

- 简介
- when关键字
 - 1. when基本使用
 - 2. 比较运算符
 - 3. 逻辑运算符
- 条件判断与tests
 - 判断变量
 - 判断执行结果
 - 判断路径
 - 判断字符串
 - 判断整除
 - 其他tests
- 条件判断与block
 - block
 - rescue
 - always
- 条件判断与错误处理
 - fail模块
 - failed_when
 - changed_when
- 在循环语句中使用条件语句

简介

在有的时候play的结果依赖于变量、fact或者是前一个任务的执行结果，或者有的时候，我们会基于上一个task执行返回的结果而决定如何执行后续的task。这个时候就需要用到条件判断。

条件语句在Ansible中的使用场景：

- 在目标主机上定义了一个硬限制，比如目标主机的最小内存必须达到多少，才能执行该task
- 捕获一个命令的输出，根据命令输出结果的不同以触发不同的task
- 根据不同目标主机的facts，以定义不同的task
- 根据目标机的cpu的大小，以调优相关应用性能
- 用于判断某个服务的配置文件是否发生变更，以确定是否需要重启服务

when关键字

1. when基本使用

在ansible中，使用条件判断的关键字就是when。

如在安装包的时候，需要指定主机的操作系统类型，或者是当操作系统的硬盘满了之后，需要清空文件等，可以使用when语句来做判断。when关键字后面跟着的是python的表达式，在表达式中你能够使用任何的变量或者fact，当表达式的结果返回的是false，便会跳过本次的任务

下面是一个基本的用法示例：

```
---
- name: Install vim
  hosts: all
  tasks:
    - name: Install VIM via yum
      yum:
        name: vim-enhanced
        state: installed
      when: ansible_os_family == "RedHat"

    - name: Install VIM via apt
      apt:
        name: vim
        state: installed
      when: ansible_os_family == "Debian"

    - name: Unexpected OS family
      debug: msg="OS Family {{ ansible_os_family }} is not supported"
      when: not ansible_os_family == "RedHat" or ansible_os_family == "Debian"
```

2. 比较运算符

在上面的示例当中，我们使用了“==”的比较运算符，在ansible中，还支持如下比较运算符：

- ==：比较两个对象是否相等，相等则返回真。可用于比较字符串和数字
- !=：比较两个对象是否不等，不等则为真。
- >：比较两个对象的大小，左边的值大于右边的值，则为真
- <：比较两个对象的大小，左边的值小于右边的值，则为真
- >=：比较两个对象的大小，左边的值大于等于右边的值，则为真
- <=：比较两个对象的大小，左边的值小于等于右边的值，则为真

下面是一些简单的示例：

```
when: ansible_machine == "x86_64"

when: max_memory <= 512
```

3. 逻辑运算符

在Ansible中，除了比较运算符，还支持逻辑运算符：

- and: 逻辑与，当左边和右边两个表达式同时为真，则返回真
- or: 逻辑或，当左右和右边两个表达式任意一个为真，则返回真
- not: 逻辑否，对表达式取反
- (): 当一组表达式组合在一起，形成一个更大的表达式，组合内的所有表达式都是逻辑与的关系

示例：

```
# 逻辑或
when: ansible_distribution == "RedHat" or ansible_distribution == "Fedora"

# 逻辑与
when: ansible_distribution_version == "7.5" and ansible_kernel == "3.10.0-327.el7.x86_64"

when:
- ansible_distribution_version == "7.5"
- ansible_kernel == "3.10.0-327.el7.x86_64"

# 组合

when: =>
( ansible_distribution == "RedHat" and ansible_distribution_major_version == "7" )
or
( ansible_distribution == "Fedora" and ansible_distribution_major_version == "28")
```

一个完整的例子：

```
# 判断register注册变量的返回结果
- name: restart httpd if postfix is running
  hosts: test
  tasks:
    - name: get postfix server status
      command: /usr/bin/systemctl is-active postfix
      ignore_errors: yes
      register: result

    - name: restart apache httpd based on postfix status
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

条件判断与tests

在shell当中，我们可使用test命令来进行一些常用的判断操作，如下：

```
# 判断/test文件是否存在
test -e /test

# 判断/testdir是否存在且为一个目录
test -d /testdir
```

事实上，在ansible中也有类似的用法，只不过ansible没有使用linux的test命令，而是jinja2模板的tests。

下面是一个简单示例：

```
# 通过条件语句判断testpath的路径是否存在
- hosts: test
  vars:
    testpath: /testdir
  tasks:
    - debug:
        msg: "file exist"
      when: testpath is exists
```

上面的示例中，我们使用了 `is exists` 用于路径存在时返回真，也可以使用 `is not exists` 用于路径不存在时返回真。也可以在整个条件表达式的前面使用`not`以取反：

```
- hosts: test
  vars:
    testpath: /testdir1
  tasks:
    - debug:
        msg: "file not exist"
      when: not testpath is exists
```

在ansible中，除了能够使用exists这种tests之外，还有一些别的tests。接下来我们详细说一说。

判断变量

- `defined`: 判断变量是否已定义，已定义则返回真
- `undefined`: 判断变量是否未定义，未定义则返回真
- `none`: 判断变量的值是否为空，如果变量已定义且值为空，则返回真

示例：

```
- hosts: test
  gather_facts: no
  vars:
    testvar: "test"
    testvar1:
  tasks:
```

```
- debug:  
  msg: "testvar is defined"  
  when: testvar is defined  
- debug:  
  msg: "testvar2 is undefined"  
  when: testvar2 is undefined  
- debug:  
  msg: "testvar1 is none"  
  when: testvar1 is none
```

判断执行结果

- sucess或succeeded: 通过任务执行结果返回的信息判断任务的执行状态，任务执行成功则返回true
- failure或failed: 任务执行失败则返回true
- change或changed: 任务执行状态为changed则返回true
- skip或skipped: 任务被跳过则返回true

示例：

```
- hosts: test  
gather_facts: no  
vars:  
  doshell: true  
tasks:  
  - shell: 'cat /testdir/aaa'  
    when: doshell  
    register: result  
    ignore_errors: true  
  - debug:  
    msg: "success"  
    when: result is success  
  
  - debug:  
    msg: "failed"  
    when: result is failure  
  
  - debug:  
    msg: "changed"  
    when: result is change  
  
  - debug:  
    msg: "skip"  
    when: result is skip
```

判断路径

- file: 判断指定路径是否为一个文件，是则为真
- directory: 判断指定路径是否为一个目录，是则为真

- link: 判断指定路径是否为一个软链接, 是则为真
- mount: 判断指定路径是否为一个挂载点, 是则为真
- exists: 判断指定路径是否存在, 存在则为真

特别注意: 关于路径的所有判断均是判断主控端上的路径, 而非被控端上的路径

示例:

```
- hosts: test
  gather_facts: no
  vars:
    testpath1: "/testdir/test"
    testpath2: "/testdir"
  tasks:
    - debug:
        msg: "file"
      when: testpath1 is file
    - debug:
        msg: "directory"
      when: testpath2 is directory
```

判断字符串

- lower: 判断字符串中的所有字母是否都是小写, 是则为真
- upper: 判断字符串中的所有字母是否都是大写, 是则为真

```
- hosts: test
  gather_facts: no
  vars:
    str1: "abc"
    str2: "ABC"
  tasks:
    - debug:
        msg: "str1 is all lowercase"
      when: str1 is lower
    - debug:
        msg: "str2 is all uppercase"
      when: str2 is upper
```

判断整除

- even: 判断数值是否为偶数, 是则为真
- odd: 判断数值是否为奇数, 是则为真
- divisibleby(num): 判断是否可以整除指定的数值, 是则为真

示例:

```

- hosts: test
gather_facts: no
vars:
  num1: 6
  num2: 8
  num3: 15
tasks:
  - debug:
    msg: "num1 is an even number"
    when: num1 is even
  - debug:
    msg: "num2 is an odd number"
    when: num2 is odd
  - debug:
    msg: "num3 can be divided exactly by"
    when: num3 is divisibleby(3)

```

其他tests

1. version

可用于对比两个版本号的大小，或者与指定的版本号进行对比，使用语法为version("版本号","比较操作符")

```

- hosts: test
vars:
  ver1: 1.2
  ver2: 1.3
tasks:
  - debug:
    msg: "ver1 is greater than ver2"
    when: ver1 is version(ver2,>")
  - debug:
    msg: "system version {{ ansible_distribution_version }} greater than 7.3"
    when: ansible_distribution_version is version("7.3","gt")

```

version中使用的比较运算符说明：

- 大于： >, gt
- 大于等于： >=, ge
- 小于： <, lt
- 小于等于： <=, le
- 等于： =, ==, eq
- 不等于： !=, <>, ne

2. subset 判断一个list是不是另一个list的子集

3. superset 判断一个list是不是另一个list的父集"

```

- hosts: test
gather_facts: no
vars:
  a:
    - 2
    - 5
  b: [1,2,3,4,5]
tasks:
  - debug:
    msg: "A is a subset of B"
    when: a is subset(b)
  - debug:
    msg: "B is the parent set of A"
    when: b is superset(a)

```

4. in 判断一个字符串是否存在于另一个字符串中，也可用于判断某个特定的值是否存在于列表中

```

- hosts: test
vars:
  supported_distros:
    - RedHat
    - CentOS
tasks:
  - debug:
    msg: "{{ ansible_distribution }} in supported_distros"
    when: ansible_distribution in supported_distros

```

5. string 判断对象是否为一个字符串，是则为真

6. number 判断对象是否为一个数字，是则为真

```

- hosts: test
gather_facts: no
vars:
  var1: 1
  var2: "1"
  var3: a
tasks:
  - debug:
    msg: "var1 is a number"
    when: var1 is number
  - debug:
    msg: "var2 is a string"
    when: var2 is string
  - debug:
    msg: "var3 is a string"
    when: var3 is string

```

条件判断与block

block

我们在前面使用when做条件判断时，如果条件成立则执行对应的任务。但这就面临一个问题，当我们要使用同一个条件判断执行多个任务的时候，就意味着我们要在某一个任务下面都写一下when语句，而且判断条件完全一样。这种方式不仅麻烦而且显得low。Ansible提供了一种更好的方式来解决这个问题，即block。

在ansible中，使用block将多个任务进行组合，当作一个整体。我们可以对这一个整体做条件判断，当条件成立时，则执行块中的所有任务：

```
- hosts: test
  tasks:
    - debug:
        msg: "task1 not in block"
    - block:
        - debug:
            msg: "task2 in block1"
        - debug:
            msg: "task3 in block1"
  when: 2 > 1
```

下面是一个稍微有用点儿的例子：

```
- hosts: test
  tasks:
    - name: set /etc/resolv.conf
      template:
        src: resolv.conf.j2
        dest: /etc/resolv.conf
        owner: root
        group: root
        mode: 0644
    - block:
        - name: ensure /etc/resolvconf/resolv.conf.d/base file for ubuntu 16.04
          template:
            src: resolv.conf.j2
            dest: /etc/resolvconf/resolv.conf.d/base
        - name: config dns for ubuntu 16.04
          template:
            src: resolv.conf.j2
            dest: /etc/resolv.conf
  when: ansible_distribution == "Ubuntu" and ansible_distribution_major_version ==
"16"
```

使用block注意事项：

1. 可以为block定义name (ansible 2.3增加的特性)
2. 可以直接对block使用when，但不能直接对block使用loop

rescue

block除了能和when一起使用之外，还能作错误处理。这个时候就需要用到rescue关键字：

```
- hosts: test
  tasks:
    - block:
        - shell: 'ls /testdir'
      rescue:
        - debug:
            msg: '/testdir is not exists'
```

在上面的例子中，当block中的任务执行失败时，则运行rescue中的任务。如果block中的任务正常执行，则rescue的任务就不会被执行。如果block中有多个任务，则任何一个任务执行失败，都会执行rescue。block中可以定义多个任务，同样rescue当中也可以定义多个任务。

always

当block执行失败时，rescue中的任务才会被执行；而无论block执行成功还是失败，always中的任务都会被执行：

```
- hosts: test
  tasks:
    - block:
        - shell: 'ls /testdir'
      rescue:
        - debug:
            msg: '/testdir is not exists'
    always:
        - debug:
            msg: 'This task always executes'
```

条件判断与错误处理

在上面讲block的使用方法的时候，我们说block除了可以将多个任务组合到一起，还有错误处理的功能。接下来我们继续说一说错误处理。

fail模块

在shell中，可能会有这样的需求：当脚本执行至某个阶段时，需要对某个条件进行判断，如果条件成立，则立即终止脚本的运行。在shell中，可以直接调用"exit"即可执行退出。事实上，在playbook中也有类似的模块可以做这件事。即fail模块。

fail模块用于终止当前playbook的执行，通常与条件语句组合使用，当满足条件时，终止当前play的运行。

选项只有一个：

- msg：终止前打印出信息

示例：

```
# 使用fail模块中断playbook输出
- hosts: test
  tasks:
    - shell: echo "Just a test--error"
      register: result
    - fail:
        msg: "Conditions established, Interrupt running playbook"
        when: "'error' in result.stdout"
    - debug:
        msg: "I never execute, Because the playbook has stopped"
```

failed_when

事实上，当fail和when组合使用的时候，还有一个更简单的写法，即 failed_when，当满足某个条件时，ansible主动触发失败。

```
# 如果在command_result存在错误输出，且错误输出中，包含了`FAILED`字串，即返回失败状态：
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

也可以直接通过 fail 模块和 when 条件语句，写成如下：

```
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  ignore_errors: True

- name: fail the play if the previous command did not succeed
  fail: msg="the command failed"
  when: " command_result.stderr and 'FAILED' in command_result.stderr"
```

ansible一旦执行返回失败，后续操作就会中止，所以failed_when通常可以用于满足某种条件时主动中止playbook运行的一种方式。

ansible默认处理错误的机制是遇到错误就停止执行。但有些时候，有些错误是计划之中的。我们希望忽略这些错误，以让playbook继续往下执行。这个时候就可以使用 ignore_errors 忽略错误，从而让playbook继续往下执行。

changed_when

当我们控制一些远程主机执行某些任务时，当任务在远程主机上成功执行，状态发生更改时，会返回 changed 状态响应，状态未发生更改时，会返回 OK 状态响应，当任务被跳过时，会返回 skipped 状态响应。我们可以通过 changed_when 来手动更改 changed 响应状态。示例如下：

```
- shell: /usr/bin/billybass --mode="take me to the river"
register: bass_result
changed_when: "bass_result.rc != 2"      #只有该条task执行以后, bass_result.rc的值不为2时, 才
                                            会返回changed状态

# this will never report 'changed' status
- shell: wall 'beep'
  changed_when: False      #当changed_when为false时, 该条task在执行以后, 永远不会返回changed状
                            态
```

在循环语句中使用条件语句

```
# 只打印大于5的值
tasks:
  - command: echo {{ item }}
    loop: [ 0, 2, 4, 6, 8, 10 ]
    when: item > 5

# 确保将mariadb-server安装到根分区且根分区的可用空间要大于300M
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: 拉特st
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 300000000
```