目 录 (五十一期·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

测试女巫自动化进化论之“由猴子进化到原始人篇”

作者 :王平平

近期一直在早上上班的地铁上听罗胖的“熊逸书院”，中午吃饭时会看窦文涛的“圆桌派”，我经常觉得自己的内心实际上是一位男士，而且是一位上了年纪的老男士，不太怎么看电视剧，反而很喜欢看这些有阅历，有很多知识积累的人在聊生活中都会遇到的一些别人认为习以为常，且经常被忽略没有人认真思考的问题，实际上这些问题蕴含了很多人生哲理，非常有趣味。

为什么说这些，因为近期女巫的工作遇到了比较大的挑战，越来越发现只有累积了比较多的智慧才能从容面对这些挑战，**近期越来越觉得技术真的不是关键，与人相处:与不同身份不同背景的人相处真的是一个非常大的挑战。**

例如对于能力比较强但是非常有个性的下属(说人话：很难管的下属)，如何相处？当你的指令与 ta 的想法或者说价值观有冲突时如何化解？是使用自己的强权压着 ta 的头去做，还是淡化当时冲突时的情绪，耐着性子了解 ta 的想法，真的找出来你的指令与 ta 的想法的冲突点在哪？有的时候真正的冲突点是超出自己的想象，很多时候下属不愿意去很顺畅地执行你的指令，可能不是你的指令本身的问题，而是 ta 需要面对的合作者的问题，或者 ta 没有理解你的指令的意义进而认为你的指令没有价值。

还有作为主管还要反思是不是因为 ta 的技术能力比较强，而忽略了 ta 的其它能力比较弱，例如沟通能力，而你作为主管是不是过多倚重 ta，造成 ta 压力比较大，而 ta 沟通能力差又无法正确描绘 ta 的真实性想法，只会用情绪化的方式来表达？还有不是每个下属都愿意承担压力，尤其是巨大的压力，即时技术方面比较强的人，不同生活背景的人，面对压力的态度是有很大的差异，所以多读“无用之书”真的非常重要。细节非常重要，很多时候沟通失败都是失败在细节。

聊完工作感受来点干货吧，这次是由原始人进化到原始人了，所以可以吧捂着脸的

手松开一点点，偷偷看看其他的反应了。上次被别人质疑“搞什么搞”时给大家介绍了我们的脚本从一盘散沙进化成有一个“班长”一起管理的阶段了，**这次我们将介绍进一步进化的历程:**

我们考虑需要将自动化和手动测试结合起来，对于一个项目需要有一个概览，需要知道哪些测试项目可以自动化执行，哪些测项必须要手动执行，这个瞬间就高大上许多，因为可以方便估算人力，在发报告时也能清楚地描述出哪些项目是人力测试，哪些是自动化测试。

如下图：在我们的测试用例中加入一列：Autotest script，用以标注哪些测试用例已经有脚本，可以直接使用自动化测试

SMS

Test Item	Test condition		Test Method	Autotest Script	Test Result	Notes
1.get_sms_info						
		执行get_sms_info	查看Response	OK	F	There are sim_max and sim_count in the response.
		执行get_sms_ind	查看Response	OK	P	
		1.执行set_sms_send，任意设置name和content，number设置为有效号码 2.执行get_sms_ind 3.再次执行get_sms_ind	1.查看Response，确认接收方是否可以收到SMS 2.查看Response 3.查看Response	OK	P	The command set_sms_send can't be executed continuously.
		1.执行set_sms_send，任意设置name，content设为长短信(1000个字符)，number设置为有效号码 2.执行get_sms_ind 3.再次执行get_sms_ind	1.查看Response，确认接收方是否可以收到SMS 2.查看Response 3.查看Response	NO	P	
		1.执行set_sms_send，任意设置name和content，number设置为无效号码或空白 2.执行get_sms_ind 3.再次执行get_sms_ind	1.查看Response 2.查看Response 3.查看Response	OK	P	The command set_sms_send can't be executed continuously.
		1.发送一封SMS给DUT，执行get_sms_ind 2.再次执行get_sms_ind	1.查看Response 2.查看Response	OK	P	
		DUT SMS的Inbox中有含有较多字符串的短信,执行get_sms_inbox_records，full_content设置为false	查看Response	OK	P	

最后整理出一份针对某个项目自动化脚本执行的比率，用于给老板汇报，是不是瞬间高大上了一些些~

Auto Test Script Test Range

此项目中自动化脚本可以测试的命令所占比例：

Function	Autotest Command	Total Command of a Function	Autotest Rate
WWAN	22	42	52.38%
Network	3	4	75.00%
System	6	14	42.86%
AT CMD	1	1	100.00%
Contacts	4	4	100.00%
SMS	16	16	100.00%
Total	52	81	64.20%
Notes: 自动化脚本中编写的 test case 针对的是这些命令的基本功能			

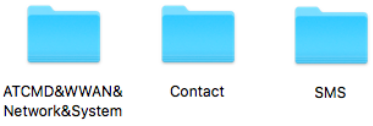
具体自动化脚本可以测试的命令如下：

1. WWAN:

get_wwan_serving_system_provider
get_wwan_serving_system_radio_mode
get_wwan_network_radio_mode
set_wwan_network_radio_mode
get_wwan_band_capability
get_wwan_serving_system_roaming
get_wwan_allow_data_roaming
set_wwan_allow_data_roaming
get_wwan_ipv4_network_state
get_wwan_apn_selection_mode
get_wwan_apn_selection

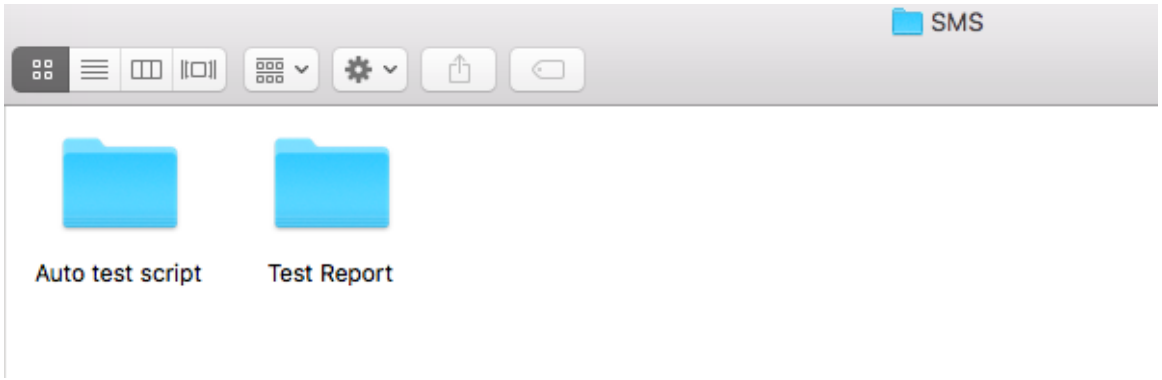
接下来可以看我们脚本的分类：

先看我们脚本如何分布的：对于我们这个产品有 6 个功能其中两个功能的功能比较多，其它四个功能可以合并为一个文件夹如下：

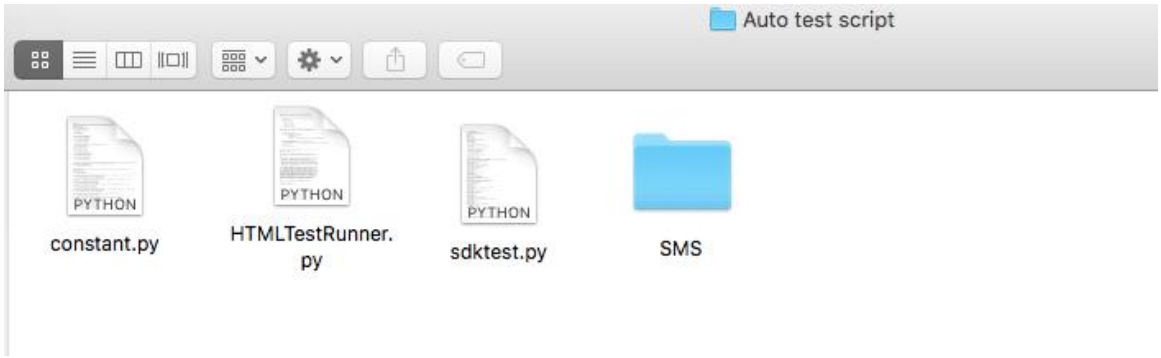


我们以 SMS 为例看我们怎么从猴子进化为原始人

我们相比猴子阶段多了一个 Test report 文件夹，我们将 Test report 和 Auto test script 分成两个文件夹：



Auto test script 文件夹



Constant.py 是我们将一些常量转换为变量被所有的脚本使用,例如发送短消息的号码设置为 18502538324，我们用函数“get_send_to_other_number()”来代表这个常量，如果你想问这样做的意义是什么，我们假设一下，我们后续需要改变发送号码这个常量，例如要改为 18502529422，那我们就需要把这个函数的常量改一下就可以，不用

改变其它的代码，因为在我们的代码中使用是替代此常量的变量:“get_send_to_other_number”所以对于后续的维护有非常大的好处。

constant.py × sdktest.py

```
import os

################################################################################
#SMS data constant
common_string_pass=""""
def get_common_string_pass():
    return common_string_pass

send_name='myname'
def get_send_name():
    return send_name

send_to_other_number='18502538324'
def get_send_to_other_number():
    return send_to_other_number

send_to_own_number='18502529422'
def get_send_to_own_number():
    return send_to_own_number

send_long_content='test1test2test3test4test5test6test7test8'
def get_send_long_content():
    return send_long_content

send_short_content='test'
def get_send_short_content():
    return send_short_content
```

sdktest.py 就是我们再猴子阶段的“班长”，只要执行它就可以执行 SMS 所有的代码;

第一部分代码是控制 Putty（这是一款开源的软件，一般的通讯公司都会使用这款软件），模拟我们测试时的操作，这部分会用到第三方模块：pywinauto(控制 Putty 软件的

界面)以及 pykeyboard(控制键盘，主要用于点击 Enter 键)

Pywinauto 的作用是一层层控制:

先通过指定 putty 所在电脑中的位置来获取 putty 的实例，然后通过先控制 putty 的 dialog 再控制 dialog 中的控件一步步来模拟我们手动测试的步骤:

```
# coding=utf-8
import HTMLTestRunner
from pywinauto import application
import sys
import os
import time
import unittest
import string
from pykeyboard import PyKeyboard
from SMS import SMS
import constant

sys.path.append(os.path.dirname(__file__))、
k=PyKeyboard()
filename=constant.get_file_path_html()

#filepath='D:\\SDK\\putty.log'
filepath=constant.get_file_path_txt()
if os.path.isfile(filepath):
    os.remove(filepath)
else:
    print "The file is not exsit"

##get the dlg of  putty
path=ur"D:\\putty.exe"
app= application.Application().start_(path)
dlg=app.window_(title_re="PuTTY Configuration",class_name="PuTTYConfigBox")
```

Putty 此软件会自动存储测试期间的 log，所以也需要将 log 的存储路径也设置一下

```
##Click the treeview item named session
TreeView=dlg.TreeView
time.sleep(1)
TreeView.GetItem([u'Session']).Click()
time.sleep(1)
##choose the connection type named telnet
serialbutton=dlg.RadioButton3
time.sleep(1)
serialbutton.Click()
time.sleep(1)
##input the com port and speed
comedit=dlg.Edit
time.sleep(1)
comedit.SetText("192.168.1.1")
## Click the treeview item named logging
time.sleep(1)
TreeView.GetItem([u'Session',u'Logging']).Click()
##chose the item named "All session output "
logsession=dlg.RadioButton3
time.sleep(1)
logsession.Click()
time.sleep(1)
##input the addresss of logging filetime.sleep(1)
logedit=dlg.Edit
time.sleep(1)
logedit.SetText("D:\SDK\putty_SMS.log")
time.sleep(1)
```

第二步:

使用 UnitTest 类

从 SMS 中调用各个函数如下:

```
#执行测试的类
class test_sdk_test(unittest.TestCase):
    def setUp(self):
        pass
    ########################################################################
    #no SMS
        #Inbox
    def test_get_sms_info(self):
        SMS.get_sms_info(self,k)

    def test_set_sms_send_and_get_sms_ind_normal_SendtoOther(self):
        SMS.set_sms_send_and_get_sms_ind_normal_SendtoOther(self,k)

    def test_set_sms_send_and_get_sms_ind_abnormal(self):
        SMS.set_sms_send_and_get_sms_ind_abnormal(self,k)

    def test_set_sms_send_and_get_sms_ind_normal_SendtoOwn(self):
        SMS.set_sms_send_and_get_sms_ind_normal_SendtoOwn(self,k)

    def test_get_sms_inbox_records_fullcontent_false(self):
        SMS.get_sms_inbox_records_fullcontent_false(self,k)
```

将这些调用的函数加入到

构造测试集：将上面调用的 SMS 中的函数加入到 unittest 的 test suit 中

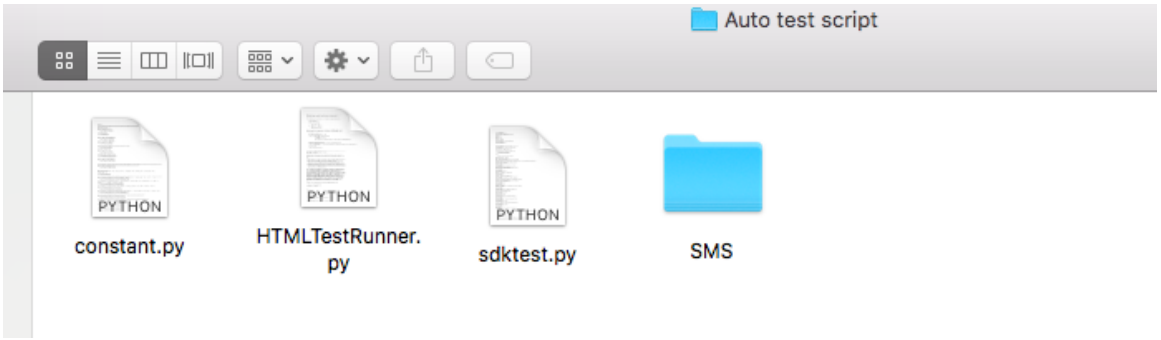
```
if __name__ == "__main__":

    #构造测试集
    testsuite=unittest.TestSuite()
    testsuite.addTest(test_sdk_test("test_get_sms_info"))
    testsuite.addTest(test_sdk_test("test_set_sms_send_and_get_sms_ind_normal_SendtoOther"))
    testsuite.addTest(test_sdk_test("test_set_sms_send_and_get_sms_ind_abnormal"))
    testsuite.addTest(test_sdk_test("test_set_sms_send_and_get_sms_ind_normal_SendtoOwn"))
    testsuite.addTest(test_sdk_test("test_get_sms_inbox_records_fullcontent_false"))
    testsuite.addTest(test_sdk_test("test_get_sms_inbox_records_fullcontent_true"))
    testsuite.addTest(test_sdk_test("test_set_sms_inbox_read"))
    testsuite.addTest(test_sdk_test("test_set_sms_inbox_deleted_one"))
    testsuite.addTest(test_sdk_test("test_get_sms_inbox_records_same_index"))
    testsuite.addTest(test_sdk_test("test_get_sms_inbox_records_different_index"))
    testsuite.addTest(test_sdk_test("test_set_sms_inbox_deleted_all"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_full_content_false"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_full_content_true"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_index_same"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_index_different"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_overlength_content_false"))
    testsuite.addTest(test_sdk_test("test_get_sms_outbox_records_overlength_content_true"))
    testsuite.addTest(test_sdk_test("test_set_sms_outbox_one_deleted"))
```

执行构建的 test suit

```
    #执行测试
    runner.run(testsuite)
```

HTMLTestRunner.py

这个脚本是开源的代码，直接将其放置 Auto test script 中，与 sdktest.py 所处同一个文件夹中



在 sdktest.py 中直接调用此 py 且将其与 unittest 一起打包就可以产生很漂亮的 html

报告

```
#构建report
#filename="D:\\SDK\\SDKresults.html"
fp=file(filename,'wb')
runner=HTMLTestRunner.HTMLTestRunner(stream=fp,title='SDK Test Result',description='Test_Report')


#执行测试
runner.run(testsuite)
```

我们看一下就是这个简单得两行代码产出的报告是什么样的，是不是太让人激动人心了 ，只要告诉他 title 以及描述还有写文档的方式，就可以产生这样的报告！！

SDK Test Result

Start Time: 2016-01-21 15:28:40
Duration: 0:20:40.514000
Status: Pass 35 Failure 4

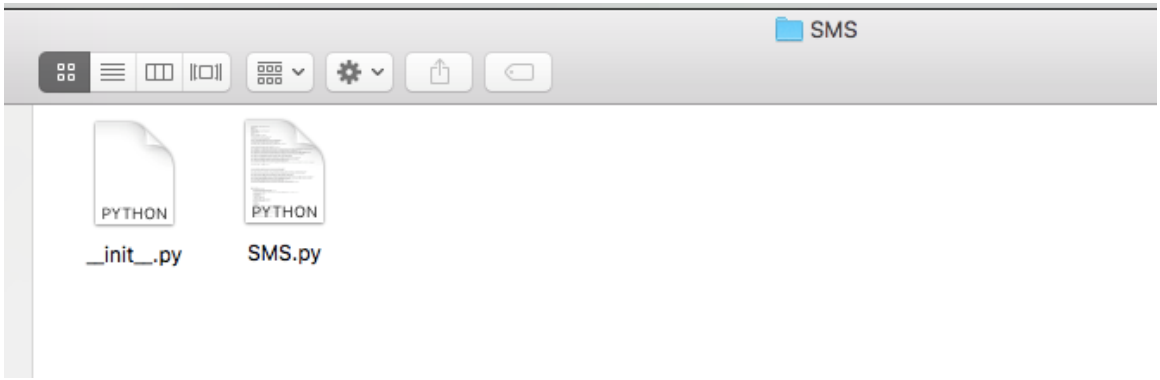
Test_Report

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
test_sdk_test	39	35	4	0	Detail
test_get_sms_info			pass		
test_set_sms_send_and_get_sms_ind_normal_SendtoOther			pass		
test_set_sms_send_and_get_sms_ind_abnormal			pass		
test_set_sms_send_and_get_sms_ind_normal_SendtoOwn			pass		
test_get_sms_inbox_records_fullcontent_false			pass		
test_get_sms_inbox_records_fullcontent_true			pass		
test_set_sms_inbox_read			pass		
test_set_sms_inbox_deleted_one			pass		
test_get_sms_inbox_records_same_index			fail		
test_get_sms_inbox_records_different_index			pass		
test_set_sms_inbox_deleted_all			pass		
test_get_sms_outbox_records_full_content_false			pass		
test_get_sms_outbox_records_full_content_true			pass		

SMS 文件夹

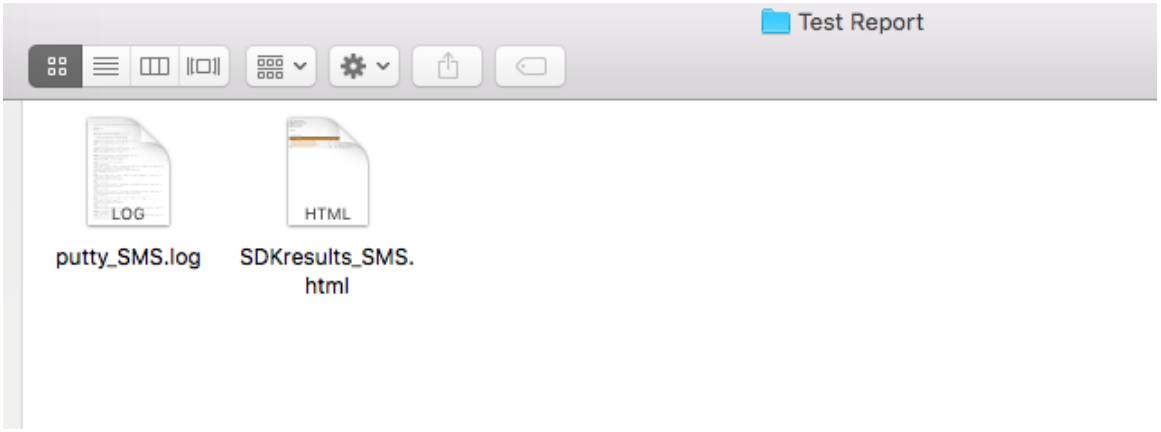
如下，是一组函数的组合，实现测试 SMS 的各个 command



就是类似下面的函数，我们的这个项目是通过各种命令以及查看下了命令后 response 是否符合 spec. 读者可以根据自己的项目需求改为自己的相应的函数。

```
def get_sms_inbox_records_fullcontent_false(self,k):
    sys.path.append(os.path.dirname(__file__))
    #The message with long content,but only display partial contents
    k.type_string('JsonClient /tmp/cgi-2-sys get_sms_inbox_records \'{ "full_content": false}\'')
    time.sleep(2)
    k.tap_key(k.enter_key)
    time.sleep(2)
    f=open(filepath,'r')
    content=f.readlines()
    contentstring=','.join(content)
    #print contentstring
    pattern=re.compile(r'errmsg.*')
    start=0
    location=[]
    while True:
        location_index=contentstring.find('errmsg',start)
        if location_index==-1:
            break
        start=location_index+1
        location.append(location_index)
    result=pattern.findall(contentstring,location[13])
    search_result=sms_inbox_records_full_content_false_pass in result[0]
    self.assertTrue(search_result)
    time.sleep(2)
    f.close()
```

最后的结果会放到 test report 文件夹中如下:



有两个报告：一个是 html 文件，另一个就是 putty 产生的 log

```
=~=~=~=~=~=~=~=~=~=~=~=~=~= PuTTY log 2016.01.21 15:28:31 =~=~=~=~=~=~=~=~=~=~=~=~=~=

home login: root
Password:


BusyBox v1.17.1 (2015-10-16 11:41:30 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

 +-------------------------------------------------------+
 | Lantiq UGW Software DWR-966_v2.1.0.0_r04 on XRX330 CPE |
 +-------------------------------------------------------+

root@home:~# JsonClient /tmp/cgi-2-sys set_sms_inbox_deleted '{ "idx": -1}'
send:
{ "action": "set_sms_inbox_deleted", "args": { "idx": -1 } }
read:
{ "set_sms_inbox_deleted": { "errno": 0, "errmsg": "" } }

root@home:~# JsonClient /tmp/cgi-2-sys set_sms_outbox_deleted '{ "idx": -1}'
send:
{ "action": "set_sms_outbox_deleted", "args": { "idx": -1 } }
read:
{ "set_sms_outbox_deleted": { "errno": 0, "errmsg": "" } }

root@home:~# JsonClient /tmp/cgi-2-sys set_sms_draft_deleted '{ "idx": -1}'
send:
{ "action": "set_sms_draft_deleted", "args": { "idx": -1 } }
read:
{ "set_sms_draft_deleted": { "errno": 0, "errmsg": "" } }

root@home:~# JsonClient /tmp/cgi-2-sys get_sms_info
send:
{ "action": "get_sms_info" }
read:
{ "get_sms_info": { "errno": 0, "errmsg": "", "inbox_max": 250, "inbox_count": 0, "outbox_max": 250, "outbox_count": 0, "draft_max": 250, "draft_count": 0, "sim_max": 50, "sim_count": 2 } }

root@home:~# JsonClient /tmp/cgi-2-sys set_sms_send '{ "sendto":[{"name":"wnc","

number":"18502538324"}],"content":"test"}'
send:
{ "action": "set_sms_send", "args": { "sendto": [ { "name": "wnc", "number": "18502538324" } ], "content": "test" } }
read:
{ "set_sms_send": { "errno": 0, "errmsg": "" } }
```

总结到现在，是不是可以自豪的说：我们已经从猴子进化为原始人了？甚至可以说已经进化到古代人了，但是亲们有没发现我们还差界面，没有界面很难推广到非开发人员使用我们开发的脚本，下一期会大家继续介绍我们的生命进程，下期见喽~

❖ 拓展学习

■打造全能人才,软件测试高级实战进阶：http://www.atstudy.com/classroom/63/introduction

Python 爬虫初探

◆作者：咖啡猫

一、什么叫爬虫

爬虫，又名“网络爬虫”，就是能够自动访问互联网并将网站内容下载下来的程序。它也是搜索引擎的基础，像百度和 GOOGLE 都是凭借强大的网络爬虫，来检索海量的互联网信息的然后存储到云端，为网友提供优质的搜索服务的。

二、爬虫有什么用

你可能会说，除了做搜索引擎的公司，学爬虫有什么用呢？哈哈，总算有人问到点子上了。打个比方吧：企业 A 建了个用户论坛，很多用户在论坛上留言讲自己的使用体验等等。现在 A 需要了解用户需求，分析用户偏好，为下一轮产品迭代更新做准备。那么数据如何获取，当然是需要爬虫软件从论坛上获取咯。所以除了百度、GOOGLE 之外，很多企业都在高薪招聘爬虫工程师。你到任何招聘网站上搜“爬虫工程师”看看岗位数量和薪资范围就懂爬虫有多热门了。

三、爬虫的原理

发起请求：通过 HTTP 协议向目标站点发送请求（一个 request），然后等待目标站点服务器的响应。

获取响应内容：如果服务器能正常响应，会得到一个 Response。Response 的内容便是所要获取的页面内容，响应的内容可能有 HTML，Json 串，二进制数据（如图片视频）等等。

解析内容：得到的内容可能是 HTML，可以用正则表达式、网页解析库进行解析；可能是 Json，可以直接转为 Json 对象解析；可能是二进制数据，可以做保存或者进一步的处理。

保存数据：数据解析完成后，将保存下来。既可以存为文本文档、可以存到数据库中。

四、Python 爬虫实例

前面介绍了爬虫的定义、作用、原理等信息，相信有不少小伙伴已经开始对爬虫感兴趣了，准备跃跃欲试呢。那现在就来上“干货”，直接贴上一段简单 Python 爬虫的代码:

1.前期准备工作：安装 Python 环境、安装 PYCHARM 软件、安装 MYSQL 数据库、新建数据库 exam、在 exam 中建一张用于存放爬虫结果的表格 house [SQL 语句: create table house(price varchar(88),unit varchar(88),area varchar(88));]

2.爬虫的目标：爬取链家租房网上（url: https://bj.lianjia.com/zufang/）首页中所有链接里的房源的价格、单位及面积，然后将爬虫结构存到数据库中。

3.爬虫源代码：如下

```
import    requests #请求 URL 页面内容
from bs4 import BeautifulSoup #获取页面元素
import pymysql #链接数据库
import time #时间函数
import lxml #解析库（支持 HTML\XML 解析，支持 XPATH 解析）
#get_page 函数作用：通过 requests 的 get 方法得到 url 链接的内容，再整合成 BeautifulSoup 可以处理的格式
def get_page(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'lxml')
    return    soup
#get_links 函数的作用：获取列表页所有租房链接
def get_links(link_url):
    soup = get_page(link_url)
    links_div = soup.find_all('div',class_="pic-panel")
    links=[div.a.get('href') for div in links_div]
    return links
#get_house_info 函数作用是：获取某一个租房页面的信息：价格、单位、面积等
def get_house_info(house_url):
  soup = get_page(house_url)
  price =soup.find('span',class_='total').text
  unit = soup.find('span',class_='unit').text.strip()
  area = 'test' #这里 area 字段我们自定义一个 test 做测试
  info = {
    '价格':price,
    '单位':unit,
```

```
            '面积':area
        }
          return    info
        #数据库的配置信息写到字典
        DataBase ={
            'host': '127.0.0.1',
            'database': 'exam',
             'user' : 'root',
             'password' : 'root',
            'charset' :'utf8mb4'}
        #链接数据库
        def get_db(setting):
            return    pymysql.connect(**setting)
        #向数据库插入爬虫得到的数据
        def insert(db,house):
            values = "'{}','"*2 + "'{}'"
            sql_values = values.format(house['价格'],house['单位'],house['面积'])
            sql ="""
            insert    into house(price,unit,area) values({})
            """.format(sql_values)
            cursor = db.cursor()
            cursor.execute(sql)
            db.commit()
        #主程序流程：1.连接数据库 2.得到各个房源信息的 URL 列表 3.FOR 循环从第一个 URL 开始获取房源具体信息（价格等）4.一条一条地插入数据库
        db = get_db(DataBase)
        links = get_links('https://bj.lianjia.com/zufang/')
        for link in links:
            time.sleep(2)
            house = get_house_info(link)
            insert(db,house)
```

首先，“工欲善其事必先利其器”，用 Python 写爬虫程序也是一样的道理，写爬虫过程中需要导入各种库文件，正是这些及其有用的库文件帮我们完成了爬虫的大部分工作，我们只需要调取相关的借口函数即可。导入的格式就是 import 库文件名。这里要注意的是在 PYCHARM 里安装库文件，可以通过光标放在库文件名称上，同时按 ctrl+alt 键的方式来安装，也可以通过命令行（Pip install 库文件名）的方式安装，如果安装失败或者没有安装，那么后续爬虫程序肯定会报错的。在这段代码里，程序前五行都是导入相关的库文件：requests 用于请求 URL 页面内容；BeautifulSoup 用来解析页面元素；pymysql 用于连接数据库；time 包含各种时间函数；lxml 是一个解析库，用于解

析 HTML、XML 格式的文件，同时它也支持 XPATH 解析。

其次，我们从代码最后的主程序开始看整个爬虫流程:

通过 get_db 函数连接数据库。再深入到 get_db 函数内部，可以看到是通过调用 Pymysql 的 connect 函数来实现数据库的连接的，这里**seting 是 Python 收集关键字参数的一种方式，我们把数据库的连接信息写到一个字典 DataBase 里了，将字典里的信息传给 connect 做实参。

通过 get_links 函数，获取链家网租房首页的所有房源的链接。所有房源的链接以列表形式存在 Links 里。get_links 函数先通过 requests 请求得到链家网首页页面的内容，再通过 BeautifuSoup 的接口来整理内容的格式，变成它可以处理的格式。最后通过电泳 find_all 函数找到所有包含图片的 div 样式，再通过一个 for 循环来获得所有 div 样式里包含的超链接页签（a）的内容（也就是 href 属性的内容），所有超链接都存放在列表 links 中。

通过 FOR 循环，来遍历 links 中的所有链接（比如其中一个链接是: https://bj.lianjia.com/zufang/101101570737.html）

用和 2）同样的方法，通过使用 find 函数进行元素定位获得 3）中链接里的价格、单位、面积信息，将这些信息写到一个字典 Info 里面。

调用 insert 函数将某一个链接里得到的 Info 信息写入数据库的 house 表中去。深入到 insert 函数内部，我们可以知道它是通过数据库的游标函数 cursor()来执行一段 SQL 语句然后数据库进行 commit 操作来实现响应功能。这里 SQL 语句的写法比较特殊，用到了 format 函数来进行格式化，这样做是为了便于函数的复用。

最后，运行一下爬虫代码，可以看到链家网的首页所有房源的信息都写入到数据里了。（注：test 是我手动指定的测试字符串）

```
mysql> select * from house;
+-------+-------+------+
| price | unit  | area |
+-------+-------+------+
| 3600  | 元/月 | test |
| 11000 | 元/月 | test |
| 20000 | 元/月 | test |
| 6000  | 元/月 | test |
| 7500  | 元/月 | test |
| 6500  | 元/月 | test |
| 6500  | 元/月 | test |
| 12000 | 元/月 | test |
| 5500  | 元/月 | test |
| 6600  | 元/月 | test |
| 15000 | 元/月 | test |
| 5500  | 元/月 | test |
| 4500  | 元/月 | test |
| 11500 | 元/月 | test |
| 7200  | 元/月 | test |
| 18000 | 元/月 | test |
| 5000  | 元/月 | test |
| 4200  | 元/月 | test |
| 12000 | 元/月 | test |
| 15000 | 元/月 | test |
| 6200  | 元/月 | test |
| 6800  | 元/月 | test |
| 6000  | 元/月 | test |
| 5800  | 元/月 | test |
| 5800  | 元/月 | test |
| 3200  | 元/月 | test |
| 8000  | 元/月 | test |
| 5500  | 元/月 | test |
| 2600  | 元/月 | test |
| 4500  | 元/月 | test |
+-------+-------+------+
30 rows in set (0.01 sec)
```

后记：其实 Python 爬虫并不难，熟悉整个爬虫流程之后，就是一些细节问题需要注意，比如如何获取页面元素、如何构建 SQL 语句等等。遇到问题不要慌，看 IDE 的提示就可以一个个地消灭 BUG，最终得到我们预期的结构。

从一个常见的面试题目说起

◆作者：修远

我 2006 年计算机专业硕士毕业至今，从事软件测试及质量保障领域已经十几个年头了，从白盒自动化测试到黑盒自动化测试，从功能测试到性能测试、安全测试，从测试流程、测试框架到后来的测试管理、质量体系建设，深爱着这个行业，经常为能和小伙伴们攻克了一个技术小山头而激动兴奋， 也为了加班加点使命必达的完成了一个艰巨任务而感到欣慰。

我负责测试技术管理、团队管理也已经有 8 年多了，期间面试过的人少说也有一两千了。从面试中常见的两个场景来聊一聊测试的基本功:

场景:

面试官：你最擅长的测试领域在哪里啊？有什么职业发展计划吗？

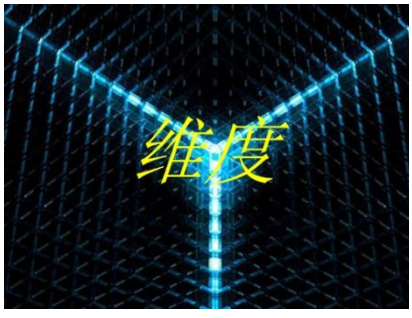
候选人：我比较多的经验在功能测试，以后希望能从事自动化测试方向。

对于这么常见的问题和答案，大家发现了什么问题没有？

我一般遇到这种答案的时候， 会启示候选人这个答案有什么问题吗？当然，有些同学在这个时候，会有点懵。

我继而会启发，那目前的经验和后续希望的方向，有哪些区别？同学们会回答功能测试主要是日常工作，自动化测试更能体现技术水平，也是很多公司喜欢的定位。

其实，候选人的这个答案，最大的问题在从不同的维度来划分了测试种类。



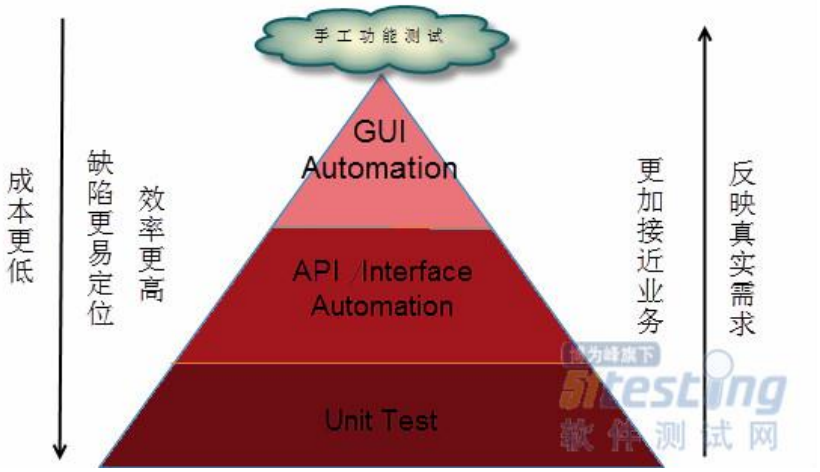
和功能测试同一维度，根据测试的软件内容和特性，可以分为功能测试、性能测

试、安全测试、易用性测试、部署测试等等;

而自动化测试却是从不同的维度，根据测试主题分为了手工测试和自动化测试，也就是说功能测试本身就会有手工的手段和自动化的手段。我们可以通过手工测试来发现很多问题，也可以通过自动化测试提升测试效率，来批量定期执行大量的稳定的测试场景，来进行快速回归。

手工测试仍然是发现 Bug 最有效的手段，尤其是针对新功能点的测试，我们往往是等待手工测试通过后，版本稳定了，再进行自动化脚本的编写和后期的维护。

但是很多同学在谈到功能测试的时候，却习惯的理解成就是手工的。而本身自动化测试也是分层的，如下图，希望能更好的帮助大家理解自动化测试的成本投入和收益分析。



朋友们会问应该如何回答文章开头的那个问题呢？其实不同的候选人根据背景不同，应该有不同的答案，比如有些同学计划后期从事性能测试，那么方向肯定可以往性能测试上说；有些同学计划后期从事安全测试，那么肯定可以往安全测试方面上提；也有部分同学计划做技术管理甚至团队管理，那么答案肯定需要往管理方向上，并且需要体现已经在管理方向上的储备;

针对这个手工功能测试背景的答复，笔者认为比较满意的答复可以这样：我已经有几年的手工功能测试经验...，但我注意到产品的快速迭代对测试人员自动化能力的要求，我们需要将更多的测试用例自动化，通过无人值守的自动化测试，来提升测试效率，所以我希望能提升自己的自动化测试的能力，将宝贵的测试资源更多的投入到新功能及 bug 验证上去，从而和开发、产品一起更有效的提升产品质量。

这样一个答案，既说出了自己的丰富的测试经验，也指出了自动化测试初衷和价值，体现出候选人的好学，对自动化测试有理解有准备，另外表达出了自己和产品、开发凝聚，注重 Team Work 的意愿，说不定你对面的面试官就是开发 leader 呢?

结束语:

相信这个场景，在实际的面试过程中，很多同学都会遇到。作为软件测试工程师，

对面试问题的严谨，注意提升自己良好的沟通和表达能力 ，在面试过程中体现出扎实的软件测试基础知识，足够的耐心、细心、信心、责任心，善于自我总结、自我督促和不断学习的能力，相信大家都能在面试中有出色的表现，找到心仪的工作。

话外篇：

说到这里，大家发现文中的 Bug 了吗？特意留了个彩蛋，一共几个场景分析啊？好像只有一个面试问题分析哈，文章篇首却提了 2 个，不知道作为严谨的测试工程师，您有没有发现呢？

❖ 拓展学习

■软件测试在线就业培训,面试通关不用愁!: http://h.atstudy.com/atstudy/jiuye/atstudy.html

TestNG 关键字 dependsOnMethods 踩坑记

作者：王 练

1. 摘要

TestNG 是使用非常广泛的自动化测试框架，提供了非常强大的功能，采用上手容易的配置文件管理测试流程，解决了自动化测试中的用例驱动和管理问题。TestNG 强大的功能，离不开其丰富的注解机制，注解丰富且含义明确，让使用者见名知意。除了常见的@Test、@Before*、@After*注解外，注解所具有的修饰词也是非常丰富且复杂的。groups、dataProvider 等是使用非常频繁的修饰词，本文介绍出场率不是特别高的 dependsOnMethods 和 dependsOnGroups。

事实上，本文的写作灵感来自一次 dependsOnMethods 的踩坑记录。在实际项目中，为保证测试方法按序执行，采用了 dependsOnMethods 修饰词，结果引发了一系列问题。本文首先简单介绍 TestNG 的注解，给出注解关键字本身的含义和匹配使用的修饰词意义。之后给出 dependsOnMethods 的一般使用方法和软硬依赖的实施策略，从中引出本文踩中 dependsOnMethods 的深坑，之后给出替代的方案。最后引申分析 dependsOnGroups 的类似使用方法、问题和解决方案。

2. dependsOnMethods 来源-TestNG 注解简介

TestNG 是灵感来自 JUnit 和 NUnit 的自动化测试框架，其中的 NG 表示 Next Generation，即下一代的含义。该框架功能强大、使用方便，在测试的各个领域都有非常广泛的应用。

该框架继承了 JUnit 的注解模式，通过丰富的注解完成强大的功能。具体包括如下注解。

注解	注解方法	修饰词
@BeforeSuite	在该套件的所有测试都运行在	在存在父类子类继承关系时，左侧的注

	注解的方法之前，仅运行一次。	解如果在父类存在，则会被子类继承。这对于将测试环境准备集中在一个通用父类完成。此时，TestNG 会确保@Before 方法根据继承的顺序执行，即父类先行，然后验证继承链，按序向下执行；@After 方法逆序向回执行，直到第一个父类。 alwaysRun：对于全部的@Before 方法，除@BeforeGroups 之外，该属性置为 true，则无论该方法属于哪个组，都会运行。对于全部@After 方法，如果设置为 true，则即使之前的测试方法出现失败或跳过，该@After 方法也会执行。 dependsOnGroups：该方法依赖的组列表。 dependsOnMethods：该方法依赖的方法列表。 enabled：该方法或类是否启用。 groups：该方法或类所属组列表。 inheritGroups：设置为 true 时，该方法会继承所属类的@Test 注解指定的组列表。 onlyForGroups：只针对@BeforeMethod 和@AfterMethod 生效，如果指定，则该@BeforeMethod 和@AfterMethod 方法，仅会在属于该组列表的测试方法执行的时候，才会被激活。
@AfterSuite	在该套件的所有测试都运行在注解方法之后，仅运行一次。	
@BeforeTest	注解的方法将在属于<test>标签内的类的所有测试方法运行之前运行	
@AfterTest	注解的方法将在属于<test>标签内的类的所有测试方法运行之后运行。	
@BeforeGroups	配置方法将在之前运行组列表。 此方法保证在调用属于这些组中的任何一个的第一个测试方法之前运行。	
@AfterGroups	此配置方法将在之后运行组列表。该方法保证在调用属于任何这些组的最后一个测试方法之后运行。	
@BeforeClass	在调用当前类的第一个测试方法之前运行，注解方法仅运行一次。	
@AfterClass	在调用当前类的第一个测试方法之后运行，注解方法仅运行一次	
@BeforeMethod	注解方法将在每个测试方法之前运行。	
@AfterMethod	注解方法将在每个测试方法之后运行。	
@DataProvider	该方法为提供测试数据的方法。方法必须返回二维数组 Object[][]，其中每一组 Object[]为测试方法的一组测试数据。以@Test 注解的方法会尝试从该方法中获取指定的同名数据。	name：提供的数据名称，如果没有指定，会自动设定为该方法的名字。 parallel：默认为 false，如果设定为 true，数据会并行生成。
@Factory	标记该方法为一个工厂方法，返回值作为 TestNG 的测试类，必须返回 Object[]。	
@Listeners	定测试类的监听器。	value:监听器数组，监听器是继承与 org.testng.ITestNGListener 的类。
@Parameters	传递给@Test 注解的方法的参数。	value:传递的参数列表。
@Test	标记该类或方法进行测试。	

TestNG 的注解是以注解关键字和修饰词组成，修饰词实际是一组 key-value 的键值对，通过注解传递给 TestNG，根据 TestNG 的处理逻辑进行处理。上述注解的@Test 的修饰词比较多，集中说明如下:

修饰词	作用
alwaysRun	如果设置为 true，即使该方法依赖的方法失败了，该方法依然会执行。否则，该方法会跳过。
dataProvider	提供给该方法的测试数据名称。
dataProviderClass	提供测试数据的类，如果没有指定需要测试数据的方法会在当前类，或当前类的父类尝试寻找。如果使用该属性指定测试数据提供类，则提供测试数据的方法需要设定为 static，否则无法调用。
dependsOnGroups	该方法依赖的组列表。
dependsOnMethods	该方法依赖的方法列表。
description	该方法的描述信息。
enabled	是否启用该方法/类。
expectedExceptions	测试方法期望抛出的异常列表，如果该方法没有抛出任何异常，或者抛出的异常不属于配置列表中的，则该方法会置为测试失败。
groups	该方法/类所属的组列表。
invocationCount	方法被调用的次数。
invocationTimeOut	测试执行的超时时间，invocationCount 次数的累积时间，以毫秒计数。如果 invocationCount 未指定，则该值无效。
priority	测试方法的优先级，数字越小级别越高，高级的会先执行。
successPercentage	测试方法期望的执行成功百分比。
singleThreaded	如果设置为 true，TestNG 会确保该测试类的全部测试方法运行在一个线程中，即使该测试设定的并行级别为 parallel="methods"。该属性只能应用在 class 上，TestNG 忽略在 method 上的该修饰。注意：该修饰当需要串行执行时使用，目前不推荐使用(deprecated)。
timeOut	测试方法执行超时时间，以毫秒计数。
threadPoolSize	该方法的线程池大小。当设定方法的 invocationCount 属性后，方法会议多线程的方式运行。注意：如果 invocationCount 属性没有设定，则该属性会被 TestNG 忽略。

本文关心的注解为 dependsOnMethods 和 dependsOnGroups，主要说明前者，后者作为引申阅读简单说明。

3. dependsOnMethods 基本使用

dependsOnMethods 作为 TestNG 的修饰词，可以对@Before*、@After*和@Test 进行修饰，这些注解都是针对测试方法使用的，所以 dependsOnMethods 作用是注入该测试方法依赖的测试方法列表。

通常测试方法依赖其他测试方法，主要的目的是要求测试方法以固定顺序执行测试方法，例如:

确保运行相关测试方法前运行且成功特定个数的测试方法;

需要使用测试方法作为其他测试的初始化方法，如果使用@Before*或@After 的方法不记录最终的报告。

一个简单的示例：测试类源码

```
public class dependsOnClass1{
    @Test
    public void method_b(){
        System.out.println("被依赖的方法首先运行method_b");
    }
    @Test(dependsOnMethods={"method_b"})
    public void method_a(){
        System.out.println("依赖方法之后运行method_a");
    }
}
```

TestNG 配置文件

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1">
      <test name="Login" preserve-order="true" verbose="1">
         <classes>
             <class name="demo.dependsOnClass1" />
         </classes>
    </test>
</suite>
```

运行结果:

```
<terminated> dependsOnClass1 [TestN
[TestNG] Running:
  E:\1-WebUIAutoTest\Source_Code

被依赖的方法首先运行method_b
依赖方法之后运行method_a

===================================
```

去掉依赖的运行结果:

```
依赖方法之后运行method_a
被依赖的方法首先运行method_b
```

4. dependsOnMethods 详细介绍

dependsOnMethods 的使用方法有两种，为描述方便称 dependsOnMethods 修饰的测试方法为方法，被依赖的方法称为被依赖方法:

硬依赖：所有方法依赖的被依赖方法必须运行且成功。如果被依赖方法列表中有一个失败，则 TestNG 不会运行方法，标记方位为跳过。

示例:

```
public class dependsOnClass1{
    @Test
    public void method_b(){
        System.out.println("被依赖的方法首先运行method_b");
        assert(1>2);
    }
    @Test
    public void method_c(){
        System.out.println("被依赖的方法首先运行method_c");
    }
    @Test(dependsOnMethods={"method_c","method_b"})
    public void method_a(){
        System.out.println("依赖方法之后运行method_a");
    }
}
```

运行结果:

```
被依赖的方法首先运行method_b
被依赖的方法首先运行method_c

===============================================
[redacted]WebUI-AutoTest
Total tests run: 3, Failures: 1, Skips: 1
===============================================
```

这里可以看到，method_a 硬依赖的方法 method_b 运行失败，method_a 并未执行，标记为 Skip。同时也可以看到，dependsOnMethods 的方法列表是不能决定最终的被依赖方法的顺序的，如果需要 method_c 方法在 method_b 方法前运行，需要指定 method_b 方法的依赖 method_c 方法。

软依赖：TestNG 永远运行方法，及时存在被依赖方法运行失败。这适用于我们只要求全部方法以特定顺序运行的场景，事实上此时方法运行的成功与否与被依赖方法并没有真正的依赖关系。通过在方法中设定 "alwaysRun=true"的修饰词实现该功能。

示例:

```
public class dependsOnClass1{
    @Test
    public void method_b(){
        System.out.println("被依赖的方法首先运行method_b");
        assert(1>2);
    }
    @Test
    public void method_c(){
        System.out.println("被依赖的方法首先运行method_c");
    }
    @Test(dependsOnMethods={"method_c","method_b"},alwaysRun=true)
    public void method_a(){
        System.out.println("依赖方法之后运行method_a");
    }
}
```

运行结果:

```
被依赖的方法首先运行method_b
被依赖的方法首先运行method_c
依赖方法之后运行method_a

===============================================
    WebUI-AutoTest
Total tests run: 3, Failures: 1, Skips: 0
===============================================
```

这里可以看到，虽然 method_a 软依赖的方法 method_b 运行失败，但 method_a 依然执行。

5. dependsOnMethods 潜在问题

dependsOnMethods 能够确保测试方法按照确定的顺序进行，貌似很完美。但当存在多个测试类的时候，就出现问题了。比如下面的示例。

第一个测试类:

```
public class dependsOnClass1{
    @Test
    public void class1_method_b(){
        System.out.println("被依赖的方法首先运行class1_method_b");
    }
    @Test(dependsOnMethods={"class1_method_b"})
    public void class1_method_a(){
        System.out.println("依赖方法之后运行class1_method_a");
    }
}
```

第二个测试类:

```
public class dependsOnClass2{
    @Test
    public void class2_method_b(){
        System.out.println("被依赖的方法首先运行class2_method_b");
    }
    @Test(dependsOnMethods={"class2_method_b"})
    public void class2_method_a(){
        System.out.println("依赖方法之后运行class2_method_a");
    }
}
```

TestNG 配置文件:

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1">
      <test name="Login" preserve-order="true" verbose="1">
         <classes>
             <class name="demo.dependsOnClass1" />
             <class name="demo.dependsOnClass2" />
         </classes>
    </test>
</suite>
```

由于配置了 preserve-order 的属性，所以我们期望的运行结果是：首先将 demo.dependsOnClass1 的全部方法按照特定的顺序运行完成，之后运行 demo.dependsOnClass2 的测试方法，实际的运行结果:

```
被依赖的方法首先运行class1_method_b
被依赖的方法首先运行class2_method_b
依赖方法之后运行class1_method_a
依赖方法之后运行class2_method_a

===============================================
WebUI-AutoTest
Total tests run: 4, Failures: 0, Skips: 0
===============================================
```

可以看到，首先运行了 demo.dependsOnClass1 和 demo.dependsOnClass2 的被依赖方法，再按照 class 的顺序运行其他测试方法。

这个显然不是我们想要的效果。在 UI 自动化测试中测试方法需要在特定的界面进行测试，如果运行了其他测试类的被依赖方法，该测试类的测试方法的初始界面就无法保证了。比如上述 dependsOnClass1 中的 class1_method_a 是测试修改用户信息的用例，被依赖方法的作用在于添加预置的待修改用户，完成后停留在用户管理界面；而 dependsOnClass2 的 class2_method_a 是测试添加主机信息的用例，被依赖方法的作用是进入主机信息管理界面。此时按照当前的情况，就会导致 class1_method_a 基于主机信息管理界面进行测试，结果为失败。

当然我们可以通过将所有测试方法都基于一个基础界面完成，但这无形中加大了复杂度，增加了出错的可能。同时，目前是两个测试类的依赖情况，如果多个类的时候，情况会更复杂，测试方法的执行顺序会更加烧脑。

上述示例测试的是在同一个 test 标签内的 dependsOnMethods 情况，如果在不同的 test 中又是怎么样的呢？

增加两个示例类，源码与 demo.dependsOnClass1 和 demo.dependsOnClass2 类似。TestNG 配置文件修改如下：

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1">
      <test name="depends1-2" preserve-order="true" verbose="1">
         <classes>
             <class name="demo.dependsOnClass1" />
             <class name="demo.dependsOnClass2" />
         </classes>
    </test>
    <test name="depends3-4" preserve-order="true" verbose="1">
         <classes>
             <class name="demo.dependsOnClass3" />
             <class name="demo.dependsOnClass4" />
         </classes>
    </test>
</suite>
```

运行结果如下：

```
被依赖的方法首先运行class1_method_b
被依赖的方法首先运行class2_method_b
依赖方法之后运行class1_method_a
依赖方法之后运行class2_method_a
被依赖的方法首先运行class3_method_b
被依赖的方法首先运行class4_method_b
依赖方法之后运行class3_method_a
依赖方法之后运行class4_method_a

===============================================
WebUI-AutoTest
Total tests run: 8, Failures: 0, Skips: 0
===============================================
```

可以看到，不同 test 间的测试方法运行还是按照既定的顺序的。当然，这是基于 parallel="tests" thread-count="1"的配置前提下，如果是 parallel="tests" thread-count="2"，就会是另一番景象。

同一个 test 标签内的 dependsOnMethods 引起的运行顺序问题，让我深深的踩进了坑中，通过不懈的努力，终于在 Stack Overflow 中找到了类似的解答。

The problem was in testng logic. Through tons of experiments it was defined, that TestNG always runs dependent methods in the end of parallel run. Means, i.e. you have 3 Test Classes: Test1.java Test2.java Test3.java

and each has some test methods.

TestNG suite contains that 3 classes will run each non-dependent method from those classes, than come back and finish run of those dependent methods which left.

简单的说，就是 TestNG 就是这样设计的，对于同一 test 标签内的所有测试方法，无论是否是同一测试类，都会将被依赖的方法先运行，再处理剩下的测试方法。使用 dependsOnMethods 会让测试代码的逻辑变得非常复杂，但是测试方法按特定顺序执行的需求如何实现呢？有没有替代方案？

6. 替代方案

6.1、group-by-instances 方法

通过修饰词 group-by-instances 能够将上述问题解决。测试类依然是 demo.dependsOnClass1 和 demo.dependsOnClass2，TestNG 配置文件修改如下：

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1" >
      <test name="depends1-2" preserve-order="true" verbose="1" group-by-instances="true">
         <classes>
             <class name="demo.dependsOnClass1" />
             <class name="demo.dependsOnClass2" />
         </classes>
    </test>
</suite>
```

运行结果如下：

```
被依赖的方法首先运行class2_method_b
依赖方法之后运行class2_method_a
被依赖的方法首先运行class1_method_b
依赖方法之后运行class1_method_a

===============================================
WebUI-AutoTest
Total tests run: 4, Failures: 0, Skips: 0
===============================================
```

可以看到，运行每个测试类的被依赖方法， 之后是该类的测试方法；然后是第二个测试类的被依赖方法……但是，会发现有另一个问题，此时 preserve-order="true"失效了。上述配置应该先运行 dependsOnClass1，之后运行 dependsOnClass2。

这个问题无法解决。通过查阅很多资料，很多人遇到该问题，有人猜测是 TestNG 的逻辑 Bug。当使用 group-by-instances 时，preserve-orde、priority 类似都会失效。或者并不是 Bug，但 TestNG 的内在逻辑，表现出来的就是，二者无法并存，结果目前不符

合预期。

6.2、methods-include 方法

使用 dependsOnMethods 目的在于将测试方法按确定的顺序执行，methods-includes 方法可以实现该需求。

测试代码去掉 dependsOnMethods 说明，以 demo.dependsOnClass1 为例：

```
public class dependsOnClass1{
    @Test
    public void class1_method_b(){
        System.out.println("被依赖的方法首先运行class1_method_b");
    }
    @Test
    public void class1_method_a(){
        System.out.println("依赖方法之后运行class1_method_a");
    }
}
```

TestNG 配置文件修改如下：

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1" >
      <test name="depends1-2" preserve-order="true" verbose="1" >
         <classes>
             <class name="demo.dependsOnClass1">
                 <methods>
                     <include name="class1_method_b" />
                     <include name="class1_method_a" />
                 </methods>
             </class>
              <class name="demo.dependsOnClass2">
                 <methods>
                     <include name="class2_method_b" />
                     <include name="class2_method_a" />
                 </methods>
             </class>
         </classes>
    </test>
</suite>
```

运行结果如下：

```
被依赖的方法首先运行class1_method_b
依赖方法之后运行class1_method_a
被依赖的方法首先运行class2_method_b
依赖方法之后运行class2_method_a

===============================================
WebUI-AutoTest
Total tests run: 4, Failures: 0, Skips: 0
===============================================
```

可以看到，运行每个测试类的被依赖方法， 之后是该类的测试方法；然后是第二个测试类的被依赖方法……并且先运行 dependsOnClass1，之后运行 dependsOnClass2。

完全可以按照测试需求运行。

使用 methods-includes 方法，会使得 TestNG 配置文件变得臃肿复杂，有利有弊。

7. dependsOnGroups 引申阅读

7.1、使用方法

dependsOnGroups 使用方法与 dependsOnMethods 类似，示例代码如下。

```
public class dependsOnGroupsClass1{
    @Test(groups={"init1"})
    public void class1_method_b(){
        System.out.println("被依赖的方法首先运行class1_method_b");
    }
    @Test(dependsOnGroups={"init1.*"})
    public void class1_method_a(){
        System.out.println("依赖方法之后运行class1_method_a");
    }
}
```

TestNG 配置文件如下:

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1" >
      <test name="depends1-2" preserve-order="true" verbose="1" >
         <classes>
             <class name="demo.dependsOnGroupsClass1" />
         </classes>
    </test>
</suite>
```

执行结果:

```
被依赖的方法首先运行class1_method_b
依赖方法之后运行class1_method_a

===============================================
WebUI-AutoTest
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

软硬依赖，即 alwaysRun 的使用方式与 dependsOnMethods 一致，此处略。与 dependsOnMethods 不同的是，dependsOnGroups 还可以通过配置文件指定依赖。上述代码修改如下。

```
public class dependsOnGroupsClass1{
    @Test(groups={"init"})
    public void class1_method_b(){
        System.out.println("被依赖的方法首先运行class1_method_b");
    }
    @Test(groups={"demo"})
    public void class1_method_a(){
        System.out.println("依赖方法之后运行class1_method_a");
    }
}
```

配置文件修改如下:

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1" >
   <test name="depends1-2" preserve-order="true" verbose="1" >
       <groups>
          <dependencies>
             <group name="demo" depends-on="init" />
          </dependencies>
        </groups>
        <classes>
            <class name="demo.dependsOnGroupsClass1" />
         </classes>
    </test>
</suite>
```

执行结果如下:

```
被依赖的方法首先运行class1_method_b
依赖方法之后运行class1_method_a

===============================================
WebUI-AutoTest
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

7.2、潜在问题和替代方案

与 dependsOnMethods 类似，dependsOnGroups 当存在多个测试类的时候，运行顺序无法满足需求。测试代码 dependsOnGroupsClass1:

```
public class dependsOnGroupsClass1{
    @Test(groups={"init1"})
    public void class1_method_b(){
        System.out.println("被依赖的方法首先运行class1_method_b");
    }
    @Test(dependsOnGroups={"init1"})
    public void class1_method_a(){
        System.out.println("依赖方法之后运行class1_method_a");
    }
}
```

测试代码 dependsOnGroupsClass1:

```
public class dependsOnGroupsClass2{
    @Test(groups={"init2"})
    public void class2_method_b(){
        System.out.println("被依赖的方法首先运行class2_method_b");
    }
    @Test(dependsOnGroups={"init2"})
    public void class2_method_a(){
        System.out.println("依赖方法之后运行class2_method_a");
    }
}
```

TestNG 配置文件:

```
<suite name="WebUI-AutoTest" parallel="tests" thread-count="1" >
   <test name="depends1-2" preserve-order="true" verbose="1" >
      <classes>
         <class name="demo.dependsOnGroupsClass1" />
         <class name="demo.dependsOnGroupsClass2" />
      </classes>
    </test>
</suite>
```

运行结果:

```
被依赖的方法首先运行class1_method_b
被依赖的方法首先运行class2_method_b
依赖方法之后运行class1_method_a
依赖方法之后运行class2_method_a

===============================================
WebUI-AutoTest
Total tests run: 4, Failures: 0, Skips: 0
===============================================
```

可见于 dependsOnMethods 一样，在同一 test 标签内的所有测试方法，无论是否是同一测试类，都会将被依赖的方法先运行，再处理剩下的测试方法。

解决方法也是类似的，使用 methods-include 方法。具体代码省略，最终运行结果如下。

```
被依赖的方法首先运行class1_method_b
依赖方法之后运行class1_method_a
被依赖的方法首先运行class2_method_b
依赖方法之后运行class2_method_a

===============================================
WebUI-AutoTest
Total tests run: 4, Failures: 0, Skips: 0
===============================================
```

8. 总结

TestNG 通过注解实现了非常强大的功能，但有些注解单从字面理解还算明确，不过

在稍微复杂的情况下，就会出现不符合预期的工作结果。本文从 TestNG 的注解介绍开始，引出 dependsOnMethods 和 dependsOnGroups 的需求场景和使用方法，并指出其问题。为防止其他使用该关键字的工程师们不再踩坑，给出了替代的解决方案。

在使用 dependsOnMethods 和 dependsOnGroups 时，一定要做好多个 class 标签、多个 test 标签的验证。如果需要满足测试方法按既定顺序执行，推荐使用 methods-include 方法。顺序完全可控，通过配置文件也能够完全获取测试方法的执行顺序。

本文使用的 TestNG 版本：6.9.9。

掌握测试驱动开发的 3 个关键因素

◆译者：枫 叶

从戴维·恩斯坦教软件开发者们如何更有效地以测试驱动开发的 10 年来，他学会了掌握测试驱动开发的 3 个关键组成部分：理解它真正是什么，使代码稳定可测，并且获得实际动手经验。让我们看这些因素，找到它在你的项目中为有效地使用测试驱动开发带来什么。

在过去 10 年来，我有教数千个专业软件开发者如何有效地测试驱动开发的权利。从这些经验中，我学会了测试驱动开发的 3 个关键因素：理解它真正是什么，使代码确实可测试，并且获取一手的经验。让我们看一看这些因素中的每一个，去看看使用测试驱动开发能有效地在你的项目中带来什么。

1、理解什么是测试驱动开发

有效的测试驱动开发的第一主要的组成部分是理解它真正是什么。我发现有很多关于如何恰当地做测试驱动开发的错误想法，并且测试驱动开发是那种，如果你做错了，你会经常付出一个很高的代价的实践。

在这篇短的文章中有更多的介绍测试驱动开发，但是我注意到的其中一件事是对人们来讲更有挑战的是人们把测试驱动开发想象成测试或者质量保证的一种形式。我认为在做测试驱动开发时，这是错误心态。

QA 的思维模式是在思考可能出现的问题并且找到保证它不再发生的方法。开发的思维模式更乐观，专注于发生的事情，为了事情顺利进行。

不考虑将测试驱动开发作为测试代码的一种方式，我想把测试驱动开发想成系统特殊行为的一种方式。这引导我创造非常不同的测试种类，它趋向于将来更有弹性地改变，因为我正在验证行为而不是测试代码块。

术语“单元测试”也有点误解,假如我们在写代码之前写测试，那么它实际上不是一次测试，因为当我们写它时，没有可测试的东西。在这个点上叫它一次测试有点奇怪。反之，我想要把它认为是一种假说。当我们在写代码之前写测试时，我们在假定代码将

如何运行，我们需要通过什么，以及将返回什么。

这类似于我们如何接近科学。我们不能随机运行实验。我们经常开始一种假设：我们正在尝试通过或不通过的东西。我们能想出一个实验去通过或者不通过我们的假设。把你的实验考虑成假设，并且你写的代码是为了使测试作为一种实验通过，来证明这个假设。

但更大的误解是，我发现人们在尝试 TDD 时真的会挂起电话，这是他们认为“单元”的意思。对很多开发来讲，当他们在“单元测试”里看到术语“单元”，他们想到一段代码，像一个方法或者一段声明，或者甚至一行简单的代码。这不是本文中“单元”的含义。就像我理解它的，术语“单元”被接受成强调一个功能独立的单元。

理想情况下，我们后面的行为是对我们正尝试获取的验收标准的间接支持。当单元测试也是验收测试时，我们自然得到需求可追溯性和可验证性。

一个“行为单元”可能包含一些以合作为工作目的的对象。举个例子，在拍卖中测试投标规则可能需要一个卖方对象来创建一个拍卖对象和一个投标人对象在拍卖中投标。有些人将把它作为一次集成测试，因为它包含一些对象的交互。我把它称为一个单元测试，因为我正在测试投标行为的一个单元。

我经常发现当我们关注于可以满足验收标准的附加特性时，我们编写的代码的维护成本要低得多，因为设计更容易理解和扩展。

2、使不可测试的代码可测

学习测试驱动开发的第二个关键因素是掌握一系列使不可测试的代码可测试的技术。很多已存在的代码是很难测试的，并且当我们不得不和那些代码交互时，很难让它接受测试。

通过我参观的很多公司，我在代码中看到的一个主要问题是为了使用一项服务，一位顾客将被实例化并且直接呼叫那项服务。从外部来看，服务和服务的客户端似乎是相同的，不能被拆分。但是，当这一系统在整个系统中反复进行时，它使系统成为一个纠缠在一起的代码鼠窝，不可能独立地进行分离和测试。

这个问题的解决方案是一种称为依赖注入的技术。你可能很熟悉依赖注入框架的独立性，比如 Spring。但是你能手动地注入依赖，不使用一个框架。代替使一个对象实例化一个服务然后使用它，我们委托实例化成一个不同的对象，然后将引用传递给使用它

的客户端。

允许对服务的引用传递给服务的使用者，当我们测试时让我们传入假。它是一个简单的概念，对做小的、可测试的单元行为和分开整体的代码是相当重要的。

我可以通过几种虚拟来代替依赖。一个通过简单的归类和覆盖你的代码交互的方法来创造手工模拟的方法。你可以调用你的 mock 的覆盖方法，而不是调用真正的依赖关系，它能返回任何有意义的东西。记住，我们这儿的目标是测试我们的代码可交互与外部依赖，而不是它自身的依赖。

3、做测试驱动开发的经验

拥有编写良好的行为测试和能够编写好的、可测试的代码的技能，只是掌握测试驱动开发所需的部分内容。第三和最重要的掌握测试驱动开发的因素是体验去做它。当开发者做了测试驱动开发并且看到他们的测试如何立即捕捉问题——结果是他们的代码有多好——他们开始在项目中做测试驱动开发。

在绿色田野项目中学习测试驱动开发是有用的，因为在遗留代码上进行测试驱动开发会有更多复杂性。这本身就是一个完成的学习领域，在项目中有一些优秀的书。我想每一个专业软件开发应该读马丁·福勒的重构：改进已有代码的设计。而且如果你正在遗留代码上工作，然后你也应该读迈克尔·西·羽毛的有效率地同老代码工作——并且不要忘了查看一下我的书，在遗留程序之上：9 种扩展你的软件生命（和价值）的实践。

当我首次启动教软件开发关于测试——驱动开发，我给他们关于真正的测试驱动开发以及如何使不可测试的代码可测试的讲座。他们知道该怎么做，但我们没有在一个项目上一起实践 TDD，所以它没有坚持下去。6 个月后我将返回，在团队中没有人再做测试驱动开发了。

但是当我开始包括 12 个小时的实践练习使用测试驱动开发作为培训的一部分，我看到人们做了一个彻底的转变，因为他们看到了做 TDD 的好处。这确实是我们学习和得到新行为的唯一的方法：通过做它们并且向我们证明它们是有价值的。你不能从听别人说一个话题而获得经验。

理解真正的测试驱动开发是什么，知道如何使不可测试的代码可测，在项目中做测试驱动开发获取实践经验的好处是掌握测试驱动开发的 3 个关键因素。当开发有这 3 个关键因素时，他们为测试驱动开发而感到高兴，并且在他们的项目中持续做这件事。

如何在 Selenium 项目中使用 GeckoDriver

◆ 译者：小布丁

为了理解什么是 GeckoDriver，首先我们需要了解 Gecko 和浏览器引擎。这个教程涵盖了 GeckoDriver 的所有特点，这里完整的叙述了这些特点。

所以首先，让我们先了解什么是 Gecko，什么是浏览器引擎?

什么是 Gecko?

Gecko 是一种浏览器引擎。有几个应用需要用到 Gecko。尤其是，火狐公司开发的应用。Gecko 也支持一些开源的项目。Gecko 是由 C++和 JavaScript 开发的。

最新的版本是 Rust 写的。Gecko 一种免费开源的浏览器引擎。

什么是浏览器引擎?

浏览器引擎只是一个软件程序。这个程序的主要功能是收集内容（像 HTML，XML，图片）&格式化信息（像 CSS）并且在屏幕上显示格式化后的内容。浏览器引擎也被称作布局引擎或者渲染引擎。

像浏览器、邮件客户端、电子书阅读器、在线帮助等工具，需要展示网页内容。为了展示这些内容，浏览器引擎是必不可少的并且是这些工具的一部分。每个浏览器都有不同德浏览器引擎。

下面的表格列出了浏览器和对应使用的浏览器引擎。

S.No	Browsers	Web Browser Engines
1	Internet Explorer	Trident
2	Firefox	Gecko
3	Safari	WebKit
4	Google Chrome	Blink
5	Opera	Presto

Gecko 可以在下面的操作系统中直接使用:

- Windows
- Mac OS
- Linux
- BSD
- Unix

塞班系统不能使用

什么是 GeckoDriver?

GeckoDriver 在 Selenium 脚本中用来连接到火狐浏览器。GeckoDriver 是一个代理，这个代理可以通过提供的 HTTP 接口和狐火浏览器保持通信(火狐浏览器举例)。

为什么 Selenium 需要 GeckoDriver?

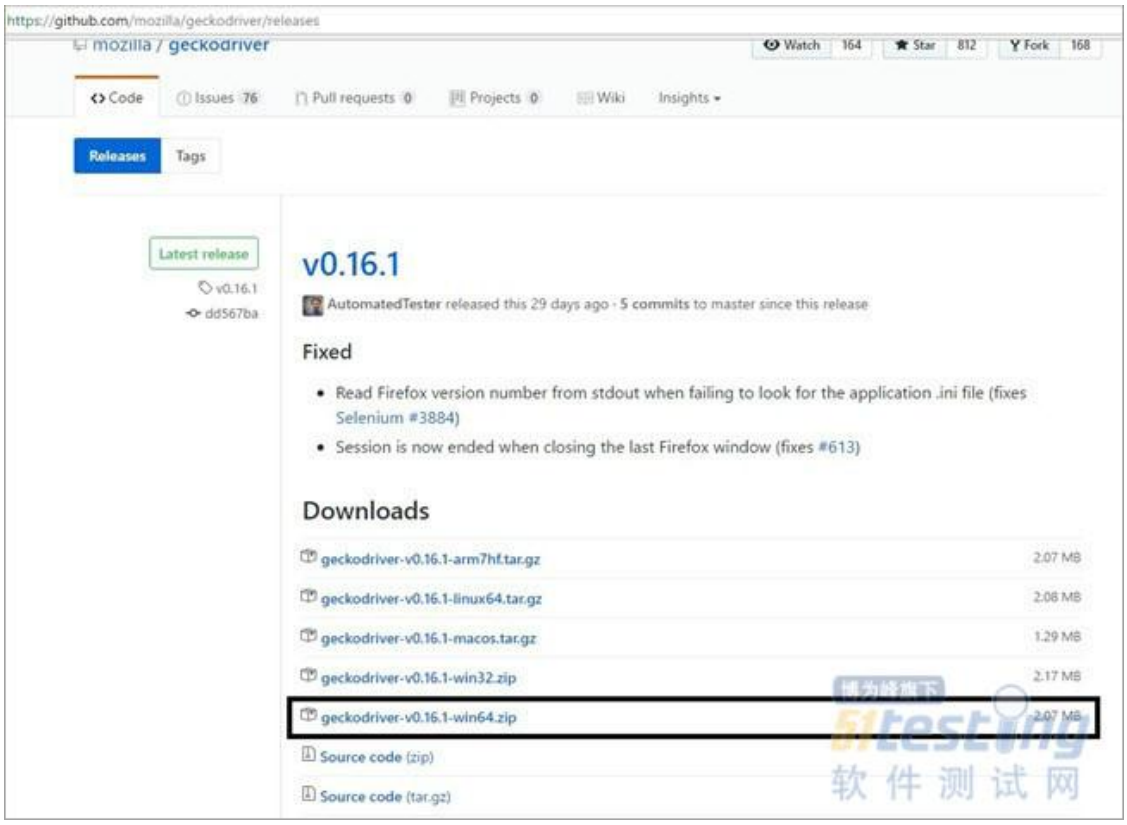
火狐 (47 版本及以上) 处于安全的考虑做了些变化，它不允许任何第三方的驱动直接和浏览器交互。因此 Selenium2 不能用于最新版本的火狐浏览器。所以要用 Selenium3。

Selenium3 有 Marionette 驱动。Selenium3 使用代理直接和火狐浏览器交互，这个代理就是 GeckoDriver。

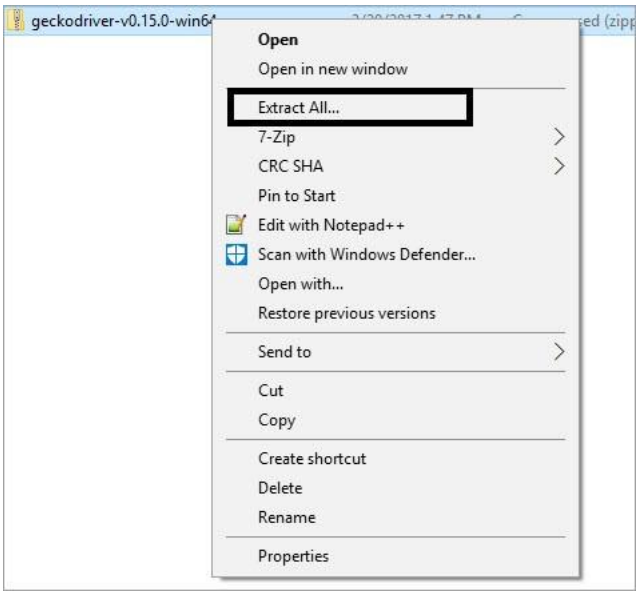
如何在 Selenium 项目中使用 GeckoDriver?

假设你已经安装了最新版本的 Selenium WebDriver 和火狐浏览器。

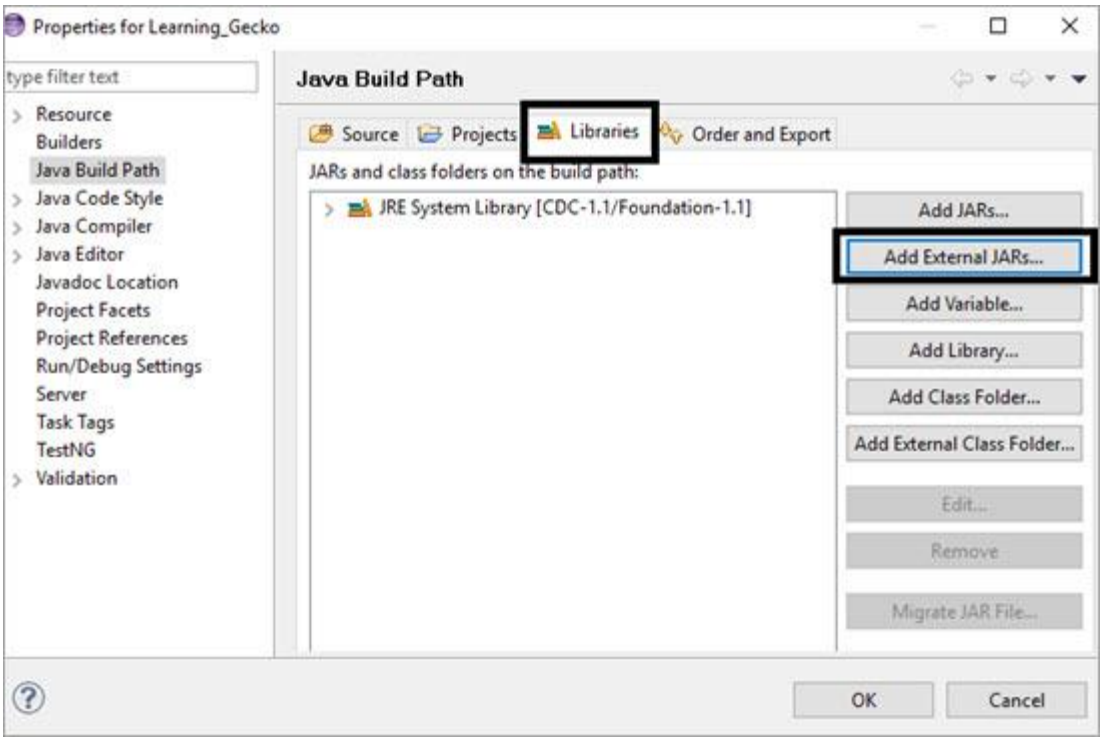
然后从这下载 GeckoDriver。最后，选择匹配你的电脑版本。



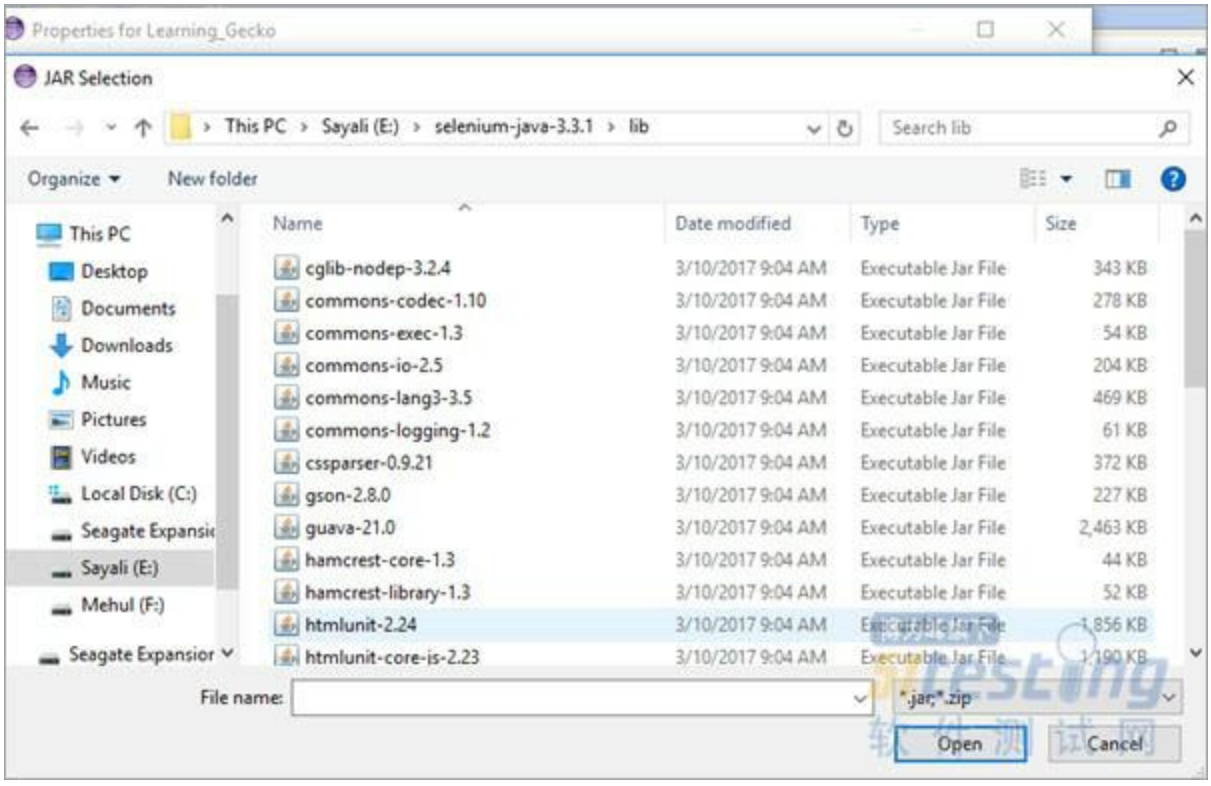
解压文件



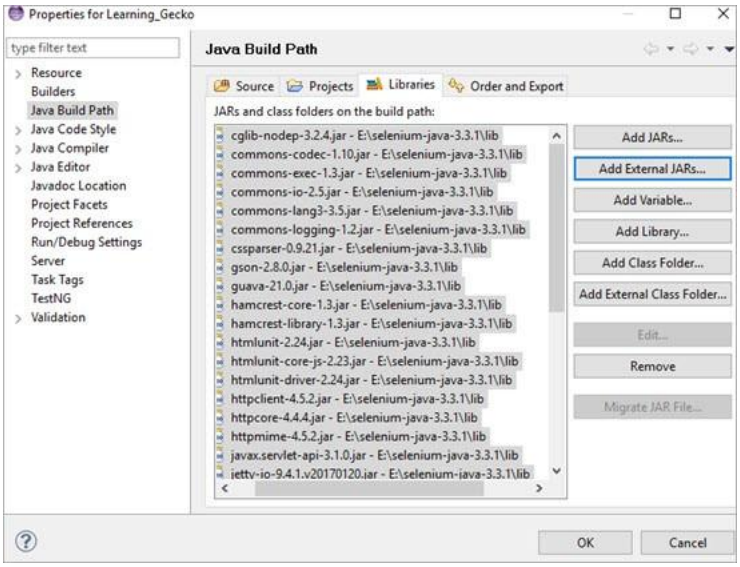
添加工程需要的 Selenium3 库-右击工程 => 建立路径 => 配置路径 => 库 => 添加 jar 包



选择 Lib 文件夹 => 点击 Clt + A =>点击打开



打开后，你会看见如下窗口:



然后点击 OK.

现在我们开始编程并且使用系统属性指定 GeckoDriver 路径

在你的代码中添加如下代码:

System.setProperty(“webdriver.gecko.driver”,”Path of the GeckoDriver file”).

让我们举个例子

举例

这是一个简单的脚本，用来使用火狐浏览器打开谷歌网页并且校验浏览器标题。

Code1:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
publicclass First_Class {

publicstaticvoid main(String[] args) {

System.setProperty("webdriver.gecko.driver","E:\\GekoDriver\\geckodriver-v0.15.0-win64\\geckodriver.exe");

WebDriver driver=new FirefoxDriver();
driver.get("https://www.google.com/");
driver.manage().window().maximize();
String appTitle=driver.getTitle();
String expTitle="Google";
if (appTitle.equals (expTitle)){
System.out.println("Verification Successfull");
}
else{
```

```
System.out.println("Verification Failed");

}
driver.close();
System.exit(0);
}
}
```

解释代码

#1) import org.openqa.selenium.WebDriver-导入所有对 WebDriver 接口的引用。然后，WebDriver 接口会实例化出一个新的浏览器。

#2) import org.openqa.selenium.firefox.FirefoxDriver-导入 FirefoxDriver 类的所有引用。

#3) setProperty(String key, String value)- 通过关键字和对应的值设置系统属性

关键字-系统属性的名字，举例：webdriver.gecko.driver.

值– Address of Gecko Driver's exe file.

#4) WebDriver driver=new FirefoxDriver() – 这行代码定义的驱动的变量 WebDriver 这个参数的值用 FirefoxDriver 的类初始化。没有扩展和插件的火狐配置文件将使用火狐实例启动。

#5) get("URL")- 用这个方法可以打开浏览器中的链接。这个 Get 方法是使用 WebDriver 的引用变量 driver 来调用的。字符串被传递给 Get 方法，这意味着我们的应用程序 URL 被传递给这个 Get 方法。

#6) manage().window().maximize()- 使用这一行代码，我们可以最大化浏览器窗口。一旦浏览器打开指定的 URL，就会使用这一行最大化它。

#7) getTitle()–使用这一行代码，我们将能够获取到网页的标题。这个方法也使用 WebDriver 的引用变量"driver"来调用。我们将这个标题保存在字符串变量 "appTitle"中。
#8) Comparison–使用 If 语句比较 appTitle(它通过 driver.getTitle()方法获得)和 expTitle(它是"谷歌")。它是一个简单的 If-else 语句。当"If" 条件满足我们输出 "校验成功" 否则我们输出 "校验失败" 。

```
if
(appTitle.equals (expTitle))
{
System.out.println ("Verification Successful");
}
else
{
System.out.println("Verification Failed");
}
```

#9) driver.close()– 这行代码用于关闭浏览器。这行代码只关闭当前窗口。

#10) System.exit(0)–这行代码用来终止 Java 程序。因此，建议在这一行之前关闭所有打开的窗口或文件。

GeckoDriver 和 TestNG

代码没有太大的差异，但是我写了一段代码供参考。

举例:

让我们看下例子。我们的例子是打开谷歌网页获取标题然后打印出标题。

Code2:

```
import org.testng.annotations.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

publicclass TstNG {

@Test
publicvoid f() {
System.setProperty("webdriver.gecko.driver","E:\\GekoDriver\\geckodriver-v0.15.0-win64\\geckodriver.exe");
WebDriver driver=new FirefoxDriver();
driver.get("https://www.google.com/");
driver.manage().window().maximize();
String appurl=driver.getTitle();
System.out.println(appurl);
driver.close();
// System.exit(0);
}
}
```

在编写 TestNG 代码时要注意的要点:

#1) 在函数中用 System.setProperty(String key, String value)方法 f() 和之前的例子一样。在那个例子里， 我们把它用在主函数了。但是，在 TestNG 中，没有主函数。如果你把它用在外部函数中会报错。

#2) 第二重要的是 System.exit(0).在你的 TestNG 脚本里没有必要加入这行代码。原因是 –TestNG 脚本运行后，在报告和结果的位置会生成输出文件夹，如果你的脚本里有 System.exit(0)这行语句，这个文件夹将不会生成，你也看不到报告了。

添加系统环境变量的步骤

- 右击我的电脑.
- 选择属性.

- 选择高级系统设置.
- 单击环境变量按钮.
- 从系统变量中选择路径.
- 单机编辑按钮.
- 单机新建按钮
- 粘贴 GeckoDriver 文件夹的路径.
- 单击 OK.

不使用 Gecko Driver 遇到的问题

你也许会遇到如下的问题

#1) 如果你用的是老版本的狐火浏览器和 Selenium3，那么你会看到下面的报错:

Exception in thread “main” Java.lang.IllegalStateException

#2) 如果你用的是最新版本的火狐浏览器和老版本的 Selenium，那么你会看到下面的报错:

org.openqa.selenium.firefox.NotConnectedException: Unable to connect to host 127.0.0.1 on port 7055 after 45000ms

#3) 如果你用的是最新版本的火狐浏览器和 WebDriver，没有用 GeckoDriver，那么你会看到下面的报错:

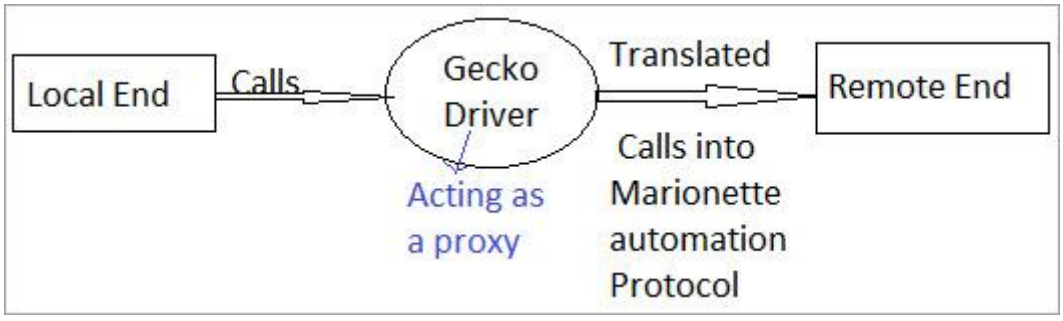
Exception in thread “main” Java.lang.IllegalStateException: The path to the driver executable must be set by the webdriver.gecko.driver system property; for more information, see here. The latest version can be downloaded from here.

附加 GeckoDriver 的信息

我们知道 GeckoDriver 是连接 gecko 核心浏览器（举例：火狐）的代理，它提供了 HTTP 的接口。

HTTP 接口可以被网页驱动协议解析，网页驱动协议有一些节点，包括本地终端、短程中孤单、中间节点和端点节点。网页驱动协议描述了这些节点之间的交互。

本地终端是网页驱动协议的客户端。远程终端是网页驱动协议的服务端，中间节点起着代理的作用。端点节点在用户使用代理或者其他类似的程序时起作用。



网页驱动发送给 GeckoDriver 的命令和响应被翻译成 Marionette 协议，然后被

GeckoDriver 传输到 Marionette 驱动。所以我们推断 GeckoDriver 是网页驱动和 Marionette 之间连接的代理。

Marionette 被分成 2 部分，分为服务端和客户端。命令有客户端发送，服务端执行。

这个命令执行工作是在浏览器中执行的。Marionette 是 gecko 原件（Marionette 服务器）和外部原件（Marionette 客户端）的结合。GeckoDriver 是 R 语言编程的。

结论

GeckoDriver 是 Selenium 脚本和基于 gecko 的浏览器(比如 Firefox)之间的中间件。

GeckoDriver 是一个代理，用于与基于 gecko 的浏览器(例如 Firefox)进行通信。Firefox(版本 47 及以上)做了一些更改，屏蔽了支持第三方驱动程序直接与浏览器交互。

这是我们需要使用 GeckoDriver 的主要原因。在脚本中使用 GeckoDriver 的最简单方法是设置系统属性。 [System.setProperty(“webdriver.gecko.driver”, ”Path of the Gecko Driver file”)].

当我们谈隐私安全时，我们在谈些什么

作者：zjyforuok

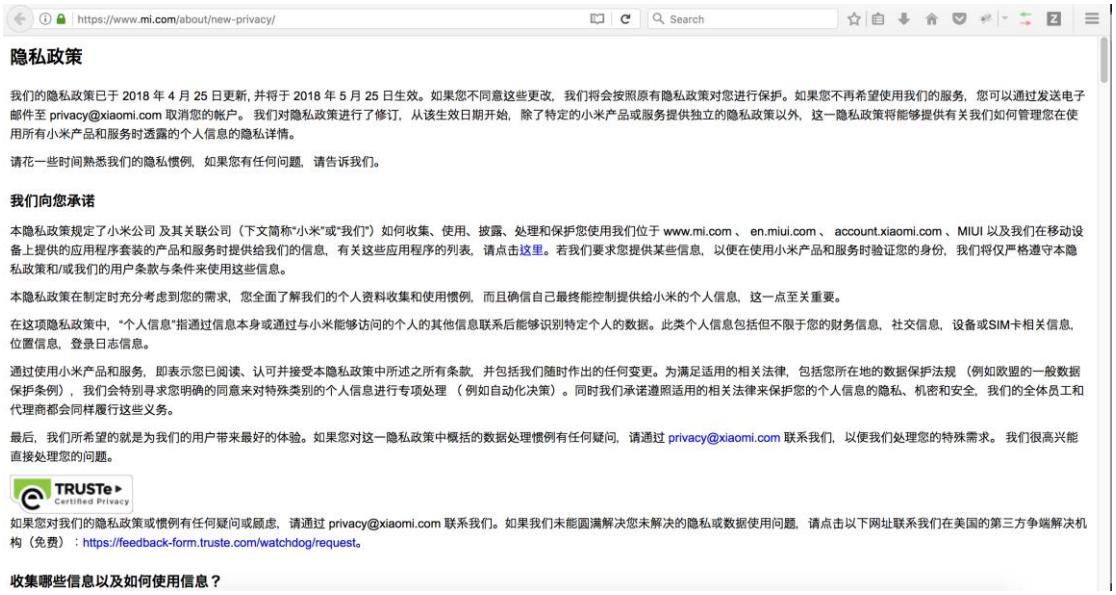
悄然而兴的隐私保护政策

不知道大家有没有意识到，最近几个月如果你下载访问一些 App，或者是注册登录某个网站—尤其是那种要搜集一些你的个人信息的应用，都会看到一个叫“隐私政策”的告知，只有在你确认“已阅读且同意该政策”后，才能使用该 App 或网站的服务。可能很多人并不会把这个隐私政策点开仔细阅读，为了使用服务，都是习惯性地在“我已阅读并同意该隐私政策”前打个勾。

在百度输入“隐私政策”，你能看到相当多的内容被检索出来。大到行业巨头、世界 500 强，小到你可能根本没听说过的某网站，都在纷纷公布自己的隐私政策，而且大多数都是在今年的 4、5 月份做了更新。下面的“隐私政策”截图来自于两个大家都非常熟悉的公司。

Apple:

小米:



隐私政策

我们的隐私政策已于 2018 年 4 月 25 日更新，并将于 2018 年 5 月 25 日生效。如果您不同意这些更改，我们将会按照原有隐私政策对您进行保护。如果您不再希望使用我们的服务，您可以通过发送电子邮件至 privacy@xiaomi.com 取消您的帐户。 我们对隐私政策进行了修订，从该生效日期开始，除了特定的小米产品或服务提供独立的隐私政策以外，这一隐私政策将能够提供有关我们如何管理您在使用所有小米产品和服务时透露的个人信息的隐私详情。

请花一些时间熟悉我们的隐私惯例，如果您有任何问题，请告诉我们。

我们向您承诺

本隐私政策规定了小米公司 及其关联公司（下文简称"小米"或"我们"）如何收集、使用、披露、处理和保护您使用我们位于 www.mi.com 、 en.miui.com 、 account.xiaomi.com 、MIUI 以及我们在移动设备上提供的应用程序套装的产品和服务时提供给我们的信息，有关这些应用程序的列表，请点击这里。若我们要求您提供某些信息，以便在使用小米产品和服务时验证您的身份，我们将仅严格遵守本隐私政策和/或我们的用户条款与条件来使用这些信息。

本隐私政策在制定时充分考虑到您的需求，您全面了解我们的个人资料收集和使用惯例，而且确信自己最终能控制提供给小米的个人信息，这一点至关重要。

在这项隐私政策中，"个人信息"指通过信息本身或通过与小米能够访问的个人的其他信息联系后能够识别特定个人的数据。此类个人信息包括但不限于您的财务信息，社交信息，设备或SIM卡相关信息，位置信息，登录日志信息。

通过使用小米产品和服务，即表示您已阅读、认可并接受本隐私政策中所述之所有条款，并包括我们随时作出的任何变更。为满足适用的相关法律，包括您所在地的数据保护法规（例如欧盟的一般数据保护条例），我们会特别寻求您明确的同意来对特殊类别的个人信息进行专项处理（例如自动化决策）。同时我们承诺遵照适用的相关法律来保护您的个人信息的隐私、机密和安全。我们的全体员工和代理商都会同样履行这些义务。

最后，我们所希望的就是为我们的用户带来最好的体验。如果您对这一隐私政策中概括的数据处理惯例有任何疑问，请通过 privacy@xiaomi.com 联系我们，以便我们处理您的特殊需求。我们很高兴能直接处理您的问题。

TRUSTe Certified Privacy

如果您对我们的隐私政策或惯例有任何疑问或顾虑，请通过 privacy@xiaomi.com 联系我们。如果我们未能圆满解决您未解决的隐私或数据使用问题，请点击以下网址联系我们在美国的第三方争端解决机构（免费）：https://feedback-form.truste.com/watchdog/request。

收集哪些信息以及如何使用信息？

如果你对这些悄然而生的“隐私政策”还无感的话，相信以下两则新闻或多或少都会有所耳闻。

在今年 3 月 27 日的中国发展高层论坛上，百度董事长兼 CEO 李彦宏表示，“中国人对隐私问题的态度更开放，也相对来说没那么敏感。如果他们可以用隐私换取便利、安全或者效率。在很多情况下，他们就愿意这么做。当然我们也要遵循一些原则，如果这个数据能让用户受益，他们又愿意给我们用，我们就会去使用它的。我想这就是我们能做什么和不能做什么的基本标准。”此言一出，引起一片质疑。为了方便，用户真的愿意用隐私为代价换取吗？到底是愿意还是无奈？李彦宏选择在 facebook 焦头烂额的时候说出这样的话，有点意外。而最让人最害怕的，不是他说错了话，而或许是他说了真心话，是科技巨头对用户核心利益的熟视无睹，成为一种脱口而出。

就在李彦宏此番言论半个月之后，4 月 13 日，QQ 连夜突然宣布：“QQ 国际版近日发出通知，宣布将从 5 月 20 日后不再为欧洲用户提供服务。”否则可能面临巨额罚款。这不得不让人联想到欧盟于今年 5 月 25 日出台的 GDPR-《通用数据保护条例》，号称史上最严的个人数据保护条例。该条例给予个人对隐私数据更多的控制权，对违规企业的罚金最高可达 2000 万欧元（约合 1.5 亿元人民币）或者其全球营业额的 4%（高者为准）。

要说《通用数据保护条例》（下文用 GDPR 代替）有多严苛，笔者深有体会。最近两年的工作主要是负责所处项目的各种安全审计和认证。从 17 年 2 季度开始，已经对 GDPR 有所耳闻。但那时对它并没有很特殊的印象，因为同时在负责着多项安全标准的评估、审核和认证，认为 GDPR 不过是其中之一而已。从 3 季度开始，根据 GDPR 的各项要求，公司出台了若干评估模版，每个项目的安全负责人要基于自己项目的具体实现情况进行评估、总结缺陷、识别风险、并制定相应的改进计划，最后要跟上一级负责人进行评审。这是做安全审计和认证的通常流程。这项工作持续了一个季度，在和上层负责人进行了若干轮评审并达成一致以后，我向项目负责人报告这项工作已经结束。就在大家皆大欢喜的时候，今年 1 月初，不同的上级部门不间断地以各种方式来审核项目在某方面是否做到了 GDPR 合规，要求之多、细节之琐碎是我们之前的评审中完全没有提及的。整个项目团队，从产品经理、到开发测试、运维都处于懵圈状态，不知道 GDPR 到底有多少要求，会持续多久以及我们要向哪个上级部门负责。这种稍稍混乱的状态持续了一个多月，随后又得知，公司进一步细化了 GDPR 的各项标准，要求所有项目组再次逐项评估，制定改进计划。并强制要求所有的改进计划必须在 5 月 25 日之前完成，否则 GDPR 不合规，产品不得在欧洲国家售卖。

什么是 GDPR

那么，下面我们来看看大名鼎鼎的 GDPR 到底是什么。

General Data Protection Regulation，简称 GDPR，中文称之为《通用数据保护条例》，是欧盟于 2018 年 5 月 25 日出台的条例。

先写几条通用数据保护条例的规定大家来感受一下。

对违法企业的罚金最高可达 2000 万欧元（约合 1.5 亿元人民币）或者其全球营业额的 4%，以高者为准。

网站经营者必须事先向客户说明会自动记录客户的搜索和购物记录，并获得用户的同意，否则按“未告知记录用户行为”作违法处理。

企业不能再使用模糊、难以理解的语言，或冗长的隐私政策来从用户处获取数据使用许可。

明文规定了用户的“被遗忘权”（right to be forgotten），即用户个人可以要求责任方删除关于自己的数据记录。

GDPR 对全球的影响

2018 年 5 月 28 日报道，Facebook 和谷歌等美国企业成为 GDPR 法案下第一批被告。具体内容大家可以自行搜索。

GDPR 在谈些什么

那么，当 GDPR 谈“隐私政策”的时候，到底在谈些什么？

在网上看一些很有趣的消息，一大批小米用户对小米的新隐私政策感到非常焦虑—不看不知道，一看吓一跳—原来小米会搜集这么多“我”的信息，还可能提供给合作商。很多人都在纠结要不要拒绝，但是拒绝就意味着不能继续使用服务。

我快速地阅读了小米的新隐私政策，发现其已经将一些 GDPR 的条例纳入其中，例如如果客户是欧盟用户（注意只是欧盟用户），则可以要求小米删除其个人数据或者是限制处理个人数据。

- 如果您是一般数据保护条例下规定的欧盟用户，您将有权要求我们删除您的个人数据。我们将会根据您的删除请求进行评估，若满足一般数据保护条例规定，我们将会采取包括技术手段在内的相应步骤进行处理。
- 如果您是一般数据保护条例下规定的欧盟用户，您将有权要求我们限制处理您的个人数据。我们将会根据您的限制请求进行评估，若满足一般数据保护条例规定，我们将会根据条例中适用的具体情况处理您的数据，并在取消限制处理前通知您。

作为非欧盟用户的米粉们，不知道有没有感受到一丝丝伤害。因为对他们来说这个政策看起来更像是一个隐私告知政策，而不是隐私保护政策。当然相比之前有进步，因为更加透明。之前或许也搜集了大量的用户个人数据，但是却没有进行告知。

相比这种告知性的政策，GDPR 的设计意图是保护个人信息，并赋予公民对信息更大的控制权，从而对企业作出限制和规定。

下面我们就来一起了解一下这个史上最严的个人数据保护政策都谈了些什么：

GDPR 的主要目标:

协调欧盟成员国间的数据保护法规；GDPR 作为条例可直接适用于所有欧盟成员国

- 保护欧盟公民的基本权利和自由
- 赋予数据主体对其个人信息的充分控制权
- 通过对政策和程序的重点关注加强合规力度
- 增加对薄弱措施和薄弱安全性的曝光
- 提高对安全数据流的关注度
- 提出罚款金额更高的新处罚制度

我们先来了解一些术语，在后文中会反复提到:

个人信息：简单来说，是指可用于直接或间接识别个人身份的任何信息。例如你的电话号码，家庭住址，身份证号，医疗记录等等。

数据主体：信息相关之个人

数据控制者：自然人或法人、公共机关或其他决定个人信息处理目的和手段的实体。

数据处理者：自然人或法人、公共机关或其他代表数据控制者处理个人信息的实体。

注意：数据处理者不决定个人信息的处理目的或方式。他们仅可按照数据控制者决定的方式处理个人信息。

收集和处理个人信息

无论何时，收集个人信息或计划将个人信息进一步用于其他目的，都必须向数据主体提供隐私声明。

企业必须出于正当的合法目的处理个人信息。根据 GDPR，处理个人信息有六种适用的法律依据。

哪种法律依据最为适用要取决于处理目的以及与数据主体的关系。在开始处理个人信息前，必须确定适用的法律依据。

第一依据 同意

数据主体同意出于特定目的处理个人信息。同意若要有效，必须:

自愿给予：数据主体不得迫于压力而同意。他们必须能够随时拒绝或撤销同意。如

果同意是作为某项服务的前提条件，则同意不属于自愿给予（这大概就是国内的很多隐私政策被称为霸王条款的原因吧，很多条款用户都是被迫同意）。

知情：应当以清晰简明的语言向数据主体说明其个人信息的用途及分享对象。

明确：必须通过声明或明确的肯定行动而予以同意，例如勾选选框。沉默、事先勾选选框（让我想起了支付宝看账单，默认勾选“我同意《芝麻服务协议》”一事）、未操作或未选择退出均不能构成有效同意。

具体：对于具有多重目的的处理活动，同意的表示应当提供细化选项。

第二依据 合约之必要性

处理若要获得允许，则其应为履行与数据主体之合同所必需，或者在签署合同前，需要应数据主体之要求采取措施。例如，达成就业协议需要处理个人信息（如工资单信息）。为正确处理个人信息，则无需经过员工同意。

第三依据 法律义务

对于数据控制者而言该处理是为遵守法律义务（不包括合同义务）所必要的。

第四依据 主要利益

该处理对于保护某人生命而言是必须的。

第五依据 公共利益

该处理对于数据控制者而言是必要的，以便其执行与公众利益有关的任务或履行官方职能。该任务或职能必须有明确的法律依据。

第六依据 合法利益

该处理对于数据控制者或第三方的合法利益而言是必要的。当数据主体的利益、基本权利和自由（要求保护其个人信息）高于合法利益时，则合法利益的法律依据将无法适用。

明确数据控制者或第三方的合法利益，并权衡其与数据主体的利益，这一点至关重要。

数据主体的权利

数据主体拥有以下权利：访问、修改、删除、拒绝、处理限制、可移植性以及不得

完全根据自动处理的结果作出决策。下面我们逐一解读解读各项权利的意义。

访问

根据 GDPR，数据主体有权要求确认数据控制者是否正在处理其个人信息，并且在任何情况下有权在对方收到要求后的一个月内获得这些信息，且不得有任何的不当延误。

修改

数据主体有权更改不准确的个人信息。

删除

若没有强制理由继续处理，数据主体有权要求删除或移除其个人信息。这种权利意味着要以无法恢复的安全方式删除个人信息。

拒绝

数据主体有权拒绝:

出于合法利益、为公共利益执行任务或行驶官方职权而处理个人信息，包括特征分析

直接营销，包括特征分析

出于科学或历史研究和统计

当数据主体行使此项权利时，除非数据控制者能证明其出于合法强制立场进行处理，并且其立场高于数据主体的利益，否则处理个人信息将是非法行为，且相关个人信息必须予以删除。

处理限制

某些情况下，数据主体有权阻止或中止处理其个人信息。当个人信息的处理受到限制时，允许数据控制者储存个人信息，但是不得在将来做进一步处理。

可移植性

数据主体有权获得一份个人信息的副本，该副本需要以常用的机器可阅读的方式提供。

不得完全根据自动处理的结果作出决策

数据主体有权不遵守完全基于自动化处理（包括特征分析）作出的决策，这些决策可能对其产生负面法律影响，或以类似方式对其产生重大影响。

数据控制者仅能在下列情况下执行完全自动化且产生法律或重大影响的决策，即该决策必须:

- 出于数据控制者和数据主体之间订立或履行合同的必要性;
- 法律授权或基于数据主体的明确同意。
- 数据控制者与数据处理者的义务
- 数据控制者的义务

指定数据处理者- 指定一个数据处理者时，数据控制者必须确保持有适当的书面数据处理协议，确保处理实施者是满足 GDPR 适当要求的特定数据处理者。

数据主体的权利 – 数据控制者必须确保和促成根据 GDPR 数据主体可行使的权利。他们也有义务及时回应来自数据主体的要求，不得有任何不当延误且最晚在收到要求后一个月内。

通知 – 数据控制者必须在 72 小时内通知数据保护机构数据泄漏的消息，在某些情况下还需通知数据主体。数据控制者还必须持有任何个人信息泄漏的记录，无论是否必须对其进行通知。

设计的隐私和默认的隐私 – 根据 GDPR，数据控制者具有实施技术和组织措施的普遍义务，体现出已对个人信息保护进行考虑并将其融入处理活动中，而不是一件后知后觉的事情。

数据隐私影响评估（DPIA）- 当一项新的数据处理活动很可能将数据主体置于高风险境地时，数据控制者必须开展数据隐私影响评估（DPIA）。数据控制者借助 DPIA 可识别并缓解他们可能尚未意识到的风险。

数据处理者的义务

遵从数据控制者指示 – 数据处理者需要根据数据控制者的书面指示处理个人信息。不得将个人信息用于数据控制者要求以外的用途。

通知数据控制者之法律义务 – 如果数据处理者认为数据控制者的指示与 GDPR 要求或其他欧盟国家法律冲突，数据处理者必须立即通知数据控制者。也必须通知数据控

制者任何数据泄漏的信息，且不得有任何延误。

指定分包处理者需获批准 – 未获得数据控制者的事先书面同意，数据处理者不得指定分包处理者（即分包商）。获得特别授权时，分包处理者必须遵照数据控制者和数据处理者协议中规定的相同条款。

配合数据保护机构 – 在数据保护机构执行任务时，数据控制者与数据处理者均需应要求予以配合。

实施安全措施 – 数据控制者和数据处理者必须实施适当的技术、程序和组织措施，以保护个人信息免受意外或非法的破坏、损失、更改、未授权泄漏或访问。

留存记录 – 在 GDPR 中包含有关文档记录的严格要求，这也是数据控制者与数据处理者的一项共同义务。一经要求，必须向数据保护机构披露这些记录。

传输个人信息

欧盟在其公民个人信息向欧盟以外地区传输方面，有着严格规定。任何时候一家公司跨境传输欧盟公民的个人信息，均需了解跨境数据传输法规。

欧盟公民的个人信息只能传输（未进行任何额外保障措施）到欧盟认为其数据保护法规完备的国家/地区。

在当今的全球市场环境中，公司依赖于跨境分享和传输个人信息。GDPR 试图要求通过额外的保障措施，以在欧盟和隐私法规相对宽松的其他国家（如美国）保护欧盟公民的个人信息。

有多种途径可用于将个人信息传输到数据保护法规不完备的国家/地区。

一、示范条款

获欧洲委员会批准及发布的合同条款，即所谓的示范条款，是在欧盟以外地区进行数据传输最常用的途径。

二、企业约束规定

企业约束规定作为跨境转移途径，允许跨国公司以遵循欧洲数据保护法规定的形式，从欧洲经济区 (EEA) 向其设立于经济区外的关联公司转移个人信息。

违规和处罚

你可能听说过一些数据泄漏事件，还有一些没有报道，让我们一起来看看。

2007 年，英国政府被迫承认 2500 万的个人记录丢失，这些信息包括个人的银行帐号和国家保险号。数据是在邮递过程中丢失的。

2015 年，1.91 亿美国人的选民登记记录通过互联网发布。这些信息包括选民的政治面貌和投票记录。网民无需密码或身份验证就可以访问这些记录。而这次泄漏仅仅是由于数据库配置不当导致的。

根据 GDPR，数据隐私的不当处理将受到极为严重的制裁。

轻度违规会导致多达 1000 万欧或 2% 全球总营业额的罚款（以较高者为准）。

重度违规会导致多达 2000 万欧或 4% 全球总营业额的罚款（以较高者为准）。

以上就是这个号称史上最严的隐私数据保护条规的主要内容，在今年 5 月 25 日强制执行。很多跨国企业，如果还想继续做欧盟国家的生意，就必需遵守这个条规的要求。在笔者所在的公司，为了合规，很多投放市场多年的成熟产品及相关流程都进行了显著的改动。

对于信息主体（拥有隐私数据的我们），当然是非常乐见于这样的规范提出并生效。而作为测试人员的我们，又多了一项需要掌握的技能。个人认为，对产品的安全性以及隐私保护的验证会发展成为测试工作的一个新的分类。对此有兴趣的同行们，现在就开始你的学习和积累吧！

《51 测试天地》（五十一）下篇 精彩预览

马上阅读