整体介绍

项目基于Python开发，使用Pytest测试框架，因此采用pytest，具备许多第三方插件，且兼容unittest以及其他框架下开发的测试脚本。

Pytest在不指定测试内容时，会收集当前文件夹以及子文件夹下所有以test*开头的py文件中的test*开头或者test结尾的函数作为测试对象进行收集，默认情况下会运行收集到的所有用例。

因此在本项目开发中，规定了:

1 所有测试用例均放在/TestCase/文件夹下，API文件夹对应为接口测试，DataBase文件夹对应数据库，UI对应UI测试

2 所有的测试文件均需要以 test_开头，被测试函数/类也需要遵守该协议

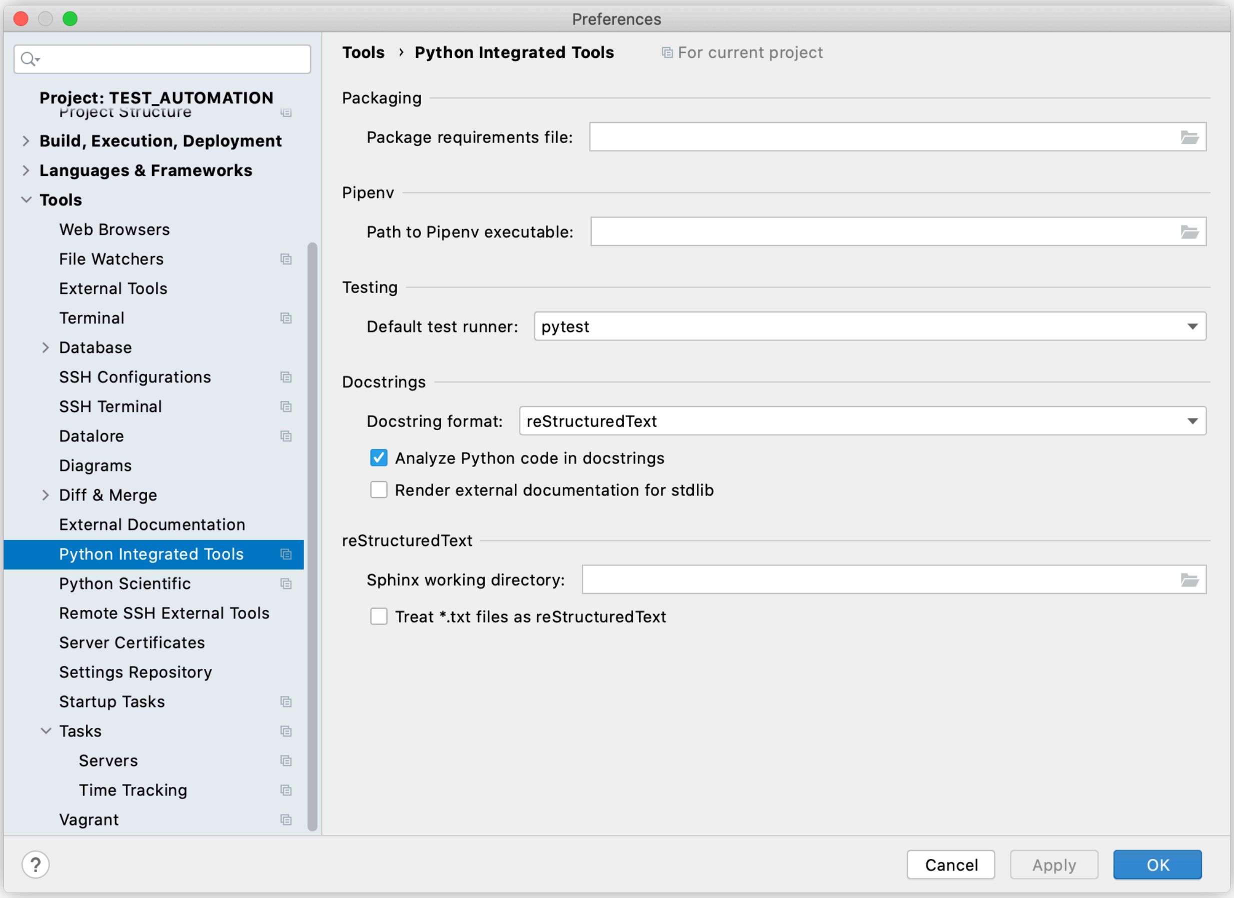
3 测试用例函数为test_开头，后续为用例编号，设计用例时需指定好用例编号，不能含有中文以及特殊字符

4 测试函数可以直接调用fixture作为实参传入，目前本工程中已定义的有database, ,driver,HTTPRequest等fixture,以上fixture均 定义在/TestCase对应文件夹的conftest中

5 fixture是pytest中的很大一个优势，可以使用它完成许多工作，fixture都需要定义在conftest.py文件中，工程目录下的conftest 为全局共享fixture，而测试目录下各个文件夹中conftest.py可自行创建并编写fixture，仅对该文件夹下的测试函数有效

6 每个测试用例需要使用@pytest.mark.TestCase("[用例等级]用例名称")装饰器,方便后续开发批量测试入口

7 本地调试/开发推荐使用Jetbrain 的 Pycharm,选择Tools 下 Python Integrated Tools 将 Default test runner修改为pytest


Preferences

Tools › Python Integrated Tools   For current project

Packaging

Package requirements file:

Pipenv

Path to Pipenv executable:

Testing

Default test runner: pytest

Docstrings

Docstring format: reStructuredText

[✓] Analyze Python code in docstrings

[ ] Render external documentation for stdlib

reStructuredText

Sphinx working directory:

[ ] Treat *.txt files as reStructuredText

要求

开发时使用Python 版本为: MACOS_x64_3.8.6, 使用VirtualENV虚拟环境运行，首次同步工程需要安装requirements.txt(cmd中pip install -r requirements.txt)

分辨率: 暂无，如果测试UI需要 1920*1080 以上

模块简介:

Config 模块:

主要为工程配置相关模块，存放工程所有配置信息

Browser.py

UI相关测试用例使用，会使用G中的browser以及对应平台，版本检测G变量下的driver_path是否有selenium驱动，如果没有，会根据G变量中指定的driver_url或者IE_driver_url 请求查询，beautifulsoup对html解析，如果没有找到驱动，会抛出异常

globalVars.py

工程核心配置文件，主体为globalVars类，借用了Flask框架中的G变量，做到工程配置类均从全局变量中获取

核心API请求模块以及Selenium自动化等模块均从该模块的G变量取值

配置详解

```
import os
import sys
from selenium.webdriver.common.by import By


class GlobalVars(object):
    OPERATION_WORKER = "PYTHON AUTOMATION TEST"
    TASK_NAME = "DAILY" # TASK_NAME为Local不会上传测试结果至数据库，
["DAILY","LOCAL","Task_name"]
    now_case_img_url = None
    now_case_startTime = None
    # API 服务器IP
    Server_IP = ""
    Server_Port = 80
    UploadFileAPI = "/FileInfoApi/uploadFileByOtherSystem"
    # API 测试验证ticket，当前api无验证，采用ticket，如后续api增加验证，则填入
Server_Checking_Username与password
    Server_Checking_ticket = {"zjugis.api.ticket": (None, "wwkj&key&zdww1402")}
    Server_Checking_Username = ""
    Server_Checking_password = ""

    # 工程路径相关
    root = os.path.dirname(__file__)
    suite_dir = os.path.join(root, "TestCase")
    skip_suite = None
    project_root = os.path.dirname(root)
    report_path = os.path.join(project_root, 'report')
    log_path = os.path.join(project_root, 'log')
    # 数据库配置，与服务器保持一致，默认根据Server_IP连接，使用ORM映射获得数据
    data_base_config = {
        'Z_AUTO_DEPLOY':
            {'ip': 'ip', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_USER_ORG_RIGHT':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_BUSSINESS_COMMOM':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_FILE_MANAGEMENT':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_MIDDLEWARE_MQ':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_SPRING_DEMO':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': 'develop'},
        'Z_WORKFLOW':
            {'ip': '1', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''},
        'Z_WEB_CONTAINER':
            {'ip': '', 'ListenerPort': 1521, 'password': '',
             'InstanceName': ''}}

    """




    UI相关
    请勿修改




    """
    SYSUsername = ""
    SYSPassword = ""
    # selenium UI测试重试次数
    RETRY = 3
    # 等待时间，如果超过该时间浏览器未返回数据，自动停止测试
    TIME_OUT = 1
    # selenium驱动存放地址
    resource_path = os.path.join(project_root, "resource")
    web_driver_path = os.path.join(resource_path, "webdriver")
    ELEMENT_PATH = os.path.join(os.path.dirname(web_driver_path),
"PageElement")
    # 默认为CHROME， IE后续可能会适配，但是IE适配难度较高，暂时不考虑
    browser = "CHROME"
    # CHROME版本，自动采集
    browser_version = None
    # selenium 驱动包TaoBao 镜像站
    ie_driver_url = "https://npm.taobao.org/mirrors/selenium/"
    driver_url = "https://npm.taobao.org/mirrors/chromedriver/"
    # 当前电脑平台，默认为mac，非mac设备运行时会自动改为windows或者linux(linux不适配，需
考虑排除无界面设备)
    platform = "mac"  # 默认为mac
    kernel = sys.platform
    if "darwin" in kernel:
        # MAC os
        chrome_app = r"/Applications/Google\ Chrome.app/Contents/MacOS/"  # mac
os chrome安装地址
```

```
    elif "win" in kernel:
        platform = "windows"
        # Win
        chrome_reg = r"SOFTWARE\Google\Chrome\BLBeacon"  # win chrome注册表地址
        instant_client = os.path.join(resource_path, "instant_client")
    else:
        platform = "linux"
        browser = "firefox"
    # 根据后续自动安装驱动返回
    DRIVER_PATH = None
    # 定位元素语法,selenium提供By
    LOCATE_MODE = {
        'css': By.CSS_SELECTOR,
        'xpath': By.XPATH,
        'name': By.NAME,
        'id': By.ID,
        'class': By.CLASS_NAME,
        "fulltext": By.LINK_TEXT,
        "parttext": By.PARTIAL_LINK_TEXT
    }


G = GlobalVars()
```

log文件夹

.gitnore忽略文件夹，需要自己在根目录创建，存放测试日志，日志文件格式为 测试函数_年月日-时分秒.log ,日志文件中日志打印格式为

年-月-日 时-分-秒，毫秒 - [日志来源文件:当前文件行号]-日志等级- 日志消息

重写pytest_make_report后的日志样本

```
当前节点: TestCase/UI/test_baidu.py::test_loginPlatform
TestCaseName: test_loginPlatform   Result: failed   Duration: 21.019038025S
--------------------Captured stdout setup--------------------
2020-11-11 16:39:44,951 - [browser.py-_jb_pytest_runner:27]- WARNING - 您的电脑为
mac 平台, 浏览器为 CHROME 版本号 86.0.4240.193
2020-11-11 16:39:45,674 - [browser.py-_jb_pytest_runner:46]- WARNING - 未查询到本
地驱动
开始下载!
Archive:  chromedriver_mac64.zip
  inflating: chromedriver
2020-11-11 16:39:48,641004 -
[/Users/wangbaofeng/PycharmProjects/TEST_AUTOMATION/logFile/logger.py-
cmdOUT:21]- INFO - TestCase: TestCase/UI/test_baidu.py::test_loginPlatform
Start, Using Fixtures: ['drivers', 'log', 'Init', 'request']
```

```
2020-11-11 16:39:48,641141 -
[/Users/wangbaofeng/PycharmProjects/TEST_AUTOMATION/logFile/logger.py-
cmdOUT:22]- INFO - UsingMarker: [Mark(name='z_user_org_right', args=(), kwargs=
{})]

2020-11-11 16:39:48,641184 -
[/Users/wangbaofeng/PycharmProjects/TEST_AUTOMATION/logFile/logger.py-
cmdOUT:23]- INFO - Start Setting UP



drivers = <selenium.webdriver.chrome.webdriver.WebDriver
(session="bffea97ac429a01c30340b2bc4936c60")>
Init = <logFile.logger.Logger object at 0x10467d040>

    @pytest.mark.z_user_org_right
    def test_loginPlatform(drivers, Init):
        Init.info("开始登录平台")
        A = LoginPlatform(driver=drivers)
        A.login()
>       assert drivers.title == "HH"
E       AssertionError: assert '万维自然资源规划一体化平台' == 'HH'

test_baidu.py:12: AssertionError
--------------------Captured log setup--------------------
WARNING  _jb_pytest_runner:browser.py:27 您的电脑为 mac 平台, 浏览器为 CHROME 版本号
86.0.4240.193
WARNING  _jb_pytest_runner:browser.py:46 未查询到本地驱动
drivers = <selenium.webdriver.chrome.webdriver.WebDriver
(session="bffea97ac429a01c30340b2bc4936c60")>
Init = <logFile.logger.Logger object at 0x10467d040>

    @pytest.mark.z_user_org_right
    def test_loginPlatform(drivers, Init):
        Init.info("开始登录平台")
        A = LoginPlatform(driver=drivers)
        A.login()
>       assert drivers.title == "HH"
E       AssertionError: assert '万维自然资源规划一体化平台' == 'HH'

test_baidu.py:12: AssertionError
--------------------Captured stdout call--------------------
2020-11-11 16:39:48,642 - [test_baidu.py-_jb_pytest_runner:9]- INFO - 开始登录平
台
2020-11-11 16:39:48,642 - [LoginPlatform.py-_jb_pytest_runner:22]- INFO - 读取登
录页元素定位配置
2020-11-11 16:39:48,645 - [LoginPlatform.py-_jb_pytest_runner:24]- INFO - 开始打
开页面
2020-11-11 16:39:49,462 - [BasePage.py-_jb_pytest_runner:31]- INFO - 打开网页:
http://ip:port/z_user_org_right/Login/index
```

```
2020-11-11 16:39:49,462 - [LoginPlatform.py-_jb_pytest_runner:26]- INFO - 输入登录名
2020-11-11 16:39:50,102 - [BasePage.py-_jb_pytest_runner:69]- INFO - 输入文本:
2020-11-11 16:39:50,102 - [LoginPlatform.py-_jb_pytest_runner:28]- INFO - 输入密码
2020-11-11 16:39:50,678 - [BasePage.py-_jb_pytest_runner:69]- INFO - 输入文本:
2020-11-11 16:39:50,678 - [LoginPlatform.py-_jb_pytest_runner:30]- INFO - 点击登录跳转
2020-11-11 16:39:51,717 - [BasePage.py-_jb_pytest_runner:75]- INFO - 点击元素:
('name', 'btnLogin')

drivers = <selenium.webdriver.chrome.webdriver.WebDriver
(session="bffea97ac429a01c30340b2bc4936c60")>
Init = <logFile.logger.Logger object at 0x10467d040>

    @pytest.mark.z_user_org_right
    def test_loginPlatform(drivers, Init):
        Init.info("开始登录平台")
        A = LoginPlatform(driver=drivers)
        A.login()
>       assert drivers.title == "HH"
E       AssertionError: assert '万维自然资源规划一体化平台' == 'HH'

test_baidu.py:12: AssertionError
--------------------Captured log call--------------------
INFO     _jb_pytest_runner:test_baidu.py:9 开始登录平台
INFO     _jb_pytest_runner:LoginPlatform.py:22 读取登录页元素定位配置
INFO     _jb_pytest_runner:LoginPlatform.py:24 开始打开页面
INFO     _jb_pytest_runner:BasePage.py:31 打开网页:
http://114.215.200.79:82/z_user_org_right/Login/index
INFO     _jb_pytest_runner:LoginPlatform.py:26 输入登录名
INFO     _jb_pytest_runner:BasePage.py:69 输入文本: wangbf
INFO     _jb_pytest_runner:LoginPlatform.py:28 输入密码
INFO     _jb_pytest_runner:BasePage.py:69 输入文本: zjugis1402!
INFO     _jb_pytest_runner:LoginPlatform.py:30 点击登录跳转
INFO     _jb_pytest_runner:BasePage.py:75 点击元素: ('name', 'btnLogin')
drivers = <selenium.webdriver.chrome.webdriver.WebDriver
(session="bffea97ac429a01c30340b2bc4936c60")>
Init = <logFile.logger.Logger object at 0x10467d040>

    @pytest.mark.z_user_org_right
    def test_loginPlatform(drivers, Init):
        Init.info("开始登录平台")
        A = LoginPlatform(driver=drivers)
        A.login()
>       assert drivers.title == "HH"
E       AssertionError: assert '万维自然资源规划一体化平台' == 'HH'

test_baidu.py:12: AssertionError
```

logFile模块

日志配置核心模块，封装logger，便于全局使用，只需了解如何使用，无需了解实现 使用方式为

```
from logFile.logger import Logger

log = Logger("DEBUG")

#打印消息,等级由低到高，括号中间为打印消息
log.debug()
log.info()
log.warning()
log.error()
log.critical()
```

Models模块

数据库ORM文件存放地址,(根据使用数据库切换，当前使用ORACLE数据库演示)存放着平台当前所有表空间的数据库模型，使用SQLAlchemy完成。每个模块对应数据库模型文件名为模块名，例z_user_org_right对应z_user_org_right.py，需搭配后续utils/DBconnect/ORACLE.py使用,

```
sqlacodegen oracle+cx_oracle://username:password@IP:port/instancename --outfile
filename.py
```

report文件夹

存放生成的测试截图，仅仅对UI用例生效，失败时保存浏览器截图至该文件夹，格式为年月日 时分秒_测试函数.png

resource文件夹

包含三个文件夹,

instant_client :存放 instant_client的路径，如果是windows需要将其中对应版本包解压将dll文件全部放到/venv/lib/site-packeges下

Page_element: 元素定位yml文件,

TestCase模块

存放测试用例模块，其中分三个小模块，API,DataBase，UI，分别对应接口自动化，数据库，以及UI自动化，在对应文件夹中写对应的测试用例代码，每个文件夹中的conftest.py可以添加自定义fixture，仅对该文件夹生效，放置在工程总目录中conftest.py是全局fixture的所在地

测试用例样本(写的比较简单，可以根据业务逻辑更改为try,except,else,finally结构并自定义final fixture)

各个模块的fixture样本均在文件夹中conftest.py有定义

API用例

```
"""API用例用例代码样本
    preInit为每个测试用例必须使用fixture,preInit  fixture自带log, 可以使用该
preInit.info preInit.debug preInit.error等打印消息




    禁止使用print打印, print函数本工程已屏蔽, 打印消息无法显示




    其余fixture根据需求使用
"""

@pytest.mark.TestCase("[1]测试获取登录API验证是否正常")
def test_Login_Api_Get_Token_GET(preInit, example_USER_fixture):
    preInit.info("这是测试一个用例")
    preInit.info("测试USER fixture")
    sss = example_USER_fixture.Login_Api_Get_Token_GET()
    preInit.info("本次测试状态码为 %s  " % sss[0])
    preInit.info("本次测试返回值为 %s  " % sss[1])
    preInit.info("本次测试响应头为 %s  " % sss[2])
```

DB用例

```
@pytest.mark.TestCase("[1]测试数据库")
def test_DataBase_TACTIVITYTEMPLATE(preInit, DataBaseSession):
    data = {
        "function": sys._getframe().f_code.co_name,
        "filename": os.path.dirname(__file__)
    }
    dbsession = DataBaseSession
    try:


        queryset =
dbsession.query(z_workflow.TActivityTemplate).filter(or_(z_workflow.TActivityTe
mplate.bz1 != None , z_workflow.TActivityTemplate.bz1 != None,

z_workflow.TActivityTemplate.bz3 != None , z_workflow.TActivityTemplate.bz4 !=
None ,

z_workflow.TActivityTemplate.isvalid != 1)
                                                                        )
        if queryset:
            with open(os.path.join(os.path.dirname(__file__),
                                   "DataBaseError_%s_%s.txt" % (
                                           data["filename"].split("/")[-1],
data["function"].split("_")[-1])),
                                  'a+') as f:
                    for i in queryset:
                        if i.bz1 != None or i.bz2 !=None or i.bz3 != None or i.bz4
!= None:
                            error_data = "ERROR:     id 为%s 的数据 bz1-bz4 分别为
bz1:%s  bz:%s  bz3:%s  bz4:%s\r" % (
                                i.id, i.bz1, i.bz2, i.bz3, i.bz4)
                            f.write(error_data)
                            preInit.error(error_data)
                        if i.isvalid != 1 :
                            error_data = "ERROR:    id 为%s 的数据 isvalid 为%s\r" %
(i.id, i.isvalid)
                            preInit.error(error_data)
                            f.write(error_data)
                        if i.create_time == None or i.create_worker == None:
                            error_data = "ERROR:     id 为%s 的数据 创建时间为%s /创建
人为 %s \r" % (i.id, i.create_time, i.create_worker)
                            preInit.error(error_data)
                            f.write(error_data)
        else:
            preInit.info("未查询到本数据库有违规数据")
    except Exception as e :
        dbsession.rollback()
```

UI用例

```
@pytest.mark.TestCase("[1]测试登录平台")
def test_loginPlatform(preInit, drivers):
    preInit.info("开始登录平台")
    A = LoginPlatform(driver=drivers)
    A.login()
    assert drivers.title == "HH"

 """其中LoginPlatform为封装完成的登录对象, 具体实现看如下"""
```

loginPlatform代码

```
from config.globalVars import G
from utils.UI.read_element import Element
from .BasePage import WebPage
from logFile.logger import Logger


"""封装登录平台基类，后续可继承自该类再进行后续操作"""
log = Logger(set_level="DEBUG")

def read_config(configname):
    return Element(configname)


class LoginPlatform(object):

    def __init__(self, driver):
        self.driver = driver
        self.ip = G.Server_IP # 根据G取ip，端口
        self.port = G.Server_Port
        self.basePage = WebPage(driver=self.driver) # 初始化WebPage类，
        self.LoginURL = "http://" + self.ip + ":" + str(self.port) +
"/z_user_org_right/Login/index"

    def login(self):
        log.info("读取登录页元素定位配置")
        #####读取定位元素，文件位于/resource/pageelement/下的yml文件，
        #####定位方法查看下方WebPage类
        self.LoginConfig = read_config('z_user_org_rightLoginindex')
        log.info("开始打开页面")
        self.basePage.get_url(self.LoginURL)
        log.info("输入登录名")
        self.basePage.input_text(self.LoginConfig["userName"], G.SYSUsername)
        log.info("输入密码")
        self.basePage.input_text(self.LoginConfig["userPwd"], G.SYSPassword)
        log.info("点击登录跳转")
        self.basePage.is_click(self.LoginConfig["btnLogin"])


class CreateWorkFlow(LoginPlatform):

    def clickOpen(self):
        log.info("读取首页菜单配置")
        menu_config = read_config("z_web_containerHomeblueIndex")
        log.info("点击新建流程")
        self.basePage.is_click(menu_config["新建流程"])
        log.info("切换iframe至当前新建流程")
        self.basePage.switch(menu_config["切换iframe"])
        self.basePage.is_click(menu_config["系统流程"])
        self.basePage.is_click(menu_config["新建新闻审核"])
```

WebPage实现

```
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from selenium.common.exceptions import TimeoutException
from config.globalVars import G
from utils.Others.TimeOperation import sleep
from logFile.logger import Logger
log = Logger("DEBUG")

"""
selenium基类
本文件存放了selenium基类的封装方法,
locator为(定位方式, 该方式对应的标志)

"""



class WebPage(object):
    """selenium基类"""

    def __init__(self, driver):
        self.driver = driver
        self.timeout = 20
        self.wait = WebDriverWait(self.driver, self.timeout)

    def get_url(self, url):
        """打开网址并验证"""
        self.driver.maximize_window()
        self.driver.set_page_load_timeout(60)
        try:
            self.driver.get(url)
            self.driver.implicitly_wait(10)
            log.info("打开网页: %s" % url)
        except TimeoutException:
            raise TimeoutException("打开%s超时请检查网络或网址服务器" % url)

    @staticmethod
    def element_locator(func, locator):
        """元素定位器"""
        name, value = locator
        return func(G.LOCATE_MODE[name], value)

    def find_element(self, locator):
        """寻找单个元素"""
```

```python
        return WebPage.element_locator(lambda *args: self.wait.until(
            EC.presence_of_element_located(args)), locator)

    def find_elements(self, locator):
        """查找多个相同的元素"""
        return WebPage.element_locator(lambda *args: self.wait.until(
            EC.presence_of_all_elements_located(args)), locator)

    def elements_num(self, locator):
        """获取相同元素的个数"""
        number = len(self.find_elements(locator))
        log.info("相同元素：{}".format((locator, number)))
        return number

    def switch(self,locator):
        sleep(0.5)
        ele = self.find_element(locator)
        log.info("切换至定位元素为%s%s的ifraeme" % locator)
        self.driver.switch_to_frame(ele)

    def input_text(self, locator, txt):
        """输入(输入前先清空)"""
        sleep(0.5)
        ele = self.find_element(locator)
        ele.clear()
        ele.send_keys(txt)
        log.info("输入文本：{}".format(txt))

    def is_click(self, locator):
        """点击"""
        self.find_element(locator).click()
        sleep()
        log.info("点击元素：{}".format(locator))

    def element_text(self, locator):
        """获取当前的text"""
        _text = self.find_element(locator).text
        log.info("获取文本：{}".format(_text))
        return _text

    def get_source(self):
        """获取页面源代码"""
        return self.driver.page_source

    def refresh(self):
        """刷新页面F5"""
        self.driver.refresh()
        self.driver.implicitly_wait(30)
```

```
if __name__ == "__main__":
    pass
```

读取yml作为locator

```
搜索框: "id==kw"
候选: "css==.bdsug-overflow"
搜索候选: "css==#form div li"
搜索按钮: "id==su"
yml内容如上，通过读取工具之后，返回值为{"搜索候选":("css",#form div li)}
再经过WebPage中的findelement方法进行定位，讲解稍微比较复杂，不做赘述
```

utils模块

HTTPRequest模块

存放API请求相关的工具代码，RequestBase存放着各自模块的一个类，对于API的请求均定义在各自模块的类中，

RequestBase.py

```
"""RequestBase: 所有模块请求的父类，后续定义其余模块均需要继承自该类，该类定义了请求方法以及
url处理等方法"""
from config.globalVars import G
import requests
from .RequestDataSource import RequestDataSource
from logFile.logger import Logger

RequestDataSource = RequestDataSource()


class RequestBase(object):
    def __init__(self, *args, **kwargs):
        self.ip = kwargs.get("ip") if kwargs.get("ip") else G.Server_IP
        self.port = kwargs.get("port") if kwargs.get("port") else G.Server_Port
        self.body_type = kwargs.get("body_type") if kwargs.get("body_type")
else "form"
        self.pattern = kwargs.get("pattern")
        self.auth = None
        self.logger = Logger()

    @staticmethod
    def remake_form(data):
        """
        :param data: 请求数据 字典形式
        :return: 重组为form表单的请求body
        """
        assert type(data) is dict
        remake_data = dict()
        for i in data:
            remake_data[i] = (None, data[i])
        return remake_data

    def auth_check(self, data):
        if self.auth:
            """
            目前平台API均未加密，暂时不定义
            """
            pass
        else:
            ticket = G.Server_Checking_ticket
            for i in ticket:
                data[i] = ticket[i]
        return data

    def remake_url(self, api_url):
        return "http://" + self.ip + ":"+ str(self.port) + "/" + self.pattern +
api_url

    def begin_request(self, *args, **kwargs):
        request_data = kwargs.get("request_data")
        request_method = kwargs.get("request_method")
        request_header = kwargs.get("request_header")
        api_url = kwargs.get("api_url")
        request_url = self.remake_url(api_url)
        cookie = kwargs.get("cookie") if kwargs.get("cookie") else None
        self.auth = kwargs.get("auth") if kwargs.get("auth") else None
        if request_data:
            request_data = self.remake_form(data=request_data)
            request_data = self.auth_check(request_data)
        self.logger.info("本地请求HEADER为 %s " % request_header )
        self.logger.info("本次请求URL为%s " % request_url)
        self.logger.info("本次请求方式为%s " % request_method)
        req = requests.request(method=request_method.upper(), url=request_url,
files=request_data, headers= request_header)
        if req:
            return req.status_code, req.text, req.headers
            """返回值为tuple类型，第一个元素为状态码，第二个为ResponseBody，第三个为
ResponseHeader"""
```

```
"""Z_USER_ORG_RIGHT模块的实例为ZUserOrgRight，后续规定，定义API请求类类名为去除模块中的
_,并改为驼峰氏命名




"""

class ZUserOrgRight(RequestBase):
    """
    各模块区分，继承至RequestBase
    一个接口一个方法，同一个接口不同的方式也分开定义
    函数定义方式为url中的/替换为_,最后_加上请求方式
    """

    def Login_Api_Get_Token_GET(self, *args, **kwargs):
        request_header = kwargs.get("request_header") if kwargs.get(
            "request_header") else RequestDataSource.RequestHeader()

        request_method = "GET"

        request_data = kwargs.get("request_data") if kwargs.get("request_data")
else RequestDataSource.DataSource_Login_Api_Get_Token_GET()

        request_api = "/LoginApi/getToken"

        auth = kwargs.get("auth") if kwargs.get("auth") else None

        cookie = kwargs.get("cookie") if kwargs.get("cookie") else None

        return
self.begin_request(request_method=request_method,request_data=request_data,
request_header=request_header, api_url= request_api,
                            auth=auth)

    """后续定义方法参照以上Login_Api_Get_Token_GET方法"""
```

与UI相关的broweser类一样，HTTPrequest也已经封装了一个实例fixture,位置是/TestCase/Api/conftest.py

代码如下

```
"""ZUserOrgRight继承至RequestBase，初始化需要一个pattern，
pattern等于 /模块名，因目前API文档中仅仅只写后续API，模块名未加入无法强求成功
```

```
本初始化一个ZUserOrgRight实例对象，并返回迭代器
"""

@pytest.fixture(scope="function")
def example_USER_fixture():
    USER_FIXTURE = ZUserOrgRight(pattern="/z_user_org_right")
    yield USER_FIXTURE

"""调用该fixture并请求"""

@pytest.mark.TestCase("[1]测试API")
def test_Login_Api_Get_Token_GET(example_USER_fixture,preInit):
    preInit.info("这是测试一个用例")
    preInit.info("测试USER fixture")
    sss = example_USER_fixture.Login_Api_Get_Token_GET()
    preInit.info("本次测试状态码为 %s  " % sss[0])
    preInit.info("本次测试返回值为 %s  " % sss[1])
    preInit.info("本次测试响应头为 %s  " % sss[2])
```

RequestDataSource.py

请求中的一些数据源: 定义一些借口的默认值，后续请求时如果使用默认值则可以直接调用，比较简单，不过分赘述，仅放示例代码

```

class RequestDataSource(object):

    def RequestHeader(self):
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36 QIHU
360SE'
        }
        return headers

    def DataSource_Login_Api_Get_Token_GET(self):
        source =  {
            "name": "Admin",
             "pwd": "zjugis1402"
                }

        return source
```

DBConnect模块

Oracle.py

oracle数据连接工具，使用SQLAlchemy封装为ORM类，初始化实例需要python dict类型，调用方式

```
configs = {
"account":"账号"
"ip": "ip地址",
"ListenerPort":"数据库监听端口",
"InstanceName":"数据库实例名",
"password":"密码"
}
```

```
import sqlalchemy
import os
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.testing import entities

Base = declarative_base()


class DataBaseOperation(object):

    def __init__(self, configs):
        os.environ["NLS_LANG"] = "GERMAN_GERMANY.UTF8"  # 解决中文乱码
        config = None
        data_base_account = None
        for k, v in configs.items():
            config = v
            data_base_account = k
        data_base_ip = config.get("ip")
        data_base_listener_port = config.get("ListenerPort")
        data_base_instance_name = config.get('InstanceName')
        data_base_password = config.get("password")
        self.db_engine = None
        self.meta = None

        if data_base_ip and data_base_password and data_base_listener_port and
data_base_instance_name and data_base_account and data_base_password:
            self.db_engine =
sqlalchemy.create_engine('oracle+cx_oracle://%s:%s@%s:%s/%s' % (
                data_base_account, data_base_password, data_base_ip,
data_base_listener_port, data_base_instance_name
            ), echo=True)
            session_maker = sessionmaker(bind=self.db_engine)
            self.session = session_maker()

    """
    原先使用的方法全部删除，改为使用session自带
    """
```

与之前一样，定义了一个实例fixture，位置/TestCase/DataBase/z_workflow/conftest.py

fixture代码如下

```
"""从G变量中获取当前模块的数据库账号密码，当前为z_workflow，因此取工作流账号密码，数据库连接
成功后，返回当前实例"""

@pytest.fixture()
def DataBaseSession():
    connection_data = G.data_base_config
    connect_data = {"Z_WORKFLOW":connection_data['Z_WORKFLOW']}
    DBsession = DataBaseOperation(connect_data)
    dbsession = DBsession.session
    yield dbsession
    dbsession.close_all()
```

UI模块

UI模块文件在TestCase模块中已经讲解过

others模块

存放处了UI，HttpRequest以及DBConnect模块以外的其他工具

OSoperiation.py

与操作系统相关的操作均通过调用该文件中的方法，不能在用例中操作os

```
def mk_dir(path):
    # 去除首位空格
    path = path.strip()
    path = path.rstrip("\\")
    path = path.rstrip("/")
    # 判断路径是否存在
    is_exists = os.path.exists(path)

    if not is_exists:
        try:
            os.makedirs(path)
        except Exception as e:
            log.error("目录创建失败: %s" % e)
    else:
        # 如果目录存在则不创建，并提示目录已存在
        log.debug("目录已存在: %s" % str(path))
        pass
```

SSHOperation.py

与SSH操作相关的工具均放置于此处,使用paramiko库完成，定义了SSH和SFTP两个类，SFTP继承自SSH，SSH类实现SSH连接，发送命令，读取终端输出功能，SFTP额外增加SFTP连接，可以上传文件以及下载文件

调用方式

```
import SSHBase
hostdict = {
"ip":"ip",
"port":int(port),
"username":用户名,
"password":密码
}
ssh = SSHBase(hostdict)
ssh.connect()  #连接
ssh.send_cmd("ls") #发送ls命令
ssh.receive_message_from_terminal() #获取目标终端信息,
ssh.root() # 当前连接Socket root
SFTP同理
```

```
import time
import paramiko


class SSHBase(object):
  def __init__(self, host_dict):
      self.host = host_dict['host']
      self.port = host_dict['port']
      self.username = host_dict['username']
```

```
        self.pwd = host_dict['password']
        self.transport = None
        self.channel = None

    def connect(self):
        self.transport = paramiko.Transport((self.host, self.port))
        self.transport.start_client()
        self.transport.auth_password(self.username, self.pwd)
        self.channel = self.transport.open_session()
        self.channel.get_pty()
        self.channel.invoke_shell()

    def send_cmd(self, string):
        send_string = '%s\r' % string
        self.channel.send(send_string)

    def receive_message_from_terminal(self, size=1024):
        rst = self.channel.recv(size)
        rst = rst.decode('utf-8')
        print(rst)

    def root(self):
        self.channel.send(r'su - root')
        time.sleep(0.2)
        rst = self.channel.recv(1024)
        rst = rst.decode('utf-8')
        if 'Password' in rst:
            self.channel.send('%s\r' % self.pwd)
            time.sleep(0.5)
            ret = self.channel.recv(1024)
            ret = ret.decode('utf-8')
          print(ret)

    def __del__(self):
        self.channel.close()
        self.transport.close()
```

```
class SFTPOperation(SSHBase):

  def upload(self, local_path, target_path):
      sftp = paramiko.SFTPClient.from_transport(self.transport)
      sftp.put(local_path, target_path, confirm=True)
      sftp.chmod(target_path, 0o755)  # 注意这里的权限是八进制的，八进制需要使用0o作为
前缀

  def download(self, target_path, local_path):
      sftp = paramiko.SFTPClient.from_transport(self.transport)
      sftp.get(target_path, local_path)

  def __del__(self):
      self.transport.close()
```

TestReportOnline模块

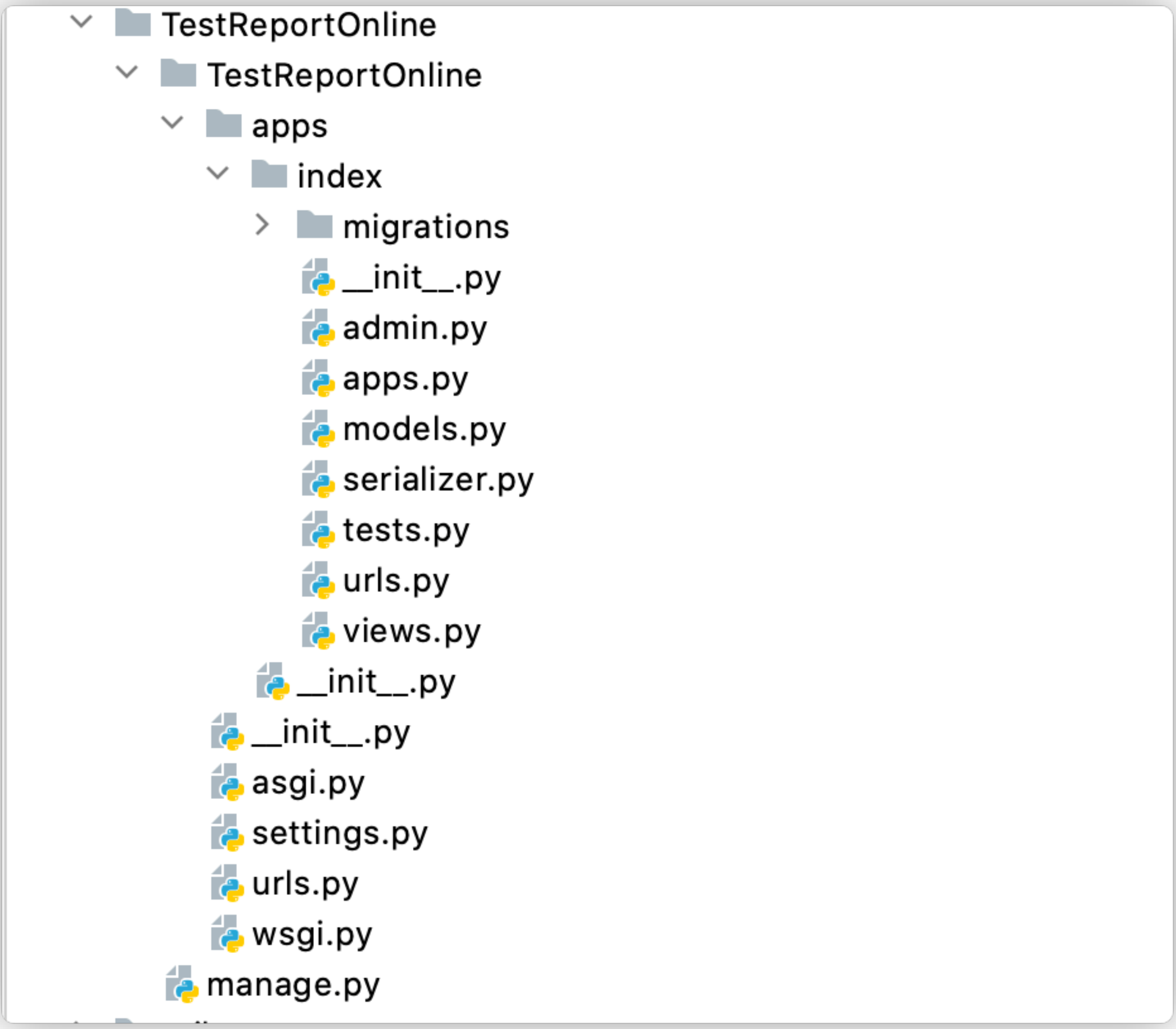
该模块为web项目，负责测试结果查询API，后续会部署至服务器，一般人员无需关心其构建，采用Django+Django RestFrameWork编写，后续会提供以下API，遵从Restful接口规范

API概览:

```
urlpatterns = [
    url(r'^$', TestCaseAPI.as_view({"get": "list", "post": "create"})),
    url(r'^(?P<pk>\d+)/$', TestCaseAPI.as_view({"get": "retrieve", "delete":
"destroy", "put": "update"})),
    url(r'^getresultByresult=(?P<result>(passed|failed))/$',
TestSystemMultipleAPI2.as_view({"get": 'retrieve'})),
    url(r'^getresultBytaskname=(?P<taskname>\w+)/$',
TestSystemMultipleAPI2.as_view({"get": 'retrieve'})),
    url(r'^getresultBycreate_worker=(?P<create_worker>.*)/$',
TestSystemMultipleAPI2.as_view({"get": 'retrieve'})),
    url(r'^getresultBycase_number=(?P<case_number>.*)/$',
TestSystemMultipleAPI2.as_view({"get": 'retrieve'})),
    url(r'getresultByVague', TestSystemVagueQuery.as_view())
]



1 /TestReport/ 返回所有用例测试结构，GET方式，ResponseBOdy为JSON格式，数据量较大，post
为新建一个测试结果，需要验证
2 /TestReport/id  id为测试结果数据库id主键，GET方式，ResponseBOdy为JSON格式，返回当前
ID的测试用例详情
3 /TestReport/getresultByresult=  填passed或者failed，GET方式，ResponseBOdy为JSON
格式，返回测试结果为通过/失败的所有用例
4 /TestReport/getresultBytaskname= 填任务名，GET方式，ResponseBOdy为JSON格式，返回测
试人物名为所查询的人物名的所有用例
```

```
5 /TestReport/getresultBycreate_worker= 填创建人，GET方式，ResponseBOdy为JSON格式，返回测试创建为所查询的创建者的所有用例
6 /TestReport/getresultBycase_number= 填用例编号，GET方式，ResponseBOdy为JSON格式，返回用例编号为所查询编号的所有记录
以上均为  精确查询

模糊查询
7 /TestReport/getresultByVague GET方式，查询参数为create_time(创建时间)，marker(标记)，ending_time(结束时间)，nodeid(测试节点) 根据查询参数模糊匹配出的结果，ResponseBody为Json格式
```

实现讲解



```
TestReportOnline
    TestReportOnline
        apps
            index
                migrations
                __init__.py
                admin.py
                apps.py
                models.py
                serializer.py
                tests.py
                urls.py
                views.py
            __init__.py
        __init__.py
        asgi.py
        settings.py
        urls.py
        wsgi.py
    manage.py
```

关于Django和DRF此处不做解释，只讲解views.py,Model与Sqlalchemy生成的有一些区别，Django也支持根据已生成库生成ORM文件，具体百度一下即可。

```
class TestCaseAPI(ModelViewSet):

  """
```

```
        序列化器以及Model很简单，不解释
        继承至DRF的ModelViewSet
        GET方式请求 返回所有测试用例执行详情的JSON
        GET方式请求 指定了ID ， 返回该ID的JSon
        POST 新增
        DELETE 删除
        PATCH 更新

    """
    serializer_class = TestCaseSerializer
    queryset = Testcaseresult.objects.all()


class TestSystemMultipleAPI2(TestCaseAPI):

    """继承至以上，因为感觉只能一次返回比较麻烦，因此重写了ModelViewSet
    中的retreive方法以及get_object()方法，这个需要具体看源码才能知道为什么需要重写这里，解释
起来很麻烦
    DRF中 ModelViewSet继承了很多Mixin，感兴趣点进去看一下就大概了解
    """
    # 局部配置过滤器类
    filter_backends = [DjangoFilterBackend]
    # 参与分类筛选的字段：
    filter_class = ResultFilter

    def get_object(self):

        queryset = self.filter_queryset(self.get_queryset())

        # Perform the lookup filtering.
        lookup_url_kwarg = self.lookup_url_kwarg or self.lookup_field

        assert lookup_url_kwarg in self.kwargs, (
                'Expected view %s to be called with a URL keyword argument '
                'named "%s". Fix your URL conf, or set the `.lookup_field` '
                'attribute on the view correctly.' %
                (self.__class__.__name__, lookup_url_kwarg)
        )

        filter_kwargs = {self.lookup_field: self.kwargs[lookup_url_kwarg]}
        obj = get_list_or_404(queryset, **filter_kwargs)

        return obj

    def retrieve(self, request, *args, **kwargs):
        path = request.get_full_path()
        from urllib import parse
        path = parse.unquote(path.split("/")[-2])
        # getresultByresult=passed
        req_pa = path.split("By")[-1]
        req_a = req_pa.split("=")[0]
        if req_a in ["result", "marker", "create_worker", "taskname",
'case_number']:
            self.lookup_url_kwarg = req_a
            self.lookup_field = req_a
        queryset = self.get_object()
        page = self.paginate_queryset(queryset)
        if page is not None:
            serializer = self.get_serializer(page, many=True)
            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)
        return Response(serializer.data)


class TestSystemVagueQuery(APIView):

    def get(self, request):

        vague_list = ["create_time", 'marker', "ending_time", 'nodeid']
        now_case_filter = None
        for i in vague_list:
            request_param = request.query_params.get(i)
            if request_param:
                now_case_filter = {"%s__icontains" % i: request_param}

        query_set = Testcaseresult.objects.filter(**now_case_filter)
        if query_set:
            seriliazer = TestCaseSerializer(query_set, many=True)
            return Response(seriliazer.data)
        else:
            raise Http404('MMP')
```

项目工程中的conftest.py

pytest框架允许多个conftest,运行时从工程最外层收集用例，并收取conftest,pytest.ini，因此一次启动会收集多个配置信息，fixture。这个正好可以满足全局配置放在最外层，局部的fixture配置等放置在里层。

最外层conftest,定义了许多fixture

```
def pytest_runtest_setup(item):
    for mark in item.iter_markers(name="TestCase"):
        print("TestCase args={} kwargs={}".format(mark.args, mark.kwargs))
        sys.stdout.flush()
"""使用pytest自定义marker"""
"""该marker 为TestCase 接收一个字符串参数，本项目规定为[用例等级]用例编号"""
```

```
@pytest.fixture(scope="function", autouse=True)
def preInit(request, log):
    """初始化fixture 返回log对象
    fixture可以调用其他fixture

    """
    log.info("Start Setting UP " + "\r")
    os.environ['NLS_LANG'] = 'SIMPLIFIED CHINESE_CHINA.UTF8'
    filepath = request.node.nodeid
    G.now_case_startTime = datetime_strftime()
    log.info("TestCase: %s Start, Using Fixtures: %s " % (filepath,
str(request.fixturenames)))
    G.now_case_level = request.node.own_markers[0].args[0].split("]")[0][1]
    G.now_case_name = request.node.own_markers[0].args[0].split("]")[1]
    G.now_case_number = request.node.name
    log.info("UsingMarker: %s CaseLevel: Level %s CaseName:%s" %
(request.node.own_markers[0].name, G.now_case_level, G.now_case_name ) + "\r")

    yield log

    """preInit fixture，所有用例必须使用的fixture，其作用为根据TestCase marker获取 用
例等级，名称，赋值给G变量，调整全局字体为UTF-8
    这里的作用是配合后续重写pytest_make_report,在生成报告时可以得到一个很详细的测试详情,
可以接着看下面
    """
```

```
@pytest.hookimpl(hookwrapper=True, tryfirst=True)
def pytest_runtest_makereport(item, call):
    """"
    pytest后置处理fixture，无需调用，用例结束自动调用

    不能修改 该fixture负责生成日志，上传记录等。



    """
    print('-------------------------------------')
    out = yield
    report = out.get_result()
    if report.when == 'call':
        logpath = os.path.join(G.log_path, "%s_%s.log" % (datetime_strftime(),
report.head_line))
        logs = ''
        with open(logpath, "a+") as  f:
            f.write("当前节点: %s " % report.nodeid + "\r")
            logs += "当前节点: %s " % report.nodeid + "\r"
            f.write("TestCaseName: %s   Result: %s   Duration: %sS " % (
                report.head_line, report.outcome, report.duration) + "\r")
            logs += "TestCaseName: %s   Result: %s   Duration: %sS " % (
                report.head_line, report.outcome, report.duration) + "\r"
            for i in report.sections:
                for j in i:
                    if "Captured" in j:
                        f.write("-" * 20 + j + "-" * 20 + "\r")
                        logs += "-" * 20 + j + "-" * 20 + "\r"
                    else:
                        f.write(j + "\r")
                        logs += j + "\r"
                if report.longreprtext:
                    f.write(report.longreprtext + "\r")
                    logs += report.longreprtext + "\r"
        if G.TASK_NAME != "Local":
            # TASK_NAME 为Local时不会上传测试记录，但会在本地留下日志信息
            now_case = Testcaseresult()
            now_case.case_number = G.now_case_number
            now_case.case_name = G.now_case_name
            now_case.Level = G.now_case_level
            now_case.result = str(report.outcome)
            now_case.create_time = G.now_case_startTime
            now_case.create_worker = G.OPERATION_WORKER
            now_case.taskname = G.TASK_NAME
            now_case.ending_time = datetime_strftime()
            now_case.ending_worker = G.OPERATION_WORKER
            now_case.marker = str(report.keywords)
            now_case.imgurl = None if not G.now_case_img_url else
G.now_case_img_url
            now_case.logs = logs
            saveCase(now_case)

 """Pytest在每次用例执行结束，无论失败均调用该fixture,重写它将后置处理调整为自己想要的，此
处我调整为生成日志，/log/时间_用例名称.log，并且如果在测试任务名不为Local时会保存本次测试数
据至数据库

 在正式自动化测试中，测试信息结果等均需保留，本地调试时无需上传，TestCasereult()是我自己定义
的Model，采用的是ORacle，也可以自定义使用Mysql 或者其他类型数据库，
 """
## /Models/TestCase.py
class Testcaseresult(Base):
    __tablename__ = 'testcaseresult'
    __table_args__ = {'comment': '测试数据管理'}

    id = Column(Integer, primary_key=True, comment='主键')
    create_worker = Column(VARCHAR(36), comment='创建人')
    create_time = Column(VARCHAR(36), comment='创建时间')
    ending_worker = Column(VARCHAR(36), comment='最后修改人')
    ending_time = Column(VARCHAR(36), comment='最后修改时间')
    logs = Column(CLOB(10000), comment='测试日志')
    result = Column(NCHAR(200), comment='测试结果')
    case_name = Column(NCHAR(200), comment='用例名')
    case_number = Column(NCHAR(200), comment='用例编号')
    marker = Column(NCHAR(200), comment='用例模块(Marker)')
    Level = Column(NCHAR(400), comment='用例等级')
    imgurl = Column(NCHAR(400), comment='截图')
    taskname = Column(NCHAR(400), comment='任务名')

    """这就是一个比较详细的测试结果保存，之前考虑过通过pytest-html自动生成结果，但是效果不太满意，allure比较麻烦，因此我个人认为保存至数据库，再通过开发web项目是一个比较好的选择，因为可自定义程度大，毕竟可以集成其他功能，大型公司均采用这个模式

    """

   def saveCase(instance):
    Session_class = sessionmaker(bind=engine)
    Session = Session_class()
    try:
        Session.add(instance)
    except Exception as e:
        Session.rollback()
    Session.commit()
    Session.close_all()

"""SaveCase是保存方法，初始化ORM对象后可以采用sqlalchemy 提供的add方法，如果保存失败，需要rollback(),Sqlalchemy是一个十分好用的三方库，有兴趣的同学可以学习Flask，里边会用到flask-sqlalchemy,相比Django自带的orm会比较容易理解，能更深的理解python中的ORM"""
```

Pytest.ini

在项目根目录下的conftest.py类似，我仅在里边增加了一个marker,因为如果仅仅在conftest.py中注册TestCase Marker,日志中会有Warning提示