整体介绍

项目基于Python开发，使用Pytest测试框架，因为unittest等其他框架三方插件比较少，开发复杂，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 pytest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, log等fixture，database为返回当前模块的数据库连接信息，log fixture会自动传入log handler，以及UI测试中日志失败插入截图等等

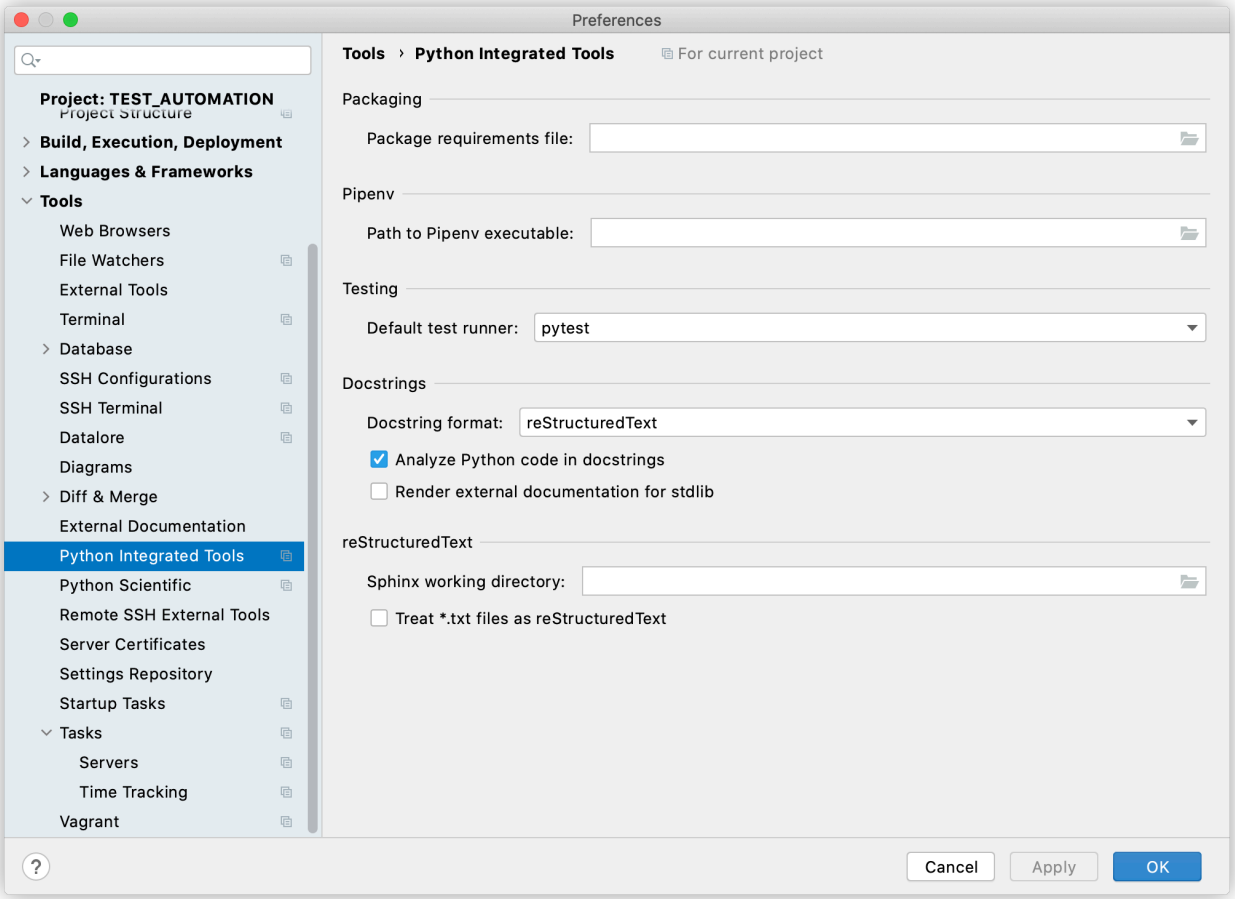
5 fixture是pytest中的很大一个优势，可以使用它完成许多工作，fixture都需要定义在conftest.py文件中，工程目录下的conftest为全局共享fixture，而测试目录下各个文件夹中conftest.py可自行创建并编写fixture，仅对该文件夹下的测试函数有效

6 每个测试用例需要加入marker装饰器，比如 这是用户模块的api测试，则需要在测试函数上添加@pytest.marker.z_user_org_right_api或者其他类型，方便区分的marker

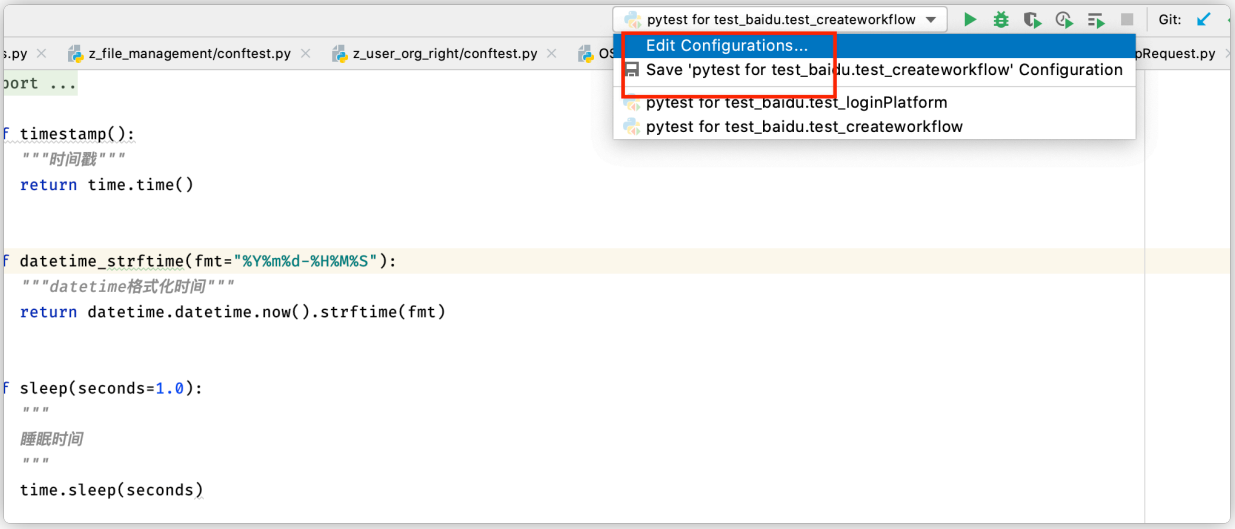
7 启动入口目前定义的不太完善，通过工程目录下的run.py启动，启用方式为cmd中 python run.py --marker=z_user_org_right 则会自动测试所有marker为z_user_org_right的用例

8 本地调试/开发推荐使用Jetbrain 的 Pycharm,需要在设置中操作以下步骤

8.1 选择Tools 下 Python Integrated Tools 将 Default test runner修改为pytest



8.2 修改启动模板



配置好后，开发测试时只需在需要运行测试函数上右键即可测试该函数

要求

Python 版本为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: 1920*1080

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

负责UI自动化的浏览器驱动检测，如果已存在驱动则返回驱动地址供Selenium使用，如不存在驱动则会根据指定源地址下载当前平台的Chrome版本适配的驱动

主体为Browser类，通过Browser实例的set_browser()方法完成驱动配置，会自动检测当前操作平台,位数(目前仅适配MAC以及Windows 的Chrome 以及 IE，firefox浏览器以及Linux不支持)

该方法已经在TestCase/UI/conftest.py中定义了fixture，编写测试用例时仅需要传入browser参数

browser实现代码

```
import os
import re
import subprocess
import zipfile

import requests
from bs4 import BeautifulSoup
from config.globalVars import G
from utils.Others.OSOperation import mk_dir
from logFile.logger import Logger
import selenium

log = Logger()


class Browser(object):

    @classmethod
    def set_browser(cls):
        # 检查Chrome版本号
        global version
        if "mac" in G.platform:
            # OS X
            result = subprocess.Popen([r'{}/Google\ Chrome --
version'.format(G.chrome_app)],
                                      stdout=subprocess.PIPE, shell=True)
            version = [x.decode("utf-8") for x in result.stdout]
[0].strip().split(" ")[-1]
```

```
            log.warning("您的电脑为 %s 平台, 浏览器为 %s 版本号 %s " % (G.platform,
G.browser, version))
        elif "win" in G.platform and G.browser == "CHROME":
            import winreg
            try:
                key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, G.chrome_reg)
                version = winreg.QueryValueEx(key, "version")[0]  # 查询注册表
chrome版本号
            except Exception:
                raise Exception("查询注册表chrome版本失败!")
            log.warning("您的电脑为 %s 平台, 浏览器为 %s 版本号 %s " % (G.platform,
G.browser, version))

        elif "win" in G.platform and G.browser == "IE":
            log.warning("您的电脑为 %s 平台, 浏览器为 %s  ,%s 是不被完整支持的浏览器 "
% (G.platform, G.browser, G.browser))
            version = selenium.__version__
        G.browser_ver = version
        file_vr = cls.search_ver(version)
        if file_vr is None:
            raise Exception("未获取到版本号! 请检查!")
        status, file = cls.check_driver(file_vr)
        if not status:
            log.warning("未查询到本地驱动")
            cls.gen_driver(file_vr)
        else:
            log.warning("系统已存在%sdriver, 无需下载!" % G.browser)
            G.DRIVER_PATH = os.path.join(G.web_driver_path, file)




    @classmethod
    def check_driver(cls, version):
        status, filename = False, None
        if os.path.exists(G.web_driver_path):
            pass
        else:
            mk_dir(G.web_driver_path)
        for root, dirs, files in os.walk(G.web_driver_path):
            for file in files:
                if version not in file:
                    try:
                        os.remove(os.path.join(root, file))
                    except Exception:
                        continue
                else:
```

```
                    status, filename = True, file

        return status, filename

    @classmethod
    def search_ver_v2(cls, version):
        ver = ".".join(version.split(".")[:2])
        r = requests.get(G.driver_url)
        bs = BeautifulSoup(r.text, features='html.parser')
        rt = [x for x in bs.select("pre a")]
        if not rt:
            raise Exception("可能淘宝镜像挂了，请重试")
        for x in rt:
            if x.text.startswith(ver):
                return x.text.rstrip("/")
        else:
            raise Exception("没有找到当前版本的合适驱动：{}".format(version))

    @classmethod
    def search_ver(cls, version):
        if version != "unknown":
            file_vr = None
            if G.browser == "CHROME":
                number = version.split(".")[0]

                url = G.driver_url + "LATEST_RELEASE"
                r = requests.get(url)
                bs = BeautifulSoup(r.text, 'html.parser')
                latest = bs.text.strip()
                record = "{}/{}/notes.txt".format(G.driver_url, latest)
                info = requests.get(record)
                text = info.text
                vr = re.findall(r"-+ChromeDriver\s+v(\d+\.+\d+)[\s|.|-|]+",
text)
                br = re.findall(r"Supports\s+Chrome\s+v(\d+-\d+)", text)
                if not br:
                    return cls.search_ver_v2(version)
                for v, b in zip(vr, br):
                    small, bigger = b.split("-")
                    if int(small) <= int(number) <= int(bigger):
                        # 找到版本号
                        log.info("找到浏览器对应驱动版本号：{}".format(v))
                        file_vr = v
                        break
            elif G.browser == "IE" and G.platform =="windows":
                global req_version
                if version.endswith('0'):
                    req_version = version[:-2]
```

```
                url = G.ie_driver_url + req_version + "/"
                r = requests.get(url)
                bs = BeautifulSoup(r.text, 'lxml')
                url_list = bs.find_all(['a'])
                import platform
                posix = platform.architecture()
                log.warning("您的设备为%s%s" % posix)
                vr = "Win32_%s" % version
                v_l = []
                for i in url_list:
                    v_l.append(i.attrs['href'])
                if vr in str(v_l):
                    log.info("找到浏览器对应驱动版本号：{}".format(file_vr))
                    file_vr = vr
            return file_vr


    @classmethod
    def gen_driver(cls, file_vr):
        if file_vr:
            driver =None
            file = None
            r = None
            if G.browser == "CHROME":
                if G.platform == "mac":
                    file = "chromedriver_mac64.zip".format(file_vr)
                    driver = "chromedriver"
                elif "win" in G.platform:
                    file = "chromedriver_win32.zip".format(file_vr)
                    driver = "chromedriver.exe"
                else:
                    file = "chromedriver_linux64.zip".format(file_vr)
                    driver = "chromedriver"
                r = requests.get("{}{}/{}".format(G.driver_url, file_vr, file))
            elif G.browser == "IE":
                file = "IEDriverServer_{}.zip".format(file_vr)
                driver = "IEdriverServer.exe"
                r = requests.get("{}
{}/IEDriverServer_{}.zip".format(G.ie_driver_url, req_version, file_vr))

            file_path = os.path.join(G.web_driver_path, file)
            print("开始下载!")
            with open(file_path, "wb") as f:
                f.write(r.content)
            cls.unzip_driver(file)
            cls.change_driver_name(file_vr, driver)

    @classmethod
    def unzip_driver(cls, filename):
```

```
        if G.platform == "mac":
            # 解压zip
            os.system('cd {};unzip {}'.format(G.web_driver_path, filename))
            os.path.join(G.web_driver_path, filename)
        elif "win" in G.platform:
            cls.unzip_win(os.path.join(G.web_driver_path, filename))
            os.remove(os.path.join(G.web_driver_path, filename))
        else:
            pass

    @classmethod
    def change_driver_name(cls, version, filename):
        if G.platform == "mac":
            new_file = "{}_{}".format(filename, version)
        elif G.platform == "windows":
            L = filename.split(".")
            new_file = "{}_{}.{}".format("".join(L[:-1]), version, L[-1])
        else:
            new_file = ""
        os.rename(os.path.join(G.web_driver_path, filename),
                  os.path.join(G.web_driver_path, new_file))
        G.DRIVER_PATH = os.path.join(G.web_driver_path, new_file)

    @classmethod
    def unzip_win(cls, filename):
        """unzip zip file"""
        with zipfile.ZipFile(filename) as f:
            for names in f.namelist():
                f.extract(names, G.web_driver_path)
```

driver fixture实现代码

fixture为Pytest一大特性，十分方便，推荐使用。

```
@pytest.fixture(scope='session', autouse=False)
def drivers(request):
    global driver
    if driver is None:
        Browser.set_browser()
        if G.browser == "CHROME":
            driver = webdriver.Chrome(executable_path=G.DRIVER_PATH)
        else:
            driver = webdriver.Ie(executable_path=G.DRIVER_PATH)
        driver.maximize_window()
    def fn():
        driver.quit()

    request.addfinalizer(fn)
```

```
    return driver
```

UI相关测试用例编写例子

在项目工程中/TestCase/UI/test_baidu.py中定义了两个简单的样本，后续可根据该例子来进行组合编写

fixture使用样例

```
@pytest.mark.z_user_org_right
def test_loginPlatform(drivers, Init):
    """ fixture可以直接当做参数传入测试代码，其中driver为上面driver fixture，传入之后driver fixture会先完成用例执行的前置操作，即设置驱动，设置好后，会将设置好的driver返回，在测试脚本中可以使用该driver完成后续操作，本例中的LoginPlatform为封装的Selenium类，后面再进行讲解
    """
    Init.info("开始登录平台")
    A = LoginPlatform(driver=drivers)
    A.login()
    assert drivers.title == "HH"
```

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
    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

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

```
2020-11-03 15:55:57,673 - [test_baidu.py-_jb_pytest_runner:20]- INFO - 开始登录平台
```

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，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

result文件夹

对测试结果进行整合以及Jinja2模板渲染，不详细讲述

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

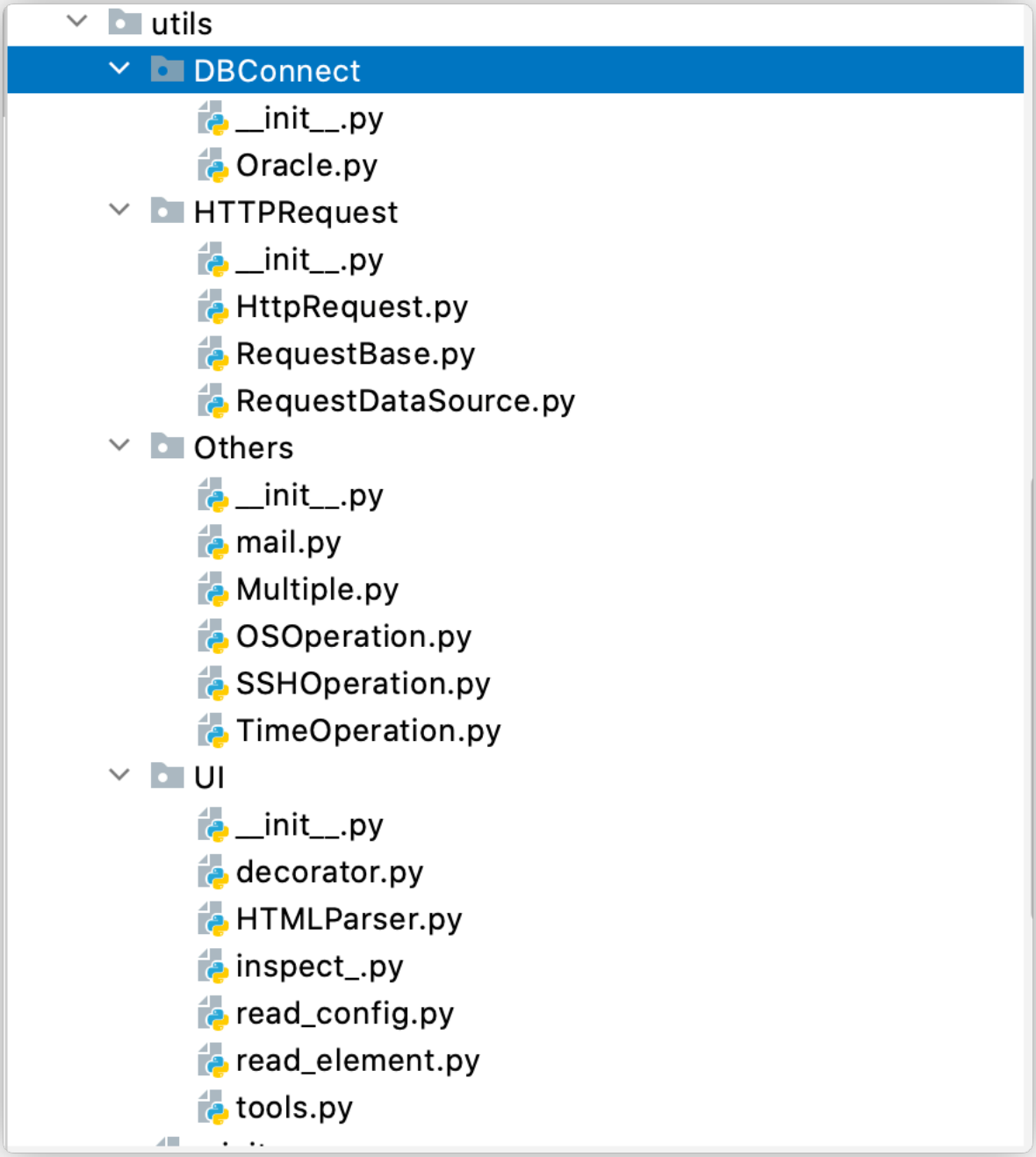
存放测试用例模块，其中分三个小模块，API,DataBase，UI，分别对应接口自动化，数据库，以及UI自动化，在对应文件夹中写对应的测试用例代码，每个文件夹中的conftest.py可以添加自定义fixture，仅对该文件夹生效，放置在工程总目录中conftest.py是全局fixture的所在地

tips

三个文件夹中如果需要自定义fixture，可以在文件夹中添加conftest.py文件，在该文件中定义fixture,该文件夹内用例可直接使用

utils模块

存放各种工具的目录，是本工程代码量最大的文件夹，结构如图。



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
ResponsHeader"""
```

```
"""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.z_user_org_right
def test_Login_Api_Get_Token_GET(example_USER_fixture,Init):
    Init.info("这是测试一个用例")
    Init.info("测试USER fixture")
    sss = example_USER_fixture.Login_Api_Get_Token_GET()
    Init.info("本次测试状态码为 %s  " % sss[0])
    Init.info("本次测试返回值为 %s  " % sss[1])
    Init.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"
}

####测试用例无需传入参数，在Database对应文件夹中定义fixture，自动返回连接，
database = DataBaseOperation(configs) # 实例初始化，  开始连接，
查询  query方法：
database.query(instance,filter,order) #查询 instance为必填参数，为需要查询的ORM对象
##例 ：查询登录表空间下某个表的所有数据，database.query(instance) return值为对象列表，
列表中的每个元素均为一条数据
filter为过滤器，非必填，例：database.query(instance,filter=instance.id<5) 表示查询ID
小于5的数据对象
order为排序，非必填，
database.query(instance,fileter=instance.id<5,order=instance.id) 表示查询ID小于5
的数据对象并根据id排序
增加 insert方法：
从Models中导入对应的表ORM，创建该实例，并赋值，例：
  a是当前登录账号下Models的一张表
  a.id = 5
  a.bz1 = 1
  a.bz2= 3
```

```
database.insert(a) #如果保存失败，session自动回滚
删除 delete方法：
  假设a是通过query查询出来的对象，
  database.delete(a) #操作失败自动回滚，暂时只支持该方法，后续会支持根据条件删
修改 update方法：
  修改后的实例a
  database.update(a) #操作失败自动回滚
```

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


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()

    def query(self, table_name, filer=None, order=None):
```

```
        """

        :param table_name: 表名
        :param filer: 过滤器
        :param order: 排序
        :return: 符合条件的数据实例
        """
        data = None
        if filer:
            if order:
                data =
self.session.query(table_name).filter_by(filer).order_by(order)
            else:
                data = self.session.query(table_name).filter_by(filer)
        else:
            if order:
                data = self.session.query(table_name).order_by(order)
            else:
                data = self.session.query(table_name)

        return data

    def insert(self, instance):
        """
        :param instance:ORM实例，通过query查询得到
        :return:
        """
        try:
            self.session.add(instance)
        except Exception as e :
            # 如果插入失败，则回退
            self.session.rollback()

    def delete(self,instance):
        """
        :param instance: 删除的ORM实例，通过query查询得到
        :return:
        """
        try:
            self.session.delete(instance)
        except Exception as e :
            # 删除实例失败，oracle回退
            self.session.rollback()

    def update(self, instance):
        """实例"""
        if hasattr(instance, 'id'):
            try:
```

```
                self.query(table_name=instance.__class__,
filer=instance.id).update(values=instance.__dict__)
            except Exception as e:
                self.session.rollback()

    def __del__(self):
        #自动commit
        self.session.commit()
        #实例销毁时关闭Oracle连接池
        self.session.close()
        self.db_engine.dispose()
```

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)
    yield DBsession
```

使用该fixture的示例代码

```
"""数据库用例的用例编号命名规则为: test_DataBase_表名
  使用该fixture后，数据库连接成功，可以直接使用它继续增删改查，参照上面ORacle.py的操作
"""


@pytest.mark.z_workflow
def test_DataBase_TACTIVITYTEMPLATECopy1(DataBaseSession):
    data = {
        "function": sys._getframe().f_code.co_name,
        "filename": os.path.dirname(__file__)
    }

    try:

        oracle_instance = DataBaseSession
        query_result = oracle_instance.query(z_workflow.TACTIVITYTEMPLATECopy1)
        if query_result:
```

```
            for i in query_result:
                logFile.debug("确认%s表id为%s的数据的isvalid字段为1" %
(i.__tablename__, i.id))
                try:
                    assert i.isvalid == 1
                except Exception as e:
                    error_data = "ERROR:    id 为%s 的数据 isvalid 为%s\r" %
(i.id, i.isvalid)
                    logFile.error(error_data)
                    with open(os.path.join(os.path.dirname(__file__),
                                           "DataBaseError_%s_%s.txt" % (
                                           data["filename"].split("/")[-1],
data["function"].split("_")[-1])),
                              'a+') as f:
                        f.write(error_data)
                logFile.debug("确认%s表id为%s的数据的bz1-bz4字段为空" %
(i.__tablename__, i.id))
                try:
                    assert i.bz1 is None and i.bz2 is None and i.bz3 is None
and i.bz4 is None
                except Exception as e:
                    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)
                    logFile.error(error_data)
                    with open(os.path.join(os.path.dirname(__file__),
                                           "DataBaseError_%s_%s.txt" % (
                                               data["filename"].split("/")
[-1], data["function"].split("_")[-1])),
                              'a+') as f:
                        f.write(error_data)
                logFile.debug("确认%s表id为%s的数据的创建时间创建人字段不为空" %
(i.__tablename__, i.id))
                try:
                    assert i.create_time is not None and i.create_worker is not
None
                except Exception as e:
                    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)
                    logFile.error(error_data)
                    with open(os.path.join(os.path.dirname(__file__),
                                           "DataBaseError_%s_%s.txt" % (
                                               data["filename"].split("/")
[-1], data["function"].split("_")[-1])),
                              'a+') as f:
                        f.write(error_data)
                logFile.debug("确认%s表id为%s的数据的最后修改时间最后修改人字段不为空"
% (i.__tablename__, i.id))
                try:
```

```
                    assert i.latest_modify_worker is not None and
i.latest_modify_time is not None
                except Exception as e:
                    error_data = "ERROR:     id 为%s 的数据最后修改时间为%s /最后修
改时间人为%s \r" % (
                        i.id, i.latest_modify_time, i.latest_modify_worker)
                    logFile.error(error_data)
                    with open(os.path.join(os.path.dirname(__file__),
                                           "DataBaseError_%s_%s.txt" % (
                                               data["filename"].split("/")
[-1], data["function"].split("_")[-1])),
                              'a+') as f:
                        f.write(error_data)
    except Exception as e:
        TestCaseException(e, data)

    finally:
        logFile.info("关闭数据库连接池")
        if oracle_instance:
            oracle_instance.__del__()
```

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

```
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

WEBPage模块

存放UI工具的地方（后续会移步至UTILS中)

BasePage.py

封装Selenium基类，以及定位元素方法

```
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
```

```python
log = Logger("DEBUG")

"""
selenium基类
本文件存放了selenium基类的封装方法
"""


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

    def __init__(self, driver):
        # self.driver = webdriver.Chrome()
        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"""
        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
```

LoginPlatform.py

封装两个简单的方法

LoginPlatform类实现登录平台操作，CreateWorkFlow类实现打开创建测试流程操作，后续可自定义封装，减少用例代码

```
from config.globalVars import G
```

```python
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
        self.port = G.Server_Port
        self.basePage = WebPage(driver=self.driver)
        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")
        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"])
```