目 录

(五十三期·上)

如果您也想分享您的测试历经和学习心得，欢迎加入我们~(*^▽^*)

投稿邮箱：editor@51testing.com

开源性能测试工具大比武

◆ 作者：晴 空

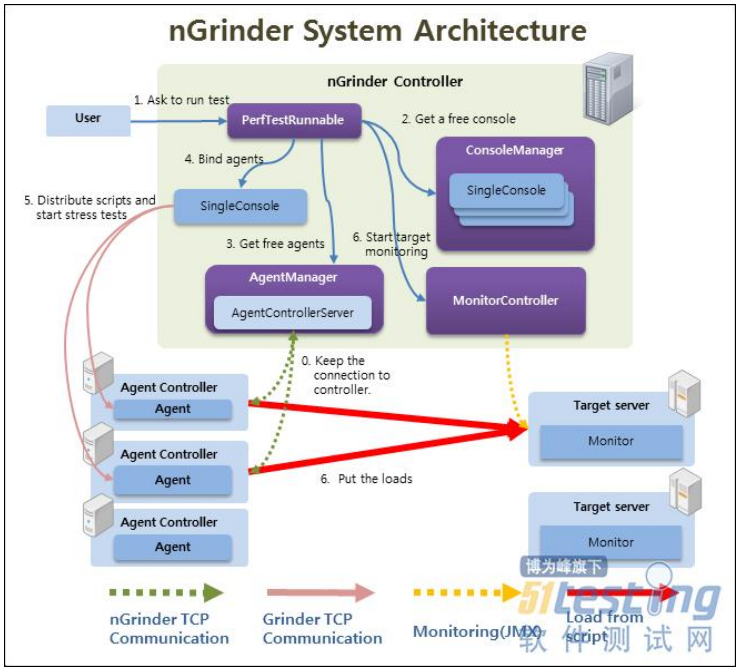
前言：工具好比兵器，熟练使用工具不一定会性能测试，但是挑选一件趁手的兵器会大大提高的我们的效率，本文将会和小伙伴们一起探究下性能测试工具中的倚天屠龙~

一：nGrinder

1.1 nGrinder 介绍

nGrinder 是韩国一家公司居于 Grinder 二次开发的一个性能平台。nGrinder 具有开源、易用、高可用、高扩展等特性，在 Grinder 基础上实现了多测试并行，通过 web 管理，实现了集群，同时支持 Groovy 和 Jython 脚本语言也实现了对目标服务的监控以及插件的扩展。

1.2 nGrinder 架构和原理



nGrinder 架构图

nGrinder 工作原理

1: 由一个控制端 controller 和多个代理端 agent 组成，通过控制端(浏览器访问)建立测试场景，然后分发到代理端进行压力测试。

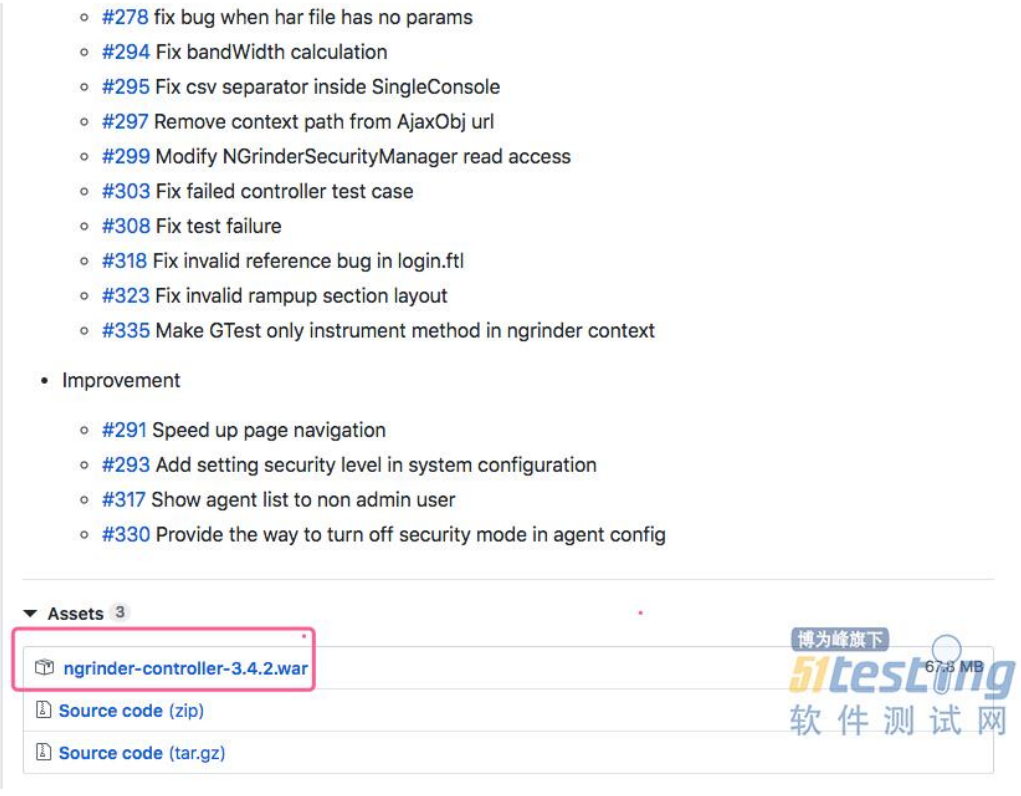
2: 用户按照一定规范编写测试脚本，controller 会将脚本以及需要的资源分发到 agent，用 jython 执行。

3: 在脚本执行的过程中收集运行情况、相应时间、测试目标服务器的运行情况等。并且保存这些数据生成测试报告，通过动态图和数据表的形式展示出来。用户可以方便的看到 TPS、被测服务器的 CPU 和内存等情况。

1.3 nGrinder 环境搭建

nGrinder 的 Github 仓库地址是 https://github.com/naver/ngrinder

我们可以在 https://github.com/naver/ngrinder/releases 这里下载最新版本的 nGrinder。



下载 ngrinder-controller-3.4.2.war。

接下来我们通常有两种方式来启动 nGrinder 的 controller 服务。

1: 命令行启动

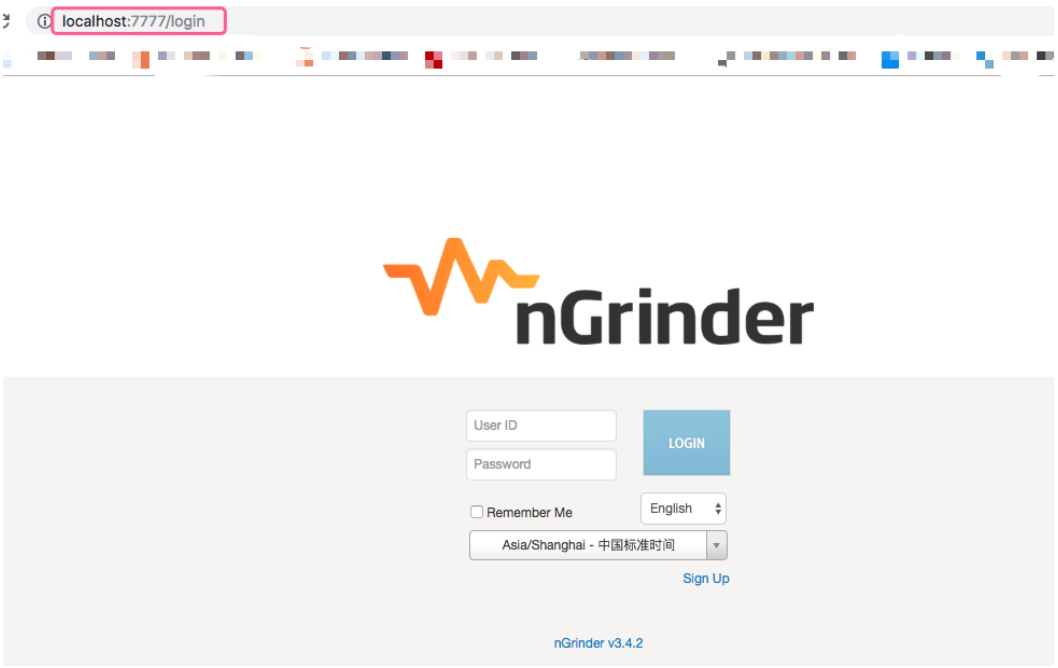
java ngrinder-controller-3.4.2.war。

当然我们可以设置它的一些默认值，比如:

java -XX:MaxPermSize=512m -jar ngrinder-controller-3.4.2.war --port 7777

上面的这条命令分别指定了 ngrinder 的内存使用和服务端口(nGrinder 默认的端口号是 8080)。

接下来我们通过访问 http://localhost:7777 访问 nGrinder 服务。

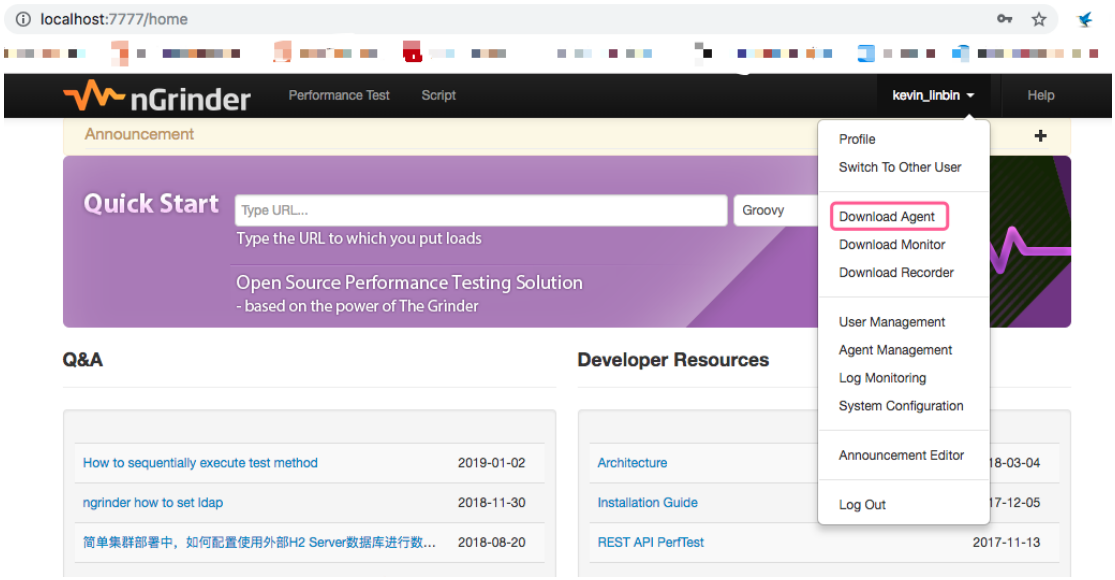


2: 配置 tomcat 环境

将下载的 war 包放置在 tomcat 的 webapps 目录下即可。

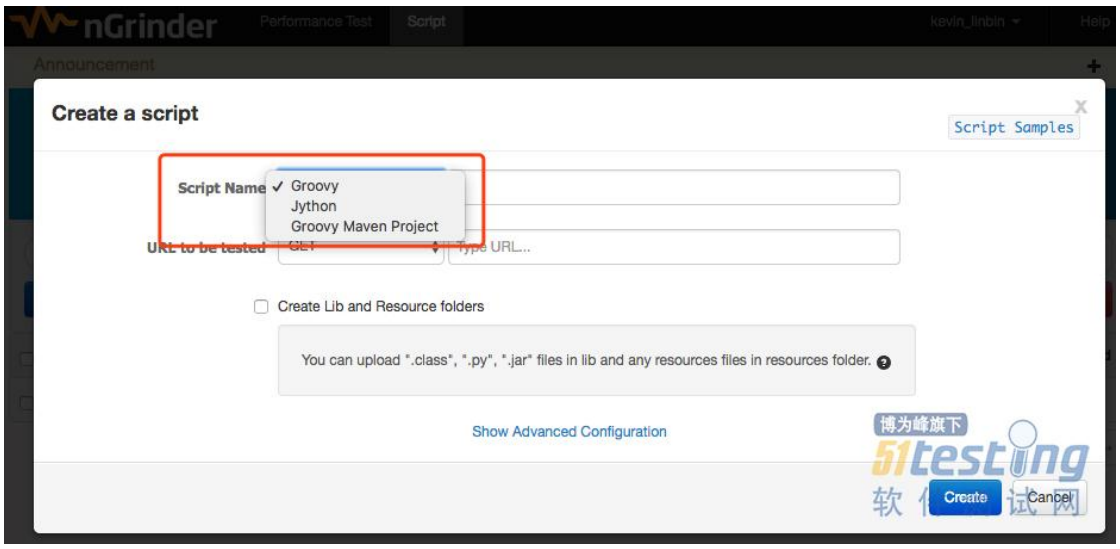
通过 http://localhost:8080/ngrinder 访问服务。

nGrinder 的默认用户名和密码是 admin/admin

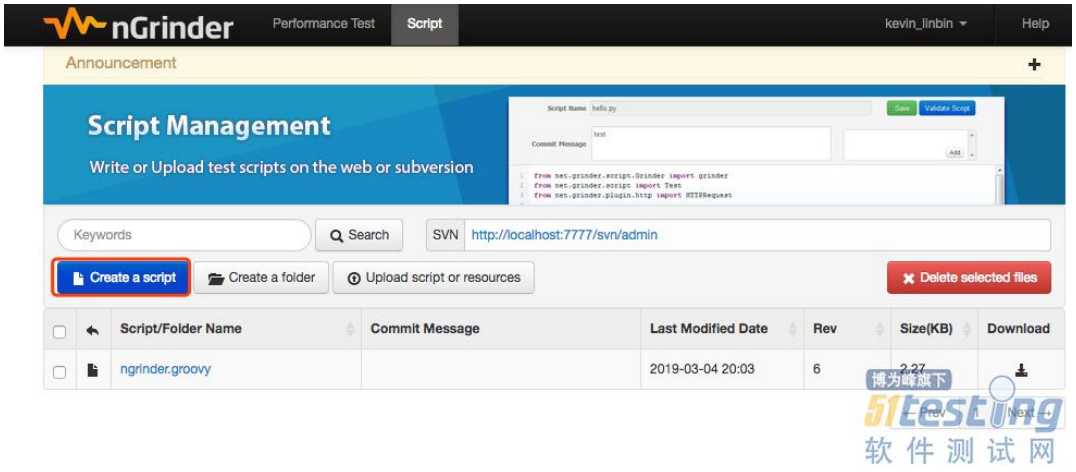


1.4 nGrinder 初体验

1.4.1 脚本管理



创建脚本，使用 admin 登录后切换到 Script 栏，点击 Create a script，nGrinder 会弹出浮层以供我们设置脚本。

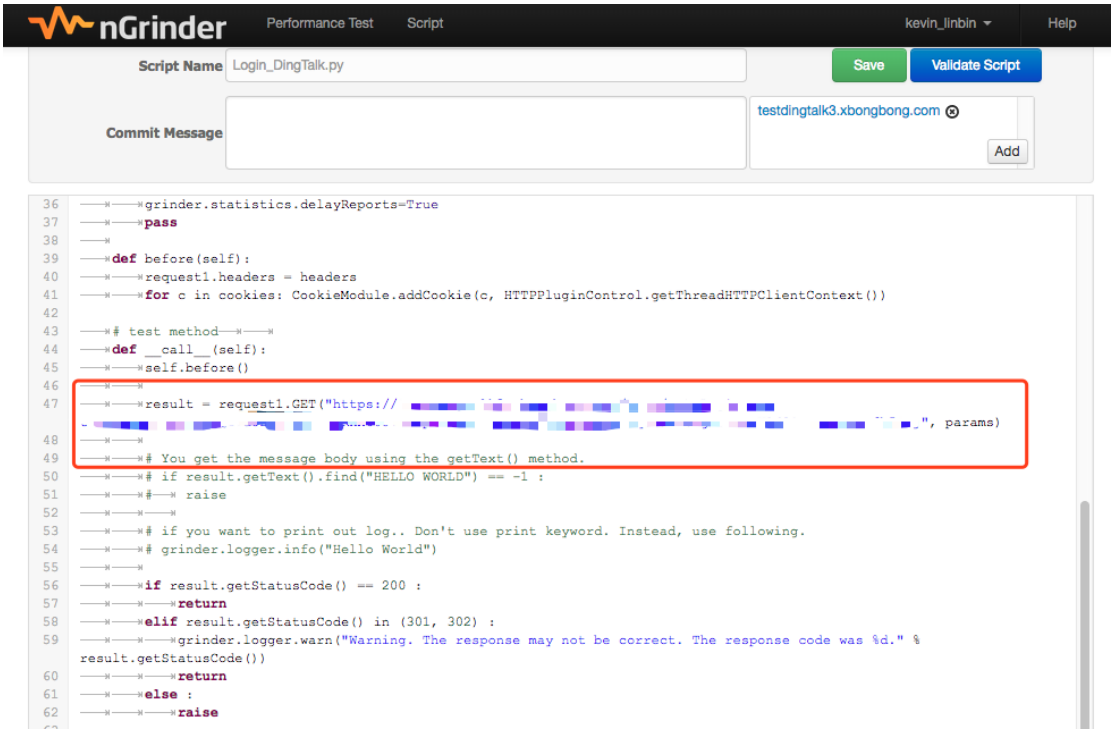


nGrinder 支持 Groovy 和 Python 两种脚本语言，我们可以自由切换。

我们所有做的，是非常简单的。

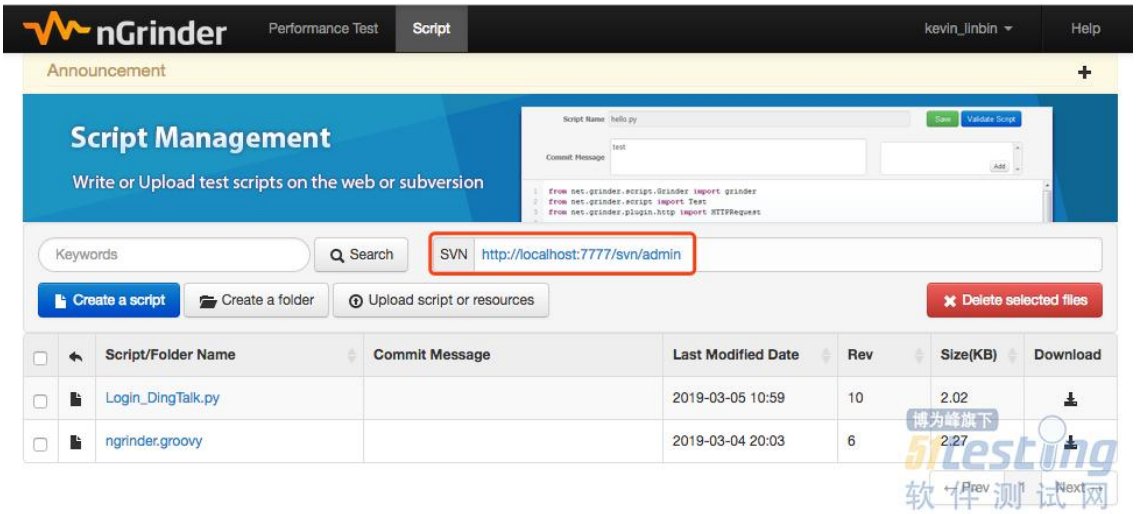
如果是 get 请求，我们只需设置下 URL 即可。当然在 Advanced Configuration 高级设置中我们可以设置 cookie 和 header。

如果是 post 请求，填写完 URL 后，在高级设置那里我们可以设置 param 以及 cookie 和 header 信息。



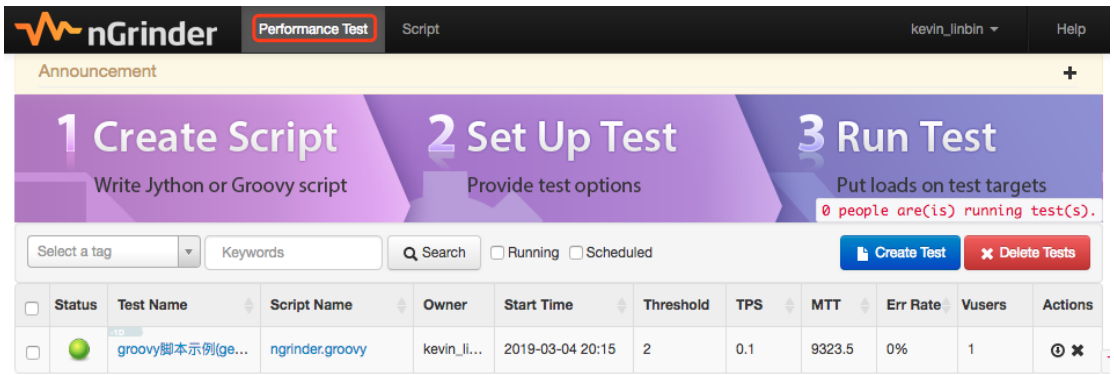
请求信息设置完成后，我们可以点击 Validata Script 来调试我们的脚本，调试通过后点击 Save，nGrinder 会自动保存我们的脚步。

那么问题来了？我们的脚本保存到哪里啦？



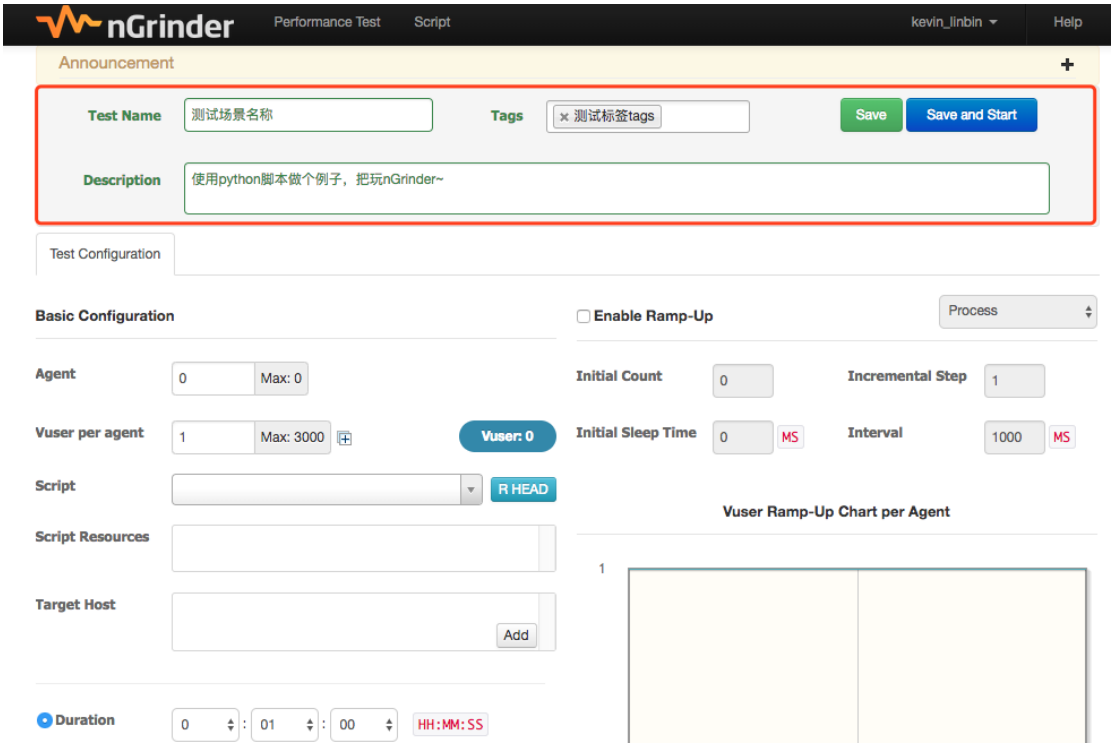
nGrinder 集成了 svn，我们设置脚本后点击 Save，nGrinder 会自动保存在自带的 svn 中。

1.4.2 测试场景

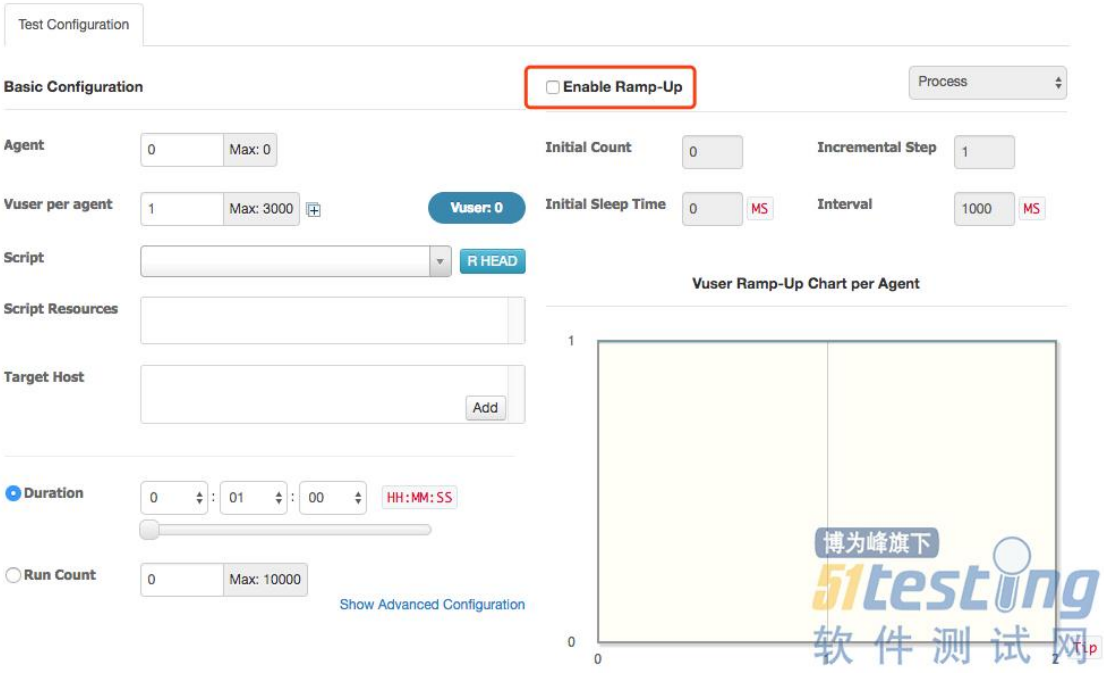


在 Performance test 栏，我们可以场景测试场景，其实叫测试场景不太准确，因为我们创建 test 后可以立即执行也可以在指定时间执行(一时间想不起来什么好词，书到用时方恨少啊~~)。

下面我们创建个 test 来把玩一下:



Announcement 部分我们必须填写 Test Name 来标识场景的唯一性。其他两项可以不填，没啥用。



Test Configuration 部分，我们能玩的就多了：

1：enable ramp-up

这个选项非常有用，允许我们初步初始化负载。

2: nGrinder 允许我们以 process(进程)或者 thread(线程)的方式生成负载。

3: Agent 这里是设置使用多少代理机。

4: Vuser per agent 顾名思义，每个代理模拟的虚拟用户数。

5: Script 这里选择我们的压测脚本。

1.5 nGrinder 实战

官方推荐 Groovy 脚本作为首选，那我们就来配置下 Groovy 的环境吧。

Mac 下 brew install groovy 安装。然后配置系统变量

export GROOVY_HOME=/usr/local/opt/groovy/libexe

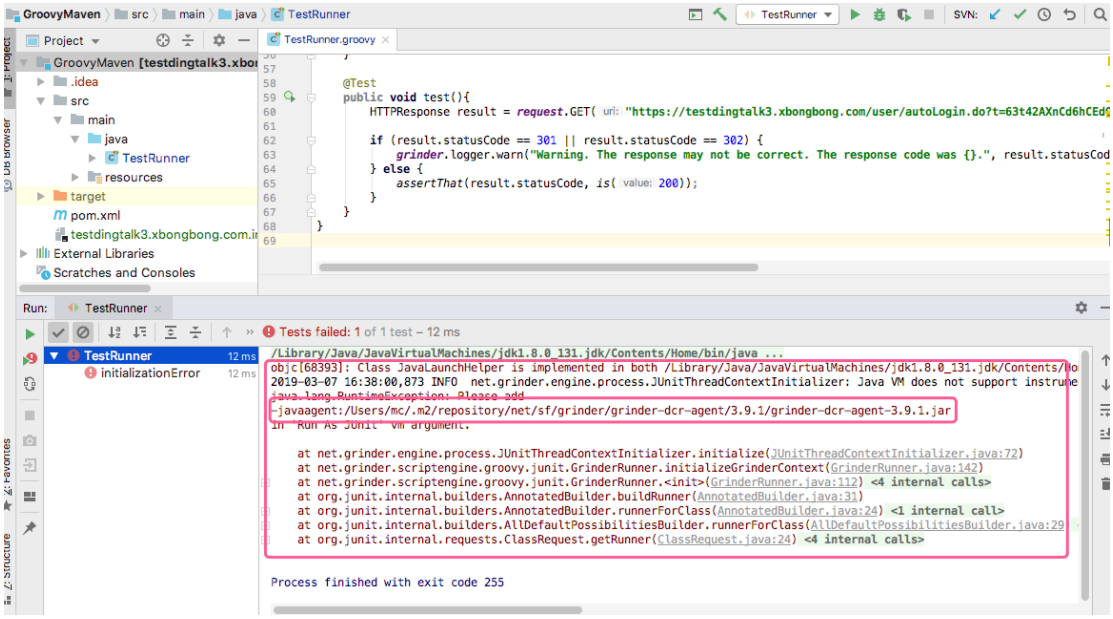
groovy -v 来查看版本信息

```
Kevinqingkong:~ mc$ brew install groovy
Updating Homebrew...
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/cask and homebrew/core).
==> Updated Formulae
dmd               fn               jruby            packer           rke
eslint            ghc              libcerf          parallelstl      roswell
exploitdb         glib             lmod             passenger        skinny
firebase-cli      ispc             lxc              pulumi           syncthing

==> Downloading https://dl.bintray.com/groovy/maven/apache-groovy-binary-2.5.6.zip
Already downloaded: /Users/mc/Library/Caches/Homebrew/downloads/4746ae1b99b82aa86cc
f2ddac83bd17573a194187--apache-groovy-binary-2.5.6.zip
==> Caveats
You should set GROOVY_HOME:
  export GROOVY_HOME=/usr/local/opt/groovy/libexec
==> Summary
  /usr/local/Cellar/groovy/2.5.6: 94 files, 18.4MB, built in 19 seconds
Kevinqingkong:~ mc$ groovy -v
Groovy Version: 2.5.6 JVM: 1.8.0_131 Vendor: Oracle Corporation OS: Mac OS X
Kevinqingkong:~ mc$ cd
Kevinqingkong:~ mc$ vi .bash_profile
Kevinqingkong:~ mc$ source .bash_profile
Kevinqingkong:~ mc$
```

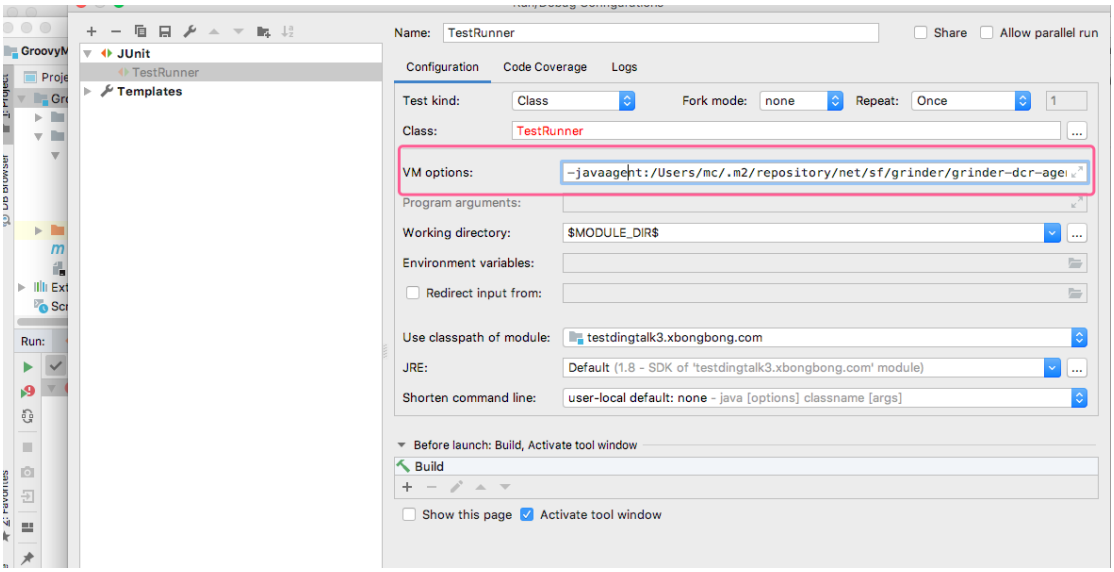
为了方便省事儿，我们可以在 Web 端创建 Groovy Maven 脚本，然后在 IDEA 中 checkout 出来就行。

从 SVN 迁出项目时的用户名和密码是登录 nGrinder 时的用户名密码(默认是 admin/admin)。

接下来，在 IDEA 下执行下我们的脚本吧

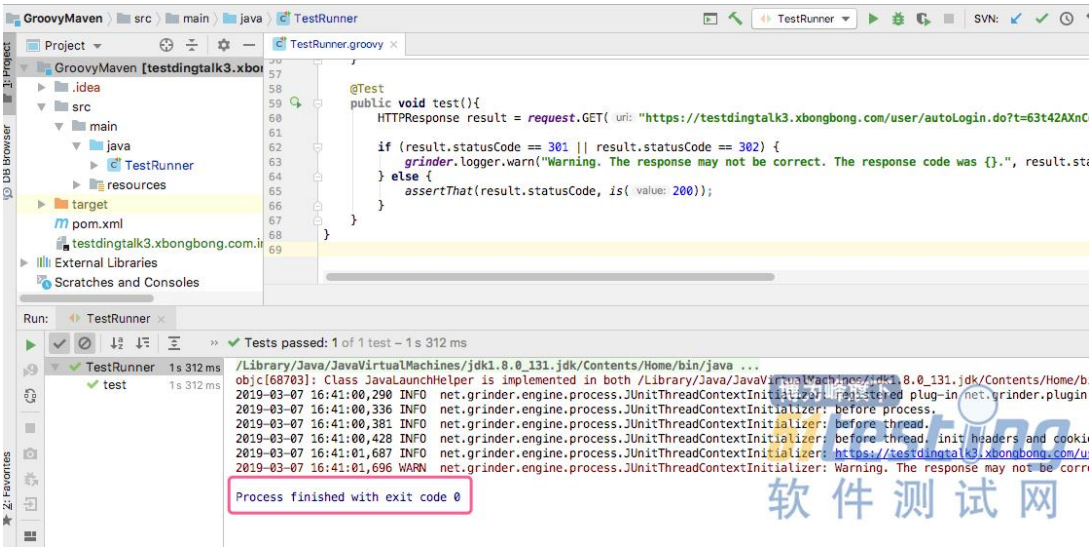


好像不行，查看错误信息后发现我们需要指定 Junit 运行时参数



将

-javaagent:/Users/mc/.m2/repository/net/sf/grinder/grinder-dcr-agent/3.9.1/grinder-dcr-agent-3.9.1.jar 填写在 VM Options 处。再来执行下脚本



完美执行！接下来我们一起看下如果在 IDEA 下开发 Groovy 脚本且使用 nGrinder 来实现性能测试过程中的常用压测场景。

1.5.0 ngrinder-groovy 脚本解释

```
import static net.grinder.script.Grinder.grinder
import static org.junit.Assert.*
import static org.hamcrest.Matchers.*
import net.grinder.plugin.http.HTTPRequest
import net.grinder.plugin.http.HTTPPluginControl
```

```
import net.grinder.script.GTest
import net.grinder.script.Grinder
import net.grinder.scriptengine.groovy.junit.GrinderRunner
import net.grinder.scriptengine.groovy.junit.annotation.BeforeProcess
import net.grinder.scriptengine.groovy.junit.annotation.BeforeThread
import org.junit.Before
import org.junit.BeforeClass
import org.junit.Test
import org.junit.runner.RunWith
import java.util.Date
import java.util.List
import java.util.ArrayList
import HTTPClient.Cookie
import HTTPClient.CookieModule
import HTTPClient.HTTPResponse
import HTTPClient.NVPair
@RunWith(GrinderRunner)
class TestRunner {
public static GTest testXBBTest
public static HTTPRequest request
public static NVPair[] headers = []
public static NVPair[] params = []
public static Cookie[] cookies = []
@BeforeProcess
public static void beforeProcess()
{HTTPPluginControl.getConnectionDefaults().timeout = 6000
testXBBTest = new GTest(1, "销帮帮测试环境")
request = new HTTPRequest()
grinder.logger.info("BeforeProcess 注解");}
@BeforeThread
public void beforeThread() {
testXBBTest.record(this, "testXBBTest")
grinder.statistics.delayReports=true;
```

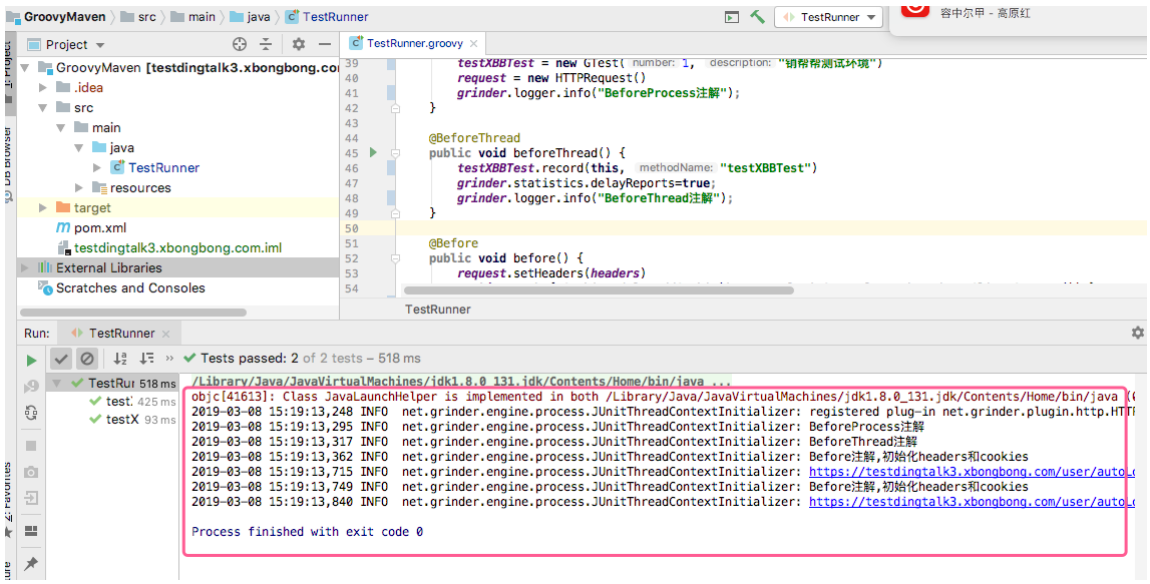
```
grinder.logger.info("BeforeThread 注解");}
@Before
public void before() {
request.setHeaders(headers)
cookies.each { CookieModule.addCookie(it, HTTPPluginControl.getThreadHTTPClientContext()) }
grinder.logger.info("Before 注解,初始化 headers 和 cookies");}
@Test
public void testXBBTest(){
HTTPResponse result =
request.GET("https://testdingtalk3.xbongbong.com/user/autoLogin.do?t=63t42AXnCd6hCEdQl+dOZC
XVIiwZVkRhX60J7lPXpv+xZt3tvEB1sT/ld8xQ0FUp7iqSfah+CNgNbYzcwW1d9Q==&nonce=wo0mgi",
params)
if (result.statusCode == 301 || result.statusCode == 302) {
grinder.logger.warn("Warning. The response may not be correct. The response code was {}.",
result.statusCode);
  } else {
assertThat(result.statusCode, is(200));
assertThat(result.text, containsString("欢迎"))}
}
@Test
public void testXBB(){HTTPResponse result =
request.GET("https://testdingtalk3.xbongbong.com/user/autoLogin.do?t=63t42AXnCd6hCEdQl+dOZCXVIiwZ
VkRhX60J7lPXpv+xZt3tvEB1sT/ld8xQ0FUp7iqSfah+CNgNbYzcwW1d9Q==&nonce=wo0mgi", params)
if (result.statusCode == 301 || result.statusCode == 302) {
grinder.logger.warn("Warning. The response may not be correct. The response code was {}.",
result.statusCode);
} else {assertThat(result.statusCode, is(200));
assertThat(result.text, containsString("欢迎"))}}}
```

划重点:

1: nGrinder 的 groovy 测试用例需要使用@RunWith()注解进行注释

2: 测试方法需要用@Test 注解进行注释，使用@Test 注解的方法将会被重复执行。

3: nGrinder 中每个线程只创建一个对象，每个被@Test 注解的方法可以共享成员变量。



nGrinder 中常用的注解如下

注解	描述	应用范围
@BeforeProcess	定义在进程被调用之前应执行的行为	static method
@AfterProcess	定义在进程被终止之前应执行的行为	static method
@BeforeThread	定义在每个线程被调用之前应执行的行为	member method
@AfterThread	定义在每个线程被终止之前应执行的行为	member method
@Before	定义每个被 @Test 注解的方法被执行前应执行的行为	member method
@After	定义每个被 @Test 注解的方法被执行后应执行的行为	member method
@Test	定义测试行为，被执行多次	member method

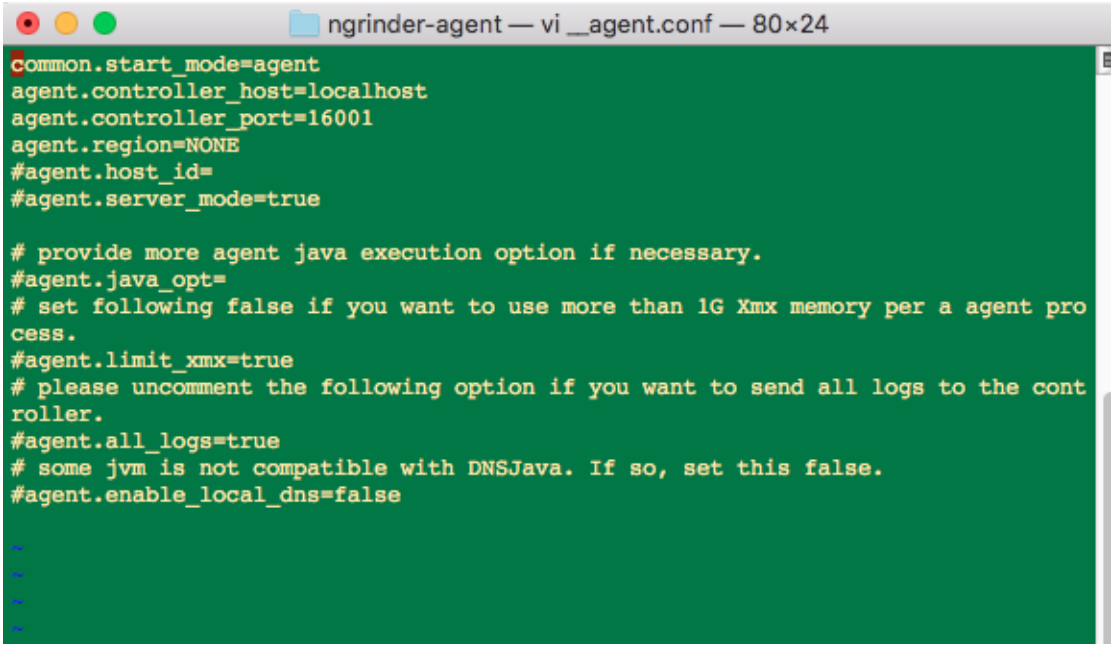
1.5.1 浪涌(阶梯)负载模拟

选择脚本创建测试，在测试配置界面选中 Ramp-Up 可用选项可以模拟简单的浪涌负载。

说实在的，nGrinder 对浪涌负载的模拟支持不友好，如果使用 Jmeter，浪涌场景的设置就非常丰富并且简单。

1.5.2 分布式执行

上图中我这边只是启动一个代理，nGrinder 会自动已注册的代理数。



如果我们要使用 nGrinder(多台代理机压测)集群，可以设置__agent.conf 也即是代理的配置文件指定 controller。随后，启动 agent 即可(*nix 系统下执行 run_agent.sh，windows 系统下执行 run_agent.bat)。

二：Locust

2.1 Locust 介绍

Locust 是一款易于使用的分布式负载测试工具，完全基于事件，即一个 locust 节点也可以在一个进程中支持数千并发用户，不使用回调，通过 gevent 使用轻量级过程（即在自己的进程内运行）。

特点

①、不需要编写笨重的 UI 或者臃肿的 XML 代码，基于协程而不是回调，脚本编写简单易读;

②、有一个基于 we 简洁的 HTML+JS 的 UI 用户界面，可以实时显示相关的测试结果;

③、支持分布式测试，用户界面基于网络，因此具有跨平台且易于扩展的特点;

④、所有繁琐的 I/O 和协同程序都被委托给 gevent，替代其他工具的局限性;

2.2 Locust 环境搭建

Locust 的环境配置非常简单，pip3 install -U locust 一条命令搞定。

我们可以通过 locust -h 看下 locust 支持的参数。

```
Kevinqingkong:~ mc$ locust -help
Usage: locust [options] [LocustClass [LocustClass2 ... ]]

Options:
  -h, --help            show this help message and exit
  -H HOST, --host=HOST  Host to load test in the following format:
                        http://10.21.32.33
  --web-host=WEB_HOST   Host to bind the web interface to. Defaults to '' (all
                        interfaces)
  -P PORT, --port=PORT, --web-port=PORT
                        Port on which to run web host
  -f LOCUSTFILE, --locustfile=LOCUSTFILE
                        Python module file to import, e.g. '../other.py'.
                        Default: locustfile
  --csv=CSVFILEBASE, --csv-base-name=CSVFILEBASE
                        Store current request stats to files in CSV format.
  --master              Set locust to run in distributed mode with this
                        process as master
  --slave               Set locust to run in distributed mode with this
                        process as slave
  --master-host=MASTER_HOST
                        Host or IP address of locust master for distributed
                        load testing. Only used when running with --slave.
                        Defaults to 127.0.0.1.
  --master-port=MASTER_PORT
                        The port to connect to that is used by the locust
                        master for distributed load testing. Only used when
                        running with --slave. Defaults to 5557. Note that
                        slaves will also connect to the master node on this
                        port + 1.
  --master-bind-host=MASTER_BIND_HOST
                        Interfaces (hostname, ip) that locust master should
                        bind to. Only used when running with --master.
                        Defaults to * (all available interfaces).
  --master-bind-port=MASTER_BIND_PORT
                        Port that locust master should bind to. Only used when
                        running with --master. Defaults to 5557. Note that
                        Locust will also use this port + 1, so by default the
                        master node will bind to 5557 and 5558.
  --expect-slaves=EXPECT_SLAVES
                        How many slaves master should expect to connect before
                        starting the test (only when --no-web used).
  --no-web              Disable the web interface, and instead start running
                        the test immediately. Requires -c and -r to be
                        specified.
  -c NUM_CLIENTS, --clients=NUM_CLIENTS
                        Number of concurrent Locust users. Only used together
```

2.3 Locust 初体验

我这边先以访问我司客户列表页做例子，摆弄下 locust 的流程。

1: 创建 py 文件，内容如下

```
# -*- coding = utf-8 -*-

from locust import HttpLocust, TaskSet, task

import hashlib

import requests

# 测试环境 Web 域名

web_host = 'https://testdingtalk3.xbongbong.com'

# 登录 Web 时请求的 URL, 每次操作钉钉重新登录时会变化,切记
```

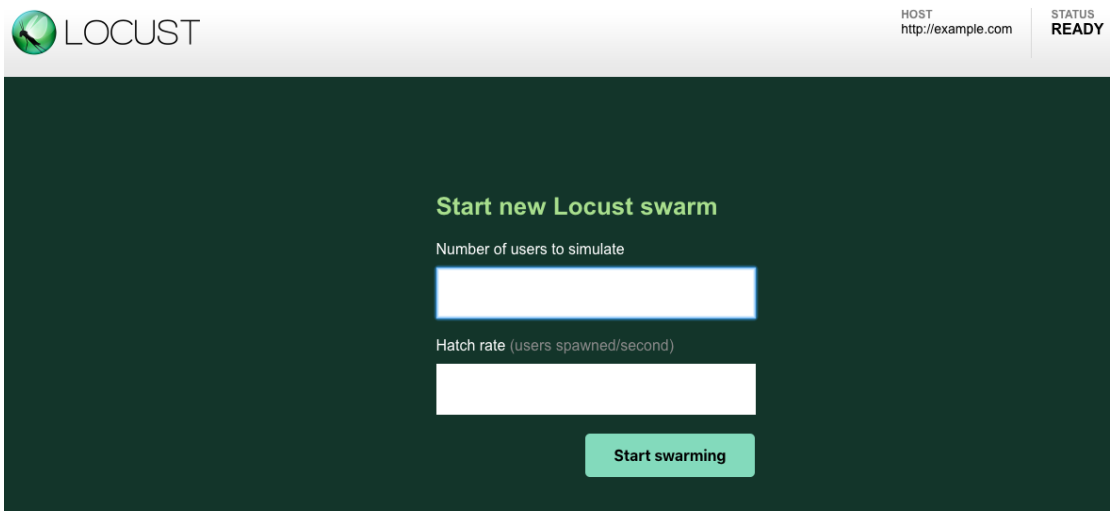
```
web_login_url='/user/autoLogin.do?t=2MxfguwveorRZqV/jW8r5Whg+i6Wcnp1zOWg+v1EXVz3zrSJTdvaI3SVPtMtSxfDIOiacByuNhDykBfyi7Gnog==&nonce=bvqyrp'

# 请求头中的 referer 信息，随登录时 url 变化而变动

web_header_referer =

'https://testdingtalkapi3.xbongbong.com//dingtalk/sns/userinfo.html?code=9331275bd3813f099f98fd9293d5fb15&state=STATE'

# 测试环境请求 header

# 注意： Referer 部分每次更新环境后会变化 抓包后替换改部分

web_login_headers = {"Host": "testdingtalk3.xbongbong.com", "Connection": "keep-alive",

"Upgrade-Insecure-Requests": "1",

"User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36",

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",

"Referer": web_header_referer, "Accept-Encoding": "gzip, deflate, br",

 "Accept-Language": "zh-CN,zh;q=0.8"}

# Web 后台 session 初始化

web_session = requests.session()

web_session.get(url=web_host + web_login_url, headers=web_login_headers, allow_redirects=False)

# 扫码登录时的 cookie 信息

web_cookie = web_session.cookies

# 登录 cookie 中的 xbbAccessToken 值,登录后所有请求都会用到这个值

web_access_token = web_cookie['xbbAccessToken']

# 登录 cookie 中的 JSESSIONID，后续所有请求中均用到

web_session_id = web_cookie['JSESSIONID']

# 生成 sign_code

def create_sign_code(request_parameters, *args):

if len(args) > 0:

parameters = str(str(request_parameters) + str(args[0])).encode('utf-8')

else:

parameters = str(request_parameters).encode('utf-8') return hashlib.sha256(parameters).hexdigest()

class Behavior(TaskSet):

def on_start(self):

Pass
```

```
def on_stop(self):
Pass
@task
def customer_list_on_web(self):
customer_list_body = '{\"corpid\":\"dinga93c84f4d89688bf35c2f4657eb6378f\",\"nowUserId\":\"0933106141722403490\",\"templateId\":175,\"page\":1,\"pageSize\":20,\"belongerType\":\"0\",\"isMain\":1,\"treeType\":\"\",\"pid\":\"\",\"nameLike\":\"\",\"nameLikeType\":\"\",\"isArchived\":0,\"child\":\"customer\",\"categoryId\":\"\"}'
sign_code = create_sign_code(customer_list_body, web_access_token)
customer_list_request_data = {"params": customer_list_body, "sign": sign_code, "platform": "web", "frontDev": 0,"JSESSIONID": web_session_id}
customer_list_url = "/customerApi/listAjax.html"
self.client.post(customer_list_url, data=customer_list_request_data, headers=web_login_headers, cookies=web_cookie)
class WebSiteUser(HttpLocust):
task_set = Behavior
min_wait = 5000
max_wait = 9000
```

2: 进入脚本所在目录执行如下命令

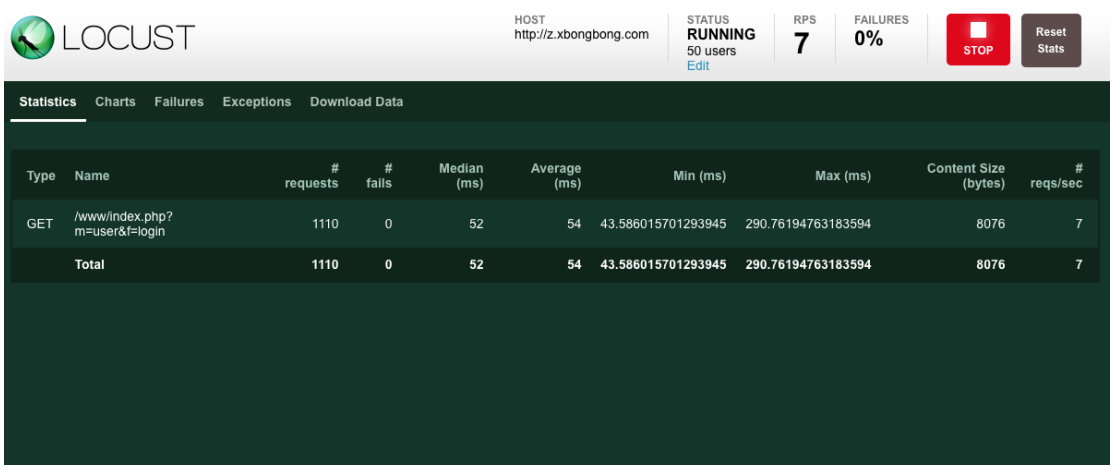
```
locust -f scenario_on_web.py --host=https://testdingtalk3.xbongbong.com
```

3: http://localhost:8089/



设置模拟的虚拟用户数和每秒启动的虚拟用户数

点击 Start swarming 开始执行测试

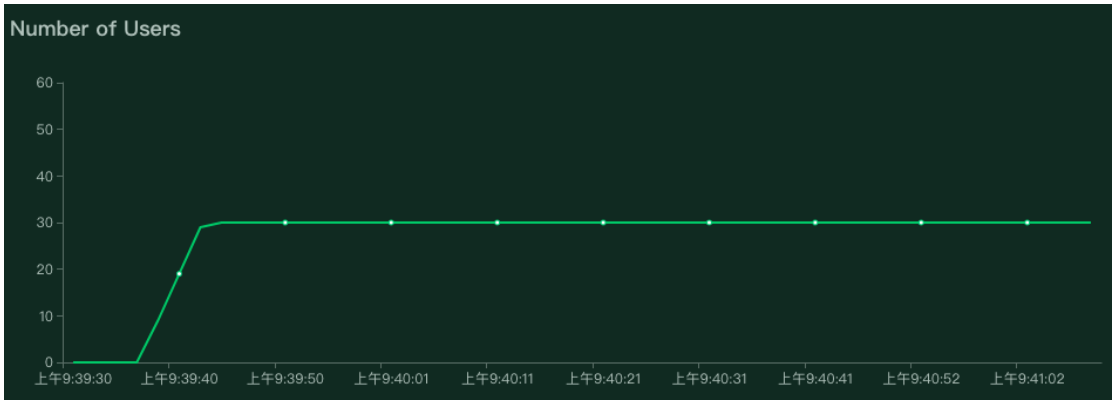


- Type：http(s)协议的方法，这里用到的是 get 方法
- Name：请求的 url 地址(不包括 host)。
- Requests：当前总请求数。
- Fails：当前总失败请求数。
- Mediam/average/Min/max：响应时间的中位值/平均值/最小值/最大值。
- Content size：请求内容总数，单位为字节。
- Reqs/sec：每秒请求数，即 QPS。

2.4 Locust 实战

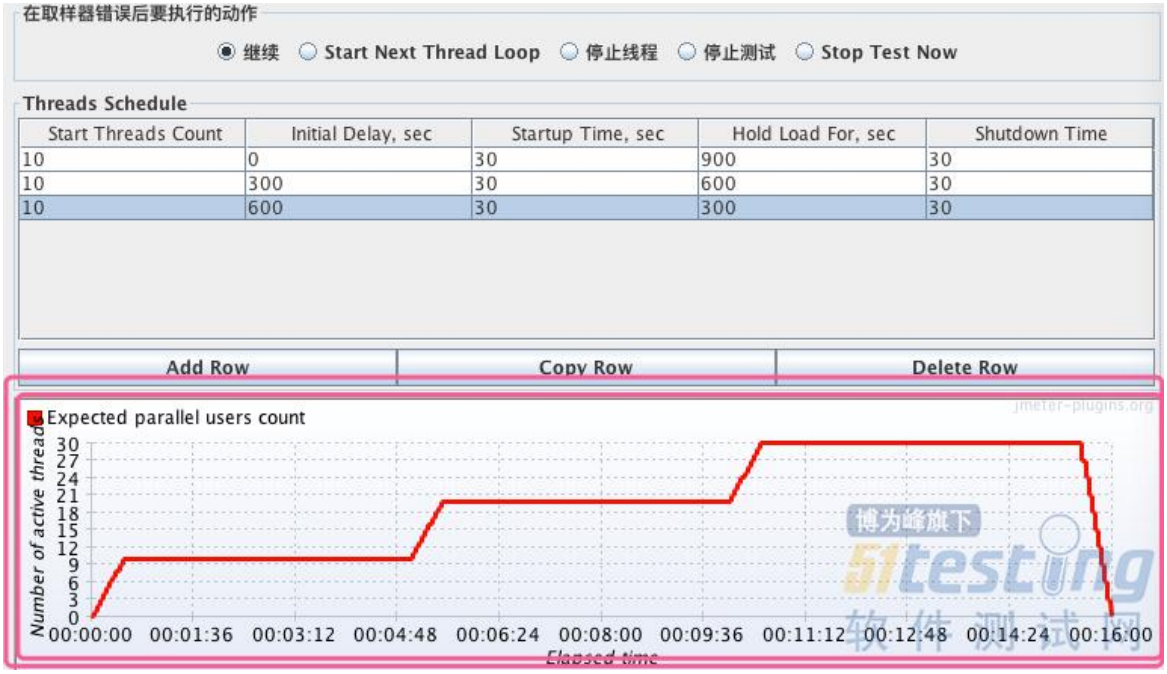
2.4.1 浪涌(阶梯)负载模拟

locust 中对阶梯式负载的模拟支持不是很友好，如果我们通过 locust web 设置，只能配置其初始化阶段也即：ramp-up 部分。



如上图所示，locust 可以模拟的测试场景相对来说比较简单。

如果我们需要模拟如下的负载，那就要想想办法咯~ 这里抛砖引玉希望熟悉这块儿的小伙伴们可以分享下踩过的坑



2.4.2 分布式执行

如果我们模拟高并发的场景，可能一台负载机是不够的这时候我们可以通过分布式压测来执行。Locust 是 master/slave 架构，接下来我们一起看下如何使用 locust 进行分布式压测。

Step 1: 启动 master

```
locust -f scenario_on_web.py --master --host=https://testdingtalk3.xbongbong.com
```

--master 参数表示当前机器(进程)的角色是 master。

master 不生成负载，它只负责任务调度和数据收集。

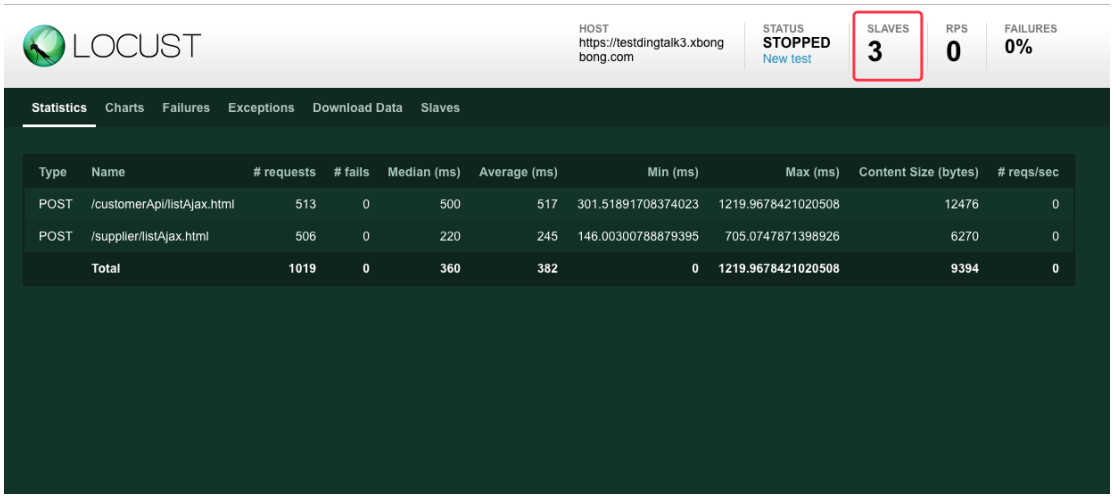
Step 2: 启动 slave

```
locust -f scenario_on_web.py --slave --master-host=192.168.10.175
```

--slave 参数表示当前机器是 locust 的 slave 节点。

Step 3: 设置压测数据

我们打开 locust web 来看下现在是什么情景，下图中显示当前 locust 服务有 3 台 slave 节点机器，我们设置模拟的虚拟用户总数会均分到这 3 台 slave 节点上。



PS: 需要注意的点是 master 和 salve 都需要有我们的脚本文件。

上面是以 locust web 服务的形式来执行压测，我们当然也可以通过 non-web 的方式来设置和执行压测场景。

非 Web UI 的方式执行 locust 时，我们经常会用到以下参数

locust -f scenario_on_web.py --no-web --host=https://testdingtalk3.xbongbong.com -c 10 -t 3m

--no-web: 以非 Web UI 方式执行 locust

-c: 并发用户数

-t: 执行时间

-r: 每秒启动用户数

--expect-slave: 期望的 slave 节点数(未达到期望节点数之前不执行压测)

以非 Web UI 方式执行分布式压测的命令如下:

locust -f scenario_on_web.py --no-web --master -c 10 -t 2m -r 1--host=https://testdingtalk3.xbongbong.com --expect-slaves=3

```
[2019-03-14 11:15:01,827] Kevinqingkong.local/INFO/locust.main: Time limit reached. Stopping Locust.
[2019-03-14 11:15:01,828] Kevinqingkong.local/INFO/locust.main: Shutting down (exit code 0), bye.
[2019-03-14 11:15:01,828] Kevinqingkong.local/INFO/locust.main: Cleaning up runner...
[2019-03-14 11:15:01,828] Kevinqingkong.local/INFO/locust.main: Running teardowns...
 Name                                                          # reqs      # fails     Avg     Min     Max  |  Median   req/s
--------------------------------------------------------------------------------------------------------------------------------------------
 POST /customerApi/listAjax.html                                 422     0(0.00%)     562     267    4267  |     470    4.00
 POST /supplier/listAjax.html                                    422     0(0.00%)     224     138     477  |     210    4.00
--------------------------------------------------------------------------------------------------------------------------------------------
 Total                                                           844     0(0.00%)                                        8.00

Percentage of the requests completed within given times
 Name                                                          # reqs    50%    66%    75%    80%    90%    95%    98%    99%   100%
--------------------------------------------------------------------------------------------------------------------------------------------
 POST /customerApi/listAjax.html                                 422    470    510    550    600    750    980   1800   3900   4300
 POST /supplier/listAjax.html                                    422    210    230    250    260    310    350    410    410    480
--------------------------------------------------------------------------------------------------------------------------------------------
 Total                                                           844    320    430    470    490    600    750   1100   1800   4300

Kevinqingkong:LocustScript mc$
```

三：**Gatling**

3.1 Gatling 介绍

Gatling 是一款基于 Scala 开发的高性能性能测试工具，它主要用于对服务器进行负载等测试，并分析和测量服务器的各种性能指标。Gatling 主要用于测量基于 HTTP 的服务器，比如 Web 应用程序，RESTful 服务等，它有商业版和开源免费版，本文将总结下开源免费版本 gatling 的实战。

Gatling 的特点:

1.支持 Akka Actors 和 Async IO，从而能达到很高的性能

2.支持实时生成 Html 动态轻量报表，从而使报表更易阅读和进行数据分析

3.支持 DSL 脚本，从而使测试脚本更易开发与维护

4.支持录制并生成测试脚本，从而可以方便的生成测试脚本

5.支持导入 HAR（Http Archive）并生成测试脚本

6.支持 Maven，Eclipse，IntelliJ 等，以便于开发

7.支持 Jenkins，以便于进行持续集成

8.支持插件，从而可以扩展其功能，比如可以扩展对其他协议的支持

Gatling 适用的场景包括：测试需求经常改变，测试脚本需要经常维护；测试环境的客户机性能不强，但又希望发挥硬件的极限性能；能对测试脚本进行很好的版本管理，并通过 CI 进行持续的性能测试；希望测试结果轻量易读等。

3.2 Gatling 环境搭建

在 https://gatling.io/download/页面点击下载。解压即可。

```
Kevinqingkong:gatling mc$ pwd
/Users/mc/gatling
Kevinqingkong:gatling mc$ ll
total 24
drwxr-xr-x@  9 mc  staff    306  3  4 21:24 .
drwxr-xr-x+ 97 mc  staff   3298  3  6 10:13 ..
-rw-r--r--@  1 mc  staff  11367  1 24 19:00 LICENSE
drwxr-xr-x@  6 mc  staff    204  1 24 19:00 bin
drwxr-xr-x@  6 mc  staff    204  3  4 21:22 conf
drwxr-xr-x@ 99 mc  staff   3366  1 24 19:00 lib
drwxr-xr-x@  4 mc  staff    136  3  4 21:40 results
drwxr-xr-x   4 mc  staff    136  3  4 21:24 target
drwxr-xr-x@  4 mc  staff    136  1 24 19:00 user-files
Kevinqingkong:gatling mc$
```

Bin: gatling 的可执行文件。

```
Kevinqingkong:conf mc$ cd ../bin
Kevinqingkong:bin mc$ ll
total 32
drwxr-xr-x@ 6 mc  staff   204  1 24 19:00 .
drwxr-xr-x@ 9 mc  staff   306  3  4 21:24 ..
-rwxr--r--@ 1 mc  staff  2993  1 24 19:00 gatling.bat
-rwxr--r--@ 1 mc  staff  2111  1 24 19:00 gatling.sh
-rwxr--r--@ 1 mc  staff  2112  1 24 19:00 recorder.bat
-rwxr--r--@ 1 mc  staff  1183  1 24 19:00 recorder.sh
Kevinqingkong:bin mc$
```

Conf: gatling 的配置文件

```
Kevinqingkong:bin mc$ cd ../conf
Kevinqingkong:conf mc$ ll
total 48
drwxr-xr-x@ 6 mc  staff   204  3  4 21:22 .
drwxr-xr-x@ 9 mc  staff   306  3  4 21:24 ..
-rw-r--r--@ 1 mc  staff   207  1 24 19:00 gatling-akka.conf
-rw-r--r--@ 1 mc  staff  8444  1 24 19:00 gatling.conf
-rw-r--r--@ 1 mc  staff   571  1 24 19:00 logback.xml
-rw-r--r--@ 1 mc  staff  3670  1 24 19:00 recorder.conf
Kevinqingkong:conf mc$
```

Lib: 依赖的包

```
Kevinqingkong:conf mc$ cd ../lib
Kevinqingkong:lib mc$ ll
total 130792
drwxr-xr-x@ 99 mc  staff      3366  1 24 19:00 .
drwxr-xr-x@  9 mc  staff       306  3  4 21:24 ..
-rw-r--r--@  1 mc  staff    123328  1 24 19:00 HdrHistogram-2.1.11.jar
-rw-r--r--@  1 mc  staff   5477882  1 24 19:00 Saxon-HE-9.9.1-1.jar
-rw-r--r--@  1 mc  staff   3472714  1 24 19:00 akka-actor_2.12-2.5.19.jar
-rw-r--r--@  1 mc  staff     16031  1 24 19:00 akka-slf4j_2.12-2.5.19.jar
-rw-r--r--@  1 mc  staff     23033  1 24 19:00 apple-file-events-1.3.2.jar
-rw-r--r--@  1 mc  staff    796532  1 24 19:00 bcpkix-jdk15on-1.60.jar
-rw-r--r--@  1 mc  staff   4189874  1 24 19:00 bcprov-jdk15on-1.60.jar
-rw-r--r--@  1 mc  staff    240397  1 24 19:00 boopickle_2.12-1.3.0.jar
-rw-r--r--@  1 mc  staff    660440  1 24 19:00 caffeine-2.6.2.jar
-rw-r--r--@  1 mc  staff    130663  1 24 19:00 commons-pool2-2.6.0.jar
-rw-r--r--@  1 mc  staff    212157  1 24 19:00 compiler-bridge_2.12-1.2.5.jar
-rw-r--r--@  1 mc  staff     91169  1 24 19:00 compiler-interface-1.2.5.jar
-rw-r--r--@  1 mc  staff    286090  1 24 19:00 config-1.3.3.jar
-rw-r--r--@  1 mc  staff      5246  1 24 19:00 fast-uuid-0.1.jar
-rw-r--r--@  1 mc  staff    105028  1 24 19:00 fastring_2.12-1.0.0.jar
-rw-r--r--@  1 mc  staff     63659  1 24 19:00 gatling-app-3.0.3.jar
```

Result: 存放测试结果

User-files: 存放测试场景文件。

3.3 Gatling 初体验

gatling 自带的有个例子，我们进入到 bin 目录下执行 gatling.sh 执行测试。

```
Kevinqingkong:bin mc$ ./gatling.sh
GATLING_HOME is set to /Users/mc/gatling
Choose a simulation number:
     [0] computerdatabase.BasicSimulation
     [1] computerdatabase.advanced.AdvancedSimulationStep01
     [2] computerdatabase.advanced.AdvancedSimulationStep02
     [3] computerdatabase.advanced.AdvancedSimulationStep03
     [4] computerdatabase.advanced.AdvancedSimulationStep04
     [5] computerdatabase.advanced.AdvancedSimulationStep05
0
Select run description (optional)

Simulation computerdatabase.BasicSimulation started...

================================================================================
2019-03-06 14:35:06                                           5s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=2      KO=0     )
> request_1                                                (OK=1      KO=0     )
> request_1 Redirect 1                                     (OK=1      KO=0     )

---- Scenario Name -------------------------------------------------------------
[--------------------------------------------------------------------------]  0%
          waiting: 0      / active: 1      / done: 0
================================================================================


================================================================================
2019-03-06 14:35:11                                          10s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=2      KO=0     )
```

选择 0，然后两次回车键开始执行压测。

我们不用关心什么时候执行完毕。执行完成后 gatling 会自动退出进程并生成报告。

```
Simulation computerdatabase.BasicSimulation completed in 27 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                         13 (OK=13     KO=0     )
> min response time                                    290 (OK=290    KO=-     )
> max response time                                    831 (OK=831    KO=-     )
> mean response time                                   409 (OK=409    KO=-     )
> std deviation                                        178 (OK=178    KO=-     )
> response time 50th percentile                        320 (OK=320    KO=-     )
> response time 75th percentile                        337 (OK=337    KO=-     )
> response time 95th percentile                        742 (OK=742    KO=-     )
> response time 99th percentile                        813 (OK=813    KO=-     )
> mean requests/sec                                0.464 (OK=0.464  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                           12 ( 92%)
> 800 ms < t < 1200 ms                                  1 (  8%)
> t > 1200 ms                                           0 (  0%)
> failed                                                0 (  0%)
================================================================================

Reports generated in 1s.
Please open the following file: /Users/mc/gatling/results/basicsimulation-20190306063501237/index.html
Kevinqingkong:bin mc$
```

打开 gatling

自动生成的报告

哇~简直是惊艳！非常详细且漂亮的报告

3.4 Gatling 实战

gatling 的脚本是 Scala 语言，我们可以使用 IDEA 安装 Scala 插件。也可以去 Scala 官网(https://www.scala-lang.org/download/)下载 tar 包。

我这里选择的是后者，将 scala 的 tar 包解压后配置一下环境变量

```
export SCALA_HOME=/Users/mc/scala
export PATH=$PATH:$SCALA_HOME/bin
```

执行 source .bash_profile 即可。

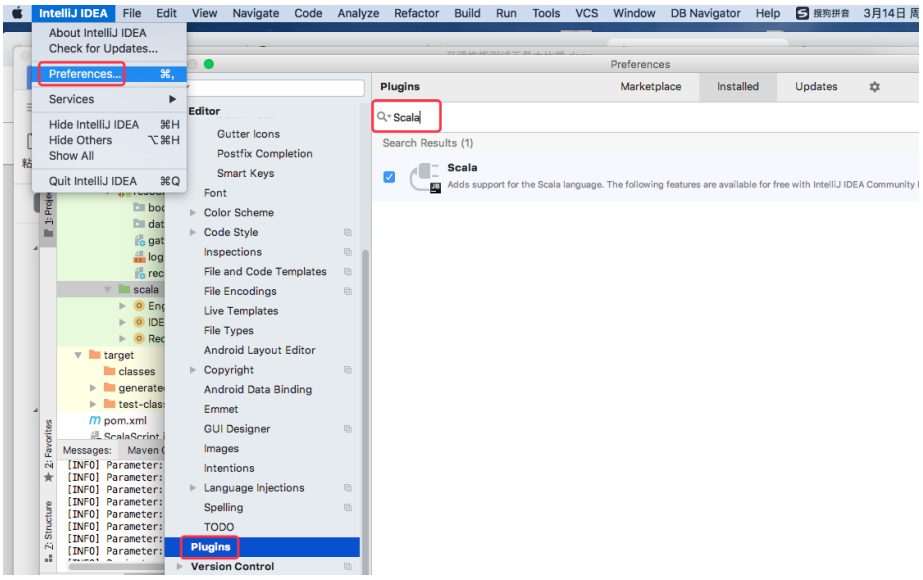
对于 Scala 语言的学习，本文就不总结了，小伙伴们自行学习吧~

使用 IDEA 开发 Scala 脚本时需要一些配置步骤，如下执行：

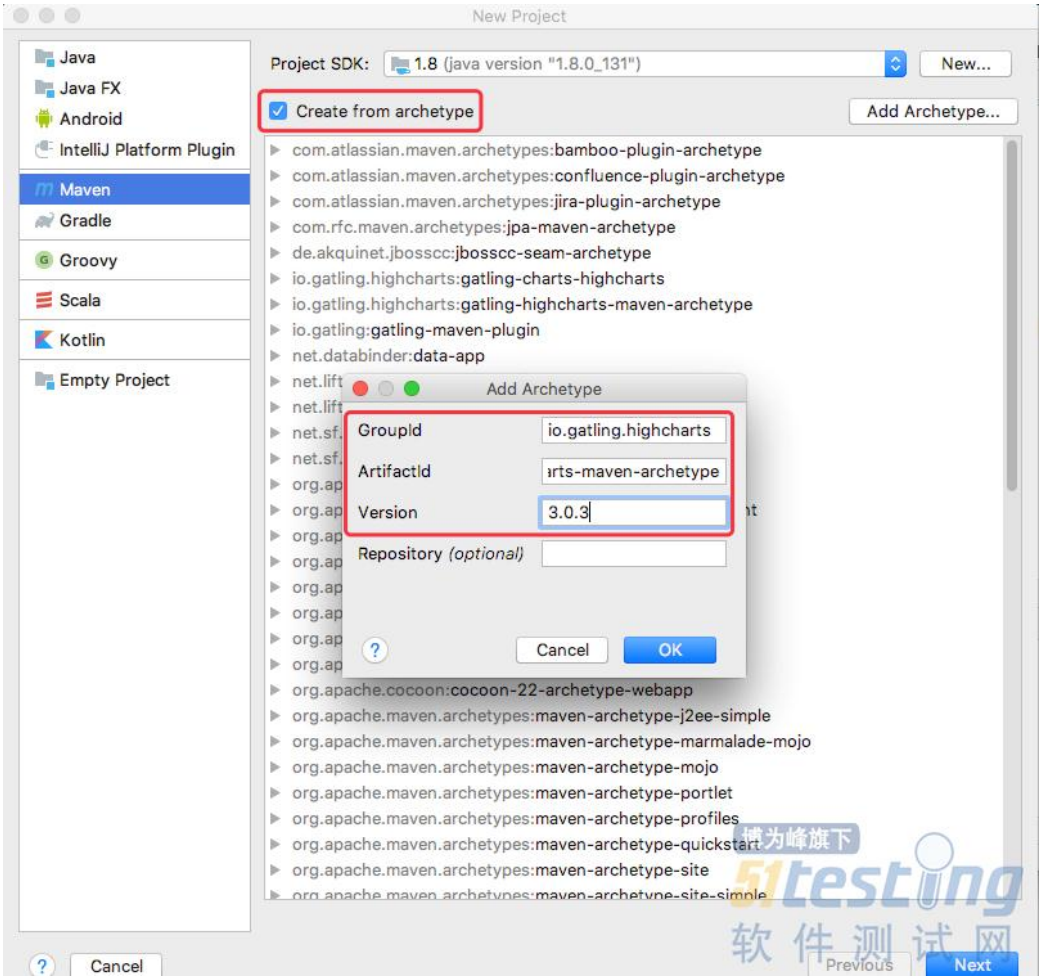
Step 1: 配置 maven(略过…)

Step2: 安装 IDEA(略过…)

Step3: 在 IDEA 的 plugins 界面中搜索 Scala 并安装



Step4: 根据 gatling 模板创建 gtling-scala 项目

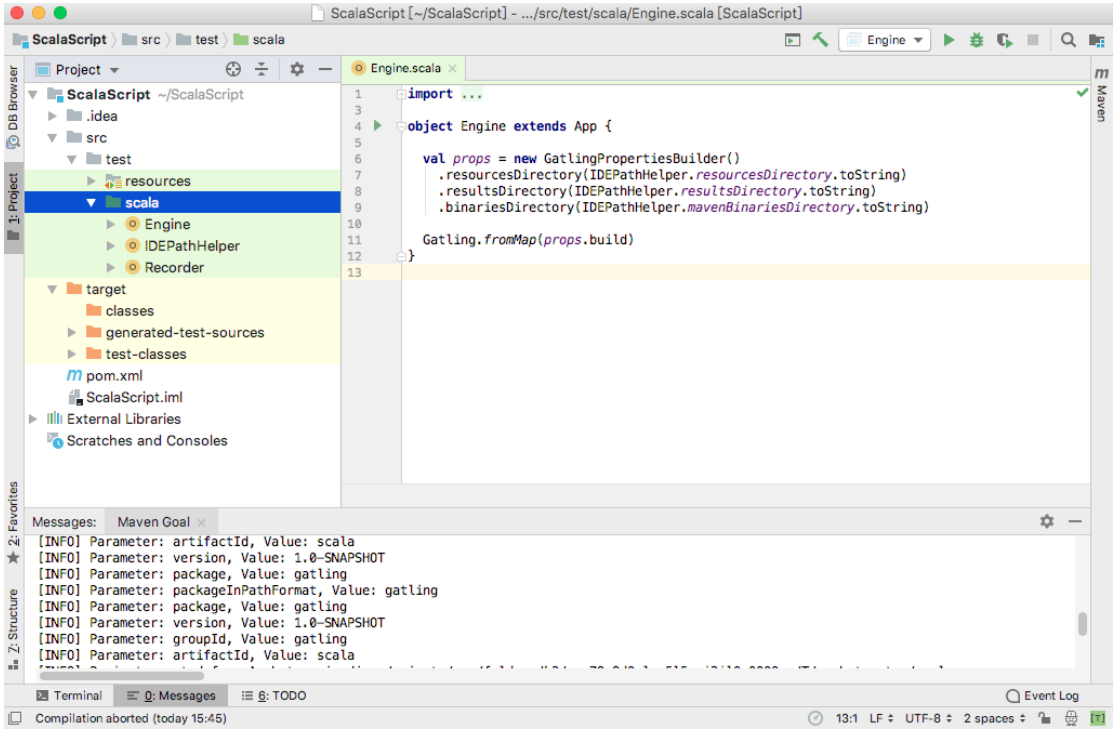


Group-id 填写为：io.gatling.highcharts

ArtifactId 填写为：gatling-highcharts-maven-archetype

Version 填写为：3.0.3(请查看最新版本，当然目前最新是 3.0.3)

模板工程生成后，工程目录结构如下:



脚本文件在 src/test/scala 目录下。

3.4.1 Gatling 常用 API

Gatling 的脚本有 3 大部分组成：header，scenario，setUp。

Header: 组装请求头。

Scenario: 具体业务执行。

SetUp: 设置执行场景。

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._
class CustomerList extends Simulation {val httpProtocol =
http.baseUrl("https://testdingtalk3.xbongbong.com")
.acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
.doNotTrackHeader("1")
.acceptLanguageHeader("en-US,en;q=0.5")
.acceptEncodingHeader("gzip, deflate")
.userAgentHeader("Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
```

Gecko) Chrome/59.0.3071.115 Safari/537.36")

val headers_10 = Map("Content-Type" -> "application/x-www-form-urlencoded")

val business_process=scenario("业务流压测")

.exec(http("钉钉扫码登录
").get("/user/autoLogin.do?t=63t42AXnCd6hCEdQl+dOZCXVIiwZVkRhX60J7lPXpv+xZt3tvEB1sT/ld8xQ0FUp7iqSfah+CNgNbYzcwW1d9Q==&nonce=jmymnp"))

.pause(1, 5).exec(http("客户列表").post("/customerApi/listAjax.html").body(StringBody("{\"sign\":\"c8fb3b2bdb32400069161840f55a525f707ae04abefccaf03a46da6072805099\", \"JSESSIONID\":\"fu9f7l5zt5pc4ca85g8fkqqtp41vhbtz\", \"frontDev\": \"0\", \"platfrom\": \"web\", \"params\":{\"corpid\":\"dinga93c84f4d89688bf35c2f4657eb6378f\",\"nowUserId\":\"030917160122954929\",\"templateId\":175,\"page\":1,\"pageSize\":20,\"belongerType\":\"0\",\"isMain\":1,\"treeType\":\"\",\"pid\":\"\",\"nameLike\":\"\",\"nameLikeType\":\"\",\"isArchived\":0,\"child\":\"customer\",\"categoryId\":\"\"}}")).check(status.is(200)))

.pause(1, 5)

.exec(http("合同列表").post("/contractApi/listAjax.html").check(status.is(200)))

setUp(business_process.inject(constantUsersPerSec(10).during(100)).protocols(httpProtocol))}

header 部分常用的 api 如下:

baseurl: 设置域名

acceptHeader: 请求头的 accept。

acceptLanguageHeader: 请求头中的 Language

acceptEncodingHeader: 请求头的编码

Scenario 部分常用的 api:

exec: 执行(发送请求)

body: 设置请求报文的 body

post: 模拟 http(s)的 post 方法

get: 模拟 http(s)的 get 方法

headers: 设置请求头信息

formParam: 设置请求报文的 body 部分(以表单形式)

feed: 提供后续请求的参数(参数化时使用)

check: 检查响应报文内容(类似断言)

during: 模拟压测时长

setUp 部分常用 api:

setUp 部分是模拟用户使用场景的设置。

先来看下 Gatling 中所有的场景模拟:

```
setUp(
scenario.inject(
// 在一段时间内不做任何事情
nothingFor(4 seconds),
// 一次性启动指定数量的虚拟用户数
atOnceUsers(10),
// 5 秒内启动 10 个虚拟用户
rampUsers(10) over(5 seconds),
// 保持 20 个并发虚拟用户数，持续 15 分钟
constantUsersPerSec(20) during(15 minutes),
// 在 20 个并发虚拟用户基数上随机递减，持续 15 分钟
constantUsersPerSec(20) during(15 minutes) randomized,
// 在 10 分钟内，虚拟用户数从 10 递增到 20
rampUsersPerSec(10) to 20 during(10 minutes),
// 在 10 分钟内，虚拟用户数从 10 上升到 20(随机增加虚拟用户数)
rampUsersPerSec(10) to 20 during(10 minutes) randomized,
// 下面这两个。。。理解不了。。。
splitUsers(1000) into(rampUsers(10) over(10 seconds)) separatedBy(10 minutes),
splitUsers(1000) into(rampUsers(10) over(10 seconds)) separatedByatOnceUsers(30),
).protocols(httpConf) )
```

3.4.3 分布式执行

非常抱歉，Gatling 不支持分布式执行~

四：wrk

4.1 wrk 环境搭建

wrk 的仓库地址是：https://github.com/wg/wrk

我使用的是 Mac，所以只能以 Mac OS 来说明下 wrk 的搭建了。

brew install wrk

4.2 wrk 初体验

安装好 wrk 后，咱们先看下 wrk 支持的参数吧，wrk 回车键即可。

```
Kevinqingkong:~ mc$ wrk
Usage: wrk <options> <url>
  Options:
    -c, --connections <N>  Connections to keep open
    -d, --duration    <T>  Duration of test
    -t, --threads     <N>  Number of threads to use

    -s, --script      <S>  Load Lua script file
    -H, --header      <H>  Add header to request
        --latency          Print latency statistics
        --timeout     <T>  Socket/request timeout
    -v, --version          Print version details

  Numeric arguments may include a SI unit (1k, 1M, 1G)
  Time arguments may include a time unit (2s, 2m, 2h)
Kevinqingkong:~ mc$
```

-c: 保持的链接数。

-d: 压测时长。

-t: 压测时使用的线程数。

-s: lua 脚本文件路径。

-H: --header 请求头

--latency 输出响应延迟信息

--timeout 请求超时时长

模拟 get 请求:

wrk -t10 -c20 -d30s --latency http://z.xbongbong.com/www/index.php?m=user&f=login

```
Kevinqingkong:~ mc$ wrk -t10 -c20 -d30s --latency http://z.xbongbong.com/www/index.php?m=user&f=login
[1] 88667
Kevinqingkong:~ mc$ Running 30s test @ http://z.xbongbong.com/www/index.php?m=user
  10 threads and 20 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   150.86ms   15.04ms 234.20ms   74.94%
    Req/Sec    13.22      4.96    20.00     60.85%
  Latency Distribution
     50%  149.41ms
     75%  158.57ms
     90%  168.82ms
     99%  197.38ms
  3978 requests in 30.10s, 2.97MB read
Requests/sec:    132.17
Transfer/sec:    101.06KB
```

wrk 的高级用法是使用 Lua 脚本实现 set_up/running/stop 三大部分的自定义，因为水平

有限，这里我就不班门弄斧了。

五：Jmeter

Jmeter 这里我就不多写什么了，之前笔者写过 Jmeter 实战的文章，感兴趣的小伙伴们可以下载第 50 期的《51 测试天地》期刊 http://www.51testing.com/html/33/n-3958933.html 这里下载~不但有 Jmeter 实战还有其他优秀的同学们分享的各种测试 topic~

六：工具对比选型

压测框架	脚本语言	支持协议	分布式执行	脚本扩展
nGrinder	Groovy/Python	http(s)，rpc，grpc	Controller/Agent	Groovy/Python
Locust	Python	http(s)，websocket	Master/slave	Python
Gatling	Scala	http(s)，其他需自己扩展	不支持	Scala
Wrk	Lua	http(s)	不支持	Lua
Jmeter	Java/BeanShell	多种协议	C/S 或命令行	Java/BeanShell

想使用界面操作的形式对我的系统做性能测试，并且希望测试数据有良好的可视化展示方式：建议使用 Jmeter 工具

性能基准测试：建议使用 wrk 工具

对系统模拟复杂场景的性能测试；建议使用 locust 工具

压测的同时，监控服务器性能指标：建议使用 Jmeter 工具

使用匀速请求的方式，对系统进行性能测试；建议使用 Jmeter 或 locust

体验编程的乐趣，自己编写脚本进行性能测试；Jmeter：使用 Java 请求，自由扩展。locust，使用 Python 语言编写脚本。

把玩了这么多，我个人比较推崇的是 Locust 和 Jmeter，当然需要根据小伙伴们的实际情况略加考虑~

PYSNMP 模拟器实战

◆作者：刘丽

摘要:
Snmp 模拟器是适用于模拟网络上大量的不同类型的支持 SNMP 设备响应请求的过程，其原理通过监听本地的本地 IP 接口和/或 UDP 端口，一旦 SNMP 请求进入，SNMP 模拟器查找 snmprec 文件并以 snmp 协议格式响应给服务器。

SNMP 模拟器依赖于 Python 库，开源免费，易于扩展，在实际应用中涉及大批量设备与网管交互测试时是首选方案。本文结合项目需求阐述其应用，文中包含了对以下请求的响应（get-request、get-next-request、set-request）。

背景: 需要模拟多台设备并发 8 条业务流的过程

操作步骤:

1、环境安装:

详参 http://snmplabs.com/snmpsim/，本文以 Python2.7,windows 安装环境为例

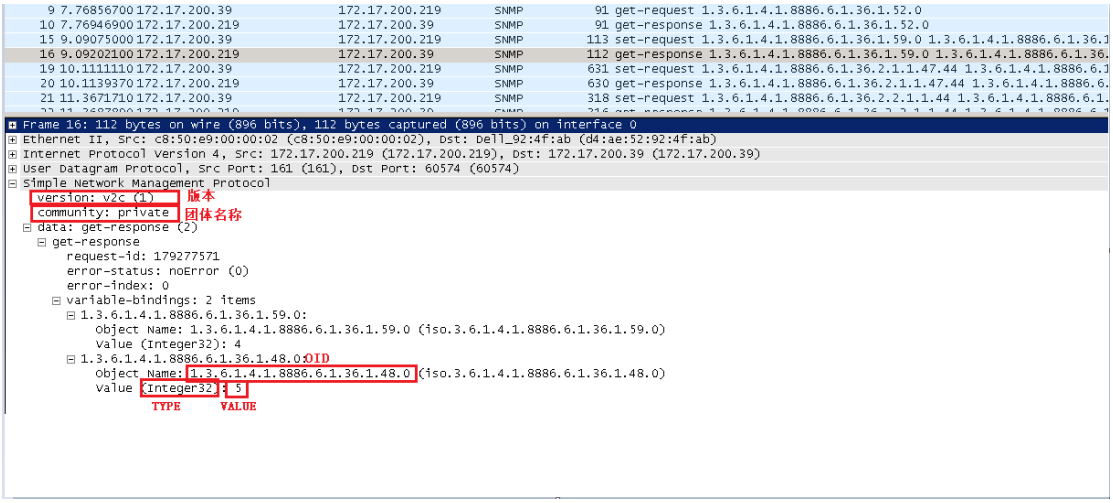
2、构造 snmpprc 文件

该文件是把获取设备 mib，以 OID | TYPE | VALUE 形式的单个 SNMP 对象组织在一起的文本文件，以.snmprec 作为后缀。

获取设备 mib 的方法有三种:

1）手工创建

小编是结合 wireshark 抓包软件，抓取真实报文后分析 SNMP 报文来获取，以下做重点介绍:



通过以上报文分析可以得出:

① SNMP 版本为 v2c,

② SNMP 读写团体字为 private,故该 mib 节点应该放在 private.snmprec

③ 根据标红的 mib,生成的单个 SNMP 对象是:

1.3.6.1.4.1.8886.6.1.36.1.48.0|2|5

PS:TYPE 对应的 ASN.1 如下:

Integer32 - 2

OCTET STRING - 4

NULL - 5

OBJECT IDENTIFIER - 6

IpAddress - 64 Counter32 - 65

Gauge32 - 66

TimeTicks - 67

Opaque - 68

Counter64 – 70

同理，整个过程的报文逐条分析，将每个 mib 生成 SNMP 对象，最后组织在一起，生成 private.snmprec 和 public.snmprec。

2）真实设备模拟

通过如下命令生成

snmprec.py --agent-udpv4-endpoint=192.168.20.221:161 --output-file=./data/demo.snmprec

--protocol-version=2c --community=public

3）MIB 文件构建

参 http://snmplabs.com/snmpsim/quickstart.html#simulate-from-mib，此处小编未做验证。

3、指定文件目录

手动将以上生成的 private.snmprec 和 public.snmprec 数据文件放入以下目录:

<Python-package-root>/data 下，在本实例中便于管理，统一放在..\data\Y1564\RAX711C

4、运行模拟器

cd /d C:\Python27\Scripts snmpsimd.py --data-dir=..\data\Y1564\RAX711C --args-from-file=..\data\DeviceIp\Y1564\RAX711C.txt

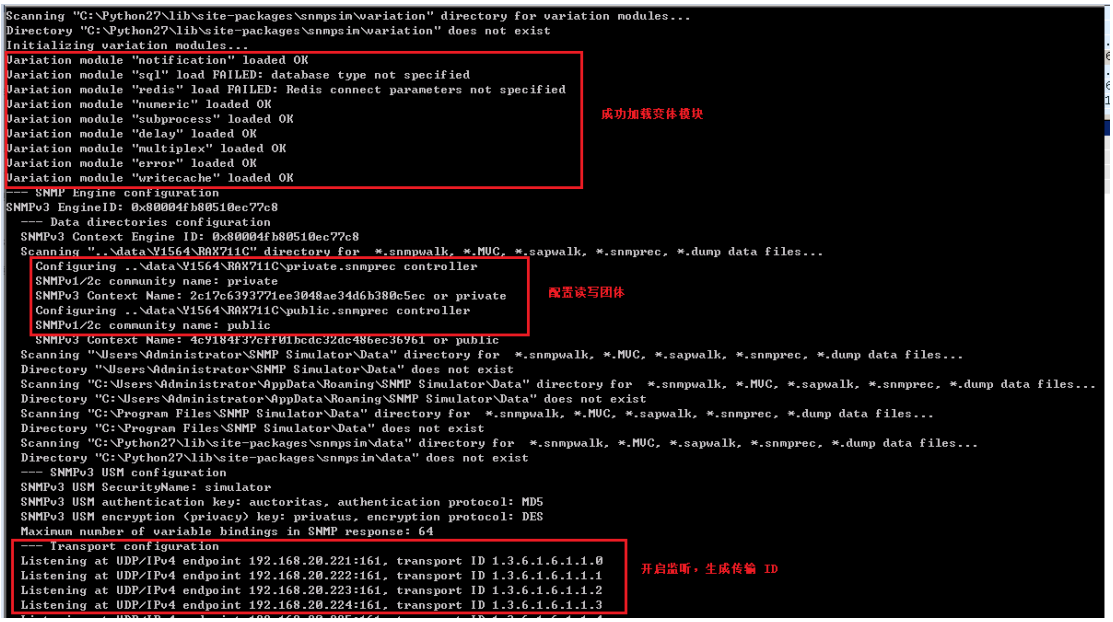
模拟器可以监听多个本地 IP 接口和/或 UDP 端口。需要传递多个 --agent-udpv4-endpoint / --agent-udpv6-endpoint 命令行参数，生成一个 txt 文件，绑定 --args-from-file 下

RAX711C.txt

```
1  --agent-udpv4-endpoint=192.168.20.221:161
2  --agent-udpv4-endpoint=192.168.20.222:161
3  --agent-udpv4-endpoint=192.168.20.223:161
4  --agent-udpv4-endpoint=192.168.20.224:161
5  --agent-udpv4-endpoint=192.168.20.225:161
6  --agent-udpv4-endpoint=192.168.20.226:161
7  --agent-udpv4-endpoint=192.168.20.227:161
8  --agent-udpv4-endpoint=192.168.20.228:161
9  --agent-udpv4-endpoint=192.168.20.229:161
10 --agent-udpv4-endpoint=192.168.20.230:161
11 --agent-udpv4-endpoint=192.168.20.231:161
12 --agent-udpv4-endpoint=192.168.20.232:161
13 --agent-udpv4-endpoint=192.168.20.233:161
14 --agent-udpv4-endpoint=192.168.20.234:161
15 --agent-udpv4-endpoint=192.168.20.236:161
16 --agent-udpv4-endpoint=192.168.20.238:161
17 --agent-udpv4-endpoint=192.168.20.239:161
18 --agent-udpv4-endpoint=192.168.20.240:161
19 --agent-udpv4-endpoint=192.168.20.241:161
20 --agent-udpv4-endpoint=192.168.20.242:161
21 --agent-udpv4-endpoint=192.168.20.244:161
```

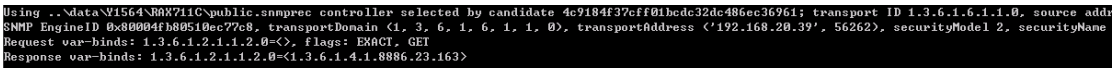
5、开始监听

以本文中实例（v2c）为例,正常启动如下所示



传输 ID 是标识本地传输端点的 OID，模拟器在启动时会报告它正在侦听的每个端点。

6、响应请求



如截图所示，当服务器 192.168.20.39 使用 “public” 团体向 192.168.20.221 模拟器发送 SNMP 请求时，模拟器从特定数据文件 public / 1.3.6.1.6.1.1.0 / 192.168.20.39 将用于构建响应（由 Simulator 根据传输 ID 1.3.6.1.6.1.1.0 报告）。

7、变体模块的使用

可以通过变体模块可以实现高度复杂行为，PYSNMP 中变体模块可扩展，详参：http://snmplabs.com/snmpsim/simulation-with-variation-modules.html#standard-variation-modules

本文针对实例中使用场景对部分变体模块的使用做简要说明：

① numeric：生成随时间变化的整数值，用于 INTEGER，Counter32，Counter64，Gauge32，TimeTicks 对象

例如：

1.3.6.1.4.1.8886.6.1.36.1.52.0|2:numeric|min=1,max=8,offset=0,wrap=1

说明：该节点用于生成 8 条业务的索引值，最小为 1，最大为 8，偏差为 0，在 0-8 循环使用；结合当前业务的话，每次产生的索引是一个随机值，不能实现自增的顺序增长，

可能会重复，该问题暂时没解决，需要扩展。

② writecache：OID 在.snmprec 通过 SNMP SET 操作文件写入内存中

例如:

1.3.6.1.4.1.8886.6.1.36.2.3.1.2.1|2:writecache|value=1,op=set

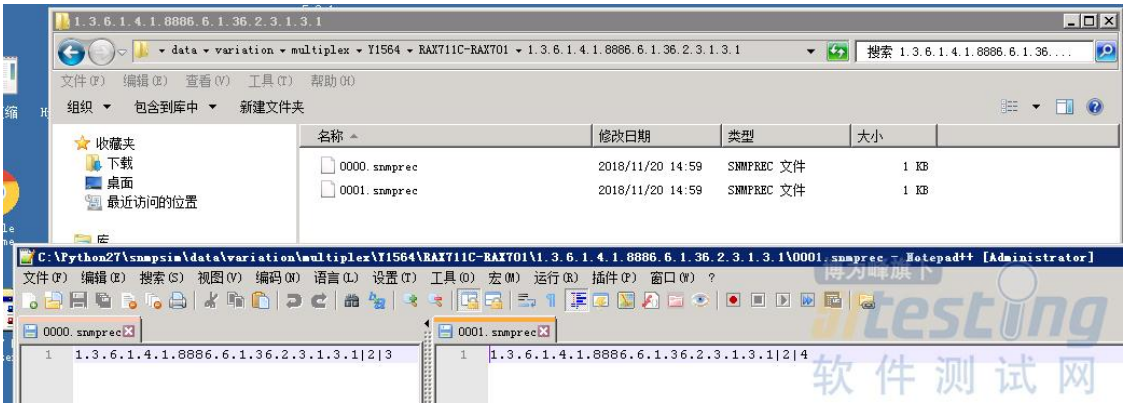
说明：该节点用于指定业务的标志位，初始值是 1，通过 set 操作修改值，再次用 get 获取时值为 Manager 配置的值

③ multiplex：单个代理服务多个快照，一次只挑选一个快照来回答 SNMP 请求。模拟了更自然的代理行为，包括随时间变化的 OID 集。

例如:

1.3.6.1.4.1.8886.6.1.36.2.3.1.3.1|2:multiplex|dir=C:\Python27\snmpsim\data\variation\multiplex\Y1564\RAX711C-RAX701\1.3.6.1.4.1.8886.6.1.36.2.3.1.3.1,period=1,wrap=true

说明：该节点是业务的状态位，3 代表进行中，4 代表已结束，同一个节点需要返回不同的值，结合当前业务，在标志位控制下只有获取到想要的想要的状态位才能进入下一步流程，否则就继续请求状态位，故使用快照来处理，详细的使用如下截图所示:



8、常见问题分析与解决

该章节汇总在实际操作中遇到的问题

1）基本环境配置

起码要保证模拟器和 Manager 之间能 ping 通;

对于监听多个 IP 接口时，需要将其逐个添加到网卡

2）模拟的返回节点报 no such instance

当节点不存在时回报该错误。

首先通过抓包，检查所属的团体字，在对应的.snmprec 文件下查找该节点是否存在。

3）get-next-request 请求的响应

.snmprec 文件的查找是基于上下文，get-next-request 请求存在顺序，对于请求内容是整表下的第一行数据，需要用 multiplex 指定到到第一行数据；对于请求的内容是下一行（非第一行）数据，返回就是.snmprec 文件中该节点的下一行。

例如，依次通过 get-next-request 请求表里面的所有内容如下:

1.3.6.1.4.1.8886.6.1.36.3.20.1.1|:multiplex|dir=C:\Python27\snmpsim\data\variation\multiplex\Y1564\RAX711C-RAX701\1.3.6.1.4.1.8886.6.1.36.3.20.1.1,period=1,wrap=true

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.1|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.2|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.3|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.4|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.5|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.6|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.7|2|2

1.3.6.1.4.1.8886.6.1.36.3.20.1.1.8|2|2

备注：1.3.6.1.4.1.8886.6.1.36.3.20.1.1 下的快照即第一行数据:
1.3.6.1.4.1.8886.6.1.36.3.20.1.1

4）同步问题

实际测试中模拟多台设备同时并发时，发现 SNMP 请求时同步执行。

该问题，通过查找相关 API,可知：SNMP Simulator 是一个单线程应用程序，这意味着它只能一次处理单个请求。最重要的是，一些变体模块可能会给请求处理带来额外的延迟，这可能导致后续请求在输入队列中累积并导致延迟增加。为了最大化吞吐量和最小化延迟，可以绑定到不同 IP 接口或端口的 snmpsimd.py 的多个实例，这有效地使 SNMP 模拟器一次执行多个请求，只要它们被发送到不同的 snmpsimd.py 实例。

5）基于 IP 接口文件的配置

实际应用中发现，IP 接口文件有个数限制，建议最多配置 500 个。

记一次难忘的腾讯面试经历

◆作者：咖啡猫

题记:
作为一个计算机科班出身的测试猿，毕业后一直混迹于二线城市的二线互联网公司，每次看到技术群或者朋友圈里 BATH 的同学晒工作照、年会照、团建照，永远是“身不能至，心向往之”的感觉。终于在两年前的夏天，我决定换个工作环境，有幸参加了深圳腾讯的面试，当时笔者由于准备不充分，很遗憾没能最终拿到 OFFER，但是当时的我入行不到两年，能够杀入终面已经非常幸运了，所以笔者愿意把这段面试经历写出来，一方面作为个人总结，另一方面也是希望能够给广大测试圈儿的朋友们一点启示。

第一关：简历　　结果：PASS

之所以把“简历”作为第一关，原因其实很简单：如果简历都写得毫无逻辑、毫无特色，HR 自然就会把你的简历筛掉。我当时花了一天的时间来整理简历，简历一般不超过两页，毕竟大公司的 HR 们工作都很忙，你就算简历写了十几页人家也没时间看，**我的简历大概分几个模块:**

1、个人信息: 包括姓名、性别、年龄、邮箱、电话、住址、工作经验、学历、学校、身体状况、英语水平、期望薪水等等信息，总之就是要简单明了，照片就用登记照就可以了，过度美颜的或者大头贴的照片还是不要了。

2、工作经历: 这里我就按时间先后顺序把自己呆过的几家公司名称、岗位、薪水、在职时间等信息列了出来。这里要特地提醒一下大家，过于频繁地跳槽真的不太好，比如我曾经做面试官时，就碰到有同学一年跳三次，结果领导直接说这种就没必要面试了，太不稳定。也许有的同学会问，那我可不可以稍微“修饰”一下工作经历呢？比如合并删除一些工作经历、之前薪水稍微报高一点点。对于一般公司、非高管类的岗位来说，HR 的背景调查没有那么严格，如果你确实因为一些不可抗力的因素跳槽有点频繁：比如创业公司还没呆几个月就垮掉了，比如刚来公司两个月发现有拖欠薪水的情况等等，那么适当删除一些较短的工作经历、适当合并一些经历也是“人之常情”，一般 HR 也不会太计较。但是投简历给大公司的时候，关键信息还是不要作假，比如学历、离职证明等。

3、技能说明：这里就列举你会什么框架、什么技术、什么工具。比如你可以说自己掌握了 selenium 自动化测试框架，会用 PYTHON 编写脚本，对于 ES 和 MySQL 会搭建会使用等等。

4、项目经历：这才是简历重中之重的部分。这里就列举你做过的一些项目，简要概括一下这个项目的背景、业务需求是什么样的，你在项目里做了哪些工作，用的哪些技术，有什么收获和提高，还有哪些不足待改进等等。后续准备面试的时候也要把做过的项目拿出来复习复习，关键技术点弄清楚，因为面试官肯定会照着简历问项目经验的。有些不是自己做的模块最好也请教一下同事，争取把做过的项目的细节都搞清楚，方便后续和面试官交流。

另外，再提一下投简历的技巧，笔者最开始在智联、BOSS 直聘、拉勾上投 BATH 的各种岗位，结果都石沉大海，而且反馈速度很慢。最后是一个在腾讯做 JAVA 后台开发的高中同学帮我内推的简历，应聘岗位是财付通业务结算测试，于是两天内就有反馈了。可见，能内推还是尽量内推吧。

第二关：电话面试　　结果：PASS

同学帮我内推简历后的第二天，我就接到了腾讯的电话。说实话大厂的员工素质还是高一些，电话面试的这个面试官说话特别客气有礼貌，先是让我自我介绍一下，我 balabala 简单说了一下个人信息和工作经历，然后问我为什么要投腾讯的岗位，我说我想去个大公司，再后面就是简单聊了一下项目情况，我当时就是介绍了一下项目业务需求，然后分了几轮进行，每一轮测试重点关注哪些方面（UI 界面、前台后台功能逻辑、算法逻辑、算法效果、稳定性测试等），用到了哪些技术（Linux\Sqlite\MySQL\Elasticsearch\Loadrunner\shell）等等。然后这个面试官就说二面三面都是要来深圳总部面试的，路费住宿费都不能报销，让我考虑一下再回复他。我当时就一口答应愿意过去面试，毕竟是自己非常心仪的公司。

第三关：FACE TO FACE Interview　　结果：PASS

腾讯的面试非常人性化，社招都安排在周末。我先订了去深圳的火车票（硬座），然后联系了在深圳安家的表姐，表示要在表姐住两个晚上。由于是绿皮车硬座，所以你懂得，一晚上都没睡好觉，第二天刚出火车站，表姐开车来接我去她家吃了个早餐又把我送到腾讯大厦。（在此非常感谢表姐的照顾）到了腾讯大厦会有实习生给办理临时工牌，

并指引你去相应的会议室等待面试。说实话，第一次去腾讯大厦面试，我还是挺紧张的，特别是一个人乘电梯去二十几楼的会议室的时候。会议室里大概坐了七八个人，都是等着面试的，各个岗位的都有。

言归正传，没等一会儿就轮到我了。我被实习生带到另外一间会议室，面试官是一位非常有气质的小姐姐。小姐姐开门见山让我在会议室的白板上画一下之前项目的系统结构图，我就把之前智慧停车项目的系统结构图画了出来。然后小姐姐仔细问了我每一个模块的作用以及模块之间如何通信，比如球机和嵌入式前端机如何通信？走什么协议？前端机和云平台之间如何通信？数据传输格式是什么？以及数据存在本地数据库的记录格式是什么？什么时候才抛向云平台？如何判断两条记录是同一辆车的入位出位记录？用的智能算法是什么？有没有参与过数据集的训练等等.....总之问得非常细，这些东西我在写简历的时候就复习过了，而且平时也非常主动地同开发同事沟通项目细节，所以总得来说答得还不错。最后，小姐姐又问了一些测试的基础知识点：比如压力测试和负载测试的区别是什么？一个 BUG 的完整周期包含哪些阶段？什么是灰盒测试？我也都回答准确。答完后小姐姐让我回到最开始的会议室等结果。没一会儿，实习生说我可以进入终面了。

第四关：终面-白板 CODING　　结果：FAIL

终面在另外一间会议室进行，这次的面试官换成了一位表情十分严肃的小哥哥。一落座，面试官问：“用过哪些数据库？SQL 熟么？”我答“用过 SQL SERVER 2000\MYSQL\SQLITE，sql 语句基本的都会”“那好请在白板上写一条 sql，实现查询某张表的前十条记录”我直接在白板上写了句“select * from students limit 10”。面试官又问“这是哪种数据库的写法？其他数据库的写法会吗？”答曰“这是 MySQL 的写法。其他数据库的写法不太了解。”接下来，面试官又问“你最拿手的是哪种编程语言”答“shell”（其实平时工作中最多用 shell 写非常简单的脚本，更多的是看懂别人的脚本然后改改参数之类，所以编程基本功并不扎实，所以当时很心虚地说 shell）“那你用 shell 写一个去重的代码，就是有个 txt 文本里有很多条记录，这些记录有些是重复的，要求把重复的记录剔除掉。”当时听完题目就蒙了，完全不会啊，于是故作沉思了一会儿，才和面试官说“真抱歉，这个题目我不会。”于是面试官就让我回最初那个等待区等结果了。

回到最初的等待区，没一会儿，实习生过来告诉我说，很遗憾，白板 coding 这关没有过，看我远道而来送一个 QQ 公仔给我做纪念。

后记：一点经验教训总结

先说教训吧。笔者觉得这次面试失败的主要原因还是那道去重的题目没有答上来。虽然笔者面试的是财付通结算业务测试的岗位，应该属于功能测试的范畴，但是类似 BATH 这种大公司是没有“纯点工”的岗位的，即便是功能测试也要有一点点代码能力，至少能看懂开发写的代码并且写一写简单脚本。而且白板 Coding 这种形式在大公司的面试里十分常见，毕竟光聊简历聊项目，看不出一个人的真实水平。由于平时工作里，我动手写脚本的机会并不多，准备面试时间又比较仓促，所以那道“去重”题目没有答上来。看来平时还是要多积累，提高代码能力，即使是工作中暂时不需要你编码，你也要利用业余时间学一学，掌握一种程序语言后，就多找机会练习实践或者上网刷刷程序员面试题，不懂地可以百度也可以和开发同事请教，正所谓艺多不压身。

再说经验吧。个人感觉前三关我答的还是挺好的，答完就知道自己有戏的感觉。首先，要精心准备简历，切中要害突出重点，能找熟人内推就内推，因为内推的效率相对来说高很多。其次，测试基础知识要扎实，一些基本的概念平时可以利用零散时间看看 51Testing 软件测试网公众号的文章，巩固一下。不要因为比较简单就掉以轻心，毕竟高楼大厦平地起，基础不扎实很难成为一名优秀的测试工程师。再次，不断地积累项目经验，切记不求甚解的行为。平时多和项目的同事们沟通，对于一些业务需求或者技术细节刨根问底地去了解，即使不是自己负责的模块，你多了解一些都是有好处的。面试之前把自己做过的项目梳理一遍，复习一遍，争取做到心中有数对答如流。最后，每参加一次笔试面试都做好总结工作，不断积累面试经验和面试技巧，这样以后无论去大公司还是小公司面试才能做到理智淡定不紧张。

学无止境，谨以此文纪念那次难忘的腾讯面试经历，愿与各位测试同行们共勉！金三银四跳槽季，祝大家早日拿到满意 offer!

机器学习与数据挖掘十大经典算法之 PageRank 算法

◆作者：咖啡猫

题记:
由于公司架构调整和业务方向的转变，笔者所在的项目组即将接手一个机器学习\数据挖掘的项目，为了后续更好地开展工作，也为了能提高自己的专业技能，笔者决定开始学习机器学习和数据挖掘方面的知识。

那么，问题就来了：到底应该从哪里开始学起呢？最开始我也买了一些机器学习相关的入门书籍，跟着听一些网络课程，但是我发现所有的课程都特别偏重理论，虽然机器学习、数据挖掘需要很强的理论基础才能做好，但是我个人更喜欢理论联系实际的学习方式，比如可以在了解某种基本原理的基础上，立刻用代码来实现它。无意间从同事口中得知机器学习与数据挖掘的十大经典算法，我决定就从十大经典算法开始学习。学习路线就是：逐个掌握每种经典算法的算法思想、数学模型及 Python 代码实现，争取各个击破并融汇贯通。

好了，废话不多说了，我们先看第一种经典算法：PageRank 算法。

一、PageRank 算法的简介

PageRank 算法即网页分级排名算法，它的提出者是谷歌的创始人之一拉里·佩奇（Larry Page），所以算法的名字就以 Page 命名。拉里·佩奇提出该算法时还是一名斯坦福大学的学生，（真是自古英雄出少年啊！）并且该算法曾在 2001 年 9 月获得美国国家专利。

PageRank 算法是 Google 算法的重要内容之一，可以说它就是 Google 算法的降龙十八掌和倚天屠龙剑啊！

二、PageRank 算法的核心思想

PageRank 根据网站的外部链接和内部链接的数量和质量，衡量网站的价值。PageRank

隐含的思想就是：每个到页面的链接都是对该页面的一次投票， 被链接的越多，就意味着被其他网站投票越多。一个网页所获得“投票”越多，说明这个网页越重要，它的被访问的概率越大，自然分级排名就越高，那么搜索结果它就越靠前。这就好比是一篇学术论文，论文被引用的次数越多，论文的影响因子越高，自然论文就越权威啦!

PageRank 的核心思想归纳起来就两条:

1.如果一个网页被很多其他网页链接到的话说明这个网页比较重要，也就是 PageRank 值会相对较高。

2.如果一个 PageRank 值很高的网页链接到一个其他的网页，那么被链接到的网页的 PageRank 值会相应地因此而提高。

三、PageRank 算法的数学模型

1、相关概念

在介绍 PageRank 的数学模型之前，我们先来了解一下相关基本概念:

- 出链：网页 A 中附加了网页 B 的链接，用户浏览 A 时可以通过点击该链接进入网页 B，此时我们称 A 出链 B。
- 入链：上面通过点击网页 A 中 B-Link 进入 B，表示由 A 入链 B。如果用户自己在浏览器输入栏输入网页 B 的 URL，然后进入 B，表示用户通过输入 URL 入链 B。
- 无出链：如果一个网页 A 中没有附加任何的 URL，则称 A 无出链。
- 只对自己出链：如果一个网页 A 中没有附加任何其他页面的 URL，只有附加自己的 URL，则称 A 只对自己出链。
- PR 值：就是指一个网站被访问的概率，PR 值越高，被访问的概率越高，自然排名就高。

2、简单数学模型（不带 a 的数学模型）

首先，我们对网络上的所有网页做一个抽象，每个网页代表一个节点，如果从网页 A 中附加了网页 B 的链接，则表示从节点 A 指点节点 B 的有向边。那么整个 WEB 就被抽象成一张有向图。现在我们假设世界上只有四个网页，它们之间关系如下图:

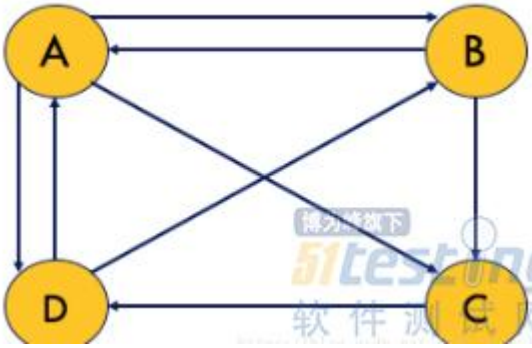


图 1

之前我们说过 PageRank 的思想就是，谁被引用的越多，谁的 PR 值越高。那么我们假设当用户停留在某个页面上时，他跳转到页面上任意一个链接的概率相同。

对任意一个网页我们用 I(p)描述其重要性，称之为网页排序值（就是 PR 值）。假定网页 Pj 有 Lj 个链接，其中一个链接指向网页 Pi,那么 Pj 将其重要性的 1/Lj 分给 Pi,即 Pi 的网页重要性就是所有指向这个网页的其他网页所贡献的重要性之和。公式表示如下:

$I(P_i) = \sum_{P_j \in B_i} \frac{I(P_j)}{l_j}$，式中，Bi 表示所有链接到 Pi 的网页集合。

为了方便数学分析，我们定义一个超链矩阵 M[Mij]:

$$M_{ij} = \begin{cases} 1/l_j & \text{if } P_j \in B_i \\ 0 & \text{otherwise} \end{cases}$$

其中第 i 行 j 列的值 Mij 表示用户从页面 j 转到页面 i 的概率。

按照这个定义，图 1 的超链矩阵为

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 0 & 1 & 0 \end{bmatrix}$$

设初始时每个页面的 rank 值为 1/N，这里就是 1/4。按 A－D 顺序得到向量 v:
v=[1/4,1/4,1/4,1/4]

此时如果做矩阵乘法，使 M*v 就得到一个新的 rank 阵 v’：M 第一行分别是 A、B、C 和 D 转移到页面 A 的概率，而 v 的第一列分别是 A、B、C 和 D 当前的 rank，因此用 M

的第一行乘以 v 的第一列，所得结果就是页面 A 最新 rank 的合理估计，故 M*v 的结果 v’就分别代表 A、B、C、D 新 rank 值。然后用 M 再乘以这个新的 rank 向量 v’，又会产生一个 rank 向量。迭代这个过程，可以证明 v 最终会收敛，即 v≈Mv，此时计算停止，最终得到的 v’就是 rankpage 的排序结果:

V’=M*V----------(1)

3、复杂数学模型（带 a 的数学模型）

但是我们也注意到，要想上述迭代结果最终收敛，必须满足一个条件：图是强连通的，即从任意网页可以到达其他任意网页。假设我们把上面图中 C 到 D 的链接丢掉，C 变成了一个终止点。再进行迭代，那么迭代的最终结果是 v’=[0,0,0,0]，显然算法失效了。除了终止点问题外，还有一个陷阱问题，即将图 1 中 D 到 C 的链接删除后，再加一条 C 指向 C 自身的链接。那么按上述迭代过程，最终 v’=[0,0,1,0]，此时算法也是失效的。

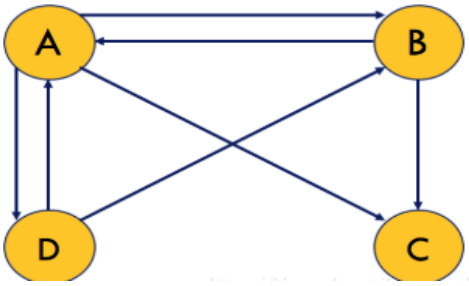


图 2 终止点问题

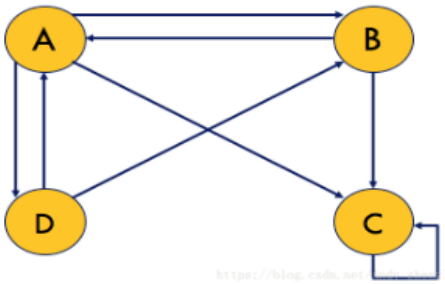


图 3 陷阱问题

为了解决终止点问题和陷阱问题，我们需要对算法进行改进：假设用户在选择下一个跳转的页面时，选择当前页及当前页上的链接的概率为 a，选择其他页面的概率为（1-a），选择其他页面中每个页面的概率都相同为 1/n，则计算 PR 值的公式演变为:

$$v' = \alpha Mv + (1-\alpha)(1/n) \text{-----(2)}$$

四、PageRank 算法的 Python 实现

下面我们以图 3 为例，分别用代码实现公式（1）和公式（2）的排序结果:

```
import numpy as np
M = [[0,1/2,0,1/2], [1/3,0,0,1/2], [1/3,1/2,1,0],[1/3,0,0,0]
U = [1/4,1/4,1/4,1/4]
```

```
U0 = np.array(U)
U_past_none_alpha = []
while True:
U = np.dot(M,U)
if str(U) == str(U_past_none_alpha):
Break
U_past_none_alpha = U
print('公式 1 的结果:',U)
U_past_has_alpha = []
while True:
U = 0.8*(np.dot(M,U))+0.2*U0
if str(U) == str(U_past_has_alpha):
Break
U_past_has_alpha = U
print('公式 2 的结果:',U)
```

输出结果如下:

C:\Users\1009\PycharmProjects\20190329\venv\Include\Include\Scripts\Python.exe C:/Users/1009/PycharmProjects/20190329/pagerank.py

公式 1 的结果: [0. 0. 1. 0.]

公式 2 的结果: [0.13172043 0.11917563 0.6639785 0.08512545]

Process finished with exit code 0

显然，公式（2）的结果更加科学准确！

五、后记:

PageRank 算法就介绍到这里了，笔者的感觉就是按公式编码实现其实并不难，难的在于公式背后的数学逻辑思维。通过和做算法开发的同事交流，笔者才知道原来算法最最重要的就是思维，没有思维的算法就如同没有灵魂的躯体一样，完全不能适应复杂的现实场景的需求。谨以此文为开端，开始我的算法之旅，希望与广大测试同行一起进步！

测试人员不得不小心那些职场套路

◆作者：周培培

摘要:
职场最长的路就是所谓的套路了。尽管互联网公司相对来说氛围要好得多，但所谓“有人的地方就有套路，是套路就会有人背锅”。这对于大多数的普通测试人员来说，要与不同角色人员打交道，并且躲过各种各样的套路，恐怕就是必修课了。下面就自己一些经验和感悟，聊聊自己的一些体会。

对“另类”成员不可一视同仁

对于大部分测试人员而言，要想让一个项目顺利上线，需要和不同的角色人员深入打交道，而且是从头到尾。这其中除了固定的流程外，感受比较深的一点就是：不同的研发人员、产品人员有不同的合作习惯，各自的强项和弱项，甚至一些行为比较“另类”的人员，需要有针对性考虑与他们的合作模式了。

举一个自己亲身经历的一个例子：众所周知，互联网公司加班属于家常便饭了，六点下班大多属于偶尔一次的奢侈行为了。当然自己所在团队也不例外，加上当时在赶一个很紧急的项目，团队成员几乎都 10 点往后才陆陆续续开始离开。但其中有个同学却一直保持 6、7 点下班的“惯例”。没办法了，既然无法控制别人的行为，只能根据他的惯例做不同的区别对待了，于是乎将与其所有的 bug 修复、验证等流程做了对应修整。

防止替“小人”背锅

不主动招惹，但如果涉及利益相关的事情，无论对谁，请谨记以下几点：

（1）各自职责划分清楚。万一到时候真发生矛盾，能够确认到底是谁的工作出现了问题

（2）事情的关键节点各方确认。会议纪要、讨论纪要，一定由邮件、聊天记录等形式由各方确认。如会议结束后的会议内容确认，下期工作安排，项目责任分工等

（3）一切证据说话。为了不做无谓的扯皮，背黑锅，一定留有邮件、聊天记录、录

音等证据来自证清白

PS：休假时，提前设置好自动回复/ 备注找谁：让对方知道你现在不能即时回复邮件，同时告知相应的对接人。

举一个身边 A 同学经历的一个事情：某次产品到即将上线已经凌晨 2 点了，由于项目上线前，流程要求必须由产品人员发出邮件后，才能操作上线。但当时产品人员已经回家了，A 同学只能打电话，发现电话打不通，只好给其 leader 打电话，然后“费了半天口舌”来发邮件，最后产品上线验证完成，基本凌晨 3，4 点了。第二天，不知产品人员内部是如何沟通的，就在群里做各种扯皮、推脱，大致就是上线当天没有提前周知自己等等之类的，还把锅甩给了测试人员为什么不提前发周知，态度强硬。幸好 A 同学有这方面的经验，直接拿出了前一天团队成员在一起开会内容截图，包括上线内容、上线时间等，才免于让自己背锅。

整件事情我们抛开流程等是否合理，但就事情本身来说，假如 A 同学没有强有力的证据，那么这个锅恐怕他是背定了吧。

被暗示让你主动离职该怎么办?

这种情况由于主观、客观原因的存在，都可能会发生的情况了。当然了，大多数的公司还是比较正规了，一般不会“赖”辞退员工应该赔付的赔偿款。但“林子大了，自然什么鸟都有”，万一这事落到你头上，此时，不要认为自己好像被优待了一样，没有给你直接辞退，而是提前告诉了你；其实此种情况，往往是没有直接让你走人的理由，假设你工作不合格，或犯有重大失误，恐怕早给你“下旨”了。

再者，对你自己而言，不要硬要留下来，也不要傻傻地选择主动离职，而是要考虑最重要的问题:

要争取赔偿。对公司而言，让你主动离职，自然是不想付你赔偿费了。而对于你，一定要留自己被暗示离职的证据，申请劳动仲裁。

这时需要注意的是:

（1）与不同人约谈，特别是 HR 时，有意识引导对方说出意图，自己说出工作无失误等类似的对话，并留下录音等作为证据。

（2）留下自己工作无失误的证据，比如绩效、薪资、福利等方面的截图

（3）不要签署任何自动离职、不明所以等方面的协议，以免以后授人以柄

以下事情万万不可做

说与不说，首先要看是否可能会对自己产生不利的影响，如果有可能，请把可能当成一定不可说的事情例如:

（1）与同事大谈自己未来的工作计划。除个别铁饭碗外，一个人不太可能在一个公司永远呆着，自己要是有跳槽的想法，或者正在纠结要不要离职的时候，不要对同事说。

（3）与同事抱怨、说丧气的话。因为说了后，对现状不会有任何改变，万一隔墙有耳，传到 leader 那里，你想会有你的好果子吃吗？最好的结果恐怕就是: leader 说这是 xxx 的问题，可以用数据说话等等，一是这是大家众所周知的方法，二是这类似的不痛不痒的话，并不会让你面临的问题因此减少

（3）答应可能需要背锅的忙。帮忙的结果一旦不如意，或者触及到他的利益，这时你不但得不到任何感激，反而会无故背锅，好像你是事情进展不顺的主因。

决定接受 offer 前务必确认的几点

决定是否接受 offer，除了发展前景、职位内容、工作地点等因素外，薪资待遇是比较重要的考虑因素了。不乏有某些模棱两可的一些信息，如果你自己不注意，入职之后可能会有不同程度的心理落差了。所以，本着对自己、公司都负责的角度出发，正式接受 offer 前，请务必确认以下信息:

（1）试用期期间薪资如何给，正式薪资的 100%，还是 80%

（2）年终奖的奖金系数如何评定的。奖金系数除了直接和自己的绩效挂钩外，如果还和部门的绩效有关系(互联网公司往往如此)，那么就要提前了解该部门的盈利情况如何了。

（3）还有一个和薪资涨幅有间接关系的一点，就是对应团队 leader 的情况了。当然了，之前的面试官大部分情况都会有团队 leader，可以自己评判一下了。当然如果可以，尽可能找内部人员打听一下吧。某个角度看，跳槽好坏很重要的一点就是能否跟对一个 leader，因为遇到一个和自己三观不和的 leader，可能一段时间内你会相当憋屈。

写在最后

无论此刻的你是已经决定好好大干一场，还是决定跳槽寻找其他的可能性，都需要权

衡好自己需要放弃什么、获得什么。但无论是哪个决定，无论这些个套路你是否遇到过，请务必提高警惕。

软件测试证书知多少

◆作者：王立东

一、概览

目前各行各业都流行考证，作为软件测试工程师的你，是否考虑过有哪些证书是适合测试人员认证的呢？本文将软件测试的证书分为 5 类，分别为国家类（指国家官方认可的考试和证书），国际类（在国际上认可度比较高的认证），培训类（各类培训机构的毕业证书），项目管理类（PMP），专业类（测试技术服务的行业相关的认证）。

证书无法完全体现能力，本文也不是规劝进行相关认证考试的，旨在给大家介绍与测试相关的认证和机构，同时说明需要理性看待证书。

二、国家类（软考，计算机等级）

本节介绍的是被国家认可的软件测试相关的证书。整个软件行业发展非常迅速，在国家层面需要对各行各业进行能力的认定和考核。软件测试作为软件的一个组成部分，也在国家相关部门的考核范围，这里重点说明两个：软考和计算机等级考试。

从考纲的变更频繁，方向的不断变化中可以看出，国家相关部门是非常想把软件，以及软件测试的认证做的非常到位的。但从实际的考题设置来看，与实际的科技生产现状脱钩比较严重，做不到指导实际用人招聘的目标，所以也不会被企业非常认可。

2.1、全国计算机等级考试

全国计算机等级考试（National Computer Rank Examination，简称 NCRE），是经原国家教育委员会（现教育部）批准，由教育部考试中心主办，面向社会，用于考查应试人员计算机应用知识与技能的全国性计算机水平考试体系。

这项考试在校学生考的比较多，对于应届生有帮助，对于有工作经验的含金量不高。其中关于软件测试的证书是四级软件测试工程师和三级软件测试技术。在 2008 年 4 月引入考试，但在 2018 年 3 月中去掉。所以针对软件测试的证书，在计算机等级考试中已经没有了。

2.2、计算机技术与软件专业技术资格（水平）考试

计算机技术与软件专业技术资格（水平）考试，就是俗称的“软考”，是原中国计算机软件专业技术资格和水平考试的完善与发展。计算机软件资格考试是由国家人力资源和社会保障部、工业和信息化部领导下的国家级考试，其目的是科学、公正地对全国计算机与软件专业技术人员进行职业资格、专业技术资格认定和专业技术水平测试。

软考设置了纵轴 3 个级别层次（初、中、高），横轴 5 个专业领域（计算机软件，计算机网络，计算机应用技术，信息系统，信息服务），共计 27 个专业资格。其中与软件测试证书是中级软件评测师，属于计算机软件领域，具体要求可以参考官网（http://www.ruankao.org.cn）。

2.3、二者区别

计算机等级考试和软考有如下区别：

（1）组织部门不同。计算机等级考试是教育部批准，由教育部考试中心主办的。软考是由人力资源和社会保障部、工业和信息化部领导下的国家级考试。

（2）目的不同。计算机等级考试的目的是，用于考查应试人员计算机应用知识与技能的全国性计算机水平考试体系。软考是科学、公正地对全国计算机与软件专业技术人员进行职业资格、专业技术资格认定和专业技术水平测试。

（3）时间不同。计算机等级是 3、9、12 月。软考为上半年 5 月底和下半年 11 月中。

其实只要区别是前两个了。教育部主办的等级考试是评价毕业院校学生需要具有一定的等级，比如一些大学的计算机学院将过三级或者四级作为毕业的条件。人社部和工信部是评价工作人员职业资格的。

二者相比，对于有工作经验的人来说，软考的价值更高些。软考还有一个好处是可以作为职称资格证书（是资格，不是职称），同时在职称评审中代替职称计算机考试。

三、国际类（ISTQB，CSTE）

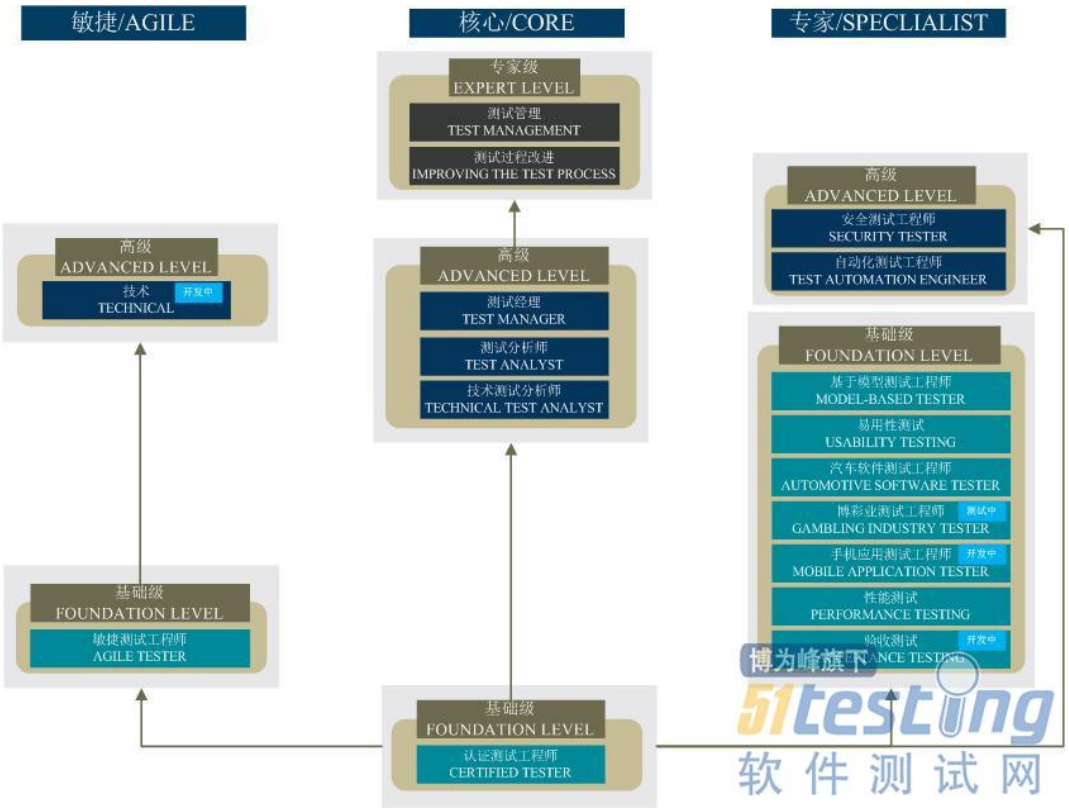
在国际上，也有软件测试相关的认证，相比较国家的认证来看，含金量和认可度相对会高一些。本节介绍两个比较知名的：ISTQB 和 CSTE。

3.1、ISTQB

ISTQB（International Software Testing Qualifications Board），全称为国际软件测试认证委员会，是一个注册于比利时的非赢利性组织，是国际唯一权威的软件测试资质认证机构。ISTQB 目前拥有 58 个分会，覆盖包括美国、德国、英国、法国、印度等在内的 120 多个国家和地区。

2006 年，ISTQB 在大中华区（包括港澳台地区）成立了唯一分会，CSTQB（Chinese Software Testing Qualifications Board）。全权代表 ISTQB 在授权区域内推广 ISTQB 软件测试工程师认证体系，认证、管理培训机构和考试机构，接受 ISTQB 的全面的业务指导和授权。

ISTQB 认证分为三个等级基础级 Foundation Level （CTFL），高级 Advanced Level（CTAL）和专家级 Expert Level （CTEL），三个方向敏捷（AGILE），核心（CORE）和专家（SPECLIALIST）。其中基础级包括敏捷测试、基于模型测试以及面对各个专业的测试。高级包括测试经理、测试分析、测试技术分析。专家级包括测试过程改进、测试管理。



从其框架可以看出，贯穿了测试体系的全部，参照这样的体系可以构建测试的职业生

涯发展方向。ISTQB 含金量很高，在金融、军工、汽车等安全要求较高的行业、BAT 以及外企大公司等普遍有硬性要求，近两年在国内的认可度和需求量明显上升。

3.2、CSTE

CSTE 全称 Certified Software Tester，是 QAI(Quality Assurance Institute)旗下的重要认证。该全球范围的测试认证已经在美国、印度等颇为普及，成为很多公司对于测试从业人员的要求之一。但是在中国，考这个证书的人只占了少部分。

QAI 全称是质量保证协会，是一个代表质量保证人员的专业协会。在 1980 年把协会和认证工作分开，就成立类 ISCB（International Software Certifications Board’s），国际软件认证委员会。目前取得认证的人士分布在六大洲 43 个国家，大约有 52000 人获得。

ISCB 的认证涵盖三个领域：软件质量保证，软件测试和软件业务分析。CSTE 是属于软件测试认证的一项。ISCB 的认证体系如下。

CSTE 的知名度没有 ISTQB 高，在国内的普及程度也没有 ISTQB 好。含金量和认可度也是仁者见仁。

四、培训类

国内有非常多的测试培训机构，这里指的是培训机构的毕业证书。当然，几乎所有的培训机构都会组织进行 ISTQB 或者 CSTE 的考试，也有专门针对软考或计算机等级的培训机构，本节的讨论就是机构的毕业证书。

由于培训机构非常多，质量和效果和各人的关系非常大，国内对机构的毕业证书都是作为一个加分项，作为混迹于 51Testing 论坛的老会员，这里对 51Testing 做简要说明。

4.1、51Testing

51Testing 是博为峰下属的专注于软件测试的培训品牌，于 2004 年成立，公司总部位于上海，并在北京、深圳等 15 地均设有分支服务机构。

51Testing 属于成立比较早的测试培训机构，业务范围包含就业培训、企业内训、周末精品班、测试开发精英班、软件测试认证等服务，被誉为"软件测试人才的摇篮"。

实际上，51Testing 除了培训之外，还有门户站，论坛，测试圈等线上测试组织，同时也有各地的测试沙龙。是非常全面的测试平台。

五、项目管理类

测试人员随着工作的深入，会更多的加入到质量控制的工作中，就会慢慢发现项目管理实际也是软件测试人员可以发挥所长的一个方向。项目管理类的认证很多，其中比较著名的是 PMP。

5.1、PMP

PMP 指的是项目管理专业人士资格认证。它是由美国项目管理协会（Project Management Institute(PMI)发起的，严格评估项目管理人员知识技能是否具有高品质的资格认证考试。PMI 是世界领先的非盈利会员协会的项目管理专业机构，在全球 185 个国家有 70 多万会员和证书持有人。

PMP 其实是 PMI 发起的一项认证，PMI 提供 8 种认证:

1. 助理项目管理专业人士（Certified Associate in Project Management, CAPM）认证面向广泛的项目管理从业人士群体。

2. 项目管理专业人士（Project Management Professional, PMP）认证始于 1984 年的这项认证向雇主、客户和同事表明项目经理具备成功完成项目的项目管理知识，经验和技能。在理想状态下，每个项目都应有一位 PMP 持证人士作为核心来领导团队、指导项目

任务，并在预算、时间和范围三重约束下管理项目。 这就是俗称的 PMP 了。

3. 项目集管理专业人士（Program Management Professional, PgMP） 认证面向具备知识、经验和权威来制定和执行战略性决策的专业人士。

4. 项目组合管理专业人士（Portfolio Management Professional, PfMP）认证面向具有先进经验和技巧的项目组合管理者。

5. PMI 敏捷管理专业人士（PMI Agile Certified Practitioner, PMI-ACP）认证是唯一要求培训、经验与考试三者相结合的敏捷认证。

6. PMI 商业分析专业人士（PMI Professional in Business Analysis, PMI-PBA）认证，在项目和项目集上采用业务分析能够让组织得以实现他们所需要的改变以及达成战略目标。

7. PMI 风险管理专业人士（PMI Risk Management Professional, PMI-RMP）认证面向在评估和识别项目风险的专业领域具备知识和技能，并能够制定计划来缓解威胁和/或利用机会的专业人士。

8. PMI 进度管理专业人士（PMI Scheduling Professional, PMI-SP）认证面向在制定和维护项目进度的专业领域拥有专长的专业人士。

PMP 实际的含金量很高，而 PgmP 和 PfmP 是难度更高的认证。对于管理岗位来说，进行 PMP 的学习和认证还是非常有必要的。近年来，随着考取 PMP 的人越来越多，让大家有些看轻了这项认证。实际上，中国人善于考试是不争的事实，而 PMP 是重在实战的一项能力，如果只是为了考取证书，而无法在项目管理中落地，是个人或者是平台的问题，于 PMP 本身是无关的。

六、专业类

如果说前面都是从单纯的软件测试技术来定位认证的化，本节就是说明测试技术服务行业的相关认证。业务知识是测试人员无论如何也绕不开的话题，各个行业也有自己特定的认证，如果可以通过本行业的权威认证，无疑对自己的价值是可以添砖加瓦的。由于行业千差万别，这里说明的只是一部分。

（1）软件开发。软件开发没有专门的证书，如果要认证，就只有国家类的软考了。但针对特定语言，比如 Java，Sun 就推出了 sun certificated java programmer (SCJP)，Sun Certified Java Developer(SCJD)等认证。由于开发技术多种多样，无法设定统一的认证，同时通过证书证明软件开发的能力并不被认可，项目才是硬道理。

（2）数据库。对于经常于数据库打交道的测试人员，如果有一个数据库相关的认证，会提高自身价值，同时工作中也会更有章法。数据的认证只要是每个数据库厂家设定的，由于传统的数据库就那么几家，所以含金量和认可度都是很高的。包括如下:

ORACLE 数据库认证：OCA（Oracle Certified Associate）、OCP（Oracle Certified Professionals、OCM（Oracle Certified Master）。

MySQL 数据库认证：Certified MySQL 5.0 Developer (CMDEV)、Certified MySQL 5.0 DBA (CMDBA)、Certified MySQL 5.1 Cluster DBA (CMCDBA)。

MS SQL 认证: 微软 MCSE 包括 Data Platform 数据平台和 MCSE: Business Intelligence 商业智能两个方向。

（3）网络。网络、通信相关专业的认证，知名的是 CCNA，CCNP 与 CCIE。

CCNA: Cisco Certified Network Associate，是初级认证，标志着具备安装、配置、运行中型路由和交换网络，并进行故障排除的能力。

CCNP: Cisco Certified Network Professional，是中级认证，表示通过认证的人员具有丰富的网络知识。

CCIE: Cisco Certified Internetwork Expert，是高级认证，是美国 Cisco 公司于 1993 年开始推出的专家级认证考试。被全球公认为 IT 业最权威的认证，是全球 Internetworking 领域中最顶级的认证证书。

这些都是思科的认证，其实民族品牌华为也有相应的配套认证: HCNA、HCNP、HCIE。

（4）操作系统。目前国际上广泛承认的 Linux 认证有 Linux Professional Institute（简称为 LPI）、Sair Linux 和 GNU、Linux+和 Red Hat Certified Engineer。Windows 服务器也有微软的相关认证。

七、证书总结

上述认证之间也不是完全孤立的，比如各大培训机构在 ISTQB、软考等都有方向性培训，同时软考、计算机等级考试也都有数据库、编程语言的认证。

实际工作中，除了专业类的个别证书（如数据库和 Linux）外，企业在招聘时对证书的看重不是非常大，只能作为加分项，无法决定最终的面试结果。

但并不代表证书存在没有必要，或者对证书无感。一方面，在同等条件下，证书还是能说明一定的问题，比如最少能说明此人是积极的，对自己负责的。另一方面，证书的结果自然非常重要，但其过程更为重要。通过系统的学习，能对测试领域知识的方方面面有系统的了解，能够构建自己的知识体系。同时，随着学习的系统化，对自身的长处、缺点，未来的职业发展都有很好的指导。

埋头干活，更要抬头看路，而证书的存在，可以作为我们的指路石。

由测试的历史追溯测试发展

◆作者：侯峥

软件测试的发展是由项目中的需要推动的。不同时期的项目对于测试的需求渴望强度和要求是不同的。简单的以人类的进化发展史来类比测试的发展史，希望可以从中窥见一些规律，来提高测试人员的素质，加强自身的竞争力。更加希望帮助其他行业的人员了解软件测试行业，端正大家对于软件测试人员的态度。

大约 450 万年前，人和猿开始分化，产生腊玛古猿，以后在由腊玛古猿演化成 200 万年前的南方古猿，进一步再发展为现代人类。关于人类的发展过程，一般将其划分为四个阶段:

1、早期猿人阶段。大约生存在 300 万年到 150 万年前，已具备人类基本特点，能直立行走，制造简单的砾石工具。——无测试

2、晚期猿人阶段。大约距今 200 万年到 30 万年前，身体象人，脑量较大，可以制造较进步的旧石器，并开始使用火，如我国北京周口店的北京猿人。——无专职测试/开发自测

3、早期智人(古人)阶段。距今 10-20 万年到 5 万年前，逐渐脱离猿的特征，而和现代人很接近，如德国的尼安德特人。 ——专职测试点点

4、晚期智人(新人)阶段。大约 4-5 万年前，这时的人类的进化出现了明显的加速，在形态上已非常象现代人，在文化上，已有雕刻与绘画的艺术，并出现装饰物。如 1933 年发现的周口店龙骨山山顶洞人。此时原始宗教已经产生，已进入母系社会。在晚期智人阶段，现代人开始分化和形成，并分布到世界各地。——自动化测试/性能测试（测试开发）

5、AI 机器智能阶段。现在阶段的发展，未来的情况靠想象。——未来测试发展猜想

依据人类的发展，软件测试发展也可以分为 5 个阶段：无测试阶段；无专职测试/开发自测阶段；专职测试点点阶段；自动化测试/性能测试（测试开发）阶段；未来测试发展

猜想。

一、无测试阶段

早期软件项目中只有开发人员，开发出的软件严格意义上讲只能称为单一程序，而这个程序又是为一个特定的目的而编制的。早期当通用硬件成为平常事情的时候，软件的通用性却是很有限的。由于要求不高，程序相对简单，仅由开发人员开发出来就可以使用。而且当时人们对于软件的容错性非常高，这个时期开发出来的程序就能满足使用，并没有测试的环节，也没有测试的概念。

二、无专职测试/开发自测阶段

随着软件的数量急剧膨胀，软件需求日趋复杂，维护的难度越来越大，开发成本令人吃惊地高，而失败的软件开发项目却屡见不鲜。“软件危机”就这样开始了！大家开始对软件测试有了基本的概念，但是还没有形成体系，更不要说具体的技术产生探究。对于测试的认识还仅限于使用前试着运行下程序，通常是开发自己测试或者是客服、财务或者其他使用软件的业务人员试用下软件。这个时候人们对于软件的容错度还是比较高的。非专业测试人员/开发自测也能满足这一段的要求。

这种测试的方式到现在为止仍然在使用，项目正式进入测试阶段之前由开发自测，项目上线之后由客服或者其他软件的使用人员反馈问题。不同的是，不同公司对于质量的要求这种测试方式的环节占的比重不同。对于质量要求不高和软件仅供内部使用的项目，非专业测试人员/开发自测便是测试的全部环节。

三、专职测试点点

“软件危机”使得人们开始对软件及其特性进行更深一步的研究，人们改变了早期对软件的不正确看法。早期那些被认为是优秀的程序常常很难被别人看懂，通篇充满了程序技巧。现在人们普遍认为优秀的程序除了功能正确，性能优良之外，还应该容易看懂、容易使用、容易修改和扩充。概括来说，软件危机包含两方面问题：一、如何开发软件，以满足不断增长，日趋复杂的需求；二、如何维护数量不断膨胀的软件产品。

软件的要求提高最直接的解决方式就是增加人员，加强管理。那么开发的大部分精力将投入到软件开发和代码管理中去，这个时候一部分专职的测试人员便出现了，他们大多由软件使用的业务人员组成，并没有计算机相关知识背景。这是的测试工作内容非常的简单和非专职人员测试相似，就是按照自己的思路点点。这种工作方式就是现在初级

测试工程师的工作主要内容。

尽管这种工作方式看起来非常的简单，但是需要测试人员具备很高的逻辑能力以及表述能力。不幸是的，部分人对于测试工程师的印象仅仅停留在“点点点”的阶段。由于外行或者同行的偏见，测试人员还要有强大的心理承受能力。

四、自动化测试/性能测试（测试开发）

互联网行业发展的越来越大，越来越多的企业加入软件创造的大军，软件功能覆盖的功能也越来越贴近人们的生活。软件企业之间的竞争更是愈加激烈，人们不再满足于功能的实现更加在乎功能的新颖和使用体验。这就使得企业需要加快软件的迭代频率来保证同类软件的市场占有率。

软件更新频率增加就意味着软件的制作周期缩短，而缩短工期就有可能难以保证软件的质量。实际上做 100 次好事的效果抵不上 1 次失误带来的后果。在这种情况就对测试人员有了更高的要求，为了提高测试效率，避免测试人员的重复工作，自动化测试就出现了，技术的发展就是为了迎合要求产生的。

自动化测试，顾名思义，自动完成测试工作。通过一些自动化测试工具或自己造轮子实现模拟之前人工点点/写写的工作并验证其结果完成整个测试过程，这样的测试过程，便是自动化测试。自动化测试的前提是功能手工测试通过，如果没有手工测试的基础，是没法进行自动化测试，投入和产出比也会很低。

对于一些软件，它的市场占有率高，用户基数大。这些软件不仅仅要满足功能上的要求，在性能上，比如并发数、负载数都有要求。在确定量级后，需要进行性能测试来发现性能上的问题，性能测试的前提也需要功能手工测试通过。

自动化测试和性能测试需要测试人员运用计算机语言编写脚本，本质上和开发是一样的，这部分的测试人员也被称为测试开发。这在很大程度上提升了测试人员的地位。但是只有测试人员知道，手工测试看似简单实则在测试过程中扮演了重要的角色。一切不以手工测试为基础的测试都是耍流氓。

五、未来测试发展猜想

当下最热的一个词莫过于 AI 智能了，机器翻译，智能控制，专家系统，机器人学，语言和图像理解，遗传编程机器人工厂，自动程序设计，航天应用，庞大的信息处理，储存与管理，执行化合生命体无法执行的或复杂或规模庞大的任务等等，这些领域都有

AI 智能都有涉及。AI 智能是对人的意识、思维的信息过程的模拟。人工智能不是人的智能，但能像人那样思考、也可能超过人的智能。那么我们是不是可以大胆的想象，未来的测试技术也将趋近于智能，比如说，根据需求简单的生成测试用例，自动化测试更加智能化。

可能现在实现还有些不切实际，但是梦想还是要有的，万一实现了呢！AI 都深入普通大众的生活了，测试的技术也需要紧跟脚步发展。提高自身的技能才能增加测试在项目中的权重，扭转大家对于测试只有“点点点”的错误认知。

APP 崩溃类问题总结

◆作者：桃子

关于这篇崩溃类问题已经有意向总结很久了，无疑这篇文章比较难写。原因之一是需要有大量的 App 测试实践经验，实践是检验整理的唯一标准么；再有准备过程也比较长，需要平时多记录多思考多归纳，如果您恰巧读到这篇文章，觉得对您有帮助，请点个小心心吧。

注：本篇文章未考虑功能正常流程下的操作。

一、什么样的场景下容易出现崩溃类问题

最近脑子里一直在想什么样的问题容易导致崩溃——那就是具有异常思维

举个例子正好前两天看了一部美剧叫《菜鸟老警》，里面有个场景是主人公 John Nolan 因为没有用警车拦住逃犯的车而导致逃犯出逃，这个场景与测试有什么关联呢

首先正常人在马路上行驶都会躲避避免与别人发生碰撞，这就好比测试功能的主流程，不会去乱点乱输入；但是警察就不一样了，他会为了抓住逃犯不顾一切，想尽一切办法，这就好比测试过程中的异常思维，要思考怎么样操作使功能不好用

宝典一、异常操作

各种异常操作都有可能导致程序崩溃，虽然客户基本上都是按流程使用 APP，但对于测试者来说，发现潜在的崩溃问题，保证产品质量是对我们工作的最好总结。包括下面总结的具体功能崩溃问题大多数都是异常操作引起的，比如图片上传过程中添加大容量图片、长图、残缺图片等等

宝典二、某一功能前后台来回切换，很容易导致崩溃

比如视频类视频播放过程中切换到后台再切换回来

宝典三、边界值类崩溃

所谓边界值类问题就是测试功能所能承受的最大值，举个例子，比如图片最大支持

5M，你上传》=5M 的图片，文本框最大支持 300 字，你就输入大于等于 300 字的内容

宝典四、多次点击某一个特定功能

这个比较常见，多次点击页面返回按钮，多次进行搜索，多次来回点击切换按钮，多次点击分享按钮等都会导致程序崩溃

宝典五、弱网条件下功能操作

宝典六、页面未加载完全情况下快速操作

二、具体功能的崩溃问题

2.1、视频类/语音类

1) 视频全屏播放中，多次点击全屏按钮和左上角返回按钮，系统崩溃

2) 视频全屏播放中，同时点击全屏和返回上页按钮后系统崩溃

3) 视频播放横竖屏切换，系统崩溃

4) 视频播放过程中，反复调整倍速崩溃

5) 视频播放过程中切换到后台再切回前台，播放不成功

6) 视频播放中多次切换章节 APP 闪退

7) 视频播放过程中被其他软件暂停后重新切换到播放页面点击继续 无法续播

8) 下载中的视频点击删除后崩溃

2.2、相机/扫一扫/头像

1) 扫一扫界面进入相册中选择 10M 以上图片出现闪退

2) 无相册权限修改头像可访问相册

3) 头像上传长图片崩溃

4) 头像上传残缺图片崩溃

5) 点击拍照按钮后切换到后台，再切换到前台，点击选择图片出现闪退

2.3、文本框/搜索：

1) 输入框中粘贴内容并全选复制程序闪退

2) 搜索界面输入文字全选添加闪退

3) 搜索成功后，再次搜索 APP crash

4) 多次输入特殊字符进行搜索，app 闪退

5) 搜索结果界面点击取消程序闪退

2.4、页面操作

1) 进入页面后马上退出再点击其他栏目

2) 连续返回到上一级页面

3) 进入界面一直加载，kill app 后点击 app ，页面打开后出现闪退

4) 界面内容较多时，上滑刷新后下滑，界面卡主，点击任意记录，出现闪退

2.5、分享功能:

1) 分享微信成功返回应用，再次点击分享按钮应用闪退

2) 多次操作留言分享按钮

3) 分享内容成功后，如在 qq 打开链接提示故事不存在

4) 微博授权登录，反复几次崩溃

最后，这些场景及操作需要临时变通才会有更好的效果，弱网情况下多次点击某一功能，个人认为比较好用的是多次点击某一功能，程序经常前后台切换等。

打破测试惯例

◆作者：周培培

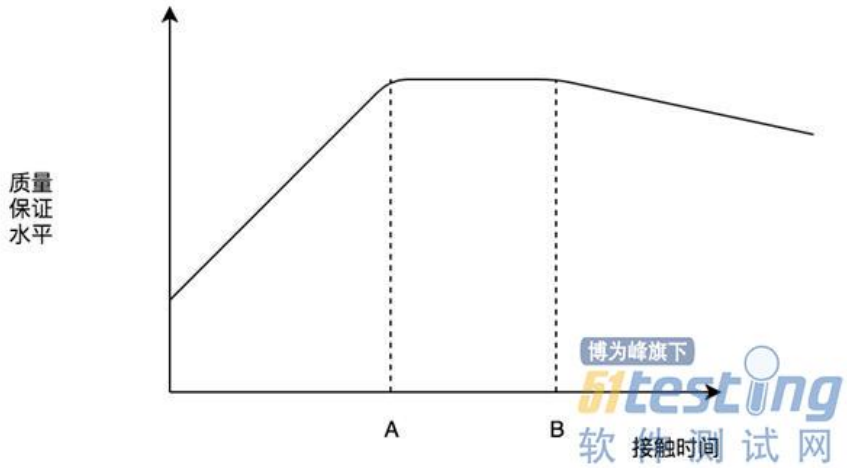
摘要:
日常测试中，作为测试人员，有了对产品、质量等熟悉到一定程度后，会给质量保证带来大大的好处。逐渐整个测试设计、测试执行过程会轻车熟路，并伴随有新鲜感渐失,思维不那么灵敏的阶段。到底是什么原因让我们的测试水平不能始终保持在最高峰呢？有没有方法来克服呢？下面就自己一些经验和感悟，聊聊自己的一些体会。

什么是测试惯例

按《辞海》解释，惯例指法律上没有明文规定，但过去曾经施行，可以仿照办理的做法。例如，国际贸易惯例、某法律惯例等。

《软件测试经验与教训》一书中有类似的描述：测试员在理解产品/功能后，在头脑中形成映射，随着对产品的了解，逐渐从各个方面提高对产品的反应能力和敏感性，并且头脑不再那么努力工作。

当然了，作为测试人员，有了对产品、质量等熟悉到一定程度后，会给质量保证带来大大的好处；但另一个方面，当对产品、质量接触的时间越来越长后，必然会新鲜感下降、由于过于熟悉而不愿做进一步探索、思考了。当测试人员在接触一段时间产品，但其间不做任何自我提升时，可以使用下面的图来大致描述测试人员对质量保证水平:



1）A 点之前，由于对产品越来越了解，整体质量保证水平是在显著上升的；

2）A 点~B 点之间，由于已经对产品充分了解了，加上尚未对产品产生倦怠，新鲜感/冲劲还在，加上测试策略越来越完善，质量保证水平会达到巅峰，同时也会逐渐形成“测试惯例”；

3）在 B 点之后，由于测试人员已经逐渐失去了新鲜感、下意识按照之前的步调行事、没有主动自我测试能力提升等，整体质量保证水平会稍微有所下降。这里之所以是“稍微下降”，原因在于以往的测试经验还在。

不同的业务，不同经验的测试人员，A，B 两个时间点的出现阶段会有所不同。作为测试人员来讲，当然希望永远保持在 A 点~B 点之间了，但或许这仍然是不够的。下面就自己的一些理解，谈谈这方面的体会和心得。

测试惯例带来的好处

对产品的“前世今生”十分熟悉。随着测试人员拥有越来越多的产品经验，在推动产品优化、甚至引导产品方向方面都会有所建树

“手到擒来”的测试经验。当测试策略已经制定完毕，测试深度、测试广度等等已经几乎 100%覆盖，自动化体系已经搭建完成后，任何产品需求、技术需求已经被现有的测试策略 cover 住了，那么这时候只需要根据测试方案“依葫芦画瓢”就够了

对技术实现十分了解。由于接触了各个服务的实现，因而无论是对于影响点、测试点的评估，还是服务间的系统架构，乃至各个服务的优势、劣势、可能的坑，都可以侃侃而谈了。

效率的保证。由于产品业务、技术实现、测试策略不用“现学现卖”了，加上十分了解团队成员的特点、合作模式，那么对于各个环节的进度，推动都可以不费吹灰之力了。

这些好处是不会随着测试惯例的到来而消失的，因而这也是所有测试人员喜闻乐见的结果。不知你是否由于考虑到上述诸多好处，而选择继续留在当下的岗位呢，这正是测试惯例对你的吸引力了。

测试惯例带来的坏处

下意识依赖惯性测试产品，而用户并没有这样的惯性。测试过 toC 产品的同学想必亲身经历、或听到过类似的故事：测试人员测试 OK，各方确认没问题上线了。不久之后，

产品人员又拉了一次需求——要改善一下产品的易用性，原因是用户 xxx 不太会用、或某某功能的入口过于隐藏了。其实究其根本原因在于，团队中的测试人员，甚至是产品人员、研发人员、设计人员，都对产品十分熟悉了，可以下意识进行惯性操作了，而用户是在没有这样的先验知识前提下，来使用产品的。

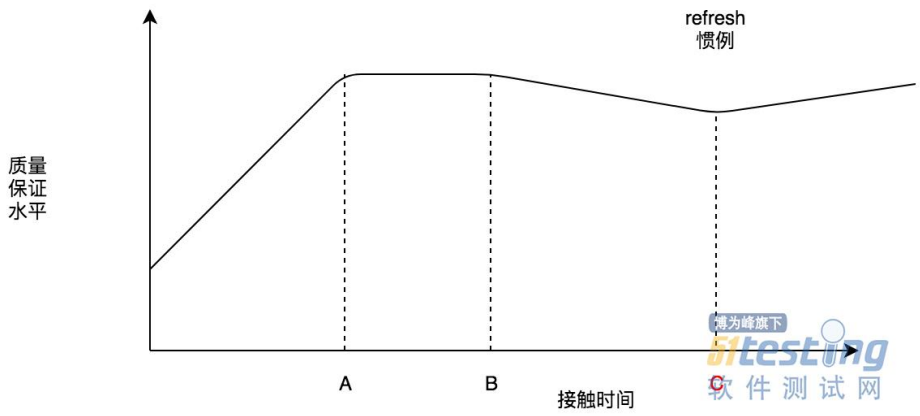
过于依赖先前的测试策略。之前说过，长时间接触一款产品的测试后，必然会形成比较成熟而稳定的测试策略，这时的测试策略当然可以省去一大波测试策略探索时间，但另一方面也会受限于此。首先，测试同学 A 制定出来的看起来成熟而稳定的测试策略，在测试同学 B 看来或许还有大幅度的提升空间；再者，随着产品越来越复杂，实现引入了越来越多的新技术，之前的测试策略未必可以 cover 住。

对用户失去敏感度。归根结底，产品是要服务于用户的，只有用户用的爽了，你的产品才能发挥最大价值。因而，充分了解用户是如何使用产品的至关重要，只有充分站在用户角度，模拟用户使用过程，才能更容易测试出产品的问题。例如，想购买一件商品，你使用直接输入网址来测试购买过程，而用户却常常从分享链接进来，而问题恰好是从分享进来的用户打开网站链接报错了。

测试惯例带来的坏处，虽然看起来不是致命的，但仍然是不是会给整个产品的迭代创造麻烦。比如，长期都遗漏了某种场景的测试，直到发生线上问题才知道；或者某次忽略某个场景带来线上故障；或产品为了逐渐增强易用性，接二连三上线...

打破测试惯例的方法

无论使用什么方法，这里有一个统一打破测试惯例的目标:



把新信息吸收到你大脑中的已知信息中，同时修改已知信息来 refresh 惯例，并让质量

保证水平符合下图走势(C 点之后):

这里举几个自己常用的 refresh 惯例的方法，仅供大家参考:

1）使用产品时，有意识关注那些让自己困惑和烦恼的地方，避免一直依赖下意识操作;

2）拉团队成员一起群策群力，会瞬间给自己测试“灵感”;

3）定期研究用户的使用轨迹、使用习惯，并将其用户测试策略制定上

4）向各个领域的大牛借鉴经验，这些经验会 refresh 你的经验

写在最后

其实无论什么工作，大概都会经历由于惯例的存在，导致让工作水平走了“下坡路”。由于每个人面对的具体情况的差异，“下坡路”到来的时间、到来的次数、持续的周期等会各有不同。了解了这一点，当你感觉自己的水平无提高，甚至在走“下坡路”的时候，记得 refresh 你的惯例啊。

《51 测试天地》(五十三)下篇 精彩预览

马上阅读