金融项目第九天--课堂笔记

昨日回顾

目标：使用Python代码完成所有的接口自动化脚本的编写

（1）常见的步骤

- 定义接口类
  - 定义发送请求时的URL和参数（可以作为变量传递）
    - 对于部分不常改动的参数，可以设置默认值，减少脚本的工作量
  - 调用python Requests库中的get/post方法来发送请求
    - 单个消息体：data、json
    - 多个消息体：同时传递多个参数格式：data+files
  - 接收响应并返回
- 编写测试用例脚本
  - setup：在每个测试方法前执行
    - 接口API的初始化
    - session的初始化
    - 数据的初始化
  - teardown：在每个测试方法后执行
    - session的关闭
    - 数据的清除
      - 如果脚本中定义的变量是局部变量（只在本测试类中使用的变量），就应该在teardown中将数据清除
      - 如果脚本中定义的变量是全局变量（在其他测试类中也会使用），就需要在所有的脚本最后将数据清除
  - 测试用例的脚本
    - 按照用例的步骤来编写代码
    - 针对每个步骤，按照接口API的定义，准备需要传入的参数
    - 调用接口类中定义的方法来传入参数，发送请求
    - 接收响应，并对响应结果进行断言

（2）BeautifulSoup库

- 作用：python提供的专门用于解析HTML/XML格式的数据内容
- 安装：pip install BeautifulSoup4
- 使用：
  - 导包：from bs4 import BeautifulSoup
  - 初始化BeautifulSoup对象
    - 当需要解析的对象是一个HTML文件时，初始化方法：
      - soup = BeautifulSoup(open(file),'html.parser')
    - 当需要解析的对象是一个HTML的字符串时，初始化方法：
      - soup = BeautifulSoup(html,'html.parser')
  - html元素的获取：

- **获取标签的对象—— soup.标签名**
- 获取标签对象的标签名称——soup.标签名.name
- 获取标签对象的值——soup.标签名.string
- **获取标签对象的属性的值 ——soup.标签名['属性名']**
- **存在多个相同的标签，返回所有的标签时 —— soup.findall('标签名')**

学习目标

- 清除接口自动化测试中测试数据
- 能够针对python自动化脚本中的测试数据进行参数化
- 能够配置持续集成任务，自动运行接口自动化的代码脚本
- 能够使用Python+Request编写APP端接口自动化测试脚本（加密接口）

清除测试数据：

连接数据库封装基础类：

```
class DButils:
    @classmethod
    def get_conn(cls):
        conn =
pymysql.connect(app.DB_host,app.DB_user,app.DB_password,app.database_mem,autocom
mit=True)
        return conn

    @classmethod
    def close_conn(cls,cursor = None,conn = None):
        if cursor:
            cursor.close()
        if conn:
            conn.close()

    @classmethod
    def execute_sql(cls,sql):
        try:
            conn = cls.get_conn()
            cursor = conn.cursor()
            cursor.execute(sql)
        except Exception as e:
            conn.rollback()
        finally:
            cls.close_conn()
```

删除对应测试数据：

```
    @classmethod
    def tearDownClass(cls) -> None:
        sql1 = "delete from mb_member_register_log where phone in
('13099881111','13099881112','13099881113','13099881114');";
        DButils.execute_sql(sql1)
        logging.info('delete sql1 = {}'.format(sql1))
```

```
        sql2 = "delete i.* from mb_member_login_log i INNER JOIN mb_member m on
i.member_id = m.id WHERE m.phone in
('13099881111','13099881112','13099881113','13099881114');";
        DButils.execute_sql(sql2)
        logging.info('delete sql1 = {}'.format(sql2))
        sql3 = "delete i.* from mb_member_info i INNER JOIN mb_member m on
i.member_id = m.id WHERE m.phone in
('13099881111','13099881112','13099881113','13099881114');";
        DButils.execute_sql(sql3)
        logging.info('delete sql1 = {}'.format(sql3))
        sql4 = "delete from mb_member WHERE phone in
('13099881111','13099881112','13099881113','13099881114');";
        DButils.execute_sql(sql4)
        logging.info('delete sql1 = {}'.format(sql4))
```

数据驱动：

定义数据文件：

1、定义存放测试数据的目录，如data

2、分模块定义数据文件

3、根据业务编写测试数据，编写的数据格式（分为三部分）

- 描述：给人看，写清楚这个测试数据的目的是什么
- 请求参数：所有的请求参数都在参数文件中定义
- 响应结果：所有要断言的响应结果在参数文件中定义

如获取图片验证码：

```
{
  "test_get_img_verify_code": [
    {
      "desc": "获取图片验证码--参数为随机小数",
      "type": "float",
      "status_code": 200
    },
    {
      "desc": "获取图片验证码--参数为随机整数",
      "type": "int",
      "status_code": 200
    },
    {
      "desc": "获取图片验证码--参数为空",
      "type": "null",
      "status_code": 404
    },
    {
      "desc": "获取图片验证码--参数为随机字母",
      "type": "char",
      "status_code": 400
    }
  ]
}
```

如注册：


```
{
  "test_register": [
    {"desc":"输入必填项，成
功","phone":"13033447711","pwd":"test123","imgVerifyCode":"8888","phoneCode":"66
6666","dyServer":"on","invite_phone":"","status_code": 200,"status":
200,"description": "注册成功"},
    {"desc":"输入所有项，成
功","phone":"13033447712","pwd":"test123","imgVerifyCode":"8888","phoneCode":"66
6666","dyServer":"on","invite_phone":"13012345678","status_code": 200,"status":
200,"description": "注册成功"},
    {"desc":"输入重复手机号，失
败","phone":"13033447711","pwd":"test123","imgVerifyCode":"8888","phoneCode":"66
6666","dyServer":"on","invite_phone":"","status_code": 200,"status":
100,"description": "手机已存在!"},
    {"desc":"输入密码为空，失
败","phone":"13033447713","pwd":"","imgVerifyCode":"8888","phoneCode":"666666","
dyServer":"on","invite_phone":"","status_code": 200,"status": 100,"description":
"密码不能为空"},
    {"desc":"输入图片验证码错误，失
败","phone":"13033447714","pwd":"test123","imgVerifyCode":"1234","phoneCode":"66
6666","dyServer":"on","invite_phone":"","status_code": 200,"status":
100,"description": "验证码错误!"},
    {"desc":"输入短信验证码错误，失
败","phone":"13033447714","pwd":"test123","imgVerifyCode":"8888","phoneCode":"12
3456","dyServer":"on","invite_phone":"","status_code": 200,"status":
100,"description": "验证码错误"},
    {"desc":"不同意注册协议，失
败","phone":"13033447714","pwd":"test123","imgVerifyCode":"8888","phoneCode":"66
6666","dyServer":"off","invite_phone":"","status_code": 200,"status":
100,"description": "请同意我们的条款"}
  ]
}
```


读取数据文件，

读取验证码的数据文件：

```
def get_img_verify_code_data():
    test_data = []
    #读取数据文件
    file_path = app.BASE_DIR + '/data/getImgCode.json'
    #将数据文件中的内容读取到参数中
    with open(file_path,'r',encoding='utf-8') as f:
        json_data = json.load(f)
        data_list = json_data.get('test_get_img_verify_code')
        for case_data in data_list:

 test_data.append((case_data.get("type"),case_data.get('statusCode')))
    print(test_data)
    return test_data
```

读取注册的数据文件：

```
def read_register_data(file_name):
    #注册的测试数据的文件路径
    file = app.BASE_DIR + "/data/" + file_name
    test_case_data = []
    with open(file,encoding="utf-8") as f:
        #将json的数据格式，转化为字典的数据格式
        register_data = json.load(f)
        #获取所有的测试数据的列表
        test_data_list = register_data.get("test_register")
        #依次读取测试数据列表中的每一条数据，并进行相应字段的提取
        for test_data in test_data_list:

 test_case_data.append((test_data.get("phone"),test_data.get("pwd"),test_data.ge
t("imgVerifyCode"),test_data.get("phoneCode"),test_data.get("dyServer"),test_dat
a.get("invite_phone"),test_data.get("status_code"),test_data.get("status"),test_
data.get("description")))
        print("test_case_data = {}".format(test_data_list))
    return test_case_data
```

读取测试数据的统一方法：

```
#定义统一的读取所有参数数据文件的方法
def read_param_data(filename,method_name,param_names):
    #filename:  参数数据文件的文件名
    #method_name: 参数数据文件中定义的测试数据列表的名称，如：test_get_img_verify_code
    #param_names: 参数数据文件一组测试数据中所有的参数组成的字符串，
如："type,status_code"

    #获取测试数据文件的文件路径
    file = app.BASE_DIR + "/data/" + filename
    test_case_data = []
    with open(file,encoding="utf-8") as f:
        #将json字符串转换为字典格式
        file_data = json.load(f)
        #获取所有的测试数据的列表
        test_data_list = file_data.get(method_name)
        for test_data in test_data_list:
            #先将test_data对应的一组测试数据，全部读取出来，并生成一个列表
            test_params = []
            for param in param_names.split[","]:
                #依次获取同一组测试数中每个参数的值，添加到test_params中，形成一个列表
                test_params.append(test_data.get(param))
            #每完成一组测试数据的读取，就添加到test_case_data后，直到所有的测试数据读取完毕
            test_case_data.append(test_params)
    print("test_case_data = {}".format(test_case_data))
    return test_case_data
```

并修改测试脚本：

读取验证码的脚本：

```
    #读取定义的参数文件，并测试
    @parameterized.expand(read_imgVerify_data("imgVerify.json"))
    def test01_get_img_verify_code(self,type,status_code):
```

```
        #根据不同的type类型准备不同的参数数据
        r = ''
        if type == 'float':
            r= str(random.random())
        elif type == 'int':
            r = str(random.randint(10000000,90000000))
        elif type == 'char':
            r = ''.join(random.sample("abcdedfhijklmn",8))
        #发送请求
        response = self.login_api.getImgCode(self.session,r)
        logging.info("r = {} response = {}".format(r,response))
        #对响应结果进行断言
        self.assertEqual(status_code,response.status_code)
```

注册的脚本：

```
            @parameterized.expand(read_register_data("register.json"))
    def
test10_register(self,phone,pwd,imgVerifyCode,phoneCode,dyServer,invite_phone,sta
tus_code,status,description):
        #1、获取图片验证码成功
        r = random.random()
        #调用接口类中的接口
        response = self.login_api.getImgCode(self.session,str(r))
        #接收接口的返回结果，进行断言
        self.assertEqual(200,response.status_code)
        #2、获取短信验证码成功
        # 定义参数（正确的手机号和验证码）
        # 调用接口类中的发送短信验证码的接口
        response = self.login_api.getSmsCode(self.session,phone,self.imgCode)
        logging.info("get sms code response = {}".format(response.json()))
        #对收到的响应结果，进行断言
        assert_utils(self,response,200,200,"短信发送成功")
        #3、使用参数化的测试数据进行注册，并返回对应的结果
        #发送注册请求
        response =
self.login_api.register(self.session,phone,pwd,imgVerifyCode,phoneCode,dyServer,
invite_phone)
        logging.info("register response = {}".format(response.json()))
        #对收到的响应进行断言
        assert_utils(self,response,status_code,status,description)
```

统一读取测试数据时，测试脚本调用数据的方法：

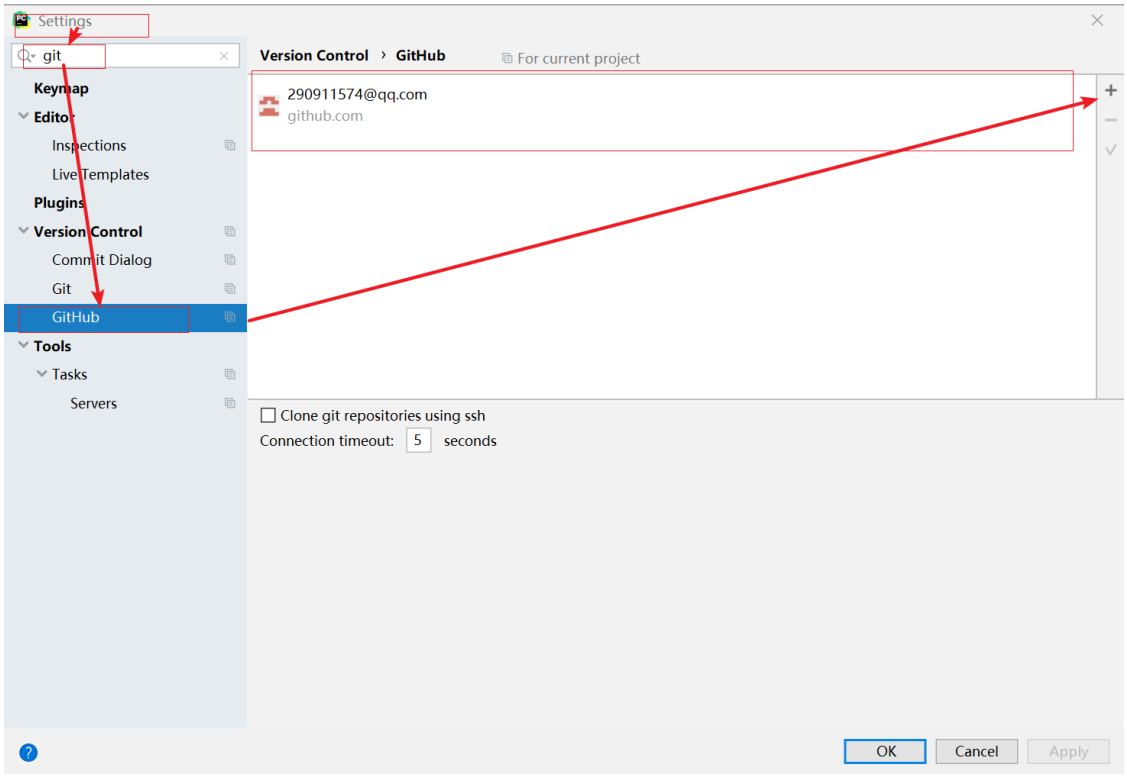
```
  @parameterized.expand(read_param_data("register.json","test_register","phone,p
wd,imgVerifyCode,phoneCode,dyServer,invite_phone,status_code,status,description"
))
Note：下面方法部分与之前的脚本完全相同
```

持续集成

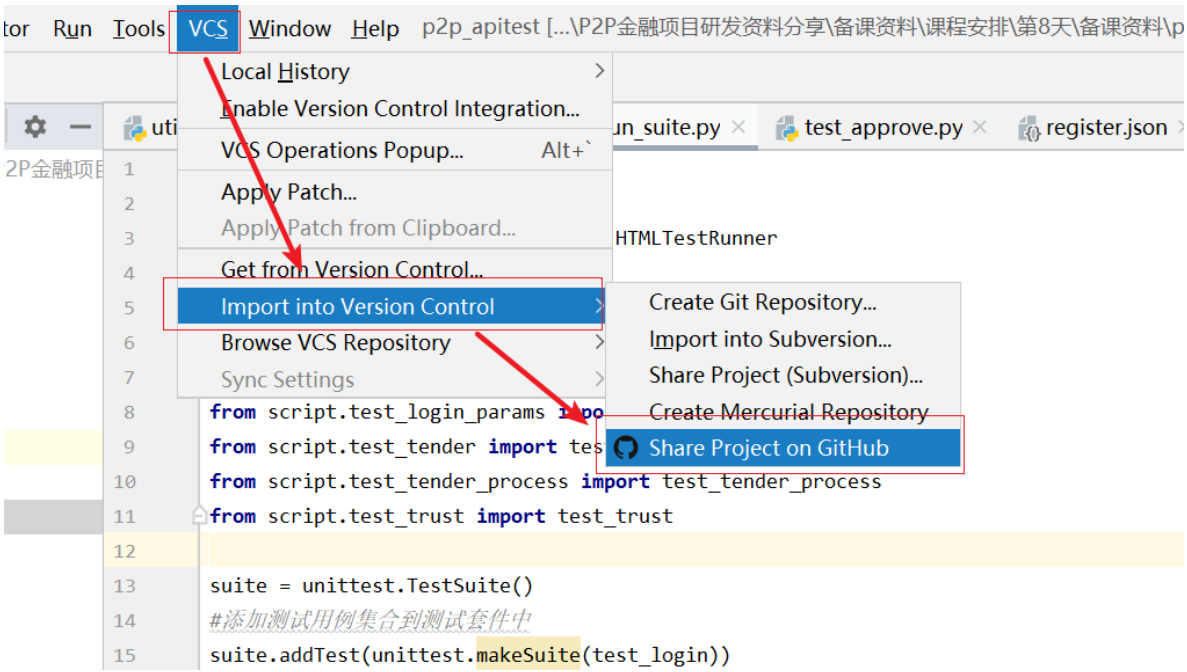
第一步：通过github管理代码（配置代码同步到服务器）：

初次上传：

- 申请github的账号，在pycharm中设置git

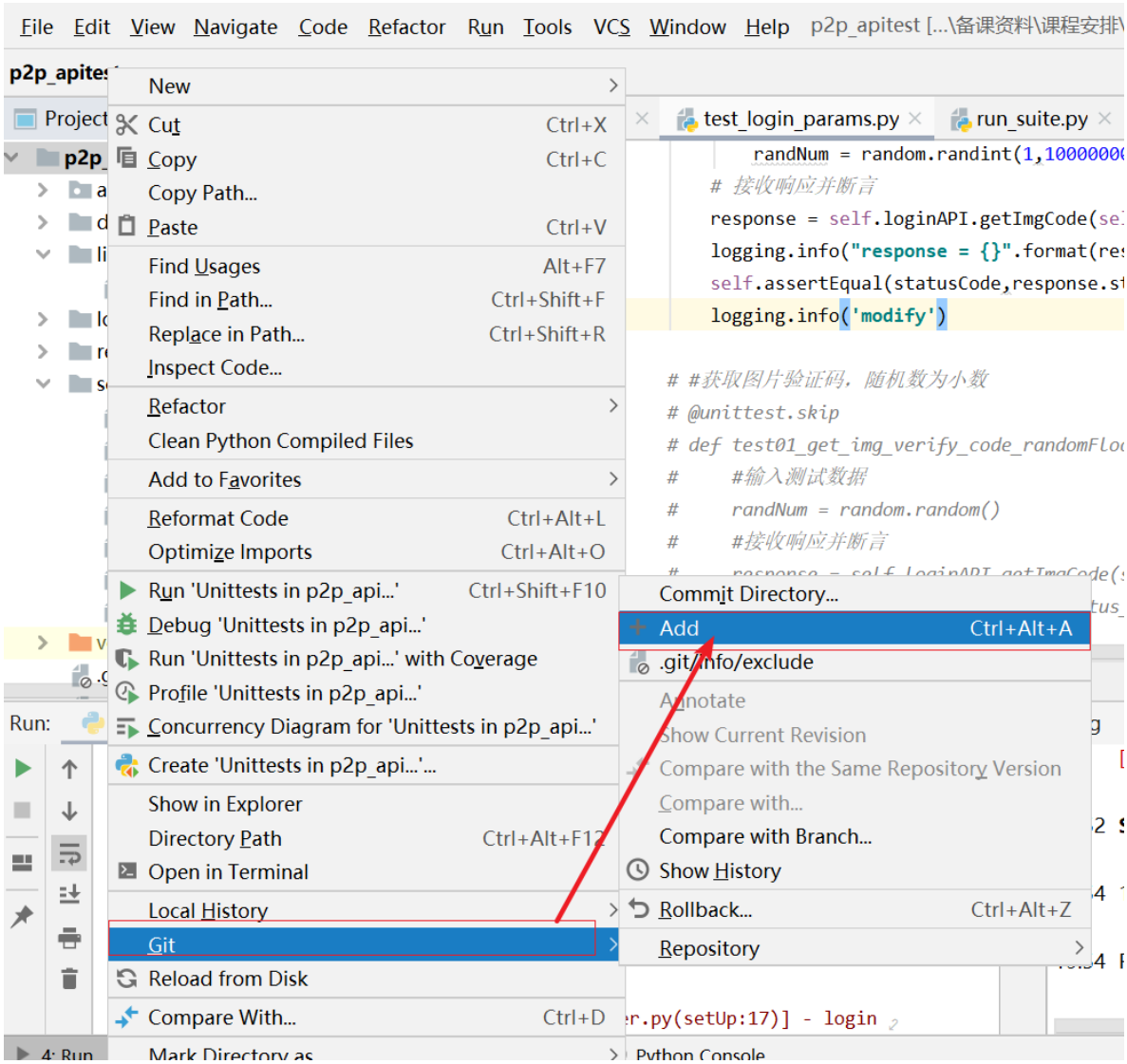


同步本地代码到github服务器

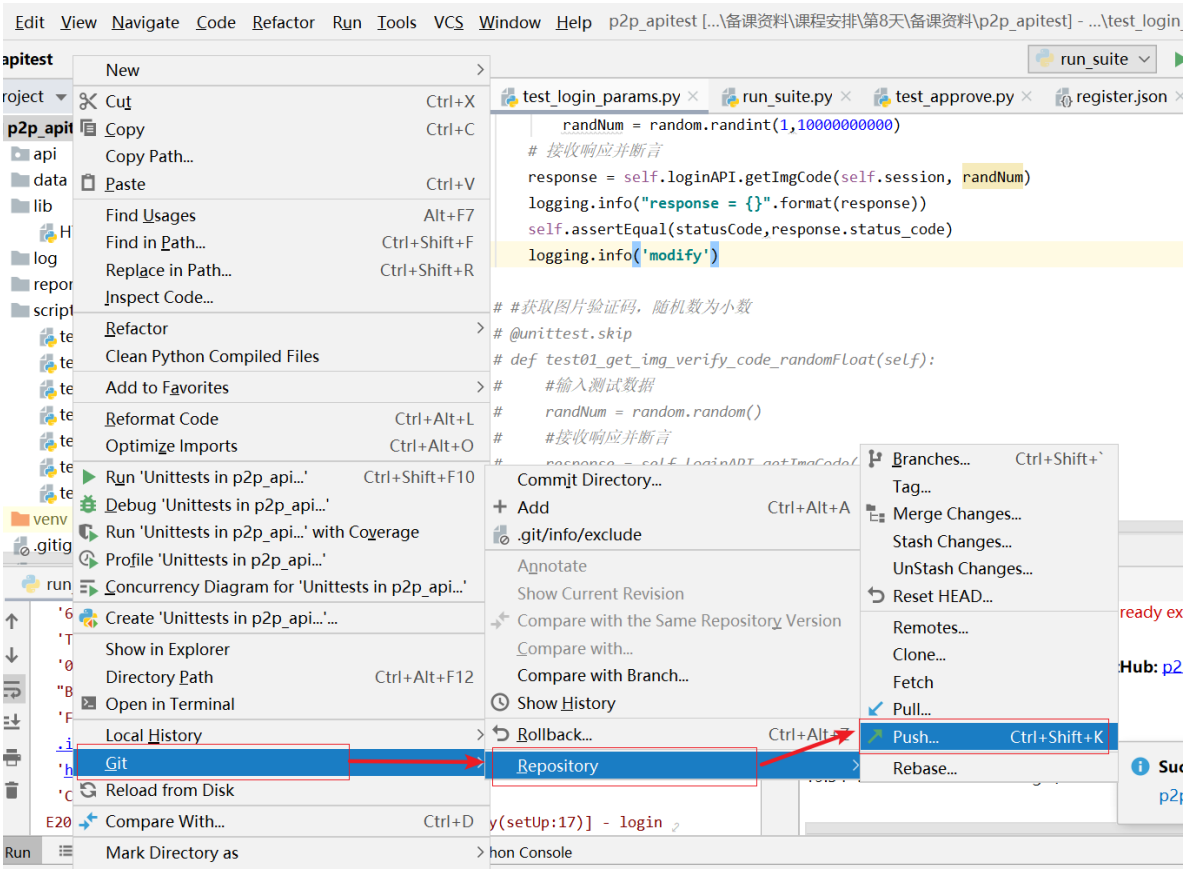


后续更新代码：

- 选中已更新的代码文件，右键git—add



- 右键git—commit，并填写更新的注释
- 右键git—Repository—push，完成代码更新到服务器



第二步：通过jenkins配置定时测试任务

1、添加Jenkins的测试项目

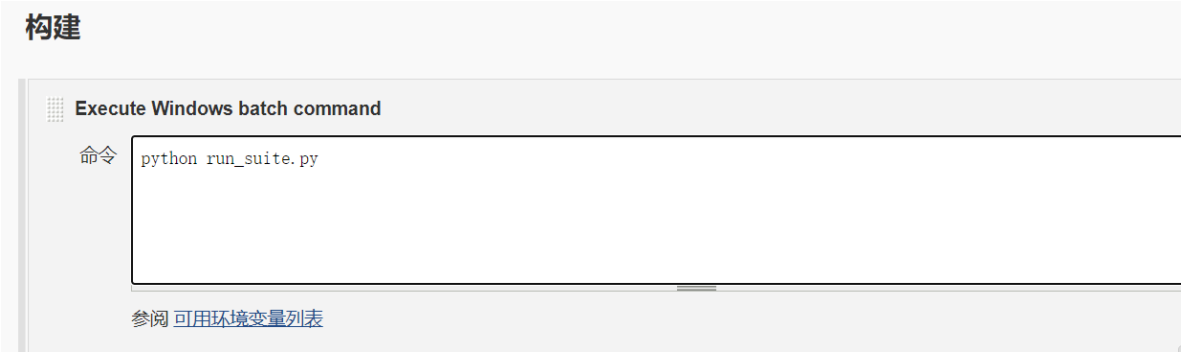
2、配置Jenkins的测试项目

- General——Discard old builds（配置构建包在服务器上存储的天数和包的个数）

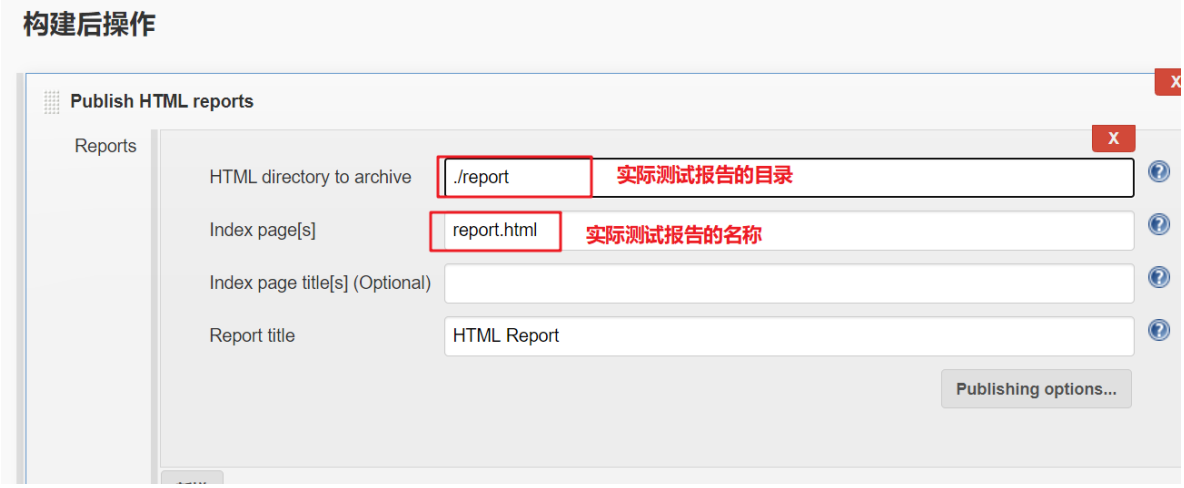
- 源码管理 —— Git（配置github中下载源代码的url地址和代码分支：默认为master）

- 设置构建触发器 （指定测试脚本运行的时间）

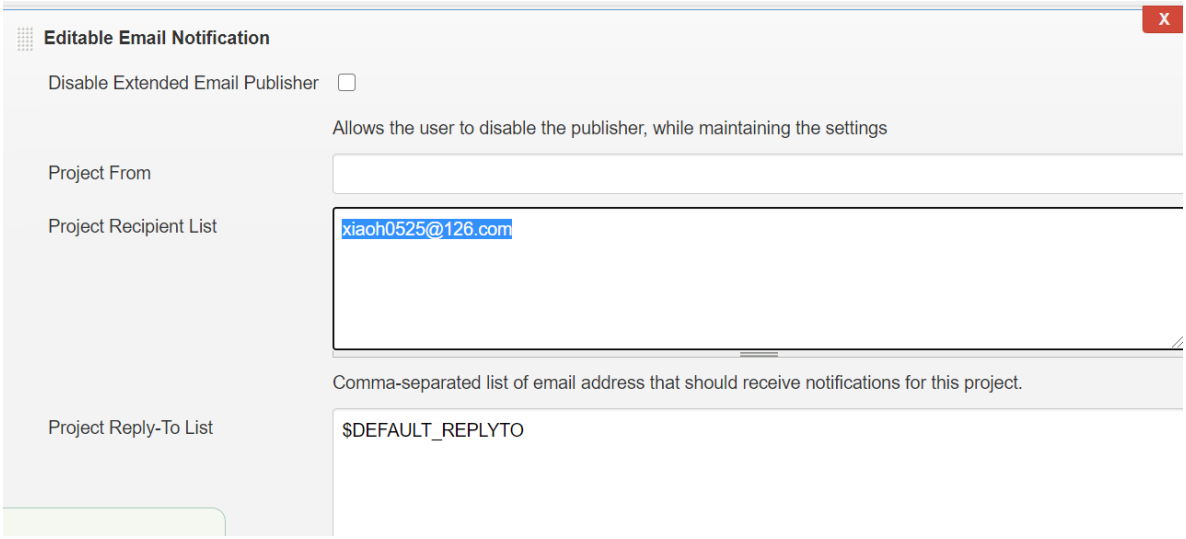
- 配置构建命令



- 配置构建后操作——Publish HTML report（测试报告的路径和对应的文件）



- 配置构建后操作—— Editable Email Notification（配置收件人的邮箱、邮件的模块格式、触发邮件的动作）



加解密接口的处理：

团队测试工作进展

- 每个人能够说出自己的工作进度
- 每个人能够说出自己接下来的要做的工作
- 组长告知小组成员目前小组的整体工作进度（单独发送）
- 组长能够根据小组的整体工作进度及时调整工作安排