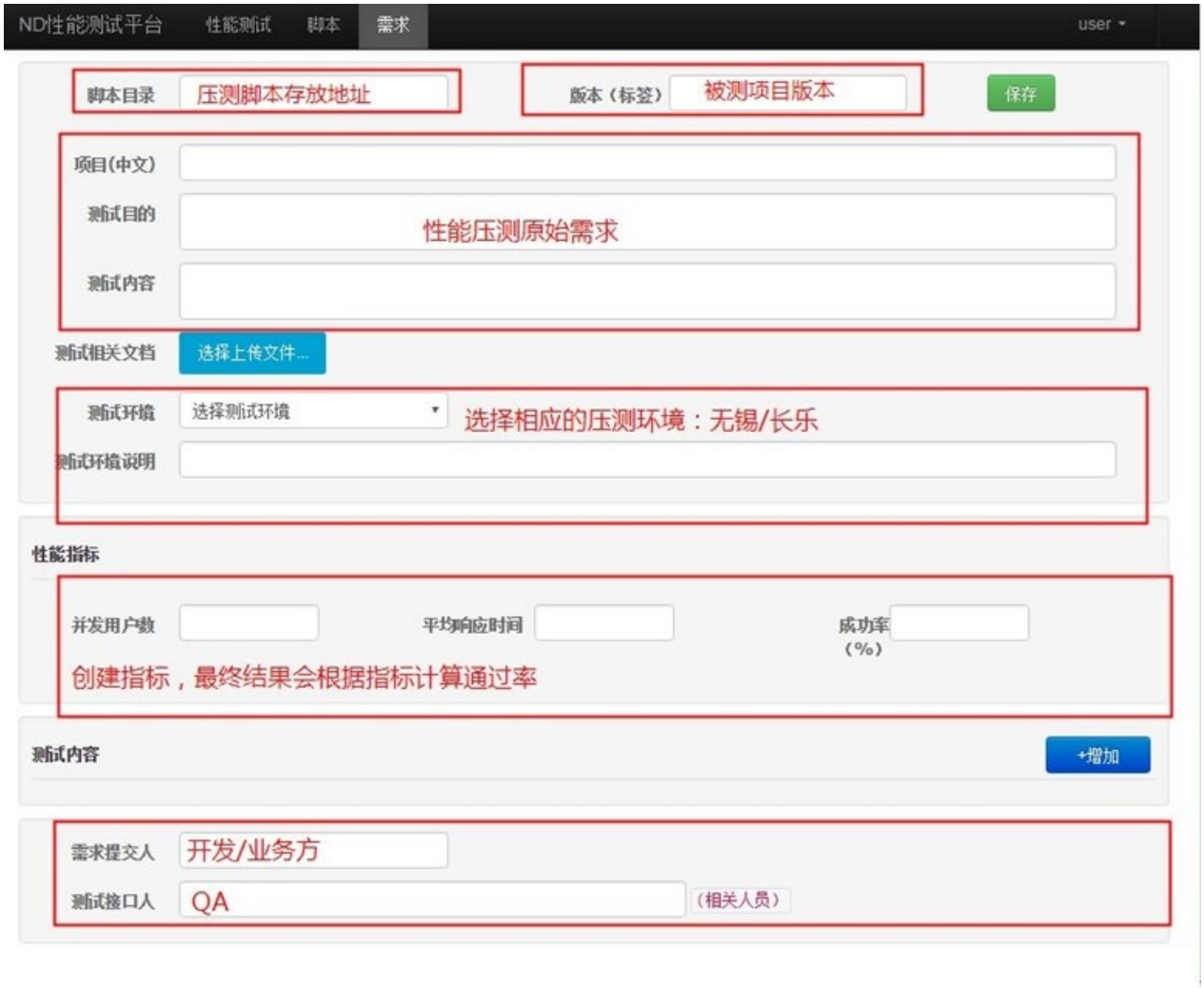
Table of Contents

1、创建需求

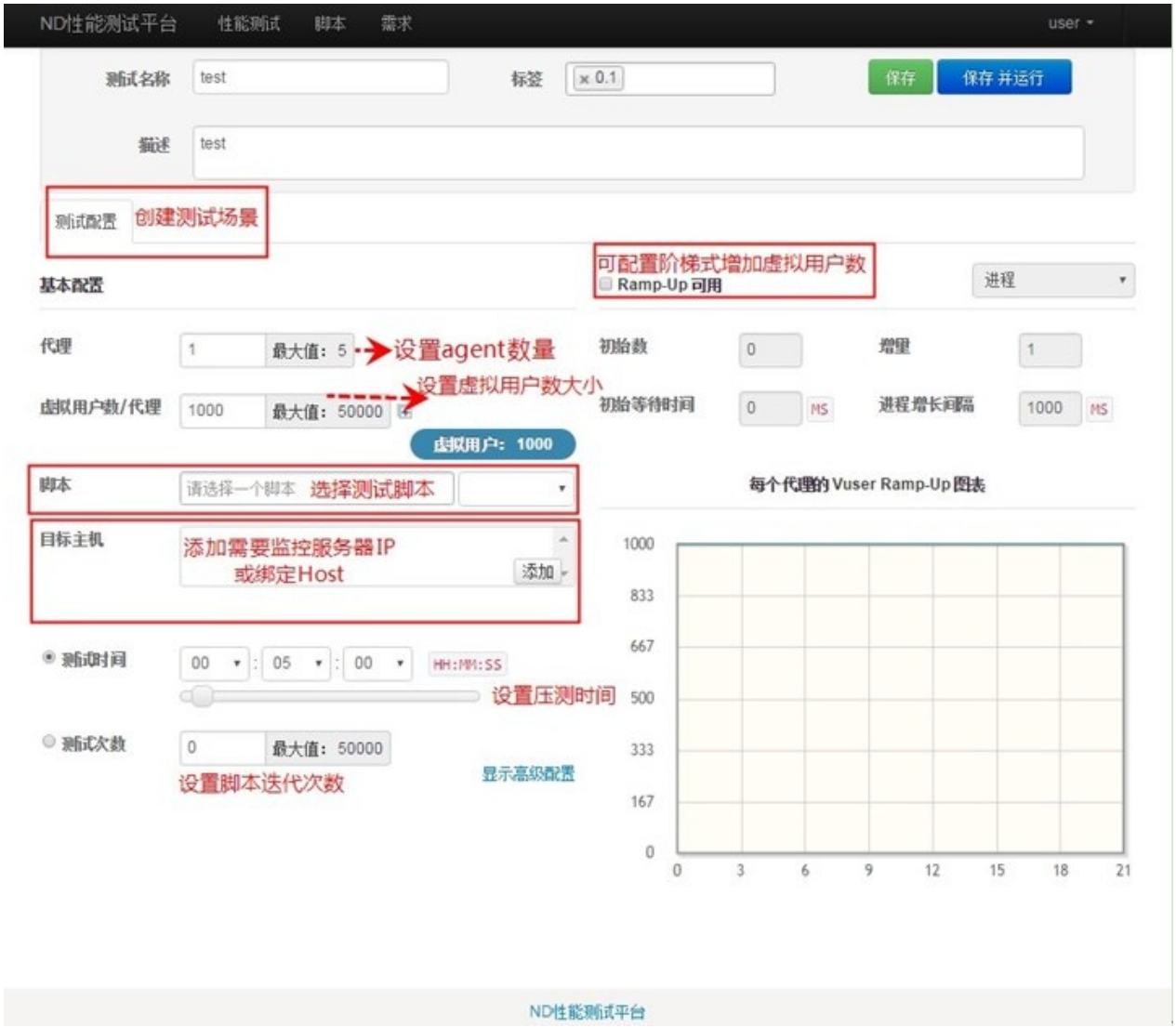
step1.新建测试需求，并在需求中创建测试，方便测试结果查看以及测试数据对比分析

step2.填写需求项



step3.在相应需求项下创建测试

2.创建测试场景


ND性能测试平台
性能测试
脚本
需求
user
测试名称
test
标签
× 0.1
保存
保存 并运行
描述
test
测试配置
创建测试场景
基本配置
可配置阶梯式增加虚拟用户数
Ramp-Up 可用
进程
代理
1
最大值：5
设置agent数量
初始数
0
增量
1
虚拟用户数/代理
1000
最大值：50000
设置虚拟用户数大小
初始等待时间
0
MS
进程增长间隔
1000
MS
虚拟用户：1000
脚本
请选择一个脚本
选择测试脚本
每个代理的Vuser Ramp-Up 图表
目标主机
添加需要监控服务器IP
或绑定Host
添加
1000
833
667
500
333
167
0
0
3
6
9
12
15
18
21
测试时间
00
05
00
HH:MM:SS
设置压测时间
测试次数
0
最大值：50000
设置脚本迭代次数
显示高级配置
ND性能测试平台

3.预约执行

预约后执行界面：

状态	测试名称	脚本文件名	修改者	开始时间	持续时间	TPS	MTT	出错率	Vusers	操作
	f_3.3.5	fireABC_14/f_3.3...	user	2016-05-17 10:51	00:05:00				1,000	■
	waf_mac_vorg	WAF_1/waf_mac...	user	2016-05-17 08:20	00:05:00	238	696.1	78.75%	4,000	
	waf_client_retry	WAF_1/waf_clien...	user	2016-05-17 08:30	00:05:00	9,935.4	100.2	0%	1,000	
	waf_client_retry	WAF_1/waf_clien...	user	2016-05-17 08:40	00:05:00	9,352.3	205.9	0.09%	2,000	
	waf_client_retry	WAF_1/waf_clien...	user	2016-05-17 08:50	00:05:00	8,820.7	371.5	0.9%	4,000	
	waf_client	WAF_1/waf_clien...	user	2016-05-17 07:20	00:05:00	9,499.8	352.6	0.74%	4,000	

状态：

1.  :准备/预约运行
2.  :绿色闪烁，运行中
3.  :运行完成，且通过
4.  :运行完成，且失败
5. TPS：Transaction Per Second 每秒事务数

6. MTT: 平均响应时间
7. Vusers: 并发用户数

4.测试结果查看

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，便可查看详细的测试报告及服务器资源消耗情况。


测试报告： 4.1.2 获取任务同步列表
虚拟用户总数 100
代理 2
进程数 线程数 2 / 25
忽略取样数量 0
TPS 4,625
TPS峰值 5,564
平均时间 20.78 ms
执行测试数量 1,359,059
测试成功数量 1,359,035
错误 24
执行报告
目标服务器
.207
.205
.213
PLUGINS
开始时间 2016-09-05 21:51:48
结束时间 2016-09-05 21:56:49
测试时间 00:05:00 HH:MM:SS
运行时间 00:05:01 HH:MM:SS
描述 4.1.2 获取任务同步列表
可下载测试记录实时数据
Performance
下载 CSV文件
TPS
6678
5565
4452
3339
2226
1113
0
Total
00:00
00:32
01:04
01:37
02:09
02:42
03:14
03:47
04:19
04:52
平均时间 (ms)
点击查看目标服务器资源详细数据
1350
1125
900
675
450
225
0
Total

1.1 ngrinder脚本结构

测试脚本的大概结构如下：

> [... import statements ...] 外部包引入
>
> [... request and test definitions ...] request请求声明
>
> class TestRunner: [method definitions - a method is defined for each recorded page] TestRunner声明
>
> def **call**(self): [... calls to defined methods, which actually runs the requests] 定义call方法

范例：

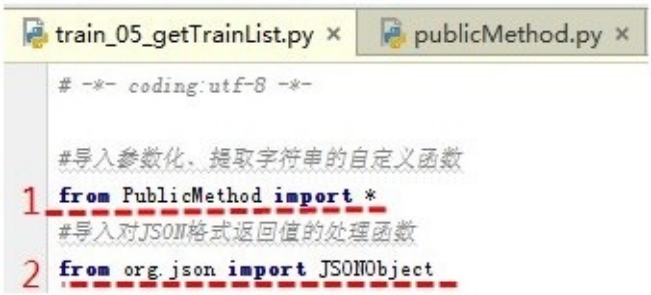
```
# -*- coding:utf-8 -*-

#导入参数化、提取字符串的自定义函数
from PublicMethod import *
#导入对JSON格式返回值的处理函数
from org.json import JSONObject
#日志级别
from org.slf4j import LoggerFactory
from ch.qos.logback.classic import Level
from ch.qos.logback.classic import Logger

control = HTTPPluginControl.getConnectionDefaults()
#请求重定向开关
# control.followRedirects = 1
#超时时间设置
control.timeout = 60000

test1 = Test(1, "Test1")
request1 = HTTPRequest()
test1.record(request1)

class TestRunner:
    #初始化，仅执行一次
    def __init__(self):
        grinder.statistics.delayReports=True
        #调整log输出级别
        logger = LoggerFactory.getLogger("worker");
        logger.setLevel(Level.ERROR); #DEBUG INFO ERROR WARN
        pass

    #类似LR的action，压测时会多次执行
    def __call__(self):
        host = 'elearning-train-gateway.qa.web.sdp.101.com'
        uri = '/v1/trains/pages?title=&page=0&size=20&order_by=1'
        url = 'http://'+host+uri
        XAuthorization='GAEA id="RwU/Ea32iQw="'
        headers = [NVPair("Content-Type","application/json"),NVPair("X-Gaea-Authorization",XAuthorization)]
        result = request1.GET(url,None,headers)
        if result.getText().find("title") != -1 :
            grinder.statistics.forLastTest.success = 1
        else :
            grinder.logger.error('url: '+url)
            grinder.logger.error('headers: '+str(headers))
            grinder.logger.error(result.toString())
            grinder.logger.error(result.getText())
            grinder.statistics.forLastTest.success = 0
```

1.2 TestRunner类说明

Jython定义方法用到关键字def，一般”—xx—”格式的方法都是语言内置的特殊方法。

1. —init—方法是类的初始化方法，当类被实例化的时候该方法会被用调用来初始化类的成员数据。
2. —call—方法用来实现类的callable接口，一般会在多线程调用时用到，同时用来放置测试的业务主体，这样可以多次循环的 调用业务逻辑。
3. —del—方法类似Java的finalize方法，通常在类的对象消亡之前执行一些销毁操作，但并不是实时。
4. —init—和—del—方法都是可选的，只有—call—方法是必需的。

2.1 引用方式



1、脚本中把经常需要用到的引用统一放到一个公共方法中，脚本只需要导入这个公共方法模块即可。

2、脚本中把各自特定需要的引用在各自脚本中另外导入。

附PublicMethod.py：

2.2 常用引用

```
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPPluginControl
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import NVPair
from java.lang import String
import os,sys,time
import random
```

注：

grinder和Test可以访问内部的测试框架。

HTTPPluginControl和HTTPRequest是操作HTTP请求。

NVPair是为POST请求提供数据。

String可以使用字符串类操作。

os,sys,time,random是导入操作系统，系统，时间，获取随机数的标准库。

3.1 get请求

Request定义好 ，对请求的url进行赋值后就可以进行get请求

```
host = 'elearning-train-gateway.qa.web.sdp.101.com'

uri = '/v1/trains/pages?title=&page=0&size=20&order_by=1'

url = 'http://'+host+uri

XAuthorization='GAEA id="RwU/Ea32iQw="'

headers = [NVPair("Content-Type","application/json"),NVPair("X-Gaea-Authorization",XAuthorization)]

result = request1.GET(url,None,headers)
```

3.2 post请求

Request定义好 ，对请求的url进行赋值，对请求的body赋值后就可以进行post请求

```
host = 'uc-bts.qa.web.sdp.101.com'

uri = '/v0.1/applys'

nonce_mac = getMacAuthorization('POST',uri,host,mac_key)

Authorization = 'MAC id="'+access_token+'",'+nonce_mac

url = 'http://'+host+uri

headers = [NVPair("Authorization",Authorization),NVPair("Content-
Type","application/json")]

json_str = '{"role_id":"1","app_id":"2","source_app_id":"'+randomOccurrence(appid)+'"}'

result = request1.POST(url,json_str,headers)
```

3.3 put请求

```
PUT (java.lang.String uri, byte[] data, NVPair[] headers)
```

参数	描述
Uri	为接口请求 URL 地址，是请求绝对路径
Data	为 post 请求的 Body
headers	请求的自定义头部

Example：

```
def __call__(self):

    #第一个请求
    host = 'uc-bts.qa.web.sdp.101.com'
    uri = '/v0.1/applications/{app_id}/appsecret/actions/reset'
    nonce_mac = getMacAuthorization('PUT',uri,host,mac_key)
    Authorization = 'MAC id="'+access_token+'",'+nonce_mac
    url = 'http://'+host+uri
    headers =  [NVPair("Authorization",Authorization),NVPair("Content-Type","application/json")]
    json_str = '{"app_secret":"1"}'
    result = request1.PUT(url,json_str,headers)
    grinder.logger.error(result.getText())
    grinder.logger.error(result.toString() )
```

3.4 delete请求

DELETE(java.lang.String uri, NVPair[] headers)

参数	描述
Uri	为接口请求 URL 地址，是请求绝对路径
headers	请求的自定义头部

Example：

```
def __call__(self):

    #第一个请求
    host = 'uc-bts.qa.web.sdp.101.com'
    uri = '/v0.1/policys/3'
    nonce_mac = getMacAuthorization('DELETE',uri,host,mac_key)
    Authorization = 'MAC id="'+access_token+'",'+nonce_mac
    url = 'http://'+host+uri
    headers =  [NVPair("Authorization",Authorization),NVPair("Content-Type","application/json")]
    result = request1.DELETE(url,headers)
    #grinder.logger.error(result.getText())
    #grinder.logger.error(result.toString())
```

4.参数化

步骤一：读取参数文件，把参数化文件保存在resourse文件目录下，在脚本中将数据读取到列表中。

步骤二：参数化，调用需要的参数化取值顺序方法，将第一步的数据列表作为传入参数。

注：在公共方法publicMethod.py中有自定义了cvs文件的读取和不同参数的取值顺序方法，导入公共方法模块后即可调用。

具体的参数化取值范例见下面具体的参数取值说明。

4.1 随机取值

请求的接口参数在并发的时候想给每个并发用户随机分配一个取值，可以使用随机模式randomOccurrence(valueList)方法；valueList为参数列表。

例如：get接口http://172.24.128.45:8088/Common/GetClientInfo?VersionCode=271 的VersionCode希望在并发的时候随机取一个值进行接口请求，脚本如下：

step1、从参数文件中读取获取对应的VsCommon的版本信息

```
filepath2 = "./resources/VsCommon.csv"

list2 = readCSVFile(filepath2)

global VsCommon

VsCommon = list2[0] ....
```

step2、在接口的request中使用randomOccurrence(valueList)方法

```
host = '172.24.128.45:8088'

uri = '/Common/GetClientInfo?VersionCode='+randomOccurrence(VsCommon)

url = 'http://'+host+uri

result = request1.GET(url,None,headers)
```

附参数csv文件：

4.2 顺序取值

请求的接口参数在并发的时候想给每个并发用户顺序分配一个取值，可以使用顺序模式，顺序模式有2个方法：

1、迭代更新（当次迭代所有用户使用同一参数）：paraSequential(valueList, valupdateMode)方法；valueList为参数列表，valupdateMode为数据更新模式，取值有2种'Iteration' 迭代更新、'Once' 固定值，不更新。

2、每次更新（每次出现更新）： paraOccurrence (valueList)方法，valueList 为参数列表。每次调用该方法，参数往下递进更新。

例如：post接口http://172.24.128.45:8088//Favorite/PutIntoFavorite 希望所有的并发用户body的SoftItemId值第一次取第一个值，第二次取第二个值，依次完成，脚本如下：

step1、从参数文件中读取获取SoftItemId的软件id信息

```
filepath2 = "./resources/SoftItemId.csv"

list2 = readCSVFile(filepath2)

global SoftItemId

SoftItemId = list2[0]

....
```

step2、在接口的request中使用paraSequential (valueList,'Iteration')方法

```
host = '172.24.128.45:8088'

uri = '/Favorite/PutIntoFavorite'

url = 'http://'+host+uri

json_str = '{"SoftItemId":'+paraSequential(SoftItemId,'Iteration')+',"Uid":53}'

result = request1.POST(url,json_str,headers)
```

注：该自定义方法现在不完善，建议待完善之后使用

4.3 唯一取值

请求的接口参数在并发的时候想给每个并发用户分配一个唯一的值，可以使用唯一模式paraUnique (valueList, valupdateMode)方法，valueList为已经按用户切分好的子参数列表；valupdateMode 为数据更新模式，'Iteration' 为迭代更新，'Once' 为固定值，不更新。

例如：登录接口https://ucbetapi.101.com/v0.93/tokens希望每次并发的登录用户信息都是同一批，脚本如下：

step1、从参数文件中读取获取对应的username用户信息

```
filepath1 = "./resources/username.csv"

list1 = readCSVFile(filepath1)

global username

username = list1[0]

....
```

step2、在接口的request中使用paraUnique (valueList, 'Once')方法

```
url = 'https://ucbetapi.101.com/v0.93/tokens'

headers = [NVPair("Content-Type","application/json")]

json_str = '{"login_name":""+paraUnique
(username,'Once')+"","password":"80fba977d063a6f7262a8a9c95f61140"}'

result = request1.POST(url,json_str,headers)
```

注：该自定义方法现在还不完善，还需后续优化

4.4 关联取值

如果一个接口有2个请求参数，且需要有一一对应的关联关系，可以在一个参数文件中存放多个参数列表并通过下标进行关联。

例如：接口http://172.24.128.45:8088/Soft/GetDeveloperSoftList?PageNO=1&PageSize=20&SoftItemId=13&DeveloperName=nd 的SoftItemId和DeveloperName存在一一对应的关系，脚本如下：

step1、从参数文件中读取获取对应的SoftItemId 和DeveloperName

```
filepath2 = "./resources/T66.csv"

list2 = readCSVFile(filepath2)

global SoftItemId,DeveloperName

SoftItemId = list2[0]

DeveloperName = list2[1]

....
```

step2、获取下标的值，根据不同的取值规则放在不同的请求方法中

```
global num

num = int(random.randint(0,1000))

step3、通过下标值关联2个参数，保证取的是同一行的值

host = '172.24.128.45:8088'

uri = '/Soft/GetDeveloperSoftList?PageNO=1&PageSize=20&SoftItemId='+SoftItemId[num]+'&DeveloperName='+DeveloperName[num]

url = 'http://'+host+uri

result = request1.GET(url,None,headers)
```

附参数csv文件：


脚本名
XStor_scripts/resources/T66.csv
提交信息
多个参数列，逗号分隔
soft_item_id,developer_name
5633,10tons Ltd
7266,17Studio
11489,24x7 Health Protect Pvt Ltd
12153,3D Magic LLC.
12,3dinteger
11388,48dreams
11380,500px
658,59Pixels
7634.6677a.com

4.5 下标取值

除了通过公共方法中定义的自定义参数化方式取值还可以自己定义下标文件的获取规则，通过下标来对参数列表进行取值。

4.4 中已经有示范了通过随机方法来获取下标值。

还可以根据公共方法中的getUniqueId()方法来获取，默认从参数文件的第一行开始取值，需要注意的是并发测试的时候设置代理的线程不为1的话取值会重复。

step1、从参数文件中读取获取对应的username

```
filepath1 = "./resources/user.csv"

list1 = readCSVFile(filepath1)

global user_name

user_name = list1[0]

....
```

step2、获取下标的值，根据不同的取值规则放在不同的请求方法中

```
global num

num = getUniqueId()
```

step3、通过下标值取参数列表值

```
url = 'https://ucbetapi.101.com/v0.93/tokens'

headers = [NVPair("Content-Type","application/json")]

json_str =
'{"login_name":"'+user_name[num]+'","password":"80fba977d063a6f7262a8a9c95f6114
0"}'

result = request1.POST(url,json_str,headers)
```

4.6 非文件方式取值

除了通过参数文件来进行参数化，还可以通过其他方式将接口的传值进行参数化。导入python相应库函数中相应的模块，就可以调用对应的方法。

例如：4.4中获取下标使用的随机方法，定义的num值。

例如：获取时间方法time()等

5.1 检查文本是否存在

result.getText().find("文本字符串")，没找到时返回-1

例如：检索HTTP返回值result中是否有sid相关信息，有，则标记为成功

```
if result.getText().find("sid") != -1:
    grinder.statistics.forLastTest.success = 1
```

5.2 检查状态码

result.getStatusCode()返回http状态码，如200 401 500等

例如：当HTTP返回值result中状态码不是200或300时，标记为失败

> if result.getStatusCode() in (200,300) :
>
> ```
>     grinder.statistics.forLastTest.success = 1
> ```

5.3 检查返回的字段值

提取请求返回信息json格式中的某个字段取值，这边返回的是字符串，需要其他类型的话要自己手动转。

例如：检查返回内容的StateCode是否为1

> json = JSONObject(result.getText())
>
> global StateCode
>
> StateCode = json.getString("StateCode")
>
> if StateCode=="1":

```
    grinder.statistics.forLastTest.success = 1
```

6.1 单事务

一个脚本如果只包含一个接口请求，直接定义一个request并在**call**方法里面进行接口请求即可。示例如下：

> def **call**(self):
>
> result1 = request1.GET("http://waf-demo.qa.web.sdp.101.com/benchmark/blank")
>
> if result1.getText().find("age") != -1 :
>
> ```
>     grinder.statistics.forLastTest.success = 1
> ```
>
> else :
>
> ```
>     grinder.statistics.forLastTest.success = 0
> ```

6.2 多事务

一个脚本里面如果需要包含多个接口请求，则需要先定义多个request并在**call**方法里面分别进行接口请求（请求为并发执行）。示例如下：

> def **call**(self):

```
    result1 = request1.GET("http://waf-demo.qa.web.sdp.101.com/benchmark/blank")
    if result1.getText().find("age") != -1 :
       grinder.statistics.forLastTest.success = 1
    else :
       grinder.statistics.forLastTest.success = 0


    result2 = request2.GET("http://waf-demo.qa.web.sdp.101.com/benchmark/blank")
    if result2.getText().find("age") != -1 :
       grinder.statistics.forLastTest.success = 1
   else :
       grinder.statistics.forLastTest.success = 0
```

也可以先自定义事务方法，在并在**call**方法中调用，实例如下：

> def tran1(self):

```
result = request1.GET("http://172.24.128.2/")

if result.getText().find("Welcome") != -1 :

    grinder.statistics.forLastTest.success = 1

else :

    grinder.statistics.forLastTest.success = 0
```

> def tran2(self):

```
result = request2.GET("http://777.nd.com.cn/html/index.html")

if result.getText().find("Object moved") != -1 :

    grinder.statistics.forLastTest.success = 1

else :

    grinder.statistics.forLastTest.success = 0
```

> def **call**(self):

```
self.tran1()

grinder.sleep(1000)

self.tran2()
```

7.1 思考时间

> grinder.sleep(ms) ，单位为毫秒，可在请求之间加入间隔时间。

例如：grinder.sleep(1000) #思考时间=1s

7.2 日志输出级别

引入日志相关的模块后，在初始化方法中对日志的级别进行设置；设置后可以进行日志打印（打印的日志在单脚本验证的时候会打印到结果，并发的时候会收集到日志文件中）

> ....
>
> from org.slf4j import LoggerFactory
>
> from ch.qos.logback.classic import Level
>
> from ch.qos.logback.classic import Logger
>
> ....
>
> def **init**(self):
>
> ....
>
> ```
> logger = LoggerFactory.getLogger("worker");
>
> logger.setLevel(Level.ERROR); #DEBUG INFO ERROR WARN
> ```
>
> ....
>
> ```
> grinder.logger.error(result.getText())
> ```

7.3 开关值

- 重定向开关

> control = HTTPPluginControl.getConnectionDefaults()
>
> control.followRedirects = 1 #设置为1，则自动重定向跳转

- 延迟汇报

> grinder.statistics.delayReports=True #"False"表示数据收集完之后立即汇报，"True"表示数据收集延迟汇报，一般我们需要对请求返回值做出判断，所以设置为“True”

- 事务结果标记

> grinder.statistics.forLastTest.success设置事务是否成功，1表示成功，0表示失败

例如：当HTTP返回值result中状态码是200时，标记为成功

> if result.getStatusCode() == 200 :
>
> ```
>     grinder.statistics.forLastTest.success = 1
> ```

7.4 共享平台UC-mac tocken

共享平台的项目都是使用UC登录，很多接口使用的是mac tocken的验证方式，压测脚本中如果需要计算mac tocken的相关值可以导入getMacAuthorization.py模块，在接口请求之前调用相应的方法获取mac和nonce的值。

例如-预生产环境的mac tocken计算：

```
from getMacAuthorization import *

....

host = 'fireabc.qa.web.sdp.101.com'

uri = '/v1/users/items?user_id='+user_id

nonce_mac = getMacAuthorization('GET',uri,host,mac_key) #获取nonce和mac

Authorization = 'MAC id="'+access_token+'",'+nonce_mac

url = 'http://'+host+uri

headers = [NVPair("Content-
Type","application/json"),NVPair("Authorization",Authorization)]

result = request1.GET(url,None,headers)
```

7.5 加密

接口请求中经常需要对请求参数进行加密，可以使用python自带的加密库进行计算或者自己编写加密方法导入。

例如：一个加密接口需要对几个值进行md5加密后作为sign参数。

```
import hashlib

....

Uid_para = randomOccurrence(Uid)

singn_str =
"GetFavoriteList120"+Uid_para+"E5630257697234D0EADD145D94F4265E"

singn_str_md5 = hashlib.md5(singn_str).hexdigest()

uri = '/Favorite/GetFavoriteList?
PageNO=1&PageSize=20&Uid='+Uid_para+'&Sign='+singn_str_md5
```

1.平台脚本示例

完整的脚本示例见平台上"脚本"-"A-DEMO_script"目录下的脚本示例：

http://221.228.81.201:8080/script/list/A-DEMO_script