# 目 录 (五十六期·上)

# 基于 Pytest 与 Postman 的数据自动采集接口自动化测试实践

◆ 作者：kayie

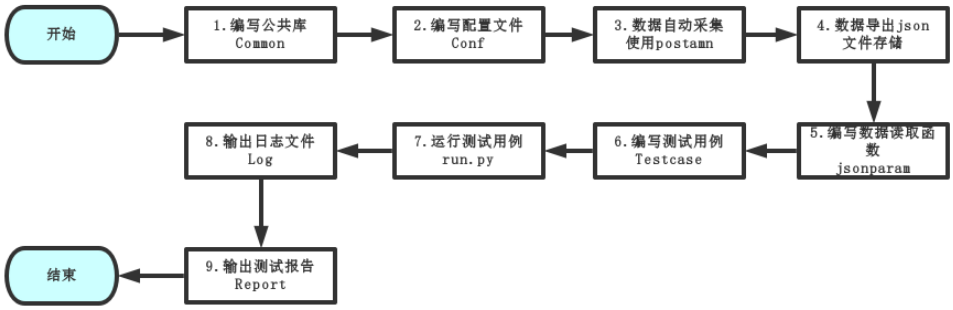
## 一、前言

本文跟大家介绍的是基于 Python 测试框架 Pytest 与 Postman 数据自动采集的接口自动化测试实践方案，首先了解下为什么是基于 Pytest 框架而不是使用 Unitest？网上的对比资料很多，概括起来就是 Pytest 相较于 unittest 代码更加的简洁和灵活，最为跳跃的一点就是 fixture 机制，并且 Pytest 有很多的第三方插件可以扩展和继承，大家可以深入去查一查。

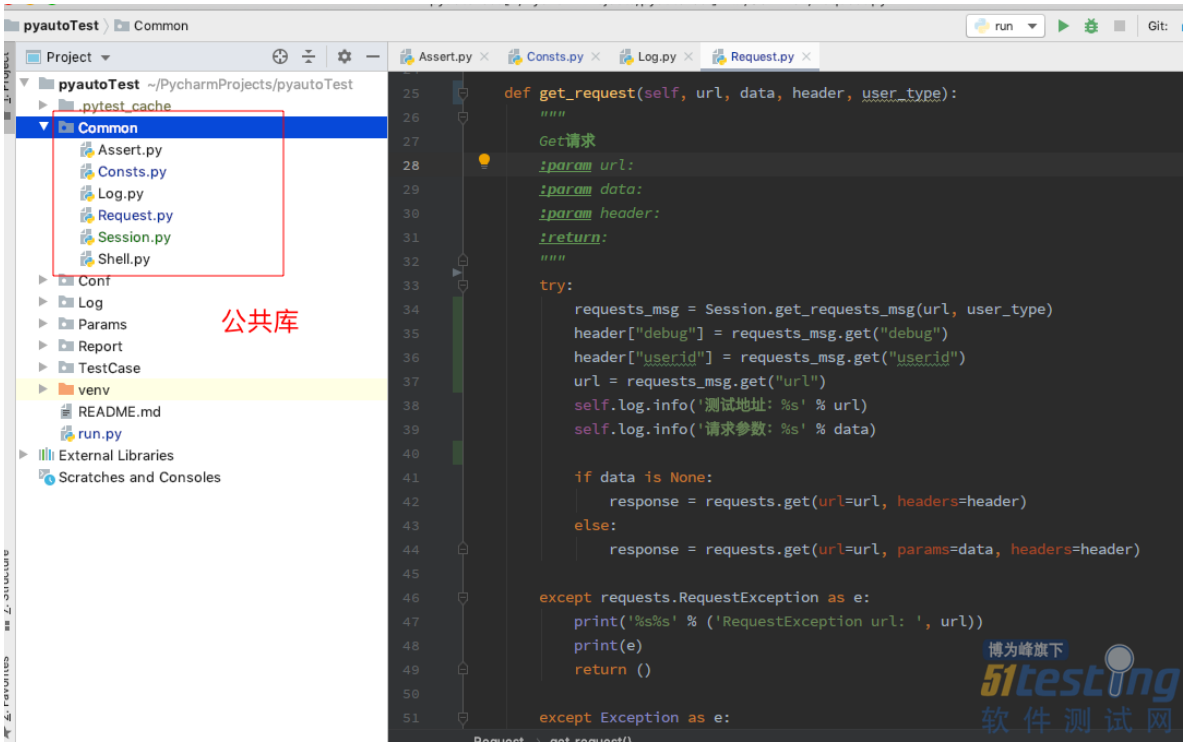
第二个了解的是为什么要做数据自动采集，做数据驱动很多教程推荐的是将数据写在 excel 中，然后通过程序去读取。但我更推荐的是通过软件自动采集所需数据，这样可以大大节省手动在 excel 录入数据的时间。推荐使用 Postman 采集数据，它是一款做开发人手必备的接口调试工具，几乎能发送所有类型的 HTPP 请求，在打开代理模式以后能够自动抓取 pc 端浏览器或者 APP 端请求的接口数据，存储并导出后为 json 格式的数据源，简单方便。

## 二、自动化框架流程

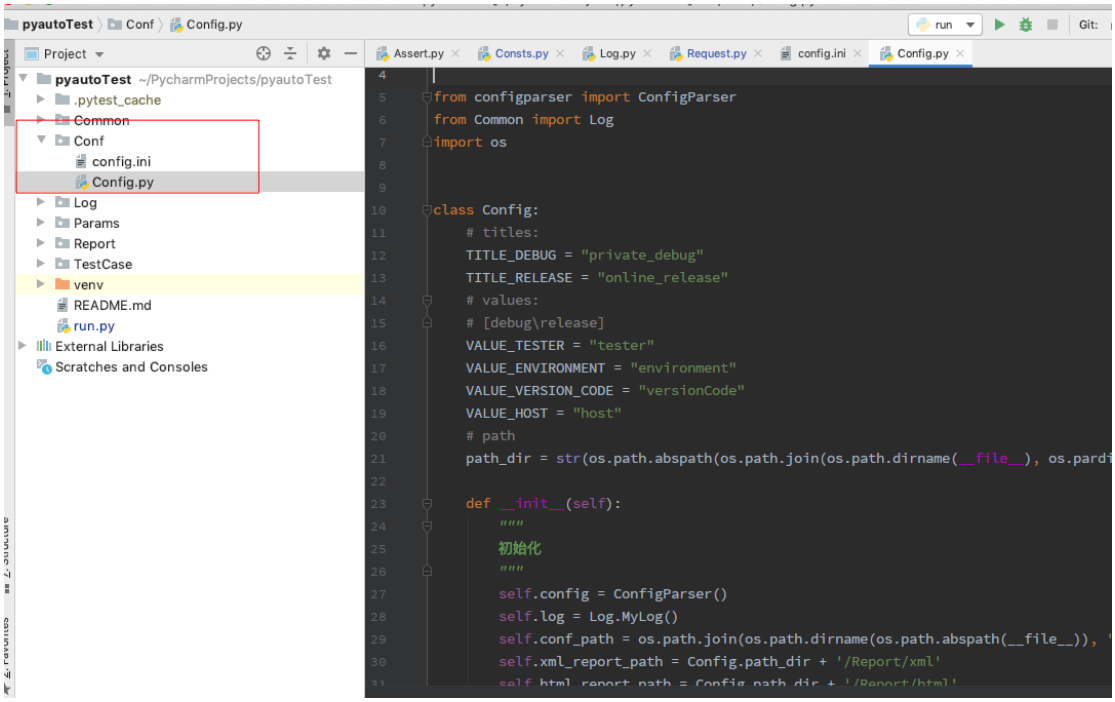
整个代码框架除了数据来源从 Postman 采集，其他所有封装的公共库以及具体的测试用例均使用代码编辑器如 pycharm 编写 python 代码，其中需要用到的 pytest、requests、allure 等库可以直接使用 pip 命令安装。从数据采集到完成自动化测试并输出测试报告的流程如下图:



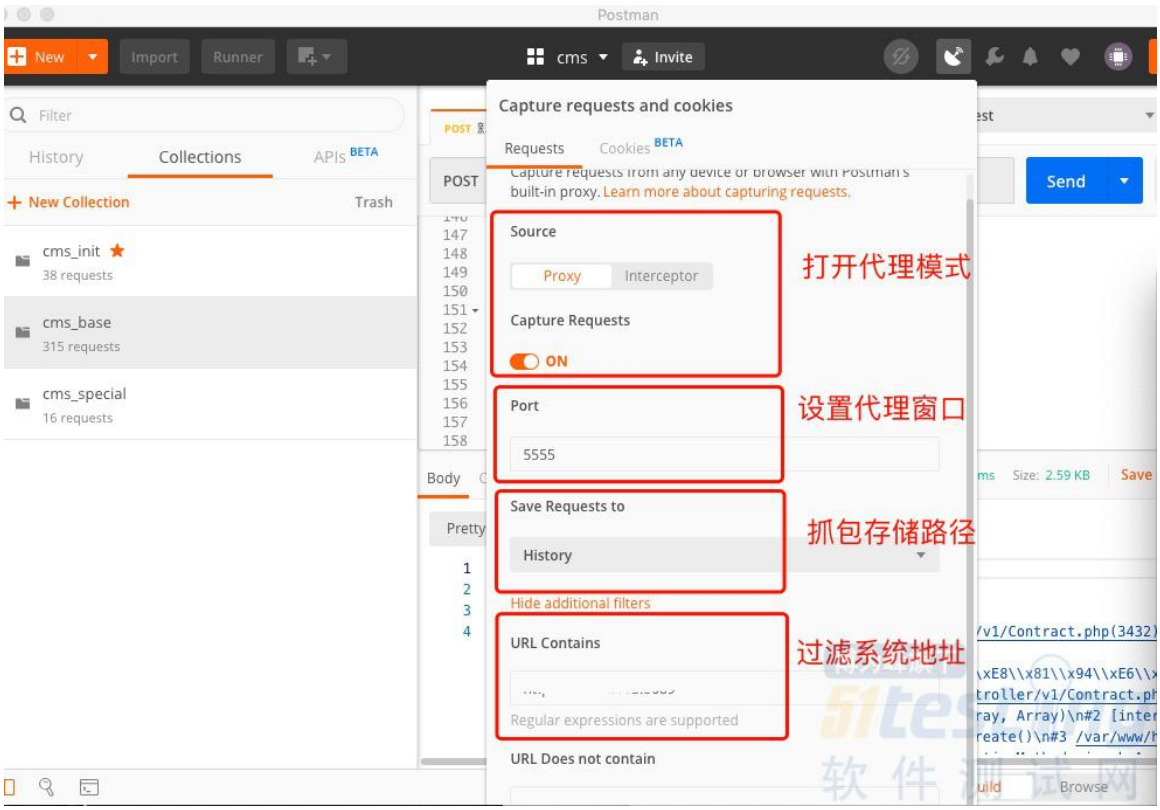

**1）编写公共库：**如果某个函数需要被多次引用，那么就可以封装成公共函数，这些封装的函数主要有：断言、全局常量、log 日志、发送请求、用户 session、加密等。跟业务无关，可以先封装好。



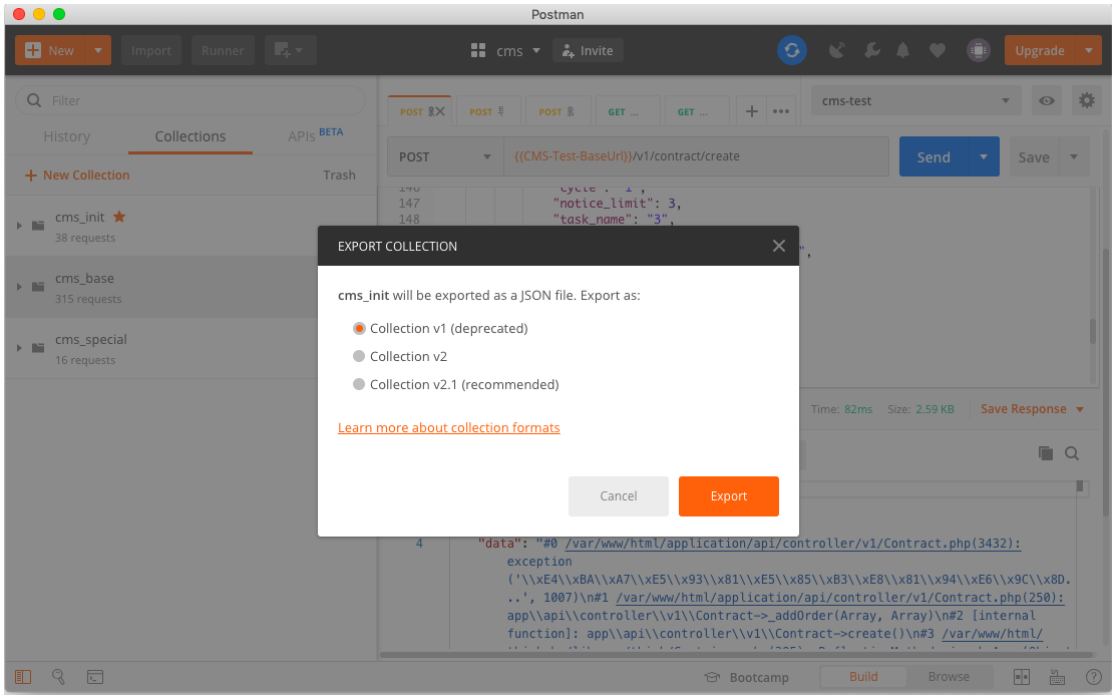

**2）编写配置文件：**配置文件内容如环境参数、文件存储路径、版本号，以及配置文件的读写封装等，也是编写测试用例之前必须封装好的内容。



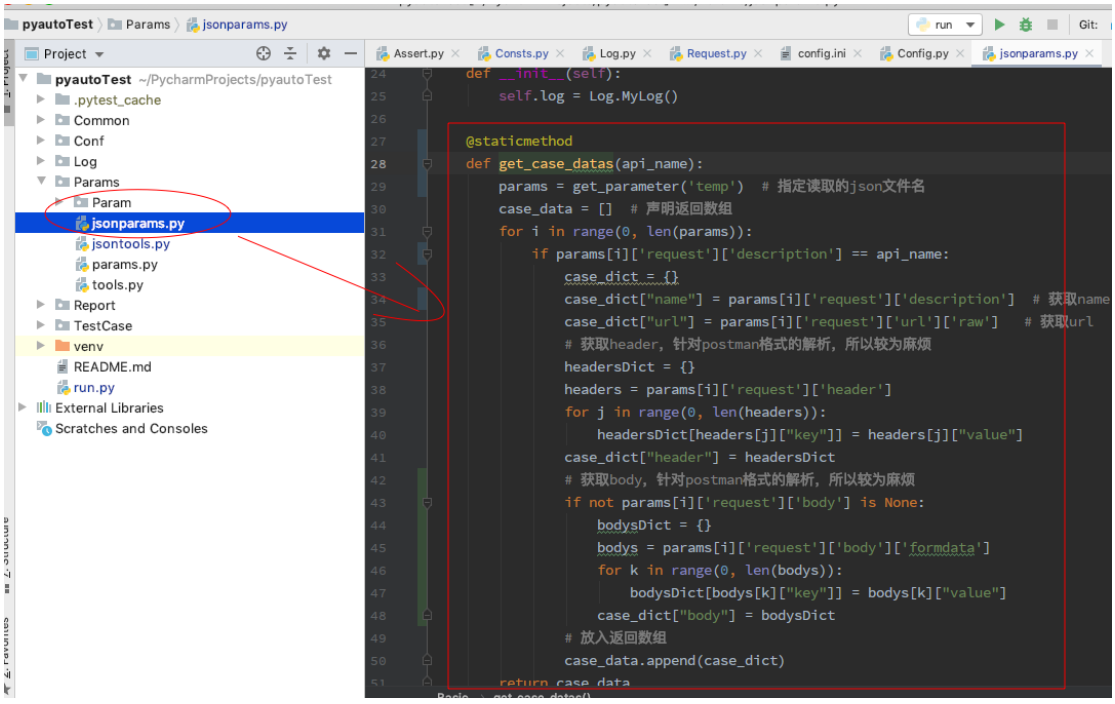

**3）数据自动采集：**如下图所示，打开代理模式，设置代理端口如：5555，设置抓包的存储路径，比如一个功能流程就可以单独存一个集合，最后设置过滤地址为本系统，以免抓到与系统无关的接口 。



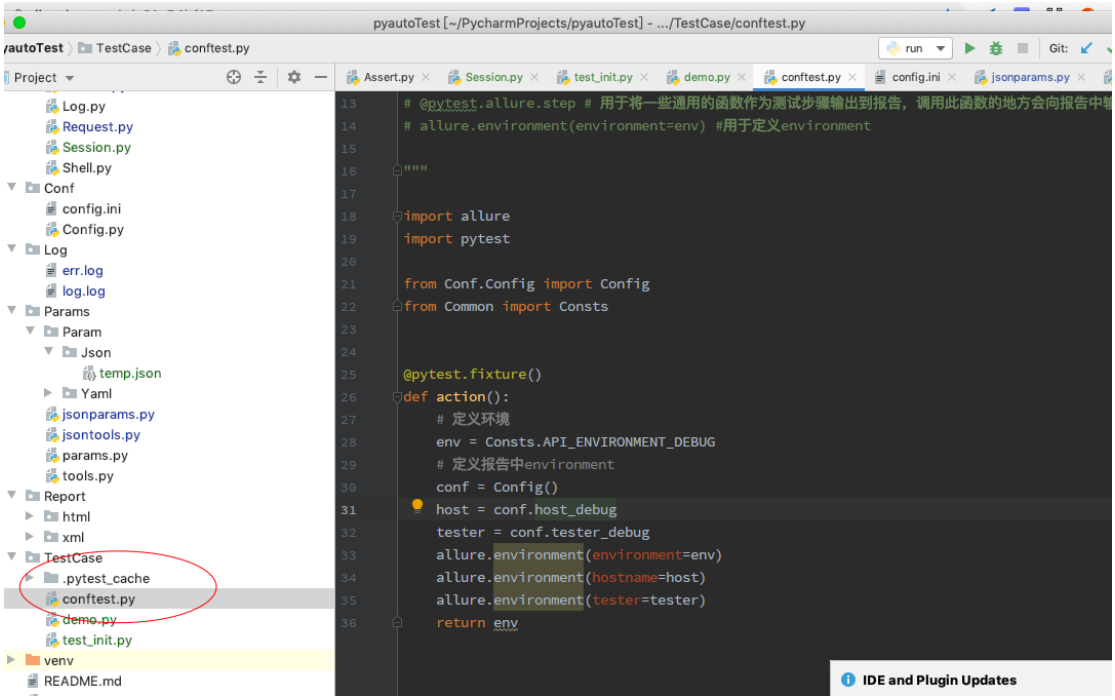

**4）数据导出 json 文件存储：**选中某个集合后，右键选择 export 导出，选择第一个 json 数据格式。导出的数据源放入项目对应的目录 Params->json 下



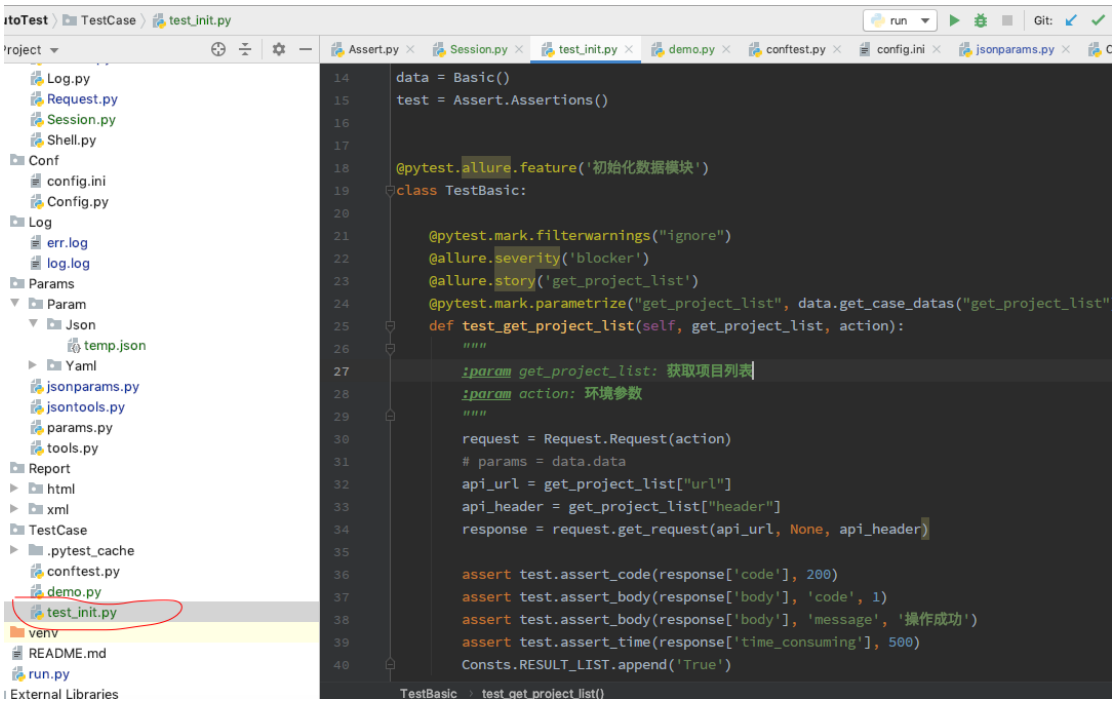

**5）编写数据读取函数：**编写 jsonparam.py 函数，解析 Postman 格式的 json 数据。因为 Postman 有些数据不是我们需要的 有些需要拼接，所以需要单独封装一个转换格式的函数，方便测试用例数据读取使用。



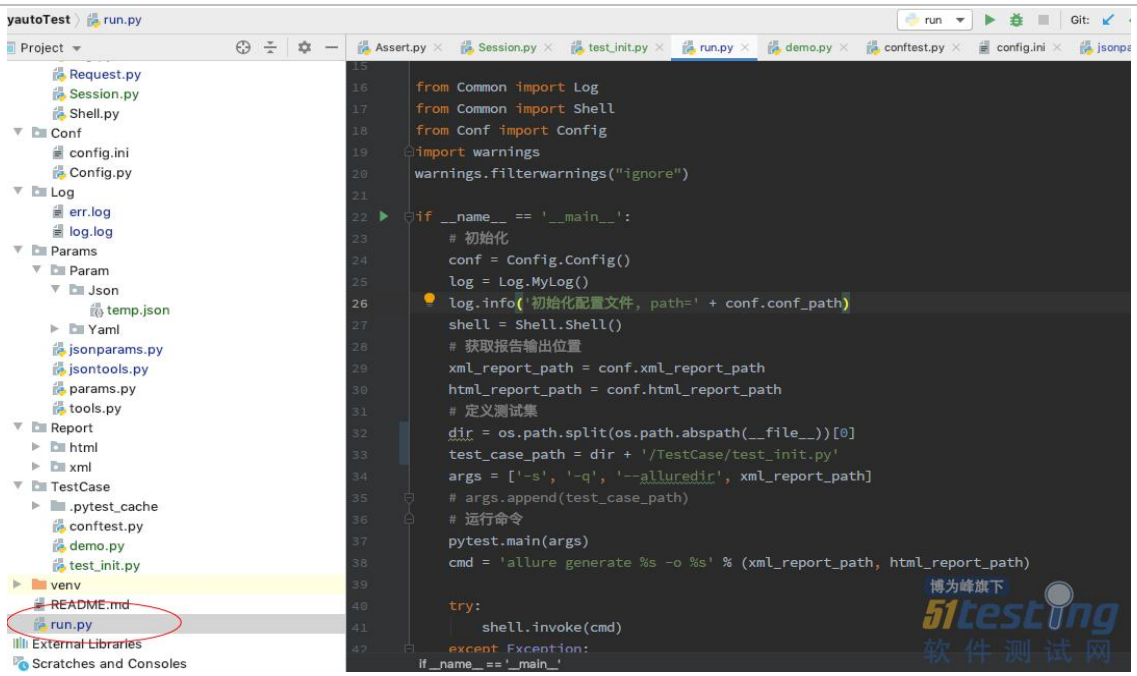

**6）编写测试用例：**在 Testcase 下编写测试用例的前置和后置参数文件 conftest.py 文件（此文件名为是固定的，不能写别的）。



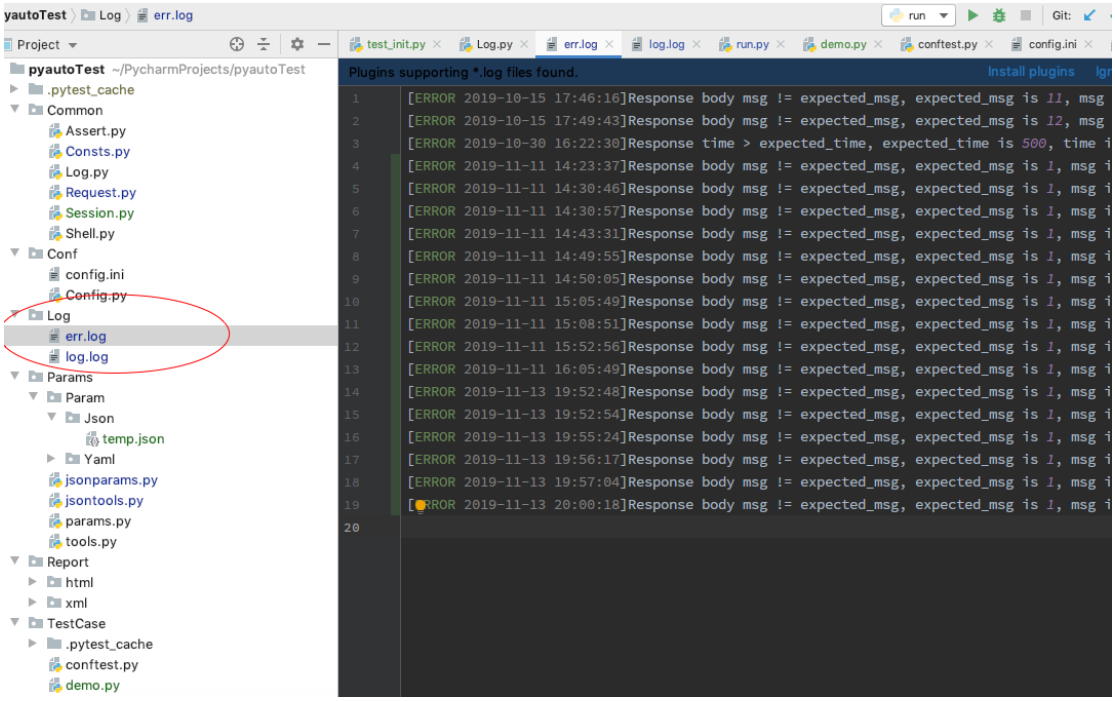

编写具体的测试用例，测试用例均要使用 test 开头或结尾（否则框架无法识别）




**7）运行测试用例：**如果需要运行 Testcase 下所有测试用例，可以在根目录建 run.py 文件，若只需要单独运行某个功能的测试用例，也可以在具体的测试文件的 main 函数里面使用 Pytest.main("test_xxx.py")命令运行。




**8）输出日志文件：**过程中使用 self.log.info('请求参数：%s' % data)打印日志信息，作为记录和调试使用。




**9）输出测试报告：**最后可以集成 allure 插件，输出更加直观漂亮的测试报告。集成过程网上也有很多教程，但是坑非常多，大家需要注意以下内容：

① 建议使用 Pytest 3.8.0 版本

命令：pip install Pytest

注意：勿使用 Pytest 过高版本，且勿使用 allure-Pytest 插件，会一直报错 Pytest 找不到 allure 错

误，网上的办法也解决不了，亲测使用 Pytest 3.8.0 搭配 pytest-allure-adaptor 1.7.10 可以解决

② 建议使用 Pytest-allure-adaptor 1.7.10 版本

命令：pip install pytest-allure-adaptor

③ 需要安装 jdk 1.8 以上版本

④ 需要安装 allure-commandline（先安装 npm 包）

命令：npm install -g allure-commandline --save-dev

**接下来按照在 run.py 文件中写的代码**

⑤ Pytest 命令基础上加--alluredir，生成 xml 报告。

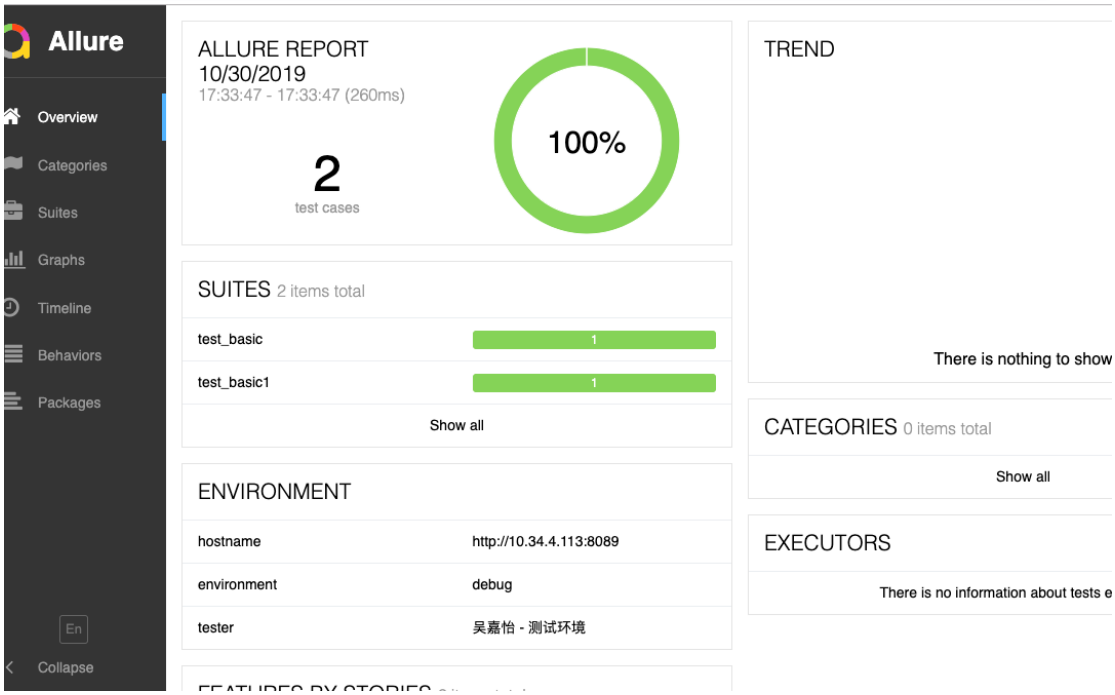
pytest -s -q --alluredir [xml_report_path]

⑥ 使用 Command Tool 来生成我们需要的美观报告。

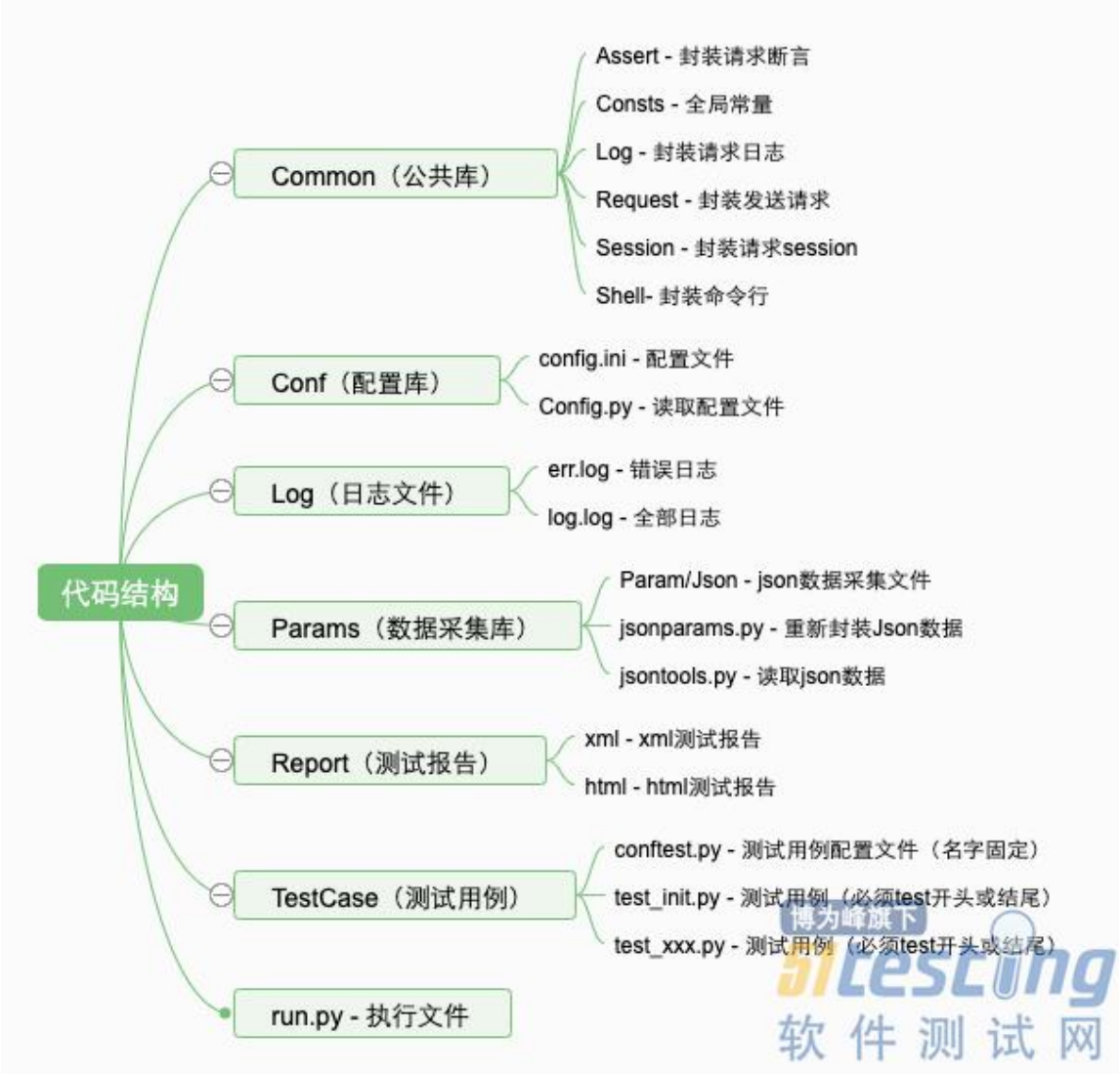
allure generate [xml_report_path] -o [html_report_path]

⑦ 直接用 chrome 浏览器打开报告，报告会是空白页面

解决办法：在 pycharm 中右击 index.html 选择打开方式 Open in Browser 即可




三、代码结构&源码




框架代码分为如下几个模块：Common 公共库、Conf 配置库、Log 封装日志、Params 数据源和读取数据、Report 测试报告、TestCase 测试用例、run.py 执行文件，几个重要的函数如下，其他代码已经放到开源 github 上，大家自行下载。

**源码地址：** https://github.com/kayie77/PyTestApiAuto

**1）Common->Assert.py 封装断言**

直接使用 python 的 assert 断言函数，用于判断一个表达式，使用方式：

assert test.assert_code(response['code'], 200)

```python
class Assertions:
    def __init__(self):
        self.log = Log.MyLog()

    def assert_code(self, code, expected_code):
        """
        验证response状态码
        :param code:
        :param expected_code:
        :return:
        """
        try:
            assert code == expected_code
            return True
        except:
            self.log.error("statusCode error, expected_code is %s, statusCode is %s " % (expected_code, code))
            Consts.RESULT_LIST.append('fail')

            raise
```

### 2）Common->Request.py 封装请求

目的是封装公共请求参数和封装返回内容

```python
    def post_request(self, url, data, header, user_type):
        try:
            requests_msg = Session.get_requests_msg(url, user_type)
            header["debug"] = requests_msg.get("debug")
            header["userid"] = requests_msg.get("userid")
            url = requests_msg.get("url")
            self.log.info('测试地址: %s' % url)
            self.log.info('请求参数: %s' % data)

            if data is None:
                response = requests.post(url=url, headers=header)
            else:
                response = requests.post(url=url, params=data, headers=header)

        except requests.RequestException as e:
            print('%s%s' % ('RequestException url: ', url))
            print(e)
            return ()

        except Exception as e:
            print('%s%s' % ('Exception url: ', url))
            print(e)
            return ()

        time_consuming = response.elapsed.microseconds/1000    # time_consuming为响应时间，单位为毫秒
        time_total = response.elapsed.total_seconds()   # time_total为响应时间，单位为秒
        Common.Consts.STRESS_LIST.append(time_consuming)
        response_dicts = dict()
        response_dicts['code'] = response.status_code
        try:
```

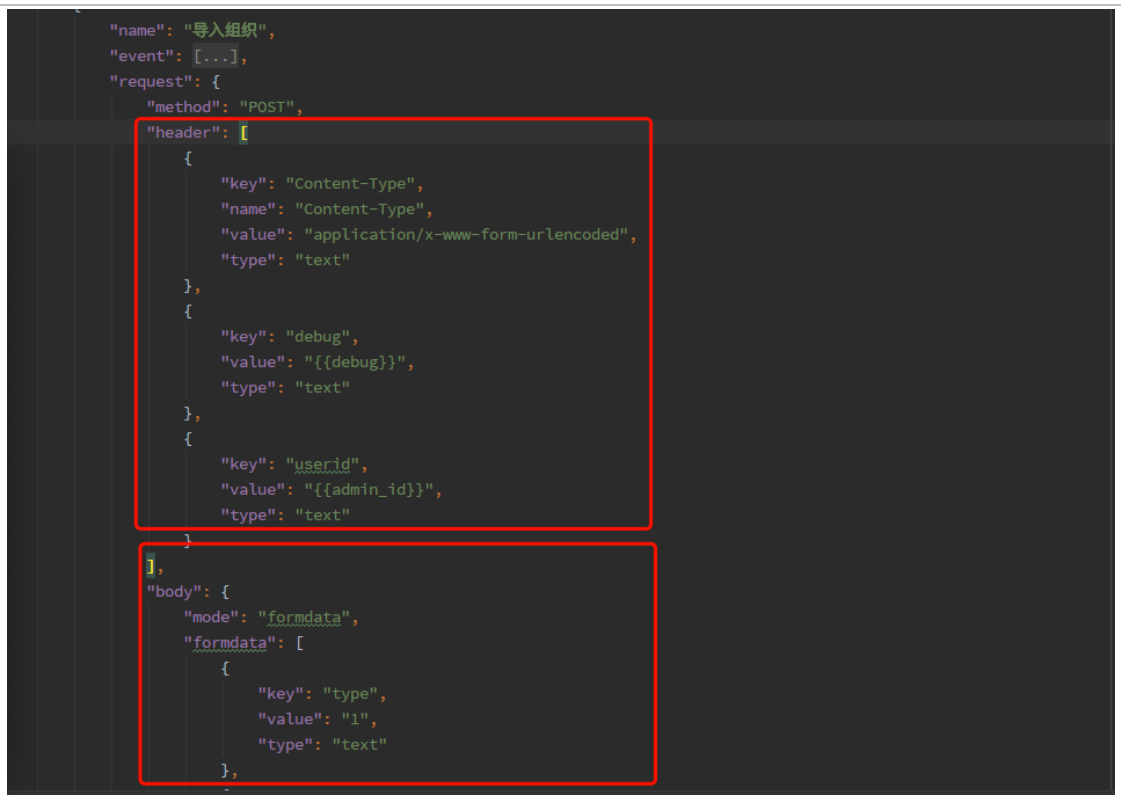
### 3）Conf->Config.py 读取配置文件

在 config.ini 定义好文件内容，需要再写一个读写配置文件的函数

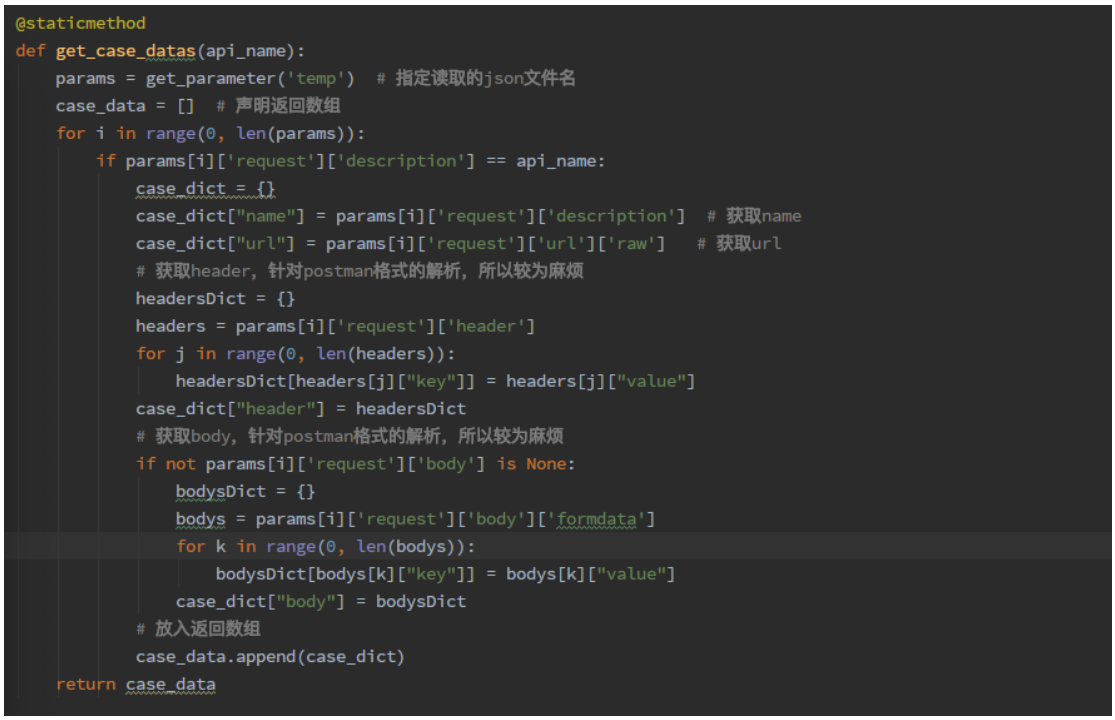
```
VALUE_ENVIRONMENT = "environment"
VALUE_VERSION_CODE = "versionCode"
VALUE_HOST = "host"
# path
path_dir = str(os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir)))

def __init__(self):
    """
    初始化
    """
    self.config = ConfigParser()
    self.log = Log.MyLog()
    self.conf_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'config.ini')
    self.xml_report_path = Config.path_dir + '/Report/xml'
    self.html_report_path = Config.path_dir + '/Report/html'

    if not os.path.exists(self.conf_path):
        raise FileNotFoundError("请确保配置文件存在！")

    self.config.read(self.conf_path, encoding='utf-8')
    self.tester_debug = self.get_conf(Config.TITLE_DEBUG, Config.VALUE_TESTER)
    self.environment_debug = self.get_conf(Config.TITLE_DEBUG, Config.VALUE_ENVIRONMENT)
    self.versionCode_debug = self.get_conf(Config.TITLE_DEBUG, Config.VALUE_VERSION_CODE)
    self.host_debug = self.get_conf(Config.TITLE_DEBUG, Config.VALUE_HOST)

    self.tester_release = self.get_conf(Config.TITLE_RELEASE, Config.VALUE_TESTER)
    self.environment_release = self.get_conf(Config.TITLE_RELEASE, Config.VALUE_ENVIRONMENT)
    self.versionCode_release = self.get_conf(Config.TITLE_RELEASE, Config.VALUE_VERSION_CODE)
    self.host_release = self.get_conf(Config.TITLE_RELEASE, Config.VALUE_HOST)
```

**4）Params->jsonparams.py 读取数据源**

Jsonparams 的作用是从 Postman 导出的 json 数据源中截取所需要的数据，如 header、url、body 等参数，拼接成 requests 请求需要的格式返回。



```
        "name": "导入组织",
        "event": [...],
        "request": {
            "method": "POST",
            "header": [
                {
                    "key": "Content-Type",
                    "name": "Content-Type",
                    "value": "application/x-www-form-urlencoded",
                    "type": "text"
                },
                {
                    "key": "debug",
                    "value": "{{debug}}",
                    "type": "text"
                },
                {
                    "key": "userid",
                    "value": "{{admin_id}}",
                    "type": "text"
                }
            ],
            "body": {
                "mode": "formdata",
                "formdata": [
                    {
                        "key": "type",
                        "value": "1",
                        "type": "text"
                    },
```



```
@staticmethod
def get_case_datas(api_name):
    params = get_parameter('temp')  # 指定读取的json文件名
    case_data = []  # 声明返回数组
    for i in range(0, len(params)):
        if params[i]['request']['description'] == api_name:
            case_dict = {}
            case_dict["name"] = params[i]['request']['description']  # 获取name
            case_dict["url"] = params[i]['request']['url']['raw']   # 获取url
            # 获取header，针对postman格式的解析，所以较为麻烦
            headersDict = {}
            headers = params[i]['request']['header']
            for j in range(0, len(headers)):
                headersDict[headers[j]["key"]] = headers[j]["value"]
            case_dict["header"] = headersDict
            # 获取body，针对postman格式的解析，所以较为麻烦
            if not params[i]['request']['body'] is None:
                bodysDict = {}
                bodys = params[i]['request']['body']['formdata']
                for k in range(0, len(bodys)):
                    bodysDict[bodys[k]["key"]] = bodys[k]["value"]
                case_dict["body"] = bodysDict
            # 放入返回数组
            case_data.append(case_dict)
    return case_data
```

**5）TestCase->test_xxx.py 测试用例**

@allure.story 用于定义被测功能的用户场景，即子功能点

@pytest.mark.parametrize 传多个参数，实现执行不同数据

从数据源获取数据后发起请求，断言结果，设置下一接口所需的变量参数

```python
@allure.story('import_org')
@pytest.mark.filterwarnings("ignore")
@pytest.mark.parametrize("import_org", data.get_case_datas("import_org"))
def test_import_org(self, import_org, action):
    """
    :param import_org: 导入组织
    :param action: 环境参数
    :return:
    """
    request = Request.Request(action)
    api_url = import_org["url"]  # 获取请求地址
    api_header = import_org["header"]    # 获取请求头
    api_body = import_org["body"]    # 获取请求体
    response = request.post_request_multipart(api_url, api_body, api_header, "file", Common.Consts.ADMIN_NAME)

    assert test.assert_code(response['code'], 200)
    assert test.assert_body(response['body'], 'code', 1)
    assert test.assert_body(response['body'], 'message', '操作成功')
    assert test.assert_time(response['time_consuming'], 500)
    Consts.RESULT_LIST.append('True')
```



**6）run.py 执行文件**

获取报告输出位置；定义好需要运行哪些测试集，如果需要全部运行，则不需要定义，会自动找到 Test 开头（结尾）的包下面 test 开头（结尾）的 py 文件；使用 pytest.main(args)执行测试用例，定义好 allure 报告所需参数，使用 shell 命令生成 allure 报告

```python
if __name__ == '__main__':
    # 初始化
    conf = Config.Config()
    log = Log.MyLog()
    log.info('初始化配置文件, path=' + conf.conf_path)
    shell = Shell.Shell()
    # 获取报告输出位置
    xml_report_path = conf.xml_report_path
    html_report_path = conf.html_report_path
    # 定义测试集
    dir = os.path.split(os.path.abspath(__file__))[0]
    test_case_path = dir + '/TestCase/test_init.py'
    args = ['-s', '-q', '--alluredir', xml_report_path]
    # args.append(test_case_path)
    # 运行命令
    pytest.main(args)
    cmd = 'allure generate %s -o %s' % (xml_report_path, html_report_path)

    try:
        shell.invoke(cmd)
    except Exception:
        log.error('执行用例失败，请检查环境配置')
        raise
```

# Jenkins 系列之自动化部署（1）

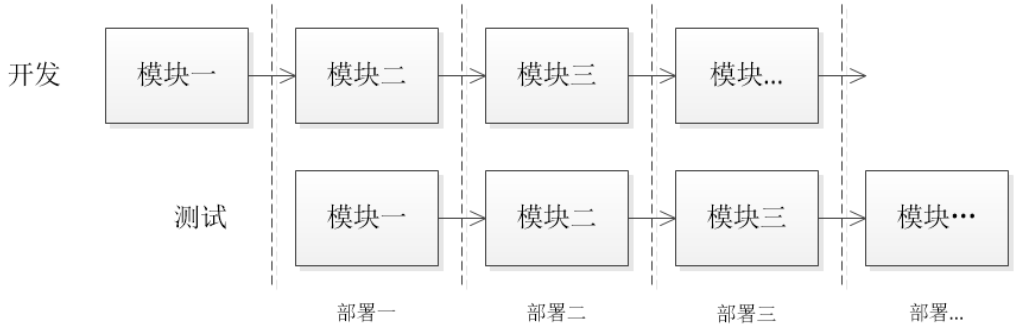
作者：合肥人真帅

在上一篇《如何检验自身的技术水平？参加一场比赛就知道了》文章中，我谈了关于 Selenium 和 Jmeter 的一些总结。这都最近比较主流的框架和工具，一个可以实现 UI 自动化测试，一个可以实现 API 和性能测试，其实还有一种工具可以更大限度的让 Selenium 和 Jmeter 在工作生产中发挥自身价值，那就是持续集成。

## 一、自动化部署的优点

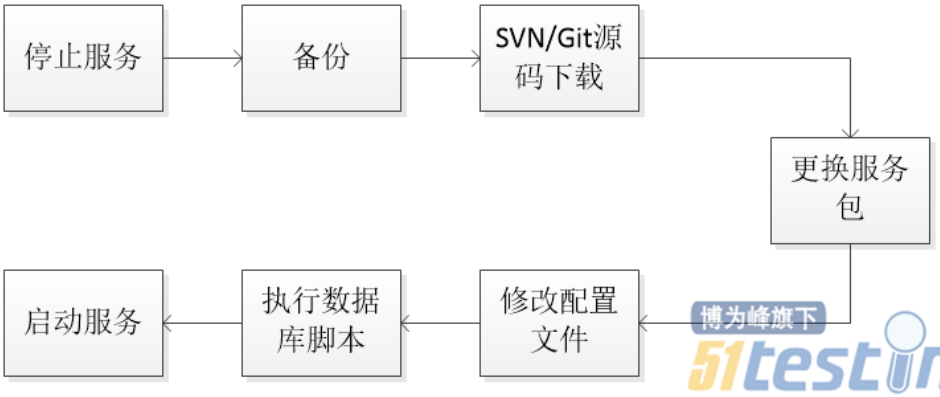
持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。这是百度百科给出的名词解释。如果大家不太理解，我可以给大家举个例子。

比如说，一个产品，定义了很多条需求。如果等到所有需求开发完成，再进行测试，需要耗费大量的时间。如果测试与开发可以并行，就会缩短项目周期。思想如下:




一条业务线运行，先开发，后测试，假设需要 10 天；那么测试开发并行可能 7、8 天就能结束工作；但是我们发现每一次测试前，都需要部署。重复的部署工作，还是耗费了一些时间的，如果可以让电脑自动完成部署工作，是不是 6、7 天就能完成工作？

这个就要具体去分析部署的工作流程了。




这是一个比较简单的部署流程，除了源码下载，其余各阶段，自动化执行都比手工敲打命令速度要快。所以部署的节省时间可以用下面这个公式去计算

节省时间=(单次手工部署耗时*部署次数)-(手工编写脚本耗时+单次自动部署耗时*部署次数)

假设每次手工部署需要 20 分钟，自动化部署需要 8 分钟，部署 10 次，编写脚本 20 分钟

那么节省时间=（20*10）-（20+8*10）=100 分钟

若系统需要分布式部署，节省时间就是成倍增长的。只要节省时间较大，自动化部署方案就可以执行。**这便是自动化部署的第一个有点，节省时间。**

**自动化部署的第二个优点是部署流程严格按照规范执行。**其实很多公司在设计、开发、测试、运维等工作领域都制定了一套适合自己公司的流程规范以及制度。但实际上，这些流程和规范随着时间的推移，逐渐被淡化，甚至有些公司在立项时，整个项目就已陷入混乱而不自知。但是机器则不同，每次都会按照你所制定的方案严格执行。

举个例子，上述部署流程的第二步是备份老系统。很多公司没有专业的运维人员，都是开发或测试兼职，在部署上，安全意识淡薄，可能前几次部署，还会去备份，然后就会觉得麻烦，渐渐抛弃这个步骤，等到部署出现问题，需要回滚时，才发现没有备份。所以最近几年，不乏有一些大厂出现删库跑路的新闻，若真的备份了，即便删了库，还原就行了，又何必跑路呢？但是自动化部署就不同，只要你在里面编写备份的脚

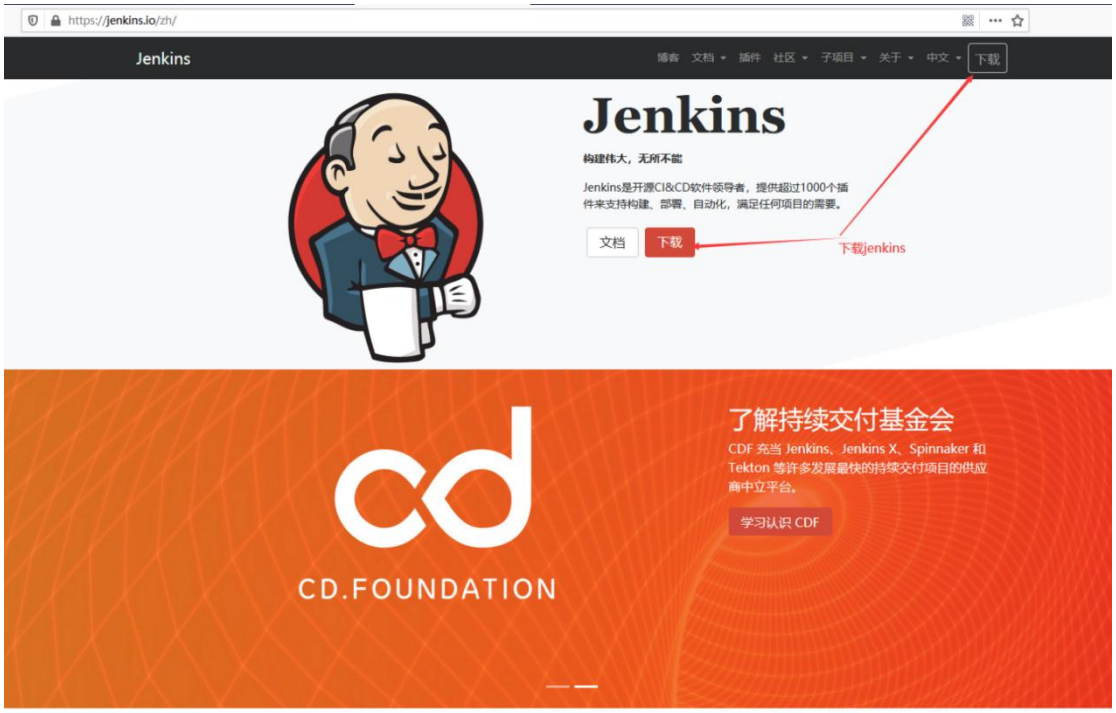
本，部署成功后，就一定可以找到备份文件。

**自动化部署的第三个优点是输入命令的准确率 100%。**这个也很好理解，人总有疲惫、心烦等各种情绪和状态，即便是满血状态，也可能会在输入命令的时候，打错个别单词，若是在上述部署流程中的“修改配置文件”环节打错字，系统不会报错，待服务启动后，发现各种问题，才回去检查之前的步骤，此时已经耽误相当多的时间了。对自动化部署而言，这条优点就完美体现出来了。

当然优点不止这 3 条，缺点也是有的，我就不一一列举了。持续集成的工具有很多，如 TeamCity、Jenkins、Buildbot 等。目前 Jenkins 的使用人数占比相对较高，下面我就从安装到工程的构建介绍一下 Jenkins 的使用。

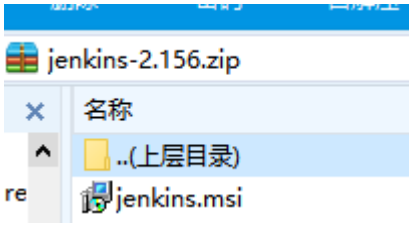
## 二、Jenkins 安装

Jenkins 的官网可以下载到 jenkins 的安装包，不同的平台，选择不同的入口

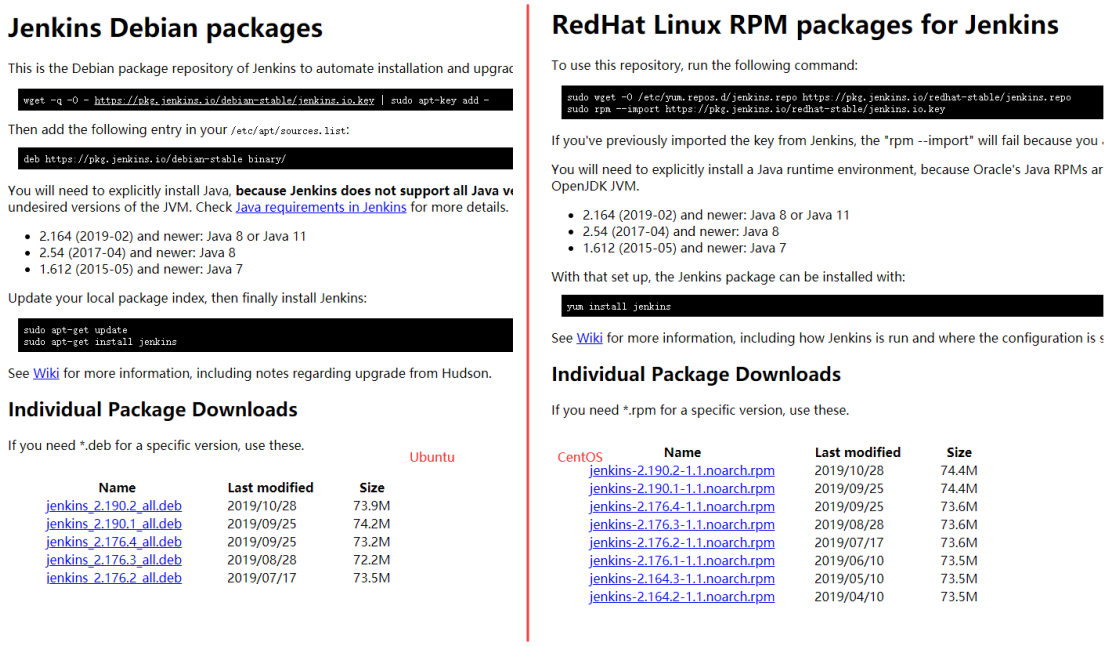




例如 Windows，Linux 的各个版本等。Windows 直接下载压缩包，解压后运行 msi，一直点击下一步，即可完成安装。




Linux 的安装，可以下载到本地安装，也可以在线安装，不同的 Linux 安装命令也不同，例如 Ubuntu 的命令是 apt，CentOS 的命令是 yum。



## Jenkins Debian packages

This is the Debian package repository of Jenkins to automate installation and upgrac

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

Then add the following entry in your /etc/apt/sources.list:

```
deb https://pkg.jenkins.io/debian-stable binary/
```

You will need to explicitly install Java, **because Jenkins does not support all Java v** undesired versions of the JVM. Check Java requirements in Jenkins for more details.

- 2.164 (2019-02) and newer: Java 8 or Java 11
- 2.54 (2017-04) and newer: Java 8
- 1.612 (2015-05) and newer: Java 7

Update your local package index, then finally install Jenkins:

```
sudo apt-get update
sudo apt-get install jenkins
```

See Wiki for more information, including notes regarding upgrade from Hudson.

### Individual Package Downloads

If you need *.deb for a specific version, use these.

Ubuntu

| Name | Last modified | Size |
|---|---|---|
| jenkins_2.190.2_all.deb | 2019/10/28 | 73.9M |
| jenkins_2.190.1_all.deb | 2019/09/25 | 74.2M |
| jenkins_2.176.4_all.deb | 2019/09/25 | 73.2M |
| jenkins_2.176.3_all.deb | 2019/08/28 | 72.2M |
| jenkins_2.176.2_all.deb | 2019/07/17 | 73.5M |

## RedHat Linux RPM packages for Jenkins

To use this repository, run the following command:

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
```

If you've previously imported the key from Jenkins, the "rpm --import" will fail because you

You will need to explicitly install a Java runtime environment, because Oracle's Java RPMs ar OpenJDK JVM.

- 2.164 (2019-02) and newer: Java 8 or Java 11
- 2.54 (2017-04) and newer: Java 8
- 1.612 (2015-05) and newer: Java 7

With that set up, the Jenkins package can be installed with:

```
yum install jenkins
```

See Wiki for more information, including how Jenkins is run and where the configuration is

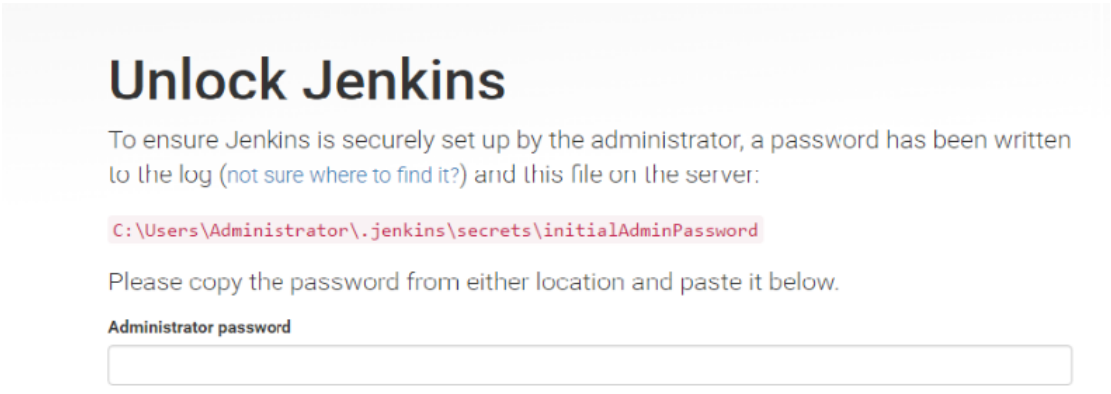
### Individual Package Downloads

If you need *.rpm for a specific version, use these.

CentOS

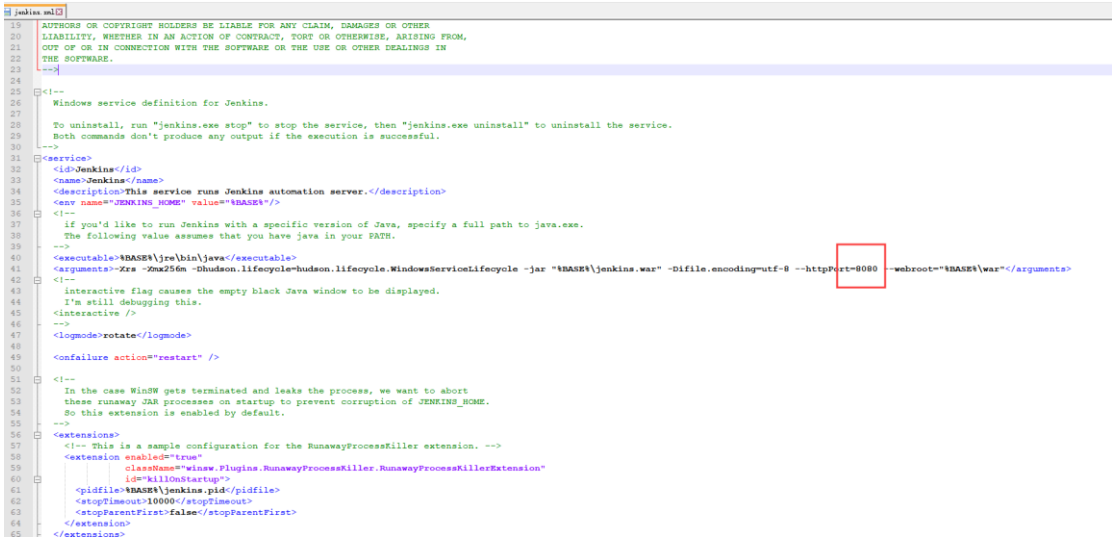
| Name | Last modified | Size |
|---|---|---|
| jenkins-2.190.2-1.1.noarch.rpm | 2019/10/28 | 74.4M |
| jenkins-2.190.1-1.1.noarch.rpm | 2019/09/25 | 74.4M |
| jenkins-2.176.4-1.1.noarch.rpm | 2019/09/25 | 73.6M |
| jenkins-2.176.3-1.1.noarch.rpm | 2019/08/28 | 73.6M |
| jenkins-2.176.2-1.1.noarch.rpm | 2019/07/17 | 73.6M |
| jenkins-2.176.1-1.1.noarch.rpm | 2019/06/10 | 73.5M |
| jenkins-2.164.3-1.1.noarch.rpm | 2019/05/10 | 73.5M |
| jenkins-2.164.2-1.1.noarch.rpm | 2019/04/10 | 73.5M |

安装完成后，打开浏览器，输入 localhost:8080




第一次安装，需要解锁，解锁密码就在红色路径的文件里。解锁后，根据提示安装插件，创建管理员账号，安装就算完成了。如果在安装后发现 8080 端口被占用，导致 jenkins 无法运行，应该先停止 jenkins 服务，然后找到配置文件修改端口号。Windows 的配置文件，就在 jenkins 安装路径 jenkins.xml 文件中修改。而 Linux 则是在

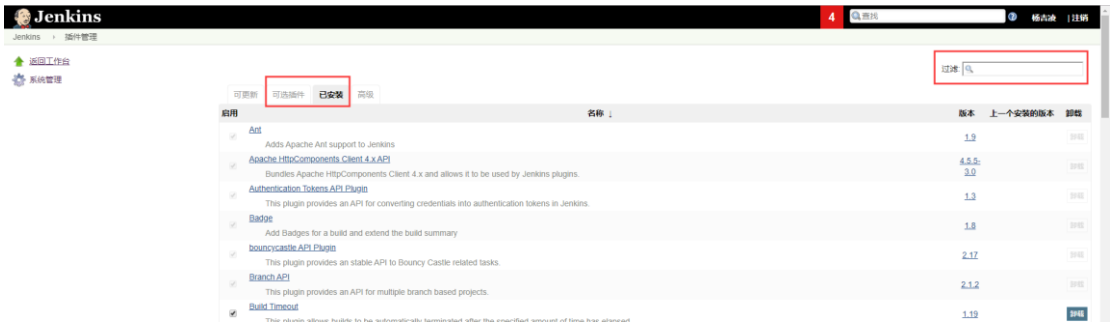
/etc/sysconfig/jenkins 文件中修改，不同版本的 linux 或许配置文件位置不同，但都在/etc 下，仔细找找是可以找到的，也可以用 whereis 来查找。



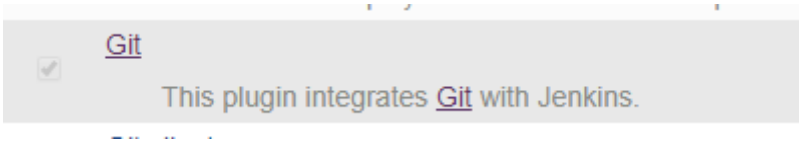

## 三、添加插件、进行各项配置

Jenkins 插件有很多，由于各个公司对源码管理，项目管理等使用的工具都各不相同，这里就不一一介绍了。这里介绍 1 个大家都用的上的插件，Email Extension Plugin，这个插件可以将每一次的构建结果以邮件的方式发送到指定邮箱。（还有 1 个 HTML Publisher 插件，我会在下一篇给大家讲解，如果感兴趣，可以自己摸索一下）

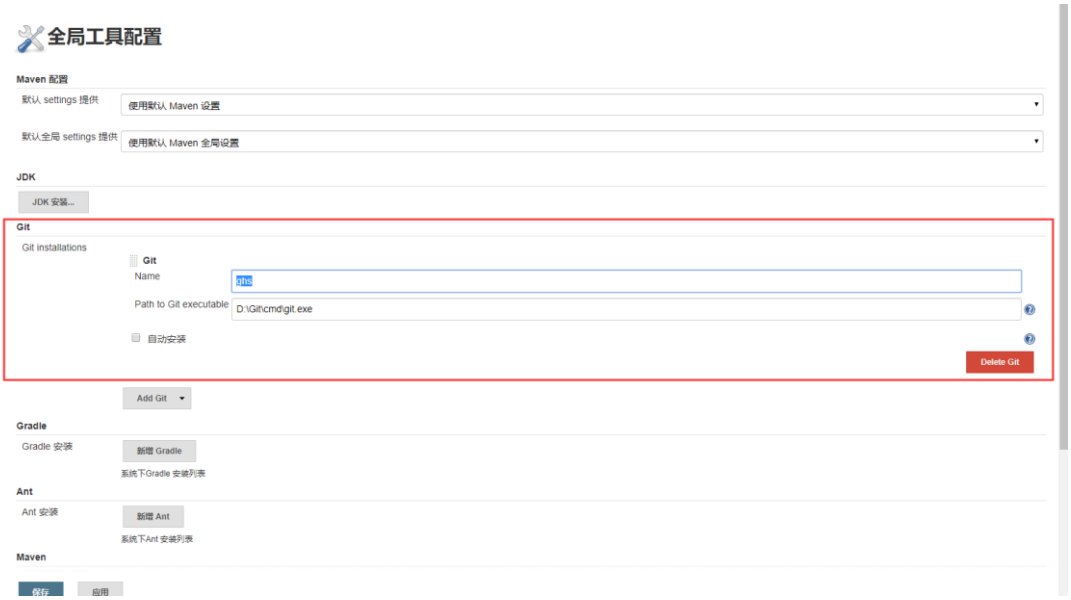

点击系统管理→插件管理→可选插件，进行搜索，选中后点击安装。安装成功后，刷新页面，就可以在已安装页面中找到。




我这里用的是 git 管理源码，如果是 svn，操作方法是相同的。安装 git 插件，搜索 git，如下图

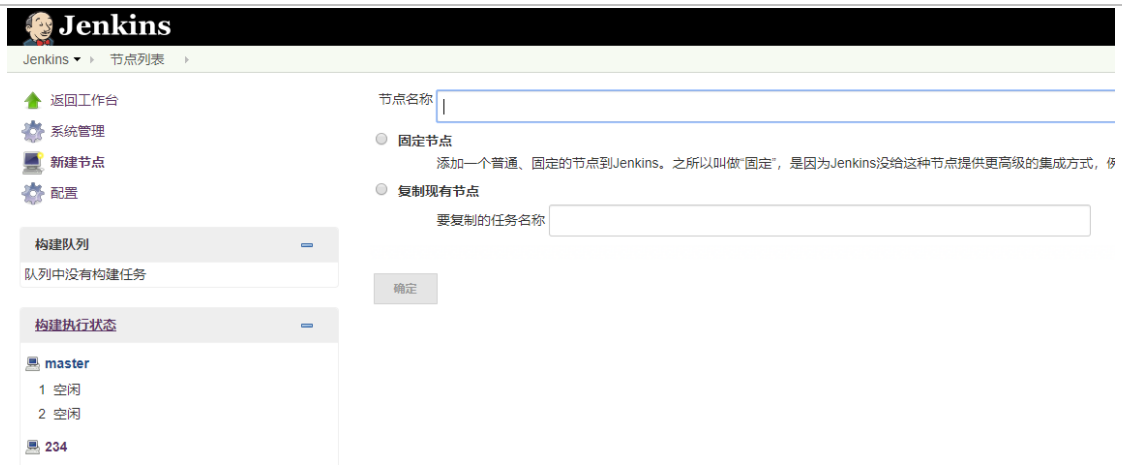



安装后，在系统管理→全局工具配置页面下，配置本机 git 环境




如果 jenkins 服务器和部署的目标服务器不是同一台机器，那么就需要配置一个远程节点。类似于 Jmeter 的分布式压测。

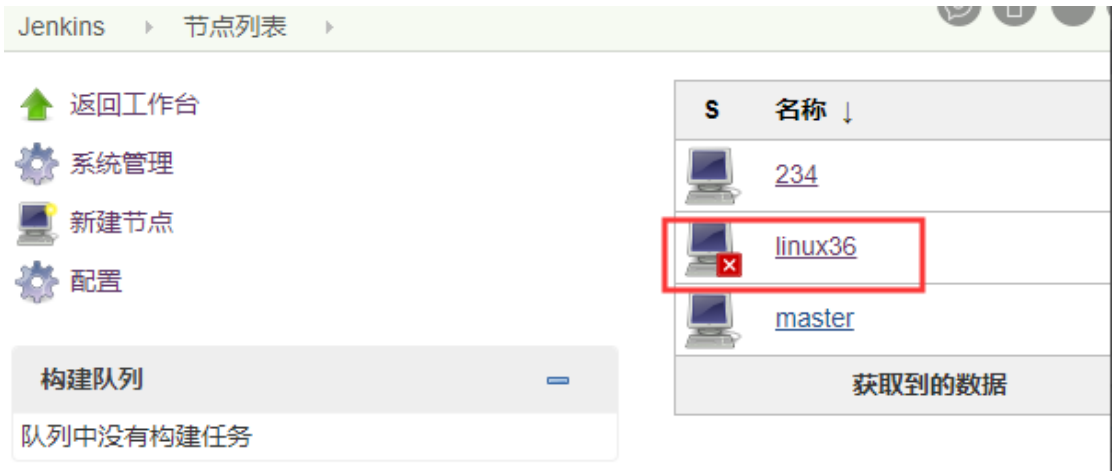
点击系统管理→节点管理→新建节点，如下图




输入节点名称，选择固定节点，点击确定


到这个页面后，linux 和 windows 的配置方法是不一样的，我先介绍一下 linux 的配置方法。描述字段为了后期方便管理，最好写上和服务器相关的信息，例如 IP，项目名称等。远程工作目录是 jenkins 调用该节点时，工作文件的保存路径。


创建好以后，返回节点列表，会显示一个红色的叉，代表没有连接成功。稍等 1 分钟，刷新页面，若红色叉消失，代表已经连接成功。如下图



若始终显示未连接，则需要检查目标服务器的 IP，帐号密码是否正确，端口号是否被防火墙拦截等。

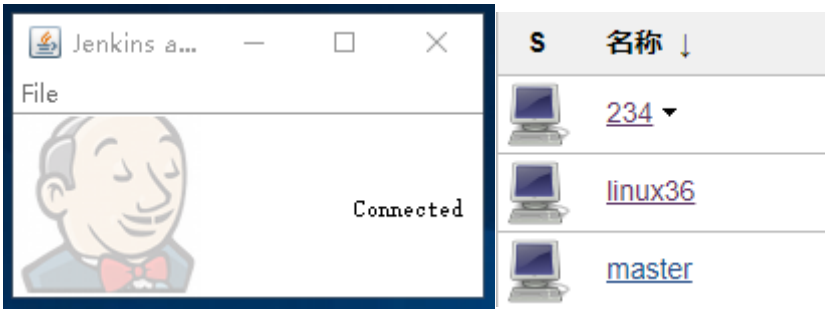
Windows 配置远程节点，如下图


配置完成后，返回节点列表


目前状态是未连接，他不会像 linux 那样自动连接，需要我们手动建立连接。点击

名称，跳转到如下页面


这里提供了两种建立连接的方式，第一种比较简单，点击 Launch 按钮，会下载一个 slave-agent.jnlp 文件，将这个文件发送到远程机上，双击运行，如下图




节点的红色叉消失，连接成功。

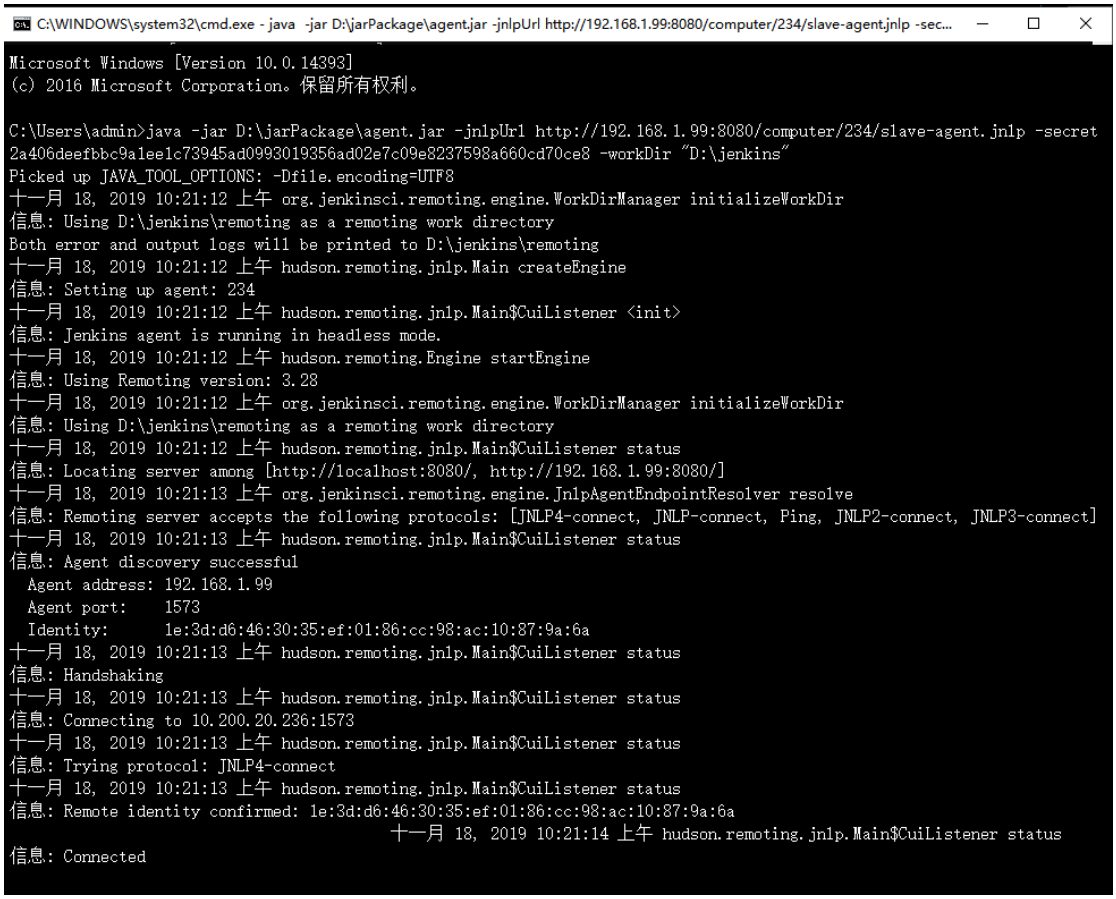
第二种方法是点击 agent.jar，下载文件。复制页面的命令

java -jar agent.jar -jnlpUrl http://localhost:8080/computer/234/slave-agent.jnlp -secret 2a406deefbbc9a1ee1c73945ad0993019356ad02e7c09e8237598a660cd70ce8 -workDir "D:\jenkins"

在 agent.jar 前面加上本地路径，例如 D:/download/agent.jar

localhost 要改成 jenkins 服务器的 ip

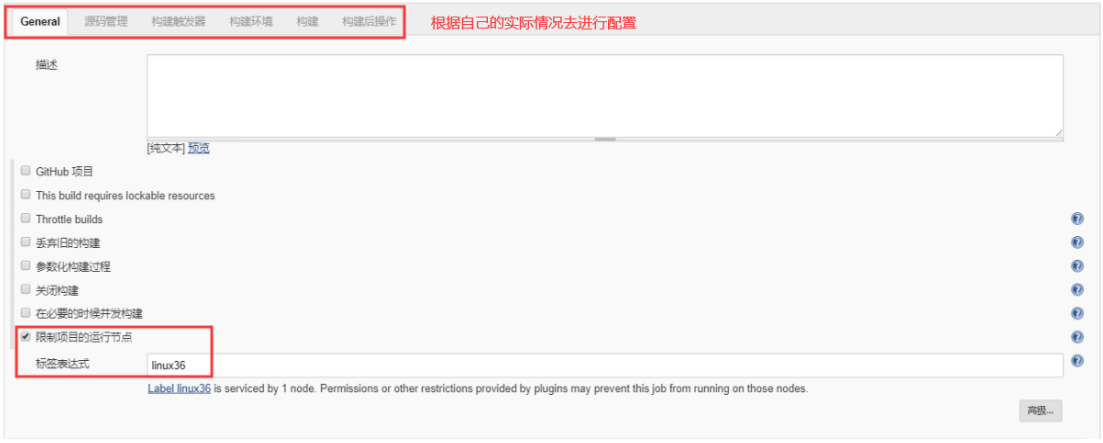
运行结果如下，出现 Connected 表示连接成功




## 四、创建任务，构建自动化部署

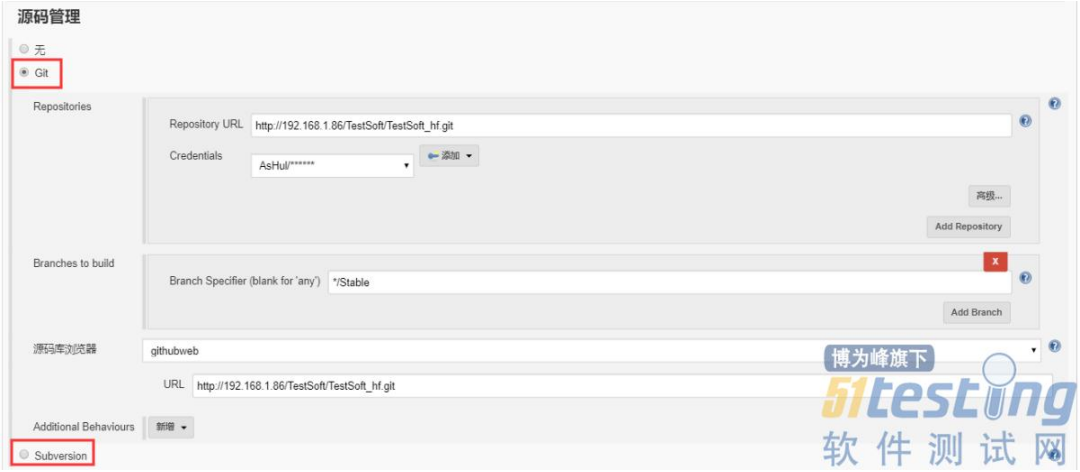
在 jenkins 首页，点击新建任务。输入任务名称，选择“构建一个自由风格的软件项目”，点击确定。


进入任务的配置页面，顶部有很多 tab 页，根据自己的实际情况去添加配置即可，我这里演示一个最简单的配置方法。如下图



若 jenkins 服务器和部署的目标服务器不是同一台，可以选择部署的目标服务器作为运行节点。例如我们要部署到 linux36 这台机器上，就可以选择我们刚才添加的 linux36 这个节点。

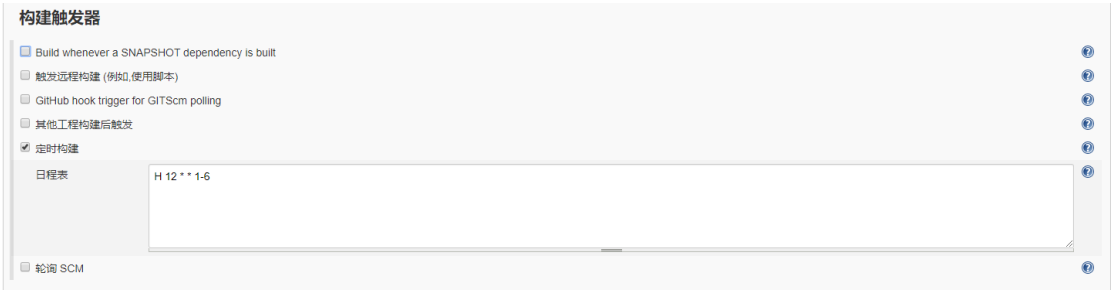
我使用的是 git 管理源码，在 URL 输入 git 的地址，Credentials 输入帐号密码，BreanchSpecifier 输入需要获取的代码的分支。



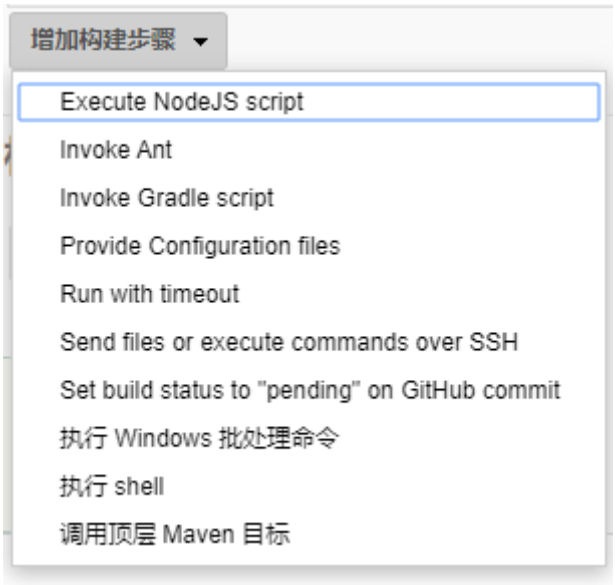

若是 SVN 管理源码，则选择 SVN，配置方法类似。

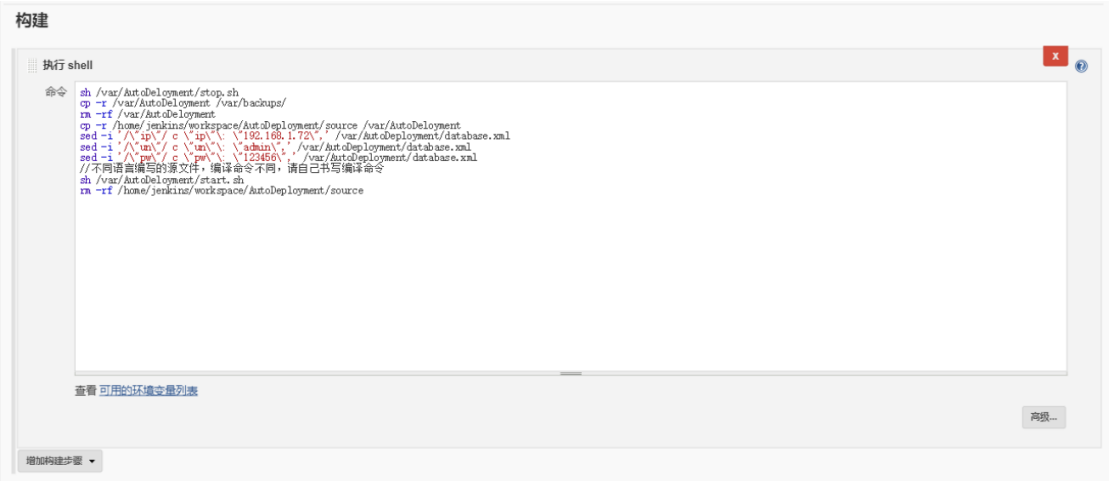
当前部署的项目，是独立项目的，可以选择定时构建。语句分为 5 段，每段用空格隔开，从左到右分别表示分钟、小时、日、月、星期，图中的语法就是每周 1 到周 6，中午 12 点整执行一次。

若是有前置条件的项目，可以选择其他触发方式。

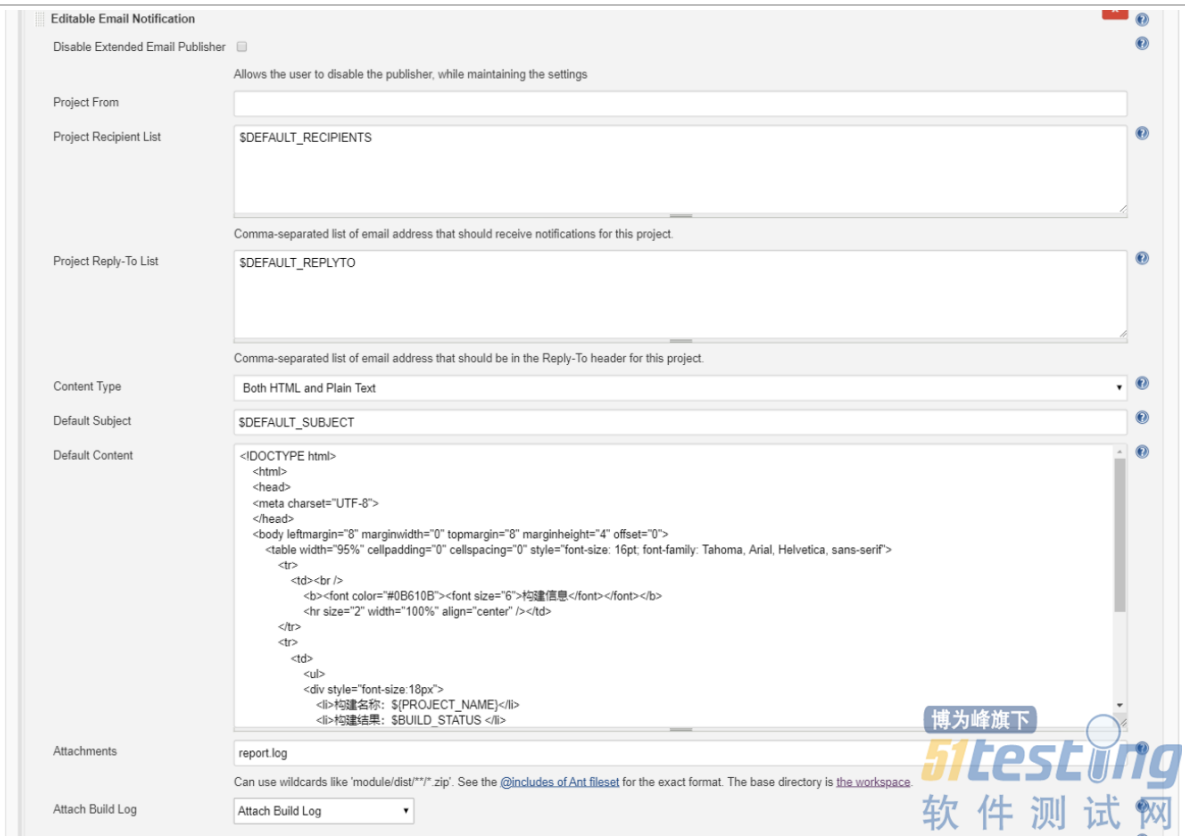



接下来是构建具体的执行命令，执行构建的节点服务器是 windows 的可以选择批处理，是 linux 的可以选择 shell，可以选择框架或工具类的命令例如 maven，ant 或各种脚本语言 nodejs，invoke Gradle ，若图中没有你想要的构建方法，可以到插件管理页面，下载想要的插件，安装后，就可以在这里看到选项了。

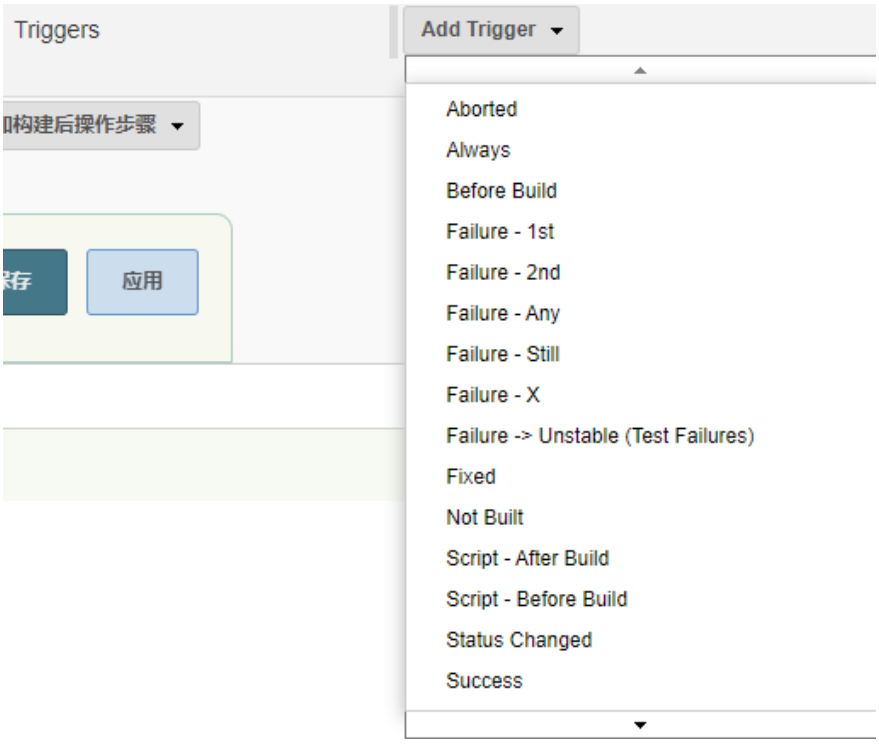



我这里用的是 shell，第一行是停止当前服务，第二行是备份当前服务，第三行是删除当前服务（相当于清理了环境），第四行是将最新的源码文件拷贝到原来的服务文件夹，第五至七行是在配置数据库服务文件，第八行由于每个公司的源码是用不同的语言编写的，有 java，C++，C#，python 等等，他们的编译命令不同，就不举例了。第九行是启动服务，第十行删除下载的源码（清理 workspace），这样一个简单的部署就完成了。部署完成后，可以让 jenkins 发送邮件，这样即使你不在电脑前，手机也可以收到邮件，来查看最新的部署结果。

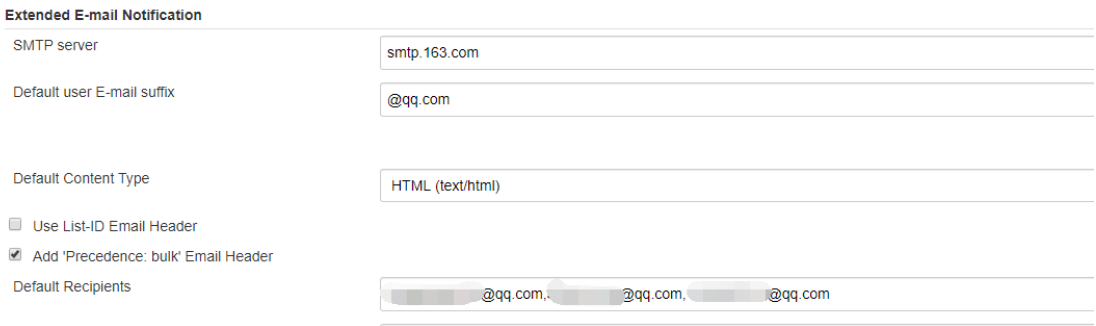



这个邮件服务，就用到了我们之前在插件环节介绍的那款 Email Extension Plugin。大部分字段使用默认值即可，Default Content 可以使用系统管理→系统设置→Extended E-mail Notification 模块设置的通用模版，也可以像上图这样给这个任务设置一个独立的邮件内容。Attachments 可以选择本次部署生成的日志文件，选择 Attach Build Log，在发送邮件时，讲日志文件以附件方式一起发送。



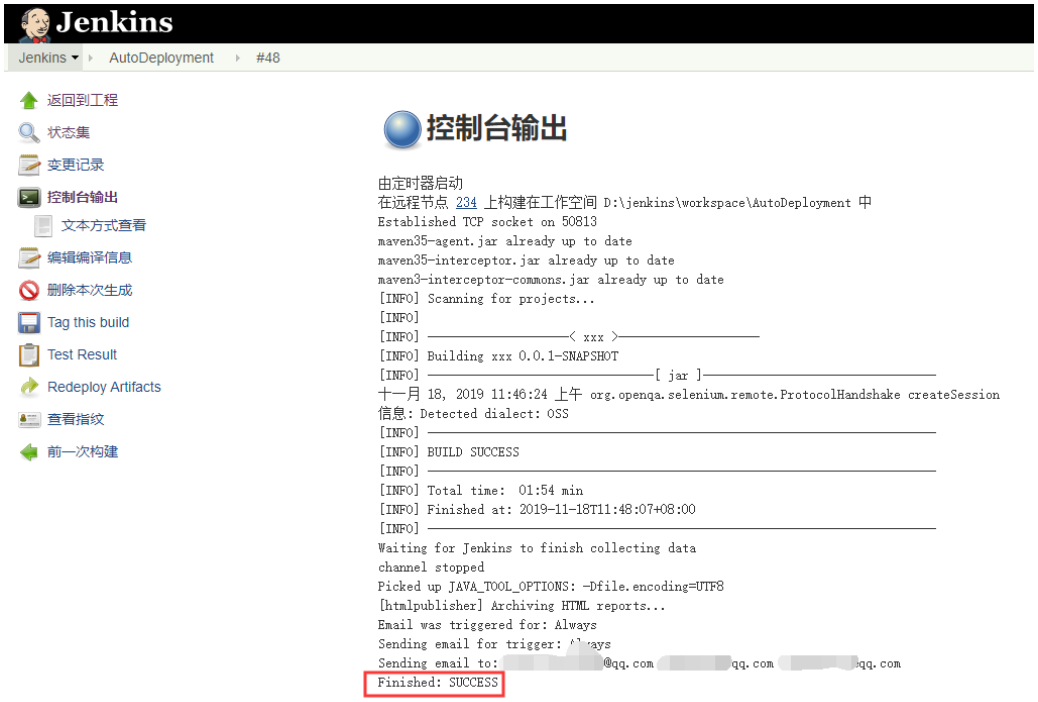

选择发送方式，Always 表示只要构建结束就发送邮件，Failure 表示构建结果是失败时，才发送邮件，Success 表示构建结果为成功时发送邮件，其余选项顾名思义。


SendTo 是发送给哪些人，可以多选。Developers 是开发者，RecipientList 是收件人列表。可以在系统管理→系统设置→Extended E-mail Notification 模块→Default Recipients 字段设置，多个人用英文逗号隔开



任务的所有配置结束，点击保存。回到首页


点击任务名称，每次构建的，都会在左边生成一个历史构建，并以数字递增，点击超链接，查看控制台输出




这里的提示，是构建成功的意思。可以手动去验证，本次自动化部署是否正确。若存在问题，可以查看构建日志，要对自己的配置及各环节的命令进行错误排查。这样一套相对简单的自动化部署就完成了。解放部署时间，将节省的时间安排其他工作，提升工作效率，从此时此刻开始。

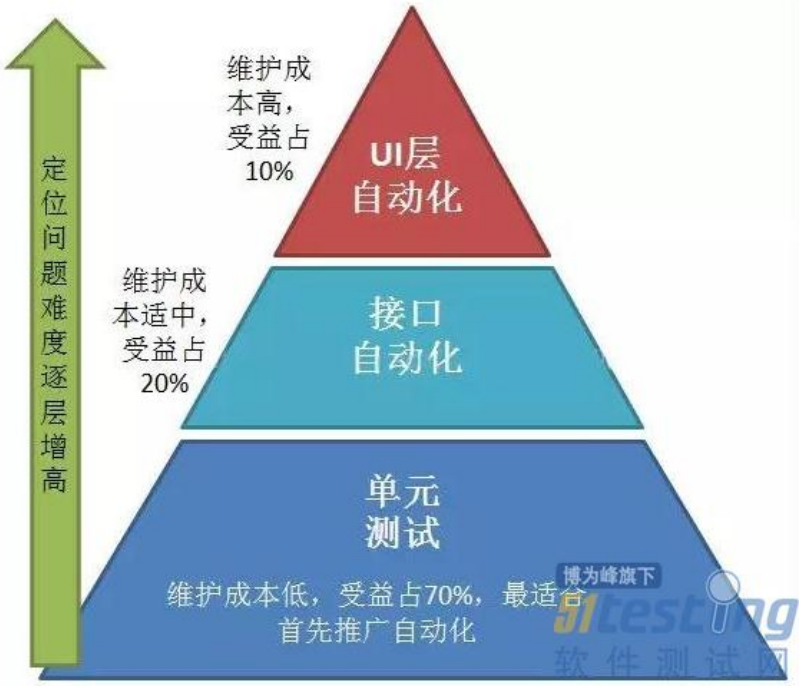
# Jenkins 系列之 Selenium-UI 自动化测试（2）

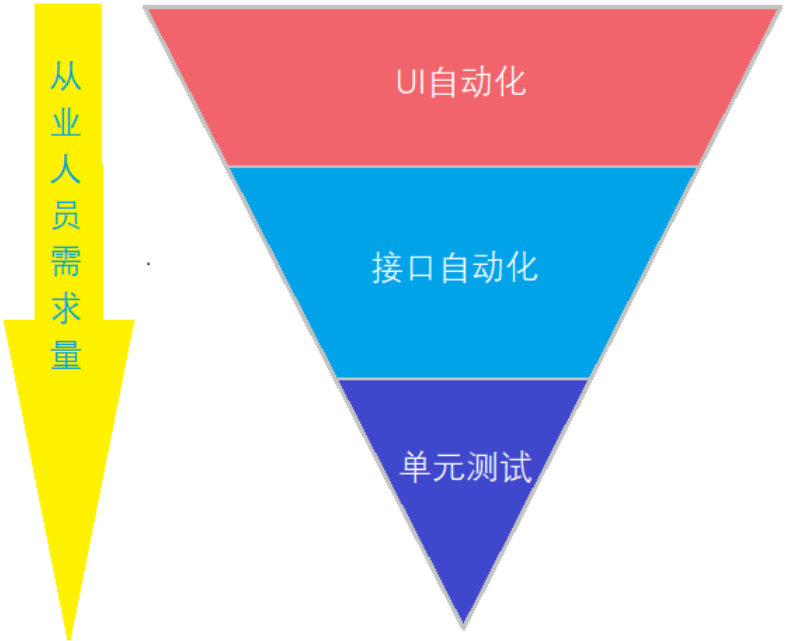
◆ 作者：合肥人真帅

持续集成的定义中包括自动化编译、部署、测试，上一篇我们讨论了自动化部署的优点，以及如何实现自动化部署。这一次我们来谈一谈 UI 自动化测试。

关于自动化，前几天我看到一篇文章，文中有句话：“不懂开发的手工测试已经成为新时代的文盲，被优化的对象”非常辣眼睛。这是一家培训机构，在招生前写的一些铺垫，目的是为了引发读者的议论，炒作，吸引流量。虽然在言语上有些激进，但也不是全无道理。自动化是一种趋势，任何时候都要顺势而为，不要逆势而动。雷军曾说过“站在台风口，猪也能飞起来”，这里的猪，我认为指的是实力不强的人，并不是什么都不懂的人。所以可以翻译成“顺着大趋势走，实力不强的人也能风生水起”，但前提是你要先成为一个实力不强的人，而不是一点实力都没有。

那么说到自动化测试，相信大家都看过自动化测试金字塔。从图形结构上看，自动化测试在 UI 层面上受益最少，但为什么在公司招聘的岗位中，UI 自动化需求量最大，接口次之，单元测试几乎没有，和自动化收益成效相比，呈倒三角。




## 一、自动化测试倒三角分析

产生上述倒三角的现象不难理解，可以以下几个方面分析:

### 1、维护成本高

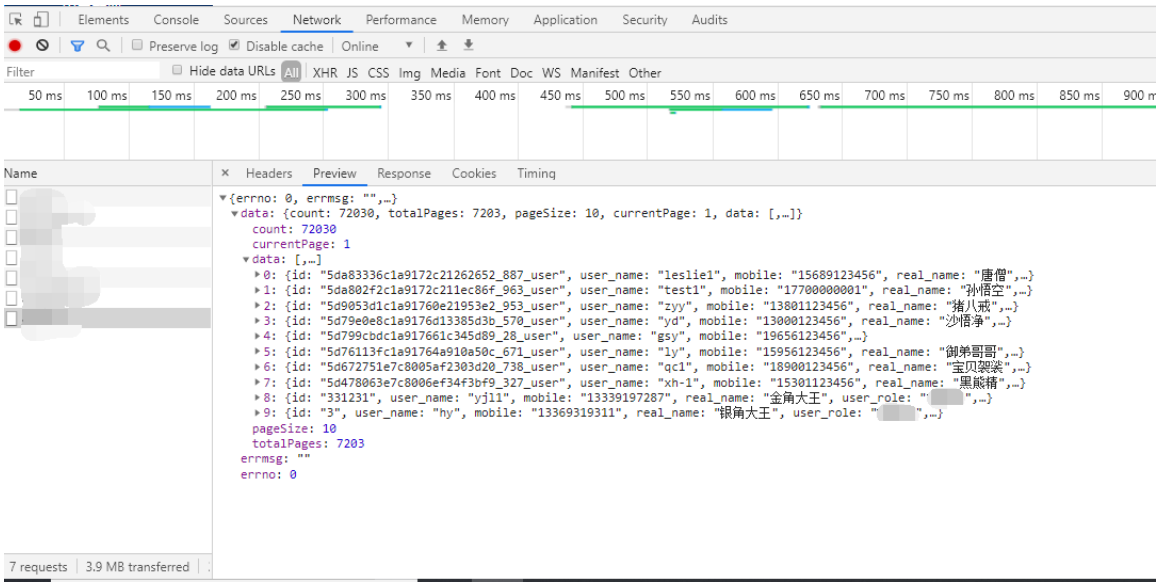
在自动化测试收益的金字塔中，UI 层自动化测试维护成本高。这个维护成本通常情

况下指的是人员费用和时间。为了缩短时间，就需要大量的人员，聘用的人员多，就会导致企业支出的总费用增加。同时需求在不断变化，前端元素改动，或多或少都会对自动化脚本造成影响，需要不断维护。所以在自动化测试中，UI 自动化测试就出现了维护成本高收益少，而且还需要大量人员。

**2、收益只占 10%，为什么还有那么多公司去做 UI 自动化**

UI 自动化测试收益只占 10%，为什么还有这么多公司去做 UI 自动化？因为他们是站在用户视角去考虑问题的。无论什么软件，最终都需要用户来使用。用户不会去关心软件内部是如何实现的，他们只关心是否操作简单，且功能可以满足需求。自动化测试中，UI 测试是模拟键盘和鼠标的操作，最接近用户的操作模式。所以测试结果最接近用户的感官。

举个例子，假设数据库存储的数据是正确的，如果不做 UI 测试，只做单元或接口测试，会产生什么问题？下面是接口返回的结果，与数据库进行对比，结果正确。




下面是 UI 展示的效果。请注意，上一步我们验证了接口返回的数据是正确的，而 UI 层面，第三列要求显示手机号码，第四列要求显示用户姓名，明显弄反了。这是因为后端开发人员在编写接口时，功能正确，但是前段开发人员，在拿到接口数据后，没有把正确的数据，放到正确的位置上，导致了这个 bug 的产生。所以即便 UI 层自动化测试的收益只有 10%，但是他是站在用户的视角去验证软件功能。所以即便收益只有 10%，但这 10%是无法替代的，就会有公司坚持去做 UI 自动化。

| 序号 | 用户名 | 手机号码 | 用户姓名 |
|---|---|---|---|
| 1 | leslie1 | 唐僧 | 15689123456 |
| 2 | test1 | 孙悟空 | 17700000001 |
| 3 | zyy | 猪八戒 | 13801123456 |
| 4 | yd | 沙悟净 | 13000123456 |
| 5 | gsy | 你是猴子请来的救兵吗 | 19656123456 |
| 6 | ly | 御弟哥哥 | 15956123456 |
| 7 | qc1 | 宝贝袈裟 | 18900123456 |
| 8 | xh-1 | 黑熊精 | 15301123456 |
| 9 | yjl1 | 金角大王 | 13339197287 |
| 10 | hy | 银角大王 | 13369819311 |



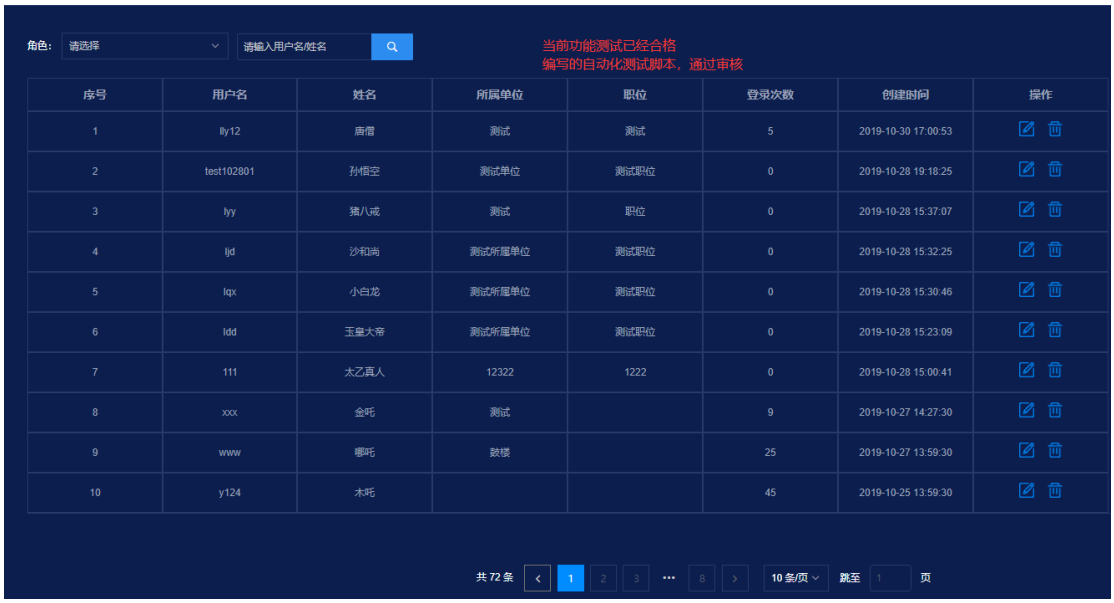
### 3、从软件复杂性出发

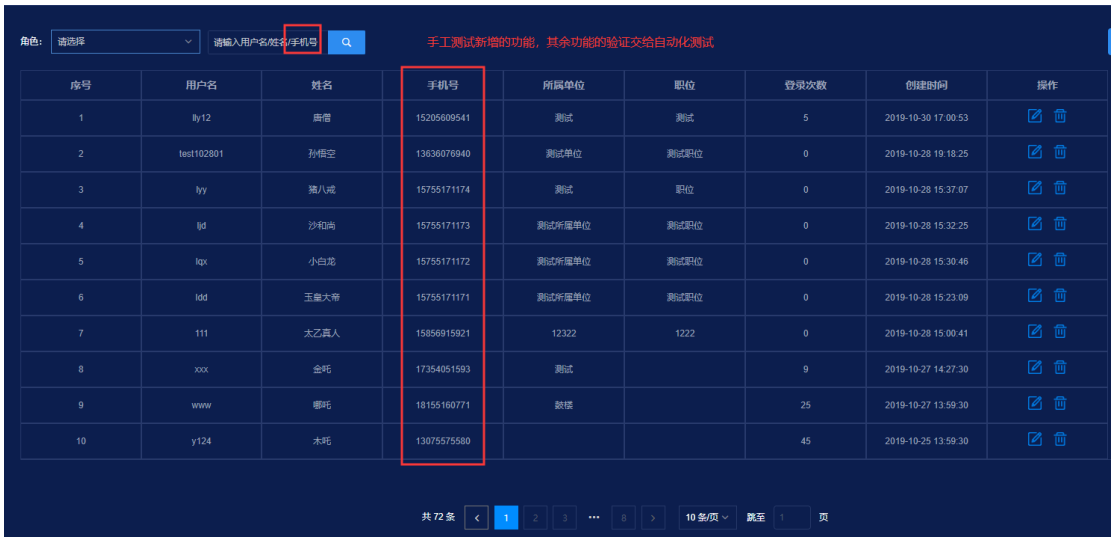
众所周知，软件开发是一件复杂的事情。同一个功能，100 个程序员去做，虽然最终的输出结果相同，但软件内部的实现方式会有 100 种。每个开发人员都非常熟悉自己编写的代码，自测效率会更高，所以大部分公司都会让开发人员完成自测，而不是招聘专业的白盒测试人员。但是考虑到自测的局限性，只有对代码要求非常高的公司，才会招聘专业的白盒才是人员，去进行单元测试。所以对单元自动化测试人员的需求量，自然少的多。毕竟看别人的代码是一件很恶心的事情，而且一直看，一直恶心。

## 二、UI 自动化的优劣势和运用

### 1、自动化测试的优点：

**①版本迭代时，节省大量回归测试时间。**例如一个产品已经测试合格，上线。在客户使用的过程中，不断有需求反馈给产品部。要对其进行不断的细节优化和新功能的开发。但这些功能都相对独立，在优化细节过程中也不会影响基础产品的功能。此时对基础功能模块进行自动化测试脚本的开发，以后版本迭代时，只需要测试新模块，老功能交给自动化测试，可以节省大量时间。例如下图，第一张图的功能已经测试合格，第二张图新增了一个手机号查询功能，那么可以手工测试手机号查询功能，其余老功能交给自动化测试




**②更好地利用资源。**这个也很好理解，我从两个方面来分析一下。第一，大部分公

司会给每位测试人员配备一台电脑（有些公司会配备 2 台或 3 台），那么为了缩短交付周期，可以在白天的时候进行新功能的手工测试，在下班以后，运行自动化测试，第二天来查看测试结果即可，并且在自动化测试过程中，如果出现故障导致无法继续执行，可以让其发送邮件，当测试人员收到邮件后，在家时可以进行远程登录（登录公司的机器），排查错误原因，让自动化测试继续执行下去；第二，在评估工作计划的时候，或多或少都会因为各种原因，导致评估不太准确，一般在时间的评估上，误差在±10%以内就算是精准了。有些公司缺乏专业的项目经理，而是让开发经理兼职这个岗位的工作任务，导致误差在±30%，也有很多。时间上出现混乱，必然有部分真空期的存在。比如测试人员原定 30 天写完测试用例，评审，修改，然后执行。但是由于功能开发还没完成，需要推迟 5 天。这 5 天测试人员可以优化自动化测试脚本，充分利用这 5 天，不让时间浪费。若没有自动化测试的公司，可能会安排员工学习新的测试技术，但笔者从实际工作经验来看，基本上是没有效果的。

**③测试具有一致性。**例如下图，是一个新增用户的对话框，若测试用例中有一个操作步骤写成“点击关闭按钮”，但是这个页面有 2 个关闭按钮，执行这个测试用例时，有些人会点击下面的关闭，有些人会点击上面的关闭，可能会导致相同的测试用例，不同的测试结果。

若是自动化测试，在编写脚本的时候，选择了下面的关闭按钮，那么以后每次执行时，都会点击下面的关闭按钮。那么可以保证每次测试步骤都是相同的。至于右上角的关闭按钮，可以在优化测试用例时，再增加一条用例，去测试这个按钮。

**④增加软件测试可信度。**例如一个产品有 500 个功能点，且已经测试合格。最新的一个版本，只新增了一个功能点。站在公司的角度去看，老板不会给你太多时间去测试，你的工作重心只能放在新功能上。当测试结束后，“测的怎么样，合格了吗？”，这个问题是不是很难回答？因为你只测了新功能，根本没时间测试老功能，或者老功能只是走马观花的过了一遍。你不确定新功能是否对老功能产生了多大影响。你不敢回答这个问题。所以如果老功能有自动化测试过了一遍，达到高百分比覆盖（不会达到 100%的），你一定会很自信的回答这个问题，增加测试结果的可信度。即便有 1，2 个遗漏的，也是情理之中的，因为软件 bug 是不可能被全部找到的，只要控制在可接受范围内就行了。

**2、自动化测试的缺点：**

**①不能取代手工测试。**例如界面的美观，友好性就无法进行自动化测试。下图中可以看出，功能并无错误，但是页面中间有一大块空白，布局较丑，这类问题，自动化测试就无法实现。


**②手工测试比自动测试发现的缺陷更多，自动化测试不容易发现新的 BUG。**自动化测试是测试已经合格的功能，是为了防止新功能对老功能产生影响而未被发现。所以已经合格的功能即便被新功能影响到了，bug 也不会太多，所以自动化测试发现新 bug 会比手工测试要少。自动化测试的主要目的，还是防止遗漏。

**③对测试质量的依赖性极大。**若老功能遗留的 bug 较多，且漏测较多，测试质量不高，自动化测试是执行不下去的。

**④自动化测试无法提高测试的有效性。**

**3、不适合自动化测试用例的情况：**

**①定制化项目。**所有功能都只为该项目的客户定制化服务，这个项目的功能，不能在其他项目上复用。例如，你去商场买衣服，你的身高 1 米 8，体重也很匀称，你可以找到很多大小合适的衣服，相同款型，相同大小的衣服有很多。而姚明，身高 2 米 27，他在市场就就买不到衣服，必须定制。为他定做的款型，只能卖给他，即便多生产一件相同的衣服出来，市场上也不会有人买。为他定制的设计稿，也无法复用。这类项目，就不适合自动化测试。

**②项目周期很短的项目。**项目周期很短，测试周期更短，就不值得花精力去投资自动化测试，好不容易建立起的测试脚本，不能得到重复的利用是不现实的。

**③业务规则复杂的对象。**业务复杂，会导致测试用例步骤较多。若写成自动化测试脚本，实现难度很大，有一种捡了芝麻丢了西瓜的感觉。

**④美观、声音、易用性，人的感观方面的测试。**例如音乐软件在测试音频播放时，整首音乐中出现过杂音，或者视频软件在播放视频时出现了不明显的卡顿，画质清晰度等，也无法使用自动化测试。

**⑤测试很少运行。**例如一个产品虽然功能测试合格，但未来几年没有迭代的打算。若对这样的产品进行自动化脚本开发。实际上也是在浪费成本，因为软件更新换代那么快。几年都不打算迭代的产品，可能有被淘汰的风险。若开发完的自动化测试脚本，没有长期运行，那将毫无意义。

**⑥软件不稳定。**会由于这些不稳定因素导致自动化测试失败。自动化测试的目的就是不断运行开发出来的测试脚本，若始终因为各种因素无法运行，开发自动化测试脚本所用的时间也就等于是浪费掉了。

**⑦涉及物理交互。**例如在宾馆、饭店、网吧都很常见的充电宝，后台是靠软件服务支撑的，而前端是靠硬件里，嵌入式开发的程序来运行的。这样前端操作影响后端数据，就无法进行自动化测试了。

**4、适合自动化测试的情况:**

**①产品或产品型项目。**每次迭代只改进少量的功能，没有改动过的功能需要反复测试。这部分测试完全可以让自动化测试来承担。新增的功能在测试合格后，也可以脚本化，加入自动化测试。

**②增量式开发、持续集成项目。**例如小米产品的服务功能，一开始手机后台服务程

序，只为手机服务；电视后台服务的程序只为电视服务；然后电视后端服务和手机后端都测试合格，产品有了一定的成熟性。手机和电视有了多屏互动，手机红外遥控等功能，多个产品交互的功能开发，是否会影响单个产品的稳定功能？我们可以把电视、电脑、手表、电话等等单个产品，合在一起看成一个增量式的，持续集成的智能家居项目，那么单个产品在测试合格后，很适合自动化测试，我们只需要手动测试各个产品之间的交互。

**③能够自动编译、自动发布的系统。**例如我下面要讲的 jenkins

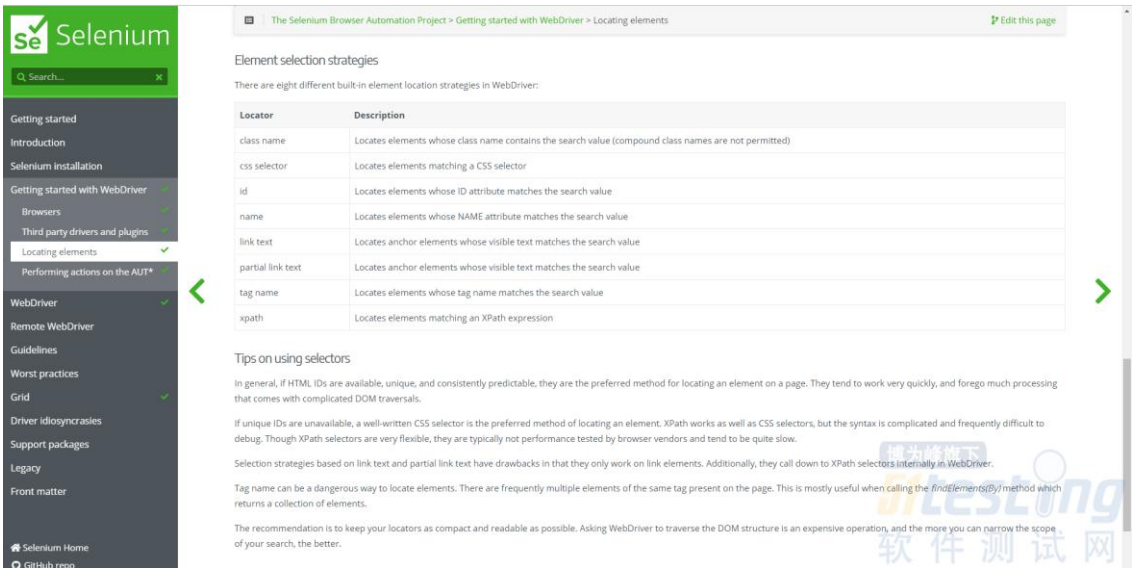
**④回归测试。**回归测试是自动化测试的强项，它能够很好的验证你是否引入了新的缺陷，老的缺陷是否修改过来了。

**⑤需要频繁进行相同的测试步骤。**对于人来说，总是执行相同的动作，就会产生厌烦，乏累的感觉。例如，从 1 写到 600，听上去很简单，但是没有几个人可以完全写对，这个骗局，利用的就是人无法在长期进行重复性动作时，始终保持注意力高度集中（百度一下就知道了）。而机器则不会，只要有电，就能一直按照预先设定的步骤运行。我之前就遇到一个 bug，就是上面那张公交车的图，在新增线路后，页面不会自动刷新，需要手动刷新才能看到新增的数据，这实际上属于一个优化类的问题。但是我无意间发现，若不进行手动刷新，新增第 28 条数据的时候，就会报服务器错误。这个 bug 修复后，我也针对这个 bug 编写了一个脚本，就是新增路线，我设置成重复执行 100 次。这种问题，回归起来，很费人力，浪费时间还影响心情，交给自动化就好了。

综上所述，自动化完成不了的，手工测试都能弥补，两者有效的结合是测试质量保证的关键。

### 三、UI 自动化核心是元素识别

Selenium 官方网站在使用教程中说到，定位方式有 8 种，分别是 class name、css selector、id、name、link text、partial link text、tag name、xpath




**并且告诉我们在选择使用定位方法的时候，建议遵循以下原则：**

1.若 id 和 name 在 html 中是唯一的，则优先使用这 2 种。

2.使用 css 或 xpath，他们都很灵活，但语法复杂。Xpath 性能应该是最慢的。

3.link text、partial link text 缺点在于只对连接元素起作用。

4.class name 不支持复合类名的元素。

5.tag name 是危险的方法，因为一个页面上有很多相同标签的元素。

定位举例：

```
<html lang="zh-CN">
▶ <head>…</head>
▼ <body>
  ▼ <div class="global">
      <!-- 大皮肤 -->
      <div id="big-skin" class="layout qq-skin" style></div>
      <!-- /大皮肤 -->
      <!-- 头部 -->
    ▶ <div class="layout qq-top cf" bossexpo="bg_top">…</div>
      <!-- /头部 -->
      <!-- 导航 -->
    ▶ <div class="layout qq-nav">…</div>
      <!-- /导航 -->
      <!-- 广告1 -->
    ▶ <div class="layout qq-gg gg-1 cf">…</div>
      <!-- /广告1 -->
      <!-- 要闻 -->
    ▼ <div id="news" class="layout qq-main cf">
        ::before
      ▶ <div class="col col-1 fl">…</div>
      ▼ <div class="col col-2 fl">
          <!-- 今日话题 -->
        ▼ <div class="mod m-topic" bossexpo="bg_jrht">
          ▼ <div class="hd cf">
              ::before
            ▼ <h2 class="tit active">
                <a id="topic" name="topic" class="side_ad" href="http://view.news.qq.com/" target="_blank" bosszone="jrht_logo">今日话题</a> == $0
              </h2>
              ::after
            </div>
```

要定位到上图选中的这一行，8 种定位方式的 java 代码如下：

```
// id 定位，一般情况下元素 id 是唯一的，可以使用这种方法
driver.findElement(By.id("topic"));
```

```
// name 定位，一般情况下元素 name 是唯一的，可以使用这种方法
driver.findElement(By.name("topic"));

// cssSelector 定位，可以使用相对路径或绝对路径
driver.findElement(By.cssSelector("body > div.global > div.layout.qq-main.cf
 > div.col.col-2.fl > div.mod.m-topic > div.hd.cf > h2 > a"));

// Xpath 定位，可以使用相对路径或绝对路径
driver.findElement(By.xpath("/html/body/div[1]/div[5]/div[2]/div[1]/
   div[1]/h2/a"));

// className 定位，一般情况下一个页面会有多个相同 className 的元素
driver.findElement(By.className("side_ad"));

// linkText 定位，只能定位链接元素，文本内容必须完全匹配
driver.findElement(By.linkText("今日话题"));

// partialLinkText 定位，只能定位链接元素，提供部分的文本内容，就可以定位
driver.findElement(By.partialLinkText("话题"));

// tagName 定位，会定位到所有 a 标签的元素
driver.findElement(By.tagName("a"));
```

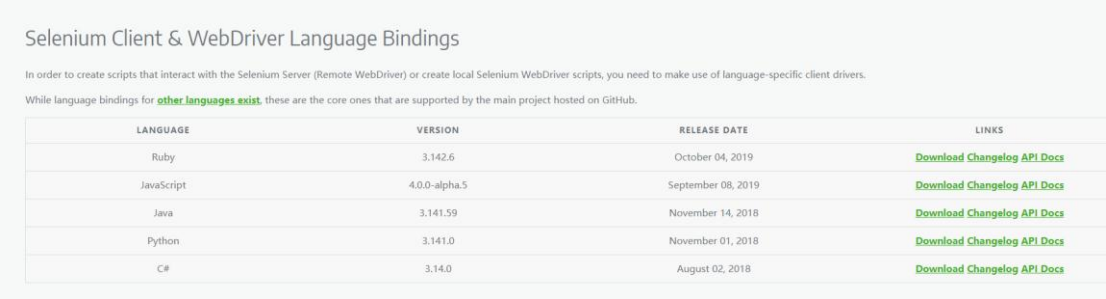
若某种方法定位到多个元素，则会返回第一个元素。其中 xpath 和 cssSelector 上述代码使用的是绝对路径，也可以使用相对路径，以 xpath 为例，css 类似，代码如下:

```
// xpath 定位，使用标签名和元素属性定位
driver.findElement(By.xpath("//a[@id='topic']"));

// xpath 定位，仅使用元素属性定位
driver.findElement(By.xpath("//*[@id='topic']"));

// xpath 定位，使用多个元素属性定位
driver.findElement(By.xpath("//*[@id='topic' and @bosszone='jrht_logo']"));

// xpath 定位，使用元素文本内容定位
driver.findElement(By.xpath("//*[text()='今日话题']"));

// xpath 定位，使用元素属性定位，支持模糊查找
```

```
driver.findElement(By.xpath("//*[contains(@class,'side')]"));

// xpath 定位，定位其他元素，查找相对位置
// 假设上图选中的这一行，没有 id="topic" 这个属性
driver.findElement(By.xpath("//div[@id='news']/div[2]/div[1]/div[1]/h2/a"));
```

### 四、Jenkins 让 Selenium-UI 自动化测试如虎添翼

Selenium 自动化测试，对于脚本的编写，语言只是一个载体，我熟悉 java，所以用 java 编写。Selenium 支持 Ruby，C#，java，python，javaScript 语言。在官网下载对应的包就行了。



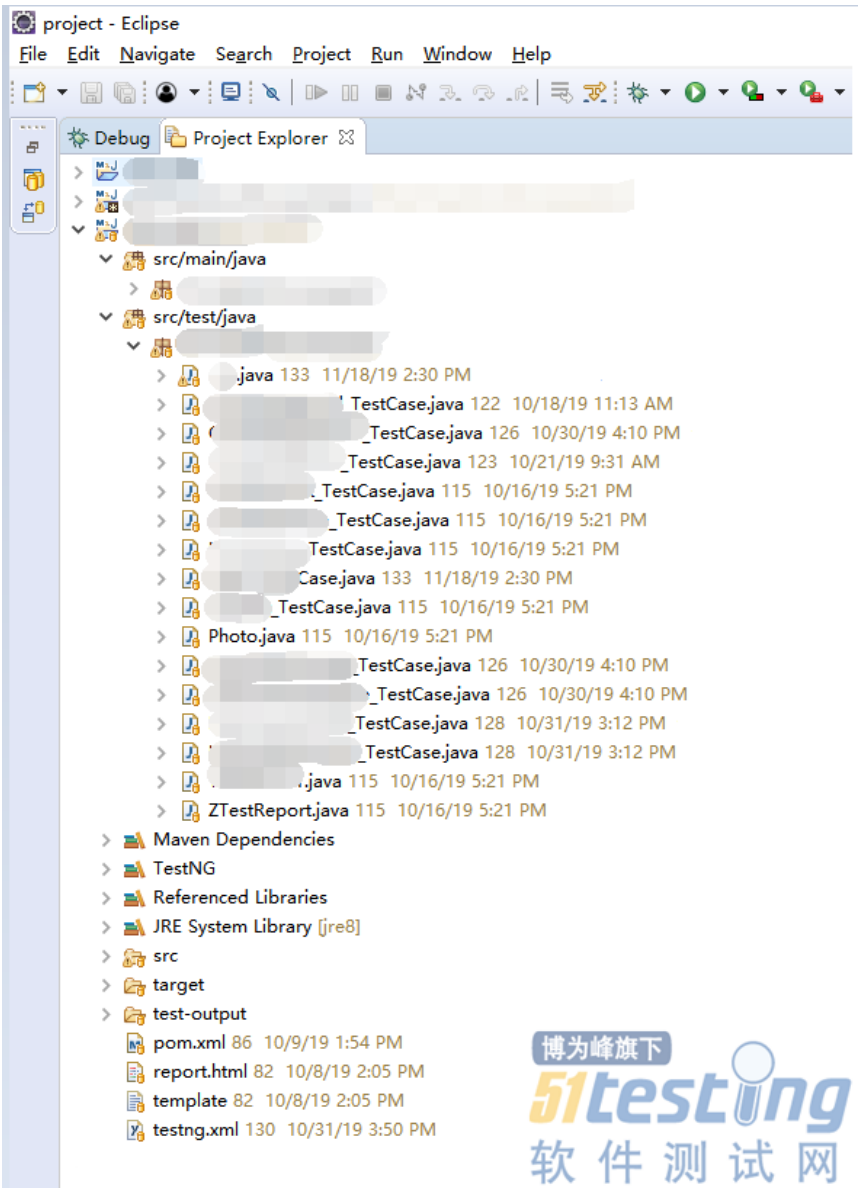
Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers.

While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on GitHub.

| LANGUAGE | VERSION | RELEASE DATE | LINKS |
|---|---|---|---|
| Ruby | 3.142.6 | October 04, 2019 | Download Changelog API Docs |
| JavaScript | 4.0.0-alpha.5 | September 08, 2019 | Download Changelog API Docs |
| Java | 3.141.59 | November 14, 2018 | Download Changelog API Docs |
| Python | 3.141.0 | November 01, 2018 | Download Changelog API Docs |
| C# | 3.14.0 | August 02, 2018 | Download Changelog API Docs |

编写完测试脚本，可以将它集成到 jenkins 上，定时运行。

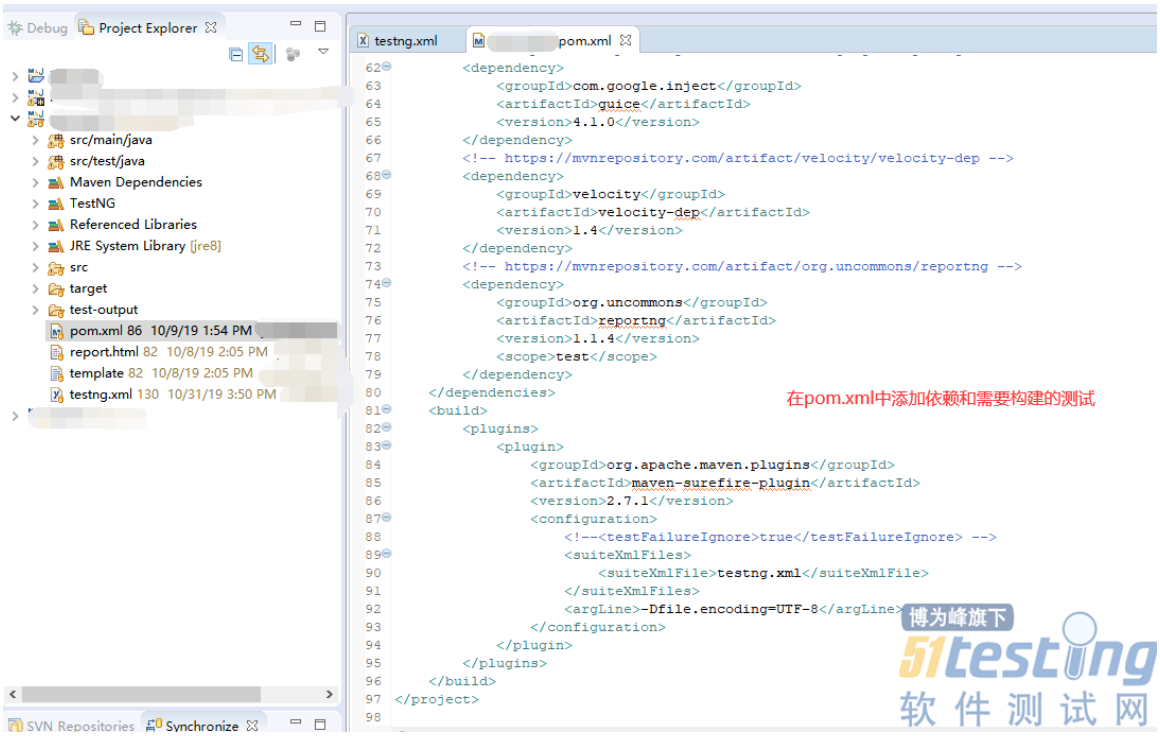



我使用的是 maven+testng+svn 进行管理。具体操作如下:

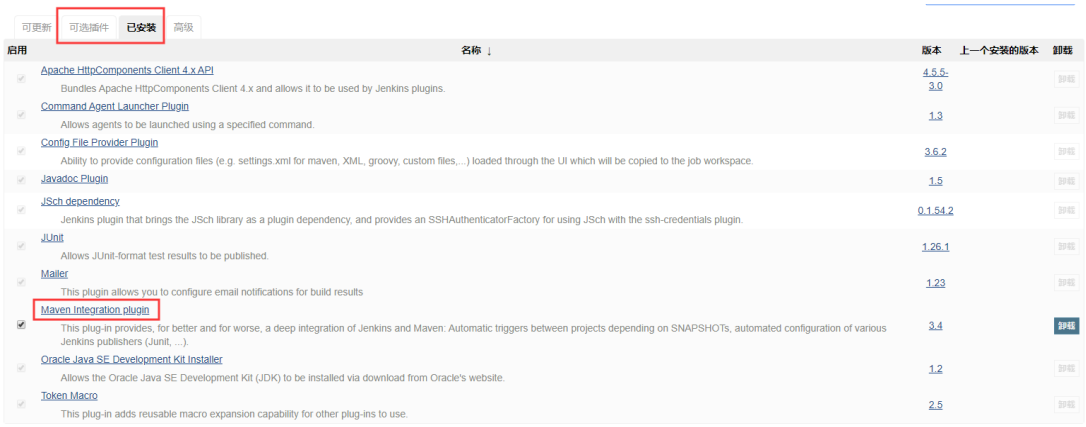
1.首先写好 testng.xml，pom.xml，作为运行的对象

testng.xml

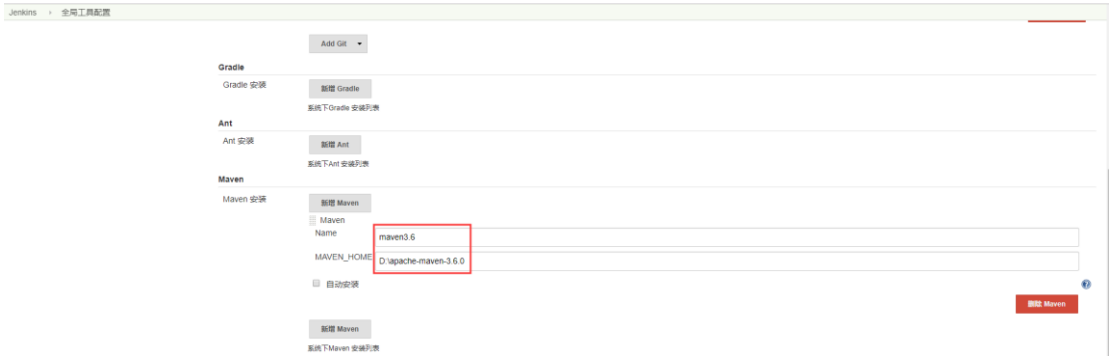
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <test thread-count="5" name="TestAllCase">
        <classes>
            <class name="[redacted]_TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted]TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted] TestCase" />
            <class name="[redacted] TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted] TestCase" />
            <class name="[redacted]_TestCase" />
            <class name="[redacted]e_TestCase" />
        </classes>
    </test>
    <listeners>
        <listener class-name="org.uncommons.reportng.HTMLReporter" />
        <listener class-name="org.uncommons.reportng.JUnitXMLReporter" />
        <listener class-name="[redacted]TestReport" />
    </listeners>
</suite>
```




2.在 jenkins 中安装 Maven Intergration plugin 插件

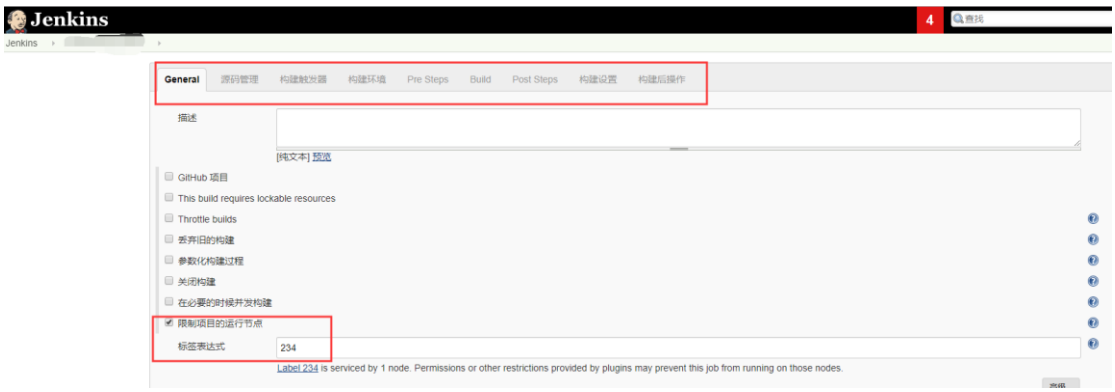



3.在 jenkins 中，系统管理→全局工具配置中设置 maven 路径

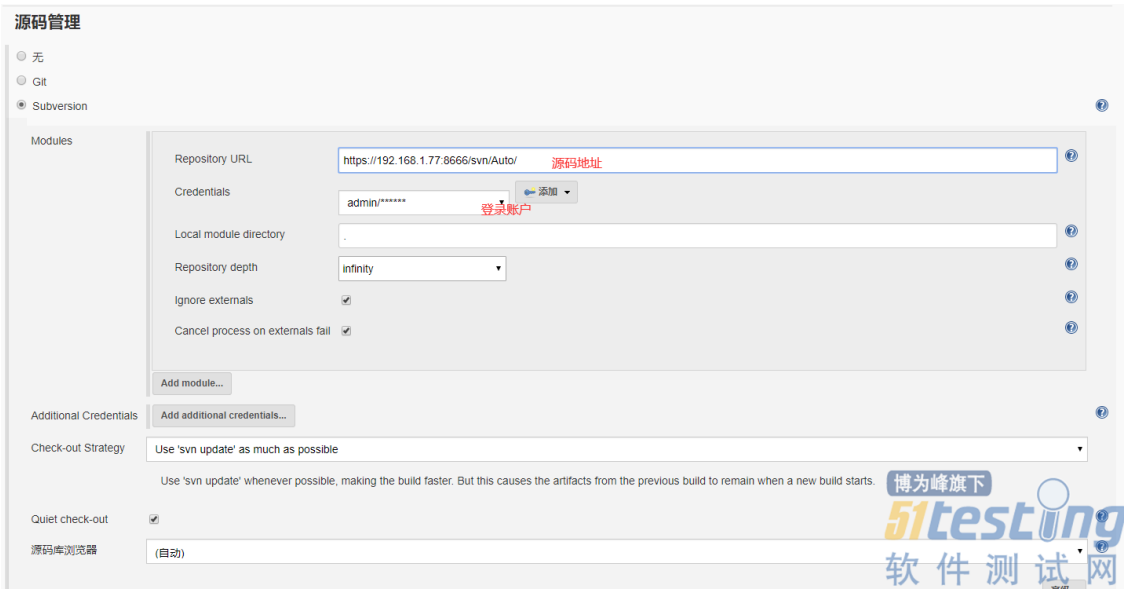



4.新建任务，选择构建一个 maven 项目


5.配置 General 内容，在上一篇《Jenkins 系列之自动化部署（1）》中，我们已经详细介绍过如何配置远程运行节点，这里不再赘述。需要注意的是，UI 自动化测试，若在本机构建可能会出现后台运行的现象，为了测试的有效性，还是建议找一台独立的，环境干净的机器，让 UI 测试在前台进行。




6.源码管理




上一篇详细介绍了 git 管理源码的配置，这一篇使用的 svn 管理源码，配置基本相同

7.构建触发器，每天执行一次，在 10 点整运行

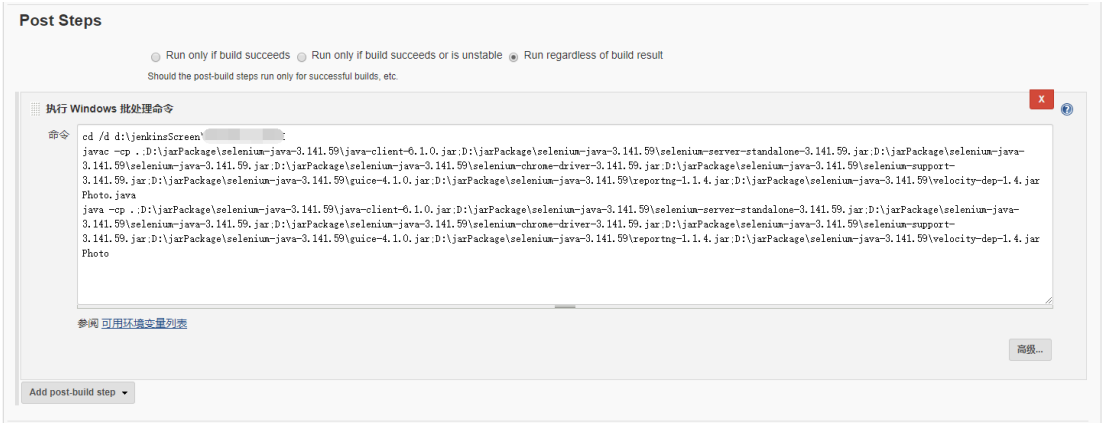

8.测试前，清理工作路径下，上一次测试结果（若需要保留，则可以不清理）


第一句是进入工作环境文件夹

后面 2 句是需要删除的文件。

9.编写构建命令


10.测试结束后，需要执行的命令可以放在 PostSteps 中执行

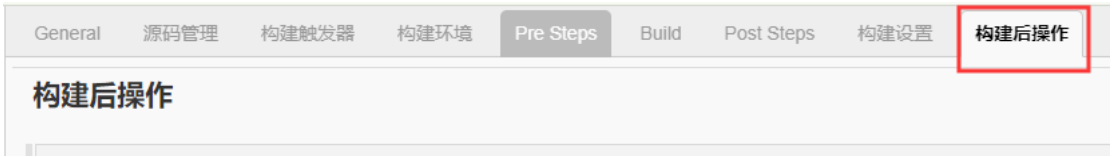



我这里是实现了，将本地测试完成后的测试报告打开，然后截图，转换成 base64 编码保存到本地 txt 文档。（目的是为了发送邮件的时候，把图片帖进去）源码如下

```
public class Photo {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        System.setProperty("webdriver.chrome.driver",
                "C:\\Program Files (x86)\\Google\\Chrome\\"
                + "Application\\chromedriver.exe");
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("D:\\Jenkins\\workspace\\Auto\\report.html");
        driver.manage().window().maximize();
        try {
            File scrFile = ((RemoteWebDriver) driver).getScreenshotAs(
                    OutputType.FILE);
            Files.copy(scrFile, new File("D:\\Jenkins\\workspace\\Auto"
                    + "\\report.jpg"));
            // 图片转换成 base64
            InputStream in = new FileInputStream("D:\\Jenkins\\workspace"
                    + "\\Auto\\report.jpg");
            byte[] data = null;
            data = new byte[in.available()];
            in.read(data);
            in.close();
```

```
            BASE64Encoder encoder = new BASE64Encoder();
            String base64 = ""
                  + encoder.encode(Objects.requireNonNull(
                        data)).replaceAll("\r|\n", "");
            // base64 覆盖写入 txt
            File txt = new File("D:\\Jenkins\\workspace\\Auto\\report.txt");
             FileWriter fw = new FileWriter(txt, false);//设置成 true 就是追加
            fw.write(base64);
            fw.close();
         } catch (IOException e) {
            e.printStackTrace();
         } catch (Exception e) {
            e.printStackTrace();
         } finally {
            driver.quit();
         }
      }
   }
```

11.构建后的操作 tab 页，我实现 3 个功能，①发送邮件，②让测试报告可以在 jenkins 的 web 页面中直接查看，③jenkins 禁用了第三方页面的样式，我们需要将其开启

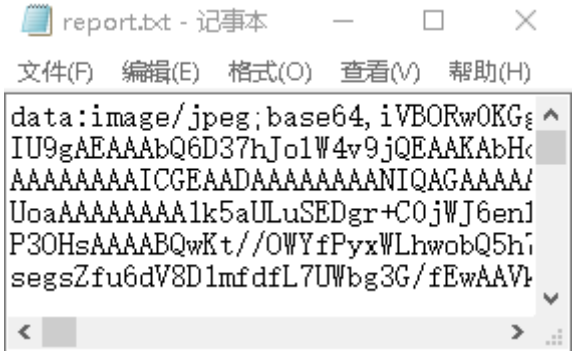



先说邮件发送，上一篇邮件如何发送已经说过了，但是这次我们加入了测试报告的截图

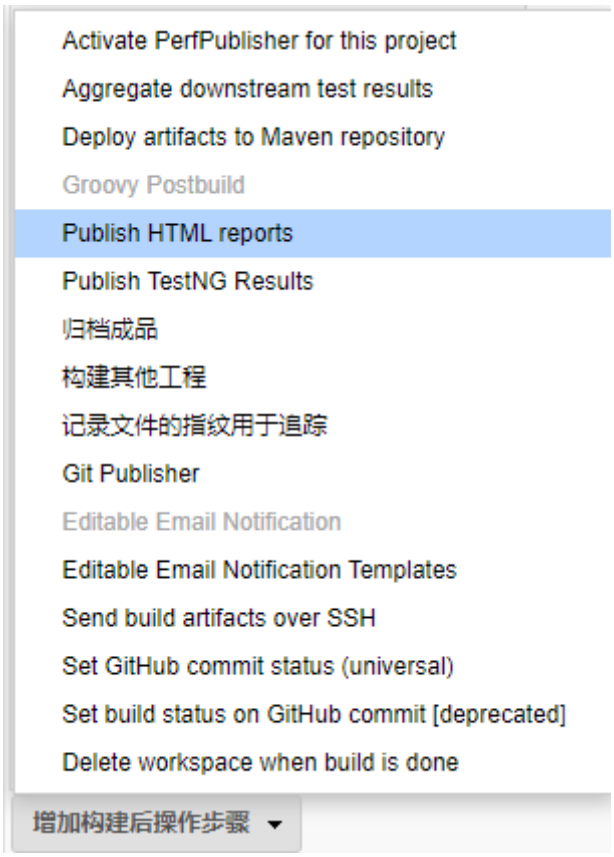
Default Content

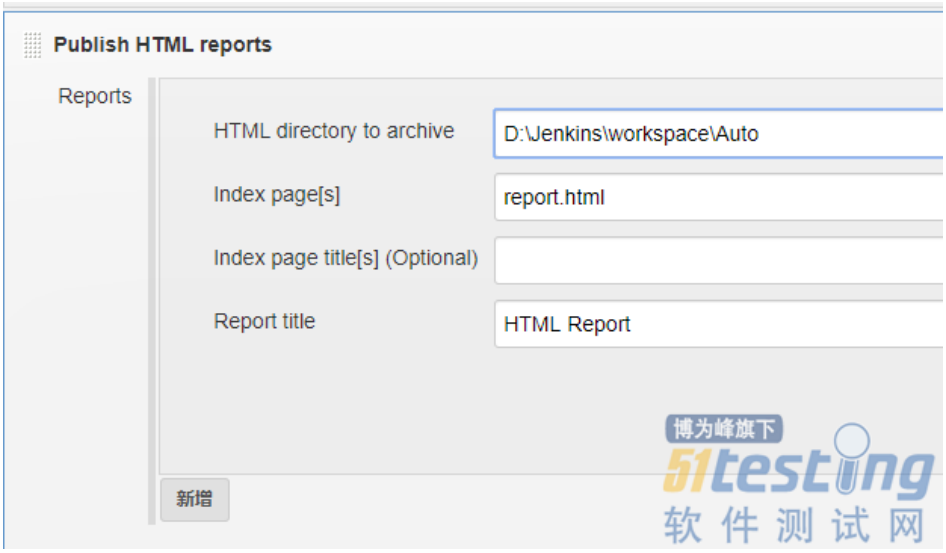
```
<!DOCTYPE html>
  <html>
  <head>
  <meta charset="UTF-8">
  </head>
  <body leftmargin="8" marginwidth="0" topmargin="8" marginheight="4" offset="0">
    <table width="95%" cellpadding="0" cellspacing="0" style="font-size: 16pt; font-family: Tahoma, Arial, Helvetica, sans-serif">
      <tr>
        <td><br />
          <b><font color="#0B610B"><font size="6">构建信息</font></font></b>
          <hr size="2" width="100%" align="center" /></td>
      </tr>
      <tr>
        <td>
          <ul>
          <div style="font-size:18px">
            <li>构建名称：${PROJECT_NAME}</li>
            <li>构建结果：$BUILD_STATUS </li>
            <li>构建编号：${BUILD_NUMBER}</li>
            <li>触发原因：${CAUSE}</li>
            <li>测试详情：见附件report.html</li>
            <img src="${FILE ,path="report.txt"}">
          </div>
          </ul>
        </td>
      </tr>
    </table></font>
  </body>
  </html>
```

因为我们发送的邮件是 html 格式的，所以除了文字，还可以放入图片。图片放入 html 中有两种方式，第一种是链接，由于我们的图片是保存在本地的，以链接的方式加入到 html 中，发送邮件后，是无法查看的。只能用第二种方式，将图片转换成 base64 编码，将编码放入 html 中，就不再受到网络的限制了。我的第 10 步操作就是将图片，转换成 base64 编码。report.txt 打开后如下图

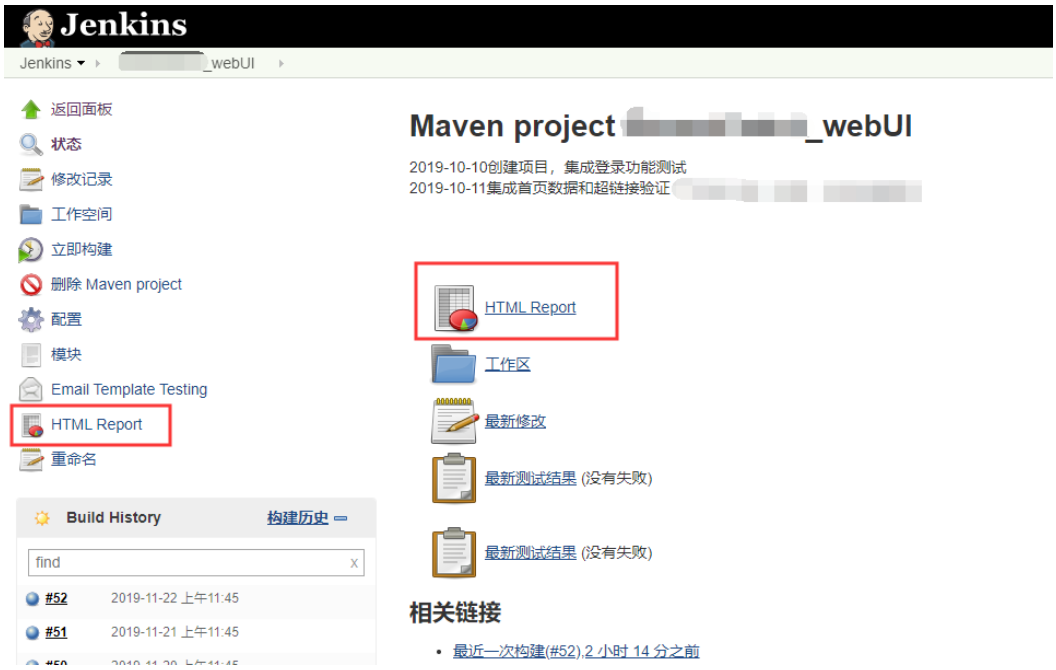



12.接下来让 jenkins 任务，可以直接打开测试报告。我们需要安装一个插件 HTML Publisher。



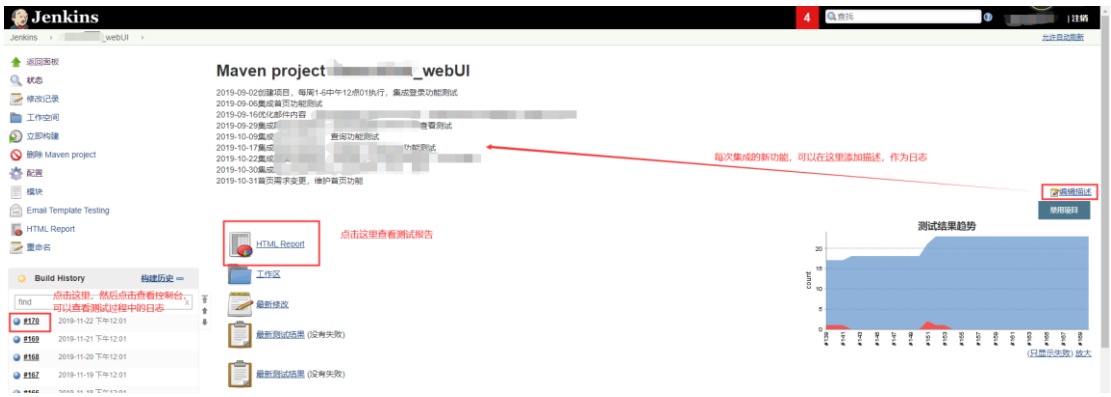

安装完插件，就可以在构建后的步骤中，找到 Publish HTML reports 选项了。然后添加配置，第一行代表本地测试报告的路径，第二行是测试报告索引文件。添加了这个功能，在项目页面就会有一个按钮，允许我们直接查看测试报告




13.由于 jenkins 禁用了第三方页面的样式，所以我们每次构建完成以后，最好都执行一次放开权限的命令。添加 Groovy Script。内容如下

System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "")


所有配置都完成了。保存，返回项目页面，构建一次。确保配置正确，可以得到预期的结果。否则根据控制台输出的日志，进行错误排查。

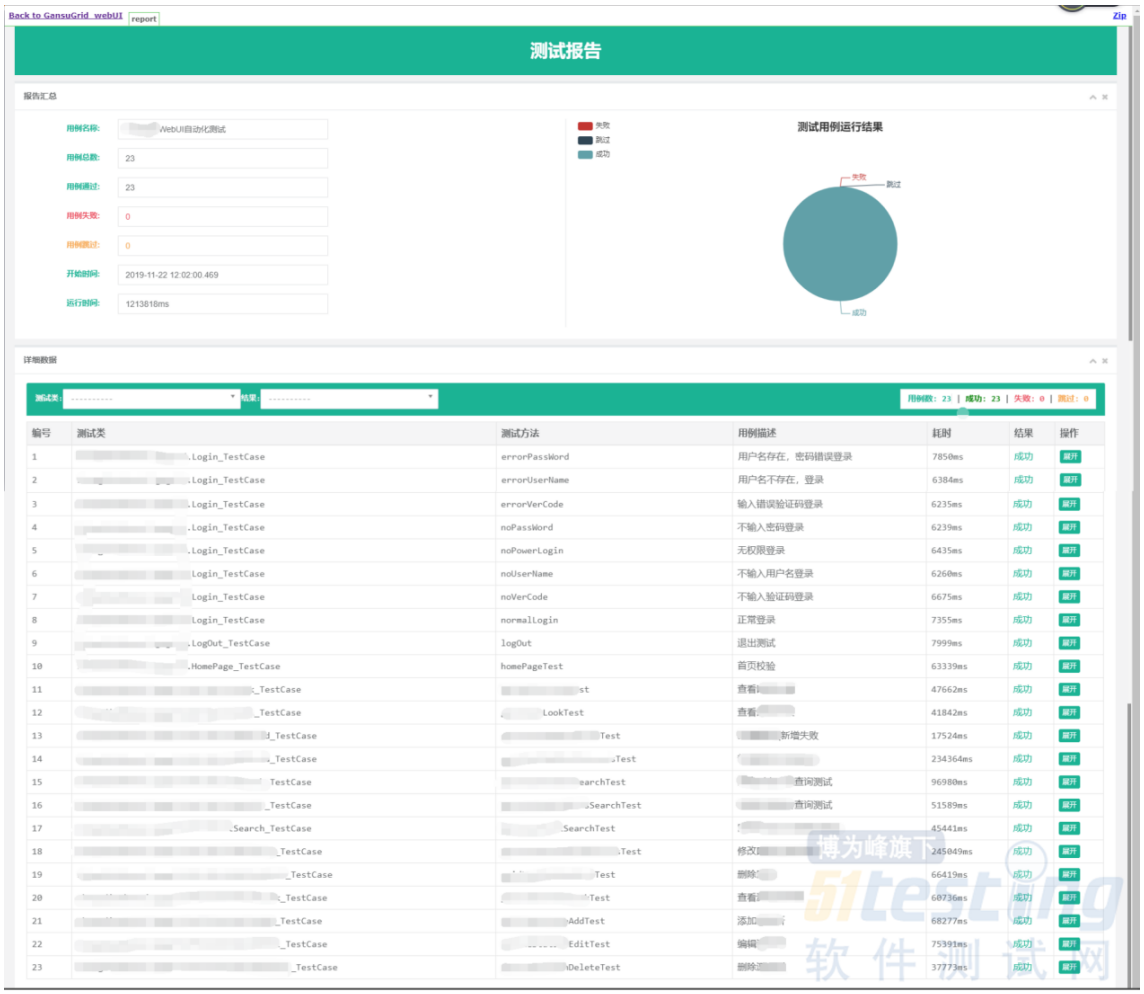



这是测试报告的页面。由于 testng 提供的测试报告样式太丑，所以我使用了 ztest 进

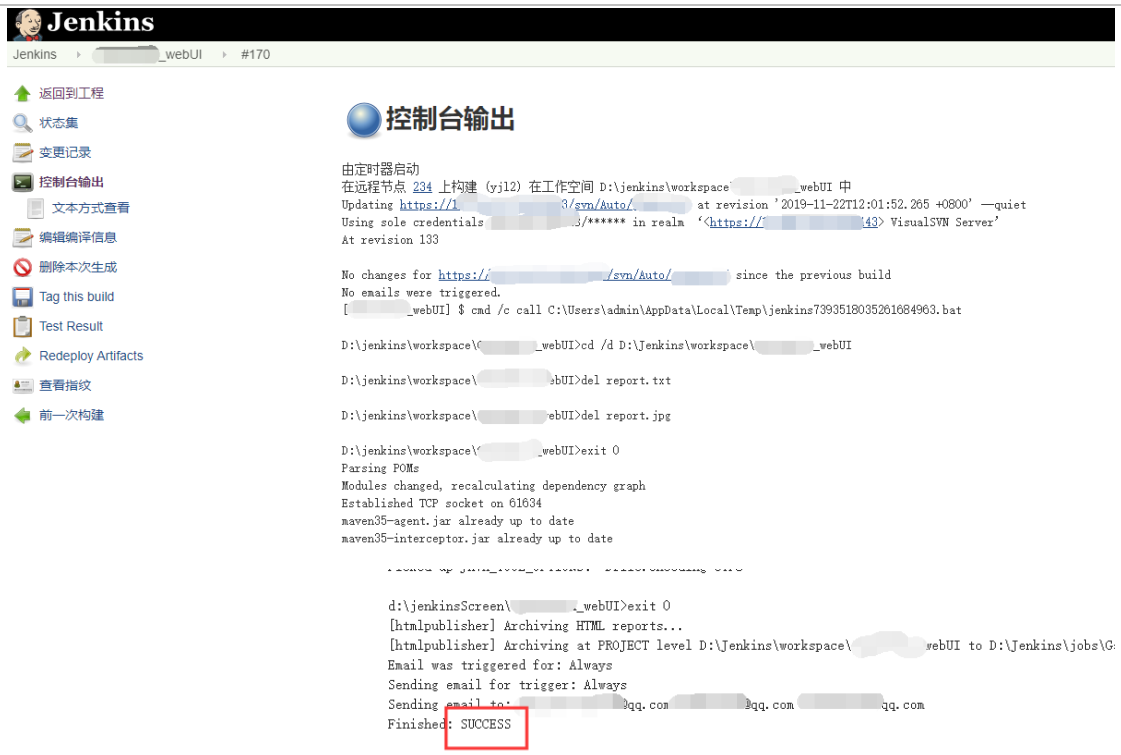
行美化，github 源码地址是 https://github.com/zhangfei19841004/ztest

这套模版最后一次更新是在 2019 年 7 月 31 日，目前还有些简单的 bug，大家可以自行修复。也可以根据自己项目的需要，在测试报告上添加信息，做二次开发。

图中运行这一列如果是失败，点击展开按钮可以查看到报错信息。




下面是控制台输入的日志




一套完整的 Selenium-UI 自动化测试持续集成方案介绍完毕。各位小伙伴是不是有动手的冲动了？既然已经坚持看完了，那就动手试试看吧。

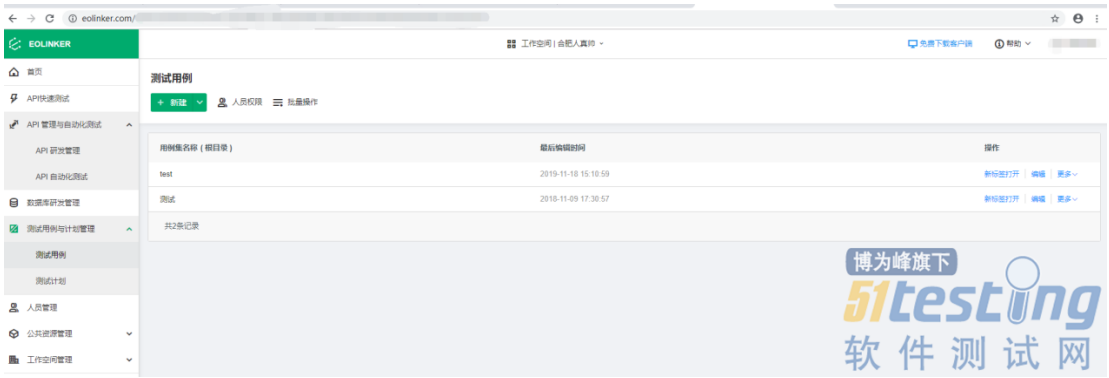
# Jenkins 系列之 Jmeter-API 自动化测试（3）

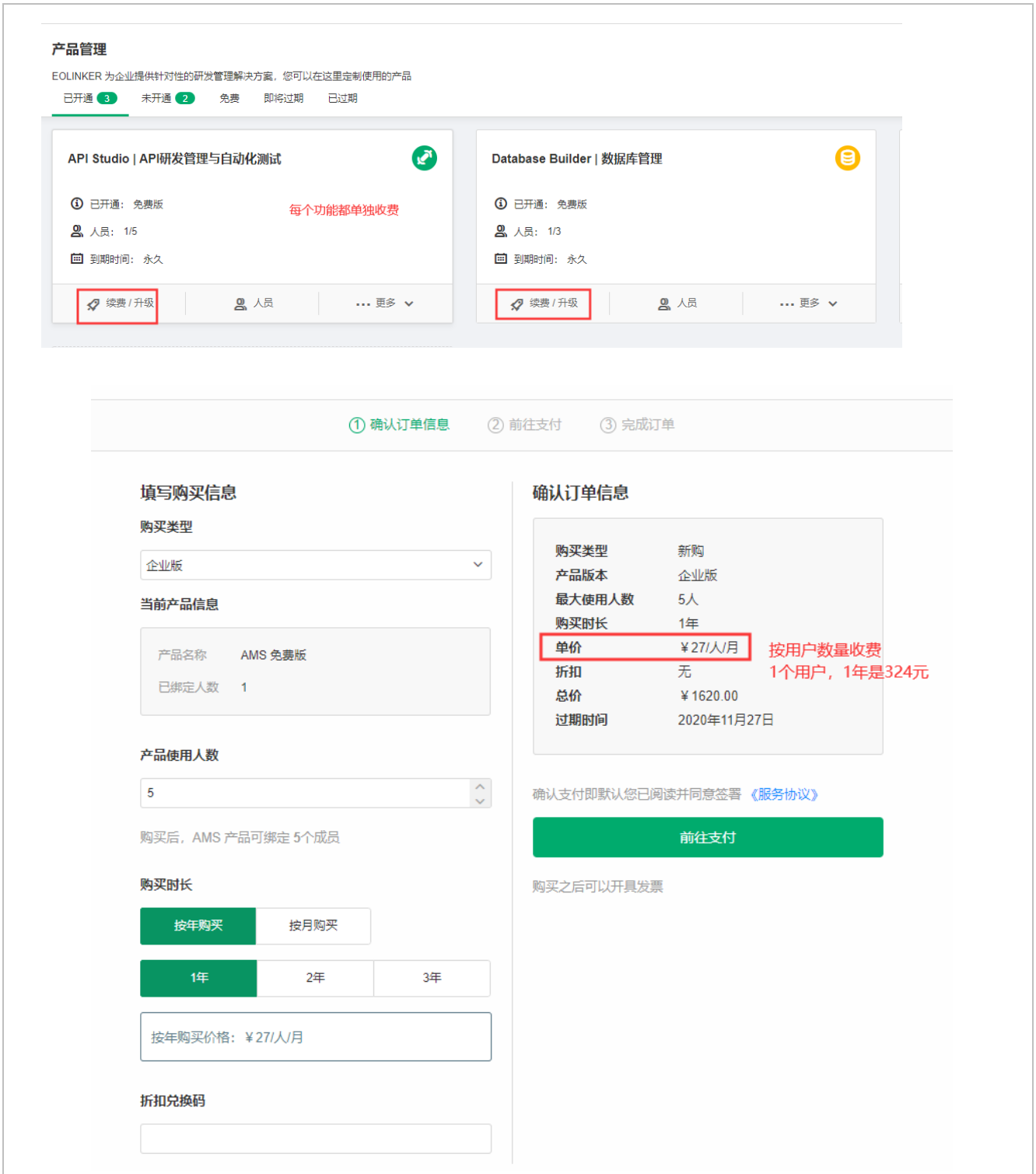
◆ 作者：合肥人真帅

自动化测试技术的火热除了表现在 UI 层，还表现在接口层。能实现接口测试的工具和框架有很多，例如 Jmeter、Postman、eoLinker、Request+Unittest 等等。每一种工具或框架都有优缺点。选择哪一种，需要根据自身和公司的实际情况来决定。

## 一、选型

eoLinker 纯图形界面、零代码量使用方便，支持在浏览器中安装插件，且提供了测试用例的管理，可以自动化执行所有用例。但网页版所有数据都是保存在服务商的数据库中，若公司涉及到保密问题，则不可以使用网页版，且纯图形界面不支持上传附件这类接口的测试。虽然开源版支持本地部署，但是功能非常的少。全功能本地部署，费用昂贵。

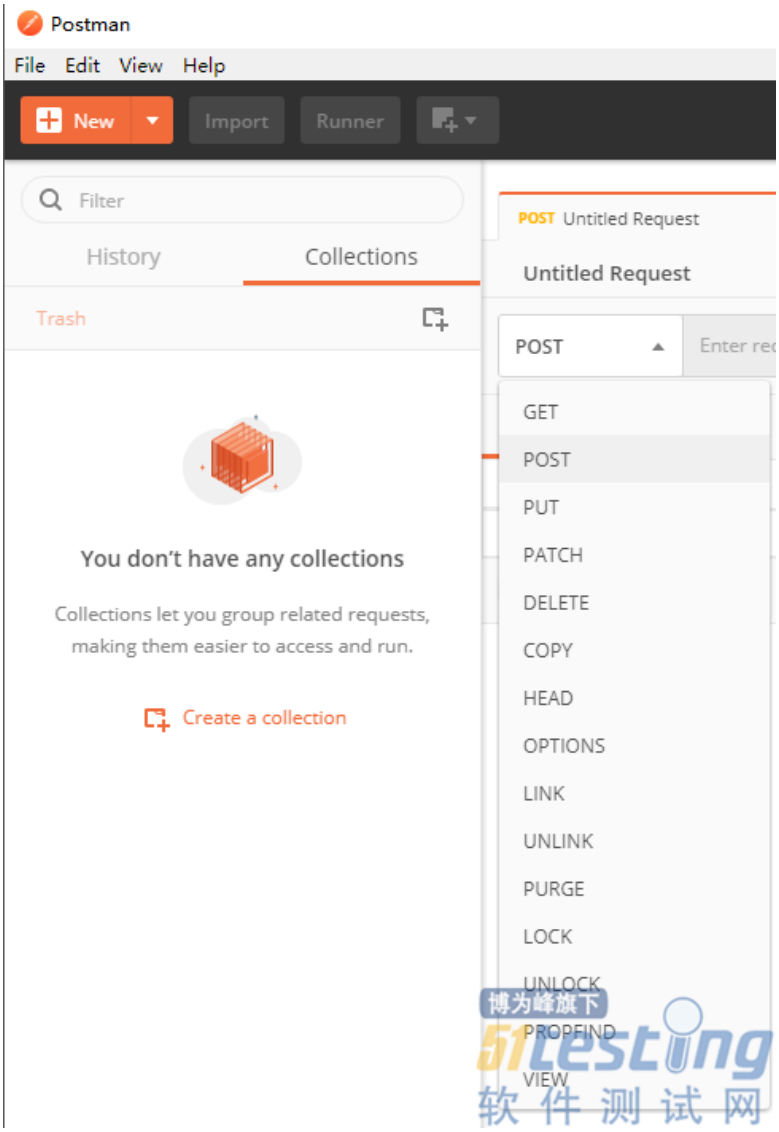


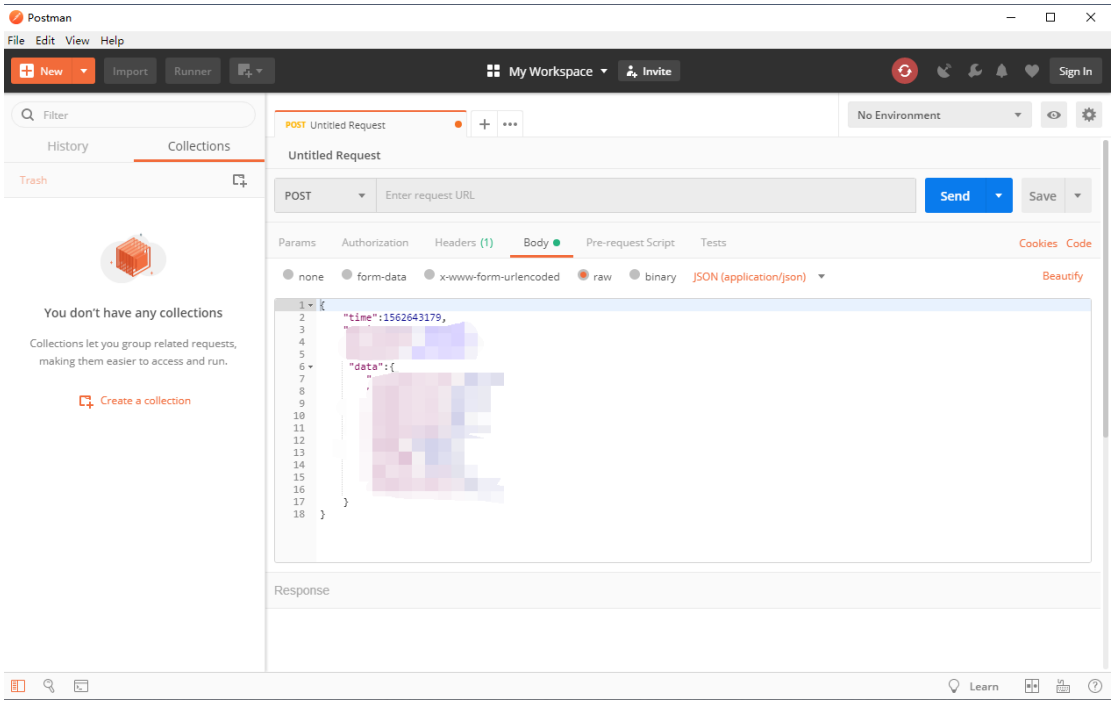



Request+Unittest 是测试框架，纯代码级，python 语言。应对各种需求，开发起来都很灵活。但是相对于图形化的工具而已，对人员水平要求较高，上手速度明显慢于图形化工具。若公司中有专门的接口自动化测试开发团队，这将是一个很好的选择。既然是语言开发类，也可以使用 java+testng，可根据团队的语言能力，选择开发框架。

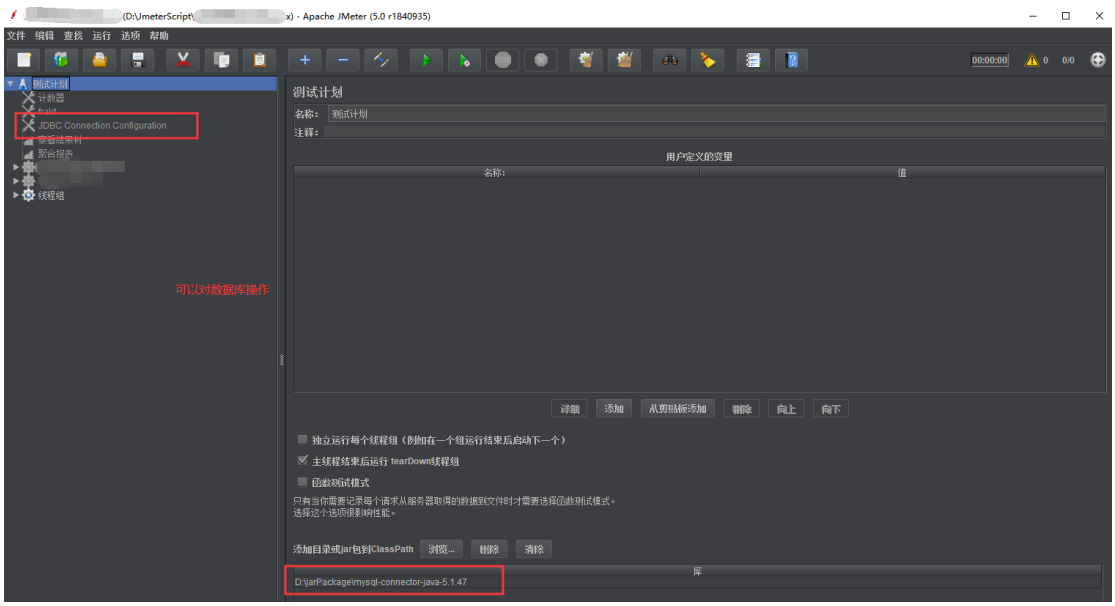
Postman 大部分 web 项目的开发人员和测试人员都会选择使用这个工具。图形化的

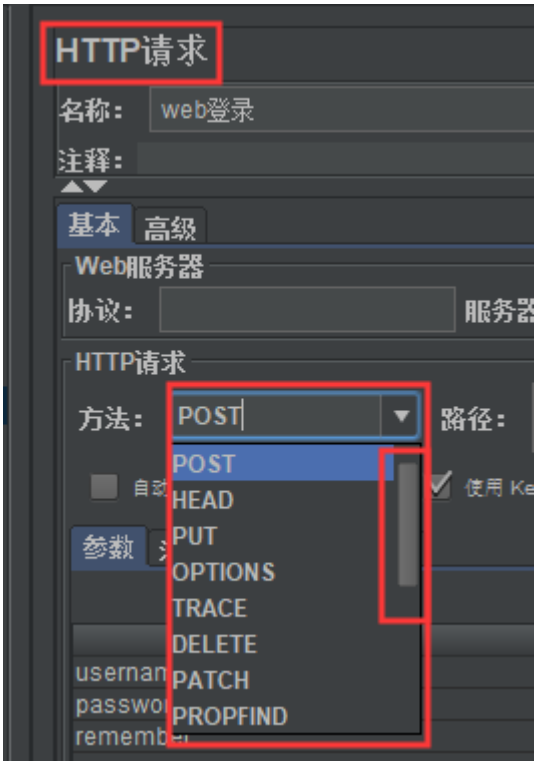
界面，操作方便，执行结果易阅读，易调试。可以实现自动化批量运行。但免费版仅支持 http 和 https 协议，且无法读写数据库。






Jmeter 支持很多种类的协议，支持从数据库动态取值，支持第三方插件，支持接口对文件的操作。由于代码是开源的，即便没有支持的协议，也可以进行二次开发来完成。但测试报告偏向于性能，大部分是性能指标，且断言功能不够强大，虽然提供了 beanshell 断言，但是提高了使用门槛，需要一点的编程能力。







为了节省成本，我选择开源软件。为了降低开发时间，我选择图形化的工具，减少脚本的编写。我自身具备一定的开发能力，且需要做性能测试，所以我选择 Jmeter。在开发完性能测试脚本后，只要稍加修改，就可以获得接口自动化测试脚本。大家还是需要根据自身情况合理选择工具。

选择好工具后，编写测试脚本。我们需要创建线程组、配置元件以及一系列的请求。这些都是入门的内容，百度上都有，这里就不说了。下面给大家分享一下每个项目都用的着的一些技巧：①如何提取响应结果中的某个参数？②获得参数后，如何使用？③如何让系统自动判断响应结果是否正确？

## 二、提取参数

在实际项目中，有很多场景需要使用到响应结果中的参数。例如登录的响应结果中包含了 token，后续的请求，都要使用这个 token 作为身份认证。例如 QQ 邮箱中修改一封草稿邮件，在修改之前，我们要获取这封邮件的 sid。例如需要确认某个请求响应时间是否在 5 秒以内，我们需要获取发送时间和响应时间。

目前很多项目的接口返回参数，都是以 json 格式去发送的。所以我们可以用 json 提取器来获取想要的参数。若不是 json 格式，我们可以使用万能的正则表达式提取器来提取。除了这 2 种常用的方法外，jmeter 还提供了近 10 种方法。




**第一种方法，json 提取。**选中登录请求，选择后置处理器→json 提取器。

Names of created variables: 变量名

JSON Path expressions: 需要提取内容的路径

Match No.(0 for Random): 提取第几个匹配的值（若有 N 个匹配的值，0 表示随机取值，-1 表示全部取值，1 表示取第 1 个，2 表示取第 2 个。。。N 表示取第 N 个）

Compute concatenation var (suffix_All): 勾选表示取所有的值，并保存到一个变量中，命名方式是第一行设置的变量名加上_All

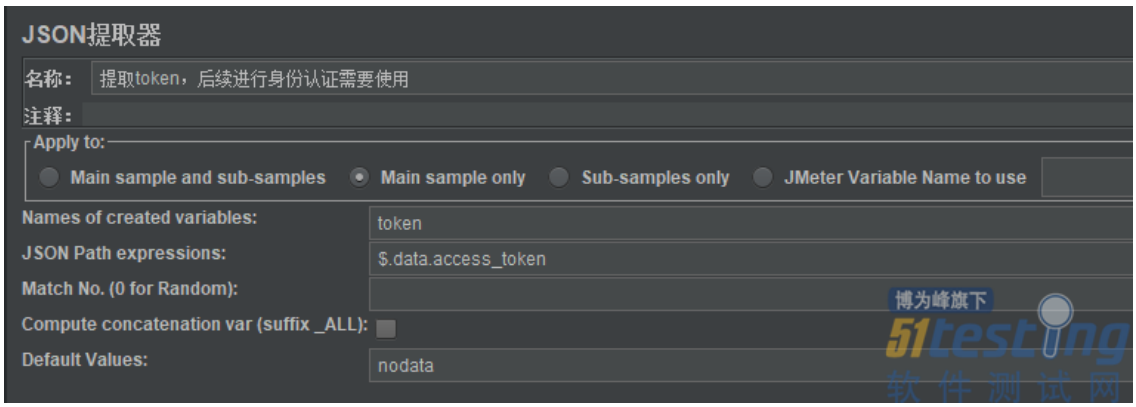
Default Values: 没有提取到值，将给变量一个默认值

下面举例说明

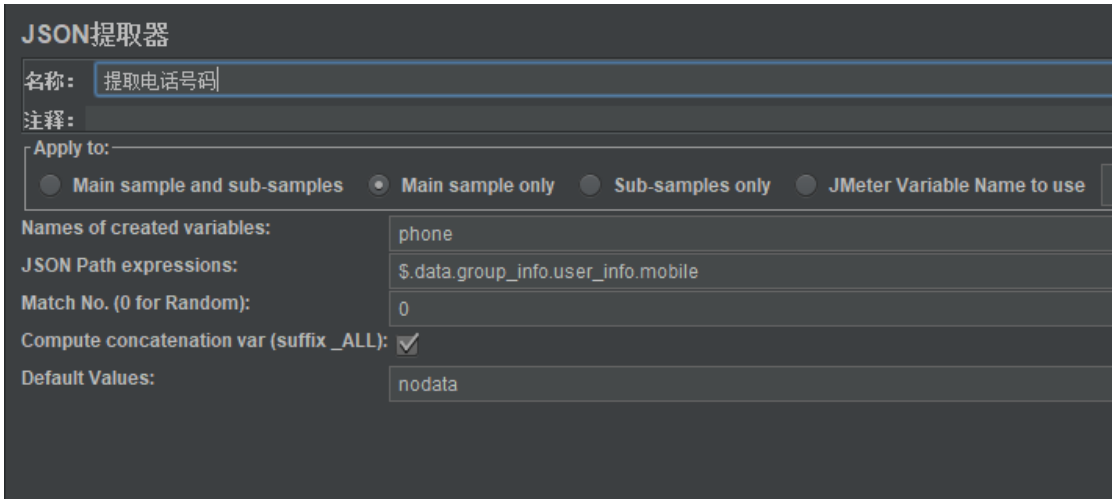

```
{
  "errno": 0,
  "data": {
    "user_name": "zhangsan",
    "mobile": "19900000001",
    "created_at": "2019-09-06 15:55:21",
    "real_name": "张三",
    "group_info": {
      "user_info": {
        "mobile": "17700000001",
        "created_at": "2019-09-06 15:55:21",
        "token_type": 2,
        "user_role": "普通用户",
        "login_count": 188,
        "password": "123456",
        "updated_at": "2019-09-06 15:55:21",
        "id": "991222",
      },
      {
        "mobile": "17700000002",
        "created_at": "2019-09-21 11:33:16",
        "token_type": 2,
        "user_role": "普通用户",
        "login_count": 157,
        "password": "123456",
        "updated_at": "2019-09-22 09:50:01",
        "id": "991223",
```

```
        },
      },
      "token_type": 2,
      "access_token": "d791640014b411eabf110bd02de67c73",
      "login_count": 871,
      "user_id": "331236",
      "finger_sign": "147"
    },
    "errmsg": ""
}
```


这是一个登录后的响应请求，若我们需要提取 access_token，则可以这样设置




第一行设定变量名为 token，第二行根据给定的路径进行查找，找到后将值赋给 token。若找不到，将最后一行的“nodata”赋给 token。根据$.data.access_token，我们可以找到第 31 行的 d791640014b411eabf110bd02de67c73 并将他赋给 token，以后的请求，我们都可以使用这个变量 token 了。




如果我们要提取电话号码，则$.data.group_info.user_info.mobile 会匹配到第 10 行和第 20 行，这里的 Match No 我设置的是 0，代表随机获取这 2 个号码中的一个赋给

phone。并将两行的内容保存到变量 phone_ALL 里。

Json 提取器很简单，能否准确提取到想要的内容，主要就是 JSON Path expressions 设置的正确与否。在第一篇 selenium 自动化测试中，我们说过 Xpath，其实 json path 与之类似，具体语法对比如下

| XPath | JSONPath | 描述 |
|---|---|---|
| / | $ | 根节点 |
| . | @ | 现行节点 |
| / | . or [] | 取子节点 |
| .. | n/a | 取父节点，Jsonpath未支持 |
| // | .. | recursive descent（就是不管位置，选择所有符合条件的条件）. JSONPath 从 E4X规范中继承了这个语法的使用法。 |
| * | * | 匹配所有元素节点 |
| @ | n/a | 根据属性访问，Json不支持，因为Json是个Key-value递归结构，不需要。 |
| [] | [] | 迭代器标示（可以在里边做简单的迭代操作，如数组下标，根据内容选值等） |
| \| | [,] | 支持迭代器中做多选。 |
| n/a | [start:end:step] | 从ES4规范继承的切片方法（非常实用） |
| [] | ?() | 支持过滤操作. |
| n/a | () | 支持表达式计算 |
| () | n/a | 分组，JsonPath不支持 |




```
{ "store": {
  "book": [
    { "category": "reference",
      "author": "Nigel Rees",
      "title": "Sayings of the Century",
      "price": 8.95
    },
    { "category": "fiction",
      "author": "Evelyn Waugh",
      "title": "Sword of Honour",
      "price": 12.99
    },
    { "category": "fiction",
      "author": "Herman Melville",
      "title": "Moby Dick",
      "isbn": "0-553-21311-3",
      "price": 8.99
    },
    { "category": "fiction",
      "author": "J. R. R. Tolkien",
      "title": "The Lord of the Rings",
```

```
      "isbn": "0-395-19395-8",
      "price": 22.99
    }
  ],
  "bicycle": {
    "color": "red",
    "price": 19.95
  }
 }
}
```


**示例表达式:**

$.store.book[*].author: 商店所有书籍的作者（四个作者）

$..author ： 所有作者

$.store.* ： 商店所有的东西，包括 book 和 bicycle

$.store..price ： 所有东西的价格

$..book[2] ： 第三本书

$..book[0,1] /$..book[:2] ： 前两本书

$..book[?(@.isbn)] ： 用 isbn 编号过滤所有书籍

$..book[?(@.price<10)] ： 过滤所有比 10 更便宜的书

$..* ： XML 文档中的所有元素

**第二种方法，正则表达式提取。**


和 json 类似，关键就在表达式"access_token":"(.*?)"

在响应的数据中，其中有一行是这样的:

"access_token": "d791640014b411eabf110bd02de67c73",

那么我们的表达式，括号中的内容，就代表要提取的内容;

.号代表匹配任意字符串;

+号代表匹配一次或多次;

*号代表匹配任意次（包括 0 次）;

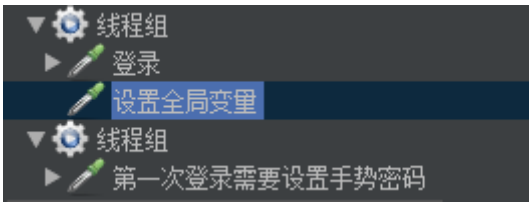
?号代表不要贪婪，找到第一个匹配项，就停止匹配;

正则表达式的语法比较复杂，这里就不过多的介绍了，有兴趣可以自己搜集资料，去练习。

### 三、参数传递

参数传递分为两类，线程内传递、线程间传递。以上面提取到的 token 为例，若想使用这个参数，在线程内是这样写的：${token}


跨线程传递，需要借助 beanshell 取样器，先将变量通过 setProperty 函数设置为全局变量，然后其他线程再通过 P 函数去引用，具体做法如下:





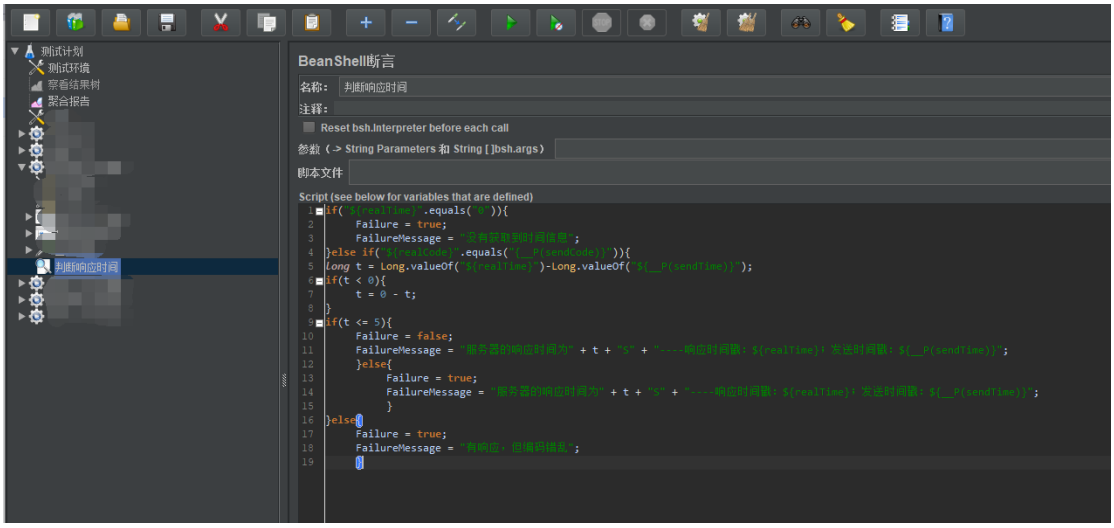
登录和设置全局变量 这 2 个请求在同一个线程组内，所以登录请求获取到的 token，在设置全局变量的请求中，可以直接使用${token}，前面的“allThreadUseToken”是全局变量名，可以随便起，自己记得住就行，最好做到见名知意。

“第一次登录需要设置手势密码”这个请求和登录请求在不同的线程组，所以无法直接使用${token}，必须使用 allThreadUseToken，格式为${__P(allThreadUseToken)}，如

下图：


## 四、断言

所有请求都可以运行起来，我们还需要检查他的响应结果是否正确。Jmeter 提供了很多断言方式，和上述的 json 提取器的样式类似，设置都很简单。下面对 beanshell 断言进行一个介绍，因为其比较灵活，可以自己写断言内容。




```
if("${realTime}".equals("0")){
    Failure = true;
    FailureMessage = "没有获取到时间信息";
}else if("${realCode}".equals("{__P(sendCode)}")){
long t = Long.valueOf("${realTime}")-Long.valueOf("${__P(sendTime)}");
```

```
if(t < 0){
    t = 0 - t;
}
if(t <= 5){
    Failure = false;
    FailureMessage = "服务器的响应时间为" + t + "S" +
        "----响应时间戳：${realTime}；发送时间戳：${__P(sendTime)}";
    }else{
        Failure = true;
        FailureMessage = "服务器的响应时间为" + t + "S" +
        "----响应时间戳：${realTime}；发送时间戳：${__P(sendTime)}";
        }
}else{
    Failure = true;
    FailureMessage = "有响应，但编码错乱";
    }
```

第一行的${realTime}是响应结果中获取到的实际时间，第四行的${realCode}是获取到的时间格式，第四行的{__P(sendCode)}是另外一个线程内自定义的时间格式，第五行的${__P(sendTime)}是另外一个线程内自定义的发送时间。

Beanshell 断言中 Failure 取 true，代表发现错误，预期结果与实际结果不符；Failure 取 false，代表预期结果和实际结果相同；FailureMessage 是输出的错误信息。

上述代码的意思为：先判断是否提取到了真实响应时间，若没提取到，代表请求失败了。若提取到，则判断时间格式是否正确。若格式正确则比较发送时间和响应时间是否大于 5 秒。

还是以登录为例，若数据库中正确的帐号密码是“zhangsan”，“123456”。发送登录请求，输入正确的帐号密码，请求成功，响应结果如下：


此时我们可以提取 errno 为 0 进行 beanshell 断言。

若发送请求时，给出了错误的帐号密码，响应结果如下

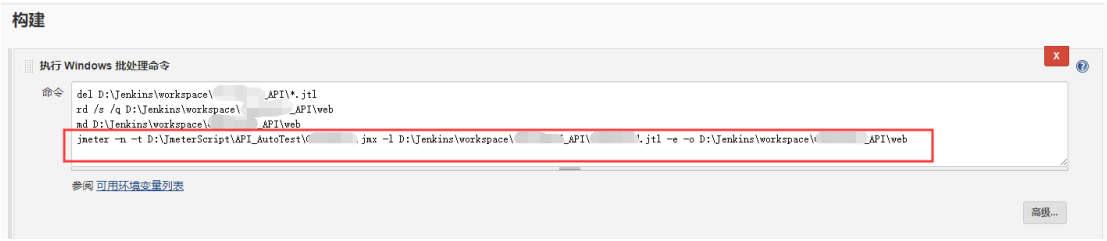

此时我们可以根据 errno 为 2000 进行 beanshell 断言，并同时判断提示信息 errmsg 内容是否正确。断言方式需要灵活选择，尽量直接使用 jmeter 提供的断言方法，减少开发成本，最后选择 beanshell 断言。接口测试最重要的部分就是断言，一个没有断言的自动化接口测试脚本，其测试结果毫无意义。

**五、集成到 jenkins**

Jmeter 图形化页面进行脚本编写，调试，相当方便。为了提高运行速度，建议使用命令行模式运行脚本。

Jenkins 中，创建一个自由风格的软件项目。


若是需要远程节点运行，在 general 中勾选“限制项目的运行节点”，如何添加远程节点，已经在本系列的第一篇中详细介绍过了。我这里的 jmeter 的脚本，都是基于图形化界面开发的，所以不需要进行源码管理。定时每周 1 到周 6，中午 12 点 30 分运行一次。


当前运行环境是 windows 环境，所以用的是批处理命令。若是 linux 就使用 shell 命令。Jmeter 的命令行模式运行的一些命令，可以在官方网站上查看介绍，具体网站是：https://jmeter.apache.org 英语不好的，可以使用 google 浏览器，直接可以将英文网站翻译成中文网站

## 1.4 Running JMeter

To run JMeter, run the **jmeter.bat** (for Windows) or **jmeter** (for Unix) file. These files are found in the **bin** directory. After a short time, the JMeter GUI should appear.

> GUI mode should only be used for creating the test script, CLI mode (NON GUI) must be used for load testing

There are some additional scripts in the **bin** directory that you may find useful. Windows script files (the .CMD files require Win2K or later):

**jmeter.bat**
run JMeter (in GUI mode by default)
**jmeterw.cmd**
run JMeter without the windows shell console (in GUI mode by default)
**jmeter-n.cmd**
drop a JMX file on this to run a CLI mode test
**jmeter-n-r.cmd**
drop a JMX file on this to run a CLI mode test remotely
**jmeter-t.cmd**
drop a JMX file on this to load it in GUI mode
**jmeter-server.bat**
start JMeter in server mode
**mirror-server.cmd**
runs the JMeter Mirror Server in CLI mode
**shutdown.cmd**
Run the Shutdown client to stop a CLI mode instance gracefully
**stoptest.cmd**
Run the Shutdown client to stop a CLI mode instance abruptly

> The special name **LAST** can be used with **jmeter-n.cmd**, **jmeter-t.cmd** and **jmeter-n-r.cmd** and means the last test plan that was run interactively.

There are a few environment variables, that can be used to customize the JVM settings for JMeter. An easy way to set those is by creating a file named **setenv.bat** in the **bin** directory. Such a file could look like:

```
rem This is the content of bin\setenv.bat,
rem it will be called by bin\jmeter.bat

set JVM_ARGS="-Xms1024m -Xmx1024m -Dpropname=value"
```

The **JVM_ARGS** can be used to override JVM settings in the **jmeter.bat** script and will get set when starting JMeter, e.g.:

```
jmeter -t test.jmx …
```

放开样式权限（系列二中已经介绍过）

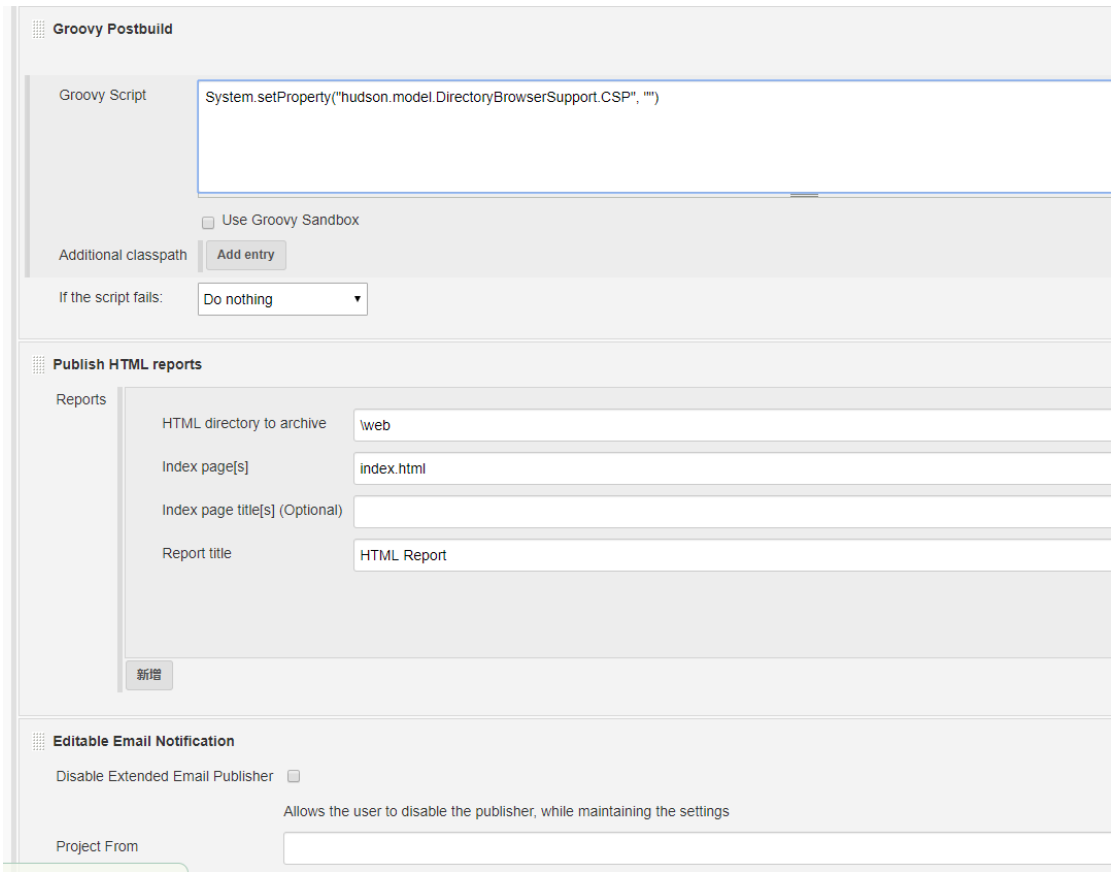
Groovy Script: System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "")

设置在线查阅测试报告（系列二中已经介绍过）

Publish HTML reports

将测试结果发送到指定邮箱（系列二中已经介绍过）

Editable Email Notification

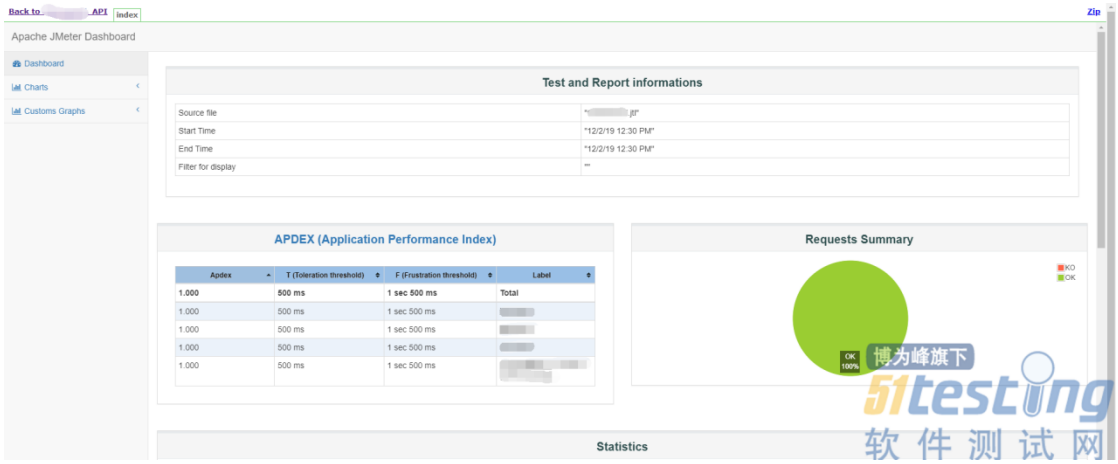


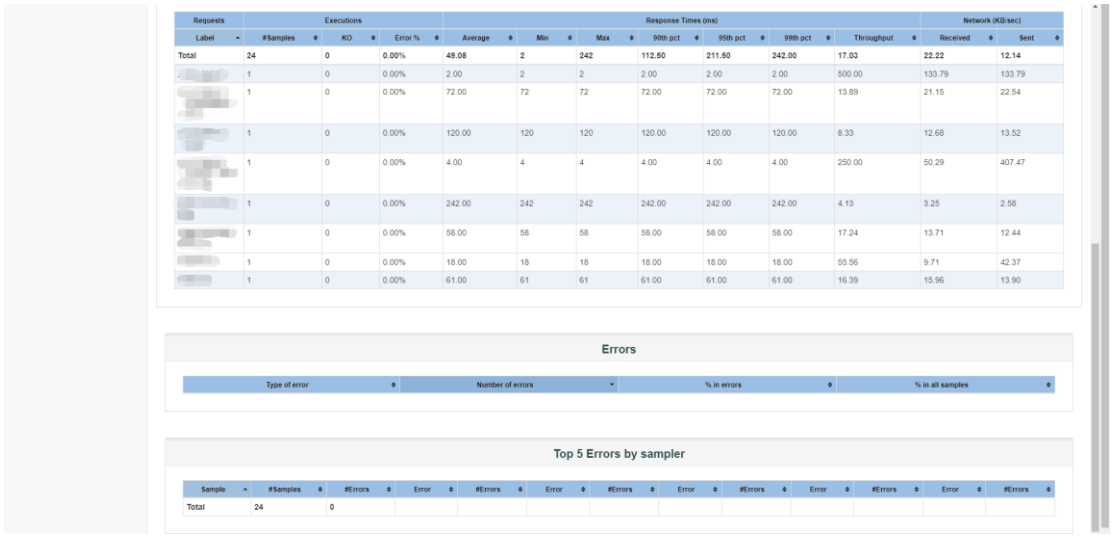

创建完成后，保存。构建失败可以在控制台中查看失败原因，重新调试程序。



构建成功可以在点击 html report 在线查看运行结果:


Jenkins
Jenkins
_API
返回面板
状态
修改记录
工作空间
立即构建
删除 工程
配置
Email Template Testing
HTML Report
重命名
工程 _API
2019-08-30创建项目，每周1-6中午12点30执行，集成 自动化测试
2019-09-03集成 接口的自动化测试
HTML Report
工作区
最新修改记录
Build History
构建历史
find
#77 2019-12-2 下午12:30
#76 2019-11-30 下午12:30
#75 2019-11-29 下午12:30
相关链接
最近一次构建(#77),3 小时 28 分之前
最近稳定构建(#77),3 小时 28 分之前
最近成功的构建(#77),3 小时 28 分之前
最近完成的构建(#77),3 小时 28 分之前


Apache JMeter Dashboard
Dashboard
Charts
Customs Graphs
Test and Report informations
Source file
Start Time
"12/2/19 12:30 PM"
End Time
"12/2/19 12:30 PM"
Filter for display
APDEX (Application Performance Index)
Apdex
T (Toleration threshold)
F (Frustration threshold)
Label
1.000
500 ms
1 sec 500 ms
Total
Requests Summary
Statistics


Requests
Executions
Response Times (ms)
Network (KB/sec)
Label
#Samples
KO
Error %
Average
Min
Max
90th pct
95th pct
99th pct
Throughput
Received
Sent
Total
Errors
Type of error
Number of errors
% in errors
% in all samples
Top 5 Errors by sampler
Sample
#Errors
Error

# 机器学习之 KNN 算法，好算法有迹可循！

◆ 作者：咖啡猫

**题记：**众所周知，机器学习中有很多算法，像前面几期文章中介绍的逻辑回归算法、PageRank 算法、K-means 聚类算法等等。其实，每一种算法都有自己的优点和不足，都有自己特定的使用场景。这就要求我们在具体应用时，学会选择和比较，一般来说，我们会比较算法的时间复杂度、运行效率和正确率等因素，在某一特定的场景里，选择运行效率高且正确率高的算法。

言归正传，本文咖啡猫将带大家了解 KNN 算法的原理及 Python 的实现过程，希望小伙伴们能够掌握它。

## 一、什么是 KNN 算法

KNN 算法的全称是 K 最近邻算法（即：K-NearestNeighbor）。它是机器学习数据挖掘分类技术中最简单的分类算法之一，它的核心思想是：每个样本的类别取决于它最近的 K 个邻居的类别。

如图 1 所示，要想知道绿色点属于哪一类（红色三角类或蓝色方框类），我们可以选取离绿色点最近的 K 个点，统计这 K 个点中大部分点属于哪一类，那么这个绿色点就属于哪一类。如图当 K 取 3 时，绿色点属于红色三角类，当 K 取 5 时，绿色点属于蓝色方块类。

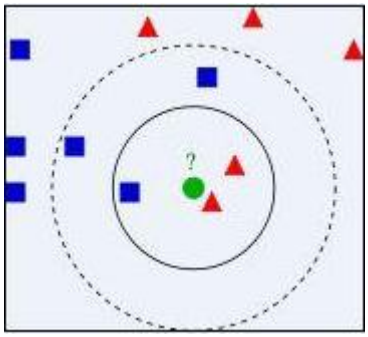



图 1

## 二、KNN 的算法流程

1) 计算测试数据与各个训练数据的距离。

2) 对训练数据按距离从小到大排序

3) 取排序中前 K 个点

4) 统计这 K 个点所在的类别（即统计每一种类别的元素个数）

5) 以元素最多的类别作为测试数据所属类别

**关于 K 值的选取：** 由图 1 中的例子可以知道，K 值的选取对分类结果的影响很大：k 过小，则受噪声的影响大；k 过大，则利用较大邻域中的数据进行分类预测，算法时间复杂度就会上升。在实际应用中，常用的方法就是 K 从 1 开始取到 N，统计不同 K 值下模型的正确率，最后选择正确率最高的 K 值。另外，K 要尽量取奇数，来保证最后有只有一个类别的元素个数较多。

**关于距离的计算公式：** 常用的距离计算公式有欧几里得距离、余弦值、相关度、曼哈顿距离等。在本文中我们选择欧几里得距离公式，即：

$$E(x,y)=\sqrt{\sum_{i=0}^{n}(x_i-y_i)^2}$$

（其中，n 为样本点的维数）

## 三、KNN 的实现方法

```
#!/usr/bin/env python
# -*-coding: utf-8 -*-
import numpy as np
from math import sqrt
import operator as opt

def maxIndex(list):#得到一个列表最大值的索引，注意索引值+1 才等于 K 值
    max_index = 0
    list_index = 0
    for number in list:
        if number> list[max_index]:
            max_index =list_index
        list_index=list_index+1
```

```
        return max_index


def normData(dataSet): #特征缩放，可以帮助梯度下降更快收敛
        maxVals = dataSet.max(axis=0)#返回矩阵中每列最大值
        minVals = dataSet.min(axis=0)#返回矩阵中每列最大值
        ranges = maxVals －  minVals#得到每列最大值和最小值的差
        retData = (dataSet - minVals) / ranges#归一化
        return retData, ranges, minVals


def kNN(dataSet, labels, testData, k):
        distSquareMat = (dataSet - testData) ** 2 # 计算差值的平方
        distSquareSums = distSquareMat.sum(axis=1) # 求每一行的差值平方和
        distances = distSquareSums ** 0.5 # 开根号，得出每个样本到测试点的距离
        sortedIndices = distances.argsort() # 从小到大排序，得到排序后的索引
        indices = sortedIndices[:k] # 取最小的 k 个
        labelCount = {} # 存储每个 label 的出现次数
        for i in indices:
                label = labels[i]
                labelCount[label] = labelCount.get(label, 0) + 1 # 次数加一
        sortedCount = sorted(labelCount.items(), key=opt.itemgetter(1), reverse=True) # 对 label 出现的次数从大到小进行排序
    #labelCount.items()将字典以元组形式输出  Key 调用每个元组索引为 1 的元素（决定排序对象为次数），
        return sortedCount[0][0] # 返回出现次数最大的 label


if __name__ == "__main__":
        A = np.zeros((900, 3), dtype=float)  # 先创建一个 900x3 的全零方阵 A，并且数据的类型设置为 float 浮点型
        B = []
        f = open('G:/testset.txt')  # 打开数据文件文件
        # lines = f.readlines()  # 把全部数据文件读到一个列表 lines 中
        n = 0
        while n < 900:
                list1 = f.readline().strip('\n').split(' ')
                A[n:] = list1[0:3]  # 把处理后的数据放到方阵 A 中。list[0:3]表示列表的 0,1,2 列数据放到矩阵 A 中的 row 行
                B[n:] = list1[3:4]  # 把列表第 3 列（标签）放到 B 矩阵中
                n = n + 1


        C = np.zeros((100, 3), dtype=float)  # 先创建一个 100x3 的全零方阵 C，并且数据的类型设置为 float 浮点型
```

```
        D = []
        n = 0
        while n < 100:
            list1 = f.readline().strip('\n').split(' ')
            C[n:] = list1[0:3]  # 把处理后的数据放到方阵 A 中。list[0:3]表示列表的 0,1,2 列数据放到矩阵 C 中的 row 行
            D[n:] = list1[3:4]  # 把列表第 3 列（标签）放到 D 矩阵中
            n = n + 1


    normDataSet, ranges, minVals = normData(A) #对训练集 A 做归一化处理 normData 就是归一化函数。
    testData = np.array(C)
    normTestData = (testData - minVals) / ranges #
    correct_rate = []  # 存储不同 K 情况下的正确率，K=1.....100
    for k in range(1,1001):


      n=0
      count =0


      while n < 100:
       test=normTestData[n]


       result = kNN(normDataSet, B, test, k)


       if result == D[n]:#比较预测值与实际值，若相同 count 加 1，便于计算正确率
            count = count +1
       n = n + 1
      #print('当 k 为%d 时，算法正确率为%f'%(k,float(count/100)))
      correct_rate.append(float(count/100))


    a=maxIndex(correct_rate)#求正确率最高的 K
    #print(correct_rate)
print('当 k 为%d 时，得到最大的算法正确率为%f' % (a+1,correct_rate[a]))
```

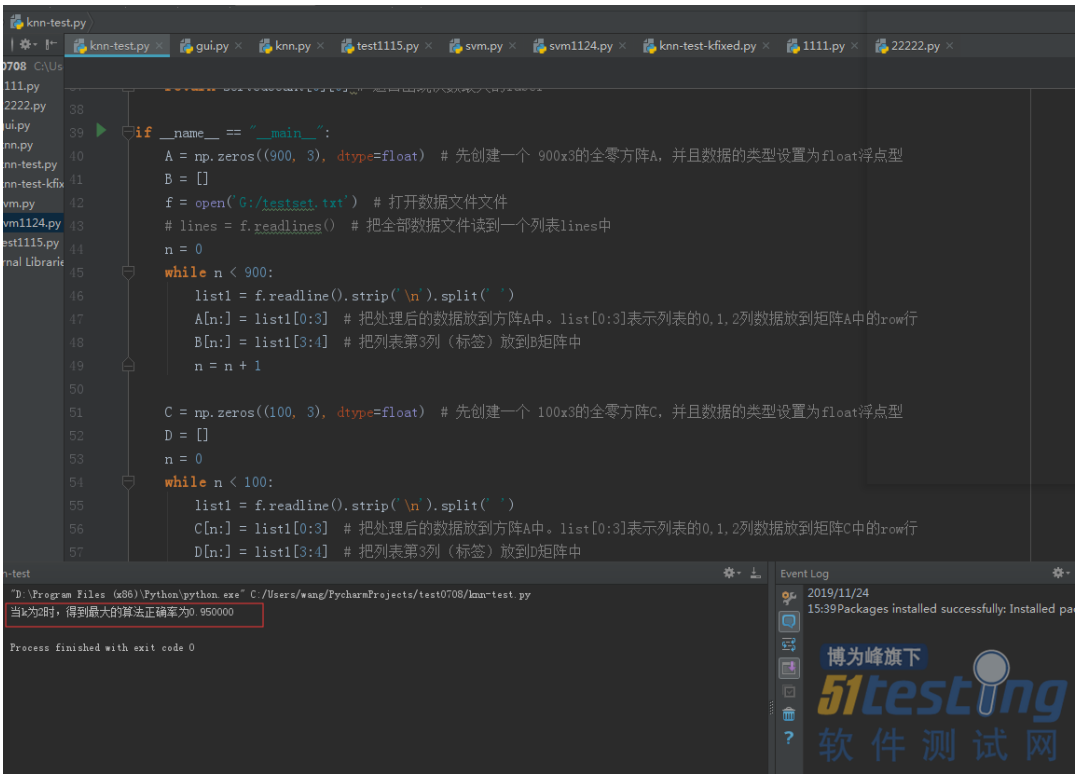
**代码要点说明:**

测试数据集是放在 G 盘一个名为 testset 的 txt 文档中的，该文档有 1000 行 4 列（前三列为样本特征、最后一列为样本分类标签），我将 1-900 行的数据作为训练集，901-1000 行数据做测试集。

在 main 函数中，首先将训练集的样本特征放矩阵 A 中、训练集的标签放矩阵 B 中，然后将测试集的样本特征放矩阵 C 中、测试集的标签放矩阵 D 中，然后对训练集和测试集进行归一化处理来提高结果的准确度，（因为如果某个特征的值域很大，那么距离计算主要取决于该特征，从而与实际情况不符），最后对不同的 K 值运用 KNN 函数得到预测结果并统计正确率，将正确率最大的 K 值打印出来。

核心函数 KNN 就是计算每一个测试样本与训练样本的距离，然后统计距离最近的 K 个训练样本的标签，选择标签中元素最多的那个标签作为测试样本的标签。其中涉及一些 numpy 数组运算的内容以及常规排序操作，需要大家熟悉。

**四、测试结果分析**




由运行结果可知：当 K 取 2 时，算法的正确率高达 95%。此外，你也可以尝试用其他距离公式来构建 KNN，选择正确率最高的距离公式，作为最终的建模参数。

**后记：**

KNN 分类算法就介绍到这里了，可见弄清原理之后算法也不是那么高不可攀的，一定不要有畏难情绪哦！希望小伙伴们能够熟知它的原理并且可以用 PYTHON 来实现它，下次再遇到数据分类任务的时候，直接套用代码稍微修改一下就能解决实际问题啦！你 GET 到了么？

# 全流程自动化测试—业务规则标准化及资产库建设探索

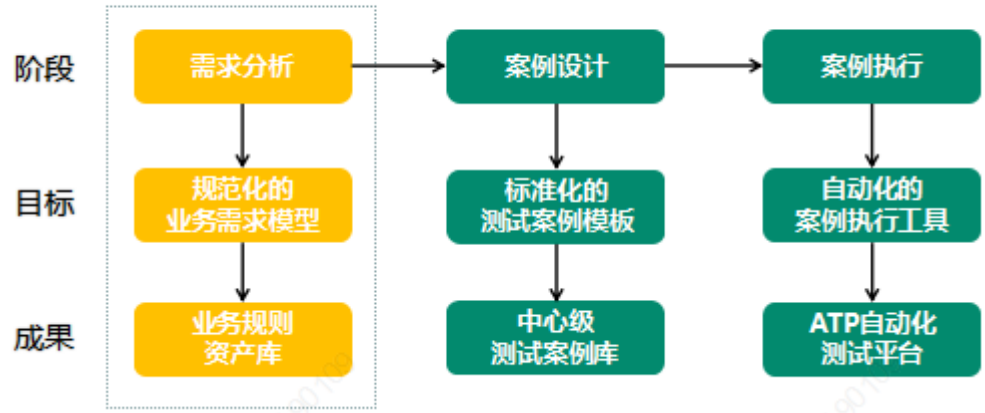
◆作者：于虹

从需求分析开始到案例自动化执行的全流程自动化测试一直是 c 测试人员孜孜不倦的追求。目前，业界已有自动化测试平台，实现测试案例的自动化执行。然而，作为测试工作第一步的需求分析，尚未有成熟的研究成果。

**全流程自动化测试探索**

一般来讲，测试的基本工作流程可分为三个阶段：需求分析、案例设计和案例执行。业务规则标准化研究小组基于案例设计和案例执行阶段已有的研究成果和测试资产，开展了需求分析阶段规范化和标准化的研究，旨在从业务需求层面积累业务规则，并将业务规则对接到高级测试案例的自动生成和执行，实现从需求分析开始，对接案例设计和案例执行的全流程自动化测试，提升整个测试流程的自动化水平，进而提高测试质量和效率，并逐步建设各个业务领域的业务规则资产库。以此来实现包括需求分析、案例设计、案例执行的全流程自动化测试。

测试的基本工作流程



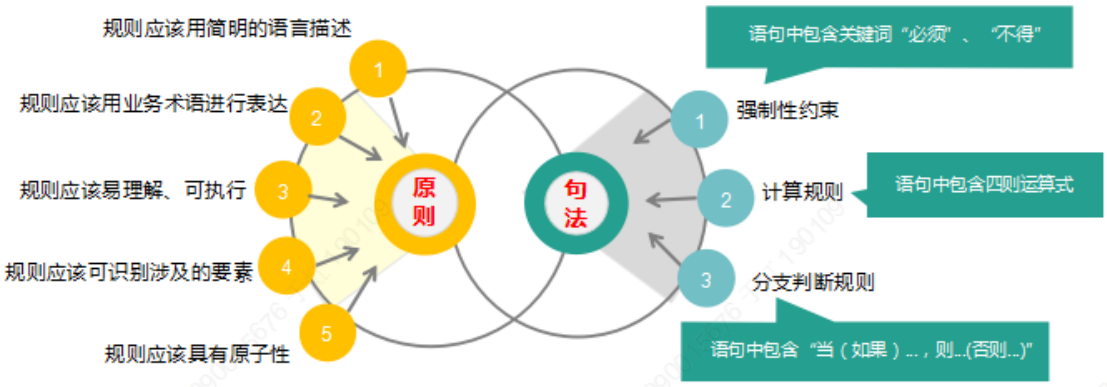
初见雏形

研究小组为了标准化、结构化的诠释和存储业务规则，设计了一种六层结构的业务需求规范化模型，将一项业务规则表达为“系统-模块-流程-功能-要素-规则”的组合。业务规则标准化是一个自上而下逐层分析的过程，总体思路是以应用系统为单位，对该系统业务需求中的各项业务规则按照这六层标准结构进行诠释和展现。


在模型的要素层中存储页面级规则，规则层中存储功能级规则和流程级规则，这三类规则分别对应高级案例中测试类型为“页面”、“功能”、“流程”的案例。研究小组为这三类规则分别制定了表达规范，以支持从业务规则到高级案例的自动生成和执行。

页面级规则约定了界面级功能的输入集合中各个独立对象的合法性。一般通过分析对象类型来确定对象合法性的分析角度。目前高级案例模板中定义了 14 种对象类型，需要从是否必输、是否有默认值、长度、数据类型等方面分析合法性，研究小组从这四个角度分别制定了表达规范，后续还可根据实际测试需要进行扩展。

功能级规则是指在输入要素合法的前提下，描述要素的不同取值组合对整体功能的影响。分析步骤包括：提取业务规则——进行规则描述——识别规则中涉及的要素——配置规则表达式——配置正/反向案例检查点。研究小组提出了提取功能级业务规则的一般方法，进行规则描述的 5 条原则和 3 种句法，配置规则表达式和正/反向案例检查点的表达规范，以及进行功能级规则维护的各项原则。




流程是由一系列相互关联或相互作用的功能串联起来实现将输入转化为输出的活动。在功能正确实现的前提下，不同的功能组合导致了流程处理的差异。分析步骤包括：提取业务规则——进行规则描述——识别规则中涉及的要素及取值——配置流程中各功能的取值——配置正/反向案例检查点。研究小组提出了提取流程级业务规则的一般方法，流程级规则的描述规范，配置流程中各功能的取值和正/反向案例检查点的表达规范，以及进行流程级规则维护的各项原则。

通过进行上述多维度业务规则的标准化，不仅可实现从需求分析开始，对接案例设计和案例执行的全流程自动化测试，还能逐步积累业务规则，开展各领域的业务规则资产库建设。

**未来可期**

基于业务规则标准化及资产库建设探索不仅仅单纯的解决目前手工编制测试案例的问题，提升整个测试流程的自动化水平，同时，实现各领域内业务规则的积累，将测试人员的经验固化为组织资产，防止因人员流失造成的知识流失。今后，测试工作将只需进行业务需求分析，案例的设计和执行都将自动化完成，实现一种全新的全流程自动化测试模式。

# JMeter 循环读取 CSV 文件实现接口批量测试

◆ 作者：朱思衡

首先要理解为什么要进行批量测试，当我们在工作中进行接口测试时，项目的接口肯定不止一个，而是很多很多，而且每个接口都需要进行正确参数，错误参数，参数为空，特殊字符等方式来测试接口是否能够正确返回所需的响应值。

在编写脚本之前可以先汉化 JMeter，找到 JMeter 解压/安装路径下的 JMeter.properties，用编辑器打开（右键用记事本打开也可以）：找到 #language=en ，在下面添加：language=zh_CN 并重启 JMeter

那么我们该如何通过 JMeter 来完成批量测试呢？ 我们先建立一个较为简单的方式进行测试脚本观察一下。

以去获取创建订单时的【省/市/区信息】接口：api.test.XXXXXX.com/api/XXXX/XXXX/getAddress 请求方法：post 请求参数：type，code 为例

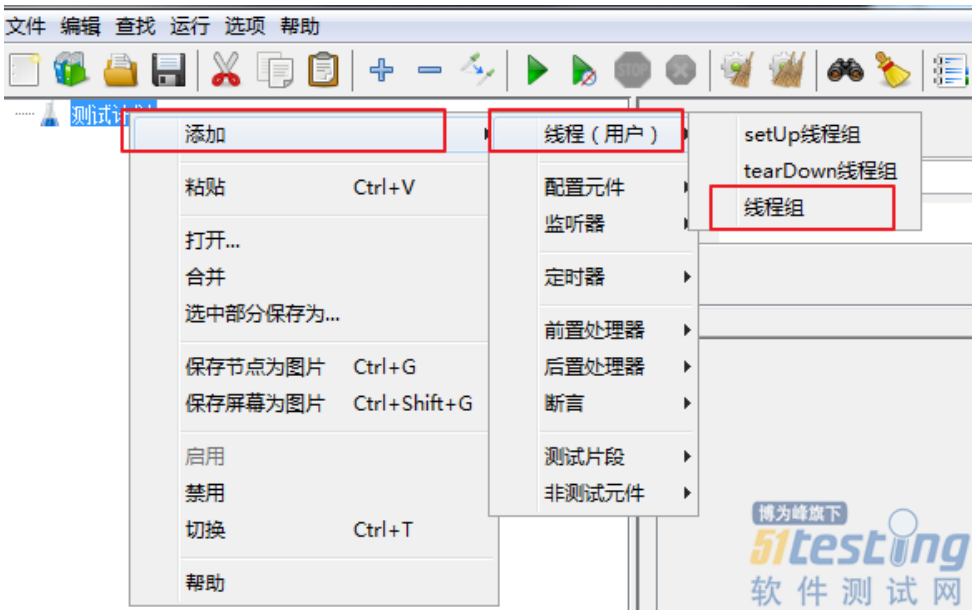
**一、添加线程组**

测试计划右键选择添加——线程用户——线程组。

Setup 线程组：用于执行预测试操作

tearDown：用于执行测试后操作

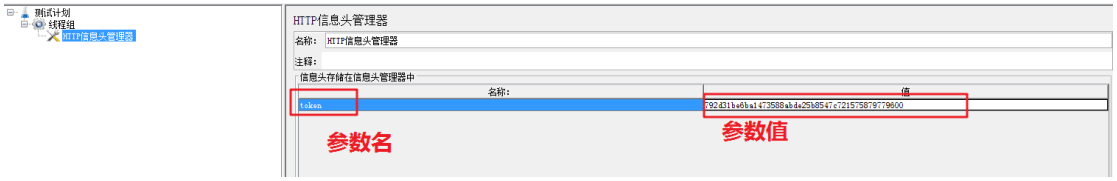
线程组：一般测试接口时添加的线程，可以把它看做一个虚拟的用户组，线程组中的每一个线程都可以理解为一个虚拟用户




## 二、添加 HTTP 信息头管理器

线程组右键选择添加——配置元件——HTTP 信息头管理器

如果接口信息头定义了值的话，需要放到信息头管理器里面，例如：用户 token，公共参数等。




## 三、添加 HTTP 请求

线程组右键添加——取样器——HTTP 请求

协议：默认 HTTP 协议，可不填写，若为 HTTPS 协议需要填写 HTTPS。

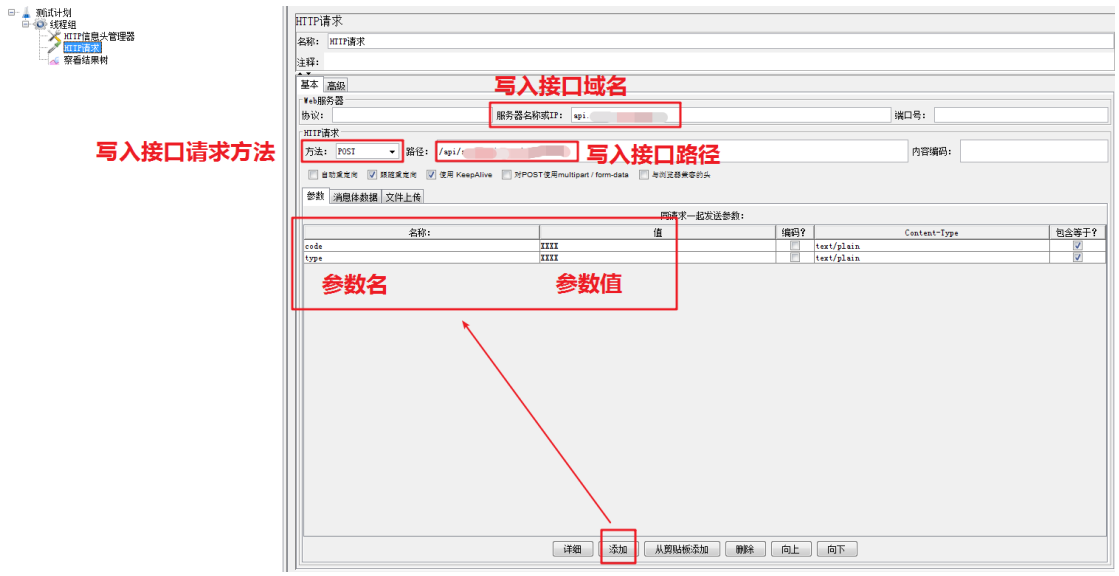
服务器名称或 IP：填写接口域名

端口号：默认 HTTPS 请求的 80，若为 HTTPS 请求需要填写 443

方法：直接选择接口的请求方法：GET,POST,PUT,DELETE 即可。

路径：填写接口的具体路径

参数：点击下方导航栏添加按钮，在已经添加的数据栏填写参数名，参数值。

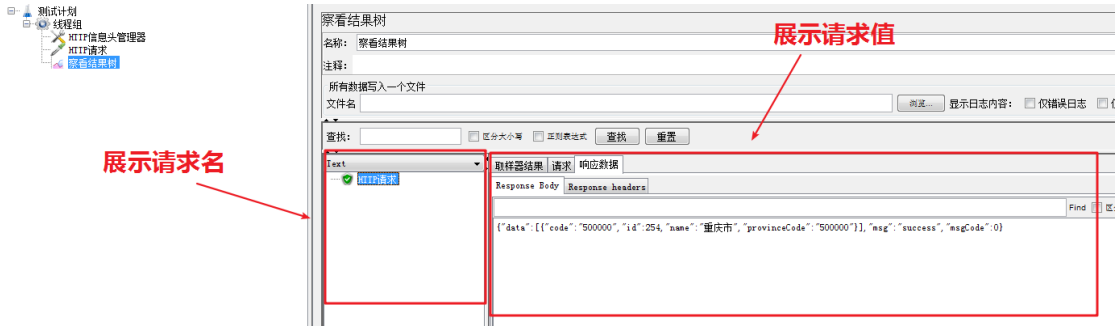



## 四、添加察看结果树

线程组右键添加——监听器——察看结果树

左边：展示请求名，绿色请求结果表示请求成功，红色请求结果表示请求失败。

右边：展示请求值，包括取样器结果，请求头，请求体，响应头，响应体。




以上步骤能够确保接口正确的进行访问并返回对应的返回值。

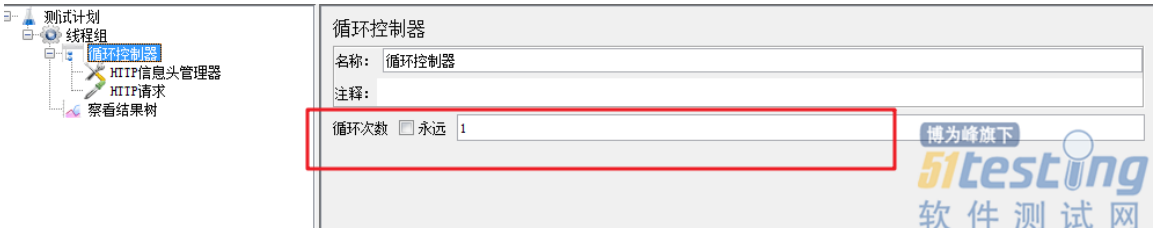
完成以上步骤后，下一步就是要把上面步骤的所有的值以参数的形式传递给 JMeter 脚本，使我们能够以参数化的形式对接口进行批量测试。

## 五、添加循环控制器

线程组右键添加——逻辑控制器——循环控制器，循环控制器添加后需要把【HTTP 信息头管理器】【HTTP 请求】移到【循环控制器下】

循环次数：运行脚本后循环控制器下的 HTTP 请求将执行的次数。例如：循环控制

器的计数为 1，则发送 HTTP 请求 1 次；计数为 5，则发送 HTTP 请求 5 次。设置为永远则需要手动停止请求，否则不会停止请求。




## 六、在循环控制器下添加 CSV 配置文件

循环控制器右键添加——配置元件——CSV 数据文件设置，CSV 文件里面定义接口所需要动态取值的参数，例如：URL，路径，请求方法，接口参数等。

a）如何创建 CSV 文件？电脑桌面新建 TXT 文档，更名为：测试参数文档.CSV（TXT 后缀需要变更为 CSV），打开 CSV 文件新增数据。

b）文件内参数如何书写？第一行定义接口所需动态参数，自第二行起填写接口详细值。

- caseSeq：定义接口序号
- apiSeq：每个接口序列
- apiName：接口名称
- url：接口域名
- api：接口地址
- function：接口请求方法
- purpose：描述接口验证类型
- parameter：接口所需参数拼接，以：“code=500000&type=city”的形式，参数间以“&”连接

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | caseSeq | apiSeq | apiName | url | api | function | purpose | parameter | | | | | |
| 2 | C1 | 1.1 | 获取省/市/区信息 | api.test.duias | /api/starkid/order/gPOST | | 参数正确 | code=500000&type=city | | | | | |
| 3 | C1 | 1.2 | 获取省/市/区信息 | api.test.duias | /api/starkid/order/gPOST | | 参数错误 | type=xxx | | | | | |
| 4 | C1 | 1.3 | 获取省/市/区信息 | api.test.duias | api/starkid/order/gePOST | | 参数为空 | type= | | | | | |
| 5 | C2 | 2.1 | 获取用户信息 | api.test.duias | /api/starkid/getUserPOST | | 参数正确 | userId=2251 | | | | | |
| 6 | C2 | 2.2 | 获取用户信息 | api.test.duias | /api/starkid/getUserPOST | | 参数为空 | userId= | | | | | |
| 7 | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | |

接口所需动态参数

c）csv 数据文件如何设置

文件名：点击浏览后选择本地的 CSV 文件

文件编码：一般情况下无需设置

变量名称：“caseSeq，apiSeq，apiName，url，api，function，purpose，parameter”，参数之间需要用英文的逗号隔开，注意参数不要写错。

忽略首行：True 则不执行 CSV 文件的首行，False 则执行 CSV 文件首行。

分隔符：英文逗号，无需变更

是否允许带引号：设置 True 和 False 以当前 CSV 文件都能正常执行

遇到文件结束符再次循环：True 则在设置的循环次数内一直循环，False 则 CSV 文件所有数据循环一次后就停止。

遇到文件结束符停止线程：True 则在设置的循环次数内执行到 CSV 文件数据为空时停止发送 HTTP 请求，False 则在设置的循环次数内执行到 CSV 文件数据为空时继续执行。

线程共享模式：选择所有现场即可

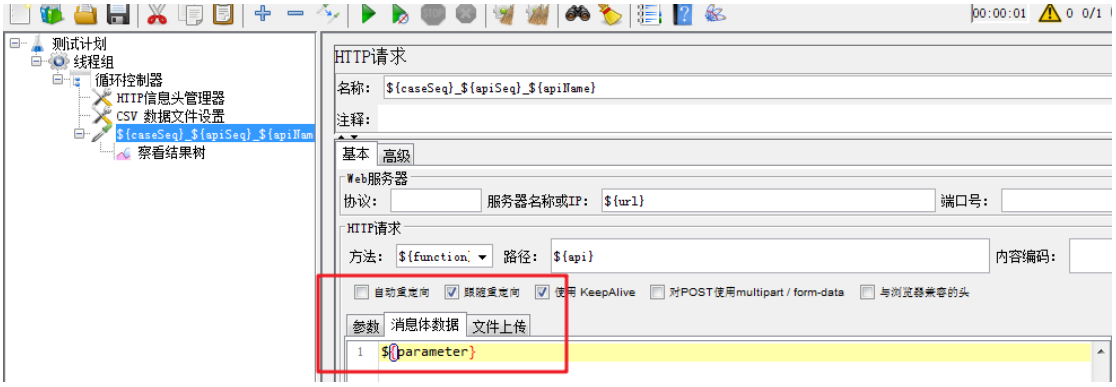
综上：本次 CSV 文件设置时：循环次数“永远”，忽略首行“True”，是否允许带引号“False”，遇到文件结束符再次循环“False”，遇到文件结束符停止线程“True”，线程共享模式“所有现场”


## 七、其他配置文件需要修改写入形式

HTTP 信息头管理器：除必要参数外，添加：“Content-Type=application/x-www-

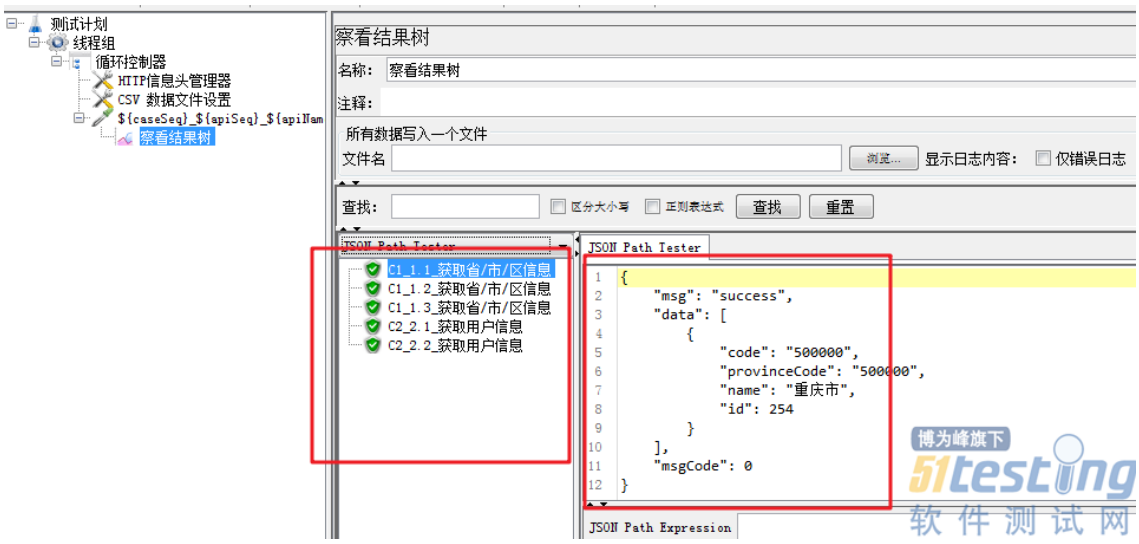
form-urlencoded”，使用 Json 参数需要用到

HTTP 请求：名称设置为：“${caseSeq}_${apiSeq}_${apiName}”动态取 CSV 文件里面配置的参数信息；IP：“${url}”；方法：“${function}”；路径：“${api}”，消息体数据：“${parameter}”消息体数据和参数只能设置一种，不能两种同时设置。




## 八、执行 CSV 文件，查看结果

可以看到左边能够很清晰的反应接口执行的信息，右边可以很明确的反应接口返回的数据。并且 CSV 文件内可以设置众多接口的不同验证方式，很方便且很适合日常工作使用。希望以上内容能给大家提供帮助，谢谢。




■对企业级项目进行接口测试，到底该做什么>>https://dwz.cn/SoysVOny

# 新人应该如何进入测试领域？

◆ 作者：黎晓萌

前些天有读者在后台问道，转行人员应该如何进入测试领域？

今天，我就这个问题做一个完整的回答，顺便说一下进入软件测试这个行业后持续努力的方向，供各位想转行做测试还没转的、已经转行做测试但时间不久的、刚刚进入测试领域的，这几类的小伙伴都可以参考下。对于其中有疑问的，也欢迎持续探讨。

## 一、软件测试的门槛

软件测试，作为软件研发流程中的一环，是软件研发质量的重要保证，而随着互联网产业、软件等需求持续增加，这样的人才需求也持续旺盛，因此，有越来越多的人看准这个机会想进入这个行业。一方面，这一行作为信息基础行业，有着相比于其他行业更高的薪酬待遇；另一方面，随着信息技术更多的进入我们的生活，这方面的发展前景也得到了大家的认可。所以，这一切也都不难理解。

那么，想从事这一行业，都需要做哪些准备呢？在我看来，**首先要了解的就是软件测试的基本概念和思想。**软件测试在软件研发流程中处于怎样的过程，软件研发的基本原理，软件测试的基本方法，这些都是需要了解和掌握的。只有清楚了这些，才能很好的认识到软件测试是怎么一回事，才能够更好的参与到软件测试工作中，进而为软件的研发质量保证贡献出自己的一份力量。

软件测试的基本概念和思想，在这里我就不展开讲了。我想讲的是，在我多年的测试管理工作中，面试了不少候选人，有很多候选人对软件测试的概念和方法都不清楚，觉得软件测试就是将需求文档中的内容点点点，确定都实现了就算测试完成，这个回答是很糟糕的。

作为软件测试行业的从业人员，我们必须清楚，**软件测试是需要用基本的测试方法来保障软件的质量的。**还有很多面试者，谈起测试的基本方法来是头头是道，但真的给了一个需求文档让他去设计测试用例时，就完全不管测试方法中交给我们的东西了，而是天马行空的发散起来。这种理论与实际脱钩的情况，就会让面试官觉得还是没有很好的掌握测试方法，经过他测试的软件产品的质量是不可控的，因此也会存在很大的质量风险。

只有将软件测试的基本概念、方法进行很好的应用到日常工作中，只有能将我们在候选人简历中经常看到的「熟练掌握软件测试方法」转换成真正的掌握，那在我看来才是迈过了从事软件测试行业最基本的门槛。而要达到这一点，**日常工作中积极的思考测试方法与被测试软件的测试用例的对照关系，在日常使用软件的过程中培养测试用例设计的意识，**都能够很好的提高自己在这方面的能力。

**二、测试的基本技能**

在迈过软件测试的门槛后，接下来就是要掌握做测试的基本技能了。这个技能，包括对**被测试软件的进一步了解和掌握，对测试工作中使用的测试工具的熟悉。**

在面试中，我们看到候选人的简历中提到自己测试项目的基本介绍，包括使用的语言、框架，但真正问起来，对这些都没有基本的了解，那在我们看来就是不合适的。还有候选人对自己测试的项目所在平台的一些基本常识的缺乏，在面试中也是很致命的。比如，在面试中我经历过这样的候选人，做的是移动端的 APP 测试，但不清楚 APP 测试的基本特点，需要关注哪些方面，兼容性测试怎么做等，这都是日常测试中很基础的东西，如果都不能很好的讲出来，那就会说很糟糕的。

在我们日常的软件测试中，首先要关注的就是**被测试软件所在平台的基本特点，**是 Web 的还是 APP？Web 类的产品有哪些特点，实现的原理是怎样的，测试过程中需要考虑哪些东西，承载 Web 类产品的浏览器是怎样的工作原理，做兼容性测试的时候需要关注哪些方面。APP 类的产品有哪些特点，所在的系统有哪些特点，与终端系统有哪些交互，需要考虑哪些方面的指标等。再此基础上，**再去结合被测试软件的需求和功能特性去进行测试用例的设计，**功能特性有哪些，边界值法怎么用，等价类怎么划分，异常场景有哪些，等等。

另外，我们在很多候选人的简历中看到「熟练使用 XXXX 测试工具」，但真的问起

来，这个工具的原理是怎么样的，都有哪些基本操作，可以辅助我们做哪些测试等，他们就不知道了。更有甚者，只是知道点哪个按钮可以产生什么东西，多问一句产生的这些东西是怎么得到的，都包括哪些信息，就不知道了。这些就比较不应该了。这样的「熟练」更多的是对最常规、最基本使用的「重复性熟练」，而非真的对这个测试工具有更深层次的了解和掌握。

在我看来，使用一个测试工具，这个**工具实现的基本原理**是怎样的，这个工具能做什么，不能做什么，为什么能做这些，使用过程中能给我们提供哪些数据，我们可以用这些数据做什么，如何辅助我们进行测试，都是我们在拿到一个工具时首先可以去了解的。只有对一个工具有了这些了解，我们才能在日常的测试工作中更好的、更熟练的使用，最大化的辅助我们的测试工作。

**三、测试的能力提升**

当我们对基本技能有了一定的掌握后，我们就需要进一步的提升自己的软件测试能力。

这时候对测试能力的定义和范围就可以进一步的扩大化，当我们测试一个软件时，这个软件更底层的东西就需要我们去开始接触，使用的语言、框架的特点，软件的技术架构等。这些对我们更好的了解被测试软件，发现其中存在的问题，都有很大的帮助。

对于较大型的软件，我们还需要了解整个系统的模块划分是怎样的，模块与模块间的调用关系是怎样的，调用过程中是否困难存在异常，这里的用例该如何设计，对错误处理的用例该怎么构造，软件发布上线后，对应的数据是否有上报，上报的问题反应了怎样的问题，等等，都是我们需要去了解的。

在测试工具的使用上，也对我们有更高的要求，比如能实现类似功能的同类工具还有哪些，为什么我们要用当前工具，这个工具能得到哪些信息，这些数据是怎么得到的，准确率是多少，误差是否在我们可接受范围内，我们是否需要再此基础上去做优化，如此等等。

另外，随着我们对软件测试的要求越来越高，测试的粒度越来越细，一些基本的测试、常规的工具不能满足时，我们是否能够开发出一些脚本、工具来满足我们的测试需要，我们是否能够持续用工具来解决测试中的问题，用工具实现测试效率的提升，测试深度的加强，等等。

**四、测试的职业发展**

最后，简单谈下软件测试这个行业的职业发展。软件测试作为一个技术岗，不断的提升自己的技术能力是最核心的发展要素。在这个大前提的指引下，不断的沉淀自己技术领域的积累，扩充自己在技术解决测试问题上的知识面和能力，是关键。

从这个意义上讲，**软件测试工程师的发展会从解决独立需求测试的初级工程师，向能独立负责较大系统测试的中级工程师，到能完成测试体系的建设的高级工程师**一步步实现前进，在这个过程中，自己的技术能力也从对被测软件的基本实现，到掌握系统架构，针对系统架构进行完整测试，从用小工具实现测试效率的改进，到用测试平台、测试体系实现一整套测试架构的实现。

测试管理也是软件测试工程师发展的一个选项，**从独立跟进需求测试，到带领小团队共同完成某个项目的测试，进而到带领更大的测试团队完成包括功能测试、工具建设等多领域的提升。**

总之，软件测试是一个需要不断理解行业，不断提升自我，不断思考测试对于项目团队意义的工作。只有不断的前进，自己的职业生涯才会有更好的前途和更大的发展。



## 《51 测试天地》（五十六）下篇 精彩预览

- 使用 Locust 进行 APP 服务端并发测试浏览器网络捕获器在 python 网络编程中的辅助作用
- 测试女巫紧跟时代脉搏之大数据分析系列
- JMeter 扩展开发之 JMeter 核心源码解读
- nGrinder 压测工具使用教程
- Postman +Newman+Jenkins+钉钉 实现持续集成的接口自动化
- WEB 自动化测试之元素定位自动生成探索实践
- 安全测试，我有话要说
- 自动化测试实现优劣浅谈

马上阅读