

- 说明
- include
 - tasks include
 - 1. include简单示例
 - 2. 在include时引入变量
 - 3. 在include中使用tag
 - 4. 在include中使用条件判断
 - 5. 在include中使用循环
 - handlers include
 - playbook include
- include_tasks
 - 基本使用
 - 在include_tasks中使用tags
- import_tasks
- 在handlers中使用include_tasks及import_tasks
- import_playbook

说明

在前面的学习当中，我们一直使用一个playbook文件来组织所有的task任务。但是，当我们项目越来越大，task越来越多的时候，如果还将所有的task都写到一个playbook当中，可读性就会变差，这个时候我们就需要重新来组织playbook了。

我们可以将一个大的playbook拆成若干个小的playbook文件，在主配置文件中将这些零碎的小文件引入进来，而这种方式就叫做playbook的"include"。

include

playbook的include其实就是使用include关键字

tasks include

1. include简单示例

下面是两个playbook示例，分别用于安装lamp和lnmp环境：

```
# cat Lamp.yml
- hosts: test
  gather_facts: no
  tasks:
```

```
- package:
  name: mysql
  state: present
- package:
  name: php-fpm
  state: present
- package:
  name: httpd
  state: present

# cat Lnmp.yml
- hosts: test
  gather_facts: no
  tasks:
    - package:
        name: mysql
        state: present
    - package:
        name: php-fpm
        state: present
    - package:
        name: nginx
        state: present
```

在上面的示例当中，我们可以看到lamp和lnmp中mysql和php的安装都是一样的，所以我们可以将这两个任务提取出来，放到一个单独的task文件中，然后在Lnmp和Lamp中引入：

```
# cat install_mysql_php.yml
- package:
  name: mysql
  state: present
- package:
  name: php-fpm
  state: present

# cat Lamp.yml
- hosts: test
  gather_facts: no
  tasks:
    - include: install_mysql_php.yml
    - package:
        name: httpd
        state: php-fpm

# cat Lnmp.yml
- hosts: test
  gather_facts: no
  tasks:
    - include: install_mysql_php.yml
    - package:
        name: nginx
        state: php-fpm
```

2. 在include时引入变量

也可以在include的时候，传入变量：

```
# cat test_include.yml
- hosts: test
  gather_facts: no
  tasks:
    - include: wordpress.yml user=timmy
    - include: wordpress.yml user=alice
    - include: wordpress.yml user=bob

# cat wordpress.yml
- debug:
  msg: "{{ user }}"
```

通过如下方式带入变量：

```
tasks:
- { include: wordpress.yml, user: timmy, ssh_keys: [ 'keys/one.txt', 'keys/two.txt' ] }
```

再给一个例子：

```
- hosts: test
  gather_facts: no
  tasks:
    - include: in.yml
      vars:
        users:
          bob:
            gender: male
          lucy:
            gender: female

# cat in.yml
- debug:
  msg: "{{ item.key }} is {{ item.value.gender }}"
  loop: "{{users | dict2items }}"
```

3. 在include中使用tag

```
# cat test_include.yml
- hosts: test
  gather_facts: no
  tasks:
    - include: in1.yml
      tags: t1
    - include: in2.yml
      tags: t2
```

```
# cat in1.yml
- debug:
  msg: "task1 in in1.yml"
- debug:
  msg: "task2 in in1.yml"

# cat in2.yml
- debug:
  msg: "task1 in in2.yml"
- debug:
  msg: "task2 in in2.yml"
```

在上面的示例当中，两个Include分别对应两个tag，如果我们在执行test_include.yml时，指定tag为t2，那么in2.yml中的所有任务都会被执行。所以tag是针对include的所有任务生效。

4. 在include中使用条件判断

```
# cat test_include.yml

- hosts: test
gather_facts: no
tasks:
  - include: in.yml
    when: 2 > 1

# cat in.yml
- debug:
  msg: "task in in.yml"
```

5. 在include中使用循环

下面是一个简单的循环示例：

```
# cat test_include.yml

- hosts: test
gather_facts: no
tasks:
  - include: in.yml
    loop:
      - 1
      - 2

# cat in.yml
- debug:
  msg: "task1 in in.yml"
- debug:
  msg: "task2 in in.yml"
```

可以看到in.yml被循环执行了两次。

我们可以稍微修改in.yml示例如下：

```
# cat in.yml
- debug:
  msg: "{{ item }} task1 in in.yml"
- debug:
  msg: "{{ item }} task2 in in.yml"
```

再次执行playbook的结果如下：

```
[root@workstation ansible]# ansible-playbook test_include.yml

PLAY [servera]
*****
*****
TASK [include]
*****
*****
included: /etc/ansible/in.yml for servera => (item=1)
included: /etc/ansible/in.yml for servera => (item=2)

TASK [debug]
*****
*****
ok: [servera] => {
    "msg": "1 task1 in in.yml"
}

TASK [debug]
*****
*****
ok: [servera] => {
    "msg": "1 task2 in in.yml"
}

TASK [debug]
*****
*****
ok: [servera] => {
    "msg": "2 task1 in in.yml"
}

TASK [debug]
*****
*****
ok: [servera] => {
    "msg": "2 task2 in in.yml"
}

PLAY RECAP
*****
```

```
*****
servera : ok=6     changed=0    unreachable=0    failed=0
```

可以看到item的值就来自test_include中的loop循环。那么这就引出了一个问题：如果正好in.yml当中也有循环时怎么办？

```
# cat in.yml
- debug:
  msg: "{{ item }} task1 in in.yml"
  loop: ['a','b','c']
```

再次执行test_include，结果如下：

```
[root@workstation ansible]# ansible-playbook test_include.yml

PLAY [servera]
*****
*****  
  
TASK [include]
*****
*****  
  
included: /etc/ansible/in.yml for servera => (item=1)
included: /etc/ansible/in.yml for servera => (item=2)

TASK [debug]
*****
*****  
  
[WARNING]: The loop variable 'item' is already in use. You should set the `loop_var` value in the `loop_control` option for the task to something else to avoid variable collisions and unexpected behavior.

ok: [servera] => (item=a) => {
    "msg": "a task1 in in.yml"
}
ok: [servera] => (item=b) => {
    "msg": "b task1 in in.yml"
}
ok: [servera] => (item=c) => {
    "msg": "c task1 in in.yml"
}

TASK [debug]
*****
*****  
  
[WARNING]: The loop variable 'item' is already in use. You should set the `loop_var` value in the `loop_control` option for the task to something else to avoid variable collisions and unexpected behavior.

ok: [servera] => (item=a) => {
    "msg": "a task1 in in.yml"
}
```

```

ok: [servera] => (item=b) => {
    "msg": "b task1 in in.yml"
}
ok: [servera] => (item=c) => {
    "msg": "c task1 in in.yml"
}

PLAY RECAP
*****
servera : ok=4      changed=0      unreachable=0      failed=0

```

这个时候，可以看到最终item的值来自in.yml中的循环。那如果我就想要使用 test_include 中的循环的值怎么办？我们再次修改 test_include.yml 以及 in.yml 如下：

```

# cat test_include.yml
- hosts: test
gather_facts: no
tasks:
  - include: in.yml
    loop:
      - 1
      - 2
    loop_control:
      loop_var: outer_item

# cat in.yml
- debug:
    msg: "{{outer_item }} {{ item }} task1 in in.yml"
  loop: ['a','b','c']

```

再次查看结果：

```

PLAY [servera]
*****
***** TASK [include]
*****
included: /etc/ansible/in.yml for servera
included: /etc/ansible/in.yml for servera

TASK [debug]
*****
***** ok: [servera] => (item=a) => {
    "msg": "1 a task1 in in.yml"
}
ok: [servera] => (item=b) => {
    "msg": "1 b task1 in in.yml"
}

```

```

ok: [servera] => (item=c) => {
    "msg": "1 c task1 in in.yml"
}

TASK [debug]
*****
***** ok: [servera] => (item=a) => {
    "msg": "2 a task1 in in.yml"
}
ok: [servera] => (item=b) => {
    "msg": "2 b task1 in in.yml"
}
ok: [servera] => (item=c) => {
    "msg": "2 c task1 in in.yml"
}

PLAY RECAP
*****
***** servera : ok=4      changed=0      unreachable=0      failed=0

```

可以看到，`outer_item` 中的值正是外层循环中`item`的值。当出现这个双层循环时，可以在外层循环中使用`loop_var` 选项指定一个变量，这个变量用于替代外层循环中的`item`变量，以便在内层循环中获取到外层循环的`item`的值，从而避免两层循环中`item`变量名的冲突。

handlers include

handlers include与tasks include大体类似，直接给例子：

```

# handlers1.yml 内容如下:
# this might be in a file like handlers/handlers.yml
- name: restart apache
  service: name=apache state=restarted

# handlers.yml 包含handlers1.yml示例:
handlers:
  - include: handlers/handlers.yml

```

playbook include

include也可以用于将一个playbook导入到另一个playbook中：

```

- name: this is a play at the top level of a file
  hosts: all
  remote_user: root
  tasks:
    - name: say hi
      tags: foo
      shell: echo "hi..."

```

```
- include: load_balancers.yml  
- include: webservers.yml  
- include: dbservers.yml
```

include_tasks

基本使用

在前面我们详细说了include的用法，然而事实上在后续的ansible版本当中，include语法可能会被弃用。而使用一些新的关键字来代替include的原始用法，`include_tasks`就是其中之一。

我们知道include可以用于包含tasks,handlers,playbooks等，而`include_tasks`则专门用于包含tasks：

```
# cat include_tasks_ex.yml  
- hosts:  
  gather_facts: no  
  tasks:  
    - debug:  
      msg: "task1"  
    - include_tasks: in.yml  
    - debug:  
      msg: "task2"  
  
# cat in.yml  
- debug:  
  msg: "{{ item }} task1 in in.yml"  
- debug:  
  msg: "{{ item }} task2 in in.yml"
```

执行结果如下：

```
PLAY [servera]  
*****  
*****  
  
TASK [debug]  
*****  
*****  
ok: [servera] => {  
  "msg": "task1"  
}  
  
TASK [include_tasks]  
*****  
*****  
included: /etc/ansible/in.yml for servera  
  
TASK [debug]
```

```
*****
ok: [servera] => {
    "msg": "task1 in in.yml"
}

TASK [debug]
*****
ok: [servera] => {
    "msg": "task2 in in.yml"
}

TASK [debug]
*****
ok: [servera] => {
    "msg": "task2"
}

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

可以看到，当我们使用 `include_tasks` 时，`include_tasks` 本身会被当做一个task，这个task会把 `include` 的文件的路径输出在控制台中，这就是 `include_tasks` 和 `include` 之间的区别。`include` 是透明的，而 `include_tasks` 是可见的，`include_tasks` 更像是一个任务，这个任务包含了其他的一些任务。

在ansible 2.7版本当中，`include_tasks` 还加入了新的参数，下面是一个简单用法示例：

```
include_tasks:
  file: in.yml
```

当然这种使用方法与 `include_tasks: in.yml` 的效果完全相同。

在`include_tasks`中使用tags

在前面我们提到过，如果为`include`添加tags，那么tags是对`include`中所有任务生效的。也就是说，如果调用`include`对应的tag，那么`include`文件中的所有任务都会执行。

但是对 `include_tasks` 添加tags，则只会对 `include_tasks` 本身生效，`include_tasks` 中所有的任务都不生效。示例如下：

```
# cat include_tasks_ex.yml
- hosts: test
  gather_facts: no
  tasks:
    - debug:
```

```

    msg: "test task1"
- include_tasks:
  file: in.yml
  tags: t1
- debug:
  msg: "test task3"

# cat in.yml
- debug:
  msg: "test task2"

```

执行结果如下：

```

# ansible-playbook include_tasks_ex.yml --tags t1

PLAY [test]
*****
****

TASK [include_tasks]
*****
****

included: /etc/ansible/in.yml for 10.1.61.187

PLAY RECAP
*****
****

10.1.61.187 : ok=1    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

```

如果想要tags对 include_tasks 中包含的所有任务生效，则需要使用 include_tasks 模块的apply参数并配合 tags: always 内置tag：

```

- hosts: test
gather_facts: no
tasks:
- debug:
  msg: "test task1"
- include_tasks:
  file: in.yml
  apply:
    tags: t1
  tags: always
- debug:
  msg: "test task3"

```

执行结果：

```

# ansible-playbook include_tasks_ex.yml --tags t1

PLAY [test]

```

```
*****
***** TASK [include_tasks]
***** included: /etc/ansible/in.yml for 10.1.61.187
*****
***** TASK [debug]
***** ok: [10.1.61.187] => {
*****   "msg": "test task2"
***** }

***** PLAY RECAP
*****
***** 10.1.61.187 : ok=2    changed=0    unreachable=0    failed=0    skipped=0
***** rescued=0   ignored=0
```

在上一篇我们讲到tags的时候说过，如果一个任务被打上了`tags: always`标签，则即使我们调用其他任务的标签，该任务也会被执行。

需要说明的是，在这里，`tags: always`标签只针对`include_tasks`本身生效，也就是说，如果其他任务的标签被调用，`include_tasks`本身会被调用，而其包含的任务不会被调用。如果要想其包含的任务也总是被调用，可修改配置如下：

```
- hosts: test
gather_facts: no
tasks:
  - debug:
    msg: "test task1"
  - include_tasks:
    file: in.yml
    apply:
      tags: t1,always
    tags: always
  - debug:
    msg: "test task3"
```

import_tasks

`import_tasks`与`include_tasks`用法类似，都用于包含一个任务列表：

```
# cat import_tasks_ex.yml
- hosts: test
gather_facts: no
tasks
  - debug:
```

```
msg: "test task1"
- import_tasks: in.yml

# cat in.yml
- debug:
  msg: "test task2"
```

执行结果：

```
# ansible-playbook import_tasks_ex.yml

PLAY [test]
*****
***** TASK [debug]
*****
***** ok: [10.1.61.187] => {
      "msg": "test task1"
}

***** TASK [debug]
*****
***** ok: [10.1.61.187] => {
      "msg": "test task2"
}

PLAY RECAP
*****
***** 10.1.61.187 : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

可以看到，`import_tasks` 模块并不会像 `include_tasks` 模块一样，在控制台输出自身的任务信息，其相对透明。

除此之外，`import_tasks` 和 `include_tasks` 还有如下不同：

1. `import_tasks` 是静态的，被`import`的文件在playbook被加载时就预处理了，而 `include_tasks` 是动态的，被`include`的文件在playbook被运行时候才开始处理。一个简单的例子：

```
- hosts: test
gather_facts: no
vars:
  file_name: in.yml
tasks:
  - include_tasks: {{ file_name }}
  - import_tasks: {{ file_name }}
```

在上面的示例中，`include_tasks` 和 `import_tasks` 均会被执行。

再看下面的例子：

```
- hosts: test
  gather_facts: no
  tasks:
    - set_fact:
        file_name: in.yml
    - include_tasks: {{ file_name }}
    - import_tasks: {{ file_name }}
```

此时，`import_tasks` 就会出错：

```
# ansible-playbook include_import_tasks_ex.yml
ERROR! Error when evaluating variable in import path: {{ file_name }}.

When using static imports, ensure that any variables used in their names are defined
in vars/vars_files
or extra-vars passed in from the command line. Static imports cannot use variables
from facts or inventory
sources like group or host vars.
```

当使用静态的import时，请确保文件名中使用到的变量被定义在vars、vars_files或者extra-vars中，不支持其他的方式传入变量。

2. 如果想要对包含的任务列表进行循环操作，则只能使用 `include_tasks`，`import_tasks` 不支持循环操作。也就是说，使用 `loop` 或者 `with_X` 对include文件进行循环操作时，只能配合 `include_tasks` 才能正常使用
3. 当使用when对include文件添加条件判断时，`include_tasks` 和 `import_tasks` 有着本质的不同：
 - 当对 `include_tasks` 使用when时，when对应的条件只会应用于 `include_tasks` 任务本身，当执行被包含的任务时，不会对这些被包含的任务重新进行条件判断
 - 当对 `import_tasks` 使用when时，when对应的条件会被应用于被import的文件中的每一个任务，当执行被import的任务时，会对每一个被包含的任务进行同样的条件判断。

示例如下：

```
# cat include_import_tasks_ex2.yml
- hosts: test
  gather_facts: no
  tasks:
    - name: set testvar to 0
      set_fact:
        testnum: 0
    - debug:
        msg: 'include_tasks: in1.yml'
    - include_tasks: in1.yml
```

```
when: testnum == 0

- name: set testvar to 0
  set_fact:
    testnum: 0
- debug:
    msg: 'import_tasks: in1.yml'
- import_tasks: in1.yml
  when: testnum == 0
```

执行结果：

```
# ansible-playbook include_import_tasks_ex2.yml

PLAY [test]
*****
****

TASK [set testvar to 0]
*****
*
ok: [10.1.61.187]

TASK [debug]
*****
****

ok: [10.1.61.187] => {
    "msg": "include_tasks: in1.yml"
}

TASK [include_tasks]
*****
****

included: /etc/ansible/in1.yml for 10.1.61.187

TASK [set_fact]
*****
****

ok: [10.1.61.187]

TASK [debug]
*****
****

ok: [10.1.61.187] => {
    "msg": "test task2"
}

TASK [set testvar to 0]
*****
*

ok: [10.1.61.187]

TASK [debug]
*****
****

ok: [10.1.61.187] => {
```

```
"msg": "import_tasks: in1.yml"
}

TASK [set_fact]
*****
ok: [10.1.61.187]

TASK [debug]
*****
skipping: [10.1.61.187]

PLAY RECAP
*****
10.1.61.187 : ok=8    changed=0    unreachable=0    failed=0    skipped=1
rescued=0   ignored=0
```

在handlers中使用include_tasks及import_tasks

我们知道，handlers中执行的其实也是任务，只不过是被触发才会运行，所以如果要在handlers中引入任务，也可直接使用 `include_tasks` 和 `import_tasks`。没有 `include_handlers` 的说法。

import_playbook

我们在前面提到过，`include`除了可以引用任务列表，还可以引用整个playbook，在之后的版本中，如果想要引入playbook，则需要使用 `import_playbook` 模块。在2.8版本后，使用`include`引用整个playbook的特性会被弃用。

示例：

```
# cat import_playbook_ex.yml
- hosts: test
gather_facts: no
tasks:
  - debug:
    msg: "test task"
- import_playbook: inplay.yml

# cat inplay.yml
- hosts: test
gather_facts: no
tasks:
  - debug:
    msg: "test task in inplay.yml"
```

