循环语句

简介

我们在编写playbook的时候，不可避免的要执行一些重复性操作，比如指安装软件包，批量创建用户，操作某个目录下的所有文件等。正如我们所说，ansible一门简单的自动化语言，所以流程控制、循环语句这些编程语言的基本元素它同样都具备。

在Ansible 2.5以前，playbook通过不同的循环语句以实现不同的循环，这些语句使用 `with_` 作为前缀。这些语法目前仍然兼容，但在未来的某个时间点，会逐步废弃。

下面列出一些较常见的 `with_X` 循环语句：

- with_items
- with_flattened
- with_list
- with_together
- with_nested
- with_indexed_items
- with_sequence
- with_random_choice
- with_dict

- with_subelement
- with_file
- with_fileglob
- with_lines

1. with_items

简单的列表循环

场景一： 循环打印inventory中所有未分组的主机

```
- hosts: test
  gather_facts: no
  tasks:
    - debug:
        msg: "{{ item }}""
      with_items: "{{ groups.ungrouped }}"
```

场景二： 直接在with_items中定义被循环的列表

```
- hosts: test
  gather_facts: no
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_items:
        - "user0"
        - "user1"
        - "user2"
```

也可以写成如下方式：

```
- hosts: test
  gather_facts: no
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_items:["user0","user1","user2"]
```

场景三： 在with_items中定义更复杂的列表

```
- hosts: test
  gather_facts: no
  tasks:
    - name: "create directory"
```

```
      file:
        path: "/{{ item.path1 }}/{{ item.path2 }}"
      with_items:
        - { path1: a, path2: b}
        - { path1: c, path2: d}
```

2. with_list

与 `with_items` 一样，也是用于循环列表。区别是，如果列表的值也是列表，`with_items` 会将第一层嵌套的列表拉平，而 `with_list` 会将值作为一个整体返回。

示例：

```
# 使用with_items的示例
- hosts: test
  gather_facts: no
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_items:
        - [1, 2]
        - [a, b]

# 返回结果：
# ansible-playbook with_items_test.yml


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

TASK [Gathering Facts]
*********************************************************************************************
*************************************************************
ok: [10.1.61.187]

▽ASK [with_items]
*********************************************************************************************
*****************************************************************
ok: [10.1.61.187] => (item=1) => {
    "msg": 1
}
ok: [10.1.61.187] => (item=2) => {
    "msg": 2
}
ok: [10.1.61.187] => (item=[3, 4]) => {
    "msg": [
        3,
        4
    ]
}
```

```
ok: [10.1.61.187] => (item=a) => {
    "msg": "a"
}
ok: [10.1.61.187] => (item=b) => {
    "msg": "b"
}
ok: [10.1.61.187] => (item=c) => {
    "msg": "c"
}

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

# 使用with_list的示例
- hosts: test
  gather_facts: no
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_list:
        - [1, 2]
        - [a, b]

# 返回结果:

# ansible-playbook with_list_ex.yml

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

TASK [with_items]
*********************************************************************************************
*****************************************************************
ok: [10.1.61.187] => (item=[1, 2]) => {
    "msg": [
        1,
        2
    ]
}
ok: [10.1.61.187] => (item=['a', 'b']) => {
    "msg": [
        "a",
        "b"
    ]
}

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

3. with_flattened

`with_flattened` 与 `with_items` 类似，当处理复杂的多级列表嵌套时，会将所有的列表全部拉平：

```
- hosts: test
  gather_facts: no
  tasks:
    - name: "with_items"
      debug:
        msg: "{{ item }}"
      with_flattened:
        - [1, 2,[3,4]]
        - [a, b]
```

返回结果：

```
# ansible-playbook with_flattened_ex.yml

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

TASK [with_items]
*********************************************************************************************
*****************************************************************
ok: [10.1.61.187] => (item=1) => {
    "msg": 1
}
ok: [10.1.61.187] => (item=2) => {
    "msg": 2
}
ok: [10.1.61.187] => (item=3) => {
    "msg": 3
}
ok: [10.1.61.187] => (item=4) => {
    "msg": 4
}
ok: [10.1.61.187] => (item=a) => {
    "msg": "a"
}
ok: [10.1.61.187] => (item=b) => {
    "msg": "b"
}
ok: [10.1.61.187] => (item=c) => {
    "msg": "c"
}

PLAY RECAP
*********************************************************************************************
*****************************************************************
```

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

4. with_together

`with_together`可以将两个列表中的元素对齐合并

示例如下：

```
- hosts: test
  remote_user: root
  vars:
    alpha: [ 'a','b']
    numbers: [ 1,2]
  tasks:
    - debug: msg="{{ item.0 }} and {{ item.1 }}"
      with_together:
         - "{{ alpha }}"
         - "{{ numbers }}"


# 输出的结果为：
ok: [10.1.61.187] => (item=['a', 1]) => {
    "item": [
        "a",
        1
    ],
    "msg": "a and 1"
}
ok: [10.1.61.187] => (item=['b', 2]) => {
    "item": [
        "b",
        2
    ],
    "msg": "b and 2"
}
```

可以看到第一个列表中的第一个元素a与第二个列表中的第一个元素1合并输出，第一个列表中的b与第二个列表中的第二个元素2合并输出了

上面的示例是基于两个列表的元素完全相同的结果，如果两个列表中的元素不同：

```
- hosts: test
  remote_user: root
  vars:
    alpha: [ 'a','b','c']
    numbers: [ 1,2]
  tasks:
    - debug: msg="{{ item.0 }} and {{ item.1 }}"
      with_together:
```

```
          - "{{ alpha }}"
          - "{{ numbers }}"


# 输出结果:

# ansible-playbook with_together_ex.yml

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

TASK [debug]
*********************************************************************************************
*********************************************************************
ok: [10.1.61.187] => (item=['a', 1]) => {
    "msg": "a and 1"
}
ok: [10.1.61.187] => (item=['b', 2]) => {
    "msg": "b and 2"
}
ok: [10.1.61.187] => (item=['c', None]) => {
    "msg": "c and "
}

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

5. with_nested

嵌套循环

```
tasks:
  - name: debug loops
    debug: msg="name is {{ item[0] }}  vaule is {{ item[1] }} "
    with_nested:
      - ['alice','bob']
      - ['a','b','c']
```

item[0]是循环的第一个列表的值['alice','bob']。item[1]是第二个列表的值；以上的执行输出如下：

```
# ansible-playbook with_nested_ex.yml

PLAY [with_nested test]
*********************************************************************************************
*************************
```

```
TASK [debug loops]
*********************************************************************************************
*****************************
ok: [10.1.61.187] => (item=['alice', 'a']) => {
    "msg": "name is alice  vaule is a"
}
ok: [10.1.61.187] => (item=['alice', 'b']) => {
    "msg": "name is alice  vaule is b"
}
ok: [10.1.61.187] => (item=['alice', 'c']) => {
    "msg": "name is alice  vaule is c"
}
ok: [10.1.61.187] => (item=['bob', 'a']) => {
    "msg": "name is bob  vaule is a"
}
ok: [10.1.61.187] => (item=['bob', 'b']) => {
    "msg": "name is bob  vaule is b"
}
ok: [10.1.61.187] => (item=['bob', 'c']) => {
    "msg": "name is bob  vaule is c"
}

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

下面是一个稍微有用点儿的示例：

```
- hosts: test
  gather_facts: false
  tasks:
    - file:
        state: directory
        path: "/data/{{ item[0] }}/{{ item[1] }}"
      with_nested:
        - [test1,test2]
        - [a,b,c]
```

> with_cartesian与其功能完全一致

5. with_indexed_items

在循环处理列表时，为列表中的每一项添加索引（从0开始的数字索引）

简单示例：

```
- hosts: test
  gather_facts: false
  tasks:
```

```
    - debug:
        msg: "{{ item }}"
      with_indexed_items:
        - test1
        - test2
        - test3
```

执行之后，返回结果如下：

```
# ansible-playbook with_indexed_items_ex.yml

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

TASK [debug]
*****************************************************************************************
************************************
ok: [10.1.61.187] => (item=[0, 'test1']) => {
    "msg": [
        0,
        "test1"
    ]
}
ok: [10.1.61.187] => (item=[1, 'test2']) => {
    "msg": [
        1,
        "test2"
    ]
}
ok: [10.1.61.187] => (item=[2, 'test3']) => {
    "msg": [
        2,
        "test3"
    ]
}

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

所以我们可以使用with_indexed_items执行如下操作：

```
- hosts: test
  gather_facts: false
  tasks:
    - debug:
        msg: "index is {{ item[0] }}, value is {{ item[1] }}"
      with_indexed_items:
        - test1
        - test2
        - test3
```

下面再看一个稍微复杂的列表结构：

```
- hosts: test
  gather_facts: false
  tasks:
    - debug:
        msg: "index is {{ item[0] }}, value is {{ item[1] }}"
      with_indexed_items:
        - test1
        - [test2,test3]
        - [test4,test5]
```

这个时候，返回的结果如下：

```
# ansible-playbook with_indexed_items_ex2.yml

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

TASK [debug]
*****************************************************************************************
************************************
ok: [10.1.61.187] => (item=[0, 'test1']) => {
    "msg": "index is 0, value is test1"
}
ok: [10.1.61.187] => (item=[1, 'test2']) => {
    "msg": "index is 1, value is test2"
}
ok: [10.1.61.187] => (item=[2, 'test3']) => {
    "msg": "index is 2, value is test3"
}
ok: [10.1.61.187] => (item=[3, 'test4']) => {
    "msg": "index is 3, value is test4"
}
ok: [10.1.61.187] => (item=[4, 'test5']) => {
    "msg": "index is 4, value is test5"
}

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

可以看到，其在处理更复杂列表的时候，会将列表拉平，类似于 `with_items` 。

与 `with_items` 一样，其也只会拉平第一层列表，如果存在多层列表嵌套，则更深的嵌套不会被拉平：

```
- hosts: test
  gather_facts: false
  tasks:
    - debug:
        msg: "index is {{ item[0] }}, value is {{ item[1] }}"
      with_indexed_items:
        - test1
        - [test2,[test3,test4]]
```

此时的返回结果：

```
# ansible-playbook with_indexed_items_ex3.yml

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

TASK [debug]
*****************************************************************************************
************************************
ok: [10.1.61.187] => (item=[0, 'test1']) => {
    "msg": "index is 0, value is test1"
}
ok: [10.1.61.187] => (item=[1, 'test2']) => {
    "msg": "index is 1, value is test2"
}
ok: [10.1.61.187] => (item=[2, ['test3', 'test4']]) => {
    "msg": "index is 2, value is ['test3', 'test4']"
}

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

6. with_sequence

用于返回一个数字序列

参数说明：

- start：指定起始值
- end：指定结束值
- stride：指定步长，即从start至end，每次增加的值
- count：生成连续的数字序列，从1开始，到count的值结束
- format：格式化输出,类似于linux命令行中的printf格式化输出

> 关于format参数，更多的格式化输出参数可参考：http://www.zsythink.net/archives/1411

```
- hosts: test
  gather_facts: false
  tasks:
    # create groups
    - group:
        name: {{ item }}
        state: present
      with_items:
        - evens
        - odds


    # create some test users
    # [testuser00,testuser01,testuser02,...,testuser32]
    - user:
        name: {{ item }}
        state: present
        groups: evens
      with_sequence:
        start: 0
        end: 32
        stride: 4
        format=testuser%02d

    # create a series of directories with even numbers for some reason
    # [4,6,8,10,...,16]
    - 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
```

7. with_random_choice

用于从一个列表的多个值中随机返回一个值

下面的示例，一个列表当中有四个值，连续执行playbook，每次都随机返回一个：

```
- hosts: test
  gather_facts: false
  tasks:
    - debug: msg={{ item }}
      with_random_choice:
         - "go through the door"
```

```
          - "drink from the goblet"
          - "press the red button"
          - "do nothing"
```

8. with_dict

循环字典

```
- hosts: test
  gather_facts: no
  vars:
    # 假如有如下变量内容:
    users:
      alice:
        name: Alice Appleworth
        telephone: 123-456-7890
      bob:
        name: Bob Bananarama
        telephone: 987-654-3210

  # 现在需要输出每个用户的用户名和手机号:
  tasks:
    - name: Print phone records
      debug:
        msg: "User {{ item.key }} is {{ item.value.name }} ({{ item.value.telephone }})"
      with_dict: "{{ users }}"
```

输出如下:

```
# ansible-playbook  with_dict_ex.yml

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

TASK [Print phone records]
*********************************************************************************************
*********************************************************************************************
*****
ok: [10.1.61.187] => (item={'key': 'alice', 'value': {'name': 'Alice Appleworth',
'telephone': '123-456-7890'}}) => {
    "msg": "User alice is Alice Appleworth (123-456-7890)"
}
ok: [10.1.61.187] => (item={'key': 'bob', 'value': {'name': 'Bob Bananarama',
'telephone': '987-654-3210'}}) => {
    "msg": "User bob is Bob Bananarama (987-654-3210)"
}

PLAY RECAP
*********************************************************************************************
*********************************************************************************************
```

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

9. with_subelement

with_subelement简单来讲，就是在一个复杂的列表当中，可以对这个列表变量的子元素进行遍历

下面是一个简单的示例：

```
- hosts: test
  gather_facts: no
  vars:
    users:
      - name: bob
        hobby:
          - Games
          - Sports
      - name: alice
        hobby:
          - Music
  tasks:
    - debug:
        msg: "{{ item }}"
      with_subelement:
        - "{{ users }}"
        - hobby
```

输出结果如下：

```
# ansible-playbook with_subelement_ex.yml

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

TASK [debug]
*********************************************************************************************
*********************************************************************
ok: [10.1.61.187] => (item=[{'name': 'bob'}, 'Games']) => {
    "msg": [
        {
            "name": "bob"
        },
        "Games"
    ]
}
ok: [10.1.61.187] => (item=[{'name': 'bob'}, 'Sports']) => {
    "msg": [
        {
            "name": "bob"
```

```
        },
        "Sports"
    ]
}
ok: [10.1.61.187] => (item=[{'name': 'alice'}, 'Music']) => {
    "msg": [
        {
            "name": "alice"
        },
        "Music"
    ]
}

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

可以看到，其按照我们指定的变量users的子项hobby进行了组合输出。with_elementes将hobby子元素的每一项作为一个整体，将其他子元素作为整体，然后组合在一起。

假如现在需要遍历一个用户列表，并创建每个用户，而且还需要为每个用户推送特定的SSH公钥以用于实现远程登录。同时为某一个用户创建独立的mysql登录帐号并为其授权。

示例如下：

```
- hosts: test
  gather_facts: false
  vars:
    users:
      - name: alice
        authorized:
          - files/keys/master1.id_rsa.pub
          - files/keys/master2.id_rsa.pub
        mysql:
            password: mysql-password
            hosts:
              - "%"
              - "127.0.0.1"
              - "::1"
              - "localhost"
            privs:
              - "*.*:SELECT"
              - "DB1.*:ALL"
      - name: bob
        authorized:
          - files/keys/master3.id_rsa.pub
        mysql:
            password: other-mysql-password
            hosts:
              - "db1"
            privs:
              - "*.*:SELECT"
              - "DB2.*:ALL"
```

```
  tasks:
    - user:
        name: "{{ item.name }}"
        state: present
        generate_ssh_key: yes
      with_items: "{{ users }}"

    - authorized_key:
        user: "{{ item.0.name }}"
        key: "{{ lookup('file', item.1) }}"
      with_subelements:
        - "{{ users }}"
        - authorized

    - name: Setup MySQL users
      mysql_user:
        name: "{{ item.0.name }}"
        password: "{{ item.0.mysql.password }}""
        host: "{{ item.1 }} priv={{ item.0.mysql.privs | join('/') }}"
      with_subelements:
        - "{{ users }"
        - mysql.hosts
```

10. with_file

用于循环主控端的文件列表，获取文件中的内容

> 注意： 循环的是主控端的文件列表，不是被控端的

```
- hosts: test
  gather_facts: false
  tasks:
    - debug:
        msg: {{ item }}
      with_file:
        - /etc/ansible/test1.yml
        - /etc/ansible/test2.yml
```

输出如下：

```
# ansible-playbook with_file_ex.yml

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

TASK [debug]
*****************************************************************************************
************************************
ok: [10.1.61.187] => (item=content: test1.yaml) => {
    "msg": "content: test1.yaml"
}
```

```
ok: [10.1.61.187] => (item=content: test2.yml) => {
    "msg": "content: test2.yml"
}

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

11. with_fileglob

上面 `with_file` 用于获取文件的内容，而 `with_fileglob` 则用于匹配文件名称。可以通过该关键字，在指定的目录中匹配符合模式的文件名。与 `with_file` 相同的是，**`with_fileglob` 操作的文件也是主控端的文件而非被控端的文件**

```yaml
- hosts: test
  tasks:
    - name: Make key directory
      file:
        path: /root/.sshkeys
        state: directory
        mode: 0700
        owner: root
        group: root

    - name: Upload public keys
      copy:
        src: "{{ item }}"
        dest: /root/.sshkeys
        mode: 0600
        owner: root
        group: root
      with_fileglob:
        - /root/.ssh/*.pub

    - name: Assemble keys into authorized_keys file
      assemble:
        src: /root/.sshkeys
        dest: /root/.ssh/authorized_keys
        mode: 0600
        owner: root
        group: root
```

12. with_lines

with_lines循环结构会让你在控制主机上执行任意命令，并对命令的输出进行逐行迭代。

假设我们有一个文件test.txt包含如下行：

```
Breeze Yan
Bernie Yang
jerry Qing
```

我们可以通过如下方法进行逐行输出：

```
- name: print all names
  debug: msg="{{ item }}"
  with_lines:
    - cat test.txt
```

13. do-Until循环

```
- action: shell /usr/bin/foo
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

重复执行shell模块，当shell模块执行的命令输出内容包含"all systems go"的时候停止。重试5次，延迟时间10秒。retries默认值为3，delay默认值为5。任务的返回值为最后一次循环的返回结果。