今日学习目标

能够借助测试套件生成测试报告

掌握如何使用logging实现日志收集

能够使用jsonschema库对响应数据进行全量字段校验

获取请求头

1. 在 common/ 下 创建 get_header.py 文件
2. 在 文件内 创建 get_header() 函数，实现 登录成功，获取令牌，拼接成 请求头，返回。
3. 在 scripts/ 的测试脚本文件中，添加 setUpClass 方法，调用 get_header() 函数。 将返回值 保存到 类属性上
4. 在 使用 请求头的位置，直接从类属性获取

```
# 在 common/ 下 创建 get_header.py 文件 实现 get_header 函数
import requests

def get_header():
    url = "http://ihrm-test.itheima.net/api/sys/login"
    data = {"mobile": "13800000002", "password": "123456"}
    resp = requests.post(url=url, json=data)
    print(resp.json())
    # 从 响应体中，获取 data的值
    token = resp.json().get("data")

    header = {"Content-Type": "application/json",
              "Authorization": "Bearer " + token}
    return header
----------------------------
# 在 scripts/ 的测试脚本文件中，添加 setUpClass 方法，调用 get_header 函数。 将返回值 保存到 类属性上
from common.get_header import get_header

class TestEmpAdd(unittest.TestCase):
    # 类属性
    header = None

    @classmethod
    def setUpClass(cls) -> None:
        cls.header = get_header()
```

```
----------------------------
# 在 使用 请求头的位置，直接从类属性获取
   resp = IhrmEmpCURD.add_emp(self.header, json_data)
```

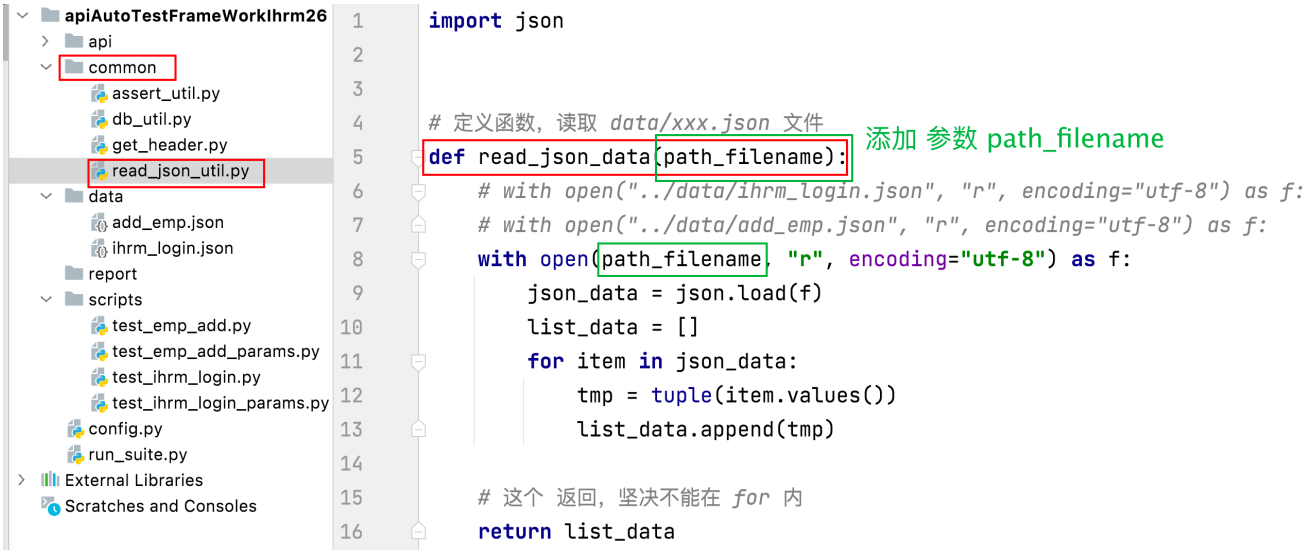
提取项目目录

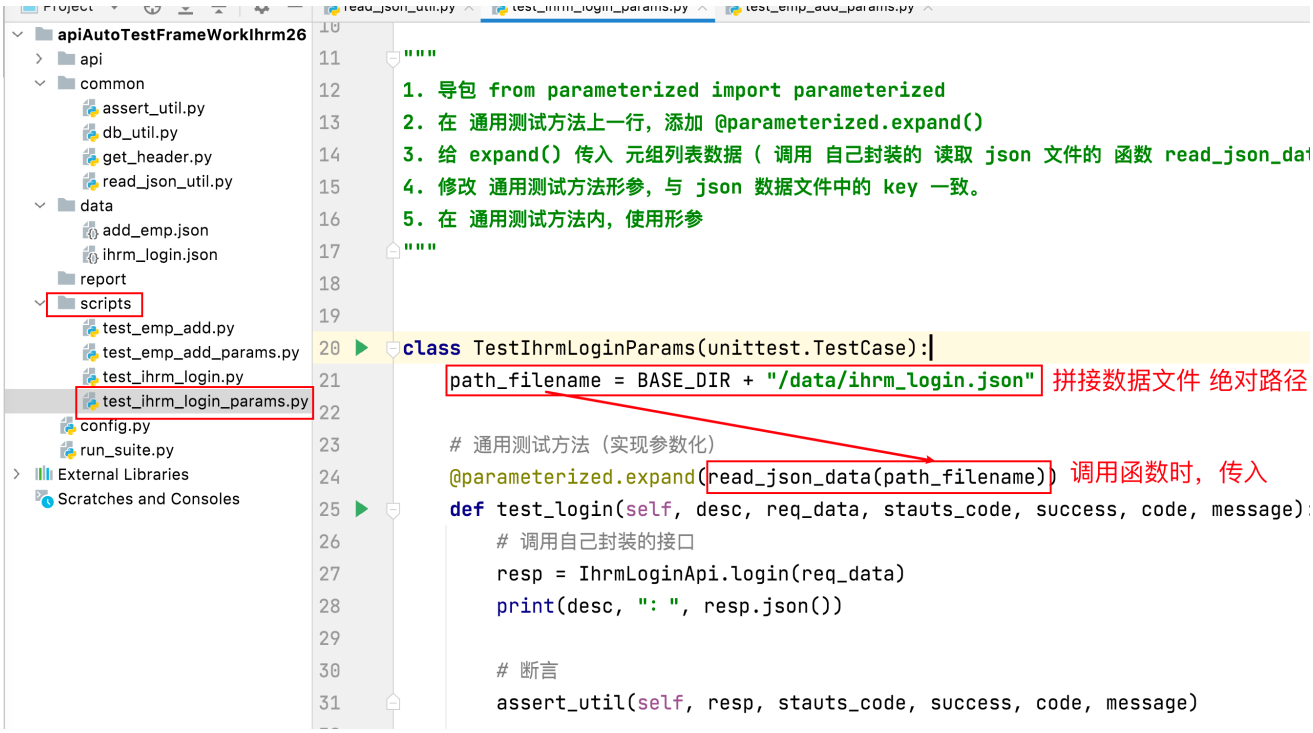
相关知识：

- `__file__`: 获取 当前文件的 绝对路径。
- `BASE_DIR = os.path.dirname(__file__)`: 获取 到 当前文件的 上一级目录。
  - 此行代码,写在 config.py 中， 可以直接获取 项目目录

项目中使用：

1. 在 config.py 文件中，添加 获取项目路径 全局变量 `BASE_DIR = os.path.dirname(__file__)`
2. 修改 common/ 下 read_json_util.py 文件中，读取 json 文件 函数read_json_data()，添加 参数 path_filename
3. 在 使用 read_json_data()函数 时， 拼接 json 文件路径， 传入到 函数中。





==生成测试报告==

步骤：

1. 创建测试套件实例。 suite
2. 添加 测试类
3. 创建 HTMLTestReport 类实例。 runner

4. runner 调用 run(), 传入 suite

实现：

```
import unittest

from config import BASE_DIR
from scripts.test_emp_add_params import TestEmpAddParams
from scripts.test_ihrm_login_params import TestIhrmLoginParams

from htmltestreport import HTMLTestReport

# 1. 创建测试套件实例。 suite
suite = unittest.TestSuite()

# 2. 添加 测试类，组装测试用例
suite.addTest(unittest.makeSuite(TestIhrmLoginParams))
suite.addTest(unittest.makeSuite(TestEmpAddParams))

# 3. 创建 HTMLTestReport 类实例。 runner
# runner = HTMLTestReport(BASE_DIR + "/report/ihrm.html")  # 绝对路径
runner = HTMLTestReport("./report/ihrm.html", description="描述", title="标题")  # 相对路径

# 4. runner 调用 run(), 传入 suite
runner.run(suite)
```

日志收集

==日志简介==

- 什么是日志
  - 日志也叫 log，通常对应的 xxx.log 的日志文件。文件的作用是记录系统运行过程中，产生的信息。
- 搜集日志的作用
  - 查看系统运行是否正常。
  - 分析、定位 bug。

日志的级别

- logging.DEBUG：调试级别【高】
- logging.INFO：信息级别【次高】
- logging.WARNING：警告级别【中】
- logging.ERROR：错误级别【低】
- logging.CRITICAL：严重错误级别【极低】

特性：

- 日志级别设定后，只有比该级别低的日志会写入日志。

- 如：设定日志级别为 info。 debug 级别的日志信息，不会写入。infowarning、error、critical 会写入

高

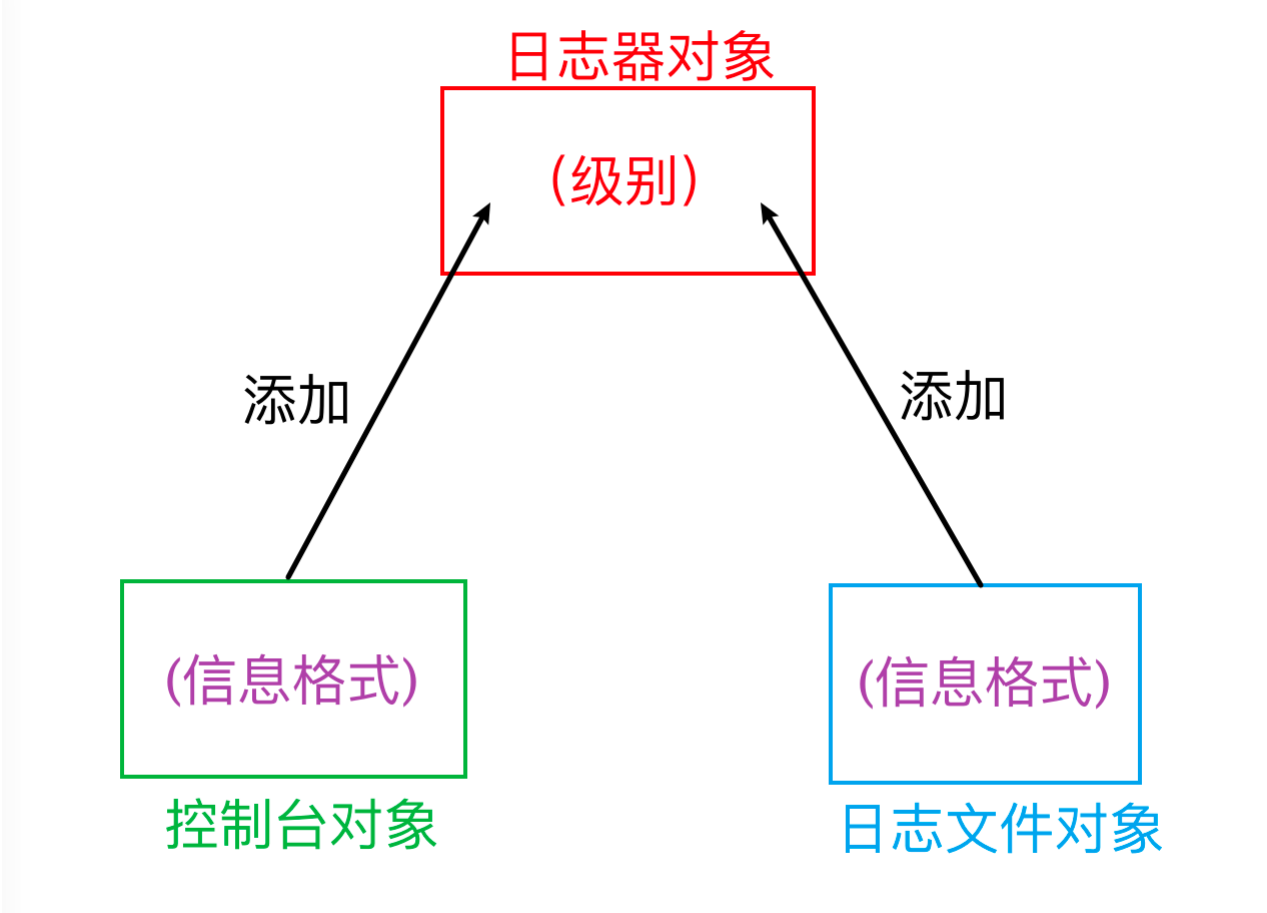
低

日志级别	描述
DEBUG	调试级别，打印非常详细的日志信息，通常用于代码调试
INFO	信息级别，一般用于记录突出强调的运行过程步骤
WARNING	警告级别，可能出现潜在错误的情况，一般不影响系统使用
ERROR	错误级别，打印错误异常信息，出现BUG
CRITICAL	严重错误级别，系统可能已经无法运行

日志代码实现分析

==日志代码，无需手写实现。会修改、调用即可！==

代码分析




```
"""
步骤：
```

```
# 0. 导包
# 1. 创建日志器对象
# 2. 设置日志打印级别
    # logging.DEBUG 调试级别
    # logging.INFO 信息级别
    # logging.WARNING 警告级别
    # logging.ERROR 错误级别
    # logging.CRITICAL 严重错误级别
# 3. 创建处理器对象
    # 创建 输出到控制台 处理器对象
    # 创建 输出到日志文件 处理器对象
# 4. 创建日志信息格式
# 5. 将日志信息格式设置给处理器
    # 设置给 控制台处理器
    # 设置给 日志文件处理器
# 6. 给日志器添加处理器
    # 给日志对象 添加 控制台处理器
    # 给日志对象 添加 日志文件处理器
# 7. 打印日志
"""

import logging.handlers
import logging
import time

# 1. 创建日志器对象
logger = logging.getLogger()

# 2. 设置日志打印级别
logger.setLevel(logging.DEBUG)
# logging.DEBUG 调试级别
# logging.INFO 信息级别
# logging.WARNING 警告级别
# logging.ERROR 错误级别
# logging.CRITICAL 严重错误级别

# 3.1 创建 输出到控制台 处理器对象
st = logging.StreamHandler()
# 3.2 创建 输出到日志文件 处理器对象
fh = logging.handlers.TimedRotatingFileHandler('a.log', when='midnight', interval=1,
                                               backupCount=3, encoding='utf-8')
# when 字符串，指定日志切分间隔时间的单位。midnight：凌晨：12点。
# interval 是间隔时间单位的个数，指等待多少个 when 后继续进行日志记录
# backupCount 是保留日志文件的个数

# 4. 创建日志信息格式
fmt = "%(asctime)s %(levelname)s [%(filename)s(%(funcName)s:%(lineno)d)] - %(message)s"
formatter = logging.Formatter(fmt)

# 5.1 日志信息格式 设置给 控制台处理器
st.setFormatter(formatter)
# 5.2 日志信息格式 设置给 日志文件处理器

fh.setFormatter(formatter)
```

```
# 6.1 给日志器对象 添加 控制台处理器
logger.addHandler(st)
# 6.2 给日志器对象 添加 日志文件处理器
logger.addHandler(fh)

# 7. 打印日志
while True:
    # logging.debug('我是一个调试级别的日志')
    # logging.info('我是一个信息级别的日志')
    logging.warning('test log sh-26')
    # logging.error('我是一个错误级别的日志')
    # logging.critical('我是一个严重错误级别的日志')
    time.sleep(1)
```

==日志使用==

可修改的位置

```
def init_log_config(filename, when='midnight', interval=1, backup_count=7):
    """
    功能：初始化日志配置函数
    :param filename: 日志文件名
    :param when: 设定日志切分的间隔时间单位
    :param interval: 间隔时间单位的个数，指等待多少个 when 后继续进行日志记录
    :param backup_count: 保留日志文件的个数
    :return:
    """
```

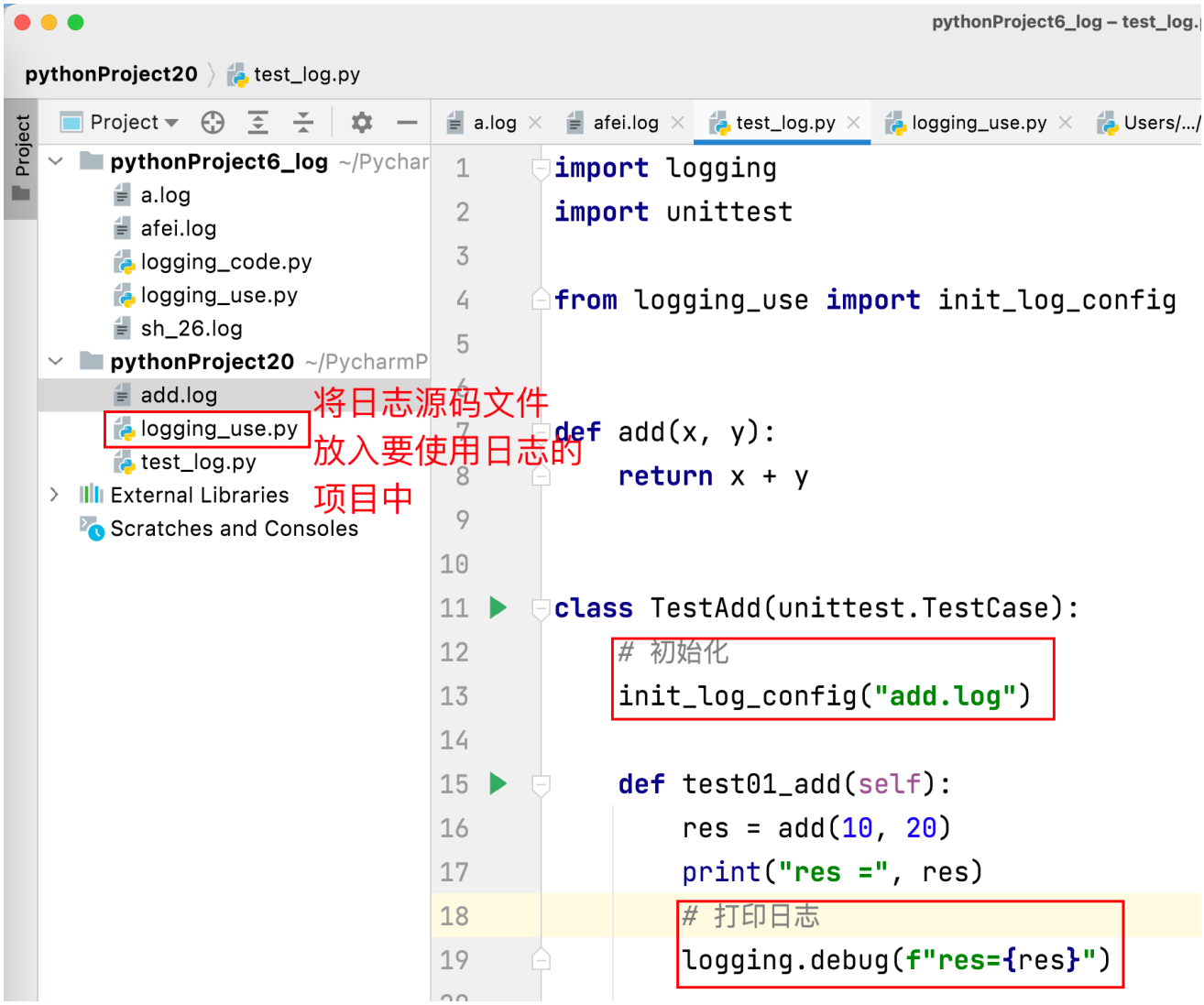
文件名　日志间隔时间单位　单位的个数　日志文件保留的个数

使用步骤：

1. 调用 init_log_config() 函数，初始化日志信息。
2. 指定 日志级别，打印 日志信息。

```
if __name__ == '__main__':
    # 初始化 日志
    init_log_config('sh_26.log', interval=3, backup_count=5)

    # 打印输出 日志信息
    # logging.warning('我是一个警告级别的日志')
    # logging.debug('xxxx - debug 日志信息')
    a = 9527
    logging.error(f'测试 error 级别日志信息 a={a}')
```



==全量字段校验==

简介和安装

- **概念**：校验接口返回响应结果的全部字段（更进一步的断言）
- **校验内容**：
  - 字段值
  - 字段名 或 字段类型
- **校验流程**：
  - 定义json语法校验格式
  - 比对接口实际响应数据是否符合json校验格式
- **安装jsonschema**：

```
pip install jsonschema -i https://pypi.douban.com/simple/
```

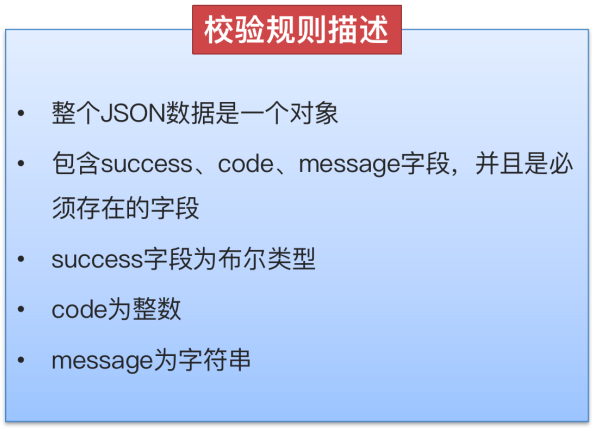
查验：

- pip 查验：pip list 或 pip show jsonschema
- pycharm 中 查验：file --- settings --- 项目名中查看 python解释器列表。

JSON Schema入门

入门案例

入门案例



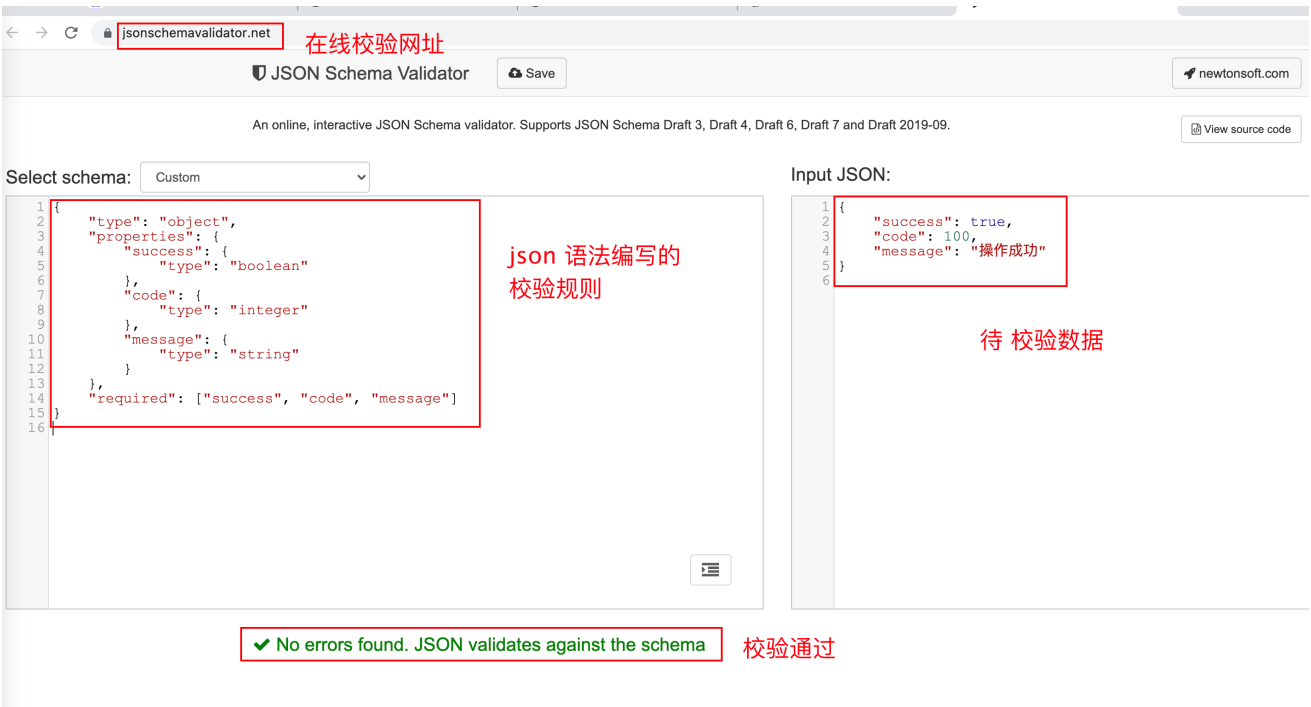

```
{
    "type": "object",
    "properties": {
        "success": {"type": "boolean"},
        "code": {"type": "integer"},
        "message": {"type": "string"}
    },
    "required": ["success", "code", "message"]
}
```

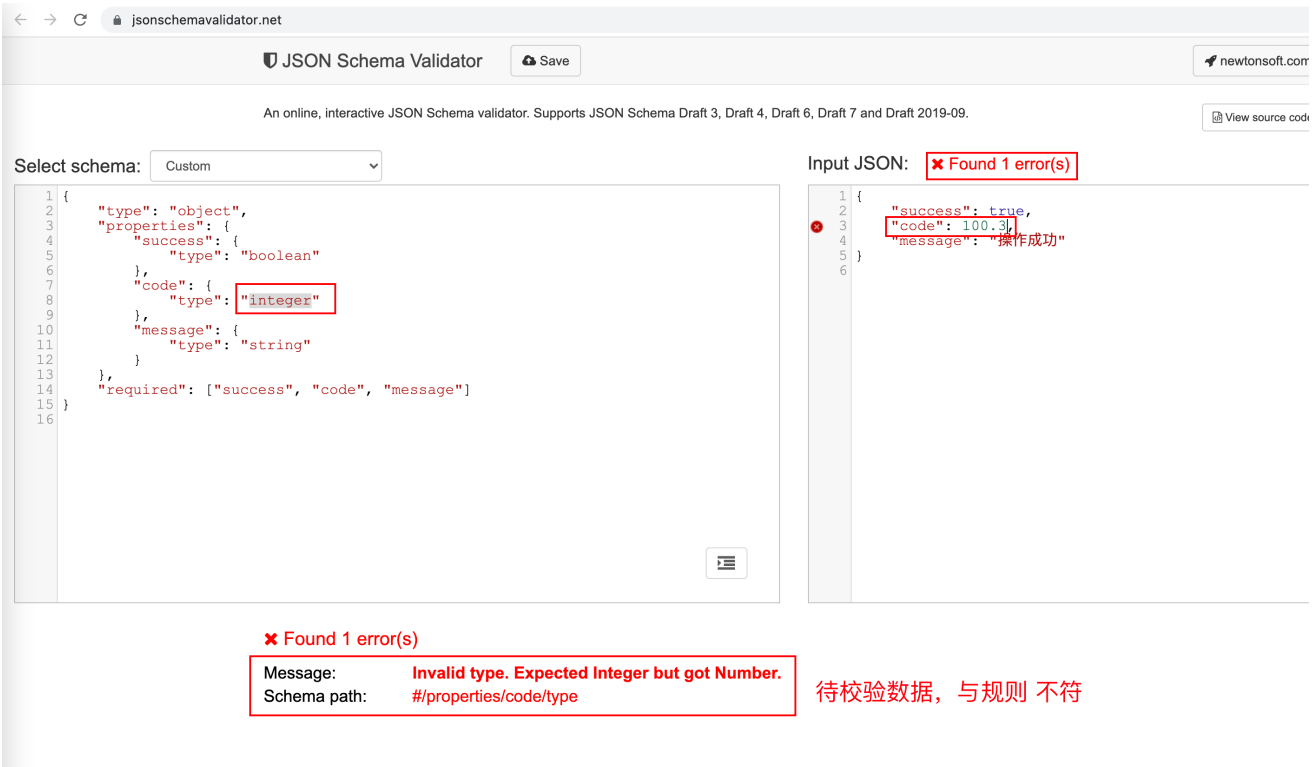
校验方式

在线工具校验

http://json-schema-validator.herokuapp.com

https://www.jsonschemavalidator.net 【推荐】





python代码校验

实现步骤：

1 导包 import jsonschema

2 定义 jsonschema格式 数据校验规则

3 调用 jsonschema.validate(instance="json数据", schema="jsonshema规则")

查验校验结果：

- 校验通过：返回 None

- 校验失败
  - schema 规则错误，返回 SchemaError
  - json 数据错误，返回 ValidationError

案例：

```
# 1. 导包
import jsonschema

# 2. 创建 校验规则
schema = {
    "type": "object",
    "properties": {
        "success": {
            "type": "boolean"
        },
        "code": {
            "type": "int"
        },
        "message": {
            "type": "string"
        }
    },
    "required": ["success", "code", "message"]
}

# 准备待校验数据
data = {
    "success": True,
    "code": 10000,
    "message": "操作成功"
}

# 3. 调用 validate 方法，实现校验
result = jsonschema.validate(instance=data, schema=schema)
print("result =", result)

# None: 代表校验通过
# ValidationError: 数据 与 校验规则不符
# SchemaError:  校验规则 语法有误
```

JSON Schema语法

关键字	描述
type	表示待校验元素的类型
properties	定义待校验的JSON对象中，各个key-value对中value的限制条件
required	定义待校验的JSON对象中，必须存在的key
const	JSON元素必须等于指定的内容
pattern	使用正则表达式约束字符串类型数据

type关键字

作用：约束数据类型

```
integer —— 整数
string —— 字符串
object —— 对象
array —— 数组  --> python: list 列表
number —— 整数/小数
null —— 空值   --> python: None
boolean —— 布尔值

语法:
{
    "type": "数据类型"
}
```

示例

```
import jsonschema

# 准备校验规则
schema = {
    "type": "object"   # 注意 type 和 后面的 类型，都要放到 ”“ 中!
}

# 准备数据
data = {"a": 1, "b": 2}

# 调用函数
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

properties关键字

说明：是 type关键字的辅助。用于 type 的值为 object 的场景。

作用：指定 **对象中** 每个字段的校验规则。 可以嵌套使用。

语法：

```
{
    "type": "object",
    "properties":{
        "字段名1":{规则},
        "字段名2":{规则},
        ......
    }
}
```

案例1：

```
{
    "success": true,
    "code": 10000,
    "message": "操作成功",
    "money": 6.66,
    "address": null,
    "data": {
        "name": "tom"
    },
    "luckyNumber": [6, 8, 9]
}
```

```
import jsonschema

# 准备校验规则
schema = {
    "type": "object",
    "properties": {
        "success": {"type": "boolean"},
        "code": {"type:": "integer"},
        "message": {"type": "string"},
        "money": {"type": "number"},
        "address": {"type": "null"},
        "data": {"type": "object"},
        "luckyNumber": {"type": "array"}
    }
}

# 准备测试数据
data = {
    "success": True,
    "code": 10000,
    "message": "操作成功",
    "money": 6.66,
    "address": None,
    "data": {
        "name": "tom"
    },
    "luckyNumber": [6, 8, 9]
```

```
}

# 调用方法进行校验
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

案例2：要求定义JSON对象中包含的所有字段及数据类型

```
data = {
    "success": True,
    "code": 10000,
    "message": "操作成功",
    "money": 6.66,
    "address": None,
    "data": {
        "name": "tom",
        "age": 18,
        "height": 1.78
    },
    "luckyNumber": [6, 8, 9]
}
```

```
import jsonschema

# 准备校验规则
schema = {
    "type": "object",
    "properties": {
        "success": {"type": "boolean"},
        "code": {"type:": "integer"},
        "message": {"type": "string"},
        "money": {"type": "number"},
        "address": {"type": "null"},
        "data": {
            "type": "object",
            "properties": {                  # 对 data 的对象值，进一步进行校验
                "name": {"type": "string"},
                "age": {"type": "integer"},
                "height": {"type": "number"}
            }
        },
        "luckyNumber": {"type": "array"}
    }
}

# 准备测试数据
data = {
    "success": True,
    "code": 10000,
```

```
    "message": "操作成功",
    "money": 6.66,
    "address": None,
    "data": {
        "name": "tom",
        "age": 18,
        "height": 1.78
    },
    "luckyNumber": [6, 8, 9]
}

# 调用方法进行校验
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

required关键字

作用：校验对象中必须存在的字段。字段名必须是字符串，且唯一

```
语法：
{
    "required": ["字段名1", "字段名2", ...]
}
```

```
import jsonschema

# 测试数据
data = {
    "success": True,
    "code": 10000,
    "message": "操作成功",
    "data": None,
}

# 校验规则
schema = {
    "type": "object",
    "required": ["success", "code", "message", "data"]
}

# 调用方法校验
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

const关键字

作用：校验字段值是一个固定值。

```
语法：
{
    "字段名": {"const": 具体值}
}
```

```
import jsonschema

# 测试数据
data = {
    "success": True,
    "code": 10000,
    "message": "操作成功",
    "data": None,
}

# 校验规则
schema = {
    "type": "object",
    "properties": {
        "success": {"const": True},
        "code": {"const": 10000},
        "message": {"const": "操作成功"},
        "data": {"const": None}
    },
    "required": ["success", "code", "message", "data"]
}

# 调用方法校验
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

pattern关键字

作用：指定正则表达式，对字符串进行模糊匹配

```
基础正则举例：
1 包含字符串：hello
2 以字符串开头 ^：^hello  如：hello,world
3 以字符串结尾 $：hello$  如：中国,hello
4 匹配[]内任意1个字符[]：[0-9]匹配任意一个数字  [a-z]匹配任意一个小写字母  [cjfew9823]匹配任意一个
5 匹配指定次数{}：[0-9]{11}匹配11位数字。

匹配 手机号：^[0-9]{11}$
```

语法：

```
{
    "字段名": {"pattern": "正则表达式"}
}
```

```
import jsonschema

# 测试数据
data = {
    "message": "!jeklff37294操作成功43289hke",
    "mobile": "15900000002"
}

# 校验规则
schema = {
    "type": "object",
    "properties": {
        "message": {"pattern": "操作成功"},
        "mobile": {"pattern": "^[0-9]{11}$"}
    }
}

# 调用方法校验
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

综合案例应用

```
# 测试数据
import jsonschema

data = {
    "success": False,
    "code": 10000,
    "message": "xxx登录成功",
    "data": {
        "age": 20,
        "name": "lily"
    }
}

# 校验规则
schema = {
    "type": "object",
    "properties": {
        "success": {"type": "boolean"},
        "code": {"type": "integer"},
```

```
        "message": {"pattern": "登录成功$"},
        "data": {
            "type": "object",
            "properties": {
                "name": {"const": "lily"},
                "age": {"const": 20}
            },
            "required": ["name", "age"]
        }
    },
    "required": ["success", "code", "message", "data"]
}

# 调用测试方法
res = jsonschema.validate(instance=data, schema=schema)
print(res)
```

作业

1. 复习当天课程内容，完成《接口测试-第09天-作业.md》中习题。