对于任何自动管理工具而言，对于文件的管理都是其绕不开的话题。同样，ansible也围绕文件管理提供了众多的模块。同时还提供了Jinja2模板语法来配置文件模板。

常用文件管理模块

1. file

我们在讲ansible ad-hoc的时候，已经说过file模块，在playbook中的使用也没什么不同，下面给个简单的示例：

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

2. synchronize

synchronize模块示例：

```
- name: synchronize local file to remote files
  synchronize:
    src: file
    dest: /path/to/file
```

3. copy

同样的，我们已经介绍过copy模块，示例如下：

```
- name: copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

4. fetch

fetch模块与copy模块正好相反，copy是把主控端的文件复制到被控端，而fetch则是把被控端的文件复制到主控端。并且在主控端指定的目录下，以被控端主机名的形式来组织目录结构。

```
- name: Use the fetch module to retrieve secure log files
  hosts: all
  user: ansible
  tasks:
    - name: Fetch the /var/log/secure log file from managed hosts
      fetch:
        src: /var/log/secure
        dest: secure-backups
        flat: no
```

在主控端文件存储的目录树如下：

```
# tree  secure-backups/
secure-backups/
└── 10.1.61.187
    └── var
        └── log
            └── secure

3 directories, 1 file
```

> 参考：https://docs.ansible.com/ansible/latest/modules/fetch_module.html#fetch-module

5. lineinfile

lineinfile是一个非常有用的模块，而且相对来说，也是用法比较复杂的模块，可直接参考《Ansible lineinfile模块》

6. stat

stat模块与linux中的stat命令一样，用来显示文件的状态信息。

```
- name: Verify the checksum of a file
  stat:
    path: /path/to/file
    checksum_algorithm: md5
  register: result

- debug:
    msg: "The checksum of the file is {{ result.stat.checksum }}"
```

> 参考： https://docs.ansible.com/ansible/latest/modules/stat_module.html#stat-module

7. blockinfile

围绕着被标记的行插入、更新、删除一个文本块。

```
#cat files/test.html
<html>
  <head>
  </head>
  <body>
  </body>
</html>


#cat blockinfile_ex.yml
---
- name: blockinfile module test
  hosts: test
  tasks:
    - name: copy test.html to dest
      copy:
        src: files/test.html
        dest: /var/www/html/test.html
    - name: add block
      blockinfile:
```

```
        marker: "<!-- {mark} ANSIBLE MANAGED BLOCK -->"
        insertafter: "<body>"
        path: /var/www/html/test.html
        block: |
          <h1>Welcome to {{ ansible_hostname }}</h1>
          <p>Last updated on {{ ansible_date_time.iso8601 }}</p>
```

执行后结果如下：

```
[root@app html]# cat test.html
<html>
  <head>
  </head>
  <body>
<!-- BEGIN ANSIBLE MANAGED BLOCK -->
<h1>Welcome to app</h1>
<p>Last updated on 2019-05-28T15:00:03Z</p>
<!-- END ANSIBLE MANAGED BLOCK -->
  </body>
</html>
```

> 更多blockinfile用法参考：
> https://docs.ansible.com/ansible/latest/modules/blockinfile_module.html#blockinfile-module

Jinja2模板管理

Jinja2简介

Jinja2是基于python的模板引擎。那么什么是模板?

假设说现在我们需要一次性在10台主机上安装redis，这个通过playbook现在已经很容易实现。默认情况下，所有的redis安装完成之后，我们可以统一为其分发配置文件。这个时候就面临一个问题，这些redis需要监听的地址各不相同，我们也不可能为每一个redis单独写一个配置文件。因为这些配置文件中，绝大部分的配置其实都是相同的。这个时候最好的方式其实就是用一个通用的配置文件来解决所有的问题。将所有需要修改的地方使用变量替换，如下示例中redis.conf.j2文件：

```
daemonize yes
supervised systemd
pidfile /var/run/redis.pid
port 6379
logfile "/var/log/redis/redis.log"
dbfilename dump.rdb
dir /data/redis

maxmemory 1G

bind {{ ansible_eth0.ipv4.address }} 127.0.0.1
```

```
timeout 300
loglevel notice

databases 16
save 900 1
save 300 10
save 60 10000

rdbcompression yes

maxclients 10000
appendonly yes
appendfilename appendonly.aof
appendfsync everysec
```

那么此时，redis.conf.j2文件就是一个模板文件。`{{ ansible_eth0.ipv4.address }}`是一个fact变量，用于获取被控端ip地址以实现替换。

在playbook中使用jinja2

现在我们有了一个模板文件，那么在playbook中如何来使用呢?

playbook使用template模块来实现模板文件的分发，其用法与copy模块基本相同，唯一的区别是，copy模块会将原文件原封不动的复制到被控端，而template会将原文件复制到被控端，并且使用变量的值将文件中的变量替换以生成完整的配置文件。

下面是一个完整的示例：

```
# cat config_redis.yml
- name: Configure Redis
  hosts: test
  tasks:
    - name: create redis group
      group:
        name: redis
        gid: 1111
    - name: create redis user
      user:
        name: redis
        uid: 1111
        group: redis
    - name: defined  redismem
      set_fact: redismem="{{ (ansible_memtotal_mb /2) | int }}M"
    - name: install redis
      yum:
        name: redis
        state: present
    - name: create data dir
      file:
        path: /data/redis
        state: directory
```

```
        recurse: yes
        owner: redis
        group: redis
    - name: copy redis.conf to dest
      template:
        src: templates/redis.conf.j2
        dest: /etc/redis.conf
      notify:
        - restart redis
    - name: start redis
      service:
        name: redis
        state: started
        enabled: yes
  handlers:
    - name: restart redis
      service:
        name: redis
        state: restarted
```

执行完成之后，我们可以看到被控端/etc/redis.conf配置文件如下：

```
daemonize yes
pidfile /var/run/redis.pid
port 6379
logfile "/var/log/redis/redis.log"
dbfilename dump.rdb
dir /data/redis

maxmemory {{ redismem }}

bind 10.1.61.187 127.0.0.1

timeout 300
loglevel notice

databases 16
save 900 1
save 300 10
save 60 10000

rdbcompression yes

maxclients 10000
appendonly yes
appendfilename appendonly.aof
appendfsync everysec
```

关于template模块的更多参数说明：

- backup：如果原目标文件存在，则先备份目标文件
- dest：目标文件路径

- force：是否强制覆盖，默认为yes
- group：目标文件属组
- mode：目标文件的权限
- owner：目标文件属主
- src：源模板文件路径
- validate：在复制之前通过命令验证目标文件，如果验证通过则复制

Jinja2条件语句

在上面的示例中，我们直接取了被控节点的eth0网卡的ip作为其监听地址。那么假如有些机器的网卡是bond0，这种做法就会报错。这个时候我们就需要在模板文件中定义条件语句如下：

```
daemonize yes
pidfile /var/run/redis.pid
port 6379
logfile "/var/log/redis/redis.log"
dbfilename dump.rdb
dir /data/redis

maxmemory 1G

{% if ansible_bond0 is defined %}
bind {{ ansible_bond0.ipv4.address }} 127.0.0.1
{% elif ansible_eth0 is defined %}
bind {{ ansible_eth0.ipv4.address }} 127.0.0.1
{% else%}
bind 0.0.0.0
{% endif %}

timeout 300
loglevel notice

databases 16
save 900 1
save 300 10
save 60 10000

rdbcompression yes

maxclients 10000
appendonly yes
appendfilename appendonly.aof
appendfsync everysec
```

我们可以更进一步，让redis主从角色都可以使用该文件：

```
daemonize yes
pidfile /var/run/redis.pid
port 6379
logfile "/var/log/redis/redis.log"
```

```
dbfilename dump.rdb
dir /data/redis

maxmemory 1G

{% if ansible_bond0 is defined %}
bind {{ ansible_bond0.ipv4.address }} 127.0.0.1
{% elif ansible_eth0 is defined %}
bind {{ ansible_eth0.ipv4.address }} 127.0.0.1
{% else%}
bind 0.0.0.0
{% endif %}

{% if masterip is defined %}
slaveof {{ masterip }} {{ masterport|default(6379) }}
{% endif %}

{% if masterpass is defined %}
masterauth {{ masterpass }}
{% endif %}

{% if requirepass is defined %}
requirepass {{ requirepass }}
{% endif %}

timeout 300
loglevel notice

databases 16
save 900 1
save 300 10
save 60 10000

rdbcompression yes

maxclients 10000
appendonly yes
appendfilename appendonly.aof
appendfsync everysec

stop-writes-on-bgsave-error no
```

我们定义一个inventory如下：

```
[redis]
10.1.61.27  masterip=10.1.61.187 masterpass=123456
10.1.61.187 requirepass=123456
```

Jinja2循环语句

定义一个inventory示例如下：

```
[proxy]
10.1.61.195

[webservers]
10.1.61.27
10.1.61.187
```

现在把proxy主机组中的主机作为代理服务器，安装nginx做反向代理，将请求转发至后面的两台webserver，即webserver组的服务器。

现在我们编写一个playbook如下：

```
#cat config_nginx.conf

# 还需要在ansible.cfg中开启facts缓存
- name: gather facts
  gather_facts: Fasle
  hosts: webservers
  tasks:
    - name: gather facts
      setup:

- name: Configure Nginx
  hosts: proxy
  tasks:
    - name: install nginx
      yum:
        name: nginx
        state: present
    - name: copy nginx.conf to dest
      template:
        src: templates/nginx.conf.j2
        dest: /etc/nginx/nginx.conf
      notify:
        - restart nginx
    - name: start nginx
      service:
        name: nginx
        state: started
        enabled: yes
  handlers:
    - name: restart nginx
      service:
        name: nginx
        state: restarted
```

模板文件 templates/nginx.conf.j2示例如下：

```
# cat nginx.conf.j2
user nginx;
worker_processes {{ ansible_processor_vcpus }};
error_log /var/log/nginx/error.log;
```

```
pid /var/run/nginx.pid;
include /usr/share/nginx/modules/*.conf;
events {
    worker_connections 65535;
    use epoll;
}
http {
    map $http_x_forwarded_for $clientRealIP {
        "" $remote_addr;
        ~^(?P<firstAddr>[0-9\.]+),?.*$ $firstAddr;
    }
    log_format  real_ip '{ "datetime": "$time_local", '
                        '"remote_addr": "$remote_addr", '
                        '"source_addr": "$clientRealIP", '
                        '"x_forwarded_for": "$http_x_forwarded_for", '
                        '"request": "$request_uri", '
                        '"status": "$status", '
                        '"request_method": "$request_method", '
                        '"request_length": "$request_length", '
                        '"body_bytes_sent": "$body_bytes_sent", '
                        '"request_time": "$request_time", '
                        '"http_referrer": "$http_referer", '
                        '"user_agent": "$http_user_agent", '
                        '"upstream_addr": "$upstream_addr", '
                        '"upstream_status": "$upstream_status", '
                        '"upstream_http_header": "$upstream_http_host",'
                        '"upstream_response_time": "$upstream_response_time", '
                        '"x-req-id": "$http_x_request_id", '
                        '"servername": "$host"'
                        ' }';
    access_log  /var/log/nginx/access.log  real_ip;
    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;
    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;
    include /etc/nginx/conf.d/*.conf;

    upstream web {
    {% for host in groups['webservers'] %}
        {% if hostvars[host]['ansible_bond0']['ipv4']['address'] is defined %}
        server {{ hostvars[host]['ansible_bond0'] }};
        {% elif hostvars[host]['ansible_eth0'] is defined %}
        server {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }};
        {% endif %}
    {% endfor %}
    }
    server {
        listen       80 default_server;
        server_name  _;
        location / {
            proxy_pass http://web;
        }
    }
}
```

下面再给一个域名解析服务bind的配置文件 named.conf的jinja2模板示例：

```
options {

listen-on port 53 {
127.0.0.1;
{% for ip in ansible_all_ipv4_addresses %}
{{ ip }};
{% endfor %}
};

listen-on-v6 port 53 { ::1; };
directory "/var/named";
dump-file "/var/named/data/cache_dump.db";
statistics-file "/var/named/data/named_stats.txt";
memstatistics-file "/var/named/data/named_mem_stats.txt";
};

zone "." IN {
type hint;
file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";

{# Variables for zone config #}
{% if 'authorativenames' in group_names %}
{% set zone_type = 'master' %}
{% set zone_dir = 'data' %}
{% else %}
{% set zone_type = 'slave' %}
{% set zone_dir = 'slaves' %}
{% endif %}

zone "internal.example.com" IN {
type {{ zone_type }};
file "{{ zone_dir }}/internal.example.com";
{% if 'authorativenames' not in group_names %}
masters { 192.168.2.2; };
{% endif %}
};
```

Jinja2过滤器

1. default过滤器

当指定的变量不存在时，用于设定默认值

简单示例：

```
"Host": "{{ db_host | default('lcoalhost') }}"
```

复杂一点儿的实例：

```
- hosts:
  gather_facts: false
  vars:
    - path: /tmp/test
      mode: 0400
    - path: /tmp/foo
    - path: /tmp/bar
  tasks:
    - file:
        path: {{ item.path }}
        state: touch
        mode: {{ item.mode|default(omit)}}
      with_items: "{{ paths }}"
```

2. 字符串操作相关的过滤器

- upper：将所有字符串转换为大写
- lower：将所有字符串转换为小写
- capitalize：将字符串的首字母大写，其他字母小写
- reverse：将字符串倒序排列
- first：返回字符串的第一个字符
- last：返回字符串的最后一个字符
- trim：将字符串开头和结尾的空格去掉
- center(30)：将字符串放在中间，并且字符串两边用空格补齐30位
- length：返回字符串的长度，与count等价
- list：将字符串转换为列表
- shuffle：list将字符串转换为列表，但是顺序排列，shuffle同样将字符串转换为列表，但是会随机打乱字符串顺序

示例：

```
- hosts: test
  gather_facts: no
  vars:
    teststr: "abc123ABV"
    teststr1: " abc "
    teststr2: "123456789"
    teststr3: "sfacb1335@#$%"
  tasks:
    - debug:
        msg: "{{ teststr | upper }}"
    - debug:
        msg: "{{ teststr | lower }}"
```

```
    - debug:
        msg: "{{ teststr | capitalize }}"
    - debug:
        msg: "{{ teststr | reverse }}"
    - debug:
        msg: "{{ teststr|first }}"
    - debug:
        msg: "{{ teststr|last }}"
    - debug:
        msg: "{{ teststr1 | trim }}"
    - debug:
        msg: "{{ teststr2 | center(30) }}"
    - debug:
        msg: "{{ teststr2 | length }}"
    - debug:
        msg: "{{ teststr3 | list }}"
    - debug:
        msg: "{{ teststr3 | shuffle }}"
```

3. 数字操作相关的过滤器

- int： 将对应的值转换为整数
- float：将对应的值转换为浮点数
- abs：获取绝对值
- round：小数点四舍五入
- random：从一个给定的范围中获取随机值

示例：

```
- hosts: test
  gather_facts: no
  vars:
    testnum: -1
  tasks:
    - debug:
        msg: "{{  8+('8'|int) }}"
    - debug:
        # 默认情况下，如果无法完成数字转换则返回0
        # 这里指定如果无法完成数字转换则返回6
        msg: "{{ 'a'|int(default=6) }}"
    - debug:
        msg: "{{ '8'|float }}"
    - debug:
        msg: "{{ 'a'|float(8.88)' }}"
    - debug:
        msg: "{{ testnum|abs }}"
    - debug:
        msg: "{{ 12.5|round }}"
    - debug:
        msg: "{{ 3.1415926 | round(5) }}"
    - debug:
        # 从0到100中随机返回一个数字
        msg: "{{ 100|random }}"
```

```
    - debug:
        # 从5到10中随机返回一个数字
        msg: "{{ 10|random(start=5) }}"
    - debug:
        # 从4到15中随机返回一个数字，步长为3
        # 返回的随机数只可能是：4，7，10，13中的一个
        msg: "{{ 15|random(start=4,step=3) }}"
    - debug:
        # 从0到15随机返回一个数字，步长为4
        msg: "{{ 15|random(step=4) }}"
```

4. 列表操作相关的过滤器

- length: 返回列表长度
- first：返回列表的第一个值
- last：返回列表的最后一个值
- min：返回列表中最小的值
- max：返回列表中最大的值
- sort：重新排列列表，默认为升序排列，sort(reverse=true)为降序
- sum：返回纯数字非嵌套列表中所有数字的和
- flatten：如果列表中包含列表，则flatten可拉平嵌套的列表,levels参数可用于指定被拉平的层级
- join：将列表中的元素合并为一个字符串
- random：从列表中随机返回一个元素
- shuffle
- upper
- lower
- union：将两个列表合并，如果元素有重复，则只留下一个
- intersect：获取两个列表的交集
- difference：获取存在于第一个列表中，但不存在于第二个列表中的元素
- symmetric_difference：取出两个列表中各自独立的元素，如果重复则只留一个

示例：

```
- hosts: test
  gather_facts: false
  vars:
    testlist1: [1,2,4,6,3,5]
    testlist2: [1,[2,3,4,[5,6]]]
    testlist3: [1,2,'a','b']
    testlist4: [1,'A','b',['C','d'],'Efg']
    testlist5: ['abc',1,2,'a',3,2,'1','abc']
    testlist6: ['abc',3,'1','b','a']
  tasks:
    - debug:
        msg: "{{ testlist1 | length }}"
    - debug:
        msg: "{{ testlist1 |first }}"
```

```
    - debug:
        msg: "{{ testlist1 | last }}"
    - debug:
        msg: "{{ testlist1 | min }}"
    - debug:
        msg: "{{ testlist1 | max }}"
    - debug:
        msg: "{{ testlist1 | sort }}"
    - debug:
        msg: "{{ testlist1 | sort(reverse=true) }}"
    - debug:
        msg: "{{ testlist2 | flatten }}"
    - debug:
        msg: "{{ testlist2 | flatten(levels=1) }}"
    - debug:
        msg: "{{ testlist2 | flatten | max }}"
    - debug:
        msg: "{{ testlist3 | join }}"
    - debug:
        msg: "{{ testlist3 |join(',')}}"
    - debug:
        msg: "{{ testlist3 | random }}"
    - debug:
        msg: "{{ testlist3 | shuffle }}"
    - debug:
        msg: "{{ testlist4 | upper }}"
    - debug:
        msg: "{{ testlist4 | lower }}"
    - debug:
        msg: "{{ testlist5 | union(testlist6) }}"
    - debug:
        msg: "{{ testlist5 | intersect(testlist6) }}"
    - debug:
        msg: "{{ testlist5 | difference(testlist6) }}"
    - debug:
        msg: "{{ testlist5 | symmetric_difference(testlist6) }}"
```

5. 应用于注册变量的过滤器

正常情况下，当某个task执行失败的时候，ansible会中止运行。此时我们可以通过 `ignore_errors` 来捕获异常以让task继续往下执行。然后调用debug模块打印出出错时的内容，拿来错误结果后，主动失败。

```
- name: Run myprog
  command: /opt/myprog
  register: result
  ignore_errors: True

- debug:
    var: result

- debug:
    msg: "Stop running the playbook if myprog failed"
  failed_when: result|failed
```

任务返回值过滤器：

- failed： 如果注册变量的值是任务failed则返回True
- changed: 如果注册变量的值是任务changed则返回True
- success：如果注册变量的值是任务succeeded则返回True
- skipped：如果注册变量的值是任务skipped则返回True

> 在ansible2.9中，该方式会被废弃，不推荐使用

6. 应用于文件路径的过滤器

- basename：返回文件路径中的文件名部分
- dirname：返回文件路径中的目录部分
- expanduser：将文件路径中的~替换为用户目录
- realpath：处理符号链接后的文件实际路径

下面是一个示例：

```
- name: test basename
  hosts: test
  vars:
    homepage: /usr/share/nginx/html/index.html
  tasks:
    - name: copy homepage
      copy:
        src: files/index.html
        dest: {{ homepage }}
```

可以通过basename改写成如下方式：

```
- name: test basename
  hosts: test
  vars:
    homepage: /usr/share/nginx/html/index.html
  tasks:
    - name: copy homepage
      copy:
        src: files/{{ homepage | basename }}
        dest: {{ homepage }}
```

7. 自定义过滤器

举个简单的例子，现在有一个playbook如下：

```
- name: test filter
  hosts: test
  vars:
    domains: ["www.example.com","example.com"]
  tasks:
    template:
      src: templates/test.conf.j2
      dest: /tmp/test.conf
```

templates/test.conf.j2如下：

```
hosts = [{{ domains | join(',') }}]
```

执行playbook后，在目标机上的test.conf如下：

```
hosts = [www.example.com,example.com]
```

现在如果希望目标机上的test.conf文件返回结果如下：

```
hosts = ["www.example.com","example.com"]
```

没有现成的过滤器来帮我们做这件事情。我们可以自己简单写一个surround_by_quote.py内容如下：

```
# 定义过滤器执行的操作
def surround_by_quote(a_list):
  return ['"%s"' % an_element for an_element in a_list]

class FilterModule(object):
  def filters(self):
    return {'surround_by_quote': surround_by_quote}
```

我们需要开启ansible.cfg的配置项：

```
filter_plugins     = /usr/share/ansible/plugins/filter
```

将刚刚编写的代码文件放入/usr/share/ansible/plugins/filter目录下，然后修改templates/test.conf.j2如下：

```
hosts = [{{ domains | join(',') |surround_by_quote }}]
```

再次执行playbook，最后返回结果：

```
hosts = ["www.example.com","example.com"]
```

> 关于jinja2更多用法参考：http://docs.jinkan.org/docs/jinja2/