YAML

> YAML目前广泛的运用于编写配置文件，它非常简洁和强大，远比json格式方便。
>
> YAML 的语法和其他高级语言类似，并且可以简单表达清单、散列表，标量等数据形态。它使用空白符号缩进和大量依赖外观的特色，特别适合用来表达或编辑数据结构、各种配置文件、倾印调试内容、文件大纲（例如：许多电子邮件标题格式和YAML非常接近）。

YAML和XML、JSON的区别

XML是指可扩展标记语言

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
        <people>
                <firstName>Brett</firstName>
                <lastName>McLaughlin</lastName>
        </people>
        <people>
                <firstName>Jason</firstName>
                <lastName>Hunter</lastName>
        </people>
```

```
</root>
```

它的格式非常严谨，就像html一样，有了标签头，必须有标签尾，因为严谨，所以导致了它非常繁琐，10年前的互联网行业制定的协议全是使用XML的，一个接口的入参使用XML来编写，真的是看的头都大了。

后来有了json，它是一种轻量级的数据交互格式。

```
{
    "people":[
        {
            "firstName":"Brett",
            "lastName":"McLaughlin"
        },
        {
            "firstName":"Jason",
            "lastName":"Hunter"
        }
    ]
}
```

它的优点就是简洁和清晰的层次结构，使得它成为了我们现在最重要的数据交互语言。那它的缺点就是，当我们的数据中有很多嵌套结构的时候，json看起来就会比较深，我们要找某一个值，可能会需要先去找它的对应的关系再去找值。

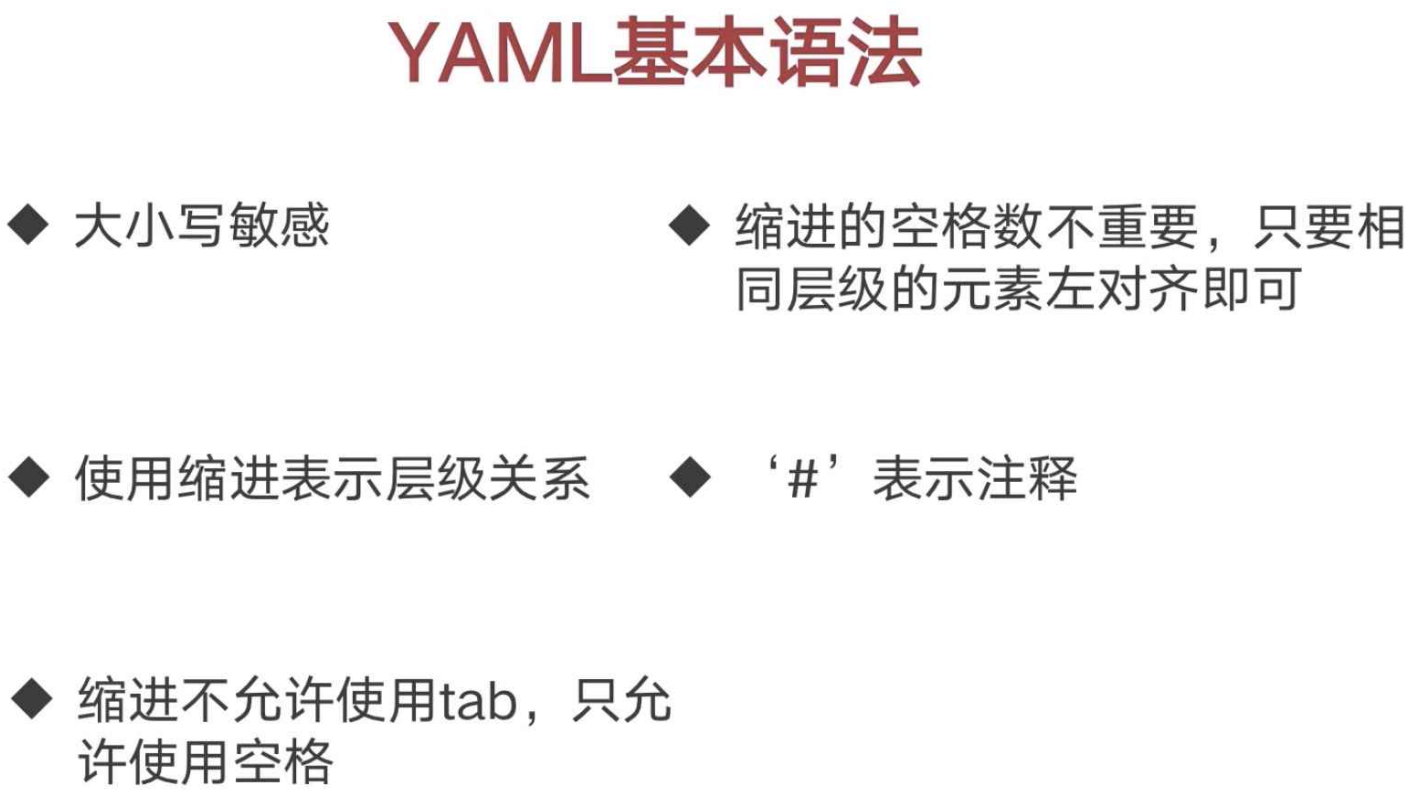
同样的数据，使用yaml，数据的层次就不会显得特别深，yaml会显得数据很美观和清晰

```
people:
  -
    firstName: Brett
    lastName: McLaughlin
  -
    firstName: Jason
    lastName: Hunter
```

YAML基本语法

- 大小写敏感
- 使用缩进表示层级关系

- 缩进不允许使用tab，只允许空格
- 缩进的空格数不重要，只要相同层级的元素左对齐即可
- '#'表示注释 (可以注释)



js-yaml网站地址：http://nodeca.github.io/js-yaml/

YAML数据类型

- 纯量（scalars）：单个的、不可再分的值-
- 数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）
- 对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes） / 字典（dictionary）

YAML数据类型

- 纯量：单个的、不可再分的值
- 数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）
- 对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes）/ 字典（dictionary）

YAML 纯量

> 纯量表示最基本的、不可再分的值

- 字符串
  - 双引号不会对特殊字符转义
  - 单引号之中如果还有单引号，必须连续使用两个单引号转义
  - 强制数据类型转换
- 布尔值
- 整数
- 浮点数
- Null
- 时间
- 日期

```yaml
# 字符串
username: test
# 单引号和双引号都可以使用，双引号不会对特殊字符转义
username2: 'test'
username3: "test"
```

```yaml
username4: 'test\ntest'
username5: "test\ntest"
# 单引号之中如果还有单引号，必须连续使用两个单引号转义。
username6: 'start ''mid'' end'
username7: "start 'mid' end"
# 字符串可以写成多行，从第二行开始，必须有一个单空格缩进。换行符会被转为空格。
william_str: 这是一行
  多行
  字符串
# 多行字符串可以使用|保留换行符，也可以使用>折叠换行。
william_n1: |
  FOO
  Bar
william_n2: >
  Foo
  Bar
# 布尔值
isAdmin: true
# 整数
bookNumbers: 10
# 浮点数
cash: 9.9
cash2: 0.099e+2
# null ~ 也表示空
lock: null
lock2: ~
# 时间 使用ISO 8601格式，时间和日期之间使用T连接，最后使用+代表时区
datetime: 2022-08-18T12:00:00+08:00
# 日期 使用ISO 8601格式，即yyyy-MM-dd
date: 2022-08-18
# yaml允许使用两个感叹号来强制转换数据类型
forceStr: !!str 123
forceStrBool: !!str true
```

YAML 数组

> 以-开头的行表示构成一个数组
>
> 行内表示法

```yaml
# 数组 '-'开头
myArray:
  - 12345
  - football
```

```yaml
# 数组 行内写法
myList: [ '12345',~ ]
```

YAML-数组

◆ 以' - '开头的行表示构成一个数组

◆ 使用行内表示法

YAML 对象

> : 和 空格 缩进定义
>
> 将所有键值对写成一个行内对象

```yaml
myObject:
  username: wangzi
  age: 25
  male: true
  birthday: 1996-11-06

  myObject1: { username: wangzi, age: 25, male: true, birthday: 1996-11-06 }
```

YAML-对象

◆ 键值对

◆ 将所有键值对写成一个行内对象

YAML 引用

> 锚点&和别名*
>
> &用来建立锚点，<<表示合并到当前数据，*用来引用锚点

```yaml
father: &father_info
  name: jordan
  age: 52

son:
  father: *father_info
  name: swift
  age: 25
```

```yaml
father: &father_last_name
  lastName: last

son:
  age: 18
  <<: *father_last_name
  firstName: first
```

Python 解析 YAML

```python
#!/usr/bin/python
# -*- coding: UTF-8 -*-

"""
@author:wangzichen
@file:yaml_load.py
"""
from pprint import pprint

import yaml

with open("test_yaml.yaml", "r", encoding="utf-8") as f:
    res = yaml.load(stream=f, Loader=yaml.FullLoader)

if __name__ == '__main__':
    print(type(res))
    pprint(res)
```