SSH 服务端口，ansible_ssh_host 为目标主机。主机变量只能在 host 文件中使用，更多保留主机变量如下：

- ansible_ssh_host：将要连接的远程主机名。与你想要设定的主机的别名不同的话，可通过此变量设置。
- ansible_ssh_port：ssh 端口号。如果不是默认的端口号，通过此变量设置。
- ansible_ssh_user：默认的 ssh 用户名。
- ansible_ssh_pass：ssh 密码，这种方式并不安全，建议使用 --ask-pass 或 SSH 密钥。
- ansible_sudo_pass：sudo 密码(这种方式并不安全,我们强烈建议使用 --ask-sudo-pass)。
- ansible_connection：与主机的连接类型。比如:local、ssh 或者 paramiko。
- ansible_ssh_private_key_file：ssh 使用的私钥文件。适用于有多个密钥，而你不想使用 SSH 代理的情况。
- ansible_shell_type：目标系统的 shell 类型。默认情况下，命令的执行使用 'sh' 语法，可设置为 'csh' 或 'fish'。
- ansible_python_interpreter：目标主机的 python 路径。

5. 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。
6. 使用 setup 模块查机器的所有 facts 信息。

```
[root@LVS_Master playbook]# ansible 192.168.170.129 -m setup
……
```

- facts 组件是 Ansible 用于采集被管机器设备信息的一个功能。

7. 使用 setup 模块的 filter 来查看指定信息：

```
[root@LVS_Master playbook]# ansible 192.168.170.129 -m setup -a
'filter=ansible_all_ipv4_addresses'
```

8. ansible 会通过 module setup 来收集主机的系统信息，这些收集到的系统信息叫做 facts，这些 facts 信息可以直接以变量的形式使用。在命令行上通过调用 setup module 命令可以查看哪些 facts 变量可以被引用：

```
# ansible all -m setup -u root
```

facts 变量可以直接在 playbook 上使用：

```
---
  - hosts: all
      user: root
      tasks:
        - name: echo system
          shell: echo {{ ansible_os_family }}
        - name install ntp on Debian linux
          apt: name=git state=installed
          when: ansible_os_family == "Debian"
        - name install ntp on redhat linux
          yum: name=git state=present
          when: ansible_os_family == "RedHat"
```

注意：变量在 when 语句中不需要加“{{”。

9. 可以通过下面的两种方式访问类似下列多层级的 facts 变量或在数组使用散列表的 YML 变量：

```
...
    "ansible_ens3": {
        "active": true,
        "device": "ens3",
        "ipv4": {
            "address": "10.66.192.234",
            "netmask": "255.255.254.0",
            "network": "10.66.192.0"
        },
        "ipv6": [
            {
                "address": "2620:52:0:42c0:5054:ff:fef2:e2a3",
                "prefix": "64",
                "scope": "global"
            },
```

```
            {
                "address": "fe80::5054:ff:fef2:e2a3",
                "prefix": "64",
                "scope": "link"
            }
        ],
        "macaddress": "52:54:00:f2:e2:a3",
        "module": "8139cp",
        "mtu": 1500,
        "promisc": false,
        "type": "ether"
    },
...
```

- 中括号：

```
{{ ansible_ens3["ipv4"]["address"] }}
```

- 点号：

```
{{ ansible_ens3.ipv4.address }}
```

10. Ansible 提供了一个在线 playbook 分享平台，地址：https://galaxy.ansible.com/。

YAML 语言

1. 在 YAML 中，字符串不一定要用双引号标示。
2. YAML 使用缩进来判断配置的层次，所以相同层次结构的元素左侧要对齐，且不能使用 TAB 字符缩进。
3. YAML 提供缩进以及内置两种格式来表示数组和散列表:

(1) 数组：习惯上数组比较常用区块格式表示，也就是用“-”+空白字符作为起始。

```
--- # 最喜爱的电影
- Casablanca
- North by Northwest
- Notorious
```

(2) 内置格式可以选择用方括号围住，并用“，”+空白区隔

```
--- # 购物清单
[milk, pumpkin pie, eggs, juice]
```

4. 散列表：散列表类似字典，都属性键值对性质。键值和数据由冒号及空白字符分开。区块形式使用缩进和换行符分隔 key: value 对。内置形式在大括号中使用逗号分隔 key: value 对。

```
--- # 区块形式
 name: John Smith
 age: 33
--- # 內置形式
{name: John Smith, age: 33}
```

5. 字符串不需要包在引号之内。
6. 有两种方法书写多行文字，一种使用|字符，另一种使用>字符：

- 使用|字符

```
data: |                    # 译者注：這是一首著名的五行民谣
 There once was a man from Darjeeling
 Who got on a bus bound for Ealing
    It said on the door
    "Please don't spit on the floor"
 So he carefully spat on the ceiling
```

- 使用>字符

```
data: >
 Wrapped text
 will be folded
 into a single
 paragraph

 Blank lines denote
 paragraph breaks
```

7. 散列表有两种形式：

- 普通形式：

```
name: Mary Smith
age: 27
```

- Json 形式：

```
{name: John Smith, age: 33}
```

8. 数组也有两种形式：

- 普通形式：

```
- a
- b
- 12
```

- Json 形式：

```
[a,b,c]
```

9.

第 4 章　playbook 详解

1. playbook 的格式大概如下：

```
---
- hosts: all
    tasks:
        - name: Install Nginx Package
          yum: name=nginx state=present

        - name: Copy Nginx.conf
          template: src=./nginx.conf.j2 dest=/etc/nginx/nginx.conf owner=root
group=root mode=0644 validate='nginx -t -c %s'
          notify:
            - Restart Nginx Service
    handlers:
        - name: Restart Nginx Service
          service: name=nginx state=restarted
```

- 第一行表示该文件是 yaml 文件，非必须，建议写上。
- "hosts:all"：定义该 playbook 针对的目标主机，all 表示针对所有主机。
- task：定义该 playbook 所有的 tasks 集合。
- "name: Install Nginx Package"：定义一个 task 的名称，建议根据 task 实际任务命名
- "yum: name=nginx state=present"： 定义一个 task 的实际动作，这里使用 yum 模块，实现 nginx 软件包的安装。
- 第 6 行-第 9 行使用 template 模块去管理/etc/nginx/nginx.conf 文件，owner、group 定义该文件的属主及属组，使用 validate 参数指文件生成后使用

nginx -t -c 检测配置文件语法，notify 是触发 handlers，如果同步后，文件 md5 值有变化的话会触发 handler。

- 第 10-12 行定一个一个 handler 状态让 Nginx 去重启。

2. playbook 由一个或多个 ‘plays’ 组成。一个 plays 对应一组 hosts，一组 hosts 下可以有多个 task。对初学者，这里有一个 playbook，其中仅包含一个 play：

```
---
- hosts: webservers
    vars:
        http_port: 80
        max_clients: 200
    remote_user: root

    tasks:
        - name: ensure apache is at the latest version
          yum: pkg=httpd state=latest
        - name: write the apache config file
           template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
        - restart apache
        - name: ensure apache is running
          service: name=httpd state=started

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

host 和 task 的 name 属性前要加“-”。

- template 模块和 copy 模块类似，都是将一个文件复制到远程地址，不同的是 copy 是直接复制，而 template 模块是先渲染模板文件，再复制到远程地址。

3. hosts 用于指定目标主机，remote_user 表示使用哪个身份去执行目标操作。

```
---
- hosts: webservers
    remote_user: root
```

remote_user 就是账户名。

4. 也可以在每一个 task 中定义自己的远程用户，它会覆盖掉上一级的 remote_user：

```
---
- hosts: webservers
    remote_user: root
    tasks:
        - name: test connection
```

```
        ping:
        remote_user: yourname
```

5. sudo 关键字用于表示执行 task 时是否需要 sudo 到 root 用户：

```
---
- hosts: webservers
    remote_user: yourname
    sudo: yes
```

同样的，也可以仅在一个 task 中，使用 sudo 执行命令，而不是在整个 play 中使用 sudo：

```
---
- hosts: webservers
    remote_user: yourname
    tasks:
        - service: name=nginx state=started
          sudo: yes
```

6. sudo_user 用于 sudo 到不同的身份，而不是使用 root：

```
---
- hosts: webservers
    remote_user: yourname
    sudo: yes
    sudo_user: postgres
```

如果你需要在使用 sudo 时指定密码，可在运行 ansible-playbook 命令时加上-K 选项。

7. 在运行 playbook 时（从上到下执行），如果一个 host 执行 task 失败，这个 host 将会从整个 playbook 的 rotation 中移除。
8. 每一个 task 必须有一个名称 name，这样在运行 playbook 时，从其输出的任务执行信息中可以很好的辨别出是属于哪一个 task 的。
9. 使用格式“module: options”声明一个 task，其中 module 是 ansible 的模块，表示执行的命令，比如 shell 模块、yum 模块等等：

```
---
- hosts: web-servers
    tasks:
        - name: Install nginx
          yum: name=nginx state=present
        - name: Copy nginx.conf
          template: src=./nginx.conf.j2 dest=/etc/nginx/nginx.conf owner=root group=root  mode=0644 validate='nginx -t -c %s'
```

```
        notify:
              - Restart nginx
    handlers:
        - name: Restart nginx
          systemd: name=nginx state=restarted enabled=yes
```

- 实际上“module: options”中的 options 和 ad-hoc 中-a 参数指定的参数是一样的。

```
$ ansible anserver -m shell -a 'hostname'
```

这条命令和下面这条命令是一样的：

```
---
- name: shell and command test
  hosts: s.hi.com
  tasks:
  - name: shell test
    shell: hostname
```

10. 下面是一种基本的 task 的定义，service 模块使用 key=value 格式的参数，这也是大多数 module 使用的参数格式：

```
tasks:
     - name: make sure apache is running
       service: name=httpd state=running
```

- 比较特别的两个模块是 command 和 shell，它们不使用 key=value 格式的参数，而是这样：

```
tasks:
 - name: disable selinux
   command: /sbin/setenforce 0
```

使用 setenforce 命令关闭 SELinux。

11. notify 用于通知系统后面的动作要等待所有动作执行完再触发。举例来说，比如因为一个配置文件多处被改动，每次改动 apache 都需要重新启动，指定 notity 后重新启动的操作只会被执行一次。

```
---
- name: test.yml just for test
  hosts: testserver
  vars:
      region: ap-southeast-1
  tasks:
      - name: template configuration file
        template: src=template.j2 dest=/etc/foo.conf
     notify:
           - restart memcached
```

```
            - restart apache
 handlers:
     - name: restart memcached
       service: name=memcached state=restarted
     - name: restart apache
       service: name=apache state=restarted
```

注意，notify 后面的内容需要和 handlers 中的 name 属性完全相同。

12. 在 notify 中列出的操作称为 handler，也就是 notify 调用 handler 中定义的操作。

13. Handlers 也是一些 task 的列表，通过名字来引用，它们和一般的 task 并没有什么区别，它是在远端系统发生改变时执行的操作：

```
---
- hosts: web-servers
      tasks:
            - selinux: state=disabled
            - name: Add repository
              yum_repository:
              name: nginx
              baseurl: http://nginx.org/packages/centos/7/$basearch/
              gpgcheck: no
              enabled: yes
              description: nginx repo
            - name: Install nginx
              yum: name=nginx state=present
            - name: Copy nginx.conf
              template: src=./nginx.conf.j2 dest=/etc/nginx/nginx.conf owner=root
group=root  mode=0644 validate='nginx -t -c %s'
            notify:
                  - Restart nginx
      handlers:
            - name: Restart nginx
              systemd: name=nginx state=restarted enabled=yes
```

14. Handlers 最佳的应用场景是用来重启服务，或者触发系统重启操作。除此以外很少用到了。

15. 使用下列语法执行一个 playbook：

```
ansible-playbook playbook.yml -f 10
```

-f 表示并发的线程数。

16. 对名为 nginx.yaml 的 yaml 文件使--syntax-check 参数 playbook 语法检测如下所示：

```
[root@python ~]# ansible-playbook nginx.yaml --syntax-check
```

使--list-task 参数显示 nginx.yaml 所有的 task 名称：

```
[root@python ~]# ansible-playbook nginx.yaml --list-task
```

使用--list-hosts 参数显示 nginx.yaml 针对的目标主机：

```
[root@python ~]# ansible-playbook nginx.yaml --list-hosts
```

确认信息都正确后，直接使用以下命令运行下 nginx.yaml playbook：

```
[root@python ~]# ansible-playbook -i hosts nginx.yaml -f 3
```

17.组变量的作用域覆盖所有成员，通过定义一个新块，块名由组名+“:vars”组成：

```
[atlanta]
host1
host2

[atlanta:vars]
net_server=net.atlanta.example.com
proxy=proxy.atlanta.example.com
```

18.通过定义一个新块，块名由组名+“:children”组成，来支持组嵌套组：

```
[atlanta]
hostl
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh
```

19.loops 循环可以减少重复使用某个模块：

(1) 标准 loops：标准 loops 是我们在编写 playbook 过程中使用最多的一种 loops，它能直接减少编写 task 的次数

```
- name: add several users
  user: name={{item}} state=present groups=wheel
  with_items:
    - testuser1
    - testuser2
```

以上写法与下面是完全等同的:

```
- name: add user testuser1
  user: name=testuser1 state=present groups=wheel
- name: add user testuser2
  user: name=testuser2 state=present groups=wheel
```

(2) 嵌套 Loops：嵌套 Loops 也是我们编写 playbook 中比较常见的一种循环，它主要实现一对多或者多对多的合并，如下所示:

```
- name: give users access to multiple databases
  mysql_user: name={{item[0]}} priv={{item[1]}}.*:ALL append_privs=yes
password=foo
  with_nested:
      - [ 'alice', 'bob' ]
      - [ 'clientdb', 'employeedb', 'providerdb' ]
```

注意，第二个数组只使用了第一个和第二个值，第三个值并没有使用。

(3) 字典 loops：标准的 loop 最外层必须是 Python 的 list 数据类型，而字典 loops 直接支持 YAML 格式的数据变量。字典的 loops 直接支持 YAML 格式的数据变量。使用 with_dict 来循环哈希表中的元素，with_dict 是接收一个 Python 字典（经过 yaml.load 后）的格式变量。假如你有以下变量:

```
---
users:
      alice:
            name: Alice Appleworth
            telephone: 123-456-7890
      bob:
            name: Bob Bananarama
            telephone: 987-654-3210
```

想打印出每个用户的名称和电话号码，可以使用 with_dict 来循环哈希表中的元素:

```
tasks:
  - name: Print phone records
    debug: msg="User {{item.key}} is {{item.value.name}} ({{ item.value.telephone }})"
    with_dict: "{{users}}"
```

(4) 文件匹配 loop：假设我们需要对一个目录下指定格式的文件进行处理，可以直接使用 with_fileglob 循环去遍历目录下所有该格式的文件。

```
---
-
  hosts: all
  gather_facts: False
```

```
tasks:
-
  name: debug loops
  debug: "msg=\"files -----> {{ item }}\""
  with_fileglob:
  -
    /root/playbook/*.yaml
```

(5) 随机选择 loop：利用 with_random_choice 在传入的 list 中随机选择一个，与使用 python random 实现原理一样。它有时可以用作一个简化版的负载均衡器，比如作为条件判断。提供的字符串中的其中一个会被随机选中。

```
---
-
 hosts: all
 gather_facts: False
 tasks:
 -
  name: debug loops
  debug: "msg=\"name -----> {{ item }}\""
  with_random_choice:
  - "web"
  - "devstack"
  - "nginx"
```

(6) 条件判断 Loops：使用 until 关键字来实现条件 Loops。

```
---
-
 hosts: all
 gather_facts: False
 tasks:
 -
  name: debug loops
  shell: cat /root/ansible
  register: host
  until: host.stdout.startswith("cat")
  retries: 5
  delay: 5
```

这种循环由三个指令完成：

- until 是一个条件表达式，如果满足条件循环结束
- retries 是重试的次数

- delay 是延迟时间

(7) register loops 循环：register 适用于 task 直接互相传递数据的，一般我们会把 register 用在单一的 task 中进行变量临时存储，其实 register 还可以同时接受多个 task 的结果当做变量临时存储：

```
---
-
 hosts: all
 gather_facts: True
 tasks:
 -
  name: debug loops
  shell: "{{ item }}"
  with_items:
  -
    hostname
  -
    uname
  register: ret
 - name: display loops
  debug: "msg=\"{% for i in ret.results %} {{ i.stdout }} {% endfor %}\""
```

(8) 整数序列循环：with_sequence 可以以升序数字顺序生成一组序列

```
---
- hosts: all

 tasks:

  # create groups
  - group: name=evens state=present
  - group: name=odds state=present

  # create some test users
  - user: name={{ item }} state=present groups=evens
    with_sequence: start=0 end=32 format=testuser%02x

  # create a series of directories with even numbers for some reason
  - file: dest=/var/stuff/{{ item }} state=directory
    with_sequence: start=4 end=16 stride=2

  # a simpler way to use the sequence plugin
```

```
  # create 4 groups
  - group: name=group{{ item }} state=present
    with_sequence: count=4
```

(9) 循环配置文件：ini 插件可以使用正则表达式来获取一组键值对。因此,我们可以遍历该集合。假设有以下 ini 文件：

```
[section1]
value1=section1/value1
value2=section1/value2

[section2]
value1=section2/value1
value2=section2/value2
```

以下是使用 with_ini 的例子：

```
- debug: msg="{{item}}"
  with_ini: value[1-2] section=section1 file=lookup.ini re=true
```

20. Ansible 的 lookups 插件支持从外部导入数据，比如我们可以从数据库里面读取信息然后定义给一个变量。

(1) lookups file：file 是我们经常使用的一种 lookups 方式，它的原理就是使用 Python 的 codecs.open 打开文件然后把结果返回给变量，下面我们来编写一个 playbook 然后了解整个使用过程，如下所示：

```
---
- hosts: all
gather_facts: False
vars:
contents: "{{ lookup('file', '/etc/sysconfig/network') }}"     #就是这个
tasks:
- name: debug lookups
debug: msg="The contents is {% for i in contents.split("\n")}"
```

打开/etc/sysconfig/network 文件，并将结果返回给 contents 变量。

(2) lookups password：password 也是我们经常使用的一种 lookups 方式，它会对传入的内容进行加密处理，如下所示：

```
---
- hosts: all
gather_facts: False
vars:
      contents: "{{ lookup('password', 'pAssW0rd') }}"
tasks:
```

```
 - name: debug lookups
 debug: msg="The contents is {{ contents }}"
```

在当前脚本目录对密码 pAssW0rd 加密，并生成一个文件名称为：pAssW0rd，该文件保存加密后的密文。

(3) lookups pipe：pipe lookups 的实现原理很简单，如果阅读过源码的读者能发现它其实就是在控制机器上调用 subprocess.Popen 执行命令，然后将命令的结果传递给变量，如下所示：

```
---
- hosts: all
gather_facts: False
vars:
      contents: "{{ lookup('pipe', 'date +%Y-%m-%d') }}"
tasks:
 - name: debug lookups
 debug: msg="The contents is {% for i in contents.split("\n")}"
```

将 date 命令的输出赋值给 contents 变量。

(4) lookups template：template 跟 file 方式有点类似，都是读取文件，但是 template 在读取文件之前需要把 jinja 模板渲染完后再读取，下面我们指定一个 jinja 模板文件：

```
worker_processes {{ ansible_processor_cores }};
IPaddress {{ ansible_eth0.ipv4.address }}
```

然后修改 playbook 如下所示：

```
---
- hosts: all
gather_facts: True
vars:
contents: "{{ lookup('template', './lookups.j2') }}"       #就是这个
tasks:
- name: debug lookups
debug: msg="The contents is {% for i in contents.split("\n"
```

渲染并读取 ookups.j2 文件，然后将结果返回给 contents 变量。

- file、password、pipe 和 template 是 lookup 里面的关键字。

21. Jinja2 是 Ansible 默认的模板语言。Ansible 默认支持 Jinja2 语言的内置 filter。可以在其他文件中引入相应的 Jinja2 语句，并将文件名加上后缀.j2，如 nginx.conf.j2：

```
nginx.conf.j2：
#{{ ansible_managed }}
{% if ansible_os_family == 'RedHat' %}
user            nginx;
```

```
{% endif %}
{% if ansible_os_family == 'Debian' %}
user            www-data;
{% endif %}

worker_processes  {{ ansible_processor_count }};
pid        /var/run/nginx.pid;


events {
   worker_connections  {{ nginx_max_clients }};
}


http {
       ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
       include /etc/nginx/mime.types;
       default_type application/octet-stream;

       access_log {{ nginx_log_dir}}/{{ nginx_access_log_name}};
       error_log {{ nginx_log_dir}}/{{ nginx_error_log_name}};

{% for k,v in nginx_http_params.iteritems() %}
       {{ k }}  {{ v }};
{% endfor %}

       gzip on;
       gzip_disable "msie6";

       include /etc/nginx/conf.d/*.conf;
       include /etc/nginx/sites-enabled/*;
}
```

Nginx.conf.j2 文件经由模板系统转化为 nginx.conf 文件。

22. 只要是在 playbook 中可以访问的变量，都可以在 template 文件中使用。
23. 可以使用 include 语句导入其他文件中的 tasks。handlers 也是 tasks，所以也可以使用 include 语句去引用 handlers 文件。

```
---
# possibly saved as tasks/foo.yml

- name: placeholder foo
  command: /bin/foo
```

```
- name: placeholder bar
  command: /bin/bar
```

使用 include 语句包含上面文件：

```
Tasks:
  - include: tasks/foo.yml
```

文件名后接等号表达式表示给 include 传递变量。

```
tasks:
  - include: wordpress.yml wp_user=timmy
  - include: wordpress.yml wp_user=alice
  - include: wordpress.yml wp_user=bob
```

被包含的文件里使用下面语法引用传递的变量：

```
{{ wp_user }}
```

24. roles 用于层次性、结构化地组织 playbook。roles 能够根据层次型结构自动装载变量文件、tasks 以及 handlers 等。

- roles 本身需要手工创建各种目录和文件。

25. site.yml 文件是 roles 的必备文件，和 roles 目录处于同一层。site.yml 是整个 roles 的入站文件，用于调用 roles 目录下的各个角色。包含 roles 的项目的结构如下：

```
site.yml
roles/
      common/
            files/
            templates/
            tasks/
            handlers/
            vars/
            defaults/
            meta/
      webservers/
            files/
            templates/
            tasks/
            handlers/
            vars/
            defaults/
            meta/
```

common 和 webservers 为 roles 名，可以自己定义。

26. site.yml 是整个 roles 的入站文件，用于调用 roles 目录下的各个角色。以 lamp 的 roles 为例，其 site.yml 文件如下:

```
---
# This playbook deploys the whole application stack in this site.

# Apply common configuration to all hosts
- hosts: all
  roles:
  - common

# Configure and deploy database servers.
- hosts: dbservers
  user: root
  roles:
  - db

# Configure and deploy the web servers. Note that we include two roles here,
# the 'base-apache' role which simply sets up Apache, and 'web' which includes
# our example web application.
- hosts: webservers
  user: root

  roles:
  - base-apache
  - web

# Configure and deploy the load balancer(s).
- hosts: lbservers
  user: root
  roles:
  - haproxy

# Configure and deploy the Nagios monitoring node(s).
- hosts: monitoring
  user: root
  roles:
  - base-apache
  - nagios
```

其中的 command、db 和 web 等都是 roles 目录下的角色。

27. roles 内各目录中可用的文件

(1) tasks 目录：至少应该包含一个名为 main.yml 的文件，其定义了此角色的任务列表;

此文件可以使用 include 包含其它的位于此目录中的 task 文件；

(2) files 目录：存放由 copy 或 script 等模块调用的文件；

(3) templates 目录：template 模块会自动在此目录中寻找 Jinja2 模板文件；

(4) handlers 目录：此目录中应当包含一个 main.yml 文件，用于定义此角色用到的各 handler；在 handler 中使用 include 包含的其它的 handler 文件也应该位于此目录中；

(5) vars 目录：应当包含一个 main.yml 文件，用于定义此角色用到的变量；

(6) meta 目录：应当包含一个 main.yml 文件，用于定义此角色的特殊设定及其依赖关系；

(7) default 目录：为当前角色设定默认变量时使用此目录；应当包含一个 main.yml 文件。

28.roles 使用 when 语句来设置触发条件：

```
---

- hosts: webservers
  roles:
    - { role: some_role, when: "ansible_os_family == 'RedHat'" }
```

如果系统版本是 RedHat，则执行该 roles。

29.给 roles 分配指定的 tags。

```
---

- hosts: webservers
  roles:
    - { role: foo, tags: ["bar", "baz"] }
```

30.roles 目录下的 meta 目录的 main.yml 文件用于定义角色依赖。角色依赖可以自动地将其他 roles 导入现在使用的 role 中，这个文件应包含一列 roles 和为之指定的参数，下面是在 roles/myapp/meta/main.yml 文件中的示例：

```
---
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, port: 80 }
  - { role: postgres, dbname: blarg, other_parameter: 12 }
```

冒号之后的第一个参数为依赖的角色名，之后的参数是依赖角色的参数，例如“{ role: apache, port: 80 }”表示依赖 apache 应用，端口为 80 端口。

31.“角色依赖” 可以通过绝对路径指定，如同顶级角色的设置：

```
---
```

```
dependencies:
  - { role: '/path/to/common/roles/foo', x: 1 }
```

“角色依赖” 也可以通过源码控制仓库或者 tar 文件指定，使用逗号分隔：路径、一个可选的版本（tag, commit, branch 等等）、一个可选友好角色名（尝试从源码仓库名或者归档文件名中派生出角色名）：

```
---
dependencies:
  - { role: 'git+http://git.example.com/repos/role-foo,v1.1,foo' }
  - { role: '/path/to/tar/file.tgz,,friendly-name' }
```

32. register 关键字用于定义注册变量。所谓的注册变量就是将 task 的执行结果返回给一个变量，待后面的 action 使用：

```
---
- hosts: web
  tasks:
    - shell: ls
      register: result
      ignore_errors: True

    - shell: echo "{{ result.stdout }}"
      when: result.rc == 5

    - debug: msg="{{ result.stdout }}"
```

此时 result 变量就是个注册变量。注册变量经常和 debug module 一起使用，这样可以得到更多 action 的输出信息，帮助用户调试。

33. Ansible 提供 when 语句，可以控制任务的执行流程。一个很简单的 when 语句的例子：

```
  tasks:
    - name: "shutdown Debian flavored systems"
      command: /sbin/shutdown -t now
      when: ansible_os_family == "Debian"
```

当节点主机系统为 Debian 的时候，执行关机操作。

34. 在 when 语句中也可以使用管道符连接过滤器。

```
tasks:
    - command: /bin/false
      register: result
      ignore_errors: True
    - command: /bin/something
      when: result|failed
```

当变量 result 的结果为 failed 时，执行/bin/something。

35. 在 Playbook 中，gather_facts 控制是否收集远程系统的信息，如果不收集系统信息，那么上面的变量就不能在该 playbook 中使用了：

```
- hosts: whatever
  gather_facts: no
```

gather_facts: no 表示禁止 Ansible 收集 facts 信息。

36. tasks 是从上到下顺序执行，如果中间发生错误，那么整个 playbook 会中止。你可以改修文件后，再重新执行。task 的基本写法：

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=running
```

使用 service 模块启动 apache。其中第一个 name 是可选的，也可以简写成下面的例子。

```
tasks:
  - service: name=httpd state=running
```

写 name 的 task 在 playbook 执行时，会显示对应的名字，信息更友好、丰富。写 name 是个好习惯：

```
TASK: [make sure apache is running]
**********************************************************
changed: [yourhost]
```

没有写 name 的 task 在 playbook 执行时，直接显示对应的 task 语法：

```
TASK: [service name=httpd state=running] ***********************************
changed: [yourhost]
```

37. 在 task 列表中，每一个“-”就是一个 task：

```
tasks:
    - command: /bin/false
      register: result
      ignore_errors: True
    - command: /bin/something
      when: result|failed
```

这里的 task 列表共有两个 task。

38. playbook 传入参数有两种形式：

- 直接在使用“module: options”的形式：

```
tasks:
  - name: Copy ansible inventory file to client
    copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts
```

```
    owner=root group=root mode=0644
```

- 用 yml 的字典传入参数：

```
tasks:
 - name: Copy ansible inventory file to client
   copy:
     src: /etc/ansible/hosts
     dest: /etc/ansible/hosts
     owner: root
     group: root
     mode: 0644
```

39. task 中每个 action 会调用一个 module，在 module 中会去检查当前系统状态是否需要重新执行。

- 如果本次执行了，那么 action 会得到返回值 changed；
- 如果不需要执行，那么 action 得到返回值 ok。

40. 只有当 TASKS 中的 action 的执行状态是 changed 时，才会触发 notify handler 的执行。

41. handlers 是按照在 handlers 中定义个顺序执行的，而不是安装 notify 的顺序执行的。

```
handlers:
 - name: define the 1nd handler
   debug: msg="define the 1nd handler"
 - name: define the 2nd handler
   debug: msg="define the 2nd handler"
 - name: define the 3nd handler
   debug: msg="define the 3nd handler"
```

42. 在 Playbook 中使用 vars 关键字来定义变量，引用变量需要使用双大括号{{}}：

```
---
- hosts: web
  vars:
      http_port: 80
  remote_user: root
  tasks:
  - name: insert firewalld rule for httpd
    firewalld: port={{ http_port }}/tcp permanent=true state=enabled immediate=yes
```

43. 使用 vars_files 关键字把变量放在单独的文件中：

```
- hosts: web
  remote_user: root
  vars_files:
     - vars/server_vars.yml
  tasks:
  - name: insert firewalld rule for httpd
    firewalld: port={{ http_port }}/tcp permanent=true state=enabled immediate=yes
```

变量文件 vars/server_vars.yml 的内容为：

```
http_port: 80
```

44. 使用中括号或点号访问复杂变量的子属性：

```
foo:
  field1: one
  field2: two
```

访问方式如下：

```
foo['field1']
foo.field1
```

45. 有时候 YAML 和 Ansible Playbook 的变量语法在一起使用时会报错，例如下面的代码会报错：

```
- hosts: app_servers
  vars:
     app_path: {{ base_path }}/22
```

解决办法是加上引号：

```
- hosts: app_servers
  vars:
      app_path: "{{ base_path }}/22"
```

46. 使用--extra-vars 关键字来实现在命令行传递参数：

```
---
- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'

  tasks:
    - ...
```

在命令行里面传值的格式有三种：

- 普通格式：

```
ansible-playbook e33_var_in_command.yml --extra-vars "hosts=web user=root"
```

- json 格式：

```
ansible-playbook e33_var_in_command.yml --extra-vars "{'hosts':'vm-rhel7-1', 'user':'root'}"
```

- 使用文件的方式传递：

```
ansible-playbook e33_var_in_command.yml --extra-vars "@vars.json"
```

47. 使用 block 关键字可以将多个 action 组装成一块，然后根据不同条件执行一段语句：

```
tasks:
  - block:
    - yum: name={{ item }} state=installed
      with_items:
        - httpd
        - memcached

    - template: src=templates/src.j2 dest=/etc/foo.conf

    - service: name=bar state=started enabled=True

    when: ansible_distribution == 'CentOS'
    become: true
    become_user: root
```

只有当 ansible_distribution 的值为 CentOS 时，才执行上面定义的 yum、template 和 service 命令。

48. with_items 使用变量的方式有两种：

- 第一种：

```
with_items:
  - "{{ mysql_pkgs }}"
```

mysql_pkgs 是一个变量。

- 第二种：

```
with_items: redhat_pkg
```

redhat_pkg 是一个变量。

49.

第 6 章　扩展 Ansible 组件

1. callback 是 Ansible 的一个回调功能，我们可以在运行 Ansible 的时候调用这个功能。比如希望在执行 playbook 失败后发邮件，或者希望将每次执行 playbook 的结

果存到日志或者数据库中。在 callback 插件里面，我们可以很方便地拿到 Ansible 执行状态信息，然后可以定义一个 callback 动作，在 playbook 的某个运行状态下进行调用。

2. callback 插件：Ansible 本身也有日志记录功能，在 ansible.cfg 中开启 log_path 即可，但是 callback 可以记录到更加详细的信息。
3. lookup 插件：Ansible 默认的所有 lookup 插件文件在当前 Python 版本的 ansible/runner/lookup_plugins/目录下，有 Python 语言基础的读者建议好好阅读该目录下的 lookup 文件。
4. Ansible 所有的 Jinja2 filter 由 Ansible 的 filter_plugins 插件和 Jinja2 自带的 filter 组成。Jina2 所有自带的 filter 在 Jina2 当前 Python 版本的 sitepackages/jinja2/filter.py 文件内。建议熟悉 Python 语言的读者好好研究这个文件。
5. 本章所说的扩展 ansible 组件，实际上就是修改现有的组件源码来生成一个新组件，比如修改 facts、callback、lookup、jinja2 模板等。

第 7 章　用 ansible-vault 保护敏感数据

1. Ansible 提供了一个 ansible-vault 工具用于管理敏感数据的数据安全（例如账号密码）。ansible-vault 可以通过调用编辑器界面创建新的加密文件，也可以加密已经创建好了的文件。不管哪一种方式，都需要输入 vault 口令，这个口令采用 AES 加密算法加密数据。在执行 ansible 时候，通过--ask-vault-pass 选项提示输入口令，或者通过--vault-password-file 选项提供包含口令的完整路径的文件。
2. 高级加密标准（AES），在密码学中又称 Rijndael 加密法。
3. 下表列出 ansible-vault 工具的子命令。

子命令	描述
create	使用编辑器创建加密文件。这需要在运行之前先配置编辑器的环境变量
edit	用编辑器编辑一个存在的加密文件，在内存中解密，退出编辑器后又保存成加密文件
encrypt	加密一个已有的结构化数据文件
decrypt	解密文件。使用这个命令要小心，不要把解密后的文件提交到版本控制系统中
rekey	改变用于加密、解密的口令

4. ansible-vault 用 create 子命令来创建新文件：

```
$ansible-vault create aws_creds.yml
Vault password:
Confirm Vault password:
```

5. 为了更新加密的 aws_creds.yml，可以用 anisble-vault 的 edit 子命令进行修改，如下所示：

```
$ ansible-vault edit aws_creds.yml
Vault password:
```

edit 子命令将做以下操作：

1）提示输入口令。

2）使用 AES 对称密钥算法对文件进行解密。

3）打开编辑器界面，运行更新文件的内容。

4）在保存之后将文件再次加密。

6. ansible-vault 提供了 rekey 子命令修改加密的密钥。

```
$ ansible-vault rekey aws_creds.yml
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

附录 模块

1. ansible 默认提供了很多模块来供我们使用。在 Linux 中，我们可以通过 ansible-doc -l 命令查看到当前 ansible 都支持哪些模块，通过 ansible-doc -s 模块名 又可以查看该模块有哪些参数可以使用。 下面介绍比较常用的几个模块：

(1) copy 模块：用于将本地文件复制到远程主机上。

(2) file 模块：主要用于文件、文件夹、超级链接类的创立、拷贝、移动、删除操作。

(3) cron 模块：用于设置 crontab。

(4) group 模块：用于进行 group 操作。

(5) user 模块：用于进行 user 操作。

(6) yum 模块：用于安装或删除应用。

(7) service 模块：用于启动或停止应用。

(8) script 模块：用于在远程服务器上执行本地编写的脚本。

(9) shell 模块：用于执行 shell 操作。

(10) command 模块：作用跟 shell 差不多。

(11) get_url 模块：用于从 http、https 或 ftp 下载文件或软件到远程主机。

(12) template 模块：用于将 Jinjia2 模块文件渲染成指定的文件。

2. copy 模块用于将本地文件复制到远程主机上。其主要参数有：

- src：需要复制的文件，需要是绝对路径。如果拷贝的是文件夹，那么文件夹会整体

拷贝，如果结尾是”/”，那么只有文件夹内的东西被考过去

- dest：复制到远程主机哪个位置，需要绝对路径。如果 src 指向的是文件夹，这个参数也必须是指向文件夹。
- owner：设定复制后的文件的所有者。
- group： 设定复制后的文件的群组。
- mode：设置复制后的文件的权限。

```
[admin@node1 ansible]$ ansible webserver -m copy -a "src='/etc/fstab' dest='/tmp' mode='600'"
```

复制 fstab 文件到 webserver 组主机，保存到/tmp 目录下，文件权限为 600。

```
- copy: src=/srv/myfiles/foo.conf dest=/etc/foo.conf owner=foo group=foo mode=0644
```

把/srv/myfiles/foo.conf 文件拷贝到远程节点/etc/foo.conf，并且它的拥有者是 foo，拥有它的群组是 foo，权限是 0644。

3. file 模块主要用于文件、文件夹、超级链接类的创立、拷贝、移动、删除操作。该模块有以下常用参数：

- force：需要在两种情况下强制创建软链接，一种是源文件不存在但之后会建立的情况下；另一种是目标软链接已存在，需要先取消之前的软链，然后创建新的软链。
- group：定义文件/目录的属组。
- mode：定义文件/目录的权限。
- owner：定义文件/目录的属主。
- path：必选项，定义文件/目录的路径。
- recurse：递归的设置文件的属性，只对目录有效。
- src：要被链接的源文件的路径，只应用于 state=link 的情况。
- dest：被链接到的路径，只应用于 state=link 的情况。
- state：代表具体的文件操作，其值可以为 file/link/directory/hard/touch/absent，file 代表即使文件不存在，也不会被创建；link 代表最终是个软链接；directory 代表文件夹；hard 代表硬链接；touch 代表生成一个空文件；absent 代表删除。

```
$ ansible webserver -m file -a "state=touch  path='/tmp/bb.txt' owner='admin' group='admin' mode='640'" -b --ask-sudo-pass
```

创建一个文件，并指定权限位 640，属主为 admin，属组为 admin。

```
$ ansible webserver -m file -a "path=/tmp/fstab.symlink state=link src=/tmp/fstab" -b --ask-sudo-pass
```

创建软链接文件。

4. cron 模块用于设置 crontab。其常用参数有：

- minute：设置计划任务中分钟设定位的值，可选项，省略则表示该值取星号*。
- hour：设置计划任务中小时设定位的值，可选项，省略则表示该值取星号*。
- day：设置计划任务中日设定位的值，可选项，省略则表示该值取星号*。
- month：设置计划任务中月设定位的值，可选项，省略则表示该值取星号*。
- weekday：设置计划任务中周几设定位的值，可选项，省略则表示该值取星号*。
- user：设置当前计划任务属于哪个用户。
- job：指定计划的任务中需要实际执行的命令或者脚本。
- name：设置计划任务的名称。
- state：当计划任务有名称时，我们可以根据名称修改或删除对应的任务，当删除计划任务时，需要将 state 的值设置为 absent。
- disabled：当计划任务有名称时，我们可以根据名称使对应的任务失效。
- backup：如果此参数的值设置为 yes，那么当修改或者删除对应的计划任务时，会先对计划任务进行备份，然后再对计划任务进行修改或者删除。

```
$ ansible webserver -m cron -a "name='sync time from ntpserver' minute='*/5' job='/usr/sbin/ntpdate time1.aliyun.com &>/dev/null' state='present' user=admin" -b --ask-sudo-pass
```

创建一个名为 sync time from ntpserver 的任务。

```
$ ansible webserver -m cron -a "name='sync time from ntpserver' state='absent'" -b --ask-sudo-pass
```

删除名为 sync time from ntpserver 的任务。

5. group 模块用于进行 group 操作。

```
$ ansible webserver -m group -a "name=hagroup gid=1052 state=present" -b --ask-sudo-pass
```

创建名为 hagroup 的组。

6. user 模块：用于进行 user 操作。

```
$ ansible webserver -m user -a "name=hacluster01 uid=1052 shell=/sbin/nologin state=present" --ask-sudo-pass -b
```

创建名为 hacluster01 的用户，并设置 uid 和使用的 shell。

```
$ ansible webserver -m user -a "name=hacluster01 state=absent" --ask-sudo-pass -b
```

删除用户。

- "state=present"表示创建，"state=absent"表示删除。

- 如果缺少最后面的"--ask-sudo-pass -b"，会出现权限不足而无法进行操作。

7. yum 模块用于安装或删除应用。其常用参数有以下几个：

- name： 程序包名。
- state： 状态，present 和 latest 都代表安装，其中 latest 代表安装最新的软件，absent 代表删除。

```
$ ansible webserver -m yum -a "name=nginx state=latest" -b --ask-sudo-pass
```

安装最新的 nginx。

8. service 模块用于启动或停止应用。其常用参数有：

- enabled：是否开机启动，值为 yes 或 no。
- name：必选项，服务名称。
- pattern：定义一个模式，如果通过 status 指令来查看服务的状态时，没有响应，就会通过 ps 指令在进程中根据该模式进行查找，如果匹配到，则认为该服务依然在运行。
- runlevel：运行级别。
- sleep：如果执行了 restarted，在则 stop 和 start 之间沉睡几秒钟。
- state：对当前服务执行启动，停止、重启、重新加载等操作，即 started、stopped、restarted 以及 reloaded。

```
$ ansible webserver -m service -a "name=nginx state=started enabled=yes" -b --ask-sudo-pass
```

启动 nginx，并设为开机自启动。

9. script 模块：用于在远程服务器上执行本地编写的脚本。

```
$ ansible webserver -m script -a 'a.sh' -b --ask-sudo-pass
```

在远端执行 a.sh 脚本。

10. get_url 模块用于将文件或软件从 http、https 或 ftp 下载到远程主机。常用参数：

- dest：指定将文件下载到哪里，该参数必选参数。
- url：指定文件的下载地址，必选参数。
- url_username：用于 http 基本认证的用户名。
- url_password：用于 http 基本认证的密码。
- validate_certs：证书验证，设置成 `no', 不验证 SSL 证书
- owner：指定属主。
- group：指定属组。
- mode：指定权限。

```
- name: 从网络下载 foo.conf
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf mode=0440

- name: 从网络下载 file 并做简单验证
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf
force_basic_auth=yes
```

11. command 模块和 shell 模块的作用差不多，区别在于 command 模块不是调用的 shell 的指令，所以没有 bash 的环境变量，也不能使用 shell 的一些操作方式，其他和 shell 没有区别，而 shell 模块调用的/bin/sh 指令执行。

```
---
- name: shell and command test
  hosts: s.hi.com
  tasks:
  - name: shell test
    shell: hostname
  - name: command test
    command: hostname
```

12. template 模块：用于将 Jinjia2 模块文件渲染成指定的文件。

- backup：是否备份。
- src：本地 Jinjia2 模版的 template 文件位置。
- dest：存放 template 文件的路径，是远程节点上的绝对路径。
- owner：指定属主。
- group：指定属组。
- mode：指定权限。

```
- template: src=/mytemplates/foo.j2 dest=/etc/file.conf owner=bin group=wheel
mode=0644
```

把/mytemplates/foo.j2 文件经过渲染后，复制到远程节点的/etc/file.conf。

13.