Tedu.cn
达内教育

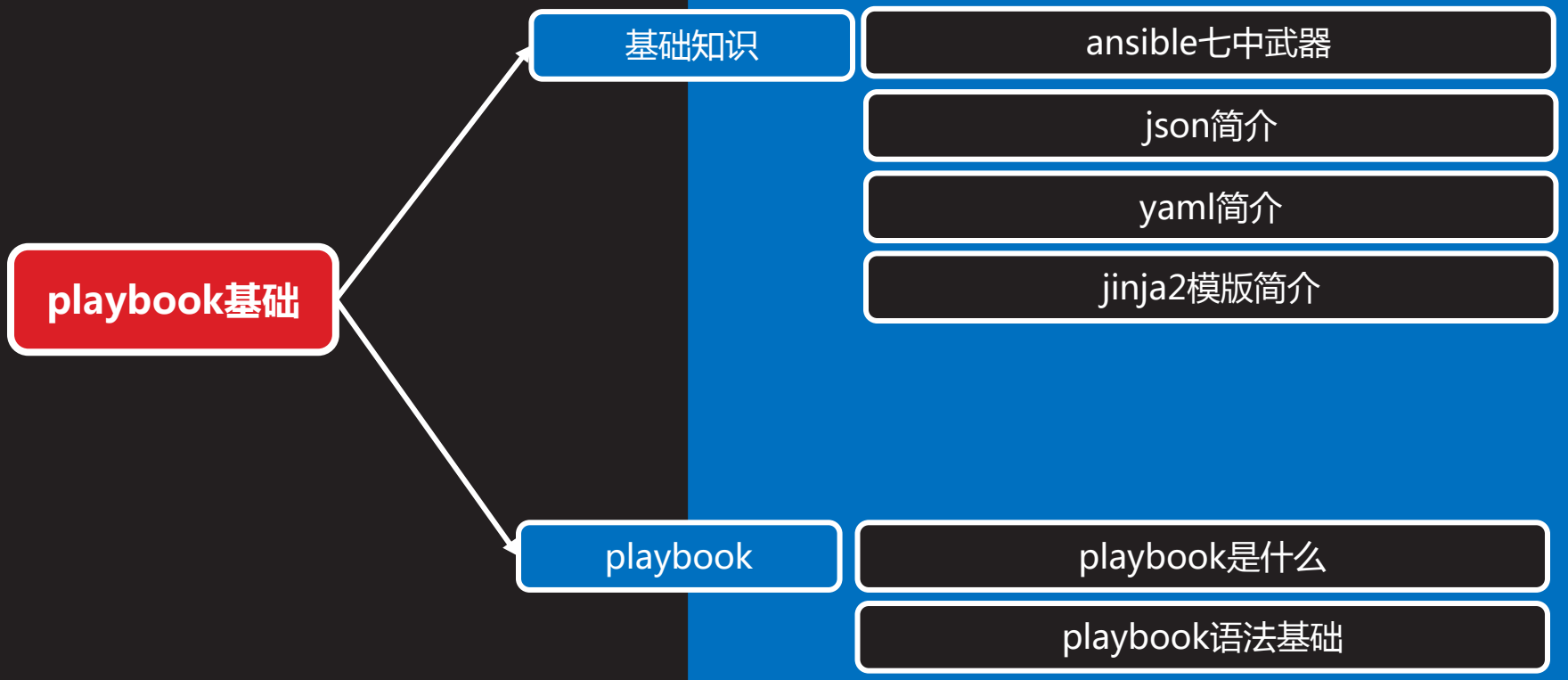
大型架构及配置技术

NSD ARCHITECTURE DAY02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	playbook基础
	10:30 ~ 11:20	
	11:30 ~ 12:00	playbook进阶
下午	14:00 ~ 15:00	
	15:20 ~ 16:00	
	16:30 ~ 17:30	
	17:30 ~ 18:00	总结和答疑

playbook基础


playbook基础
基础知识
ansible七中武器
json简介
yaml简介
jinja2模版简介
playbook
playbook是什么
playbook语法基础

Tedu.cn
达内教育

基础知识

ansible七种武器

- 第一种武器
  - ansible 命令，用于执行临时性的工作，也是我们之前主要学习的功能，必须掌握

- 第二种武器
  - ansible-doc 是 Ansible模块文档说明，针对每个模块都有详细的用法说明及应用案例介绍，功能和Linux系统man命令类似，必须掌握

知识讲解

ansible七种武器

- 第三种武器
  - ansible-console 是 Ansible 为用户提供的一款交互式工具，用户可以在 ansible-console 虚拟出来的终端上像 Shell 一样使用 Ansible 内置的各种命令，这为习惯于使用 Shell 交互方式的用户提供了良好的使用体验。
- 第四种武器
  - ansible-galaxy 从 github 上下载管理 Roles 的一款工具，与 python 的 pip 类似。

知识讲解

ansible七种武器

- 第五种武器
  - ansible-playbook 是日常应用中使用频率最高的命令，其工作机制是：通过读取预先编写好的 playbook 文件实现批量管理。要实现的功能与命令 ansible 一样，可以理解为按一定条件组成的 ansible 任务集，必须掌握

- 第六种武器
  - ansible-vault 主要用于配置文件加密，如编写的 Playbook 配置文件中包含敏感信息，不希望其他人随意查看， ansible-vault 可加密/解密这个配置文件

知识讲解

ansible七种武器

- 第七种武器
  - ansible-pull
  - Ansible 有两种工作模式 pull/push ，默认使用 push 模式工作，pull 模式和通常使用的 push 模式工作机理刚好相反
  - 适用场景：有数量巨大的机器需要配置，即使使用高并发线程依旧要花费很多时间；
  - 通常在配置大批量机器的场景下会使用，灵活性稍有欠缺，但效率几乎可以无限提升，对运维人员的技术水平和前瞻性规划有较高要求。

知识讲解

json简介

- json 是什么？
  - json 是 JavaScript 对象表示法，它是一种基于文本，独立于语言的轻量级数据交换格式。
  - JSON中的分隔符限于单引号 ' 、小括号 ()、中括号 [ ]、大括号 { } 、冒号 ： 和逗号 ，
- json 特性
  - JSON 是纯文本
  - JSON 具有"自我描述性"（人类可读）
  - JSON 具有层级结构（值中存在值）
  - JSON 可通过 JavaScript 进行解析

知识讲解

json简介

- json 语法规则
  - 数据在名称/值对中
  - 数据由逗号分隔
  - 大括号保存对象
  - 中括号保存数组
- json 数据的书写格式是：名称/值对。
  - 名称/值对包括字段名称（在双引号中），后面写一个冒号，然后是值，例如：

    "漂亮姐"："庞丽静"

知识讲解

json简介

- json 语法规则之数组

```
{   "讲师":
    ["牛犇", "丁丁", "静静","李欣"]
}
```

- 复合复杂类型

```
{   "讲师":
    [ {"牛犇":"小逗逼", "负责阶段":"1"},
      {"丁丁":"老逗逼", "负责阶段":"2"},
      {"静静":"漂亮姐", "负责阶段":"3"},
      {"李欣":"老司机", "负责阶段":"4"}
    ]
}
```

知识讲解

yaml简介

- yaml 是什么
  - 是一个可读性高，用来表达数据序列的格式。
  - YAML：YAML Ain't Markup Language
  - YAML参考了其他多种语言，包括：C语言、Python、Perl，并从XML、电子邮件的数据格式（RFC 2822）中获得灵感。Clark Evans在2001年首次发表了这种语言[1]，另外Ingy döt Net与Oren Ben-Kiki也是这语言的共同设计者[2]。目前已经有数种编程语言或脚本语言支持（或者说解析）这种语言。

知识讲解

yaml简介

- yaml 基础语法
  - YAML的结构通过空格来展示
  - 数组使用"- "来表示
  - 键值对使用": "来表示
  - YAML使用一个固定的缩进风格表示数据层级结构关系
  - 一般每个缩进级别由两个以上空格组成
  - # 表示注释
- 注意：
  - 不要使用tab，缩进是初学者容易出错的地方之一
  - 同一层级缩进必须对齐

知识讲解

yaml简介

- YAML的键值表示方法
  - 采用冒号分隔
  - : 后面必须有一个空格
  - YAML键值对例子

```
"庞丽静": "漂亮姐"
```

  - 或

```
"庞丽静":
  "漂亮姐"
```

知识讲解

yaml简介

- 复杂YAML的键值对嵌套

```
"讲师":
    "庞丽静": "漂亮姐"
```

- 或

```
"讲师":
   "庞丽静":
      "漂亮姐"
```

- 数组

```
["牛犇", "丁丁", "静静", "李欣"]
```

知识讲解

yaml简介

- YAML 数组表示方法
  - 使用一个短横杠加一个空格
  - YAML 数组例子

```
- "牛犇"
- "丁丁"
- "静静"
- "李欣"
```

  - 哈希数组复合表达式

```
"讲师":
  - "牛犇"
  - "丁丁"
  - "静静"
  - "李欣"
```

知识讲解

yaml简介

- 高级复合表达式

```
"讲师":
    -
        "牛犇": "小逗比"
        "阶段": 1
    -
        "丁丁": "老逗比"
        "阶段": 2
    -
        "静静": "漂亮姐"
        "阶段": 3
    -
        "李欣": "老司机"
        "阶段": 4
```

知识讲解

yaml简介

- yaml高级语法
  - | 与 > 表示对应的值为多行字符， > 与 | 的区别是会把 \n 转换为空格
  - ! 可以设置类型，!! 可以强制类型转换
  - 为了维持文件的简洁，并避免数据输入的错误，YAML提供了结点参考（*）和散列合并（<<）参考到其他结点标签的锚点标记（&）。参考会将树状结构加入锚点标记的内容，并可以在所有数据结构中运作，合并只有散列表可以使用，可以将键值自锚点标记复制到指定的散列表中

知识讲解

jinja2模版简介

- jinja2 是什么
  - Jinja2是基于python的模板引擎，包含 变量 和 表达式两部分，这两者在模板求值的时候会被替换为值。模板中还有标签，控制模板的逻辑。

- 为什么要学习 jinja2 模版
  - 要使用 ansible 就要深入学习 playbook 配置及模板。playbook 的模板使用 python 的 jinja2 模块来处理的

知识讲解

jinja2模版简介

- jinja2 模版基本语法
  - 模板的表达式都是包含在分隔符 "{{ }}" 内的；
  - 控制语句都是包含在分隔符 "{% %}" 内的；
  - 另外，模板也支持注释，都是包含在分隔符 "{# #}" 内，支持块注释。
  - 调用变量

    {{varname}}
  - 计算

    {{2+3}}
  - 判断

    {{1 in [1,2,3]}}

知识讲解

jinja2模版简介

- jinja2 模版控制语句

```
{% if name == '小逗逼'  %}
    讲故事，吹牛B
{% elif name == '老逗逼' %}
    黑丝（82年的）
{% elif name == '漂亮姐' %}
    约会
{% else %}
    沉迷学习，无法自拔
{% endif %}
```

知识讲解

jinja2模版简介

- jinja2 模版控制语句

```
{% if name ==  ... ... %}
    ... ...
{% elif name == '漂亮姐' %}
    {% for  method in [约会, 逛街, 吃饭, 看电影, 去宾馆] %}
        {{do method}}
    {% endfor %}
    ... ...
{% endif %}
```

知识讲解

jinja2模版简介

- jinja2 过滤器
  - 变量可以通过 过滤器 修改。过滤器与变量用管道符号（ | ）分割，并且也 可以用圆括号传递可选参数。多个过滤器可以链式调用，前一个过滤器的输出会被作为 后一个过滤器的输入。
  - 例如：
  - 把一个列表用逗号连接起来: {{ list|join(', ') }}
  - 过滤器这里不一一列举，需要的可以查询在线文档 http://docs.jinkan.org/docs/jinja2/templates.html#builtin-filters

知识讲解

playbook

playbook是什么

- playbook 是什么？
  - playbook 是 ansible 用于配置，部署，和管理托管主机剧本。通过 playbook 的详细描述，执行其中的一系列 tasks，可以让远端主机达到预期的状态。
  - 也可以这么理解，playbook 字面意思，即剧本，现实中由演员按照剧本表演，在 Ansible 中由计算机进行表演，由计算机安装，部署应用，提供对外服务，以及组织计算机处理各种各样的事情

知识讲解

playbook是什么

- 为什么要使用playbook
  - 执行一些简单的任务，使用ad-hoc命令可以方便的解决问题，但是有时一个设施过于复杂，需要大量的操作时候，执行的 ad-hoc 命令是不适合的，这时最好使用 playbook，就像执行 shell 命令与写 shell 脚本一样，也可以理解为批处理任务
  - 使用 playbook 你可以方便的重用编写的代码，可以移植到不同的机器上面，像函数一样，最大化的利用代码在使用 Ansible 的过程中，你也会发现，你所处理的大部分操作都是编写 playbook

知识讲解

playbook语法基础

- playbook 语法格式
  - playbook由 YAML 语言编写，遵循 YAML 标准
  - 在同一行中，#之后的内容表示注释
  - 同一个列表中的元素应该保持相同的缩进
  - playbook 由一个或多个 play 组成
  - play 中 hosts，variables，roles，tasks 等对象的表示方法都是键值中间以 ": " 分隔表示
  - YAML 还有一个小的怪癖. 所有的 YAML 文件开始行都应该是 ---. 这是 YAML 格式的一部分, 表明一个文件的开始

知识讲解

playbook语法基础

- playbook 构成
  - Target： 定义将要执行 playbook 的远程主机组
  - Variable： 定义 playbook 运行时需要使用的变量
  - Tasks： 定义将要在远程主机上执行的任务列表
  - Handler： 定义 task 执行完成以后需要调用的任务

知识讲解

playbook语法基础

知识讲解

- Playbook执行结果
- 使用 ansible-playbook 运行playbook文件，得到输出内容为 JSON 格式。并且由不同颜色组成，便于识别。一般而言

- 绿色代表执行成功
- ***代表系统代表系统状态发生改变
- 红色代表执行失败

playbook语法基础

- 第一个playbook

```
---                                    # 第一行，表示开始
- hosts: all
  remote_user: root
  tasks:
    - ping:

ansible-playbook  myping.yml  -f  5
```

  - -f 并发进程数量，默认是 5
  - hosts 行的内容是一个或多个组或主机的 patterns，以逗号为分隔符
  - remote_user 就是账户名

知识讲解

playbook语法基础

- 续 … …
  - tasks
  - 每一个 play 包含了一个 task 列表（任务列表）.
  - 一个 task 在其所对应的所有主机上（通过 host pattern 匹配的所有主机）执行完毕之后，下一个 task 才会执行.
  - 有一点需要明白的是（很重要），在一个 play 之中，所有 hosts 会获取相同的任务指令，这是 play 的一个目的所在，也就是将一组选出的 hosts 映射到 task，执行相同的操作

知识讲解

playbook语法基础

- playbook 执行命令
  - 给所有主机添加用户 plj，设置默认密码 123456
  - 要求第一次登录修改密码

```
---
- hosts: all
  remote_user: root
  tasks:
    - name: create user plj
      user: group=wheel uid=1000 name=plj
    - shell: echo 123456 | passwd --stdin plj
    - shell: chage -d 0 plj
```

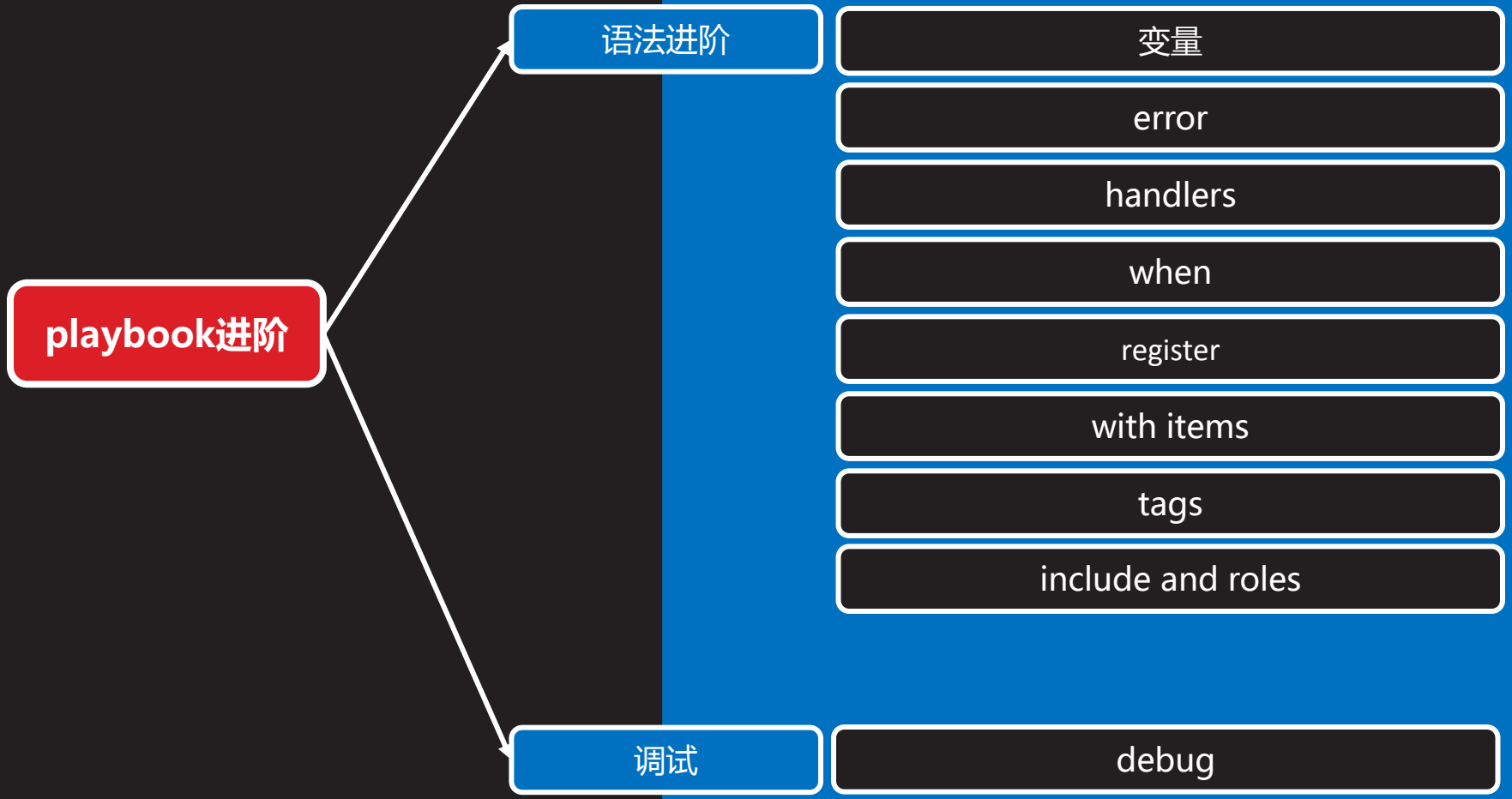
知识讲解

playbook练习

- 编写 playbook 实现以下效果
  - 安装 apache
  - 修改 apache 监听的端口为 8080
  - 为 apache 增加 NameServer 配置
  - 设置默认主页 hello world
  - 启动服务
  - 设置开机自启动

课堂练习

playbook进阶


playbook进阶
语法进阶
变量
error
handlers
when
register
with items
tags
include and roles
调试
debug

Tedu.cn
达内教育

语法进阶

变量

- 添加用户
  - 给所有主机添加用户 plj，设置默认密码 123456
  - 要求第一次登录修改密码（使用变量）

```
---
- hosts: 192.168.1.16
  remote_user: root
  vars:
    username: plj
  tasks:
    - name: create user "{{username}}"
      user: group=wheel uid=1000 name={{username}}
    - shell: echo 123456 | passwd --stdin plj
    - shell: chage -d 0 {{username}}
```

知识讲解

变量

- 续 … …
  - 解决密码明文问题
  - user 模块的 password 为什么不能设置密码呢？
  - 经过测试发现，password 是把字符串直接写入 shadow，并没有改变，而 Linux 的 shadow 密码是经过加密的，所以不能使用
  - 解决方案：
  - 变量过滤器 password_hash
  - {{ 'urpassword' | password_hash('sha512')}}

知识讲解

变量

- 变量过滤器
  - 给所有主机添加用户 plj，设置默认密码 123456
  - 要求第一次登录修改密码（使用变量）

```
---
- hosts: 192.168.1.16
  remote_user: root
  vars:
    username: plj
  tasks:
    - name: create user "{{username}}"
      user: group=wheel uid=1000 password={{'123456' |
password_hash('sha512')}} name={{username}}
    - shell: chage -d 0 {{username}}
```

知识讲解

课堂练习

- 练习使用 user 模块添加用户
- 练习使用变量简化task，让 play 通用性更强
- 练习使用 过滤器

error

- ansible-playbook 对错误的处理
  - 默认情况判断 $?，如果 值 不为 0 就停止执行
  - 但某些情况我们需要忽略错误继续执行

```
---
- hosts: 192.168.1.16
  remote_user: root
  vars:
    username: plj
  tasks:
    - name: create user "{{username}}"
      user: group=wheel uid=1000
password={{'123456'|password_hash('sha512')}}
name={{username}}
    - shell: setenforce 0
    - shell: chage -d 0 {{username}}
```

error

- 续 … …
  - 我们要关闭 selinux，如果 selinux 已经是关闭的，返回 1 ，但我们的目的就是关闭，已经关闭不算错误，这个情况我们就需要忽略错误继续运行，忽略错误有两种方法
  - 第一种方式：

```
shell:  /usr/bin/somecommand  || /bin/true
```

  - 第二种方式：

```
- name: run  some command
  shell: /usr/bin/somecommand
  ignore_errors: True
```

知识讲解

error

- 完整 playbook

```
---
- hosts: 192.168.1.16
  remote_user: root
  vars:
    username: plj
  tasks:
    - name: create user "{{username}}"
      user: group=wheel uid=1000
password={{'123456'|password_hash('sha512')}}
name={{username}}
    - shell: setenforce 0
      ignore_errors: true
    - shell: chage -d 0 {{username}}
```

知识讲解

handlers

- 用于当关注的资源发生变化时采取一定的操作。
- "notify" 这个action可用于在每个play的最后被触发这样可以避免多次有改变发生时每次都执行指定的操作取而代之仅在所有的变化发生完成后一次性地执行指定操作。
- 在 notify 中列出的操作称为 handler 也即 notify 中调用 handler 中定义的操作

知识讲解

handlers

- 前面我们安装了 apache，很多情况是要修改 httpd 的配置文件的，修改配置文件以后要重新载入配置文件让服务生效
- 这时候，我们可以使用 handlers 来实现

```
handlers:
  - name: restart apache
    service: name=apache state=restarted
```

知识讲解

handlers

- 结合之前试验，完整 playbook

```
---
- hosts: 192.168.1.16
  remote_user: root
  tasks:
    - name: config httpd.conf
      copy: src=/root/playbook/httpd.conf
dest=/etc/httpd/conf/httpd.conf
      notify:
        - restart httpd
  handlers:
    - name: restart httpd
      service: name=httpd state=restarted
```

知识讲解

handlers

- 注意事项：
  - notify 调用的是 handler 段 name 定义的串，必须一致，否则达不到触发的效果
  - 多个 task 触发同一个 notify 的时候，同一个服务只会触发一次
  - notify 可以触发多个条件，在生产环境中往往涉及到某一个配置文件的改变要重启若干服务的场景，handler 用到这里非常适合.
  - 结合 vars 可以写出非常普适的服务管理脚本

知识讲解

课堂练习

- 使用 handlers 重写安装 apache 的 playbook

when

知识讲解

- 某些时候我们可能需要在满足特定的条件后在触发某一项操作，或在特定的条件下终止某个行为，这个时候我们就需要进行条件判断，when 正是解决这个问题的最佳选择，远程中的系统变量 facts 变量作为 when 的条件，这些 facts 我们可以通过 setup 模块查看
  - when 的样例：

```
tashs:
  - name: somecommand
    command: somecommand
    when: expr
```

when

- 一个使用 when 的例子

```
---
- 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"
```

知识讲解

register

- register
  - 有时候我们可能还需要更复杂的例子，比如判断前一个命令的执行结果，根据结果处理后面的操作，这时候我们就需要 register 模块来保存前一个命令的返回状态，在后面进行调用

```
- command:  test command
  register: result
- command:  run command
  when: result
```

知识讲解

register

- 变量注册
  - 例如我们需要判断 plj 这个用户是否存在
  - 如果存在我就修改密码，如果不存在就跳过

```
tasks:
    - shell: id {{username}}
      register: result
    - name: change "{{username}}" password
      user: password={{'12345678'|password_hash('sha512')}}
name={{username}}
      when: result
```

知识讲解

register

- 变量注册进阶
  - 我们还可以针对运行命令结果的返回值做判定
  - 当系统负载超过一定值的时候做特殊处理

```
---
- hosts: 192.168.1.16
  remote_user: root
  tasks:
    - shell: uptime |awk '{printf("%f\n",$(NF-2))}'
      register: result
    - shell: touch /tmp/isreboot
      when: result.stdout|float > 0.5
```

知识讲解

课堂练习

- 编写 playbook，把所有监听端口是 8080 的 apache服务器全部停止

with_items

- with_items 是 playbook 标准循环，最常用到的就是它，with_items 可以用于迭代一个列表或字典，通过{{ item }}获取每次迭代的值
  - 例如创建多个用户

```
---
- hosts: 192.168.1.16
  remote_user: root
  tasks:
    - name: add users
      user: group=wheel password={{'123456' |
password_hash('sha512')}} name={{item}}
      with_items: ["nb", "dd", "plj", "lx"]
```

知识讲解

with_items

- with_items进阶
  - 为不同用户定义不同组

```
---
- hosts: 192.168.1.16
  remote_user: root
  tasks:
    - name: add users
      user: group={{item.group}} password={{'123456' |
password_hash('sha512')}} name={{item.name}}
      with_items:
        - {name: 'nb',  group: 'root'}
        - {name: 'dd',  group: 'root'}
        - {name: 'plj', group: 'wheel'}
        - {name: 'lx',  group: 'wheel'}
```

知识讲解

with_nested

- 嵌套循环：

```
---
- hosts: 192.168.1.16
  remote_user: root
  vars:
    un: [a, b, c]
    id: [1, 2, 3]
  tasks:
    - name: add users
      shell: echo {{item}}
      with_nested:
        - "{{un}}"
        - "{{id}}"
```

知识讲解

tags

- tags：给指定的任务定义一个调用标识；
- 使用格式：
  - name: NAME
  - module: arguments
  - tags: TAG_ID
- playbook 调用方式
  - -t TAGS, --tags=TAGS
  - --skip-tags=SKIP_TAGS
  - --start-at-task=START_AT

知识讲解

tags

- tags样例:

```
vars:
  soft: httpd
 tasks:
  - name: install {{soft}}
    yum:  name={{soft}}
  - name: config httpd.conf
    copy: src=/root/playbook/httpd.conf
dest=/etc/httpd/conf/httpd.conf
  - name: config services
    service: enabled=yes state=restarted name={{soft}}
    tags: restartweb
```

- 调用方式

```
ansible-playbook i.yml --tags=restartweb
```

知识讲解

include and roles

- 我们在编写 playbook 的时候随着项目越来越大，playbook 也越来越复杂，修改起来也越来越麻烦。这时候可以把一些 play、task 或 handler 放到其他文件中，然后通过include指令包含进来是一个不错的选择

```
tasks:
   - include: tasks/setup.yml
   - include: tasks/users.yml user=plj #users.yml 中可以通过{{ user }}来使用这些变量

handlers:
  - include: handlers/handlers.yml
```

知识讲解

include and roles

- roles 像是加强版的 include，他可以引入一个项目的文件和目录
- 一般所需的目录层级有
  - vars　变量层
  - tasks　任务层
  - handlers 触发条件
  - files　文件
  - template 模板
  - default　默认，优先级最低

知识讲解

include and roles

- 假如有一个play包含了一个叫 "x" 的role，则

```
---
- hosts: host_group
  roles:
    - x
```

  - x/tasks/main.yml
  - x/vars/main.yml
  - x/handler/main.yml
  - x/... .../main.yml
  - 都会自动添加进这个 play

Tedu.cn
达内教育

调试

debug

- 对于 python 语法不熟悉的同学，playbook 书写起来容易出错，且排错困难，这里介绍几种简单的排错调试方法
  - 检测语法
    ansible-playbook --syntax-check playbook.yaml
  - 测试运行
    ansible-playbook -C playbook.yaml
  - 显示收到影响到主机 --list-hosts
  - 显示工作的 task --list-tasks
  - 显示将要运行的 tag --list-tags

debug

- debug 模块可以在运行时输出更为详细的信息，来帮助我们排错，debug 使用样例：

```
---
- hosts: 192.168.1.16
  remote_user: root
  tasks:
    - shell: uptime |awk '{printf("%f\n",$(NF-2))}'
      register: result
    - shell: touch /tmp/isreboot
      when: result.stdout|float > 0.5
    - name: Show debug info
      debug: var=result
```

知识讲解

总结和答疑