# 目 录 (五十六期·下)

# 使用 Locust 进行 APP 服务端并发测试

◆作者：王 练

摘要：APP 的性能测试，对于服务端的压力其实和 Web 差别不大，通过 JMeter/Loadrunner 进行服务器的并发请求即可完成。

基于 Python 协程的 Locust 是一款优秀的性能测试工具，单机并发量远超于 JMeter 和 Loadrunner。本文通过实例介绍 Locust 的使用，通过解决产品开发中的实际问题作为 Locust 实战的说明，最后结合该实例总结 Locust 的使用方法，为其他需要使用 Locust 的工程师提供思路。

## 一、Locust 简介

Locust 是一款开源的 Web 性能测试框架。该单词的原意是蝗虫，原作者之所以选择这个名字，估计也是听过这么一句俗语，“蝗虫过境，寸草不生”。

Locust 工具生成的并发请求就跟一大群蝗虫一样，对我们的被测系统发起攻击，以此检测系统在高并发压力下是否能正常运转。在 Locust 测试框架中，测试场景是采用纯 Python 脚本进行描述的。对于最常见的 HTTP(S)协议的系统，Locust 采用 Python 的 requests 库作为客户端，使得脚本编写大大简化，富有表现力的同时且极具美感。而对于其它协议类型的系统，Locust 也提供了接口，只要我们能采用 Python 编写对应的请求客户端，就能方便地采用 Locust 实现压力测试。从这个角度来说，Locust 可以用于压测任意类型的系统。

**主要特点如下:**

**1、使用纯粹的 Python 编写脚本。**

用 Locust 就是写 Python 代码，这点比其他性能测试工具要先进。事实上界面点击代替编程，本身就是伪命题。不会编程的用不好 JMeter，会编程的更喜欢写纯粹的代码。

**2、单机可支持千级并发压力，且支持分布式。**

Locust 采用协程并发，单机比 LoadRunner、JMeter 的并发都会高很多。JMeter 一个并发用户一个线程，当用户变多时，本身的性能会急剧下降。

**3、Web 管理界面。**

内嵌的 Web 服务，并且支持扩展。

**4、可以测试任何系统。**

基于协议，通过协议模拟，可尝试任何系统。

**5、简单耐玩。**

Locust 核心代码只有几百行，简单。扩展性强，有很强的二次开发属性，耐玩性很高。

关于 Locust 下载、安装和运行 Demo 测试的方法，可以参考官网的示例，以及网络上其他的信息。下面以一个实际的例子说明如何使用 Locust 进行并发测试。

## 二、并发测试需求说明

需求说明：测试的 APP 是面向 C 端客户的理财类软件，目前上线一个类似红包的空投优惠卷拉新活动，通过红包的发放宣传 APP，现在需要测试并发抢该优惠卷红包，是否存在问题。

需求分析：并发抢红包，在不考虑服务器并发性能的基础上，需求很简单，就是确认在并发的过程中，一个用户只能抢一次，红包不能被多抢。

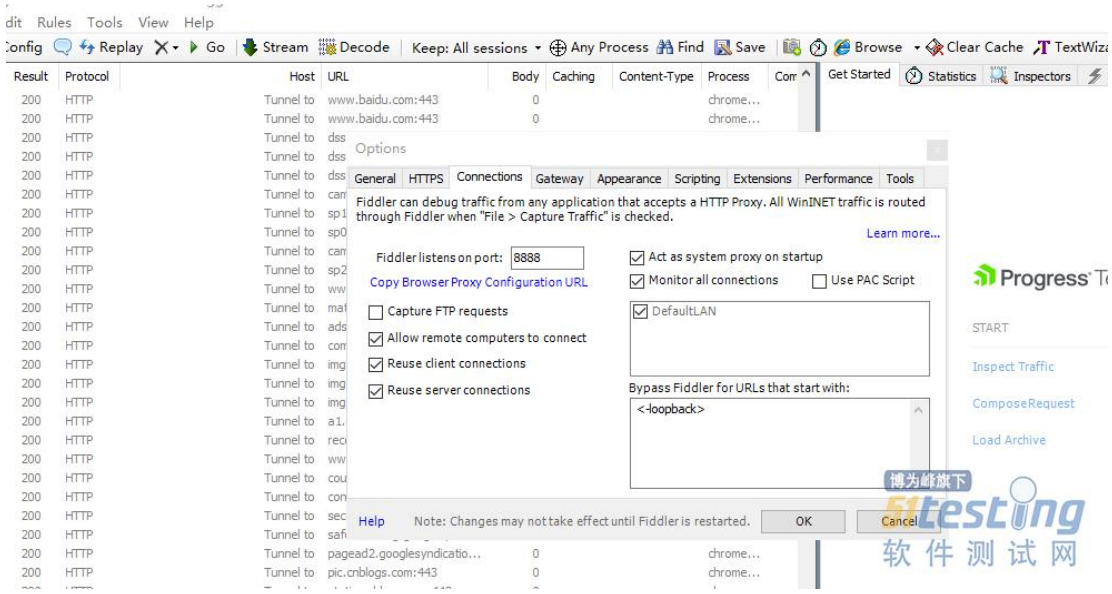
## 三、Locust 脚本开发

### 3.1 抓包确定接口逻辑

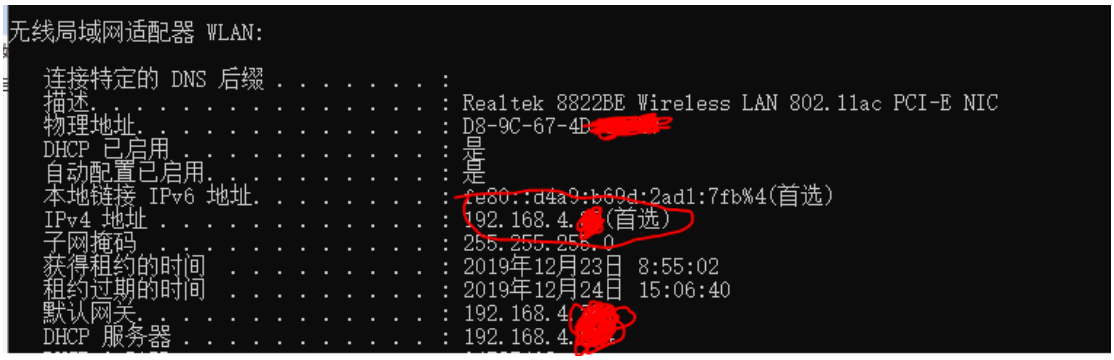
由于 Locust 使用接口进行并发请求发送，所以需要确定需要对哪个接口进行并发，以及如何串联整个场景。

APP 和服务器的抓包可以使用 Fiddler 进行。

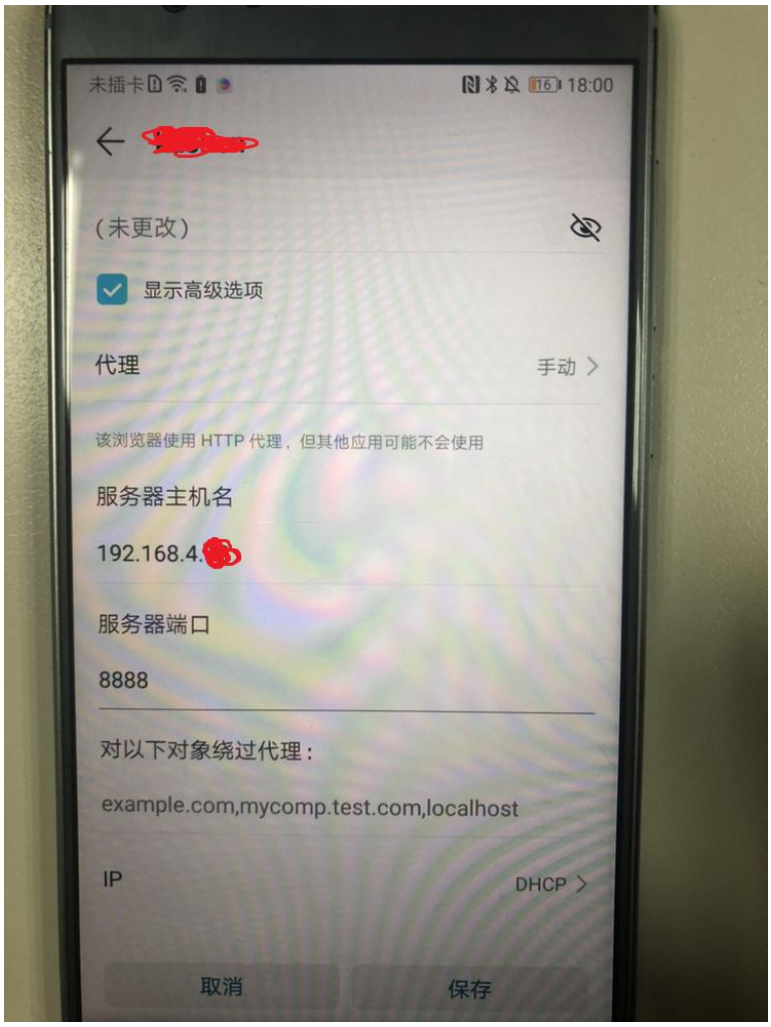
PC 端安装 Fiddler，之后在菜单栏选择->Tools->Options->Connections，设置监听断开，默认为 8888。



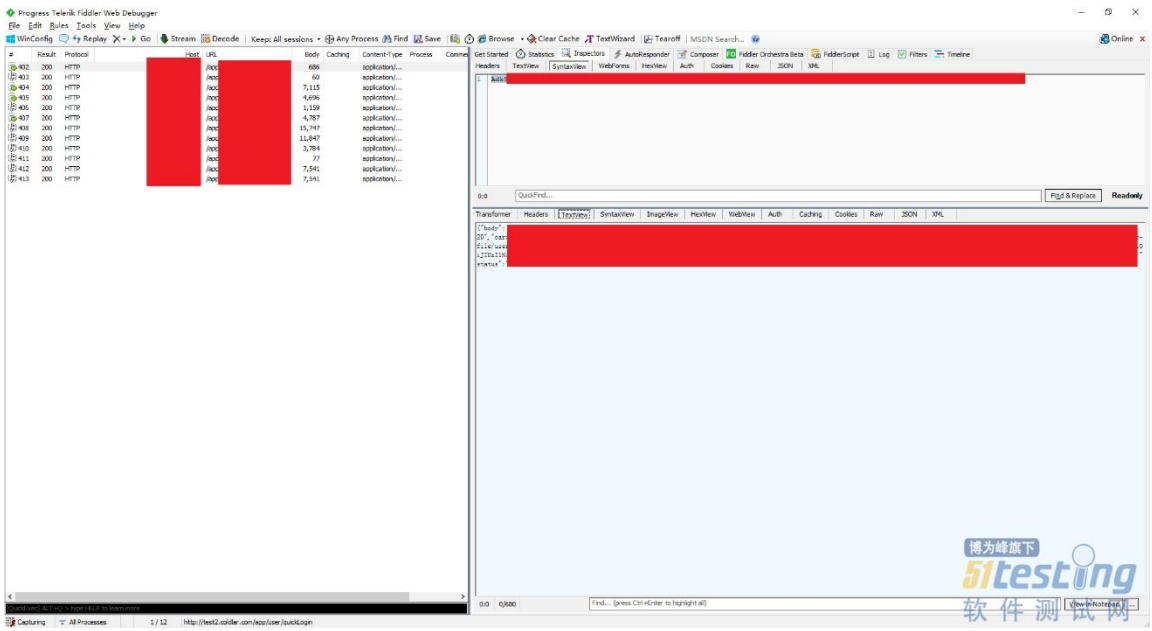

查看 PC 和 APP 连接同一 Wi-Fi 的 IP 地址。




之后将 APP 所在的手机连接同一 Wi-Fi，并设置代理为上述 IP 和端口（默认 8888）。




之后通过 APP 操作的请求就可以在 Fiddler 上抓到包了。



通过抓包可以确定如下流程：

1) 发红包：登录-获取验证码-发红包请求。

2) 抢红包：（1 步获得红包分享链接）抢红包。

上述需要测试并发的就是 2 中抢红包的接口。

### 3.2 编写脚本：获取红包链接

对照 Locust 官网的示例，我们知道 Locust 是通过单纯的 Python 脚本表达并发测试的过程。在一次并发测试中，每个用户就是一个 Locust 的实例，按照既定的流程进行测试。其含义就是指定的如此多的 Locust，在系统中以随机又符合程序设定的方式进行测试。

```
from locust import HttpLocust, TaskSet, task, between

class UserBehavior(TaskSet):
    def on_start(self):
        """ on_start is called when a Locust start before any task is scheduled """
        self.login()

    def on_stop(self):
        """ on_stop is called when the TaskSet is stopping """
        self.logout()

    def login(self):
        self.client.post("/login", {"username":"ellen_key", "password":"education"})

    def logout(self):
        self.client.post("/logout", {"username":"ellen_key", "password":"education"})

    @task(2)
    def index(self):
        self.client.get("/")

    @task(1)
    def profile(self):
        self.client.get("/profile")

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    wait_time = between(5, 9)
```

上述示例表达将要对系统的根目录“/”和“/profile”进行访问，每次访问随机在 5 到 9 秒之间发起，每发起两次根目录访问，发起一次“/profile”访问。on_start 和 on_stop 是对 TaskSet 的初始化和清理，类似 TestNG 中的 BeforeClass 和 AfterClass。（之所以不是 BeforeSuite 和 AfterSuite，是因为 Locust 还有整体的 setup 和 teardown 函数）。

获取红包链接虽然不需要并发，但是通过接口访问的方式能够更快速的获得链接，节省了手工点击的时间。通过对 Locust 示例的改造，实现如下。

```
class WebsiteSequenceTasks(TaskSequence):
    def __init__(self, parent):
        super().__init__(parent)
        self.cl_token = ""  #登录token
        self.cl_send_code = ""  #验证码
        self.aes = AesCbs()   #加解密工具
        self.mysql_conn = MysqlDbOperator("a.b.c.d", 3306, "user", "pwd", "dbname") #数据库连接

    # 登录请求
    def on_start(self):
        login_info = "{\"account\":\"xxxx\",\"password\":\"xxxx\"}"
        #加密登录信息
        login_info_aes = self.aes.encrypt(login_info)
        response_json = self.client.post("/app/xxx/login", login_info_aes)
        #获取登录请求响应的token
        self.cl_token = response_json.json()["body"]["token"]

    # 获取验证码请求
    @seq_task(1)
    def send_code_msg(self):
        code_msg = "{\"account\":\"xxx\"}"
        #加密获取验证码请求
        code_msg_aes = self.aes.encrypt(code_msg)
        response_json = self.client.post("/app/xxx/sendCodeMsg", code_msg_aes)
        #通过数据库获取验证码
        self.cl_send_code = self.get_code_msg_from_db()

    # 获取红包链接请求
    @seq_task(2)
    def add_red_packet(self):
        red_packet_msg = "{\"code\":\"" + self.cl_send_code + "\",\"other\":\"other\"} "
        #加密获取红包请求
        red_packet_msg_aes = self.aes.encrypt(red_packet_msg)
        #头部加入token信息
        header = {'token': self.cl_token}
        response_json = self.client.post("/app/xxx/addRedPaper", red_packet_msg_aes, headers=header)

    def get_code_msg_from_db(self):
        db_str = self.mysql_conn.run_query(
            "SELECT content from xxxtable WHERE account='xxxx' ORDER BY create_time DESC LIMIT 1;")
        return db_str.split(": ")[1].split(",")[0]


class WebsiteUser(HttpLocust):
    task_set = WebsiteSequenceTasks
    host = "http://my.appsys.com"
    wait_time = between(1, 5)
```

在官方示例中，增加了如下变化：

token 获取和使用。App 与后端的交互，在第一次登录后，后续请求都在请求头部增加 token 信息，表明链接的有效性。这里首先在 on_start 请求中通过用户名和密码方式登录接口，之后将返回体中的 token 字段获取，保存在成员变量 cl_token 中。在需要填入 token 的时候，如 add_red_packet 函数中，通过 header 的设定，放入请求中。

加解密的调用。为安全考虑，目前的 App 与后端交互的信息都是加密的，所以实现了加解密的工具类：AesCbs。在使用加解密的地方调用对应函数，如登录接口中，设定登录信息的 JSON 后，调用 AesCbs 的 encrypt 函数完成加密。

数据库的操作。这里是对 MySQL 数据库的操作，请求验证码后，后续操作需要自动完成，不可能查阅手机记录。通过查询数据库的方式达成目的。MysqlDbOperator 类是数据库工具类，完成数据库的连接，执行 SQL 语句等功能，在函数 get_code_msg_from_db 中调用工具类，查询得到结果，由于验证码是一串字符，在其中根据格式截取验证码。

seq_task 的使用。另一个变化是@seq_task 装饰器的使用，官网示例中的@task 装饰器的请求，请求次数与其括号后面的权重有关，但顺序是随机的，如果想要请求按既定顺序执行，就需要使用@seq_task 装饰器。这里实现的就是首先登录（on_start)，之后执行获取验证码（send_code_msg)，最后执行获取红包链接（add_red_packet)。

通过上述实现后，就可以直接访问后端接口，获取红包的分享链接。

### 3.3 编写脚本：并发抢红包

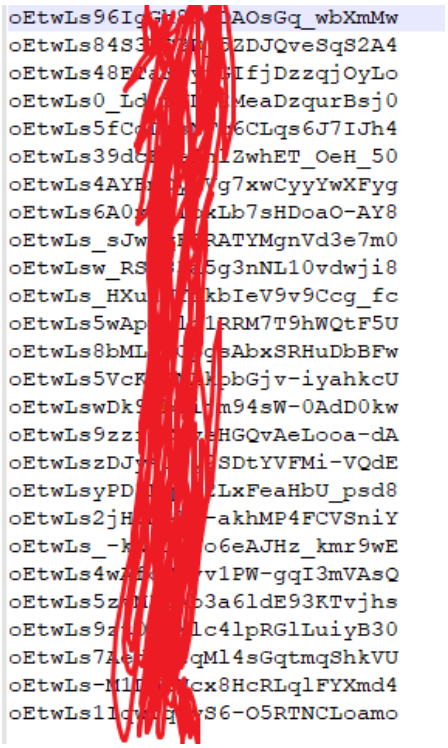
通过上面的讲解，我们直接给出并发抢红包的脚本实现。

```
class WebsiteSecondRecTasks(TaskSequence):
    def __init__(self, parent):
        super().__init__(parent)
        self.aes = AesCbs()

    # 请求抢红包
    @task
    def send_sec_rec(self):
        #获取id数据
        try:
            data = self.locust.user_data_queue.get()
        except queue.Empty:
            print('account data run out, test ended.')
            exit(0)

        code_msg = "{\"shareCode\":\"前面获取的分享链接\",\"id\":\"" + data['id'] + "\"}"
        #加密抢红包请求
        code_msg_aes = self.aes.encrypt(code_msg)
        #设定抢红包请求头
        header = {"Content-Type": "text/plain"}
        #请求抢红包
        response_json = self.client.post("/抢/红/包/接/口", code_msg_aes, headers=header)
        #解密抢红包返回结果
        print("1-二次抢红包请求返回:" + self.aes.decrypt(response_json.text))


class websiteUser(HttpLocust):
    task_set = WebsiteSecondRecTasks
    host = "http://my.appsys.com"
    #获取id文件中的数据
    data_set = load_data_set("./data/id")
    user_data_queue = queue.Queue()
    i = 0
    for index in range(10000):
        open_id = ""
        if index < len(data_set):
            i = index
        else:
            i = 0
        data = {
            "id": "".join(data_set[i])
        }
        user_data_queue.put_nowait(data)
        i = i + 1
    wait_time = between(1, 5)
```

对比之前的脚本，这里增加了一个变化：在数据文件中获取 id 变量，作为每个 Locust 实例中的请求参数。

在 websetUser 类中（该类为 HttpLocust 的子类，所有 Locust 请求都是该类的子

类），通过访问数据文件“./data/id”获取 id 的数据，该文件格式如下：




脚本中循环读取内容，放入队列中共后续使用。在抢红包请求 send_sec_rec 函数中，读取队列中的数据，为该用户抢红包。

### 3.4 发起测试

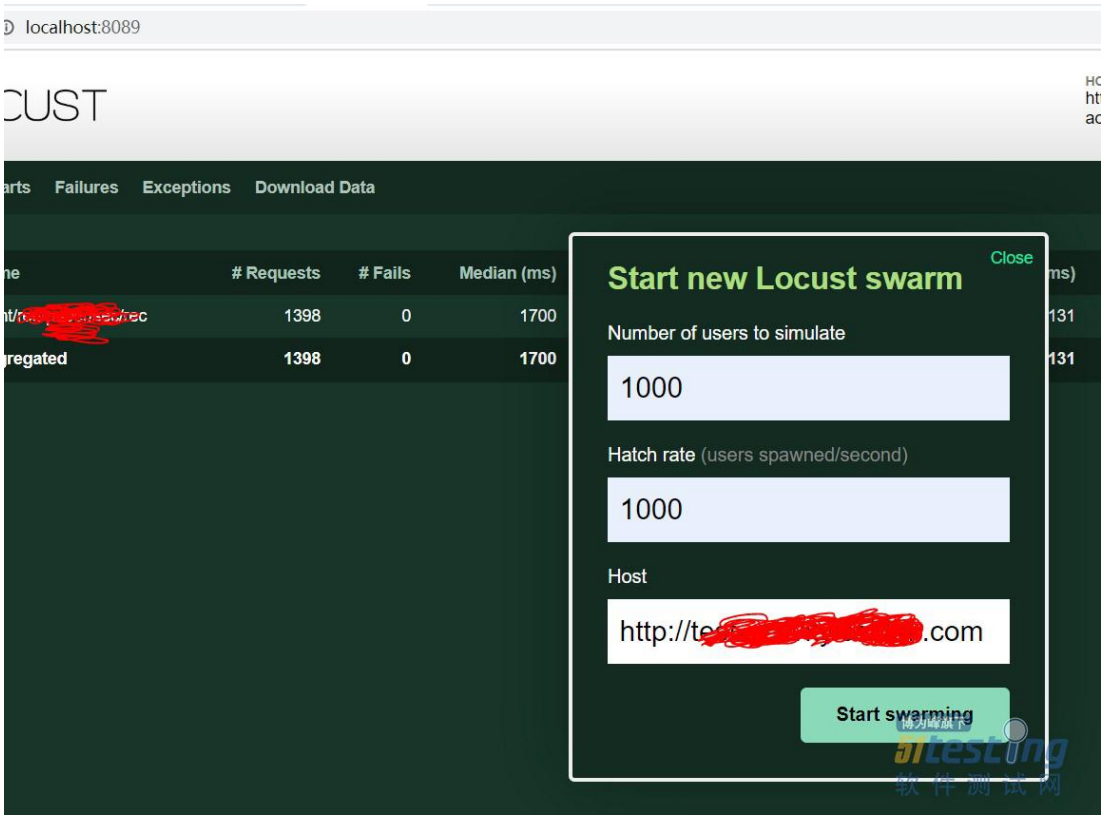
通过如下命令，获取红包分享链接。

```
F:\Python\project\Locust_Project\Locust_Perf_Testing>locust -f ./locustfiles/send_phone_code.py  --port=7089 --host=http//t
[2019-12-23 21:23:46,857] DESKTOP-WLD/INFO/locust.main: Starting web monitor at *:7089
[2019-12-23 21:23:46,858] DESKTOP-WLD/INFO/locust.main: Starting Locust 0.13.2
```

获取后设定抢红包的链接为该分享链接，之后运行抢红包的 Locust 文件。

```
F:\Python\project\Locust_Project\Locust_Perf_Testing>locust -f ./locustfiles/second_rec.py --port=8089 --host=http://test
[2019-12-24 09:29:27,412] DESKTOP-WLD/INFO/locust.main: Starting web monitor at *:8089
[2019-12-24 09:29:27,413] DESKTOP-WLD/INFO/locust.main: Starting Locust 0.13.2
```

在 Web 管理界面中设定并发数目。




完成并发的测试，查看打印日志、后台流水等确定并发抢红包的结果。

## 四、Locust 使用总结

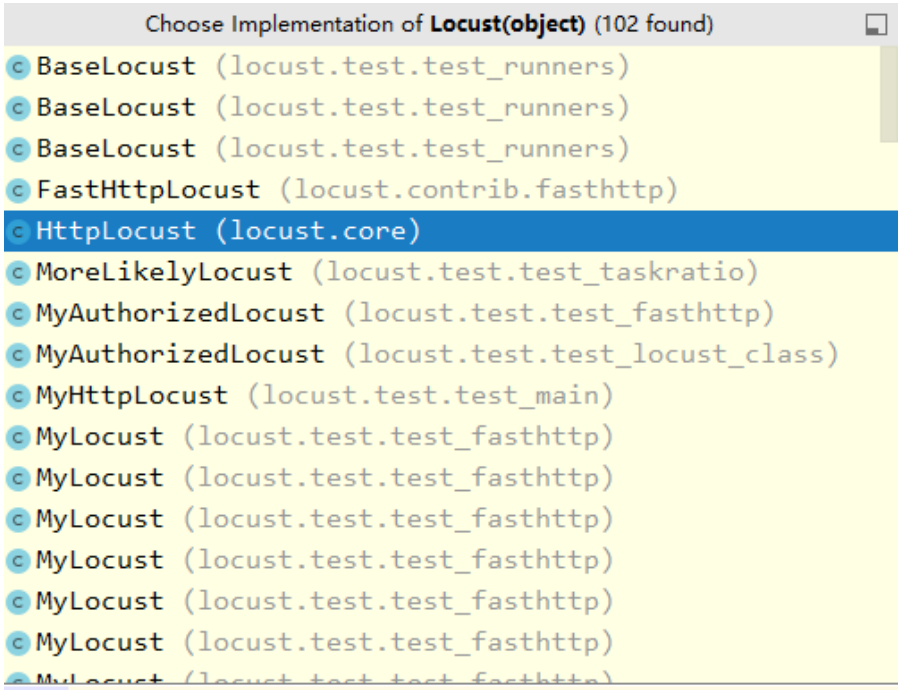
总体说来，Locust 通过纯 Python 编写压力测试脚本，而 Locust 框架提供了底层的 API 帮我们解决请求、并发、统计等问题。工程师需要做的是通过对业务的梳理，设计压力测试的场景，串联业务逻辑，之后通过 Locust 的 API 完成程序编写。

**Locust 支持的特性总结如下:**

### 4.1 Locust 类

**Locust 是请求类。所有的 HTTP 并发请求都继承 Locust 的 HttpLocust 类，该类是 Locust 类的子类。**

就像之前说明的一样，Locust 类就是模拟的一个一个用户，类比如蝗虫在系统中进行冲击。在 Locust 类中，具有一个 client 属性，它对应着虚拟用户作为客户端所具备的请求能力，也就是我们常说的请求方法。通常情况下，我们不会直接使用 Locust 类，因为其 client 属性没有绑定任何方法。因此在使用 Locust 时，需要先继承 Locust 类，然后在继承子类中的 client 属性中绑定客户端的实现类。实际上 Locust 有一些实现子类。




其中 HttpLocust 子类实现了常见的 HTTP(S)协议，其 client 属性绑定了 HttpSession 类，而 HttpSession 又继承自 requests.Session。因此在测试 HTTP(S)的 Locust 脚本中，我们可以通过 client 属性来使用 Python requests 库的所有方法，包括 GET/POST/HEAD/PUT/DELETE/PATCH 等，调用方式也与 requests 完全一致。另外，由于 requests.Session 的使用，因此 client 的方法调用之间就自动具有了状态记忆的功能。常见的场景就是，在登录系统后可以维持登录状态的 Session，从而后续 HTTP 请求操作都能带上登录态。当然也可以利用 header 的设置传递 token 等需要的信息。

一个 Locust 文件是一个或多个蝗虫的范本，可以定义多个继承于 HttpLocust（或者说 Locust)的 Locust 类，通过权重（weight）属性区别蝗虫的实例比例。如果不设定，则按照 1：1 的方式进行实例化。

而对于 HTTP(S)以外的协议，我们同样可以使用 Locust 进行测试，只是需要我们自行实现客户端。在客户端的具体实现上，可通过注册事件的方式，在请求成功时触发 events.request_success，在请求失败时触发 events.request_failure 即可。然后创建一个继承自 Locust 类的类，对其设置一个 client 属性并与我们实现的客户端进行绑定。后续，我们就可以像使用 HttpLocust 类一样，测试其它协议类型的系统。

在 Locust 类中，除了 client 属性，还有几个属性需要关注下：

task_set：指向一个 TaskSet 类，TaskSet 类定义了用户的任务信息，该属性为必填；

wait_time: 每个用户执行两个任务间隔时间的上下限（秒，支持小数），具体数值在上下限中随机取值，若不指定则默认间隔时间固定为 1 秒;

host: 被测系统的 host，当在终端中启动 locust 时没有指定--host 参数时才会用到;

weight: 同时运行多个 Locust 类时会用到，用于控制不同类型任务的执行权重。

**以获取红包验证码的脚本为例，测试开始后，每个虚拟用户（Locust 实例）的运行逻辑都会遵循如下规律:**

（1）先执行 WebsiteSequenceTasks 中的 on_start（只执行一次），作为初始化;

（2）按照 WebsiteSequenceTasks 设定的@seq_task 顺序执行每个任务，如果以@task 装饰器设定，则随机挑选（如果定义了任务间的权重关系，那么就是按照权重关系随机挑选）一个任务执行;

（3）根据 Locust 类 WebsiteUser 中 wait_time = between(1, 5)定义的间隔时间范围（如果 TaskSet 类中也定义了 min_wait 或者 max_wait，以 TaskSet 中的优先），在时间范围中随机取一个值，休眠等待;

（4）重复 2~3 步骤，直至测试任务终止。

### 4.2 TaskSet 类

**TaskSet 是请求任务类。如果说 Locust 类代表了一群冲击系统的蝗虫，那么请求任务类 TaskSet 就是这些蝗虫的冲击计划，官网的说法是“大脑”（brain）。**

具体地，TaskSet 类实现了并发用户所执行任务的调度算法，包括规划任务执行顺序（schedule_task）、挑选下一个任务（execute_next_task）、执行任务（execute_task）、休眠等待（wait）、中断控制（interrupt）等等。在此基础上，我们就可以在 TaskSet 子类中采用非常简洁的方式来描述虚拟用户的业务测试场景，对虚拟用户的所有行为（任务）进行组织和描述，并可以对不同任务的权重进行配置。

有两种方式声明 TaskSet，使用@task 装饰器和配置 TaskSet 类的 tasks 属性。

```
from locust import Locust, TaskSet, task
from locust.wait_time import between

class MyTaskSet(TaskSet):
    wait_time = between(5, 15)

    @task(3)
    def task1(self):
        pass

    @task(6)
    def task2(self):
        pass

class MyLocust(Locust):
    task_set = MyTaskSet
```

上述示例中，使用@task 定义了 TaskSet 类的两个执行任务，并且通过括号中的 weight 属性对执行量进行了设定：task2 的执行量是 task1 的两倍。

使用@task 装饰器是推荐的做法，这样代码更为清晰，当然也可以通过 tasks 属性来定义 TaskSet 执行的任务，实际上@task 装饰器就是设置 task 属性的实现。

```
from locust import Locust, TaskSet, task
from locust.wait_time import between

def task1(self):
    pass

def task2(self):
    pass

class MyTaskSet(TaskSet):
    wait_time = between(5, 15)
    tasks = {task1: 3, task2: 6}

class MyLocust(Locust):
    task_set = MyTaskSet
```

上述代码实现的功能与@task 装饰器的功能一致。

TaskSet 类支持嵌套定义，从而更便捷的对分层构建的网站设计模拟用户的行为。

之前我们使用的 TaskSequence 是按序请求类。上面介绍的项目实例中，抢红包我们

使用的是 TaskSequence，它也是 TaskSet 的子类。但其任务将按顺序执行。

```
class MyTaskSequence(TaskSequence):
    @seq_task(1)
    def first_task(self):
        pass

    @seq_task(2)
    def second_task(self):
        pass

    @seq_task(3)
    @task(10)
    def third_task(self):
        pass
```

上面的示例中，首先执行 first_task 一次，之后执行 second_task 一次，最后执行 third_task 十次。从这里也可以看出，@seq_task 和@task 装饰器可以组合使用，还可以将 TaskSet 嵌套在 TaskSequences 中，当然也可以反过来嵌套。

### 4.3 环境设置和恢复类

**通过 setup、teardown、on_start、on_stop 可以完成 Locust 并发测试环境设置和恢复。**

Setup 和 teardown 在一次测试中只会执行一次。Setup 在所有任务开始运行之前运行，teardown 在所有任务完成并在 Locust 退出之后运行。类似于 TestNG 的 BeforeSuite 和 AfterSuite，这样可以利用这两个函数完成一些环境设置（如连接数据库），并在最后进行环境清理（如关闭数据库）。

on_start 和 on_stop 函数，当对应的 TaskSet 任务执行时调用 on_start 方法，该 TaskSet 停止时调用 on_stop 方法。每运行一次调用一次。类似于 TestNG 的 BeforeClass 和 AfterClass。典型的 on_start 用法是登录请求放在该函数中。

### 4.4 参数关联

**通过参数关联完成请求间的参数传递。**

请求间的参数传递是并发测试中的必要需求，最典型的就是登录状态，如本例中的 token 传递。LoadRunner 使用 web_reg_save_param 进行请求的参数提取，JMeter 使用参

数定义，也可以使用自带的 vars 字典进行参数存储和获取。Locust 是纯 Python 脚本的编写，所以可以调用任意的 Python 库进行参数关联。

在本例中，我们在 on_start 函数中利用返回 JSON 内容提取的方式将 token 存储在 self.cl_token 中，在后续需要使用该参数的地方直接调用该参数。其他的情况都可以通过 Python 程序编写的方式完成参数关联。

### 4.5 参数化

**参数化设定并发测试的初始参数。**

在并发初始前，设定测试的参数，最典型的就是登录的用户和密码。LoadRunner 利用 Parameter List 进行参数化，支持界面配置和外部文件。JMeter 有著名的 CSV Data Config。与参数关联一样，通过 Python 脚本编写完成。

在本例中，我们抢红包的 ID 是需要参数化的，将参数存储在 id 配置文件中（格式见上），利用 load_data_set 函数完成该配置数据的读取，并放入队列中。在请求中直接完成调用。

### 4.6 检查点

**通过 Python 自带的 assert 实现断言。**

通过 Python 脚本编程，完成需要断言数据的提取，之后使用 assert 进行判断。

### 4.7 集合点

**Locust 不支持集合点的设置。**

Locust 框架并不支持集合点的设置，没有类似于 LoadRunner 的 Rendezvous 和 JMeter 的同步定时器的特性。

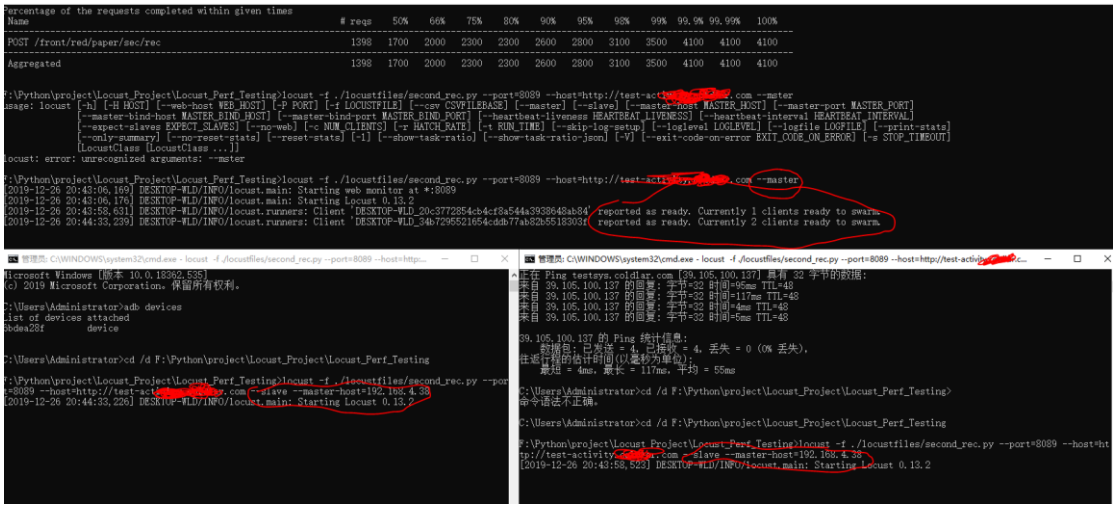

### 4.8 分布式运行

**通过分布式运行提高性能测试的并发量。**

当单台计算机不足以模拟所需的用户数量时，通过分布式运行 Locust，利用多台计算机提高负载。

Locust 采用主从模式进行分布式运行，这点和 JMeter 的分布式很类似。在其中一台负载测试机采用--master 参数运行 Locust，其他负载机使用--slave 确定自己的身份，并使用--master-host 指定从属于那个主节点。

下面的例子是通过多个进程运行 Locust 的方法， 分布式运行的方法与之一致。

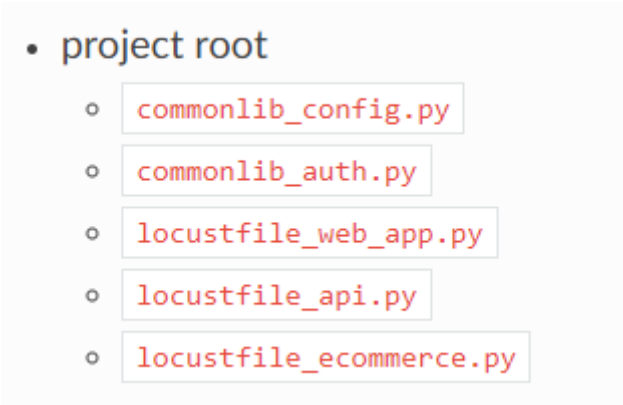


当我们打开 Locust 的 Web 管理界面是，也会看到 Slave 的个数为 2。

### 4.9 典型工程设计

**通过更清晰的工程设计，提高 Locust 工程的可读性。**

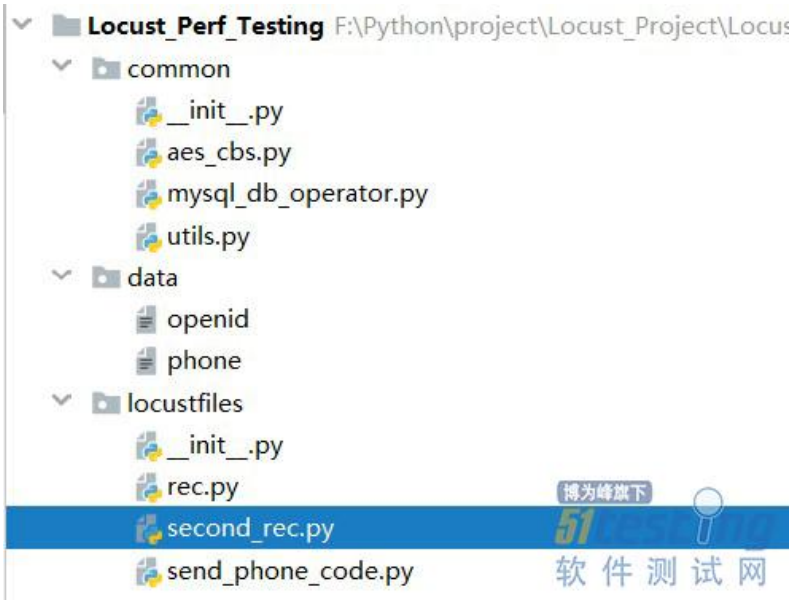
由于纯 Python 编程，可以使用各种工程组织方式，如平面化的文件排布。




以 Locustfile 开头的是 Locust 请求类，其他的为通用函数，类似于本例中的加解密、数据连接等。这样的设计缺点是不能将普通的库文件和 Locust 文件分开，不利于管理。通过分目录管理的方式，会使得工程更清晰，比如下面的设计。

- project root
  - __init__.py
  - common/
    - __init__.py
    - config.py
    - auth.py
  - locustfiles/
    - __init__.py
    - web_app.py
    - api.py
    - ecommerce.py

需要注意的是，由于 Locust 只会导入相对于运行的 Locust 文件所在路径下的模块，所以需要导入工程根目录的模块（比如在根目录下运行 Locust 测试），我们需要在 Locust 文件中使用“sys.path.append(os.getcwd())”来导入其他库文件。这样就可以确保根目录下的模块可以引用了。本例中我们采用类似的方式组织 Locust 工程。




■从零开始做项目，开启 Locust 性能压测 >> https://dwz.cn/DUOOUP3I

# 测试女巫紧跟时代脉搏之大数据分析系列

◆ 作者：王平平

## 一、前言

转眼已经 2020 年了，这几年女巫过着高三冲刺班式的生活，真的把自己塞得满满的=_=，早上地铁与当代思想隐士熊逸在书中相遇；白天在公司使用 angular 和 python 以及使用逻辑三段论解决各种奇怪的问题；晚上学习相对论；周末学习线上课程：AI，大数据……，是的我已经不年轻了，但是为什么总觉得对一切事物依然充满了好奇，总想知道得多一些再多一些；因为是年末我也回顾了我的 51Testing 杂志之旅：从 2014 年开始，几乎每一期都贡献了自己的工作总结，到今年也坚持了整整 5 年了，真的为自己觉得自豪；天道酬勤，是的，我发现对于新鲜事物总能很快抓住问题的本质，并快速解决问题，前段时间评估一个被其他同事已经研究了大半年，还未能实现自动化的 5G 的仪器；研究几天就发现，控制它其实与控制其它的仪器并没有什么本质的不同，很快就突破了控制它的难点；这几年过得非常辛苦，但是非常开心，真的有了实质的进步，非常感谢遇到让我看到外面世界的主管，虽然这几年有苦有乐，但是还是心存感激，好吧，真的到了快 40 岁才来到了美丽新世界，希望我也能带你进入这个世界^_^，计划将近期学习的数据分析，进行总结预计数据分析系列，会有十期，从基础知识到嘴壶的实战由浅入深地给大家介绍，我会奉献我对于数据分析的感悟，希望可以让大家也能有一些所得。

这一期我们学习数据分析的基础知识准备之 Jupyter Numpy Pandas 的介绍

## 二、Jupyter

**1.理念介绍**

它是介于 IDE 与 Editor 之间的，让你可以写 code 的工具，它之前的名字叫做: ipython notebook;所以它可以逐行执行 code,且可以将资料做到视觉化，而且可以将文档输出成 HTML 嵌入到任意网页上或者 blog 上，或者直接用 notebook 直接分享，由于容易分享，许多 AI 的课程都是使用 Jupyter notebook 进行分享

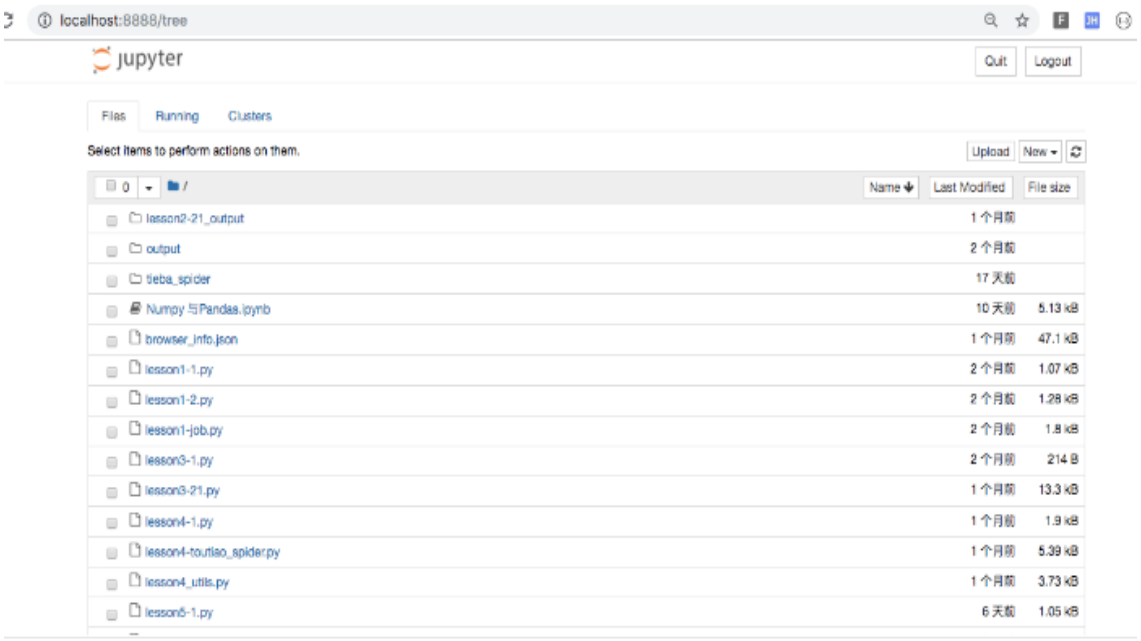
**2.使用说明**

1）安装

使用 pip install -user jupyter 安装 jupyter

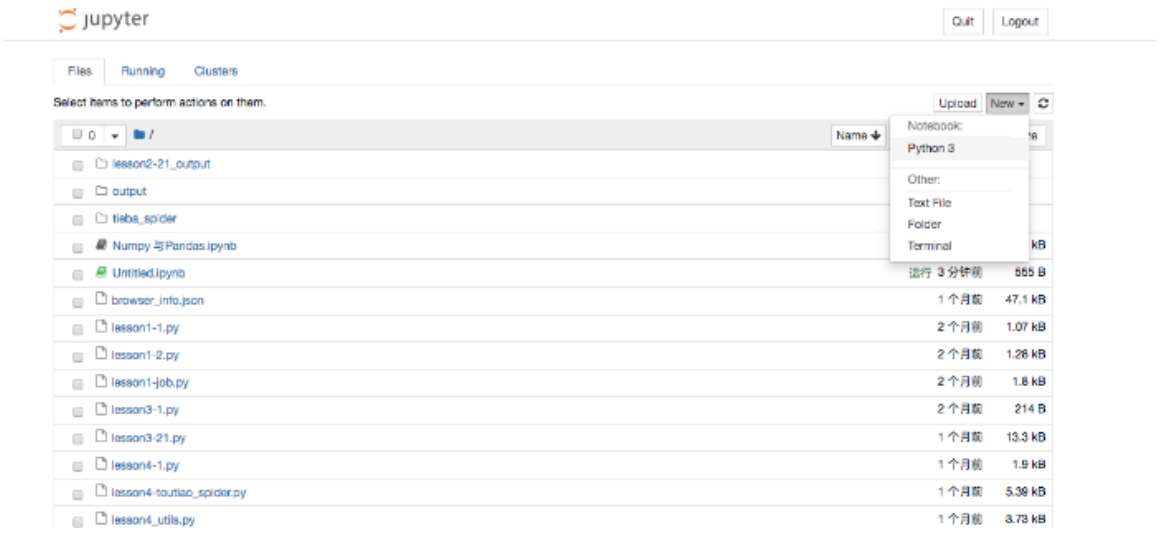
2）启动 jupyter notebook

用命令：jupyter notebook 即可以启动，如下图 1




3）新建

新建 python3 的 jupyter notebook，选择 new,选择 python3 就可以建立 python3 的 notebook,如下图 2

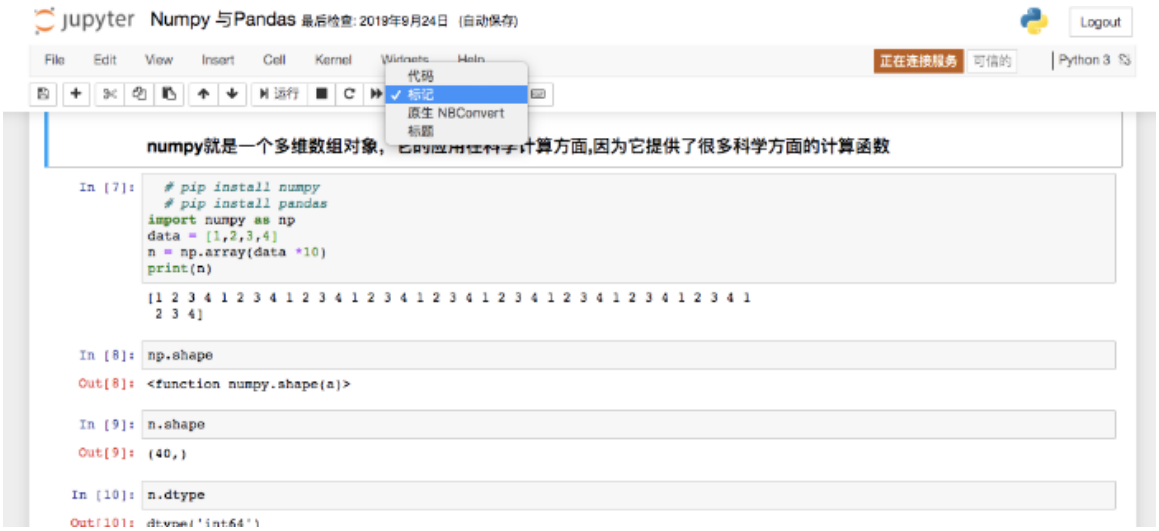



假如取一个名称为：Numpy 与 Pandas

新建完毕后会出现一个文件，如下图 3


4）编辑文档

选择"标记"即为 markdown 的格式的编辑方式，使用 markdown 的语法进行编辑即可



5）运行代码

当选择代码时，可以点击 gui 上的运行就可以运行；也可以使用快捷键

shift+enter 就是运行并打开下一行

alt+enter 就是运行并插入一行

ctrl_enter 就是只执行这一行，不会多出任意形式的命令行

## 三、 numpy

### 1.理念介绍

numpy 就是一个多维数组对象，它的应用在科学计算方面,因为它提供了很多科学方面的计算函数

查看官网可以看出，有很多数学的工具，在 numpy 中都提供如下图



### 2.基本用法

#### 1）创建 numpy

通过 list 创建 numpy，即使用 np.array 的方式创建 10 个 data，此时 n 的类型就是 numpy 类型

```
  # pip install numpy
  # pip install pandas
  # pip install xlrd  #读取excel文件时使用这个模组

import numpy as np
data = [1,2,3,4]
n = np.array(data *10)
print(n)
print(type(n))
```

运行结果:

```
[1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1
 2 3 4]
 <class 'numpy.ndarray'>
```

**2）numpy 的属性**

a.行和列形状描述

np.shape：得到 numpy 的维度

```
n.shape
```

有 40 行，没有列，所以运行结果如下：

```
(40,)
```

b.维度

```
n.ndim
```

运行结果：因为只有 1 行没有列，所以维度为 1 维

```
1
```

c.数据类型

```
n.dtype
```

运行结果：

```
dtype('int64')
```

**3)嵌套 numpy**

```
arr = [[1,2,3,4],[3,4,5,6]]
arr2 = np.array(arr)
print(arr2)
```

运行结果如下：

```
[[1 2 3 4]
 [3 4 5 6]]
```

**4)确认 numpy 对象的属性**

a.确认 numpy 对象的形状

```
arr2.shape
```

运行结果如下:

即将一维的嵌套 list 转为 2 行 4 列的 numpy 对象

```
(2,4)
```

b.确认维度

```
arr2.ndim
```

运行结果如下:

即 2 行 4 列，所以有两个维度

```
2
```

**3.Numpy 数学方面的应用**

**1) Mean**

先产生一个 4 行 4 列的二维数组

```
arr = np.random.rand(4,4)
arr
```

运行结果如下:

```
array([[0.73824019, 0.72777245, 0.74648963, 0.11777866],
       [0.73338719, 0.94115832, 0.28946755, 0.18763299],
       [0.42527463, 0.79316918, 0.44636211, 0.0268116 ],
       [0.75104438, 0.78403299, 0.89953596, 0.69413876]])
```

**2)计算平均数**

```
arr.mean()
```

运行结果如下:

```
0.5813935366560115
```

**3)计算相加的结果**

```
arr.sum()
```

运行结果如下:

```
9.302296586496183
```

**4)计算标准差**

```
arr.std()
```

如果是求这个矩阵的一行的标准差则直接 arr.std(0)即求第一行的标准差运算结果如下:

```
0.2806032510408064
```

## 四、Pandas 介绍

### 1、numpy 与 pandas 的区别

pandas 是基于 numpy 构建的,让 numpy 为中心的

pandas 是专注于数据处理，这个库可以帮助数据分析，数据挖掘，算法等工程师岗位的人轻松快速的解决处理预处理的问题，比如说数据类型的转换，缺失值的处理，描述性统计分析，数据汇总等功能

### 2、Series

可以看做是有标签的一维数组，对于整数来说，标签就是 index；对于 string 来说，标签可以是 key，也可以是自动产生的 index

```
import pandas as pd
obj = pd.Series([1,2,3,4])
obj
```

运行结果左边是 index，右边是 value

```
0    1
1    2
2    3
3    4
dtype: int64
```

提取其中的数据

```
obj[0]
```

运行结果:

```
1
```

提取更多的数据

```
obj[[0,1,2]]
```

运行结果:

```
0    1
1    2
2    3
dtype: int64
```

**3、DataFrame**

（1）定义

用来建立二维的尺寸可变的数组

（2）构建 DataFrame 方法一

```
df1 = pd.DataFrame([['李四','上海','30'],['王五','北京','24'],['赵六','天津','60']])
df1
```

运行结果:

可以看出它的索引比 Seiral 多了横排的索引：即有行索引，也有列索引

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 李四 | 上海 | 30 |
| 1 | 王五 | 北京 | 24 |
| 2 | 赵六 | 天津 | 60 |

a) 如何获取其中的数值，承接上面的代码

```
df1[0]
```

运行结果:

```
0    李四
1    王五
2    赵六
Name: 0, dtype: object
```

b) 如何获取多维数组

```
df1[0][1]
```

运行结果：列为第一列；行为第二行

```
'王五'
```

（3）构建 DataFrame 方法一

```
data = {
    "60":["张三","李四","王五"],
    "70":["赵六","丫丫","啦啦"]
}
frame_data = pd.DataFrame(data)
frame_data
```

运行结果：

```
   60   70
0  张三   赵六
1  李四   丫丫
2  王五   啦啦
```

（4）读取文本文件

注意读取文本文件都是使用 pd.read_csv 这个函数，所以文件扩展名为 csv,txt 等都是使用这个函数进行读取文档的内容

```
pd.read_csv('testpandas.csv')
```

运行结果，会将 csv 中内容以多维 array 的方式显示出来；默认第一行是索引

```
   a  b  c  d
0  1  2  3  4
1  5  6  7  8
```

a) 如果希望文档的第一行不要作为索引

```
pd.read_csv('testpandas.csv',header=None)
```

运行结果，如下：

```
   0   1   2   3
0  a   b   c   d
1  1   2   3   4
2  5   6   7   8
```

b) 如果希望以某一列作为索引

```
pd.read_csv('testpandas.csv',index_col='b')
```

运行结果，如下:

```
a   c   d
b
2   1   3   4
6   5   7   8
```

（5）读取 excel 文档

```
pd.read_excel('testpandas.xlsx')
```

运行结果，如下:

```
   a   b   c   d
0  1   2   3   4
1  5   6   7   8
```

a) 指定 sheetname

```
excel_data = pd.read_excel('data.xlsx',sheet_name='worksheet2')
excel_data
```

运行结果，如下:

```
   id   age   place   u
0   0    1    11    101
1   1      2    12    102
2   2      3    13    103
3   3      4    14    104
4   4    5    15    105
5   5      6    16    106
6   6      7    17    107
7   7      8    18    108
8   8      9    19    109
9   9      10    20    110
```

b) 查看 shape

```
excel_data.shape
```

运行结果如下：

```
(10, 4)
```

c) 查看每列的数据类型

```
excel_data.dtypes
```

运行结果如下：

```
id        int64
age       int64
place     int64
u         int64
dtype: object
```

## 五、统计学之叙述统计

```
excel_data.describe()
```

运行结果如下：

即计算每一列的叙述统计，注意它只针对数据类型为数值，string 类型的数据类型是不能做叙述统计的。

```
             id         age         place         u
count   10.00000    10.00000    10.00000    10.00000
mean     4.50000     5.50000    15.50000   105.50000
std      3.02765     3.02765     3.02765     3.02765
min      0.00000     1.00000    11.00000   101.00000
25%      2.25000     3.25000    13.25000   103.25000
50%      4.50000     5.50000    15.50000   105.50000
75%      6.75000     7.75000    17.75000   107.75000
max      9.00000    10.00000    20.00000   110.00000
```

## 结尾

这是我们数据分析系列的第一篇，所以我们讲了很多基础概念，当然基础概念没有那么吸引人，但是无论是 AI 还是大数据分析，上述讲解的基础知识都是必备的，万丈高楼也是需要平地起，希望大家真的可以静下心来，一步步走下去，下一期我会开始给大家一个范例，看如何将上述的基础知识应用到实际的数据的分析。

# JMeter 扩展开发之 JMeter 核心源码解读

◆作者：杨易寰

JMeter 作为开源性能测试工具，近年来社区发展活跃，业界应用广泛，工具功能愈发完善成熟，成为领域内十分流行的性能测试工具之一。得益于开源特性，JMeter 具有得天独厚的可灵活扩展性，用户可根据实际功能需求和测试场景，对 JMeter 进行自定义扩展开发。

如何对更好地实现 JMeter 扩展开发？本文先从核心源码层入手，对 JMeter 启动、加载、运行的应用原理进行解读，涉及到 NewDriver、JMeter、JMeterEngine、HashTree 等相关内容。

**1. 运行入口—NewDriver 类**

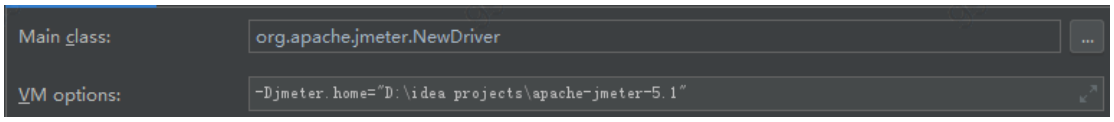
NewDriver 是整个 JMeter 的入口类， 完整路径是 org.apache.JMeter.NewDriver。NewDriver 类中包括 jar 包扫描、命令解析、全局参数定义、类加载器路径操作等静态方法，核心部分是 main 方法，main 方法通过反射调用 JMeter 类，启动其 start 方法，通过对传入参数的解析判断是否执行 GUI 或 NON_GUI 模式，这里即区分了日常使用中的界面调试和命令行发压两种应用形态。需要注意的是，在日常应用中，GUI 模式也就是界面模式由于存在大量的界面资源加载和元件数据监听，尤其是监听器元件存在大量数据同步和更新，所以在编译源码启动 JMeter 时官方在日志中特别强调若要执行发压工作，请务必使用命令行模式，否则会影响发压效率和资源消耗。在实际使用中，笔者认为应综合考虑测试场景，若存在小并发且对响应时间和 TPS 敏感度不高的场景，通过界面级操作发压也非并不可。

在首次用 IDE 进行源码编译时，NewDriver 会报错“配置文件无法找到”的错误，原因是在 NewDriver 配置 JMeter Home 地址方法中，默认会把 Parent 目录当做安装根目录，所以会存在无法找到 properties 文件而产生的错误。

```
} else {// e.g. started from IDE with full classpath
    tmpDir = System.getProperty( key: "jmeter.home", def: "");// Allow override $NON-NLS-1$ $NON-NLS-2$
    if (tmpDir.length() == 0) {
        File userDir = new File(System.getProperty("user.dir"));// $NON-NLS-1$
        tmpDir = userDir.getAbsoluteFile().getParent();
    }
}
JMETER_INSTALLATION_DIRECTORY=tmpDir;
```

所以通过编译执行的源码，在启动 NewDriver 时需首先在 Run Configration 中手动配置 JMeter.home 参数。

**-Djmeter.home="JMeter 项目的根目录（bin 文件夹所在目录）"**




继续看 main 函数，首先设置当前线程的类加载器，随后将启动命令参数中关于日志文件和日志类型的字符做解析并配置好日志属性。随即进入反射调用部分。代码如下所示，主要是加载类、获取实例、获取方法类型、反射调用四个步骤。

```
// 加载 JMeter 类
Class<?> initialClass = loader.loadClass("org.apache.JMeter.JMeter");
// 获取 JMeter 类实例
Object instance = initialClass.getDeclaredConstructor().newInstance();
// 获取 start 方法类型实例
Method startup = initialClass.getMethod("start",  new Class[] { new String[0].getClass() });
// 反射调用 JMeter 类的 start 方法
startup.invoke(instance,  new Object[] { args });
```

此时关于 JMeter 配置方面的内容包括类加载器、日志设置、系统环境变量、工具目录等部分已经完成，随着进入 JMeter 实例的 start 方法，开始执行更具体的资源加载、属性更新、命令解析工作。

**2. 逻辑操作—JMeter 类**

JMeter 类由 NewDriver 入口类通过反射调用，是实际的业务逻辑类，通过接收用户启动参数进行解析。JMeter 类做的事情主要有四个方面。

1) 解析命令参数及加载配置文件。

2) 将 .JMX 文件解析成 HashTree。

3) 实例化一个 StandardJMeterEngine，并把测试的工作交给 JMeterEngine。

4) 监听所有的 JMeterEngine，当接收到 GUI 的 StopTestNow / Shutdown 等命令时，调用 JMeterEngine 接口相应的方法。

其中 start 方法是 JMeter 的重点部分。因为其承担了传入指令的第一步解析和对后续应用模式的逻辑判断。

首先解析启动时候传入的外部命令行参数并加载相应配置文件。

```
// 解析命令行参数
CLArgsParser parser = new CLArgsParser(args， options);
//初始化参数配置+初始化 JMeter 日志
initializeProperties(parser);
```

随后执行 updateClassLoader 方法，把 search_paths 和 user.classpath 里面设置的 jar 文件添加到自定义的 classloader 中，它的作用可以让我们在某些场景下更方便使用 JMeter。例如我们自定义的 Java Sampler，理论上是要打包成 Jar 放到 lib/ext 目录下，但通过设置 search_paths 和 user.classpath 的方式，利用本方法可以来简化操作，将自定义开发中的 Sampler 放在不同位置。

```
//将 jar 文件添加到自定义的 classloader 中
updateClassLoader();
```

随后根据传入参数的解析结果，进行两个核心调用。

**一个是启动 GUI 的模式，即启动带界面的工具模式。**

```
else if (parser.getArgumentById(NONGUI_OPT) == null) { // not non-GUI => GUI
  startGui(testFile);
  startOptionalServers();
```

传入参数为 JMX 测试脚本文件路径。在 startGui 中需进行界面显示资源的加载、布局监听的初始化、界面插件的装配以及存储结构初始化等一系列操作伴随进行，最后用户通过界面面板进行操作，触发不同的监听事件执行不同的工具操作调用。GUI 模式是脚本设计和测试调试期间应用的模式，有比较直观的显示效果，但因其中包含大量的监听事件以及数据资源更新，因此本模式下对宿主机资源消耗和发压执行效率会有一定影响。

**另一个是启动 NON_GUI 模式，也就是日常使用的命令行执行动作。**

```
else { // NON-GUI must be true
  extractAndSetReportOutputFolder(parser, deleteResultFile);

  CLOption rem = parser.getArgumentById(REMOTE_OPT_PARAM);
  if (rem == null) {
      rem = parser.getArgumentById(REMOTE_OPT);
  }
  CLOption jtl = parser.getArgumentById(LOGFILE_OPT);
  String jtlFile = null;
  if (jtl != null) {
      jtlFile = processLAST(jtl.getArgument(), suffix: ".jtl"); // $NON-NLS-1$
  }
  CLOption reportAtEndOpt = parser.getArgumentById(REPORT_AT_END_OPT);
  if(reportAtEndOpt != null && jtlFile == null) {
      throw new IllegalUserActionException(
              "Option -"+ ((char)REPORT_AT_END_OPT)+" requires -"+((char)LOGFILE_OPT )+ " option");
  }
  startNonGui(testFile, jtlFile, rem,  generateReportDashboard: reportAtEndOpt != null);
  startOptionalServers();
```

startNonGui 方法执行前，需进一步对命令行参数进行解析，包括远程执行、日志文件、测试报告等，随后向 startNonGui 传入对应参数并启动。startNonGui 中创建了一个新的 JMeter 实例 driver，在非分布式的场景下，调用执行 runNonGui 方法。

```
private void startNonGui(String testFile, String logFile, CLOption remoteStart, boolean generateReportDashboard)
        throws IllegalUserActionException, ConfigurationException {
    // add a system property so samplers can check to see if JMeter
    // is running in NonGui mode
    System.setProperty(JMETER_NON_GUI, "true");// $NON-NLS-1$
    JMeter driver = new JMeter();// TODO - why does it create a new instance?
    driver.remoteProps = this.remoteProps;
    driver.remoteStop = this.remoteStop;
    driver.deleteResultFile = this.deleteResultFile;

    PluginManager.install( plugin: this,  useGui: false);

    String remoteHostsString = null;
    if (remoteStart != null) {
        remoteHostsString = remoteStart.getArgument();
        if (remoteHostsString == null) {
            remoteHostsString = JMeterUtils.getPropDefault(
                    propName: "remote_hosts", //$NON-NLS-1$
                    defaultVal: "127.0.0.1");//NOSONAR $NON-NLS-1$
        }
    }
    if (testFile == null) {
        throw new IllegalUserActionException("Non-GUI runs require a test plan");
    }
    driver.runNonGui(testFile, logFile,  remoteStart: remoteStart != null, remoteHostsString, generateReportDashboard);
}
```

runNonGui 解析脚本内容并生成 HashTree 结构保存，再根据 remoteStart 变量区分是否为 Remote 模式。以本地运行场景为例，JMeter 会新建 StandardJMeterEngine 引擎实例，传入已解析完成的 HashTree 数据结构，执行 engine.runTest()启动新线程执行异步调用，并在将该引擎加入引擎列表中。

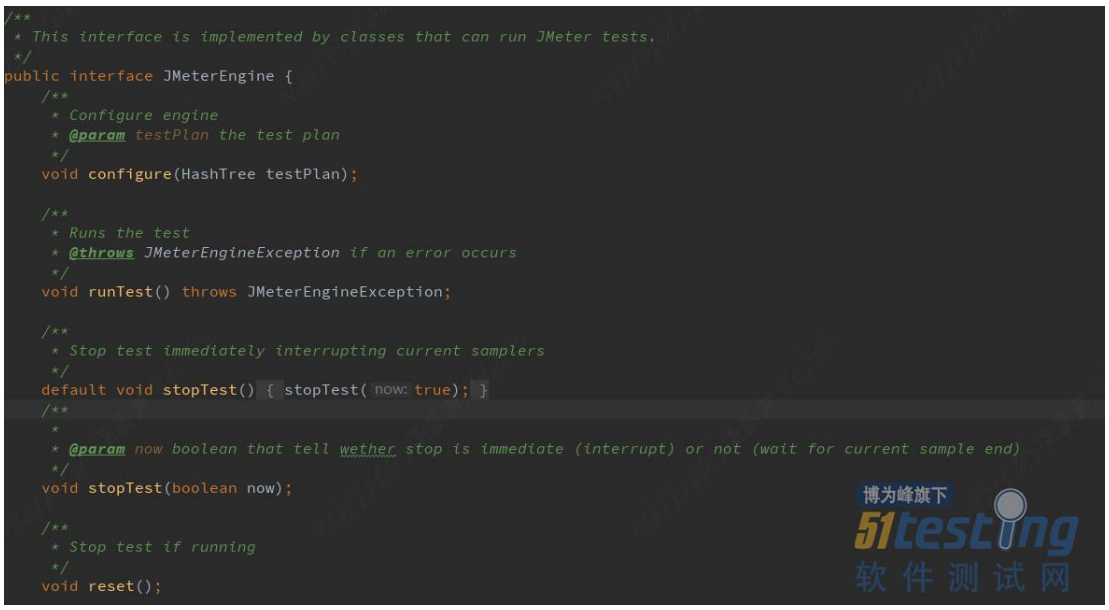
runNonGui 方法最后调用的 startUdpDdaemon(engines) 是一个 JMeter 管理服务，传入维护的引擎列表，用来管理 JMeter 的运行状况，通过 waitForSignals 方法启动 socket 监听，识别 Shutdown/StopTestNow/HeapDump/ThreadDump 等操作指令，这样达到了 JMeter 对发压引擎的管理。

```
if (!remoteStart) {
    JMeterEngine engine = new StandardJMeterEngine();
    engine.configure(clonedTree);
    long now=System.currentTimeMillis();
    println("Starting the test @ "+new Date(now)+" ("+now+")");
    engine.runTest();
    engines.add(engine);
} else {
    java.util.StringTokenizer st = new java.util.StringTokenizer(remoteHostsString, delim: ",");
    List<String> hosts = new LinkedList<>();
    while (st.hasMoreElements()) {
        hosts.add((String) st.nextElement());
    }

    DistributedRunner distributedRunner=new DistributedRunner(this.remoteProps);
    distributedRunner.setStdout(System.out); // NOSONAR
    distributedRunner.setStdErr(System.err); // NOSONAR
    distributedRunner.init(hosts, clonedTree);
    engines.addAll(distributedRunner.getEngines());
    distributedRunner.start();
}
```

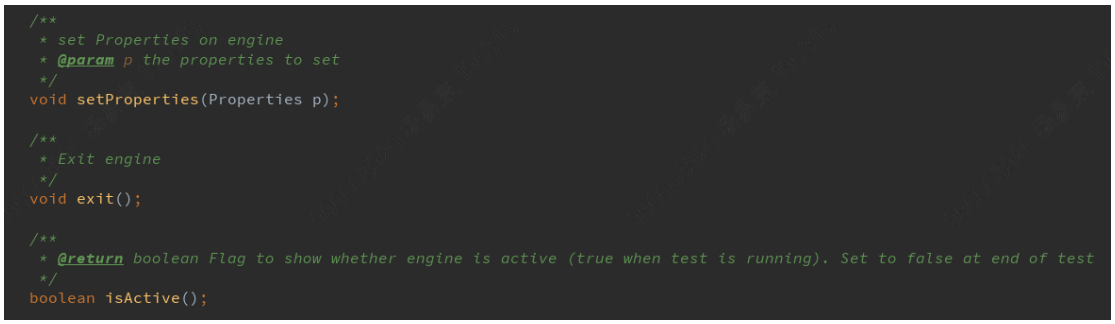
在启动 GUI 和 NON_GUI 模式中，都执行了 startOptionalServers()方法.这个方法的核心逻辑就是先在配置文件中读取相关 beanshell 的配置文件信息，然后调用 Runnable t = new BeanShellServer(bshport， bshfile)来启动一个 BeanShell 服务。BeanShell 实际作用可以理解为对 runtime 的 JVM 做一些动态的调整，例如获取 JMeter 变量，或者修改一些变量来改变最后 JMeter 的运行行为等，以达到方便对 JMeter 进程做一些动态调整的目的。

### 3. 发压引擎—JMeterEngine 引擎接口

JMeterEngine 是发压引擎接口，具体实现为 StandardJMeterEngine 。JMeterEngine 中主要定义了引擎的操作方法，包括配置 HashTree、执行测试、停止测试、设置属性和引擎状态识别等。JMeterEngine 依赖于由 JMX 文件解析而来的 HashTree ，每一个 JMeter 测试计划都会对应一个 JMX 文件。所以只要能合理的定义各测试组件并组装起来，按规则生成 JMX 文件，就可通过 JMeterEngine 压测引擎去执行测试任务而不用依赖 JMeter 工具本身，这也是调用发压引擎隐式开展发压操作的新方向。JMeterEngine 引擎接口定义如下所示。



```java
/**
 * This interface is implemented by classes that can run JMeter tests.
 */
public interface JMeterEngine {
    /**
     * Configure engine
     * @param testPlan the test plan
     */
    void configure(HashTree testPlan);

    /**
     * Runs the test
     * @throws JMeterEngineException if an error occurs
     */
    void runTest() throws JMeterEngineException;

    /**
     * Stop test immediately interrupting current samplers
     */
    default void stopTest() { stopTest( now: true); }
    /**
     *
     * @param now boolean that tell wether stop is immediate (interrupt) or not (wait for current sample end)
     */
    void stopTest(boolean now);

    /**
     * Stop test if running
     */
    void reset();
```



```java
    /**
     * set Properties on engine
     * @param p the properties to set
     */
    void setProperties(Properties p);

    /**
     * Exit engine
     */
    void exit();

    /**
     * @return boolean Flag to show whether engine is active (true when test is running). Set to false at end of test
     */
    boolean isActive();
```

StandardJMeterEngine 是本地单机版本 JMeter 进行发压操作的实际处理类。StandardJMeterEngine 还实现了 Runnable 接口，在 StandardJMeterEngine 的 runTest()的实现中可以发现，this 作为参数放置到了 Thread 进行调用，所以 StandardJMeterEngine 自身会是多线程执行的模式。

```java
@Override
public void runTest() throws JMeterEngineException {
    if (host != null){
        long now=System.currentTimeMillis();
        System.out.println("Starting the test on host " + host + " @ "+new Date(now)+" ("+now+")"); // NOSONAR Intentional
    }
    try {
        Thread runningThread = new Thread( target: this,  name: "StandardJMeterEngine");
        runningThread.start();
    } catch (Exception err) {
        stopTest();
        throw new JMeterEngineException(err);
    }
}
```

runTest()中启动 thread，会进入其 run 方法正式开始执行测试脚本。在 run 方法中的 JMeterContextService.startTest()是用来设置运行启动的状态。JMeterContextService 是在整个 JMeter 的线程之间是共享的，但其中的 threadContext 的变量属于 ThreadLocal 独享，也就是每个线程独享。

```
SampleEvent.initSampleVariables();

JMeterContextService.startTest();
```

运行状态和 HashTree 数据遍历完成后，用 Setup Thread Group 启动 N 个线程组来做初始化，随后主动触发 JMeterUtils.helpGC()动作，紧接着开始执行测试的 Abstract Thread Group 和最后的 Post Thread Group 。在线程组的操作中都是根据脚本中设置的线程组数量进行循环操作，每个循环内部是一个线程组的执行操作,通过调用 startThreadGroup 方法将本线程组信息传入，随即启动 group.start()开始执行。

```
while (running && setupIter.hasNext()) {//for each setup thread group
    AbstractThreadGroup group = setupIter.next();
    groupCount++;
    String groupName = group.getName();
    log.info("Starting setUp ThreadGroup: {} : {} ", groupCount, groupName);
    startThreadGroup(group, groupCount, setupSearcher, testLevelElements, notifier);
    if (serialized && setupIter.hasNext()) {
        log.info("Waiting for setup thread group: {} to finish before starting next setup group",
                groupName);
        group.waitThreadsStopped();
    }
}
```

这些操作和 Setup Thread Group 的调用都大同小异，因为实际执行发送请求和并发线程的管理都在 Thread Group 中，根据线程组设置的线程数量启动循环新建线程 JMeterThread 再调起具体发压。

上文提到，StandardJMeterEngine 仅是 JMeterEngine 的一种实现，它主要是实现了单机版本的发压引擎功能。而 JMeterEngine 的另一个实现是分布式模式下的 ClientJMeterEngine，其承担 Master 角色对分布式测试任务进行管理，这里不做详细介绍。

**4. 存储结构—HashTree 存储结构**

JMeter 的测试脚本是 JMX 文件，其中保存了不同元件和变量的数据内容，可以用于在 GUI 模式下加载整个测试组件，也可以直接运行于 NON_GUI 的执行模式，文件格式基于 XML 类型。而 HashTree 则是在内部存储文件信息的数据结构，是 JMX 文件在内存的一份映射。

HashTree 将数据组织到一个递归树结构中并提供了操作该结构的方法。HashTree 数据结构设计的理论基础是“质数分辨定理”，简单来说就是“n 个不同质数可以分辨的连续整数的个数和这些质数的乘积相等”，分辨是指连续整数不可能有完全相同的余数序

列。HashTree 保证每层节点的子节点个数为连续质数，因此 HashTree 是动态的，子节点可以随意创建，而由于结构本身是一个单项增加结构，所以这种结构会随着存储量的增加而不断增大。在 JMeter 中，HashTree 用于创建测试对象的树结构，树中每一个元素也是下一个节点的键。利用 HashTree 在十次比较之内即可查找到目标对象是否存在，可以进行快速匹配和查找，效率可观。

**观察 JMX 文件可知测试脚本是具有格式要求的，每一层都只有 2 个类型的节点：**

1) Object 节点 - 代表一个 Test Component。

2) HashTree - 一个 HashTree 的子节点。

```
public class HashTree implements Serializable, Map<Object, HashTree>, Cloneable {

    private static final long serialVersionUID = 240L;

    // Used for the RuntimeException to short-circuit the traversal
    private static final String FOUND = "found"; // $NON-NLS-1$

    // N.B. The keys can be either JMeterTreeNode or TestElement
    protected final Map<Object, HashTree> data;
```

查看 org.apache.jorphan.collections.HashTree 这个类的定义可以发现本结构核心存储就是一个 protected final Map<Object， HashTree> data 的 Map ，通过对外提供 Map 接口来给上层提供读写的能力。ListedHashTree 是 HashTree 的子类，在 startThreadGroup 具体执行线程组操作实际是将 HashTree 向下转型为子类 ListedHashTree 对象，利用到其顺序特性 。

```
/**
 * ListedHashTree is a different implementation of the {@link HashTree}
 * collection class. In the ListedHashTree, the order in which values are added
 * is preserved . Any listing of nodes
 * or iteration through the list of nodes of a ListedHashTree will be given in
 * the order in which the nodes were added to the tree.
 *
 * @see HashTree
 */
public class ListedHashTree extends HashTree implements Serializable, Cloneable {
    private static final long serialVersionUID = 240L;

    private final List<Object> order;

    public ListedHashTree() {
        super();
        order = new LinkedList<>();
    }
```

JMeter 在 HashTree 数据遍历上采用了访问者设计模式，HashTreeTraverser 定义了访问者的标准接口，实现解耦并提供更好灵活的扩展性。

```java
public interface HashTreeTraverser {
    /**
     * The tree traverses itself depth-first, calling addNode for each object it
     * encounters as it goes. This is a callback method, and should not be
     * called except by a HashTree during traversal.
     *
     * @param node
     *            the node currently encountered
     * @param subTree
     *            the HashTree under the node encountered
     */
    void addNode(Object node, HashTree subTree);

    /**
     * Indicates traversal has moved up a step, and the visitor should remove
     * the top node from its stack structure. This is a callback method, and
     * should not be called except by a HashTree during traversal.
     */
    void subtractNode();

    /**
     * Process path is called when a leaf is reached. If a visitor wishes to
     * generate Lists of path elements to each leaf, it should keep a Stack data
     * structure of nodes passed to it with addNode, and removing top items for
     * every {@link #subtractNode()} call. This is a callback method, and should
     * not be called except by a HashTree during traversal.
     */
    void processPath();
}
```

SearchByClass 是访问者的实现，它主要实现了 addNode 方法，并且通过泛型定义了需要访问的 Object 的类型。

```java
@Override
public void addNode(Object node, HashTree subTree) {
    if (searchClass.isAssignableFrom(node.getClass())) {
        objectsOfClass.add((T) node);
        ListedHashTree tree = new ListedHashTree(node);
        tree.set(node, subTree);
        subTrees.put(node, tree);
    }
}
```

StandardJMeterEnginer 的 configure 方法是获取 TestPlan 的数据节点，其中有利用访问者模式访问数据的实际代码，可看出主要是借助 HashTree 自身的 traverse 方法，传入通过 TestPlan.class 构造的新建 SearchByClass 对象。最后使用 testPlan.getSearchResults().toArray()获取访问结果。

```
@Override
public void configure(HashTree testTree) {
    // Is testplan serialised?
    SearchByClass<TestPlan> testPlan = new SearchByClass<>(TestPlan.class);
    testTree.traverse(testPlan);
    Object[] plan = testPlan.getSearchResults().toArray();
    if (plan.length == 0) {
        throw new IllegalStateException("Could not find the TestPlan class!");
    }
    TestPlan tp = (TestPlan) plan[0];
    serialized = tp.isSerialized();
    tearDownOnShutdown = tp.isTearDownOnShutdown();
    active = true;
    test = testTree;
}
```

以 HashTree 为核心的 JMeter 数据存储结构很好的确保运行时检索数据的工作效率，在整个 JMeter 运行源码中处处可见其身影，也是 JMeter 运行重要的存储设计。

以上就是对 JMeter 核心源码进行的简单解读。本次分享主要从 JMeter 启动运行角度，对重点的 NewDriver 类、JMeter 类、JMeterEngine 类以及 HashTree 结构进行了简单介绍，分析了内部组件的交互关系，展示了启动运行指令的执行链路，希望能对大家在扩展开发 JMeter 的相关工作中提供一些思路和帮助。

# nGrinder 压测工具使用教程

◆ 作者：陈　杰

网站或应用接口上线前，往往都需要做压力测试，来确定系统稳定性，考察其性能极限和存在的隐患。本文提供 nGrinder 压测工具的简易使用方法，便于压测脚本的录制与执行。

本文默认用户已经安装 nGrinder，如尚未安装，请访问官网按照提示进行安装。

需要准备的工具有自定义文件 PTS.py，脚本录制工具阿里云 PTS。

自定义文件 PTS.py 内容如下:

```
# -*- coding:utf-8 -*-
# 性能测试框架公共方法
#统计单页 http code 数量
def addHttpCode(statusCode,statusCodeList=[0L,0L,0L,0L]):
    if statusCode < 300:
        statusCodeList[0] += 1
    if 300 <= statusCode < 400:
        statusCodeList[1] += 1
    if 400 <= statusCode < 500:
        statusCodeList[2] += 1
    if statusCode >= 500:
        statusCodeList[3] += 1
    return statusCodeList
#统计所有页 http code 数量
def sumHttpCode(statusCodeList,sumStatusCodeList=[0L,0L,0L,0L]):
    for i in range(len(statusCodeList)):
        sumStatusCodeList[i] += statusCodeList[i]
    return sumStatusCodeList
#统计通过检查点的数量
def sumCheckPointStatus(checkPointStatus,checkPointStatusList=[0L,0L]):
```

```
        if checkPointStatus == 1:
            checkPointStatusList[0] += 1
        else:
            checkPointStatusList[1] += 1
        return checkPointStatusList
```

### 录制工具的安装和使用

因为阿里云 PTS 跟 nGrinder 自带的录制工具极度相似，所以录制工具也是可以通用的，因为生成的脚本略作修改即可使用。这里我们采用阿里云 PTS 脚本录制工具。

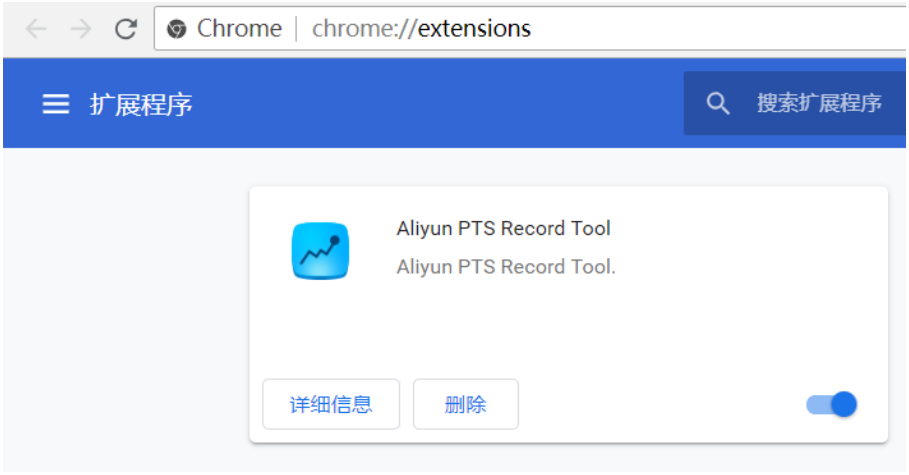
### 安装插件

Chrome 插件下载地址:

http://pts.aliyun.com/common/Aliyun-PTS-Record-Tool_v.0.2.6.4.crx?file=Aliyun-PTS-Record-Tool_v.0.2.6.4.crx

具体安装步骤如下:

1、打开 Chrome 浏览器，URL 框内输入 chrome://extensions/，进入扩展程序安装页面。

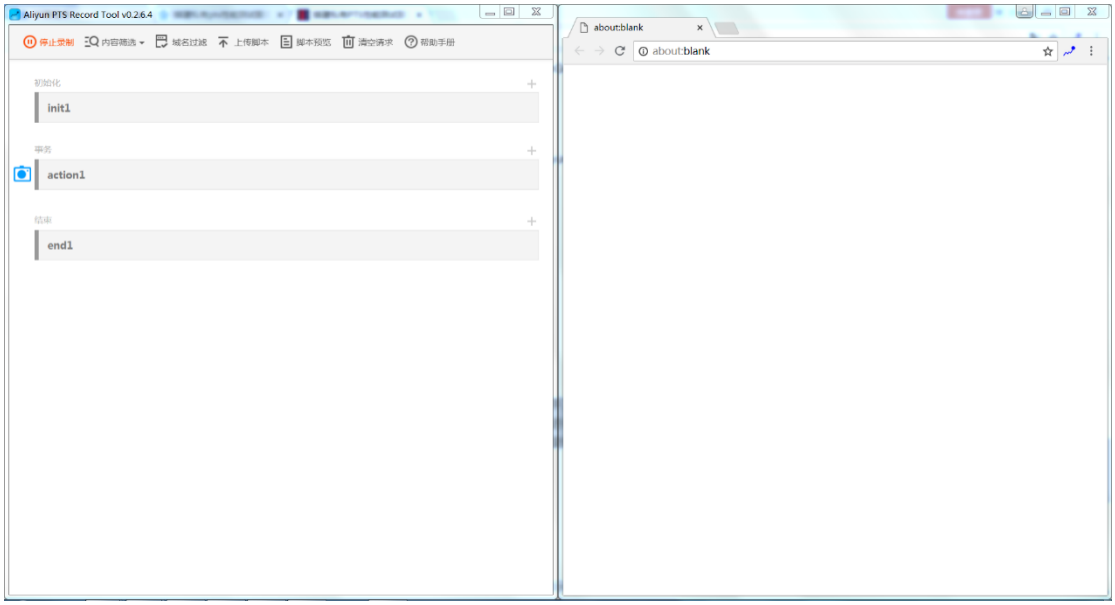



2、将下载的文件拖动至 Chrome 浏览器窗口，浏览器弹出确认新增扩展程序框，点击添加安装。（如果提示程序包无效，将.crx 文件扩展名修改为.rar，解压缩到一个文件夹，打开浏览器插件的开发者模式，点击“加载已解压的扩展程序”，选中刚刚解压出的文件夹）


3、安装完成后浏览器右上角会出现 PTS 的 icon。

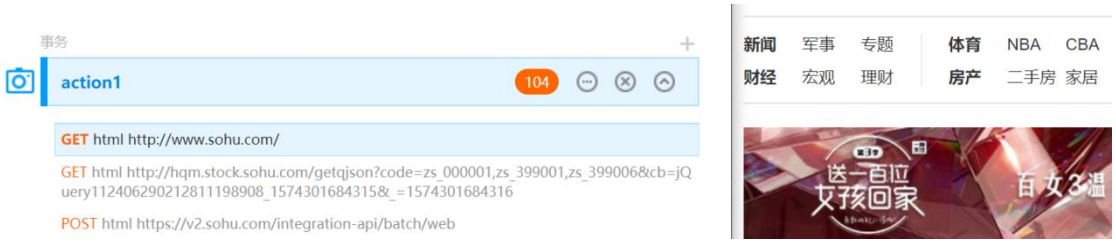
## 录制脚本的作用

对于复杂的业务，例如登录、订购、发帖、退出等业务，由于捕获请求内容或者手工编写脚本工作量稍大，利用 PTS 基于 Chrome 浏览器的插件录制工具可以很大程度上减少编写脚本复杂度。通过使用此工具，用户在被测系统中进行手工操作业务，录制工具会将用户的操作行为进行录制，录制完成以后，自动生成脚本，根据业务规则可能稍微修改一下脚本，就可以运行脚本了。录制下来的脚本模拟了用户真实的操作行为，极大地方便用户的使用。工具安装后打开浏览器点击右上角 PTS 录制工具 Logo，弹出录制工具框和浏览器，如下图所示。

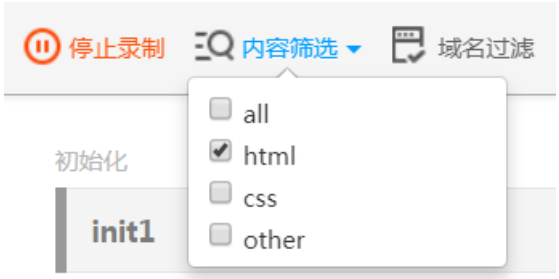



在浏览器输入 URL 进行访问操作，录制工具会自动记录访问操作过程中的 HTTP

请求。




录制工具默认只显示 HTML 类型 HTTP 的录制请求，如需显示其他类型请求请点击内容过滤选择需要显示的类型请求。




录制完成后点击停止录制按钮，如果需要预览录制生成的脚本，请点击脚本预览按钮。


接下来，我们进行一个简单的访问操作测试。

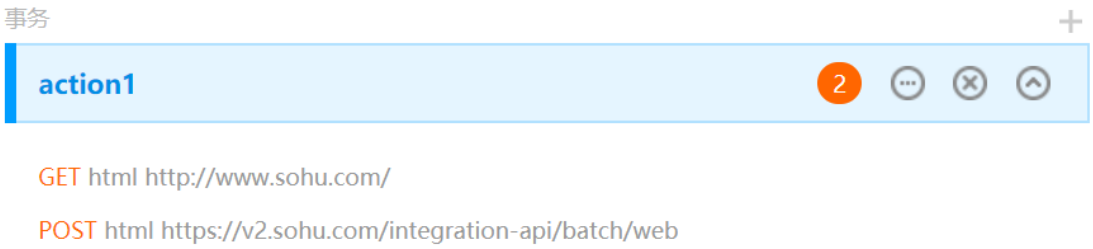
以 http://www.sohu.com 网站为例。

在右边的浏览器中正常输入网址并访问。

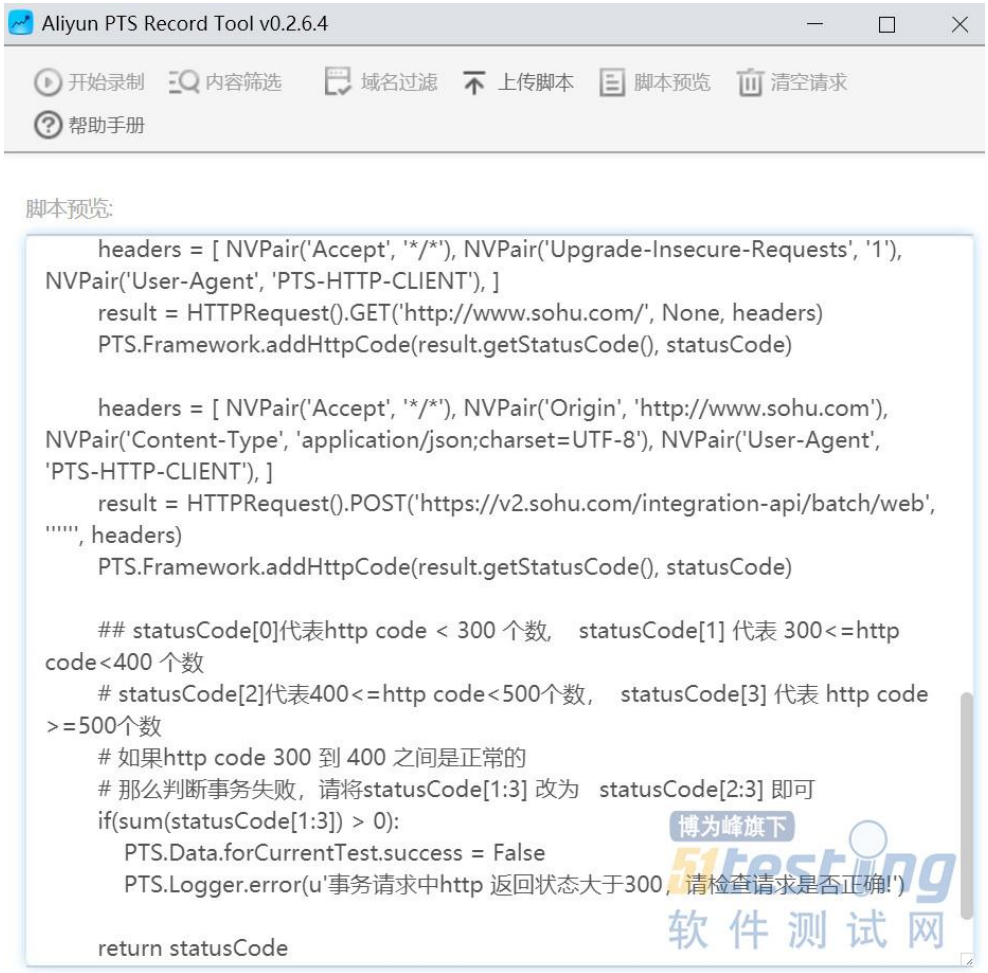

左边的录制工具会自动记录相关接口。因为我们只测试接口，所以把不需要的请求

删除掉，在不需要的请求上右键选择删除。


仅保留需要的接口请求链接。



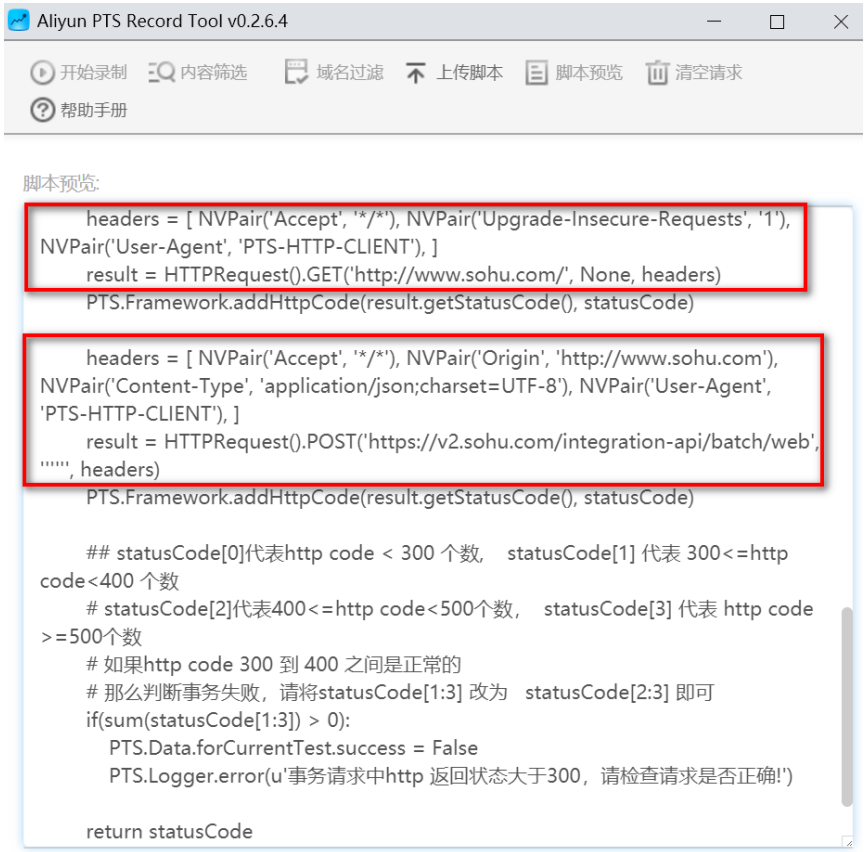

预览下脚本。




至此，脚本就算录制完了。

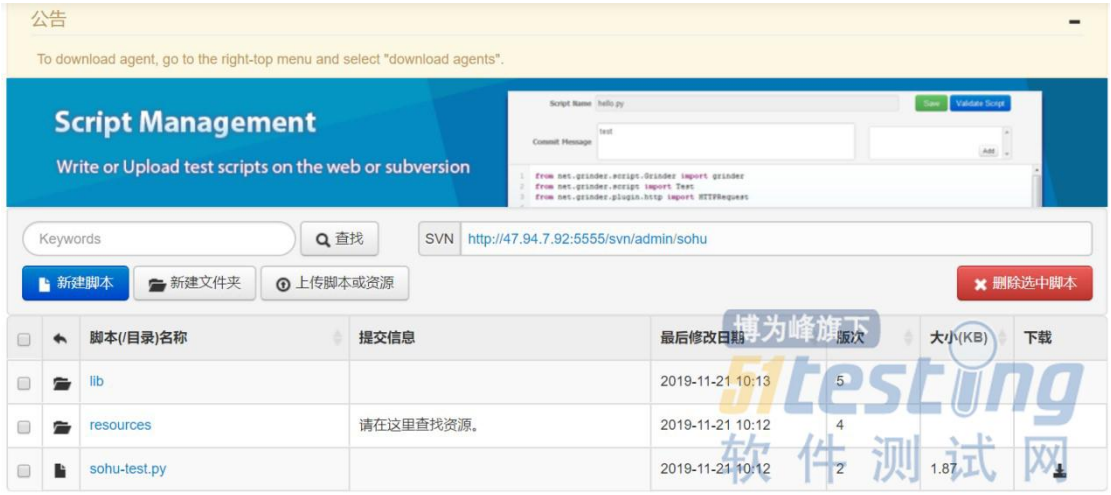
### 修改脚本

录制的脚本，我们并不是都需要，复制出下面这段代码。




在 nGrinder 的脚本管理中新建脚本。

记得勾选上“创建资源和库目录”，这项并不是每次都需要，请视情况而定，因为我们需要引入自定义的模块，之前已经配置好的可以不用再创建。填写脚本名 sohu-test，选择脚本类型为 Jython 后，就可以点击创建按钮创建了。




在 lib 目录中我们需要上传自定义模块的脚本 PTS.py，内容如上篇所述，可根据情况修改和补充。


编辑 sohu-test.py 脚本文件，将内容修改为以下代码。因为脚本跟阿里云 PTS 的结构也不完全相同，阿里云 PTS 所有的执行过程操作函数是在 TestRunner 类里面的，而 nGrinder 所有执行过程操作函数在 TestRunner 类外面，nGrinder 默认生成的脚本也没有对 cookie 的处理，所以需要对脚本做一些修改如下（具体修改请看脚本注释）。

图示标灰的位置替换为录制脚本中复制的代码，注意代码缩进。

```
# -*- coding:utf-8 -*-
# A simple example using the HTTP plugin that shows the retrieval of a
# single page via HTTP.
#
# This script is automatically generated by ngrinder.
#
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
```

```
from net.grinder.plugin.http import HTTPRequest
from net.grinder.plugin.http import HTTPPluginControl
from java.util import Date
from HTTPClient import NVPair, Cookie, CookieModule
import PTS #引入自定义模块

control = HTTPPluginControl.getConnectionDefaults()
# if you don't want that HTTPRequest follows the redirection, please modify the following option 0.
# control.followRedirects = 1
# if you want to increase the timeout, please modify the following option.
control.timeout = 6000
control.useContentEncoding = 1 #允许用户自定义文本编码格式
control.useTransferEncoding = 1 #允许用户自定义传输编码格式
def func():
  statusCode = [0L, 0L, 0L, 0L]   #自定义模块需要，统计 http 响应码个数
  checkPointStatus = [0L, 0L]     #自定义模块需要，统计通过检查点的个数
  #headers 与阿里云 PTS 录制的保持一致
  headers = [ NVPair('Accept', '*/*'), NVPair('Upgrade-Insecure-Requests', '1'), NVPair('User-Agent',
'PTS-HTTP-CLIENT'), ]
  result = HTTPRequest().GET('http://www.sohu.com/', None, headers)
#自定义文本检查点，检查返回值，根据需要修改内容
  #if result.getText().find('"success":true') != -1:
  #     PTS.sumCheckPointStatus(1, checkPointStatus)
  #else:
  #     PTS.sumCheckPointStatus(0, checkPointStatus)
  PTS.sumCheckPointStatus(1, checkPointStatus) #无文本检查内容，则执行该代码
  PTS.addHttpCode(result.getStatusCode(), statusCode)
  return statusCode, checkPointStatus #返回 http 响应码和检查点的统计数据

# Make any method call on request1 increase TPS
Test(1, 'Func').record(func) #设置测试对象函数

class TestRunner:
  # initlialize a thread
  def __init__(self):
        grinder.statistics.delayReports=True
        #自动处理 cookie
        self.threadContext = HTTPPluginControl.getThreadHTTPClientContext()
        self.login_cookies = CookieModule.listAllCookies(self.threadContext)
```

```
# test method
def __call__(self):
    sumStatusCode = [0L, 0L, 0L, 0L]
    #因为每次执行测试 cookie 会被清空，所以这里需要每次重新设置 cookie
    for c in self.login_cookies:
        CookieModule.addCookie(c, self.threadContext)
    httpCode, checkPoint = func() #执行相关操作
    PTS.sumHttpCode(httpCode, sumStatusCode) #统计总的 http 响应码个数

    # if you want to print out log.. Don't use print keyword. Instead, use following.
    # grinder.logger.info("Hello World")

    # statusCode[0]代表 http code < 300 个数，  statusCode[1] 代表 300<=http code<400 个数
    # statusCode[2]代表 400<=http code<500 个数, statusCode[3] 代表 http code >=500 个数
    # 如果 http code 300 到 400 之间是正常的
    # 那么判断事务失败，请将 statusCode[1:4] 改为  statusCode[2:4] 即可
    if sum(sumStatusCode[2:4]) > 0 :
        grinder.statistics.forLastTest.success = 0
        grinder.logger.error(u'事务请求中 http 返回状态大于 300，请检查请求是否正确！')
    elif checkPoint[1] > 0 :
        grinder.statistics.forLastTest.success = 0
        grinder.logger.error(u'事务请求中有 %s 个检查点没通过！' % checkPoint[1])
    else:
        grinder.statistics.forLastTest.success = 1
```

如果有多个相关接口需要做联合测试，需要将相关的接口代码全部粘贴到上述标灰处，注意只粘贴 headers 和 result。在每一个 result 代码行后，都需要增加两行校验代码，如下所示：

```
PTS.sumCheckPointStatus(1, checkPointStatus)
PTS.addHttpCode(result.getStatusCode(), statusCode)
```

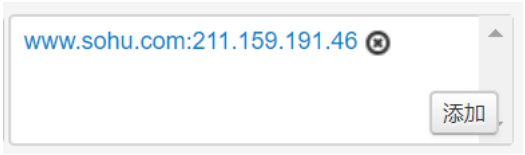
核心示例代码如下：

```
headers = [ NVPair('Accept', '*/*'), NVPair('Upgrade-Insecure-Requests', '1'), NVPair('User-Agent', 'PTS-HTTP-CLIENT'), ]
result = HTTPRequest().GET('http://www.sohu.com/', None, headers)
PTS.sumCheckPointStatus(1, checkPointStatus)
PTS.addHttpCode(result.getStatusCode(), statusCode)
```

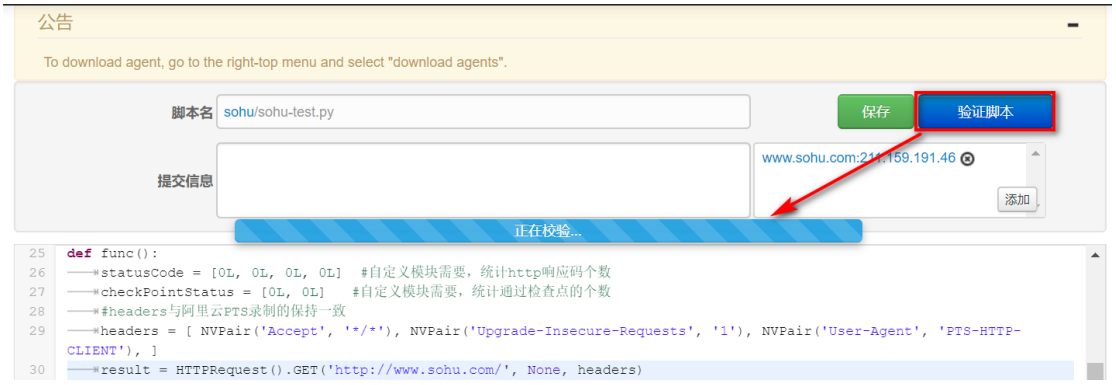
```
headers = [ NVPair('Accept', '*/*'), NVPair('Origin', 'http://www.sohu.com'), NVPair('Content-Type', 'application/json;charset=UTF-8'), NVPair('User-Agent', 'PTS-HTTP-CLIENT'), ]
result = HTTPRequest().POST('https://v2.sohu.com/integration-api/batch/web', """", headers)
PTS.sumCheckPointStatus(1, checkPointStatus)
PTS.addHttpCode(result.getStatusCode(), statusCode)
```

### 调试脚本

先自定义配置 DNS 解析到 nGrinder 服务器，请注意，nGrinder 无法读取配置机器的/etc/hosts，所以当运行 nGrinder 脚本时，需要手动配置 host 信息，一般无需配置。




然后点击“验证脚本”按钮开始调试脚本。




点击页面下面的“+”按钮可展开调试日志，推荐复制调试日志到本地文本编辑器，方便查看。

没提示报错即调试通过，保存脚本。

### 配置测试

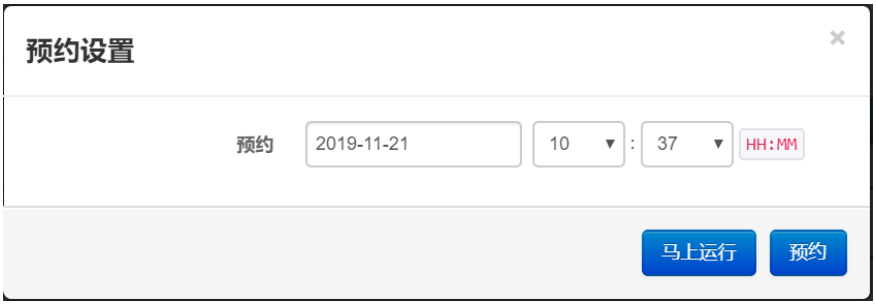
在测试任务管理页面，新建一个测试任务。


配置测试场景。


如图，可配置虚拟用户数的总大小，增加虚拟用户数的方式，测试脚本的版本，需要监控的目标服务器，测试时间等信息。

### 运行测试

配置完后，选择右上方的“保存并运行”按钮，如果是已经运行过的选择“复制并运行”按钮。

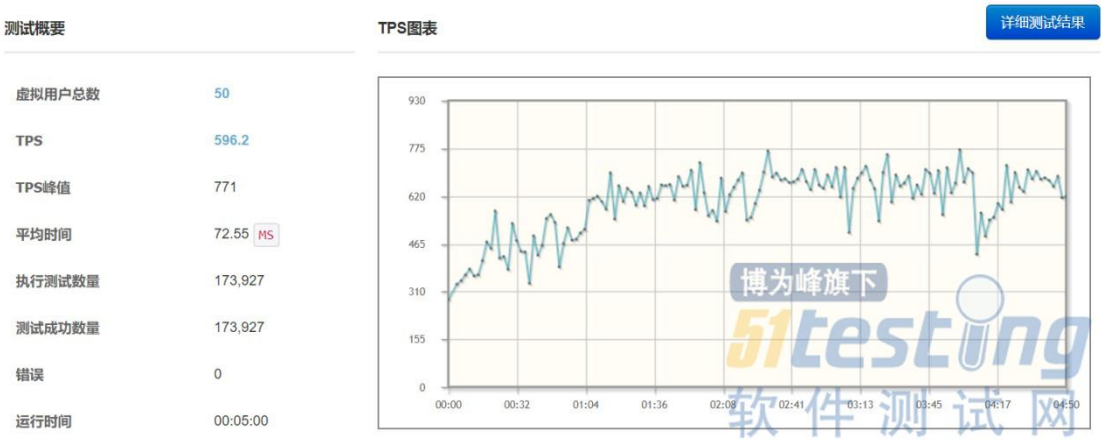



在弹出的窗口中，可以设置定时执行的时间，或者是立即执行。

### 查看结果

启动测试任务后，会看到实时的测试状态，包括目标服务器的 CPU/内存消耗的信息，代理服务器 CPU/内存消耗的信息，实时的 TPS 变化等信息。


测试完成后，会列出概要的测试结果信息，包括平均响应时间，TPS，虚拟用户数，出错率等信息。




点击“详细测试结果”按钮，可查看详细的测试报告及服务器资源消耗情况。

## 测试报告： 测试接口

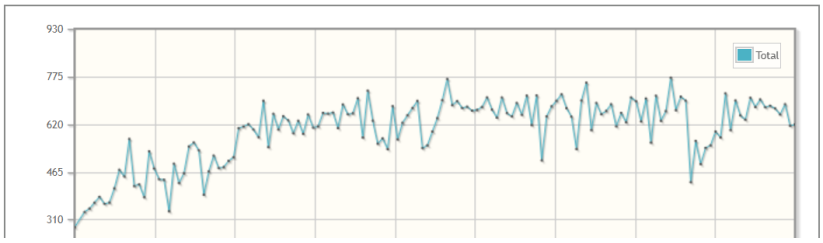
| | |
|---|---|
| 虚拟用户总数 | 50 |
| 代理 | 1 |
| 进程数<br>线程数 | 2 / 25 |
| 忽略取样数量 | 0 |
| TPS | 596.2 |
| TPS峰值 | 771 |
| 平均时间 | 72.55 ms |
| 执行测试数量 | 173,927 |
| 测试成功数量 | 173,927 |
| 错误 | 0 |

| | | | |
|---|---|---|---|
| 开始时间 | 2019-11-21 10:38:31 | 结束时间 | 2019-11-21 10:43:31 |
| 测试时间 | 00:05:00 HH:MM:SS | 运行时间 | 00:05:00 HH:MM:SS |
| 描述 | 测试接口，持续5分钟 | | |

### Performance

下载 CSV文件

TPS

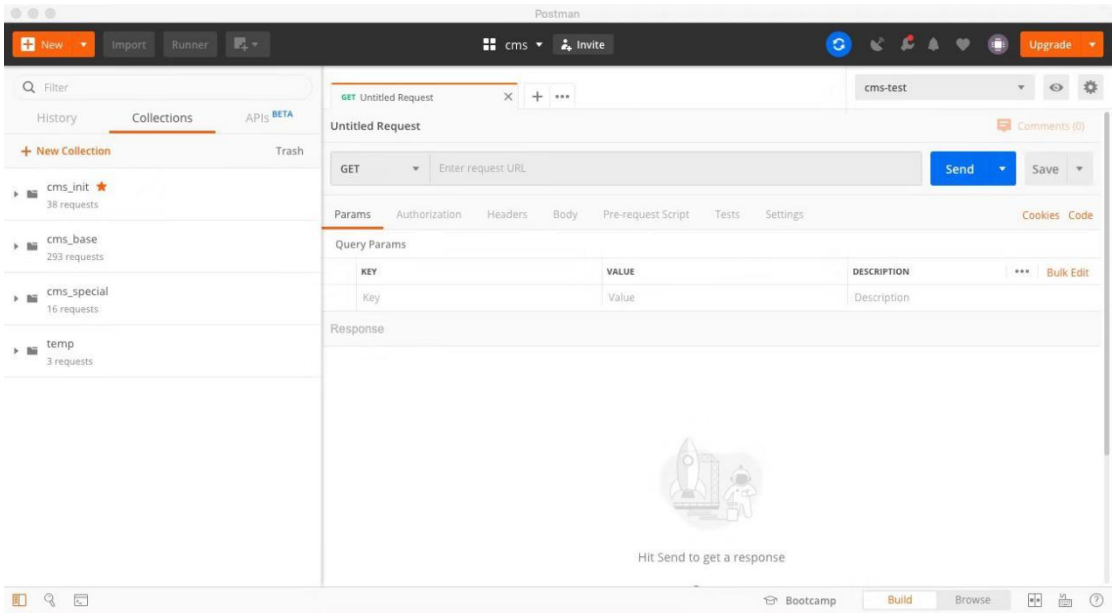


# Postman +Newman+Jenkins+ 钉钉实现持续集成的接口自动化

◆作者：Kayie

## 一、为什么选用 Postman 工具做接口自动化测试

Postman 调试工具无论对于开发和测试小白，还是技术大牛来说应该都耳熟能详，在过去的几年里大家对这款工具应用最广的用途是把当作接口调试的测试工具，它能发送几乎所有类型的 HTTP 请求，操作界面非常简洁美观（大家来欣赏下图），支持抓包，保存历史记录，有用户组管理机制，方便多端同步用例等等。最主要的是各位小伙伴已经对它用于接口调试的功能非常熟悉，已经为我们接下来使用它做持续集成的接口自动化打下了良好的基础。




## 二、Postman 基础知识

相信大家使用 Postman 来调试接口应该是相当熟悉了，所以还不懂使用的同学可以在网上找到相关的资料（非常多），或者直接查阅官网，我建议大家还是要学会看官方

文档，度娘很多教程可能会不全或者已经过时了，官网是最新最全的学习文档，不懂就慢慢看，这篇文章只能带着大家入门，修行只能靠自己（微笑）。

官方文档：https://learning.getpostman.com/docs/postman/launching_postman/

**1、Postman 版本选择**

Postman 分为 chrome 插件版和 native 版本，插件版的有很多功能都受限制，比如：必须要安装扩展才可以操作 cookie，headers 设置受限制，没有 native 版本的 Postman console，非常不方便调试等，不方便我们后续做接口自动化，所以版本必须选择 native 版，版本号用最新的就可以。

**2、Postman 代理抓包**

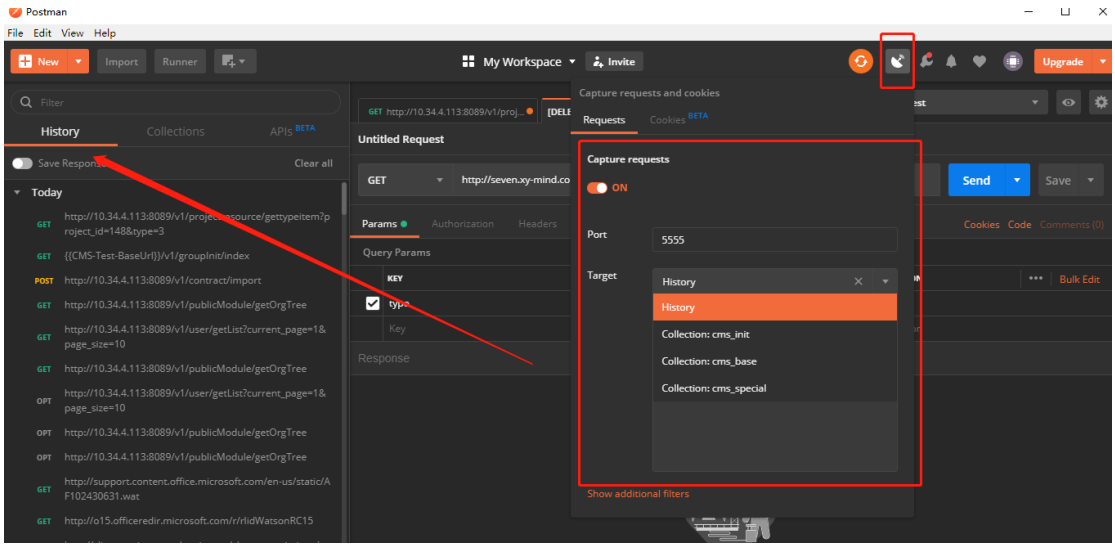
由于 Postman 不是类似 fiddle 这种专门用于抓包的软件，fiddler 软件安装后默认打开抓包功能，默认端口是 8888，但是 Postman 需要自己手动打开代理设置。设置方法如下：

①点击右上角的拦截图标，打开【Capture requests】，设置好 Port 和 Target

注意 Capture requests 按钮一定是要处于 on 的状态；

Port：就是端口，只要不设置系统和浏览器的冲突端口就可以，比如 9999；

Target：就是抓到的 HTTP 包存放的目标地，我一般选择放在 history，方便查看。




②打开浏览器的代理设置，设置相同端口

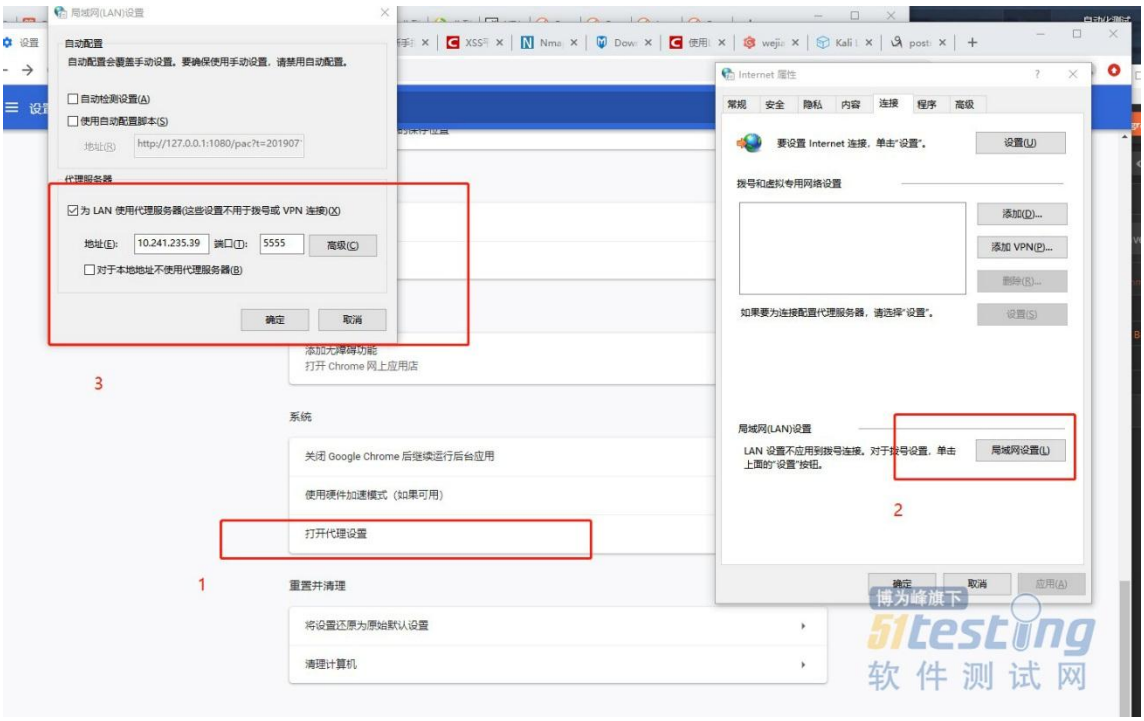
打开浏览器，找到代理服务器，将地址修改为本地机器的 ip 地址，端口设置为第①

步设置的端口号，点击确定即可。

我们演示的这个设置方法是以 PC 端为例子，有些同学需要抓 APP 端的包。设置方法原理也大概相同，大家可以参看官网文档中的设置方法即可。

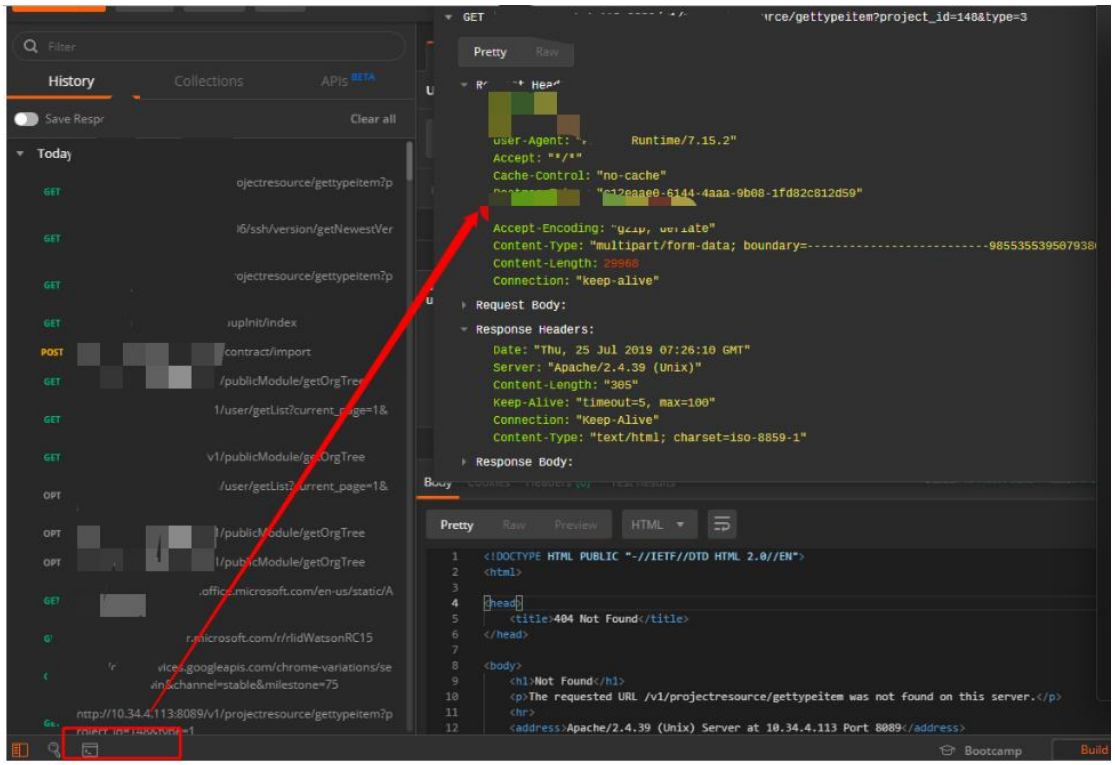
移动端设置方法:

https://learning.getpostman.com/docs/postman/sending_api_requests/capturing_http_requests




### 3、Postman 调试控制台

打开左下角打开【Postman console】控制台，当我们某个测试用例跑不通的时候需要调试，可以打开这个 Postman 的控制台方便查看接口调试的打印信息，当然我们后续用 js 写的脚本可以在控制台中看到输出信息。

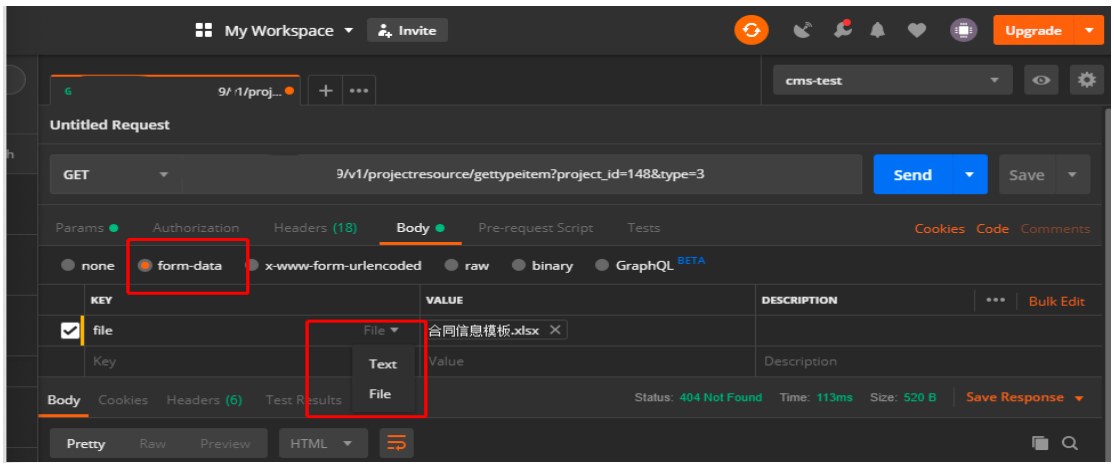


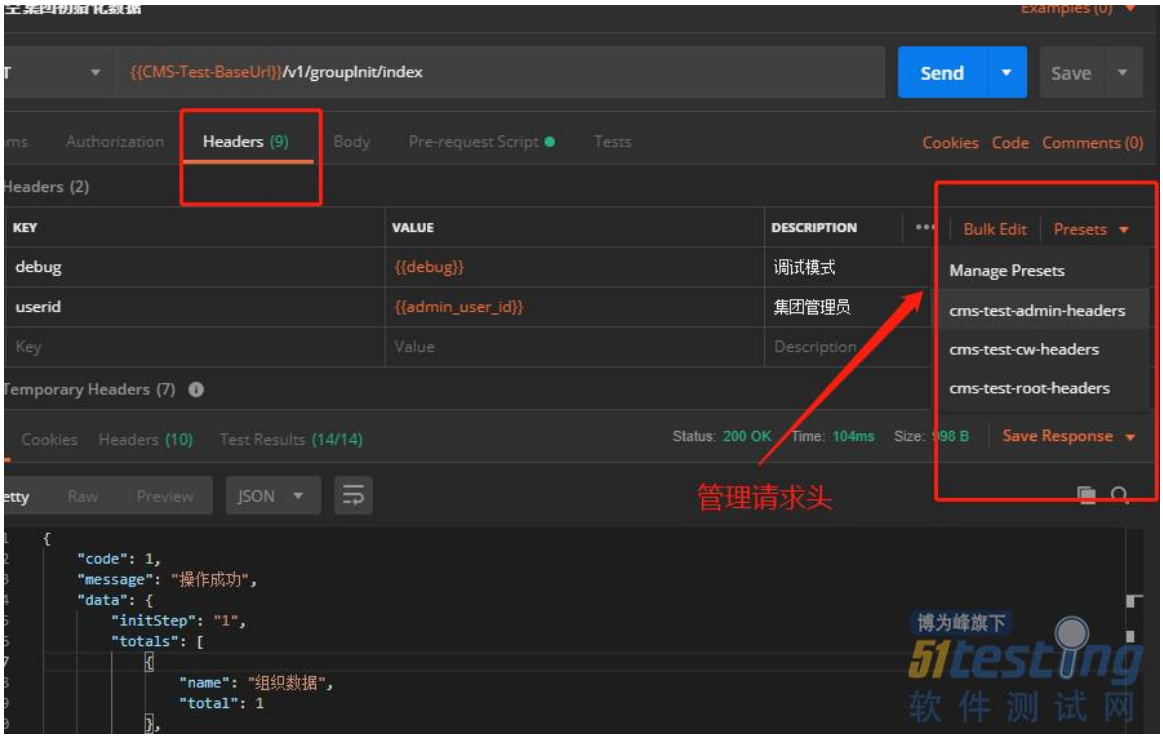

### 4、Postman 发送请求

发送请求是 Postman 最核心的功能，如下图选择好请求方式及接口地址、相应参数，点击发送即可。有两个地方强调下:

①需要发送上传文件的请求，在 body 中选择选择 form-data ，选择 File 选项后即可选择文件

②可以将请求头统一管理，打开任意一个请求 tab，点击 Headers->Presets->Manage Presets 可以设置好请求头信息，下一个请求再需要相同的请求头便可以从这里读取。






## 三、Postman 变量与集合

### 1、什么是变量

首先我们来思考一个问题，为什么要使用变量，如果某个参数值只需要在某个独立接口的使用一次，那么我们可以不使用变量，但是如果变量需要在多个位置重复使用，或者发生接口传递（即前一个接口的返回值用于下一个接口的请求参数），那么借助变量，Postman 就可以实现业务逻辑与测试数据分离，帮助我们建立健壮的测试用例。

### 2、变量的作用域

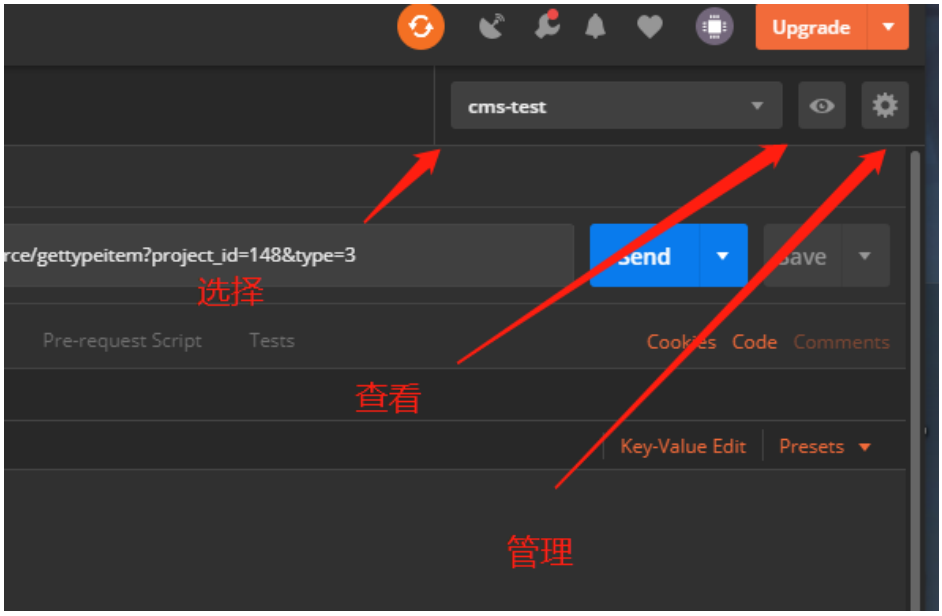
- 全局变量（Global）：主要用于需要在多个位置重复使用的几乎不改变的数据
- 环境变量（environment）：主要用于接口传递的参数
- 局部变量（local）：集合/文件/接口中 某个 js 脚本中的变量
- 数据变量（data）：主要用于并发测试（json/csv 文件）

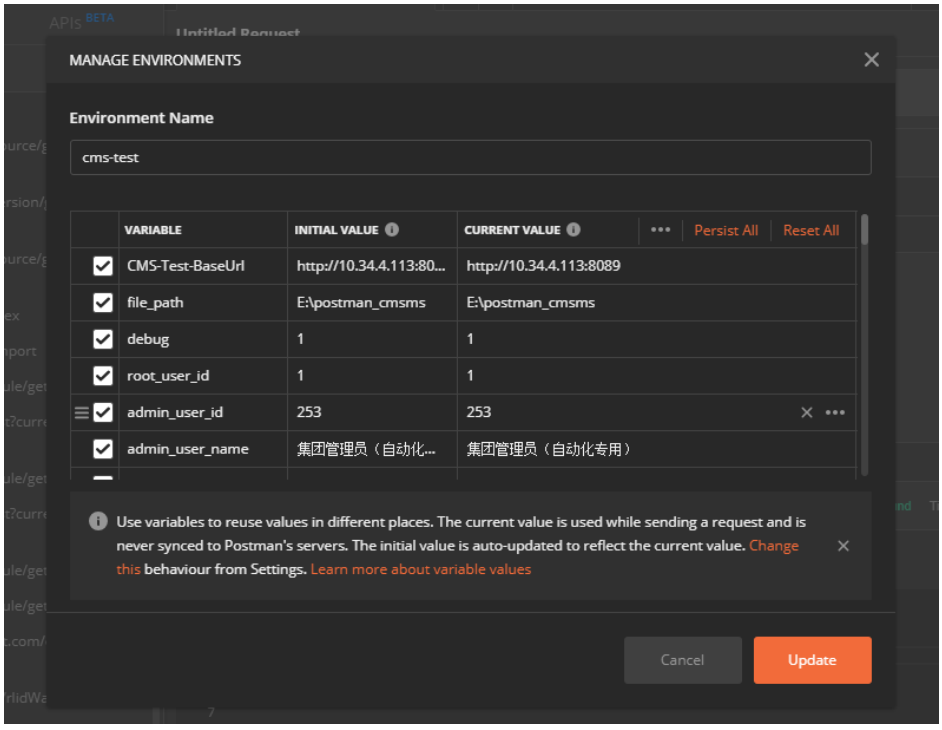
同变量名的情况下，优先级范围：**局部变量>数据变量>环境变量>全局变量**

### 3、如何设置变量

**设置变量的方法有两种：**

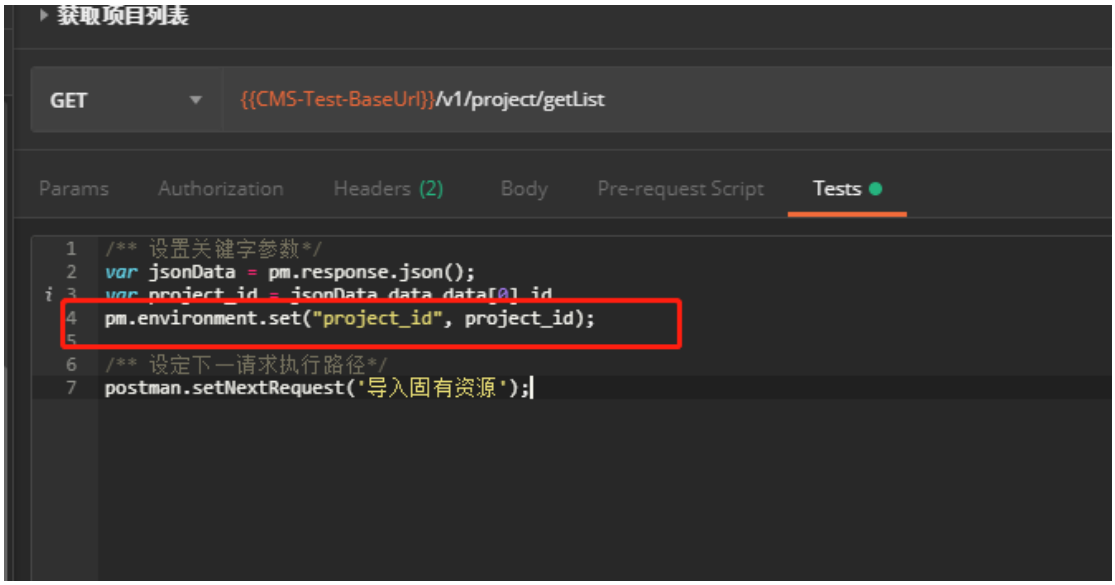
① 在 Postman 的 ui 上进行定义：如下图所示：




② 使用脚本定义，在 Test 选项卡中填入脚本:

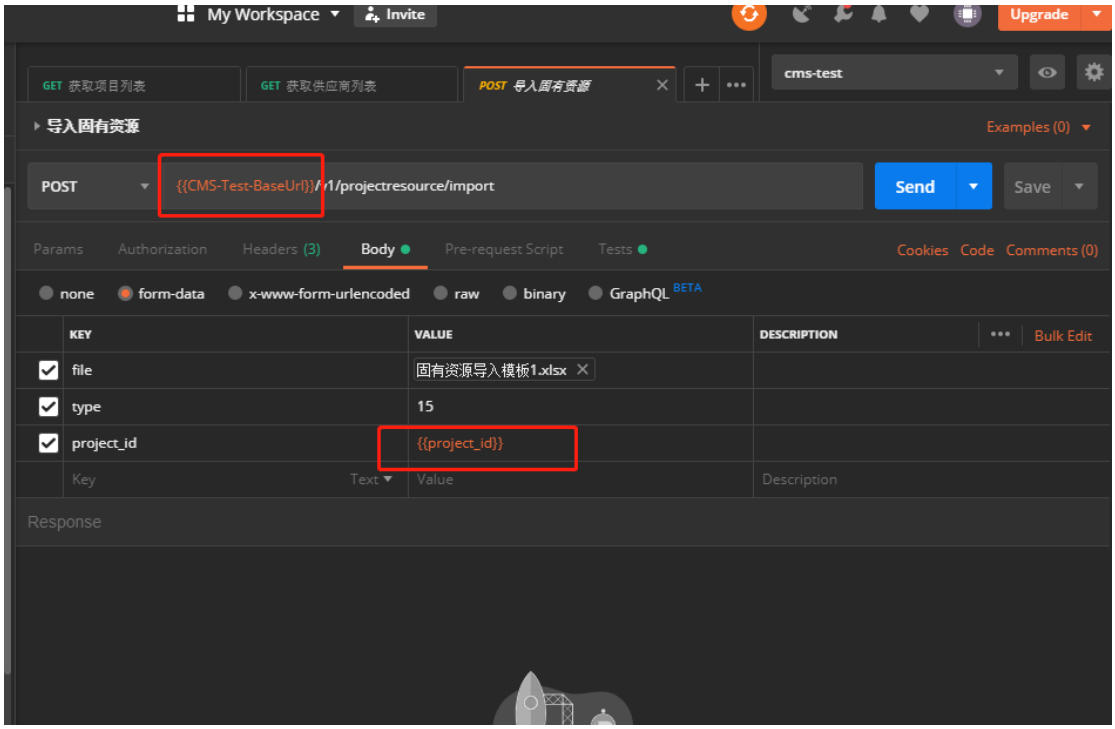
pm.environment.set("new_name", "zhougang");



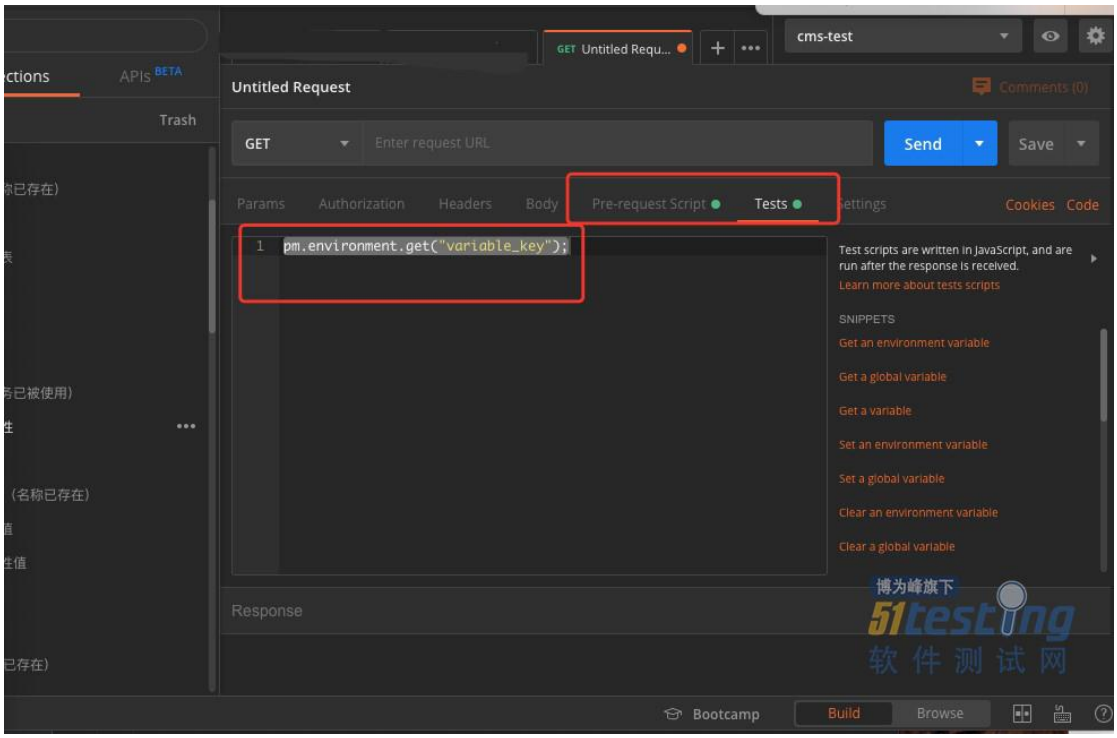

**4、如何使用变量**

使用变量也是有两种写法，用在不同的场景中

① 在 Builder 中使用：{{varname}}，一般用于请求头、请求体、请求地址中




② 在 js 脚本中使用：pm.environment.get("variable_key")，一般用于 Test 或 pre-request Script 选项卡中




## 5、集合

**我们为什么要使用集合以及使用集合的好处在哪？**

①可以组织业务逻辑：分类和存储接口

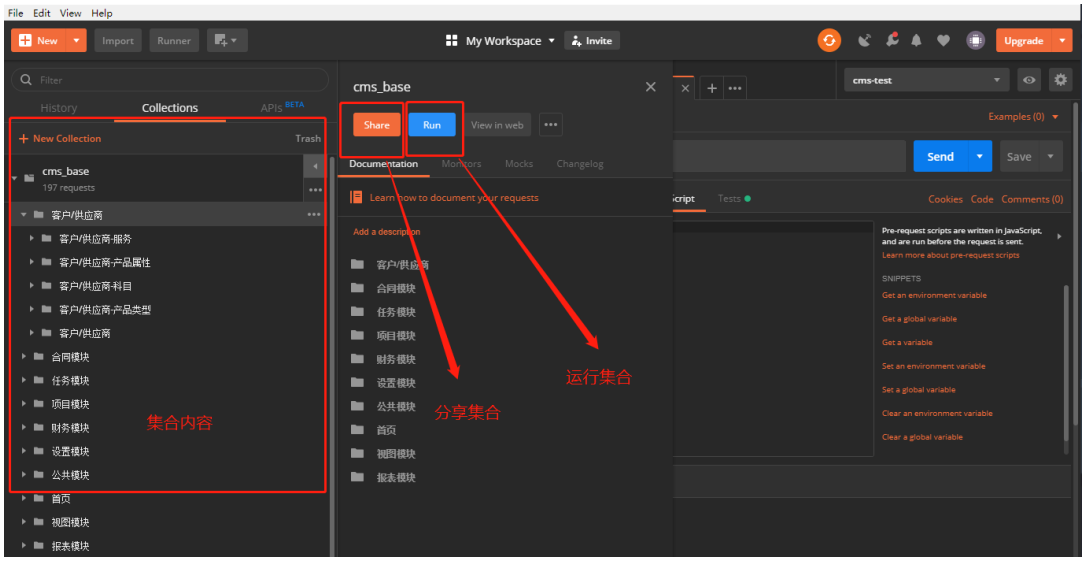
②方便一键运行：在 runner 中选择要运行的集合或者里面的文件夹进行运行

③方便导入导出：导出和导入 json 文件，如使用 newman 在命令行执行；

注意点：集合和变量的导出是分开的。

④方便分享：分享至自己团队的共享目录下

基于以上几点，我们想要写出好维护且高效的接口测试用例，一定要好好利用集合。

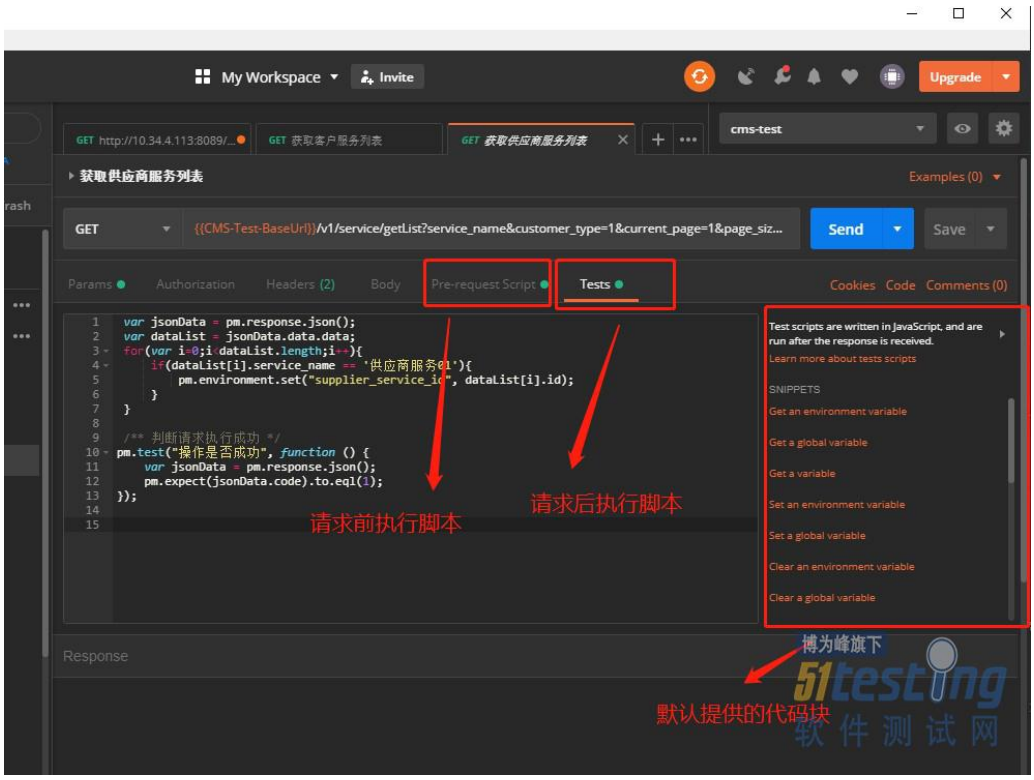



## 四、Postman 脚本应用—接口断言

前三节内容我们了解到了做自动化测试前必须掌握的基础知识，这一节我们进入到接口自动化测试中的核心内容：如何判断一个接口的返回值与期望值相符，也就是我们说的接口断言成功。这个时候 Postman 的测试沙箱 sandbox 就发挥了巨大的作用。官方解释：Postman sandbox 沙箱是一个 JavaScript 执行环境，在编写 pre-request scripts 和 test scripts 编写的脚本在此沙箱中执行（Postman 和 Newman 中都可以使用）

①在请求发送前，在“pre-request script”选项卡下输入脚本，主要用于设置测试需要的测试参数

②在请求发送后，在“test”选项卡下输入脚本，主要用于做断言。




③因此如果我们需要断言一个接口返回是否是预期值，可以参考如下例子:

1）判断 code 是不是返回 200

```
var jsonData = pm.response.json();
pm.test("操作是否成功", function () {
    pm.expect(jsonData.code).to.eql(200);
});
```

2）判断字段是不是跟预期值相等

```
tests["名称是否正确"] = jsonData.data.name === pm.environment.get("customer_contract_name");
```

3）判断返回时间是否在预期 2000ms 内

```
pm.test("接口返回时间是否超过 2s", function () {
    pm.expect(pm.response.responseTime).to.be.below(2000);
});
```

4）执行顺序设定（此代码只在 runner 和 Newman 中生效）:

```
postman.setNextRequest('接口名称');
```

上边的例子都可以在工具的右侧找到事例，大大节省我们编写断言的时间，如果这里没有你需要的，可以查看帮助文档

（https://learning.getpostman.com/docs/postman/scripts/postman_sandbox_api_reference/）

**pm.test**

`pm.test(testName:String, specFunction:Function):Function`

You can use this function to write test specifications inside either the `Pre-request Script` or `Tests` sandbox. Writing tests inside this function allows you to name the test accurately and this function also ensures the rest of the script is not blocked even if there are errors inside the function.

In the following sample test, we are checking that everything about a response is valid for us to proceed.

```
pm.test("response should be okay to process", function () {
    pm.response.to.not.be.error;
    pm.response.to.have.jsonBody('');
    pm.response.to.not.have.jsonBody('error');
});
```

An optional `done` callback can be added to `pm.test`, to test asynchronous functions.

```
pm.test('async test', function (done) {
  setTimeout(() => {
      pm.expect(pm.response.code).to.equal(200);
      done();
  }, 1500);
});
```

**重点强调：**你可以将请求和测试脚本添加到一个集合，一个文件夹，一个请求中，优先级是：请求>文件夹>集合。举个例子，判断返回时间的断言就可以放在集合或者文件中，因为多个接口可以共用这一个断言。

## 五、Postman 接口自动化

前面我们已经做好了各种准备工作，包括抓包、编写接口、存入集合、设置变量、断言等，接下来就可以把我们的接口自动的跑起来。

### 1、使用 Postman 自带的 runner 工具

①按照下图进入 runner 页面，在 runner 中选择要运行的 collection 或者里面的文件夹进行运行

②enviroment:选择用例跑的环境变量集合

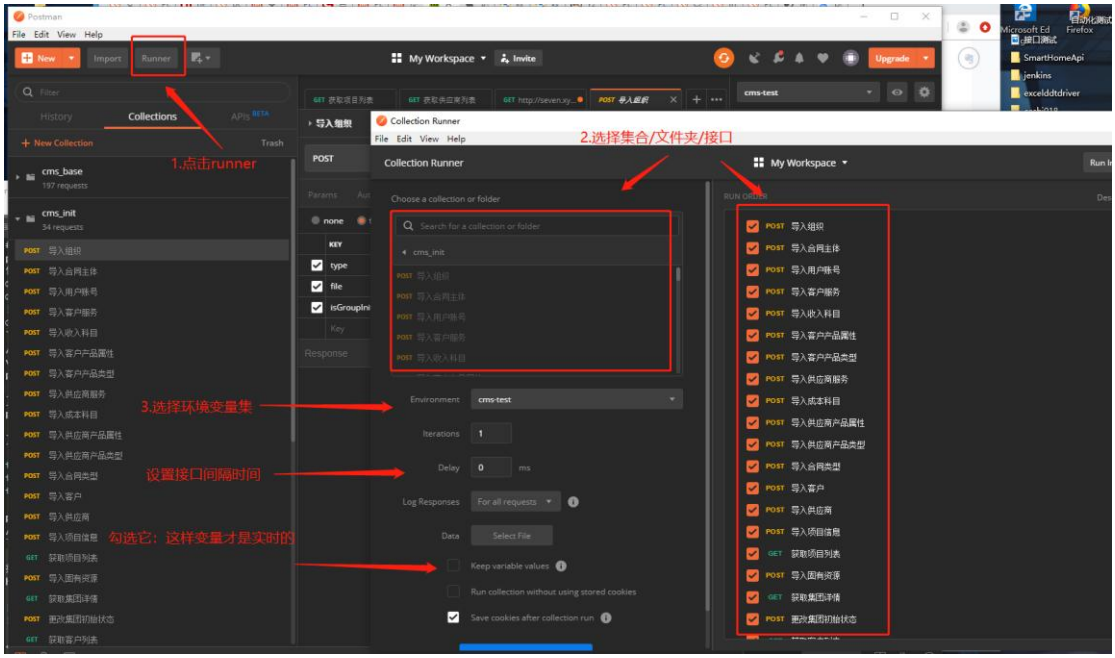
③iterations:接口迭代次数（接口自动化 默认使用 1 就行，1 以上一般拿来做并发测试）

④delay: 每个接口发起请求的的间隔时间（建议 100ms）

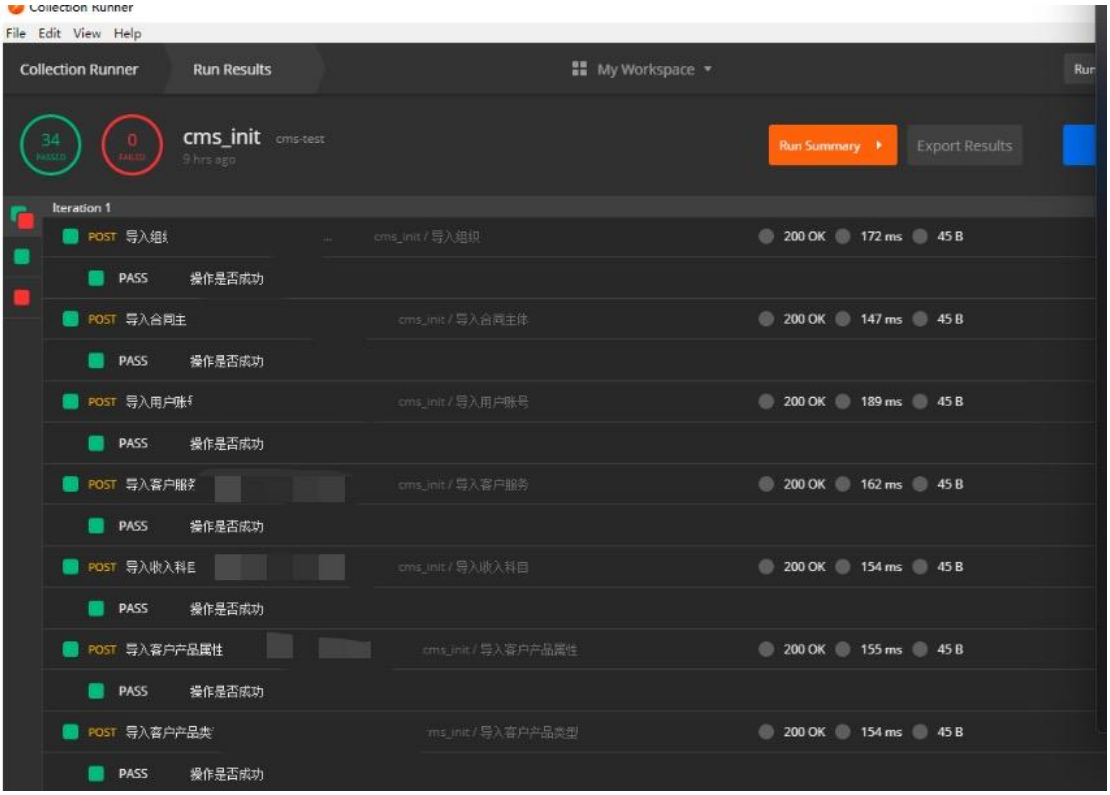
⑤data: 数据文件，可以是 json 格式的，也可以是 csv 格式的（一般用于并发测试）

⑤keep variable values: 重点!!! 必须要勾上，环境变量将会更新

⑥start run：运行结果集，并查看结果页，可导出




运行结果如下显示，示例中 34 条成功，0 条失败。如果有失败的用例可以用控制台查看是请求没发送成功还是断言失败，单独调试。




**2、使用 postman 的插件：Newman**

到了这一步，其实我们已经讲完用 postman 做接口自动化的一个完整过程了。但是

再来思考一个问题，如果我要将这个接口自动化测试加入持续集成的环节，并且再给领导发一份这个测试结果的报告呢？这时候 Postman 的持续集成方案，便是使用自带的插件 Newman + 持续集成工具如 Jenkins 的结合使用

**1）官方说明文档：** https://www.npmjs.com/package/newman

**2）使用 Newman 运行用例并生成测试报告**

①首先需要安装 node.js 环境（同时安装了 npm）

②使用命令行安装 Newman

npm install -g newman

安装导出 html 报告的包： npm install -g newman-reporter-html

③Newman 运行命令脚本，一般只需要指定运行集合、运行环境、间隔时间，数据文件，指定输出报告的模板和位置

原始脚本： newman run xxx.json

添加下述定制化参数:

指定运行环境变量集合： --environment xxx.environment.json

指定输出环境变量集合： --export-environment xxx.environment.json

指定输出报告形式：-r cli,html

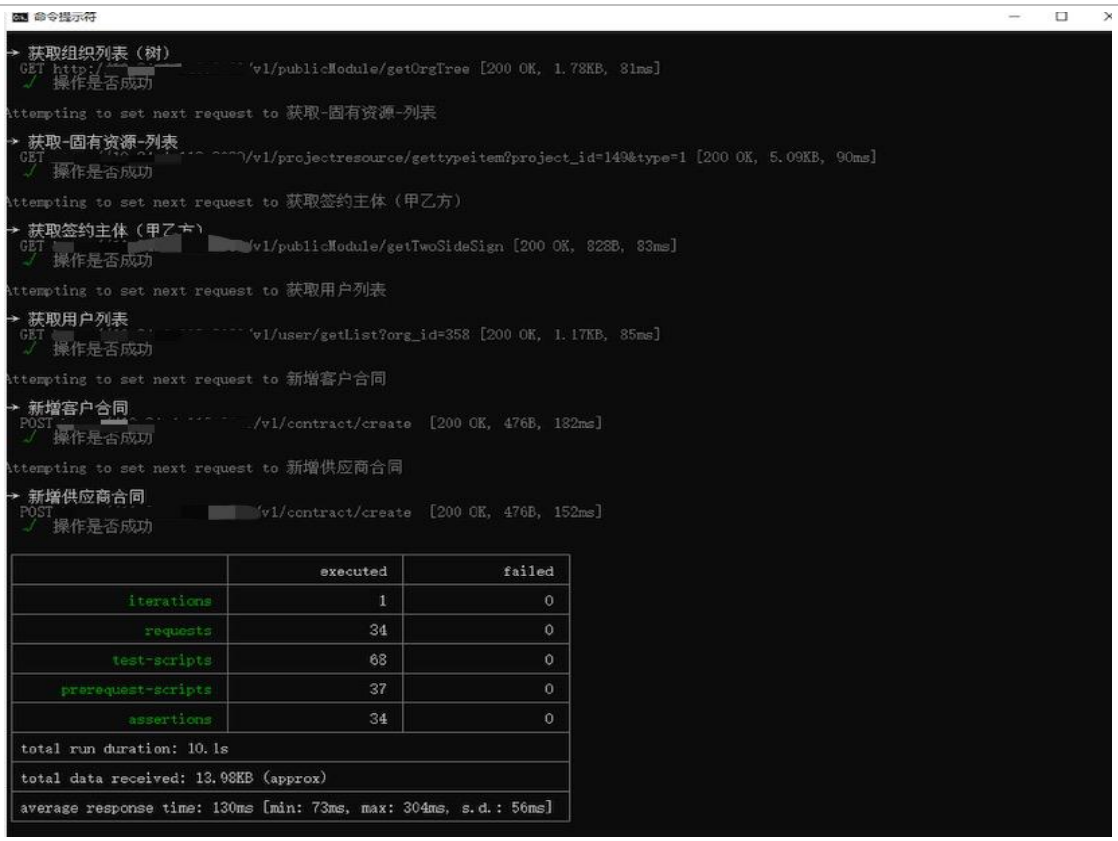
指定输出报告文件： --reporter-html-export xxx.htm

指定接口间隔时间：--delay-request 100

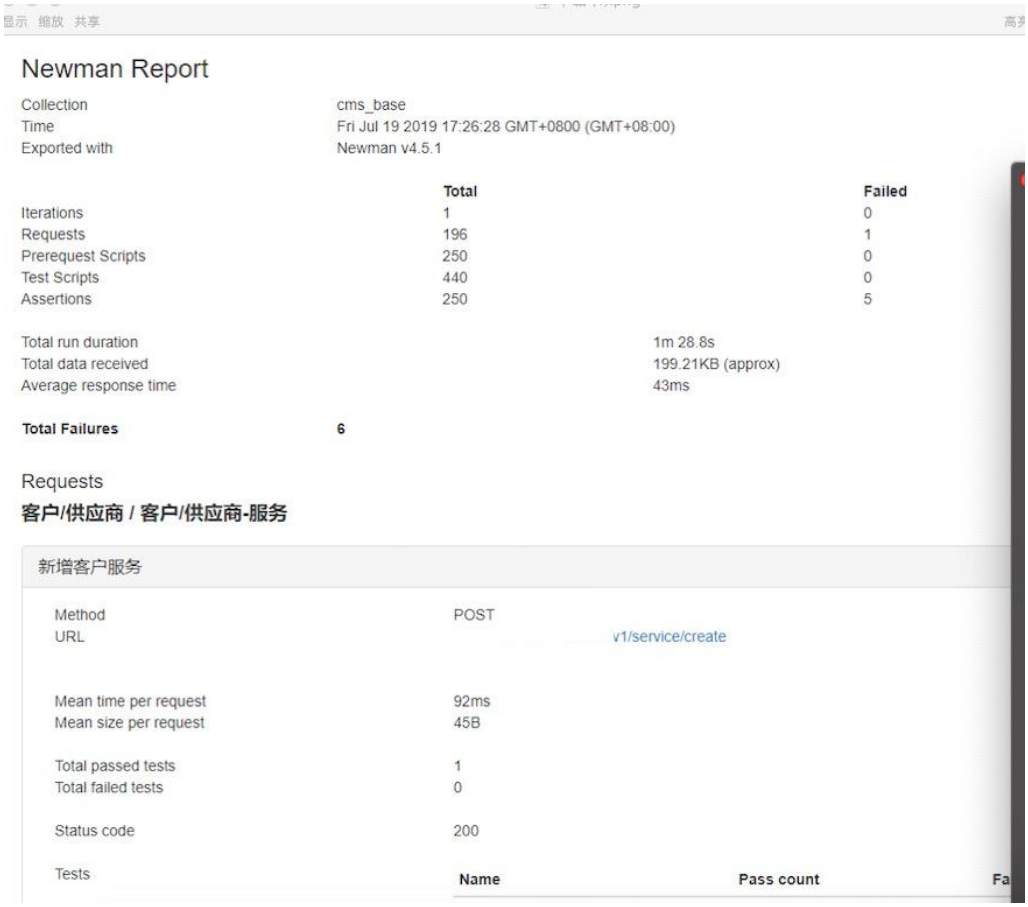
进入脚本所在文件目录，运行最终脚本

newman run cms_init.postman_collection.json --environment cms-test.environment.json --export-environment cms-test.environment.json -r cli,html --reporter-html-export ./reports/cms_init_html_output.htm --delay-request 120

运行结果（命令行）:


命令提示符
→ 获取组织列表（树）
GET http:/ /v1/publicModule/getOrgTree [200 OK, 1.78KB, 81ms]
✓ 操作是否成功
Attempting to set next request to 获取-固有资源-列表
→ 获取-固有资源-列表
GET /v1/projectresource/gettypeitem?project_id=149&type=1 [200 OK, 5.09KB, 90ms]
✓ 操作是否成功
Attempting to set next request to 获取签约主体（甲乙方）
→ 获取签约主体（甲乙方）
GET /v1/publicModule/getTwoSideSign [200 OK, 828B, 83ms]
✓ 操作是否成功
Attempting to set next request to 获取用户列表
→ 获取用户列表
GET /v1/user/getList?org_id=358 [200 OK, 1.17KB, 85ms]
✓ 操作是否成功
Attempting to set next request to 新增客户合同
→ 新增客户合同
POST /v1/contract/create [200 OK, 476B, 182ms]
✓ 操作是否成功
Attempting to set next request to 新增供应商合同
→ 新增供应商合同
POST /v1/contract/create [200 OK, 476B, 152ms]
✓ 操作是否成功
executed failed
iterations 1 0
requests 34 0
test-scripts 68 0
prerequest-scripts 37 0
assertions 34 0
total run duration: 10.1s
total data received: 13.98KB (approx)
average response time: 130ms [min: 73ms, max: 304ms, s.d.: 56ms]

运行结果（html 格式）:


Newman Report
Collection cms_base
Time Fri Jul 19 2019 17:26:28 GMT+0800 (GMT+08:00)
Exported with Newman v4.5.1
Total Failed
Iterations 1 0
Requests 196 1
Prerequest Scripts 250 0
Test Scripts 440 0
Assertions 250 5
Total run duration 1m 28.8s
Total data received 199.21KB (approx)
Average response time 43ms
Total Failures 6
Requests
客户/供应商 / 客户/供应商-服务
新增客户服务
Method POST
URL v1/service/create
Mean time per request 92ms
Mean size per request 45B
Total passed tests 1
Total failed tests 0
Status code 200
Tests
Name Pass count

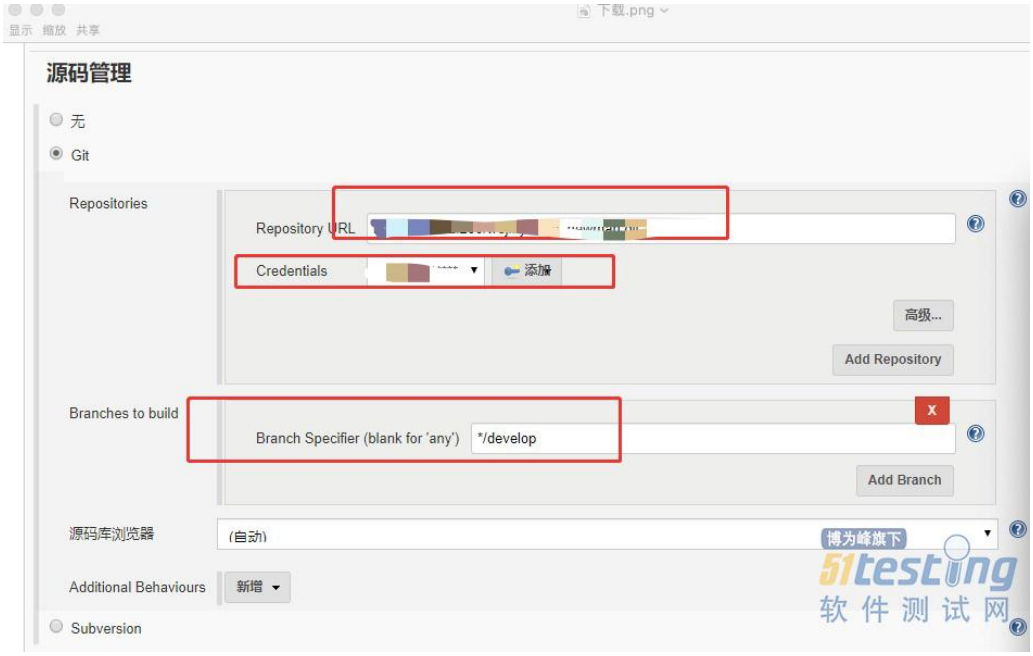
### 3）使用 Newman 与 Jenkins 结合

当我们将代码推送到 git 上后，可以使用 jenkins 直接构建任务即可，省去了脚本打包放到对应服务器再启用对应环境脚本的麻烦。

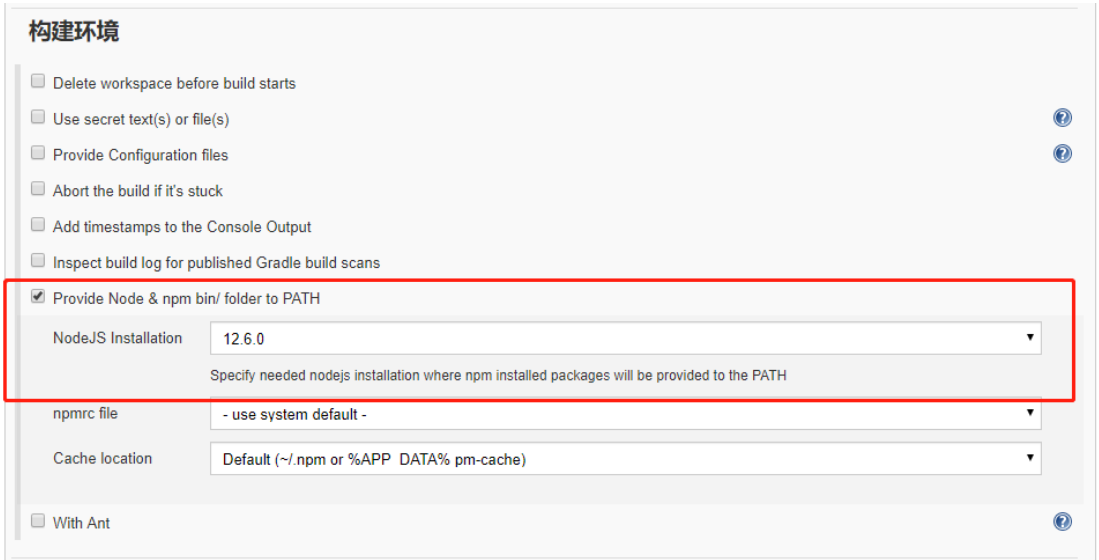
①下载 jenkins 的 war 包，使用 java -jar jenkins.war httpPort=8080 启动，按照配置 jenkins 完成

②在 Jenkins 中安装对应的插件，Git、NodeJS 插件

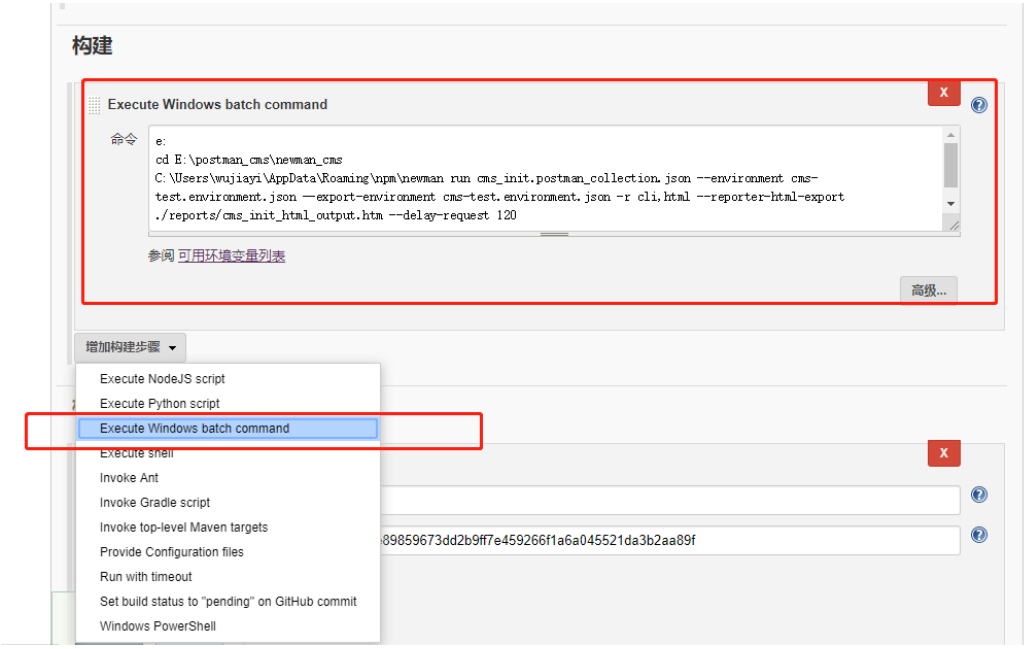
③新建 job，配置相关内容




构建环境配置



构建脚本




选择构建，查看控制台输出信息


## 五、与钉钉机器人结合

钉钉是我们常用于工作沟通的工具，配合钉钉强大的机器人功能，可以让我们在持

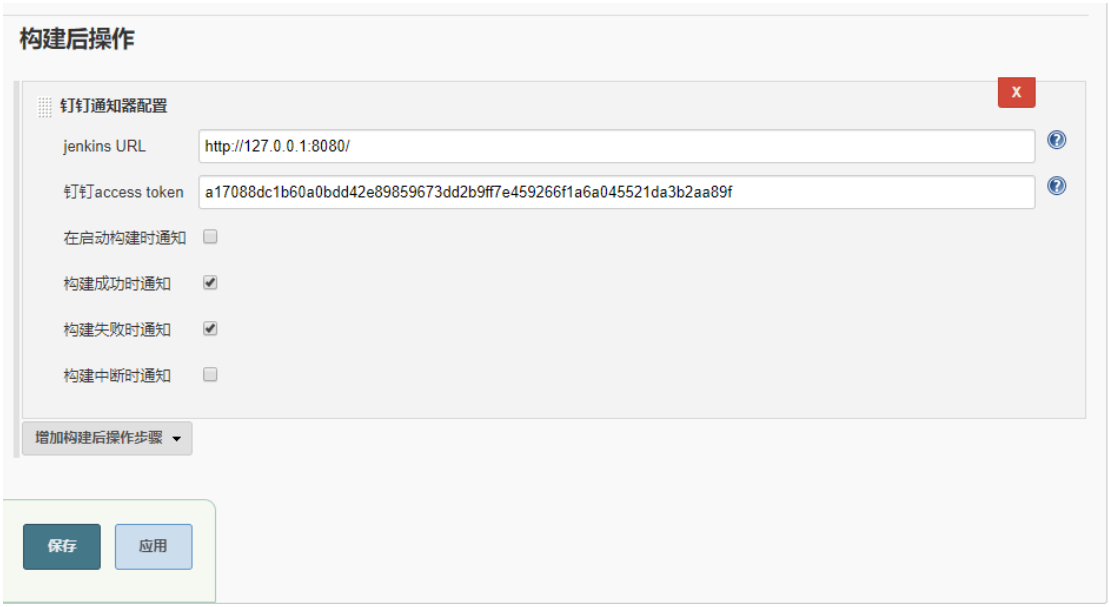
续集成的测试环节中实时、快速接收到测试结果的消息推送。

1、建一个自动化测试群，点击群设置-》群机器人


2、选择自定义机器人


3、添加机器人，获取到 access token


4、将 access token 设置到 Jenkins 中的钉钉通知器插件中（在 Jenkins 插件中下载安装），最后保存即可




5、Jenkins 任务跑完以后，钉钉就会收到消息推送

# WEB 自动化测试之元素定位自动生成探索实践

◆ 作者：张勇 李辉

## 一、WEB 自动化测试

随着 WEB 程序的不断演化，WEB 程序的功能愈发全面和完善。随之而来软件的规模和复杂度也与日俱增，系统内各组件之间的交互也愈发频繁。从而经常出现修改某一组件的代码，另一个组件功能出现意想不到的异常反应。这就要求测试人员执行大量的回归案例来解决这一问题，确保整个应用程序功能正常。

日益复杂的系统和愈发严格用户体验，使得软件测试人员的测试任务愈发繁重。手工测试解决这一矛盾，需要投入大量的人力资源重复执行相似的测试步骤。客观上造成了人力资源的浪费。同时，由于测试人员技能的差异，使得软件测试质量不稳定也不可期。

自动化测试是一种把人为驱动的测试行为转化为机器执行测试的方法。相较于传统的手工测试，这种使用机器代替人工执行测试、比对结果的方法，不仅节约了人力、时间等资源，也提高了测试效率。自动化测试具有的测试覆盖面广、测试流程规范等特点，也有力保障了测试质量的稳定可期。

## 二、基于 ATP 平台的自动化测试框架

ATP (Automation Testing Platform)是一款我行自主研发的自动化测试平台，是一款高效、高质完成大量占用人工成本较高的业务逻辑覆盖测试及回归测试的案例执行及结果分析的工具平台。如图-1 所示，系统可以粗略的分为代码库、Object 库和 ATP 平台三部分。代码库存放一种基于商语言描述的自动化测试脚本；Object 库存放页面元素的定位描述信息；ATP 平台依据 Object 库的元素定位信息和代码库的自动化脚本，进一步翻译

成计算机可执行语言。计算机按步执行，完成测试工作。

在这个过程中，测试人员需要手工录制和建设 Object 库。由于丰富的元素种类、不规范的程序编码、动态展示元素、结构复杂的 DOM 树等客观因素的存在，这就造成测试人员在手工录制和建设 Object 库时困难重重。尤其是给非计算机专业人员带来的巨大困惑和繁重工作。针对这一痛点，本文提出一种元素定位信息自动生成方法。该方法基于 Page 信息表（一种更直观的信息表）自动生成元素定位信息，自动构造 Object 对象库。

如图-1 红色虚线方框内容所示，该元素定位信息自动生成方法以 Page 信息表为输入，自动生成 Object 库中的各界面子库、总库和公共库。Page 信息表、子库、总库、公共库的相对关系如图红框区域所示。

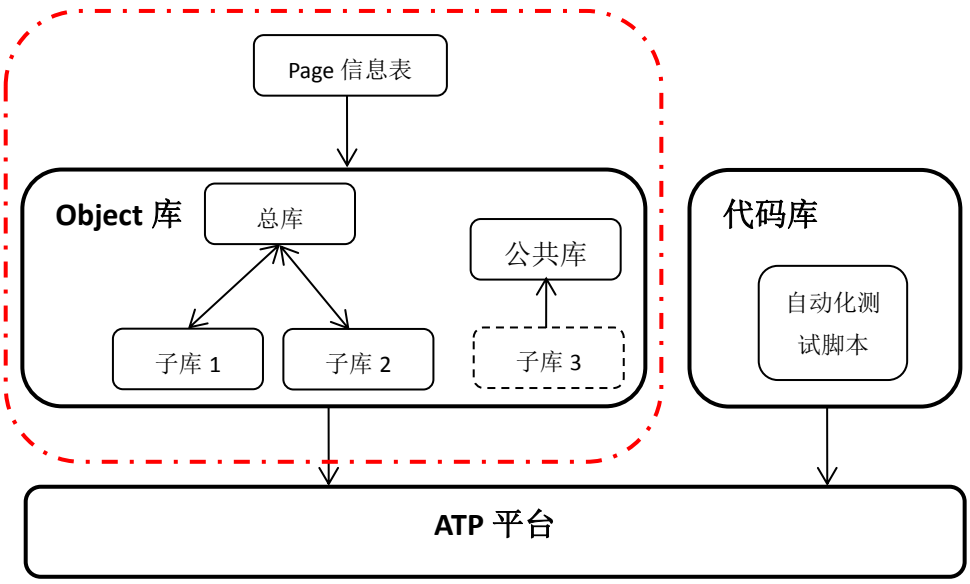



图-1 ATP 自动化测试框架

## 三、元素定位信息自动生成方法

Object 库中元素定位信息描述直接决定了程序能否定位到元素，可以说 Object 库就是自动化测试成功的基石。因此，正确、快捷地获取元素定位信息就变得愈加重要。本文提出的元素定位信息自动生成方法是基于测试人员标注的页面信息自动生成、构建元素定位信息对象库。通过该方法，一、大量减少测试人员抓取元素定位信息的工作量；二、显著减少巨量元素信息的维护成本；三、降低了自动化测试的门槛，测试人员不需关注自动化的底层实现也可进行自动化测试；四、通俗易懂，用更接近交易操作逻辑的方式描述界面信息。使得测试人员摆脱底层实现的困惑，可以更专注于编写测试案例。

### 3.1 Page 信息表

Page 信息表是一种更直观、更接近元素操作的信息描述表，存储了交易界面上元素的名称、操作类型和定位信息。其中，元素名称列填写交易界面上可见的需要手工输入、点击等操作要素名称；元素操作类型列填写该要素会被执行的操作，如：输入、点击、勾选等；定位信息列记录了该元素或其容器的定位信息。即该元素有名字/ID，则填写该元素的名字/ID。该元素无名字/ID，则填写其具有名字/ID 的容器信息。Page 信息表格式如表-1 所示：

表-1 Page 信息表

| 序号 | 元素名称 | 操作类型 | 定位信息 | 备注 |
|---|---|---|---|---|
| 示例 1 | 项目编号 | 文本输入 | projNoSearch | |

Page 信息表采用的这种信息描述格式更直观更易于操作，测试人员观察交易界面，查看页面源码即可填写相关内容。程序会根据各类元素的定位规律，自动转换为 ATP 可接受的定位信息描述。这使得测试人员不用在纠结元素精确定位信息之间的差异，降低了测试人员获取元素信息定位的难度。

Page 信息表以交易为单元，一个交易对应一个 Page 信息表，一个交易窗口对应一个 sheet 页。窗口 sheet 页的前后顺序遵循交易窗口弹出的连贯性原则建立，如在交易主界面点击“明细信息”按钮弹出明细信息界面，则第一个窗口 sheet 也为交易主界面，紧随其后建立“明细信息”窗口 sheet 页。子界面紧随母界面建立，这种排序有助于测试人员理解界面之间的先后关系。

### 3.2 元素定位方法

在 WEB 自动化测试中，常用的 WEB 元素定位方法有 ID 定位、NAME 定位、LINK 定位、PARTIAL_LINK 定位、CSS 定位、XPATH 定位、CLASS 定位和 TAG 定位八种定位方法。同一 WEB 元素可通过多种定位方式获取，不同的定位方式稳定性、可读性不同。为了方便构建 Object 对象库和减少后期的维护成本，元素定位方法应优先选择简单、稳定的定位方式。

分析上述八种定位方式，可知：NAME 和 ID 定位更易懂和更便捷；XPATH 定位功能强大、定位方式灵活，定位精度高；LINK 定位是获取超链接时使用，具有单一不可替代性；PARTIAL_LINK 相较于 LINK 只需要部分文本内容，形式更加灵活。

通过分析，我们制定了定位方法优先级。优先级共分为三层，如表-2 所示：

表-2 定位方法优先级

| | |
|---|---|
| 第一优先级 | NAME 定位、ID 定位 |
| 第二优先级 | XPATH 定位、PARTIAL_LINK 定位 |
| 第三优先级 | CSS 定位、LINK 定位、CLASS 定位、TAG 定位 |

打开页面源码，通过观察元素信息我们发现同类元素的定位存在相同的规律。例如：立项层级右侧的倒三角下拉框按钮（如图-2 蓝色标注），该元素的标签无 ID 信息。且该元素描述信息较少，直接定位存在困难。通过观察其父节点的兄弟节点<input name=”projEstLvl”>，我们发现该元素存在 ID 且两个元素相对位置固定。故可以通过父兄节点元素 ID（projEstLvlSearch）+相对位置（../SPAN[2]/SPAN[1]")）定位倒三角下拉框。

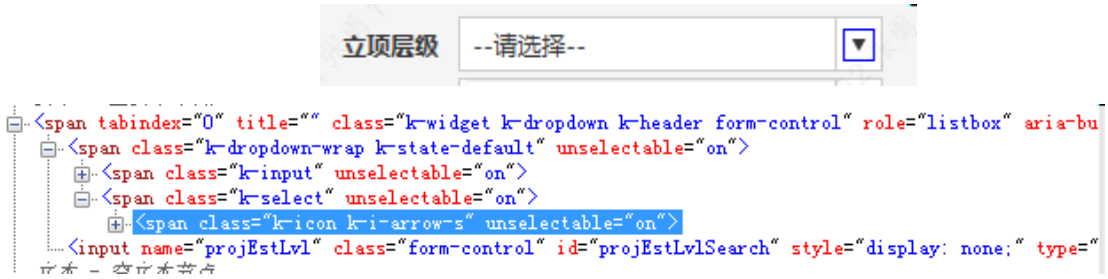



图-2 立项层级（上）及其源码（下）

诸如此类的元素定位规律还有很多，我们结合元素定位方法优先级表和元素分布规律，对不同的分布现象采取了不同的定位方法。如表-3 所示：

表-3 元素规律表

| 序号 | 现象 | 规律 | 定位方法 | 登记原则 |
|---|---|---|---|---|
| 1 | 各交易主界面都有查询、重置、关闭、新增、提交、保存等按钮。 | 按钮名称在交易主界面名字唯一 | NAME 定位 | 登记元素 NAME |
| 2 | 各交易主界面通常都有新增、明细信息、报表下载等按钮。 | 元素都有 ID 且在主界面唯一 | ID 定位 | 登记元素 ID |
| 3 | 子界面的项目编号、保存等要素在主界面也存在。 | 子界面元素 ID 在子界面唯一 | 子界面 ID+元素 ID 定位 | 登记元素 ID |
| 4 | 起始立项日期，截止立项日期等日期元素存在两种输入方式：文本直接输入和按钮点击输入。 | 日期文本框有 ID 且日期按钮控件无 ID | 日期文本框 ID+文本直接输入 | 登记日期文本框 ID |

| | | | | |
|---|---|---|---|---|
| 5 | 项目阶段、项目状态等下拉框要素文本框被设置为不可输入且右侧下拉框无 ID。 | 文本有 ID 且文本框与右侧下拉框相对位置固定 | 文本 ID+相对位置 | 登记文本 ID |
| 6 | 项目显示表格中的记录 ID 是动态生成。 | 表格容器有 ID | 容器 ID+CSS 定位 | 登记容器 ID |
| 7 | 机构树等列表值无元素 ID。 | 列表值唯一 | PARTIAL_LINK 定位 | 登记列表值 |

### 3.3 自动生成元素定位信息

如图-2 所示，测试人员点击打开交易的主界面和子界面，观察页面输入要素的信息。将这些信息填写到一个 Page 信息表中。程序以 Page 信息表为输入，按照元素定位规律生成各界面对应的子库。部分元素组件（如：机构代码树等）因被大量交易整体复用，则直接生成到公共库中；部分界面（如：复核界面等）作为一个整体被其他交易调用，则从子库中移除，转入到公共库中。最后，程序扫描各子库和公共库的记录，将元素定位信息逐一登记到总库中，相似定位信息登记为一条记录。当元素定位信息发生改变时，测试人员通过修改总库中的元素定位信息，程序自动更新各子库、公共库中对应的元素定位信息，从而减少了维护成本。

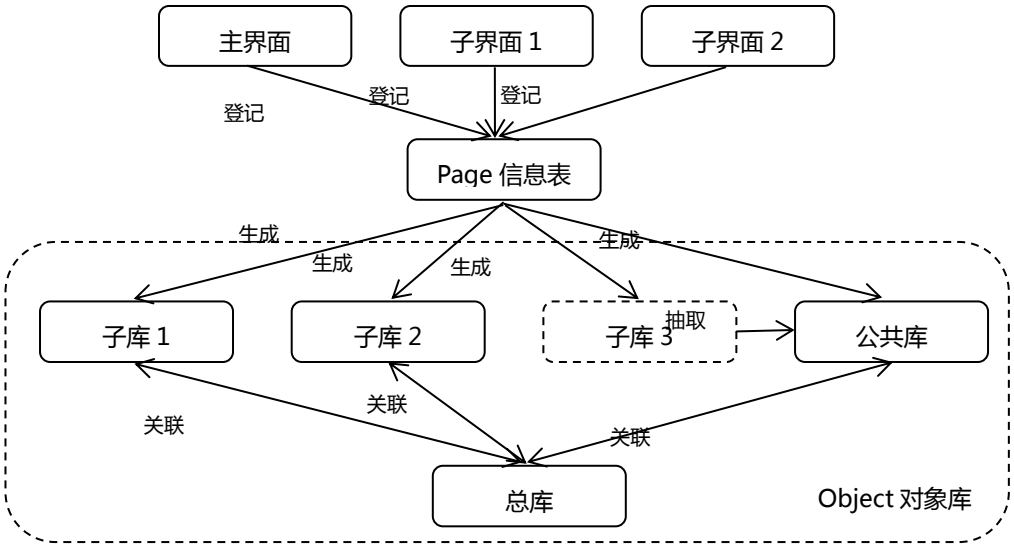



图-2 元素定位信息自动生成流程

我们用固定资产管理系统的房地产盘盈交易主界面为例，详细演示元素定位信息自动生成流程。如图-3 所示，观察房地产盘盈交易主界面发现有项目编号、项目名称等文本元素，有查询、重置、明细信息等公共按钮，有明细建设类型、立项层级等下拉列表，有项目所属机构代码等公共组件，有记录显示表格等元素。


图-3 房地产盘盈主界面

测试人员将观察到的元素名称填写到 Page 信息表元素列中；根据测试人员手工测试时，该元素的输入方式选择操作类型；查看页面源码根据表-3 的登记原则填写定位信息，如用 Chrome 浏览器打开网址，按 F12 弹出页面源码，通过左上角的查看按钮选中元素，查看该元素信息。在定位信息列登记该元素的 NAME/ID 或其父兄的 ID。如表-4 所示：

表-4 房地产盘盈主界面的 Page 信息表

| 序号 | 元素名称 | 操作类型 | 定位信息 | 备注 |
|---|---|---|---|---|
| 1 | 项目编号 | 文本直接输入 | projNoSearch | |
| 2 | 项目名称 | 文本直接输入 | projNamSearch | |
| 3 | 项目所属机构代码 | 按钮选择输入 | fbNamAndCdProjOwnOrgCdSearch | |
| 4 | 明细建设类型 | 下拉列表输入 | dtalBldTypSearch | |
| 5 | 立项层级 | 下拉列表输入 | projEstLvlSearch | |
| 6 | 项目阶段 | 下拉列表输入 | projPhasSearch | |
| 7 | 项目状态 | 下拉列表输入 | projStsSearch | |
| 8 | 批复文号 | 文本直接输入 | apvlRefNoSearch | |
| 9 | 起始立项日期 | 日期文本输入 | start | |
| 10 | 截止立项日期 | 日期文本输入 | end | |
| 11 | 是否贫困县 | 下拉列表输入 | searchForm | |

| | | | | |
|---|---|---|---|---|
| 12 | 查询 | 按钮 | 查询 | |
| 13 | 重置 | 按钮 | 重置 | |
| 14 | 明细信息 | 按钮 | 明细信息 | |
| 15 | 关联合同查询 | 按钮 | purchaseHeTong | |
| 16 | 关联资产查询 | 按钮 | purchaseZiChan | |
| 17 | 下载报表 | 按钮 | downloadExcel | |
| 18 | 附件列表 | 按钮 | fileList | |
| 19 | 进度查询 | 按钮 | progressBtn | |
| 20 | 搜索显示列表 | 表格 | searchGrid | |

填写完成房地产盘盈的 Page 信息表后，程序依据元素定位信息规律，自动转换生成该界面的子库定位信息。然后，程序根据子库和公用标识，生成总库和公共库信息。房地产盘盈主界面元素对象库如表-5 所示。

表-5 房地产盘盈主界面对象库部分元素展示

| 序号 | 对象名* | 对象类型* | 对象定位描述（自动化）* | 备注 |
|---|---|---|---|---|
| 1 | 房地产盘盈主界面 | 页面 | Browser("CreationTime:=1").Page("title:=") | |
| 2 | 项目编号 | 文本框 | WebEdit("xpath:=//INPUT[@id='projNoSearch']") | |
| 3 | 项目名称 | 文本框 | WebEdit("xpath:=//INPUT[@id='projNamSearch']") | |
| 4 | 项目所属机构代码 | 超链接 | Link("xpath:=//DIV[@id='fbNamAndCdProjOwnOrgCdSearch']/../A[1]") | |
| 5 | 明细建设类型 | 下拉框 | WebElement("xpath:=//DIV[@id='dtalBldTypSearchl']/../SPAN[2]/SPAN[1]") | |
| 6 | 立项层级 | 下拉框 | WebElement("xpath:=//DIV[@id='projEstLvlSearch']/../SPAN[2]/SPAN[1]") | |
| 7 | 起始立项日期 | 文本框 | WebEdit("xpath:=//INPUT[@id='start']") | |
| 8 | 截止立项日期 | 文本框 | WebEdit("xpath:=//INPUT[@id='end']") | |
| 9 | 查询 | 按钮 | WebButton("name:=查询") | |
| 10 | 重置 | 按钮 | WebButton("name:=重置") | |
| 11 | 明细信息 | 按钮 | WebButton("name:=明细信息") | |
| 12 | 搜索显示列表 | 表格控件 | WebTable("xpath:=//DIV[@id='purchaseGrid']/*/TABLE[@role='grid'][1]") | |

表-5 总库部分元素展示

| 序号 | 元素 ID | 元素类型 | 元素位置 1 | 元素位置 2 | 元素位置 3 |
|---|---|---|---|---|---|
| 1 | projNoSearch | 文本类型 I | 房地产盘盈主界面- | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | 项目编号 | | |
| 2 | ivtPftEttAddr | 文本类型 II | 房地产盘盈新增-盘盈房产地址 | | |
| 3 | fbNamAndCdProjOwnOrgCdSearch | 机构树 | 房地产盘盈主界面-项目所属机构代码 | | |
| 4 | dtalBldTypSearchl | 下拉框 I | 房地产盘盈主界面-明细建设类型 | | |
| 5 | projEstLvlSearch | 下拉框 I | 房地产盘盈主界面-立项层级 | | |

元素类型用于区分同一类型操作不同的要素。如：项目编号和盘盈房产地址都是文本框，但是项目编号是单行文本框，只能输入一行文本字符，盘盈房地产地址是多行文本框，支持输入多行文本字符。属于操作不同的两种元素，故分别标记为文本类型 I 和文本类型 II 进行区分。

表-6 公共库部分元素展示

| 序号 | 对象名* | 对象类型* | 对象定位描述（自动化）* | 备注 |
|---|---|---|---|---|
| 1 | 搜索机构树 | 树形控件 | WebElement("html tag:=SPAN","innertext:=99H999 - 总行") | |
| 2 | 搜索机构树_确认 | 按钮 | WebButton("xpath:=//DIV[@id='fbNamAndCdProjOwnOrgCdSearchCommonTreeWindow']/DIV[2]/DIV[1]/BUTTON[3]") | |
| 3 | 搜索机构树_清空页面元素 | 按钮 | WebButton("xpath:=//DIV[@id='fbNamAndCdProjOwnOrgCdSearchCommonTreeWindow']/DIV[2]/DIV[1]/BUTTON[2]") | |
| 4 | 搜索机构树_取消 | 按钮 | WebButton("xpath:=//DIV[@id='fbNamAndCdProjOwnOrgCdSearchCommonTreeWindow']/DIV[2]/DIV[1]/BUTTON[1]") | |

在元素对象库系统中有总库、子库和公共库三种类型的对象库。由于大量的交易会复用同一个元素对象，这些元素对象分散在不同的子库中。为了方便管理这些元素设置总库，总库中登记一条该元素定位信息，并记录存储该元素的子库信息。如：“房地产盘盈主界面”子库中存储了“项目编号”元素，则元素位置列登记为“房地产盘盈主界面_项目编号”。若有其他子库也存储了“项目编号”元素，则按照“子库名_元素名”，如表-5 所示。

由于部分元素、子窗口整体被大量复用，为了减少重复录制的次数和提高元素复用率。将此类元素、子窗口定义为公共对象。如表-6 展示了“搜索机构树”组件的元素定

位信息。提取公共库的原则如下：1）该组件稳定且整体被大量调用；2）该页面对象独立存在且被大量交易调用，从交易界面移除该界面不影响交易的可理解性；3）公共对象库的个数不超过 20 个，避免造成公共对象库冗余，从而不便于测试人员记忆查找。

## 四、结语

面对日新月异的测试任务，自动化测试给测试人员指明了一种新途径。随着自动化测试方法的普及和应用，元素定位信息获取和管理将变得越来越重要。如何有效的建设和管理元素定位库是摆在测试人员面前的一个难题。文章中的元素定位自动生成方法很好的解决了此类问题，有力地帮助测试人员快速上手、实践和驾驭自动化测试。

# 安全测试，我有话要说

◆ 作者：aj

**前言：**随着信息技术发展，各国都对网络安全予以了高度重视，并进行了持续性战略资源投入。近几年来，网络安全成为各国制定重要政策，网络治理的战略谋划和顶层设计不断优化升级，网络防御和威慑能力建设并重，网络安全领域的合作和博弈互斗双向且并存。

## 一、网络安全概念及形势

### 1、什么是网络安全

广义的网络安全是网络空间安全的简称。

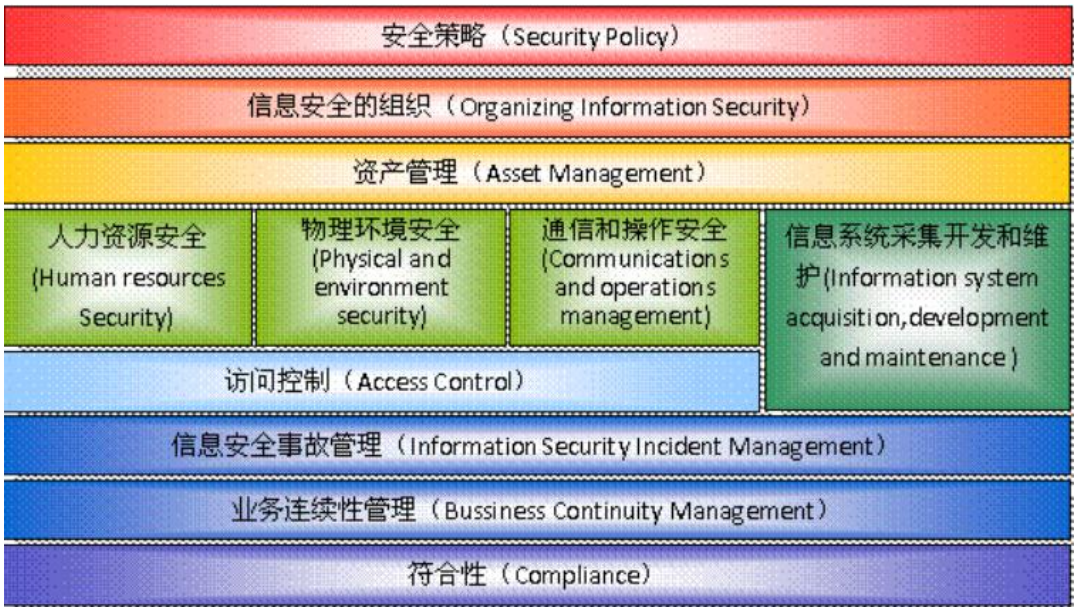
机密性：保证业务数据不被窃取，防止泄露。

完整性：保证业务数据真实、可靠、不被篡改。

可用性：保证业务数据实时可用，系统不中断服务。

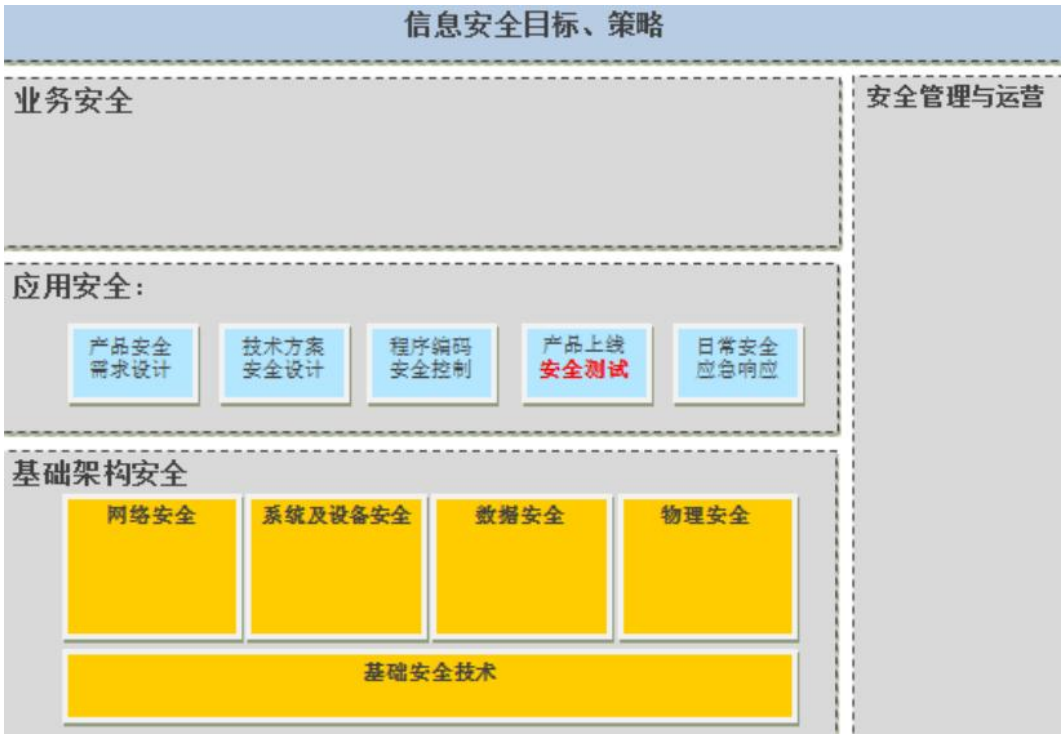
### 2、安全体系基本概念

（1）国际标准信息安全管理体系




（2）信息安全标准等级保护


（3）企业信息安全保障体系




### 3、网络安全的国际形势

走在国际科技领域前沿的美国在 2018 年在其国防部发布网络空间战略，同时也强调了前沿防御的理念，美国总统也赋予了军方不受阻挠部署先进网络武器的自由。

中美贸易战背后的竞争是关于网络空间的一场科技战，网络战则是这种博弈的极端方式；我们不仅可以看到美军在网络空间加紧备战，而且北约也正在整合各成员国的网络攻击能力，在贸易战的背景下，要充分考虑面对网络攻击和重大风险时，能否“扛得住、过得去”的严峻挑战。

中情局研发黑客工具，针对全球大量应用的信息化产品

网络司令部正在提高效率并保持战备状态

**4、我国对网络安全的重视**

（1）党的十八大以来，国家高度重视网络安全。

（2）中央网络安全和信息化领导小组在第一次会议讲话中指出，网络安全和信息化是一体化之两翼、驱动之双轮、必须要统一谋划、统一部署、统一推进、统一实施。

（3）2015 年 7 月 1 日，全国人民代表大会通过《中华人民共和国国家安全法》，首次将网络空间正式上升为我国继陆、海、空、天之后的第五疆域。

（4）2016 年 4 月 19 日，在网络安全和信息化工作座谈会上指出，网络安全和信息化是相辅相成的；面对复杂严峻的网络安全形势，我们要保持清醒的头脑，各方面齐抓共管，切实维护网络安全。

（5）2016 年 11 月 7 日，全国人民大表大会上通过《中华人民共合国网络安全法》。

（6）2018 年 4 月 21 日，全国网络安全和信息化工作会议上突出没有网络区安全就没有国家安全，就没有经济社会稳定运行，人民群众的利益也难以得到保障。

**二、安全测试基础与问题**

**1、为什么会出现安全问题**

开发人员：缺乏安全开发的概念，不知道如何在代码中避免安全漏洞。

测试人员：测试较为传统，只注重功能测试与性能测试，缺乏安全测试的基本理念与环节。

架构组件：框架和大量组件的使用，缩短了开发周期，降低了开发的难度，但是一旦框架与组件出现问题，整个系统都会受到影响。

**2、安全测试**

定义：检查应用系统对非法侵入及渗透的防范能力。

原则：理论上讲，只要有足够的时间和资源，没有绝对安全的的系统；因此，系统安全设计的原则是让非法入侵的成本和代价超过被保护系统的价值。

目标：对系统进行全面的安全测试，发现系统未知的安全隐患并提出相关建议，以确保系统的安全性。

**3、关于 SDL**

（1）Security Development LifeCycle 安全开发生命周期。

（2）最初由微软创建的软件安全保证过程，作为微软公司范围内的强制性策略。

（3）SDL 由下图几个阶段共 16 个步骤组成，统称为软件开发生命周期。


**4、安全测试的日常工作**

（1）安全开发

（2）0 安全分析

（3）等级评测

（4）0 安全规划

（5）安全检查

（6）风险评估

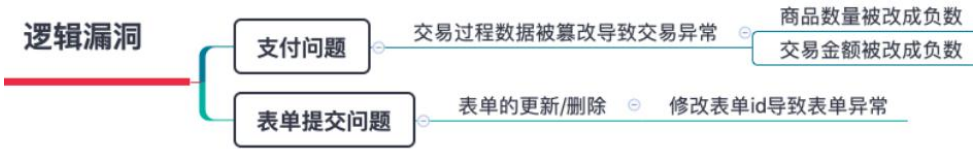
（7）重大保障

（8）应急响应

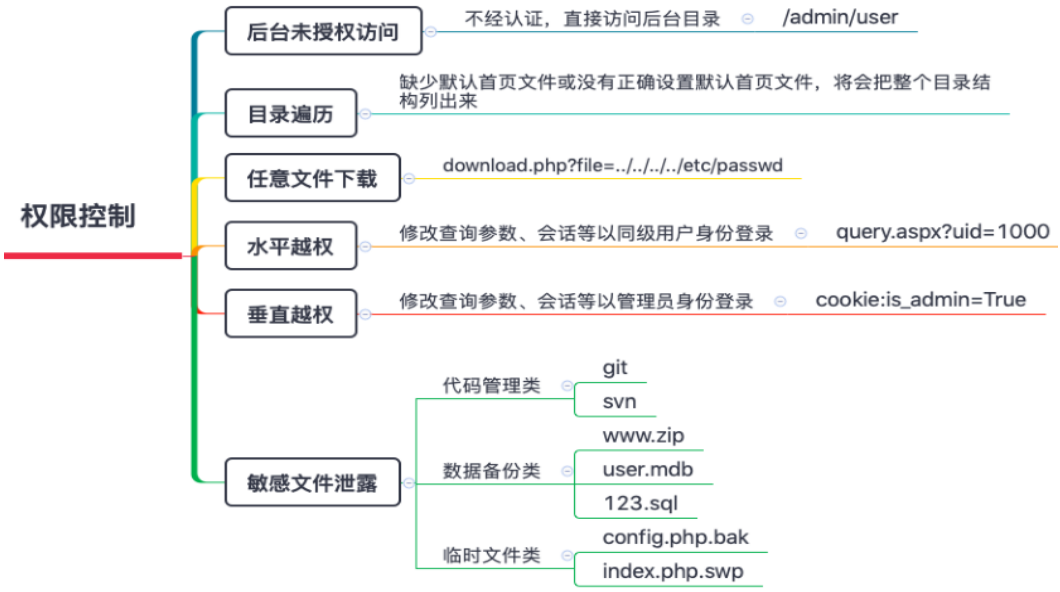
（9）上线检查

## 5、常见的安全测试点与分析

（1）逻辑漏洞



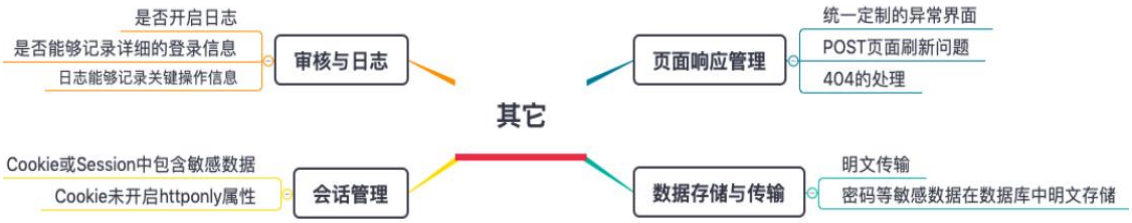

（2）权限控制




（3）配置管理


（4）身份管理与认证

（5）其他



（6）web 应用漏洞


（7）文件上传漏洞

**6、项目上常见的漏洞与说明**

（1）敏感数据泄露：许多 web 程序和 API 都无法正确保护敏感数据，攻击者可以通过窃取或修改未加密的数据来实施信用卡诈骗、盗窃身份等；未加密的敏感数据易受到破坏，因此，我们需要对敏感数据包括传输过程中的数据、存储的数据及浏览器进行交互的数据。

（2）失效的身份认证：通过错误使用应用程序的身份认证和会话管理功能，攻击者能够破译密码、秘钥及令牌等；或者利用开发上缺陷来冒充其它用户的身份。

（3）安全配置错误：此为最常见的安全问题，通常由于不安全的默认配置、不完整的临时配置、开源云存储、错误的 APP 标头配置以及包含敏感信息的详细错误信息造成的，因此不只要对系统、架构及应用程序进行安全配置，还要对其不断升级和及时修补。

（4）注入问题：将不受信任的数据作为命令或查询的一部分发送到解析器使，就会产生 SQL 注入、OS 注入等注入缺陷；相当于攻击者的恶意数据诱使解析器在没有授权的情况下执行非预期命令或访问数据，最严重的情况下可调用数据库中特殊存储过程，从而接管数据库以及运行数据库的主机。

（5）使用含有漏洞的组件：如框架和其它组件拥有和应用程序相同的权限；如果程序中含有已知漏洞的组件被攻击者利用，可能会造成严重的数据丢失和服务器接管，同时也会破坏应用程序的防御，产生严重影响。

（6）不足的监控和日志记录：不足的监控和日志记录，以及事件响应确实或无效的集成，使攻击者进一步攻击系统，以及篡改、提取或销毁数据。

## 三、安全测试框架与工具

### 1、安全测试的必经之路

（1）开发前：审查策略和标准、开发衡量标准和指标

（2）设计阶段：安全需求审查、设计和架构审查、创建并审查 UML 模型和威胁模型

（3）开发阶段：代码浏览、代码审查

（4）部署阶段：应用程序安全测试、配置管理测试

（5）运行维护阶段：操作管理审查、定期状态检查、变更后的验证

### 2、Web 安全测试框架

| | | |
|---|---|---|
| 认证测试 | 会话管理测试 | 授权测试 |
| 数据验证测试 | 信息收集 | DOS 测试 |
| Web 服务测试 | 业务逻辑测试 | AJAX 测试 |
| 配置管理测试 | | |

### 3、常用的检测工具与介绍

（1）Sqlmap：一款支持 MySQL， Oracle，PostgreSQL，Microsoft SQL Server，Microsoft Access， IBM DB2， SQLite， Firebird，Sybase 和 SAP MaxDB 等数据库的各种安全漏洞检测工具；同时支持 5 种不同的注入模式，即：基于布尔的盲注、基于时间的盲注、基于报错注入、联合查询注入、堆查询注入。

（2）IBM Appscan：由 IBM 公司开发，业界领先的 web 应用安全检测工具，是一款通过分析应用程序运行的结果来报告问题，广泛应用于 web 应用程序的渗透性测试，可根据个人需求生成所需格式的报告；

测试阶段,Appscan 通过攻击来测试应用中的漏洞，通过释放出的实际攻击的有效载荷，来确定在探索阶段建立的安全漏洞的情况.并根据风险的严重程度排名；另外在测试阶段可能会发现网站的新链接，因此 Appscan 在探索和测试阶段完成之后会开始另一轮的扫描，并继续重复以上的过程，直到没有新的链接可以测试，扫描的次数也可以在用户的设置中配置。

（3）Jsky：一款国内著名的网站漏洞扫描工具，基于 XML 框架设计的一款深度 web 应用安全评估工具，能轻松应对各种复杂的 WEB 应用，可检测出如：SQL 注入、跨站脚本、目录泄露、网页木马等在内的所有 web 应用漏洞

强大的 Web 结构扫描引擎能够准确且全面分析 Web 应用的结构。多线程运行极大地加速扫描过程。同时，Jsky 也能从 Javascript 脚本或者 Flash 文件中提取隐藏的 URL 路径；模块化的漏洞扫描设计 XML 语言描述的漏洞脚本能让您在不开发一行代码的情况下定制与您的 Web 应用系统具有针对性的漏洞扫描。

（4）Checkmarx Suite：是一个独特的源代码分析解决方案，该工具可用于识别、跟踪和修复源代码中技术上和逻辑上的缺陷，扫描和分析的输入数据是软件源代码，不是二进制代码，所以毋须构建或编译软件项目的源代码，通过虚拟编译器自动对软件源代码分析，并直接建立了代码元素及代码元素之间关系的逻辑图，然后再对这个内部代码图进行查询。

扫描结果可以以静态报表形式展示，也可以通过可以对软件安全漏洞和质量缺陷在代码的运行时的数据传递和调用图跟踪的代码缺陷的全过程，同时还可以提供对安全漏洞和质量缺陷进行修复提供指导建议。也可以对结果进行审计，从而消除误报。

（5）dirsearch：一个 python 开发的目录扫描工具，通过暴力扫描页面结构和网站的敏感文件及目录来找出问题；

拥有多线程、支持多种后缀、生成报告、递归得暴力扫描、支持 HTTP 代理、用户代理随机化、批量处理等特点。

**四、安全测试实践与分享**

**1、注入与避免**

SQL 注入，一个永不过时的黑客技术，最早期可以追溯到 20 世纪末，微软就出现了一系列的 SQL 注入问题。

轻则造成数据泄露，重则服务器被攻击者写入恶意代码造成服务器被远程操控。

（1）使用参数化查询：让其所有开发者先定义好 SQL 代码，再将每个参数逐个传入，其目的是让其数据库辨明代码和数据。

（2）确保攻击者无法改变查询内容。

```
参数化查询JAVA示例：
String custname = request.getParameter( " customerName " );
String query = " SELECT account_balance FROM user_data WHERE user_name= ? " ;

PreparedStatement pstmt = connection.prepareStatement(query);
Pstmt.setString(1,custname);
ResultSet results = pstmt.executeQuery();
```

（3）使用存储过程：与参数化查询作用大致相同，不同处在于存储过程是预先定义并存放在数据库中，从而被应用程序调用。

```
存储过程JAVA示例：
String custname = request.getParameter( " customerName " );
try {
    CallableStatement cs = connection.prepareCall( " call
sp_getAccountBalance(?)} " );
    cs.setString(1,custname);
    Result results = cs.executeQuery();
}catch(SQLException se){
//error handling
}
```

（4）创建白名单验证：在数据传递到 SQL 查询前觉察到输入是否合法，相比较黑名单能在更大程度上保证数据的合法性。

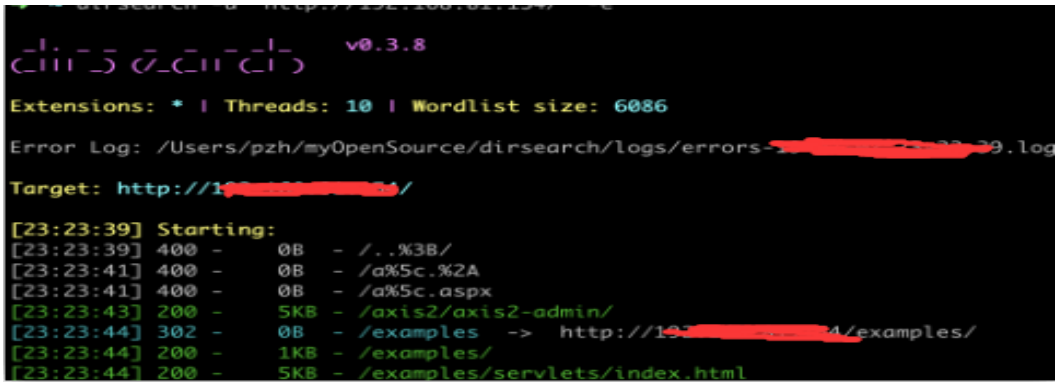
（5）最小权限方法：把每个数据库的用户权限尽可能缩小相当于只给用户能完成工作的最小权限。

2、Asix2（一个 WebService 的框架）任意文件读取 getshell

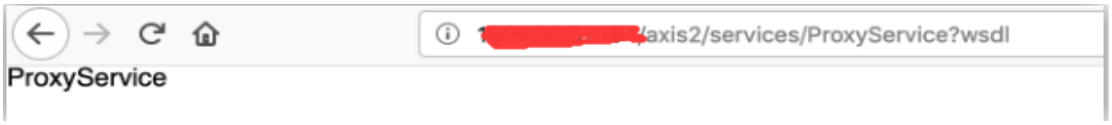
Tomcat 是一款免费开源的 Servlet 容器，可以通过在后台部署 war 包的形式进行 Web 应用的发布，所以一旦被攻击者控制，便可上传恶意代码实现对服务器的远程控制。

（1）访问首页无法登录 tomcat 后台

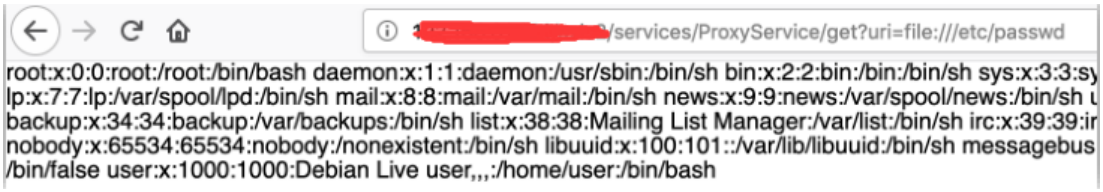
（2）利用枚举路径得到 axis2

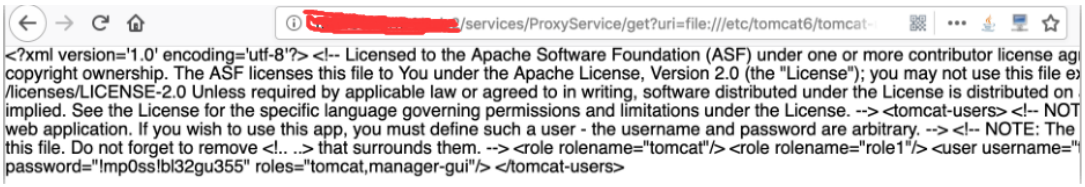



（3）通过链接来到 ProxyService




（4）利用 ProxyService 读取系统的 password 文件及 tomcat 文件




（5）如图得到 tomcat 账号密码；即，可通过 tomcat 上传含有远程木马的 war 包

小结

安全测试不是空泛的纸上谈兵，要充分重视并行动起来，同时也要对其进行不断的优化升级，将安全测试的好方法，好经验融入到项目中，测试人员要将自己想象成“黑客”去攻击自己手中的项目，以此来保证项目的安全性，避免遗漏测试点，以达到最终的测试目的。

# 自动化测试实现优劣浅谈

◆ 作者：多则惑少则明

## 一、背景

目前互联网测试江湖中，几乎所有团队、所有测试人员都在做自动化测试。从自动化测试方法论层面来说，之所以要实现自动化测试，大致有以下几方面的原因:

1、提升测试效率

2、提升测试覆盖度，包括深度和广度

3、提升测试发现问题后的解决效率

4、补充手动测试无法覆盖/不易覆盖的场景

以上是实现自动化测试一些方向性的指导原则，但要评价自动化测试实现的优劣，就需要拿具体的数据来说话了。

究竟自动化测试的效果如何呢？自动化测试有没真正发挥出来其作用，又如何来评价一个团队自动化测试工作做得好坏呢？下面就自己的一点经验，来说说自己对自动化测试优劣的一些体会，不足之处，欢迎大家交流指正。

## 二、评价自动化测试优劣的常见指标

互联网公司中，由于绝大多数团队都在重点攻克自动化测试，因而每个团队都指定了相应的评价指标。虽然各个公司，各个团队对指标的侧重不同，但几乎都会关注如下指标。

### 1、自动化运行通过率/成功率

这里排除了代码 bug 导致的失败。为了避免自动化运行经常失败，大部分团队都会将自动化运行通过率作为一项重要指标，来评判不同模块/业务线自动化实现的好坏。还将这个指标设置一定的阈值，例如，经历过的有个团队要求自动化运行成功率要大于 90%+，最好能 100%运行成功。

这个指标是几乎所有团队都会强调的一个指标了，因为自动化运行通过率/成功率是自动化发挥作用的前提条件。但看这个指标，其实也无法评估自动化实现效果，只能说明自动化运行的比较稳定。

**2、自动化执行频率**

这里的执行频率，排除业务测试，特指每天定时的自动执行。不同团队，根据实际情况不同，有不同的要求。例如，有的团队要求早晚至少各一次，有的团队则要求每天至少执行一次，但无论哪一种，几乎都会形成这样一种现象：各个团队为了让定时自动执行时都通过，每天都要花费一定时间来维护自动化代码。

这个指标其实也是一种评估自动化运行是否稳定的指标。执行的频率越高，加上执行的通过率/成功率越高，说明自动化运行的越稳定。

**3、自动化 case 数量/覆盖场景数**

这个指标不同业务线之间，其实没有太大的可比性。但同一个业务线上，还是可以作为参考指标的。自动化 case 多、覆盖的场景多，至少一定程度上说明了自动化的覆盖范围。这个指标可以一定程度上来评估自动化测试的广度、深度。

**4、自动化发现 bug 比例**

如果说自动化运行通过率/成功率、自动化执行频率、自动化 case 数量/覆盖场景数还是评价自动化优劣的过程指标，那么自动化发现 bug 比例就是最有力的结果性指标了。试想一种极端情况，如果一个自动化项目，跑了 N 久后，还没有发现过 bug，那么这个自动化的价值是不是要打一个大大的问号呢？

自己经历的团队，以及曾经接触过的团队，其实对自动化发现 bug 比例的侧重反而不是那么高。

这个指标可以说是众多指标中，最能立竿见影的说明自动化实现效果的指标了。自动化发现 bug 的比例越高，说明自动化在实际的项目中发挥的作用越大。甚至可以用自

动化发现 bug 率是 100%，来作为自动化测试的终极目标。

### 三、评价自动化测试优劣的隐性指标

关于评价自动化测试的优劣，除了上述常见指标外，还有一些不太容易拿数据说话的指标，这里叫做隐性指标。

隐性指标主要包括：自动化的维护成本、自动化的运行成本

#### 1、自动化的维护成本

针对同一个业务，不同的自动化测试实现方案，对应的维护成本可能天壤之别。诚然，自动化的维护成本，受业务成熟度、迭代速度、项目规范程度影响，但不妨考虑以下情况下，你的维护成本如何：

新增了一些逻辑(如，接口/服务/应用)，对新增部分维护自动化，你需要多长时间；

删除了一些逻辑(如，接口/服务/应用)，对删除部分维护自动化，你需要多长时间；

修改了一些逻辑(如，接口/服务/应用)，对修改部分维护自动化，你需要多长时间；

在项目迭代速度加快时，并伴有增删改逻辑时，你的自动化脚本还能跟得上吗？其实，这是不少团队都会面临的严峻考验。 一个正处于快速发展的业务，每次业务测试、回归时，可能都会想放弃自动化测试，转而来手工执行测试。因为，自动化测试还要不断调试自动化代码，大概率来不及这次的测试，还不如直接手动测试的效率高。

#### 2、自动化运行成本

这里的自动化运行成本是说，从想执行自动化 ~?执行结束 需要符合的人力&时间成本。一般的自动化运行过程大致如下：

1）创造一些自动化执行条件。比如，找运行数据，设置运行环境等等，这一步如果没有被自动化掉，需要花费人力&时间；

例如，实现的自动化只能”一条腿走路“，即只实现了半自动化，并没有实现100%的自动化，运行前/中/后可能需要人为参与。

2）执行自动化。这里主要是自动化运行所需时间，时间越长，导致的等待时间越长。可能你会说，自动化执行的时候，你可以去干别的事情啊，没必要一直等待执行结束。但既然执行了自动化，肯定想像手工测试一样，”马上“看到执行结果，得到及时

反馈，才能避免在不同工作间来回切换。

3）验证自动化结果。一般结果的验证都包含在上一个步骤里面了，但不排除有些验证仍然需要人工来 check 的情况。这种情况，其实也属于半自动化的一种实现形式。

4）自动化失败问题排查。各种各样的原因，都会导致自动化运行失败了，比如，数据问题、环境问题、自动化维护不及时、第三方问题等等。自动化失败后，能否比较清晰的给出失败原因，甚至能根据自动化失败结果，直接定位到失败原因，这些做到位了，会让你对自动化爱不释手。

**四、写在最后**

由于业务或团队的各自特点，除了上述的常见指标、隐性指标外，可能还有其他的一些指标来评价自动化实现的优劣，也会对指标有不同程度的侧重。要想实现一套十分优雅的自动化方案，并不是一件容易的事情，需要考虑方方面面的因素，也可能被各种因素所干扰。但在实现和评价自动化的过程中，都应该“不忘初心,莫失初衷”：自动化测试的 ROI >手工测试的 ROI。 如果经过了足够长时间的自动化测试实践，发现?自动化测试的 ROI <=?手工测试的 ROI, 那么，要么是当前阶段还不适合使用自动化测试，要么你就需要来重新审视目前的自动化实现方案了。

《51 测试天地》（五十六）上篇 精彩预览

- 基于 pytest 与 postman 的数据自动采集接口自动化测试实践
- Jenkins 系列之自动化部署（1）
- Jenkins 系列之 Selenium-UI 自动化测试（2）
- Jenkins 系列之 Jmeter-API 自动化测试（3）
- 机器学习之 KNN 算法，好算法有迹可循！
- 全流程自动化测试·业务规则标准化及资产库建设探索
- JMeter 循环读取 CSV 文件实现接口批量测试
- 新人应该如何进入测试领域？

马上阅读