6、Yaml技术

1、Yaml基本语法

1. 基本规则

> - 大小写敏感
> - 使用缩进表示层级关系
> - 缩进时不允许使用Tab，只允许使用空格
> - 缩进的空格数目不重要，只要相同层级的元素左对齐即可
> - `#` 表示注释，从它开始到行尾都被忽略

2. yaml转字典

```yaml
# 下面格式读到Python里会是个dict
name: 灰蓝
age: 0
job: Tester
```

输出：

```
{'job': 'Tester', 'age': 0, 'name': u'\u7070\u84dd'}
```

3. yaml转列表

yaml中支持列表或数组的表示，如下：

```
# 下面格式读到Python里会是个list
- 灰蓝
- 0
- Tester
```

输出：

```
[u'\u7070\u84dd', 0, 'Tester']
```

4. 复合结构

字典和列表可以复合起来使用，如下：

```
# 下面格式读到Python里是个list里包含dict
- name: 灰蓝
  age: 0
  job: Tester
- name: James
  age: 30
```

输出：

```
[{'job': 'Tester', 'age': 0, 'name':
u'\u7070\u84dd'}, {'age': 30, 'name':
'James'}]
```

5. 基本类型

yaml中有以下基本类型：

- 字符串
- 整型
- 浮点型
- 布尔型
- null
- 时间
- 日期

我们写个例子来看下：

```yaml
# 这个例子输出一个字典，其中value包括所有基本类型
str: "Hello World!"
int: 110
float: 3.141
boolean: true  # or false
None: null  # 也可以用 ~ 号来表示 null
time: 2016-09-22t11:43:30.20+08:00  # ISO8601，写法百度
date: 2016-09-22  # 同样ISO8601
```


输出：

```
{'date': datetime.date(2016, 9, 22),
'None': None, 'boolean': True, 'str':
'Hello World!', 'time':
datetime.datetime(2016, 9, 22, 3, 43, 30,
200000), 'int': 110, 'float': 3.141}
```

如果字符串没有空格或特殊字符，不需要加引号，但如果其中有空格或特殊字符，则需要加引号了

```
str: 灰蓝
str1: "Hello World"
str2: "Hello\nWorld"
```

输出：

```
{'str2': 'Hello\nWorld', 'str1': 'Hello
World', 'str': u'\u7070\u84dd'}
```

这里要注意单引号和双引号的区别，单引号中的特殊字符转到Python会被转义，也就是到最后是原样输出了，双引号不会被Python转义，到最后是输出了特殊字符；可能比较拗口，来个例子理解下：

```
str1: 'Hello\nWorld'
str2: "Hello\nWorld"
```

```
# -*- coding: utf-8 -*-
import yaml

y = yaml.load(file('test.yaml', 'r'))
print y['str1']
print y['str2']
```

输出：

```
Hello\nWorld
Hello
World
```

可以看到，单引号中的'\n'最后是输出了，双引号中的'\n'最后是转义成了回车

字符串处理中写成多行、'|'、'>'、'+'、'-'的意义这里就不讲了。

6. 引用

`&` 和 `*` 用于引用

```
name: &name 灰蓝
tester: *name
```

这个相当于一下脚本：

```
name: 灰蓝
tester: 灰蓝
```

输出：

```
{'name': u'\u7070\u84dd', 'tester':
u'\u7070\u84dd'}
```

7. 强制转换

yaml是可以进行强制转换的，用 `!!` 实现，如下：

```
str: !!str 3.14
int: !!int "123"
```

输出：

```
{'int': 123, 'str': '3.14'}
```

明显能够看出123被强转成了int类型，而float型的3.14则被强转成了str型。另外PyYaml还支持转换成Python/object类型，这个我们下面再讨论。

8. 分段

在同一个yaml文件中，可以用 `---` 来分段，这样可以将多个文档写在一个文件中

```
---
name: James
age: 20
---
name: Lily
age: 19
```

这时候我们就得用到我们的 `load_all()` 方法出场了，`load_all()` 方法会生成一个迭代器，可以用for输出出来：

```
# -*- coding: utf-8 -*-
import yaml

ys = yaml.load_all(file('test.yaml', 'r'))
for y in ys:
    print y
```


输出：

```
{'age': 20, 'name': 'James'}
{'age': 19, 'name': 'Lily'}
```

对应的也有 `dump_all()` 方法，一个意思，就是将多个段输出到一个文件中，举个栗子：

```yaml
# -*- coding: utf-8 -*-
import yaml

obj1 = {"name": "James", "age": 20}
obj2 = ["Lily", 19]

with open('test.yaml', 'w') as f:
    yaml.dump_all([obj1, obj2], f)
```

打开test.yaml看看：

```yaml
{age: 20, name: James}
--- [Lily, 19]
```


`dump()` 和 `dump_all()` 方法可以传入列表，也可以传入一个可序列化生成器，如 `range(10)`， 如下：

```yaml
# -*- coding: utf-8 -*-
import yaml
fo = open('../config/test.yaml','w',encoding='utf-8')
yaml.dump(range(10),fo)

```

输出：

```yaml
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

在 `dump` 和 `dump_all()` 的时候还可以配一堆参数

目标：了解Yaml基本使用语法

内容：

小结：课后去实现下yaml作为配置文件的用法

2、Yaml操作

1、yaml使用场景

- 配置文件
- 测试用例

2、yaml语法

- 字典
- 列表
- 嵌套
- 注释
- & * 变量操作
- 多用例
- 空格与颜色变化

```yaml
- #test4
  url: /api/mgr/loginReq
  method: post
  detail: 不传账号，不传密码
  data:
    username: "" #如果不填  None
    password: ""
  check:
    retcode: 1
    reason: 用户或者密码错误
```

3、yaml里引用yaml文件

问题并没有要求Python解决方案，但下面是一个使用 PyYAML...

PyYAML允许您附加自定义构造函数(如`!include`)到YAML加载程序。

基于类的解决方案

这里有一个基于类的解决方案，它避免了我最初响应的全局根变量。

使用元类注册自定义构造函数：

```
import yaml
import os.path
class Loader(yaml.Loader):#继承
    def __init__(self, stream):
        self._root = os.path.split(stream.name)[0]
        super(Loader, self).__init__(stream)
    def include(self, node):
        filename = os.path.join(self._root, self.construct_scalar(node))
        with open(filename, 'r') as f:
            return yaml.load(f, Loader)
Loader.add_constructor('!include', Loader.include)
```

例如:

a.yaml

```
a: 1
b:
    - 2
    - 3
c: !include b.yaml
```

`b.yaml`

```
- 10
- [100, 200, 300]
```

现在可以使用以下方法加载文件：

```
with open('a.yaml', 'r') as f:
    data = yaml.load(f, Loader)
    print(data)
{'a': 1, 'b': [2, 3], 'c': [10, [100, 200, 300]]}
```

windows环境下批处理文件方案

```
@echo off
echo 松勤-教管系统接口自动化运行准备开始.....
@echo on



del /f /s /q G:\SongQin\Python\Demo\teach_sq\report\tmp\*.json
del /f /s /q G:\SongQin\Python\Demo\teach_sq\report\tmp\*.jpg
del /f /s /q G:\SongQin\Python\Demo\teach_sq\report\report



```

```
@echo off
echo 环境文件删除工作完成，开始运行脚本......
@echo on


cd G:/SongQin/Python/Demo/teach_sq/test_case
pytest  -sq --alluredir=../report/tmp

allure serve ../report/tmp


@echo off
echo 接口自动化运行成功
pause
```


运行模式

- 1- 使用 终端(win/linux)运行： python -m run.py
- 2- 使用执行文件运行： run.bat(Win) ; run.sh(Linux) 运行指令 ./run.sh
- 3- jenkins运行 会使用自带的sh 功能运行

----------本次课程任务----------:

完成Yaml基本操作、Yaml用例执行--不需要提交

4、获取项目工程路径

```
"""
在一些代码里使用相对路径会报文件找不到!
../data/xxxxxx
解决方案:
    通过代码自动获取当前运行项目的路径:
"""
import os  #
print(__file__)#当前运行文件的路径
print(os.path.realpath(__file__))#当前运行文件的绝对路径
project_path=os.path.split(os.path.realpath(__file__))[0].split('configs')[0]
print(project_path)#项目路径
```


5、Allure报告优化

1、浏览器打开allure报告

建议使用火狐浏览器- 谷歌是loading和404 不要用chrome,ie浏览器打开

2、定制化标签

> @allure.epic("外卖系统")
> @allure.feature("商铺模块")
> @allure.tag("核心关注")

allure用例描述

使用方法	参数值	参数说明
@allure.epic()	epic描述	敏捷里面的概念，定义史诗，往下是feature
@allure.feature()	模块名称	功能点的描述，往下是story
@allure.story()	用户故事	用户故事，往下是title
@allure.title(用例的标题)	用例的标题	重命名html报告名称
@allure.testcase()	测试用例的链接地址	对应功能测试用例系统里面的case
@allure.issue()	缺陷	对应缺陷管理系统里面的链接
@allure.description()	用例描述	测试用例的描述
@allure.step()	操作步骤	测试用例的步骤
@allure.severity()	用例等级	blocker，critical，normal，minor，trivial

使用方法	参数值	参数说明
@allure.link()	链接	定义一个链接，在测试报告展现
@allure.attachment()	附件	报告添加附件

```python
import pytest
import allure
@allure.feature('这里是一级标签')
class TestAllure():
    @allure.title("用例标题0")
    @allure.story("这里是第一个二级标签")
    @allure.title("用例标题1")
    @allure.story("这里是第二个二级标签")
    def test_1(self):

 allure.attach.file(r'E:\Myproject\pytest-allure\test\test_1.jpg', '我是附件截图的名字',
attachment_type=allure.attachment_type.JPG)

    @allure.title("用例标题2")
    @allure.story("这里是第三个二级标签")

@allure.severity("critical")
@allure.description("这里只是做一个web ui自动化的截图效果")
```

3、设置用例级别

```
pytest  -sq --alluredir=../report/tmp --allure-severities=normal,critical
```

```
import pytest
import allure

'''
@allure.severity装饰器按严重性级别来标记case
执行指定测试用例 --allure-severities blocker
BLOCKER = 'blocker'    阻塞缺陷
CRITICAL = 'critical'  严重缺陷
NORMAL = 'normal'      一般缺陷
MINOR = 'minor'        次要缺陷
TRIVIAL = 'trivial'    轻微缺陷
'''


@allure.severity("normal")
def test_case_1():
    '''修改个人信息-sex参数为空'''
    print("test case 11111111")


@allure.severity("critical")
def test_case_2():
```

```
23      '''修改个人信息-sex参数传F和M两种类型，成功(枚举类型)'''
24      print("test case 22222222")
25
26
27  @allure.severity("critical")
28  def test_case_3():
29      '''修改个人信息-修改不是本人的用户信息，无权限操作'''
30      print("test case 333333333")
31
32  @allure.severity("blocker")
33  def test_case_4():
34      '''修改个人信息-修改自己的个人信息，修改成功'''
35      print("test case 4444444")
36
37
38  def test_case_5():
39      '''没标记severity的用例默认为normal'''
40      print("test case 5555555555")
```


4、设置allure显示环境

在Allure报告中添加环境信息，通过创建environment.properties或者environment.xml文件，并把文件存放到allure-results(这个目录是生成最后的html报告之前，生成依赖文件的目录)目录下

environment.properties

```
Browser=Firefox
Browser.Version=77
Stand=songqin_teach
ApiUrl=127.0.0.1/login
python.Version=3.6
```