# 第3天自动化运维利器Ansible-Playbook的任务控制

## 一、 Ansible 任务控制基本介绍

这里主要来介绍PlayBook中的任务控制。

任务控制类似于编程语言中的if ... 、for ... 等逻辑控制语句。

这里我们给出一个实际场景应用案例去说明在PlayBook中，任务控制如何应用。

在下面的PlayBook中，我们创建了 tomcat、www 和 mysql 三个用户。 安装了Nginx 软件包、并同时更新了 Nginx 主配置文件和虚拟主机配置文件，最后让Nginx 服务处于启动状态。

整个PlayBook从语法上没有任何问题，但从逻辑和写法上仍然有一些地方需要我们去注意及优化:

1. Nginx启动逻辑欠缺考虑。若Nginx的配置文件语法错误则会导致启动Nginx失败，以至于PlayBook执行失败。
2. 批量创建用户，通过指令的罗列过于死板。如果再创建若干个用户，将难以收场。

```
---
- name: task control playbook example
  hosts: webservers
  tasks:
    - name: create tomcat user
      user: name=tomcat state=present

```

```
    - name: create www user
      user: name=www state=present

    - name: create mysql user
      user: name=mysql state=present

    - name: yum nginx webserver
      yum: name=nginx state=present

    - name: update nginx main config
      copy: src=nginx.conf dest=/etc/nginx/

    - name: add virtualhost config
      copy: src=www.qfedu.com.conf
dest=/etc/nginx/conf.d/

    - name: start nginx server
      service: name=nginx state=started
```

```
# cat nginx.conf
user  www;
worker_processes 2;

error_log  /var/log/nginx/error.log;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

```

```
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    sendfile        on;
    tcp_nopush     on;

    keepalive_timeout  0;

    gzip on;
    gzip_min_length  1k;
    gzip_buffers     8 64k;
    gzip_http_version 1.0;
    gzip_comp_level 5;
    gzip_types   text/plain application/x-javascript text/css application/json application/xml application/x-shockwave-flash application/javascript image/svg+xml image/x-icon;
    gzip_vary on;

    include /etc/nginx/conf.d/*.conf;
}
```

```
# cat www.qfedu.com.conf
server {
    listen 80;
    server_name www.qfedu.com;
    root /usr/share/nginx/html;
    access_log /var/log/nginx/www.qfedu.com-access_log main;
```

```
    error_log  /var/log/nginx/www.qfedu.com-error_log;

    add_header Access-Control-Allow-Origin *;

    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
            expires      1d;
    }

    location ~ .*\.(js|css)?$ {
            expires      1d;
    }
}
```

我们下面将以解决一个个问题的形式去优化上例中的PlayBook。

通过问题的解决，来达到我们学习任务控制的目的。



# 二、条件判断

**解决第一个问题**

Nginx启动逻辑欠缺考虑。 若Nginx的配置文件语法错误则会导致启动Nginx失败，以至于PlayBook执行失败。

如果我们能够在启动之前去对Nginx的配置文件语法做正确性的校验，只有当校验通过的时候我们才去启动或者重启Nginx；否则则跳过启动Nginx的过程。这样就会避免Nginx 配置文件语法问题而导致的无法启动Nginx的风险。

**Nginx 语法校验**

```
- name: check nginx syntax
  shell: /usr/sbin/nginx -t
```

那如何将Nginx语法检查的TASK同Nginx启动的TASK关联起来呢?

如果我们能够获得语法检查的TASK的结果，根据这个结果去判断“启动NGINX的TASK”是否执行，这将是一个很好的方案。 如何和获取到语法检查TASK的结果呢? 此时就可以使用之前学到的 Ansible中的注册变量。

**获取Task任务结果**

```
- name: check nginx syntax
  shell: /usr/sbin/nginx -t
  register: nginxsyntax
```

此时有可能还有疑问，我获取到任务结果，但是结果里面的内容是个什么样子， 我如何根据内容在后续的PlayBook中使用呢?

**通过debug模块去确认返回结果的数据结构**

```
- name: print nginx syntax result
  debug: var=nginxsyntax
```

通过debug 模块，打印出来的返回结果。 当nginxsyntax.rc 为 0 时语法校验正确。

```
"nginxsyntax": {
        "changed": true,
        "cmd": "/usr/sbin/nginx -t",
        "delta": "0:00:00.012045",
        "end": "2017-08-12 20:19:04.650718",
        "rc": 0,
        "start": "2017-08-12 20:19:04.638673",
        "stderr": "nginx: the configuration file
/etc/nginx/nginx.conf syntax is ok\nnginx:
configuration file /etc/nginx/nginx.conf test is
successful",
```

```
        "stderr_lines": [
            "nginx: the configuration file /etc/nginx/nginx.conf syntax is ok",
            "nginx: configuration file /etc/nginx/nginx.conf test is successful"
        ],
        "stdout": "",
        "stdout_lines": []
    }
```

**通过条件判断(when) 指令去使用语法校验的结果**

```
    - name: check nginx syntax
      shell: /usr/sbin/nginx -t
      register: nginxsyntax

    - name: print nginx syntax
      debug: var=nginxsyntax

    - name: start nginx server
      service: name=nginx state=started
      when: nginxsyntax.rc == 0
```

**改进后的PlayBook**

```
---
- name: task control playbook example
  hosts: webservers
  gather_facts: no
  tasks:
    - name: create tomcat user
      user: name=tomcat state=present

    - name: create www user
      user: name=www state=present
```

```
11
12      - name: create mysql user
13        user: name=mysql state=present
14
15      - name: yum nginx webserver
16        yum: name=nginx state=present
17
18      - name: update nginx main config
19        copy: src=nginx.conf dest=/etc/nginx/
20
21      - name: add virtualhost config
22        copy: src=www.qfedu.com.conf
    dest=/etc/nginx/conf.d/
23
24      - name: check nginx syntax
25        shell: /usr/sbin/nginx -t
26        register: nginxsyntax
27
28      - name: print nginx syntax
29        debug: var=nginxsyntax
30
31      - name: start nginx server
32        service: name=nginx state=started
33        when: nginxsyntax.rc == 0
```



以上的逻辑，只要语法检查通过都会去执行 "start nginx server"这个TASK。 在这个问题的解决里，我们学习了`when` 条件判断和注册变量的结合使用。学习了when条件判断中是可以支持复杂逻辑的。比如现在用到的逻辑运算符 `and`。

另外 `when` 支持如下运算符:

```
==
!=
> >=
< <=
is defined
is not defined
true
false
支持逻辑运算符: and or
```

# 三、 循环控制

**解决第二个问题**

批量创建用户，通过指令的罗列过于死板。如果再创建若干个用户，将难以收场。

如果在创建用户时,抛开PlayBook的实现不说, 单纯的使用shell去批量的创建一些用户。通常会怎么写呢?

```shell
#! /bin/bash
createuser="tomcat mysql www"
for i in `echo $createuser`
do
   useradd $i
done
```

那么如果PlayBook中也存在这样的循环控制，我们也可以像写shell一样简单的去完成多用户创建工作。

在PlayBook中使用`with_items`去实现循环控制，且循环时的中间变量(上面shell循环中的`$i`变量)只能是关键字`item`，而不能随意自定义。

**在上面的基础上，改进的PlayBook**

在这里使用定义了剧本变量 createuser(一个列表)，然后通过 with_items 循环遍历变量这个变量来达到创建用户的目的。

```
- name: variable playbook example
  hosts: webservers
  gather_facts: no
  vars:
    createuser:
      - tomcat
      - www
      - mysql
  tasks:
    - name: create user
      user: name={{ item }} state=present
      with_items: "{{ createuser }}"

    - name: yum nginx webserver
      yum: name=nginx state=present

    - name: update nginx main config
      copy: src=nginx.conf dest=/etc/nginx/

    - name: add virtualhost config
      copy: src=www.qfedu.com.conf
dest=/etc/nginx/conf.d/

    - name: check nginx syntax
      shell: /usr/sbin/nginx -t
      register: nginxsyntax

    - name: print nginx syntax
      debug: var=nginxsyntax

```

```
    - name: start nginx server
      service: name=nginx state=started
      when: nginxsyntax.rc == 0
```

解决了以上问题，整个PlayBook已经有了很大的改进。

**新版本循环**

```
- name: loop item
  hosts: all
  gather_facts: no
  vars:
    some_list:
      - "a"
      - "b"
      - "c"
    num_list:
      - 1
      - 2
      - 3
      - 5
  tasks:
    - name: show item
      debug:
        var: "{{ item }}"
      loop: "{{ some_list }}"

    - name: show item when item > 3
      debug:
        var: "{{ item }}"
      loop: "{{ num_list }}"
      when: item > 3
```

考虑这样一个情况：

若更新了Nginx 的配置文件后，我们需要通过PlayBook将新的配置发布到生产服务器上，然后再重新加载我们的Nginx 服务。但以现在的PlayBook来说，每次更改Nginx 配置文件后虽然可以通过它发布到生产，但整个PlayBook都要执行一次，这样无形中扩大了变更范围和变更风险。

下面的 Tags 属性就可以解决这个问题。

# 三、 Tags属性

我们可以通过Play中的tags 属性，去解决目前PlayBook变更而导致的扩大变更范围和变更风险的问题。

在改进的PlayBook中，针对文件发布TASK 任务

"update nginx main config" 和 "add virtualhost config"

新增了属性 tags ，属性值为updateconfig。

另外我们新增"reload nginx server" TASK任务。当配置文件更新后，去reload Nginx 服务。

那重新加载需要依赖于 Nginx 服务是已经启动状态。所以，还需要进一步通过判断 Nngix 的 pid 文件存在，才证明 Nginx 服务本身是启动中，启动中才可以 reload Nginx 服务。

判断一个文件是否存在使用 `stat` 模块

```
    - name: check nginx running
      stat: path=/var/run/nginx.pid
      register: nginxrunning
```

观察结果，会发现 `nginxrunning.stat.exists` 的值是 `true` 就表示启动状态，是 `false` 就是关闭状态。

接下来下来就可以依据这个结果，来决定是否重新加载 Nginx 服务。

**改进PlayBook**

```
- name: tags playbook example
  hosts: webservers
  gather_facts: no
  vars:
    createuser:
      - tomcat
      - www
      - mysql
  tasks:
    - name: create user
      user: name={{ item }} state=present
      with_items: "{{ createuser }}"

    - name: yum nginx webserver
      yum: name=nginx state=present

    - name: update nginx main config
      copy: src=nginx.conf dest=/etc/nginx/
      tags: updateconfig

    - name: add virtualhost config
      copy: src=www.qfedu.com.conf
dest=/etc/nginx/conf.d/
      tags: updateconfig

    - name: check nginx syntax
      shell: /usr/sbin/nginx -t
```

```
      register: nginxsyntax
      tags: updateconfig

    - name: check nginx running
      stat: path=/var/run/nginx.pid
      register: nginxrunning
      tags: updateconfig

    - name: print nginx syntax
      debug: var=nginxsyntax

    - name: print nginx syntax
      debug: var=nginxrunning

    - name: reload nginx server
      service: name=nginx state=started
      when: nginxsyntax.rc == 0 and
nginxrunning.stat.exists == true
      tags: updateconfig

    - name: start nginx server
      service: name=nginx state=started
      when:
        - nginxsyntax.rc == 0
        - nginxrunning.stat.exists == false
      tags: updateconfig
```



**指定tags 去执行PlayBook**

执行时一定要指定tags，这样再执行的过程中只会执行task 任务上打上tag 标记为 updateconfig 的任务

```
# ansible-playbook -i hosts site.yml -t updateconfig
```

# 四、 Handlers 属性

观察当前的 Playbook，不能发现，当我的配置文件没有发生变化时，每次依然都会去触发TASK "reload nginx server"。

如何能做到只有配置文件发生变化的时候才去触发TASK "reload nginx server"，这样的处理才是最完美的实现。此时可以使用 handlers 属性。

**改进PlayBook**

```
- name: handlers playbook example
  hosts: webservers
  gather_facts: no
  vars:
    createuser:
      - tomcat
      - www
      - mysql
  tasks:
    - name: create user
      user: name={{ item }} state=present
      with_items: "{{ createuser }}"

    - name: yum nginx webserver
      yum: name=nginx state=present

    - name: update nginx main config
      copy: src=nginx.conf dest=/etc/nginx/
      tags: updateconfig
      notify: reload nginx server

    - name: add virtualhost config
```

```
23        copy: src=www.qfedu.com.conf
dest=/etc/nginx/conf.d/
24        tags: updateconfig
25        notify: reload nginx server
26
27      - name: check nginx syntax
28        shell: /usr/sbin/nginx -t
29        register: nginxsyntax
30        tags: updateconfig
31
32      - name: check nginx running
33        stat: path=/var/run/nginx.pid
34        register: nginxrunning
35        tags: updateconfig
36
37      - name: start nginx server
38        service: name=nginx state=started
39        when:
40          - nginxsyntax.rc == 0
41          - nginxrunning.stat.exists == false
42    handlers:
43      - name: reload nginx server
44        service: name=nginx state=reloaded
45        when:
46          - nginxsyntax.rc == 0
47          - nginxrunning.stat.exists == true
```



在改进的PlayBook中，我们针对文件发布TASK 任务 "update nginx main config" 和 "add virtualhost config" 增加了新属性 notify, 值为 "reload nginx server"。

它的意思是说，针对这两个文件发布的TASK,设置一个通知机制，当Ansible 认为文件的内容发生了变化(文件MD5发生变化了)，它就会发送一个通知信号，通知 handlers 中的某一个任务。具体发送到handlers中的哪个任务，由notify 的值"reload nginx server"决定。通知发出后handlers 会根据发送的通知，在handlers中相关的任务中寻找名称为"reload nginx server" 的任务。

当发现存在这样名字的TASK，就会执行它。若没有找到，则什么也不做。若我们要实现这样的机制，千万要注意notify属性设置的值，一定要确保能和handlers中的TASK 名称对应上。

执行**

首次执行，若配置文件没有发生变化，可以发现根本就没有触发handlers 中TASK任务

```
# ansible-playbook  -i hosts site.yml -t updateconfig
```

人为对Nginx 配置文件稍作修改，只要MD5校验值发生变化即可。此时再执行，发现触发了handlers 中的TASK任务

```
# ansible-playbook -i hosts site.yml -t updateconfig
```