目 录 (五十二期·下)

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“原始人遮羞布及石器篇”

◆ 作者：王平平

上一期我们终于到了原始人阶段，虽然还是赤身裸体，以及还没有捕猎的工具，但是最起码我们可以称之为“人”，为什么呢因为我们有了自动化和手动测试结合的报告，我们的脚本有了“班长”，再也不是一盘散沙了，而且我们还有了基于 unittest 模块的漂亮的报告了!! 而从这一期开始我们疯狂原始人终于有了工具，以及有了遮羞布 ^_^，我们的第一代工具就是：石器!

在进入介绍石器之前，我们还是先来总结一下近期工作的心路历程，还记得 2018 年 1 月 1 号微信上有一个用手机号码来预测 2018 的小程序：我的预测结果是“血拼的一年”，转眼 2018 年就要过去了，这一期也是 51Testing 杂志 2018 年的最后一期，回首 2018 年对于自己真的是血拼的一年：做了一个自动化工具：需要控制三个之前都没听说过的仪器，其中一个是价值几百万的机械手臂=_=，且开发时间相当的紧张；每天都在惴惴不安，一个月期间几乎没有在夜里 12 点前睡过觉，经常干到凌晨一两点，第二天 6 点多起床*_*；五一假期期间还在每天看资料，想逻辑；还好，还好，一切虽然真的很不容易但是真的还算顺利：一个非常重要的原因哈哈有一个心脏很大颗的超人协助；虽然期间愤怒过，绝望过，也同时温暖着，满足着，开心着；罗胖的“得到”的吴伯凡老师说过：“痛快”即“痛”过后再“快乐”才是真正的快乐，越来越觉得这种“痛快”上瘾了，这样才会真的觉得自己是在活着；当然今年也升级了我们的自动化框架，期间也是“痛快”感觉十足，甚至哭过，骂过，争执过，但是同时也是满足着，开心着；非常开心，自己可以真的突破自己，真的可以透过现象看问题的本质，真的现在越来越可以控制自己的情绪，最起码在情绪上来的时候，我可以暂时用冷漠来控制它，让我在盛怒下，还可以正常思考：冷漠，有时比失控抓狂真的好太多，哈哈真的要到四十岁才能体会到中年人的淡然，虽然可以进步的空间还是很大。11 月底还非常荣幸代表南京团队

到总部汇报这一年的成果，这个过程也是“痛快”十足，PPT 如何写，汇报如何在 15 分钟之内抓住老板们的注意力，以及说服老板，真的非常之“痛快”！2018 年，真的配的上年初的预言：血拼的一年，每走一步都是一个血印*_*；不过 2018 年真的能实实在在感受到活着的一年，真的是收获满满！

好吧，心路历程总结完毕，我们真的可以开始介绍我们的工具，我们的工具是 selenium:这个是测试网页的万能神奇！

1、我们先看一下 selenium 是什么？

1）它主要提供了网页自动化测试的解决方案，而且是开源的，很多公司都在使用，它的官网：http://docs.seleniumhq.org/

2、Selenium 主要的作用：

1）通过自动化的方式测试 Web 中包含的一些应用

2）基于 Web 的一些重复的令人厌烦的管理任务也可以通过它实现自动化测试。

3）根据自己的需求进行一些客制化的开发。

4）它包含一套工具使 web browser 可以实现自动化测试，这个测试是可以跨平台的测试

5）它可以被很多语言控制

3、Selenium 支持的浏览器

Selenium 与一些比较大的 browser 供应商是兼容的，这些 browser 的供应商已经设置了几个步骤目的是使 Selenium 作为一个 browser 本地部分。

对比其它的浏览器自动化测试工具和 APIs 和 frameworks 它是一个核心的技术。

- 它支持的 Browser 如下：Firefox, IE, Safari, Opera, Chrome
- 它支持的语言如下：C# Java Perl PHP Python Ruby Others

Selenium 有一系列的工具可供选择，例如 selenium IDE 进行录制脚本我们这次主要用的是 Selenium Web Driver 它可以控制一个浏览器或者本地或远程控制浏览器。

4、Selenium-Web driver API 命令和操作

下面介绍一下经常使用的一些 API

1）读取页面

第一件事就是你希望使用 web driver 来访问一个网页，通常的方法是”get”即：driver. get(“http://www.google.com”)

为了避免在页面还未加载完毕就进行某些操作，可以使用 Explicit 或者 Implicit 函数进行等待，等页面完全加载完毕，再进行一些动作的进行。

2）定位 Web UI 界面的元素(即 Web Elements)

每个绑定中的语言都会有”Find Element”和”Find Elements”的方法。第一个方法返回的 WebElement 对象，否则它会丢出一个异常。后面一个方法返回的是 WebElements 的列表，如果没有匹配的列表，见返回一个空的列表 Find 的方法获得一个定位器，或者询问的对象，在 web driver 中称 find 为 By 它的方法如下:

By ID

这是一个最有效率和比较好的定位元素的方法，比较平常的问题是，元素 ID 的名称不唯一，对于此 Page 中或者自动产生的 ID，此两种情况都被避免。建立一个在 html 元素的类，是与自动产生 ID 对比而言是更适合的方法。

```
element = driver.find_element_by_id("coolestWidgetEvah")
or
from selenium.webdriver.common.by import by
element=driver.find_element(by=By.ID,value="coolestWidgetEvah”)
```

By Class Name

这里的 Class 指的是 DOM 元素的属性。通常状况下很多 DOM 元素有相同的 Class name 因此发现多重的元素变得更有实际的意义，相比发现第一个 element 而言。

```
cheeses = driver.find_elements_by_class_name ("cheese")
or
from selenium.webdriver.common.by import by
cheeses = driver.find_elements(By.CLASS_NAME, "cheese")
```

By Tag Name

返回的是这个元素的 DOM 标签姓名。

```
frame = driver.find_element_by_tag_name("iframe")
```

```
or
from selenium.webdriver.common.by import By
frame = driver.find_element(By.TAG_NAME, "iframe")
```

By Name

找到符合 name 属性的元素

```
cheese = driver.find_element_by_name("cheese")
or
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.NAME, "cheese")
```

By Link Text

通过匹配可见的文本来找到 Link 元素

```
cheese = driver.find_element_by_link_text("cheese")
or
from selenium.webdriver.common.by import by
cheese = driver.find_element(By.LINK_TEXT, "cheese")
```

By Partial Link Text

通过部分匹配可见文本来找到 link 元素

```
cheese = driver.find_element_by_partial_link_text ("cheese")
or
from selenium.webdriver.common.by import by
cheese = driver.find_element(By.PARTIAL_LINK_TEXT, "cheese"
```

By CSS

连接姓名使用的定位策略是 CSS，如果 Browser 不支持 CSS 则会使用 Sizzle.IE6，IE7 和 FF3.0 目前使用 Sizzle。

注意不是所有的 Browser 都支持 CSS，即使是同一个 Browser 不同的版本也有可能某个版本支持 CSS，某个版本不支持 CSS。

```
cheese = driver.find_element_by_css_selector ("#food span.dairy.aged")
or
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.CSS_SELECTOR, "#food span.dairy.aged")
```

By XPATH

在高层只要有可能 Web Driver 就会使用 Browser 本地的 XPath 能力。在那些本地没有 Xpath 支持的 Browser，我们已经提供了我们自己的执行。这些可能导致一些不期望出现的行为出现。除非你已经了解不同的 Xpath 引擎有什么不同。

此函数较复杂不建议使用它。只有在实在没有办法了再考虑用这个函数

5、掌握也上述方法我们终于可以我们以 Router 的一个 smoke test plan 为例来说明 Selenium 的好处

1）对于 Router 测试，或者不管什么产品的测试都会有“冒烟测试”，即紧急 check 一下基本功能是否正常，这样的测试是非常简单的测试，也是让测试人非常反感的测试，不断的重复，不是以发现 bug 为目的，只是以确认功能是否正常为目的，经常在决定是否投入很多人力进行测试或者 release 给客户前做一个简单的 check

2）我们先开看一下 Smoke 测试的 test case

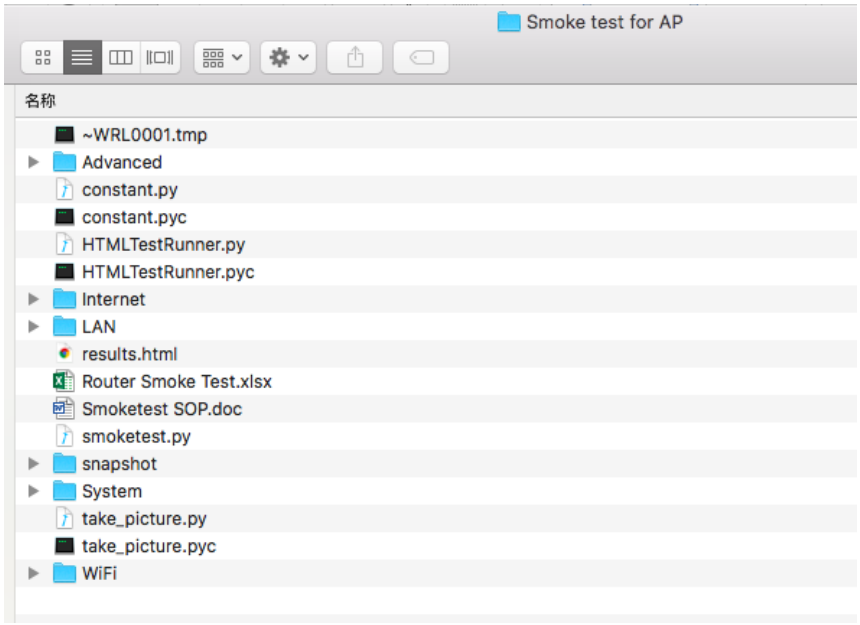
Testing information	
Tester :	
Test Date :	
Version :	

Total Amount	89
Pass	0
Fail	0
NA	0
NT	0

Router Smoke Test

TestItem Name	Test Step		Test Result	Tester	Notes
Internet					
WIFI					
LAN					
Advanced					
System					

3）看一下我们的 some test for AP 这个自动化测试工具的架构，其中的文件夹除了 Snapshot 其它的文件夹都是与 TestItem Name 一一对应的：



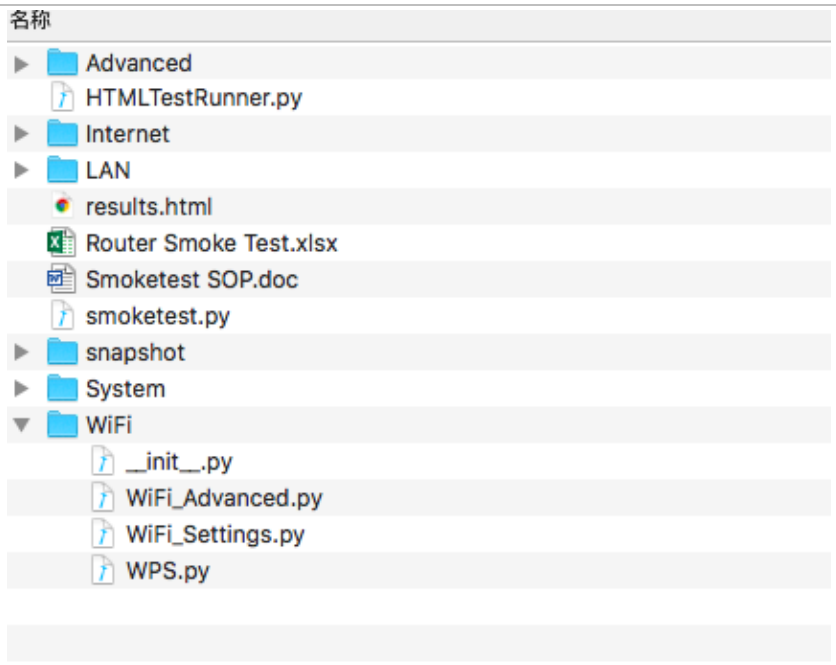
4）我们随便挑一个 Test Item 看一下代码的如何写的：

我们挑一个 WIFI Function

a. Test case 的架构很简单如下

A	B	C	D	E
TestItem Name	**Test Step**		**Test Result**	**Tester**
Internet				
WIFI	Wi-Fi Settings	SSID Enable		
		SSID Disable		
		Add a SSID Schedule rule		
		Edit the SSID Schedule rule		
		ACL_Allow everybody		
		ACL_Allow		
		ACL_Deny		
	WPS	Enable WPS		
	Wi-Fi Advanced	Mode:ng		
		Channel Spacing:20/40MHz		
		Control Side Band:Upper		
		Channel:6-2.437GHz		
		Transmit Power:50%		

b. 测试脚本的架构，与 Test case 的架构是一一对应的



c. 我们看一下 WIFI_Settings.py 中是如何组成的：

一个函数对应一个 test item 如下：

欢迎使用 WiFi_Settings.py ×

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.ui import Select
import time,take_picture,constant
testcase=__name__.encode('utf-8')

Host1_name=constant.get_Host1_name()
Host1_address=constant.get_Host1_address()
Host1_MAC_address=constant.get_Host1_MAC_address()
Host2_name=constant.get_Host2_name()
Host2_address=constant.get_Host2_address()
Host2_MAC_address=constant.get_Host2_MAC_address()
Enable_wifi_SSID=constant.get_Enable_wifi_SSID()
Enable_wifi_password=constant.get_Enable_wifi_password()
 #Disable SSID,and PC can't connect to DUT via wifi
def Edit_Disable(self,driver):
     #Disable SSID and set some fields
    driver.find_element_by_id("wifi_menu_img").click()
    driver.find_element_by_id("wifi_settings_href").click()
    driver.find_element_by_id("wifi_ck_1").click()
    driver.find_element_by_id("edit_wifi").click()
    driver.find_element_by_id("SSIDenable").click()
    driver.find_element_by_id("ssid1").clear()
    driver.find_element_by_id("ssid1").send_keys("WNC博为峰旗下")
    driver.find_element_by_id("bssid_en1").click()
    Select(driver.find_element_by_id ("enc_idx1")).select_by_index(3)
    driver.find_element_by_id("pskascii1").clear()
    driver.find_element_by_id("pskascii1").send_keys("123456789")
```

A	B	C	D	E
TestItem Name	Test Step		Test Result	Tester
Internet				
WIFI	Wi-Fi Settings	SSID Enable		
		SSID Disable		
		Add a SSID Schedule rule		
		Edit the SSID Schedule rule		
		ACL_Allow everybody		
		ACL_Allow		
		ACL_Deny		
	WPS	Enable WPS		
	Wi-Fi Advanced	Mode:ng		
		Channel Spacing:20/40MHz		
		Control Side Band:Upper		
		Channel:6-2.437GHz		
		Transmit Power:50%		

5）我们有一个班长，这个与上一期的班长并没有什么不同，这里就不再赘述：

欢迎使用 smoketest.py

```
# coding=utf-8
import HTMLTestRunner,unittest,time,sys,os
from selenium import webdriver,selenium
from Internet import Status,PIN_Configuration,Radio_Configuration,Profile_Configuration,Network_Scan
from WiFi import WiFi_Settings,WPS,WiFi_Advanced
from LAN import LAN_Settings,DHCP
from Advanced import DNS_DynDNS,Firewall,QoS_Settings,Port_Forwarding,NAT,Static_Routing,DLNA
from System import Time_Settings,Users,Remotemanagement,Systemlogs,Reboot_Reset,FW_Upgrade

sys.path.append(os.path.dirname(__file__))
driver = webdriver.Firefox()
driver.implicitly_wait(30)
driver.get("http://192.168.1.1/")
driver.find_element_by_id("login_username").clear()
driver.find_element_by_id("login_username").send_keys("admin")
driver.find_element_by_id("login_password").clear()
driver.find_element_by_id("login_password").send_keys("admin")
driver.find_element_by_id("login").click()

#执行测试的类
class test_smoke_test(unittest.TestCase):
    def setUp(self):
        pass

    ################################################################################
    #no1.Internet Function
     #Status
```

6、抓图功能

为什么需要截图呢，大家想象一下，当你下班前把自动化工具执行起来，万一在执行过程中遇到问题，我们希望看当时的界面出了什么问题，最理想的方法是什么？当然把当时的情况拍照下来，所以才会需要用到这个功能

1) 运用等待函数说明 selenium 屏幕截图的时机

问题描述：

在什么时候进行截图，按照之前的经验都是采用 sleep 的做法，如：time.sleep（3），但是这个 sleep 的时间就不太好把握，如果时间太短，则没有达到延时的效果，时间太长，则又浪费时间，如果能智能的等待页面元素加载完成再进行截图，那是最好的了

解决办法：

调用 webDriver 中的 WebDriverWait 类可以起到智能等待的效果。

他有两个函数 until 及 until_not

- until：看到什么内容的情况下做什么事情
- until-not：看不到什么内容的情况下做什么事情

一般情况下，我们调用其中的函数可以解决现有的问题

遇到的困难：

如果是同一界面设置 apply，发现截图的时间不是我们要的，查找原因，发现有些路由器（D57）在设置完成 apply 的过程中，界面的元素是就可以定位到的，所以通过 until 函数等待所要定位的元素出现没有达到我们的目的

使用 until_not 函数

也就是说，我们在看不到什么内容的情况下做什么事情。

在加载的过程中，我们可以得到一个 xpath，那么问题就迎刃而解了。

```
wait=WebDriverWait(driver,20)
wait.until_not(lambda driver :driver.find_element_by_xpath("/html/body/div[3]/img").is_displayed())
```

注意：测试模块中导入此类

```
from selenium.webdriver.support.ui import WebDriverWait
```

紫红色标记的内容是超时的时间

黄色标记的内容为加载的 xpath

绿色比较的内容为等待定位的元素干什么

整句话的意思就是等待定位的元素消失

此时如果加上截图的语句，就是在加载消失后开始截图

使用 unti 函数

只有切换不同页面的时候，我们可以使用 until 函数。

```
driver.find_element_by_id("top_menu_wwan_img").click()
WebDriverWait(driver,200).until(lambda driver :
   driver.find_element_by_id("data_roaming").is_displayed())
```

2) 屏幕截图的运用

a. 屏幕截图的方法:（2 种）

```
from selenium import webdriver
driver = webdriver.Firefox()
```

#截图保存在 D 盘根目录，名字为 pic1

方法 1，driver.save_screenshot("D:\pic1.png")

#截图保存在 D 盘根目录，名字为 pic2

方法 2，driver.get_screenshot_as_file('D:\pic2.png')

b. unittest 中屏幕截图的运用

在 Auto test for AP 文件夹中添加截图的脚本（假设命名为 take_picture.py）脚本中定义屏幕截图的函数，并规定好图片存放的路径和图片的名称

```
import os
current_dir='E:/Auto test for AP/'

def snapshot(driver,testcase,filename):
    ssDri=current_dir+'snapshot/'+"/" + testcase
    ssFile=ssDri+'/'+filename+'.png'
    try:
        os.makedirs(ssDri)
    except OSError:
        pass
    driver.save_screenshot(ssFile)
```

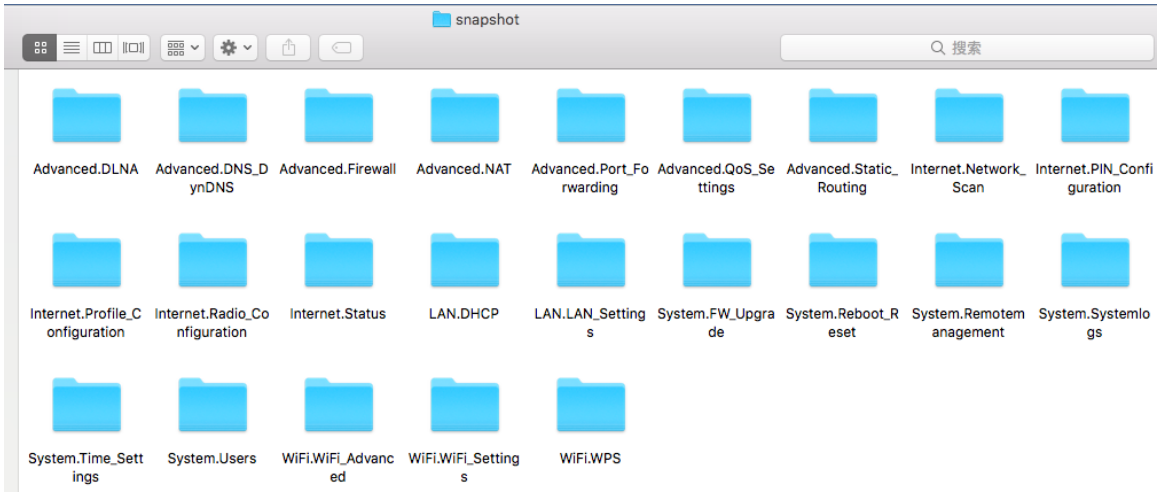

测试模块脚本中添加调用屏幕截图函数的语句

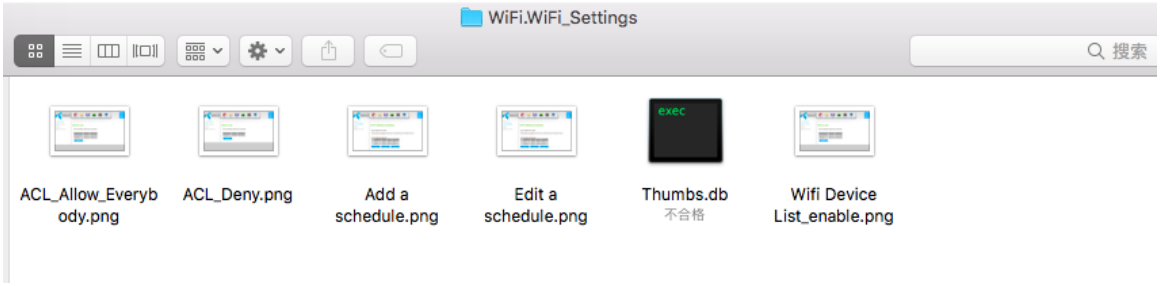
```
take_picture.snapshot(driver,testcase,'wifi_mode')
```

注意：橘黄色标记的为图片名称 测试模块脚本别忘记导入截图脚本 import take_picture

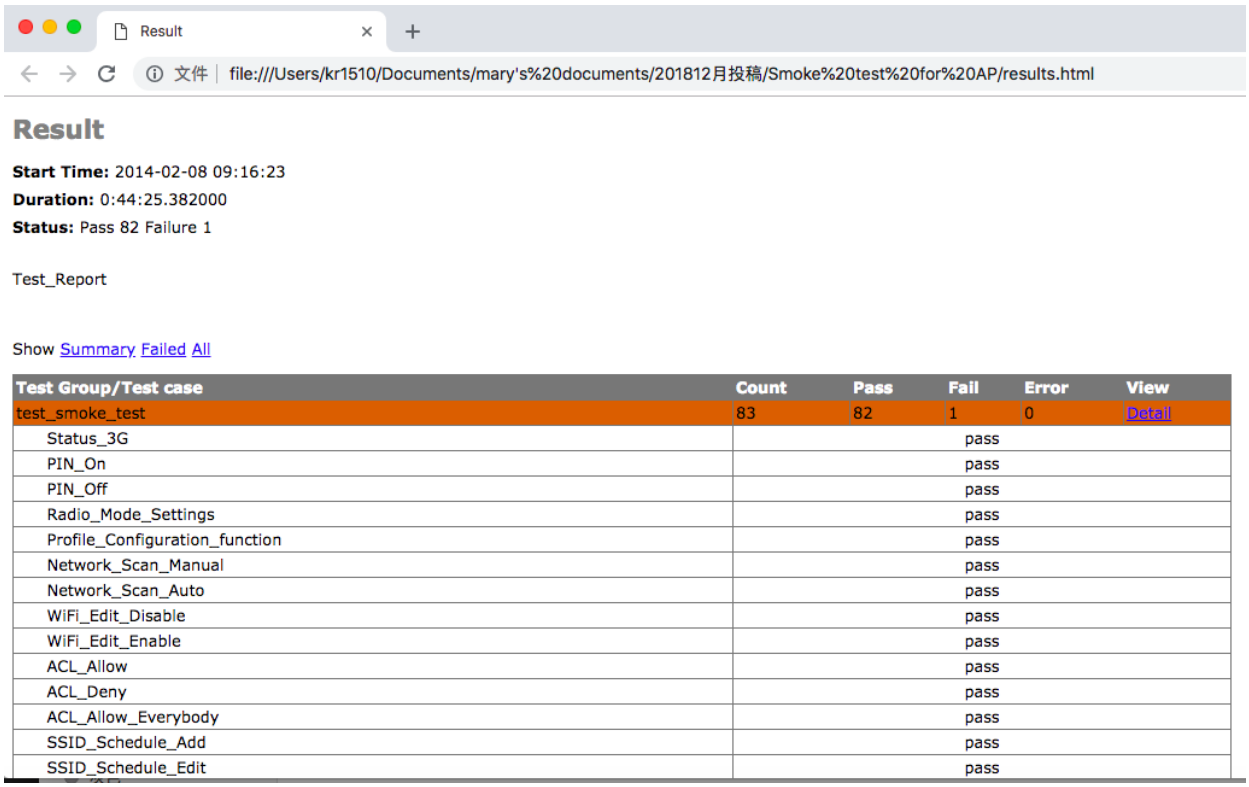
3) 看一下这段代码的结果:

多么惊人的结果，每个 Test Item 都有当时测试的情况的截图：简直是无人值守的经典用法！





看看我们最后的测试结果：是不是很酷，既有测试结果，又有每个 Test Item 的测试截图，简直完美！



Test Group/Test case	Count	Pass	Fail	Error	View
test_smoke_test	83	82	1	0	Detail
Status_3G			pass		
PIN_On			pass		
PIN_Off			pass		
Radio_Mode_Settings			pass		
Profile_Configuration_function			pass		
Network_Scan_Manual			pass		
Network_Scan_Auto			pass		
WiFi_Edit_Disable			pass		
WiFi_Edit_Enable			pass		
ACL_Allow			pass		
ACL_Deny			pass		
ACL_Allow_Everybody			pass		
SSID_Schedule_Add			pass		
SSID_Schedule_Edit			pass		

这次总结对得起那块遮羞布吧，这下不会有人问我搞什么搞了，哈哈我们搞定了只要黑盒测试的对象是网页的产品，就可以实现自动化测试，我们在上一次的基础上增加了 selenium 以及截图功能，以及也有了融入自动化测试的 test plan，想象一下：针对这种一直重复做的工作，都可以自动化执行，是不是很酷，而且自动产生漂亮的测试报告，以及将当时测试的界面截图下来，只是针对脚本类的项目，是不是已经足够了？哈哈没有啦，差得还很远，所以还是那句话：路漫漫其修远兮，吾将上下而求索……

浅谈自动化测试的职业发展

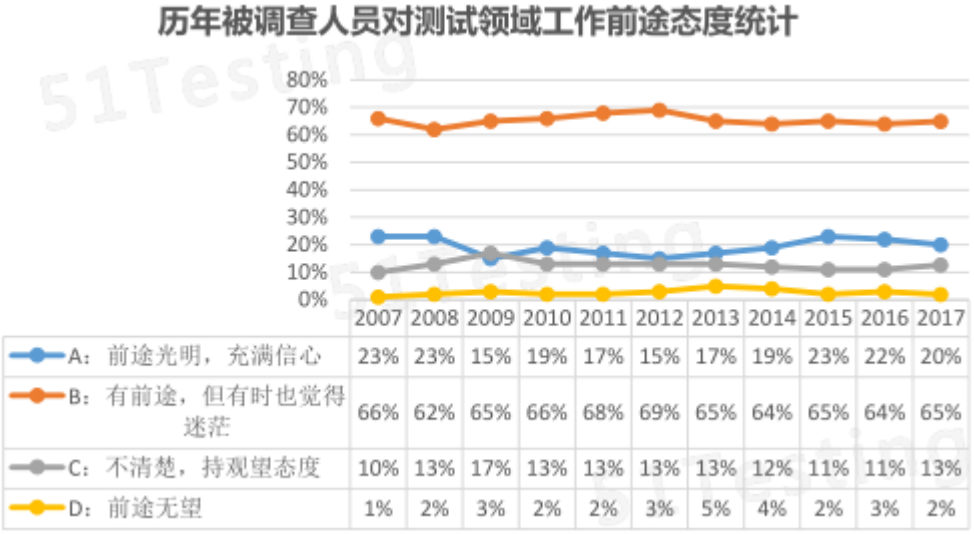
◆ 作者：Chloe Zhang

摘要：本文主要从自动化测试从业者的职业方向以及职业规划两大方面展开论述自动化从业者的职业发展。从多个角度出发分析不同职业方向的各种优缺点及应具备技能，详细讨论职业规划应该考虑的因素。对自动化从业者的职业发展及择业具有重要的参考意义。

背景

随着敏捷开发在软件行业的推广与发展，自动化测试在软件测试行业的地位也在逐渐上升。然而，对于自动化从业者将何去何从？自动化从业者的发展方向有哪些？身为自动化测试工程师，我们又该如何为自己定制职业发展规划？这些问题一直困扰着我们，众多自动化从业者都是在迷茫中前进。由此可见，做好职业规划是自动化测试从业者选择发展之路，明确自身的成长空间，选择就业方向的风向标。

根据 51Testing 发布的测试人员对测试领域工作前途态度的统计结果（图一）显示，大于 60%的从业人员对软件测试行业的发展前景感到迷茫，10%以上的不清楚甚至感觉前途无望。由此可见，自动化测试的职业发展对于自动化从业者择业与从业具有现实意义。



	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
A：前途光明，充满信心	23%	23%	15%	19%	17%	15%	17%	19%	23%	22%	20%
B：有前途，但有时也觉得迷茫	66%	62%	65%	66%	68%	69%	65%	64%	65%	64%	65%
C：不清楚，持观望态度	10%	13%	17%	13%	13%	13%	13%	12%	11%	11%	13%
D：前途无望	1%	2%	3%	2%	2%	3%	5%	4%	2%	3%	2%

图一　测试人员对测试领域工作前途态度

自动化测试现状

近年来随着我国软件产业的蓬勃发展以及对软件质量的重视，软件测试行业随之兴起。软件测试在国内已经成为一个较为成熟的产业，逐步与国际水平拉近距离。具体表现在:

（1）软件测试门槛低，薪资相对其他行业高，可谓是低报酬高回报的行业;

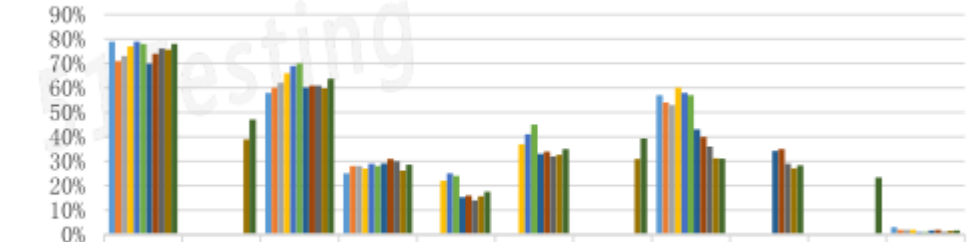
（2）软件质量问题的影响越来越大,而软件测试就是保证软件质量的一个重要并且有效的手段，因此现在软件公司越来越重视软件测试;

（3）软件测试行业发展势头良好，测试行业依旧处于供不应求阶段，自动化测试的发展整体上仍处于起步阶段。

自动化测试的发展前景

纵观人类发展方向，感应灯，扫地机器人，智能家具等的出现无一不诠释着自动化就是人类社会发展的大方向，软件测试行业也不例外，自动化测试一定是未来的方向。目前流行的敏捷、持续集成等都是以自动化为基础。所以说自动化测试的发展前景是乐观的。

由图一可见，超过 80%的从业者认为测试行业有前景或者前途光明。同时超过 75%的测试从业人员希望提高自动化测试的技能（图二），由此可见，自动化测试在软件测试行业的发展前景也是受到广大软件测试从业人员的广泛认可的。



	A：自动化测试	B：接口测试	C：性能测试	D：白盒测试	E：可靠性测试	F：安全性测试	G：大数据测试	H：测试管理	I：测试分析设计	J：云计算测试	K：其它
2007年	79%		58%	25%				57%			3%
2008年	71%		60%	28%				54%			2%
2009年	73%		62%	28%				53%			2%
2010年	77%		66%	27%	22%	37%		60%			2%
2011年	79%		69%	29%	25%	41%		58%			1%
2012年	78%		70%	28%	24%	45%		57%			1%
2013年	70%		60%	29%	15%	33%		43%	34%		2%
2014年	74%		61%	31%	16%	34%		40%	35%		2%
2015年	76%		61%	30%	14%	32%		36%	29%		1%
2016年	76%	39%	60%	26%	16%	33%	31%	31%	27%		2%
2017年	78%	47%	64%	29%	18%	35%	39%	31%	28%	23%	2%

图二 软件测试人员希望提高的技能

一、自动化测试介绍

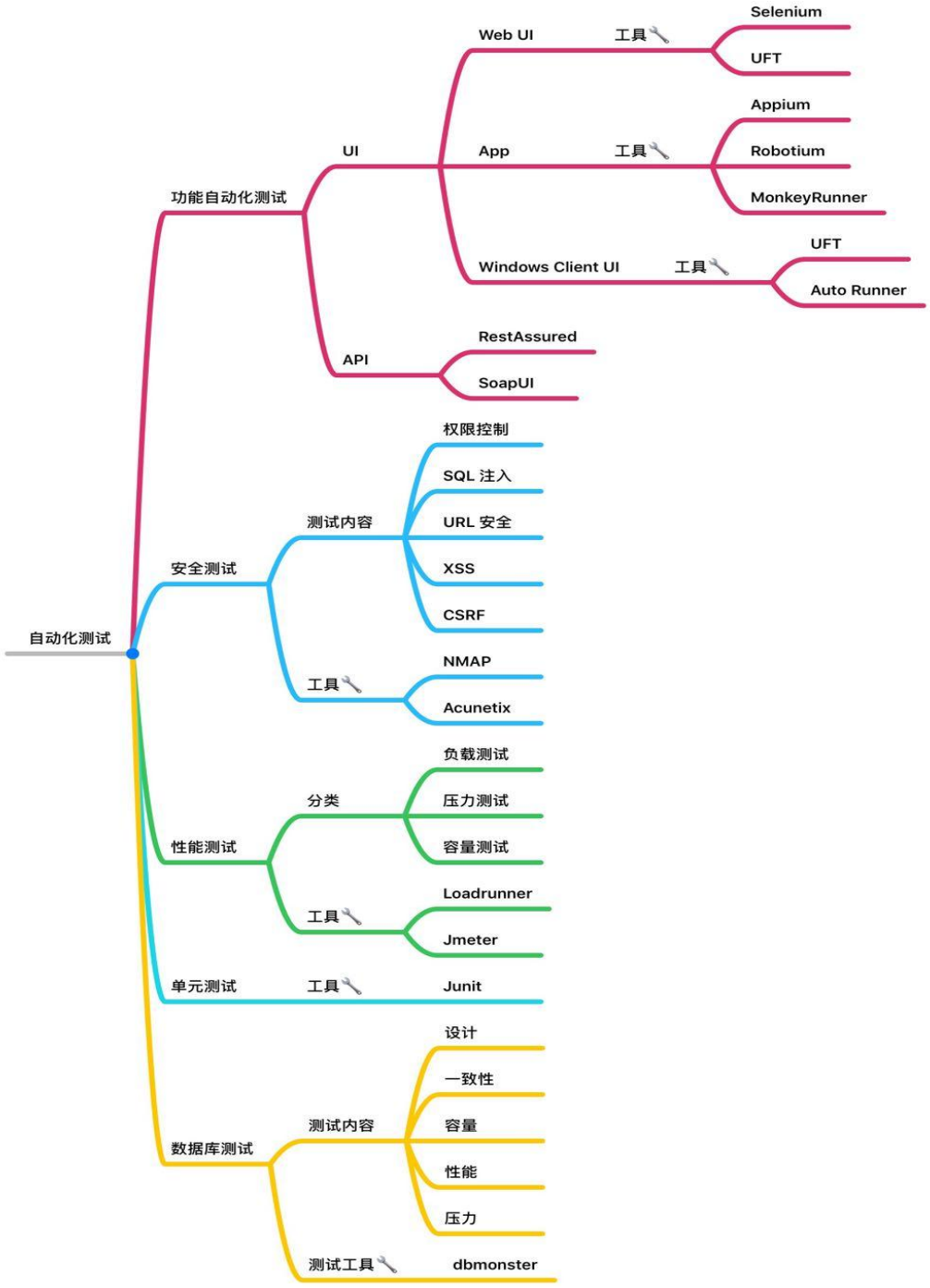
什么是自动化测试，自动化测试有哪些，自动化测试的意义以及什么样的项目适合做自动化，作为自动化测试人员我们应该具备哪些素质？了解自动化测试框架都对于自动化职业从业者的职业发展具有重要意义。

1.1、自动化测试的定义

自动化测试是通过一些自动化测试工具模拟人工操作验证其结果完成整个测试过程。自动化测试属于测试但是又不局限于测试，也是将软件开发过程中手动重复度高的，机械化的，人容易出错的步骤用自动化实现，再将这类工作的人力释放出来做自动化替代不了、有成长性或者创造性的工作。

1.2、自动化测试的分类

提到自动化测试，大多数人的第一反应就是 UI 的自动化测试，其实不然，它还包括了接口测试，单元测试，安全测试，性能测试以及数据库测试等，具体分类情况和每一类的测试的体系及工具如下如图三所示：



图三 自动化测试的分类

1.4、自动化测试的意义

为什么要进行自动化测试？因为自动化测试的优点显而易见:

- 提高测试效率，节省时间和人力成本;

每次新的 build 发布之前，可以通过自动化进行 BVT 测试，大大节省了时间和人力

成本。对于回归测试的效果也非常明显。

- 可以执行难以或不可能用手工完成的测试;

比如网站的压力测试，同时找很多人去测试不现实。便可以以自动化的方式通过机器去模拟。

- 一致性和可重复性强

由于测试是自动执行的，每次的执行内容和测试的结果的一致性是可以得到保障，从而达到可重复的结果。

- 项目质量流程需要

比如测试工具的开发，自动化在 DevOps 中的贡献。

自动化在 DevOps 中很重要因为它提供了准确性和速度。应用交付需要高效，而手动安全测试难以满足进度要求。更重要的是，第三方在外部手动测试中往往会遗漏测试错误。

1.5、什么样的项目适合做自动化

我们可以通过以下几方面来判断当前的项目适合做自动化:

1）项目维护周期长。如果维护周期太短，自动化框架有可能还没开发完，产品已经下线了。

2）比较频繁的回归测试。如果选用手工测试，第一轮可能还没完，第二轮就开始了。

3）产品比较稳定，不会频繁变动。如果产品不稳定，维护自动化用例成本太大，自动化测试投资回报率太低。

1.6、自动化测试人员应具备的能力

在早期，大多企业对测试人员的能力要求较低，导致人们的测试行业的认知都是:门槛低，水平低，能轻松胜任。而自动化测试，作为进阶，在测试基础欠缺的情况下，即使写出测试脚本也不能从根本解决问题。自动化测试人员应具备的能力可分为如下几个方面，这几方面在各个职业生涯发展阶段的要求也不尽相同，下一节会详细介绍各个阶段的要求。

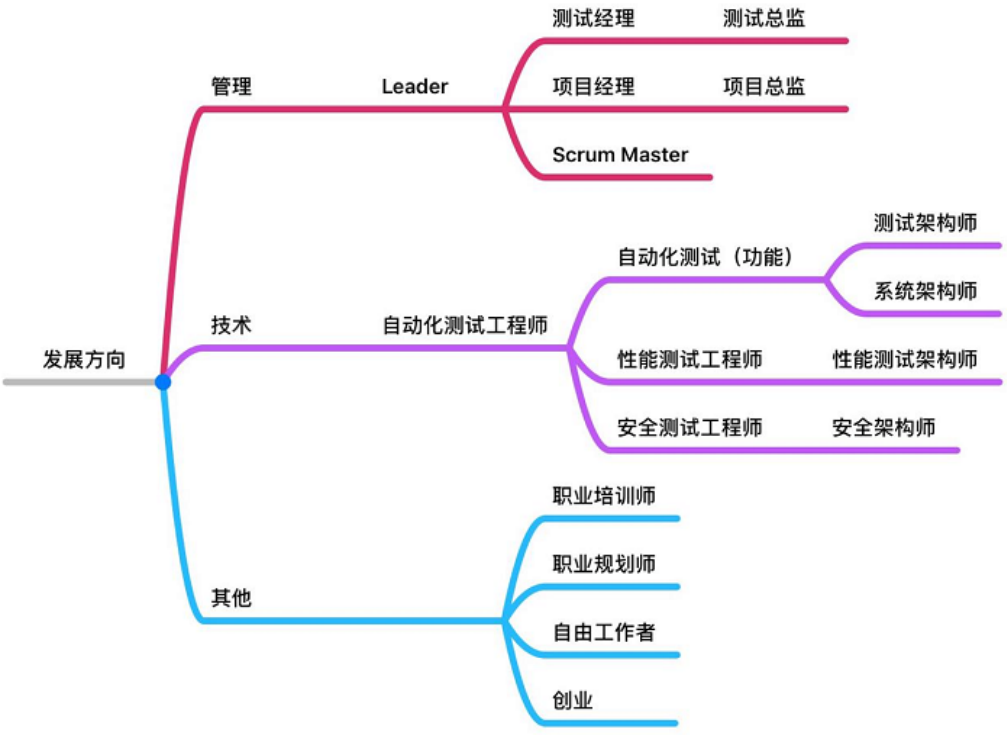
1）基础能力。基础能力是测试能力和代码能力的基础，是对自动化测试人员的基础要求，包括细心，耐心，责任心，学习能力，分析问题和解决问题的能力，沟通能力，团队协作的能力以及总结问题的能力。外企或者国外的项目同时也会有外语能力的要求。

2）测试能力。测试能力是做自动化测试的基础，能够很好的帮助自动化促使人员决定什么时候引入自动化，有效的设计自动化测试用例，很好的发现 bug，很好的理解自动化测试的目的与价值。测试能力主要包括业务分析能力，bug 的洞察能力以及基本的软件测试知识（bug 的生命周期，测试理论及方法等）

3）代码能力。良好的代码能力是自动化测试人员写出高质量的自动化测试脚本的前提，开发自动化测试脚本的过程中，需要考虑代码的稳定性，健壮性和可扩展性，同时结合数据结构和算法来减少后期的维护成本。代码能力主要包括自动化脚本的设计能力，框架开发能力，框架设计能力。

二、自动化测试的发展方向

俗话说：“方向比努力重要”，自动化测试行业也是如此。了解自动化测试的发展方向是自动化测试职业者做好职业规划的前提。下面是一个简单的发展路线（图四）:



图四 自动化测试从业者发展路线

2.1、在管理上的发展

2.1.1 所需技能

一般来说管理者不热衷于技术，但是一定对管理工作充满持续热情，善于沟通，组织协调能力强，长期抗压能力强。总的来说，管理者须具备如下三大技能:

1）技术性技能，是指运用专业能力来组织完成工作。如果技术和业务能力欠缺，难以很好的制定计划，预估风险，同时难以服众。对管理者技术和业务的能力要求是广而不深。

2）人际技能，是指管理者处人事关系的技能。即理解激励他人并与他人共事的能力，主要包括领导能力，影响能力和协调能力。人际技能要求管理者了解他人的信念、思考方式、感情、个性及态度等。

3）概念技能，是指一种洞察既定环境复杂程度的能力和减少这种复杂性的能力。包括理解事务的相互关联性从而找出关键影响因素的能力，确定和协调各方面关系的能力以及权衡不同方案优劣和内在风险的能力等。

2.1.1 资格认证

- PMP: 成为企业项目经理必须具备的任职资格，如果要进入跨国公司，PMP 资格认证是必备条件之一。
- PGMP: 是 PMP 的高级进阶，认可度非常高，不过难度要求也高。
- PRINCE2: 受控环境下的项目资格认证，与 PMP 相比更注重于实现方法。如果想作为项目主管或者项目经理，PRINCE2 提供了明确的程序，步骤以及模板。
- PMIACP: PMI 推出的敏捷认证，适用于任何行业，且不对从业者限定某一种敏捷方法。
- CSM: 专门面向 Scrum Master 的认证。
- PSM: 也是专门面向 Scrum Master 的认证，不同的是 PSM 的认证是终身的，CSM 没两年需要重新认证一次。

2.1.3 发展空间

如图五所示，自动化测试管理的基本角色包括组长（Leader），测试/项目经理

(PM)，测试/项目总监（Director）, Scrum Master。

2.1.3.1 Leader

因项目难易程度与企业规模而异，组长的要求也不尽相同， 一般来说组长不是纯粹的管理者，他们也会负责一部分自动化测试工作，同时负责项目组的管理工作，管理的对象是组。要求要有一定的自动化测试经验与管理能力。同时要有一定的业务能力与技术能力，便于更好的判断任务的优先级，更好的进行管理工作。组长的管理工作主要集中在任务的追踪与计划的制定及组内人员的协调，同时也会涉及时间的管理及质量的管理，与开发人员及客户的沟通

Leader 的职责:

- Team 内部 task 的追踪与管理;
- 自动化测试计划的制定，实施与追踪;
- 进度与目标的管理;
- Team 内部的沟通及人员的协调;
- 根据自动化测试情况编写与总结报告;
- 承担部分自动化测试工作。

2.1.3.2 PM

项目经理管理的对象是整个项目，一般不会参与具体自动化的任务。主要工作集中在从宏观上把控整个项目的进度，质量。同时需要预估项目风险，并使客户满意。同时对团队与人员的发展负责，要进行人员的招聘与培养，提高团队的胜任力。

PM 的职责:

- 负责项目的实施与管理，跟进并推动项目进度
- 负责与客户进行方案、需求、计划的交流与沟通;
- 向管理层汇报重大项目的情况及部门内部事务的管理;
- 软件产品测试方案的制定，开展和实施;
- 负责项目的交付，验收;

- 协调解决项目测试过程中的重大问题;
- 组建并管理团队，指导团队成员工作;
- 面试，选择聘用管理人员，并定期进行评价，考核和奖惩。

2.1.3.3 Director

总监直接负责项目的财务管理以及资源的调控。对于外包公司而言，项目经理与总监同时也会负责项目的周期，与销售人员的沟通等。

Director 的职责:

- 负责整个项目的财务管理及资源调控;
- 负责部门的组织及建设，员工的招聘及任免;
- 贯彻及执行公司的各项管理制度与标准;
- 负责跨部门的协调与沟通

2.1.3.4 Scrum Master

Scrum Master 是 Scrum 过程负责的人，确保 scrum 的正确使用并使得 Scrum 的收益最大化。与 PM 不同的是，Scrum Master 更侧重于教练和引导者，是设立在项目和客户之间的角色。Scrum Master 不管理实际工作的团队；反而，他会协助 PM，辅导团队，确保项目遵循 Scrum 流程。Scrum Master 只负责 Scrum 流程能够正确、持续的实施，发挥出它的最大效用。

Scrum Master 职责:

- 负责帮助团队每个人更好的理解和实践 Scrum;
- 服务型领导，服务于整个团队;

2.2、在技术上的发展

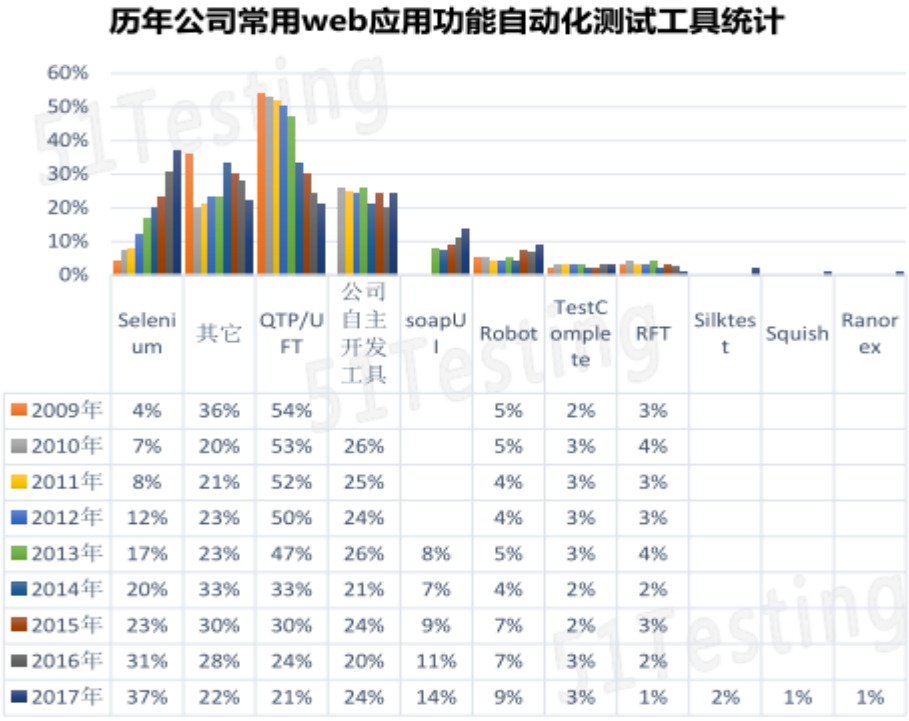
自动化测试从业者不仅具备测试理论，同时也有开发技术。所以自动化测试在技术上的发展比较广，按照资历可逐步由初级发展为高级再到资深，最后成为领域的专家。总体可分为测试设计方向的发展和测试开发方向的发展。

2.2.1 自动化测试及管理工具

2.2.1.1 测试工具

- Web UI 自动化测试:

根据软件现状调查报告（图五）显示，Selenium 是目前最受欢迎的 Web UI 自动化测试工具，比例呈逐年上升趋势。

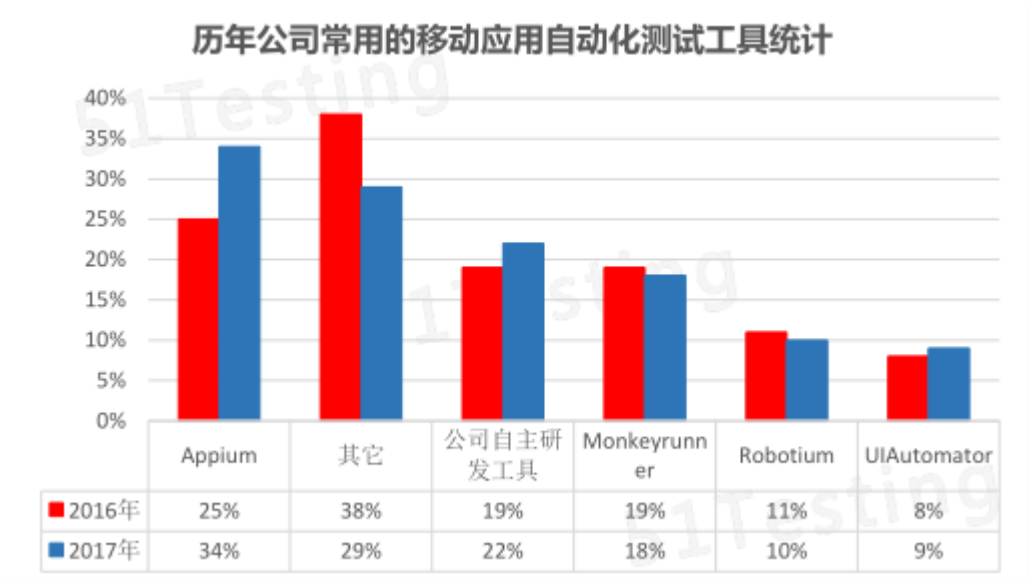


	Selenium	其它	QTP/UFT	公司自主开发工具	soapUI	Robot	TestComplete	RFT	Silktest	Squish	Ranorex
2009年	4%	36%	54%			5%	2%	3%			
2010年	7%	20%	53%	26%		5%	3%	4%			
2011年	8%	21%	52%	25%		4%	3%	3%			
2012年	12%	23%	50%	24%		4%	3%	3%			
2013年	17%	23%	47%	26%	8%	5%	3%	4%			
2014年	20%	33%	33%	21%	7%	4%	2%	2%			
2015年	23%	30%	30%	24%	9%	7%	2%	3%			
2016年	31%	28%	24%	20%	11%	7%	3%	2%			
2017年	37%	22%	21%	24%	14%	9%	3%	1%	2%	1%	1%

图五 web 应用功能自动化测试工具

- APP UI 自动化测试:

根据软件现状调查报告（图六），根据数据可知，Appium 是主流的移动应用自动化测试工具，近年呈上涨趋势。

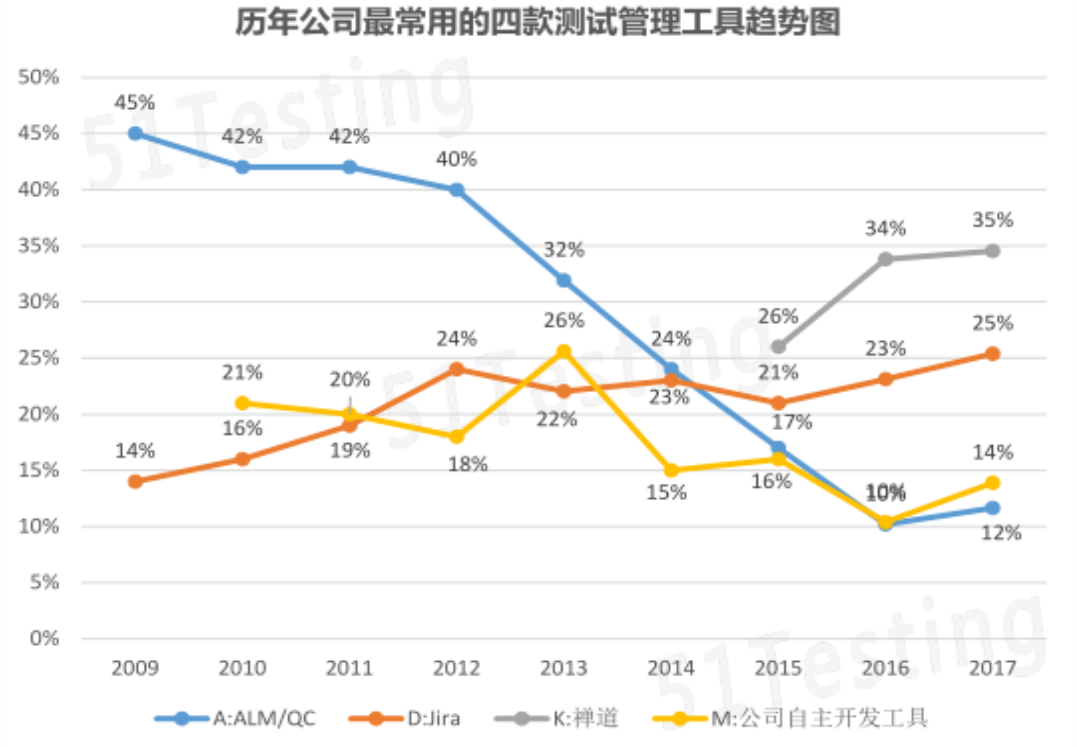


图六 常用移动应用自动化测试工具

- 性能测试及安全测试主流工具请参考图三。

2.2.1.2 测试管理工具

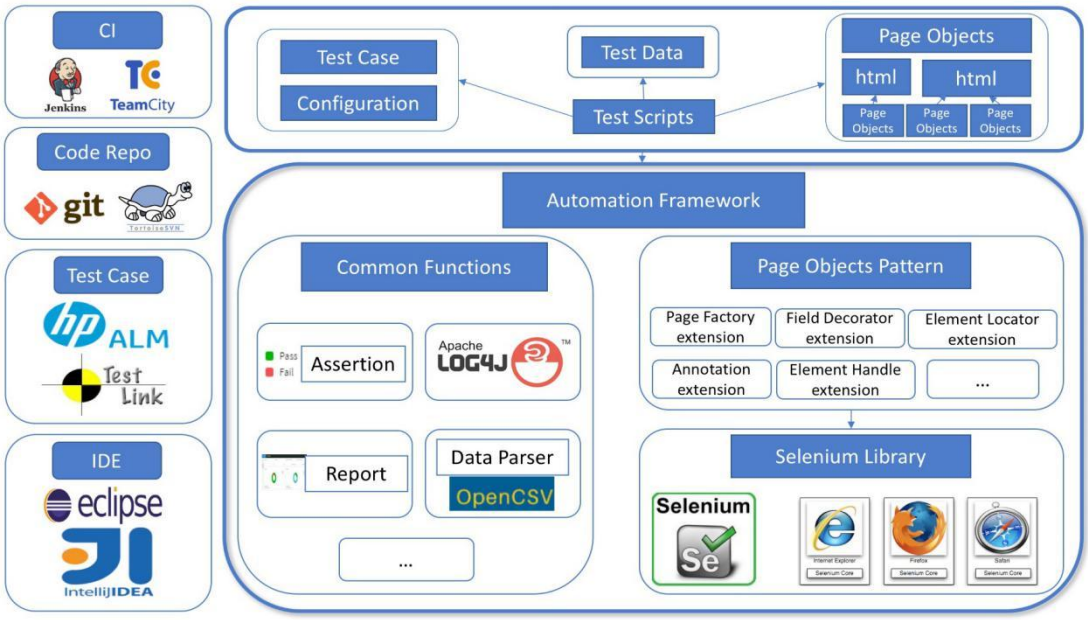
根据图七可知，目前主流的管理工具为禅道和 Jira，ALM 呈逐年下降趋势。



图七 主流测试管理工具

2.2.1.3 测试框架

自动化框架包括自动化测试框架本身以及执行的关系，自动化测试脚本的管理，测试用例的管理，以及自动化测试工具的选择。如下图八所示：



图八 自动化测试框架

2.2.2 功能自动化测试的发展

2.2.3.1 发展方向

初级自动化测试工程师 --> 中级自动化测试工程师 --> 高级自动化测试工程师 --> 测试架构师 --> 系统架构师

初级自动化测试工程师工作经验一般不超过两年。要求具备自动化测试的基础知识，有一定的代码能力，能够独立开发自动化测试用例，同时具有测试理论，了解软件开发及测试的生命周期。

中级工程师相比初级而言要求知识面更广更深一些，通常具有一定的开发经验，对质量管理，脚本语言数据库等领域都有了解。要求能够独立设计开发并维护自动化测试用例，并且有维护自动化测试框架的能力。

高级自动化测试工程师不仅需要有强大编程能力，同时需要丰富的测试经验，需要深刻理解开发与测试领域。要求能够理解并维护自动化测试框架。同时具备解决 team 内部各种疑难杂症的能力。

测试架构师须具备自动化测试框架的设计可开发能力，负责产品测试的整体架构设

计及对测试重点及难点进行研究，为整个产品提供最优的测试方案。同时负责协助测试经理制定测试计划和控制项目进度。

系统架构师负责产品开发的整体结构设计及产品开发过程中的技术重点与难点的研究，同时协助项目经理和控制项目进度。

2.2.3.2 所需技能

除了自动化测试人员所需具备的基本能力外，与管理者要求不同的是，选择技术路线的技术人员要求热衷于技术，对技术敏感。在自己经营的领域里，具有个人独到的见解和深厚的技术实力。

以当前最流行的 Selenium+TestNG+Java 为例，UI 自动化测试技能要求如下：

1）深厚的 Java 功底，所设计出的代码可读性，易用性，可重复性，可维护性，稳定性高；

2）熟悉并深入理解 Selenium+TestNG 框架，便于更好的理解并维护甚至设计自动化测试框架；

3）熟悉设计模式及代码重构原则；

4）对 DevOps 的理念和实践有比较深入的理解；

5）熟悉 Git 等代码管理工具；

6）熟悉 CI，Jenkins job 的创建，维护，执行；

7）熟悉测试理论方法，bug 的生命周期；

8）能较快适应变化及对新技术特别强的学习能力；

9）对分析问题与解决问题的能力。

API 所需技能：

1）熟悉被测 API 的协议；

2）熟悉协议的传输，响应和返回；

3）熟悉 API 测试框架

2.2.3 性能测试的发展

作为性能测试工程师，我们需要对系统进行分析，系统调用是如何进行的，CPU 是如何调度线程，有限的内存是如何影响性能的，文件系统是如何处理 I/O 的等。对于性能不佳的系统，我们需要建立性能数据分析模型，收集原始资料，分析性能产生原因。性能测试的分类参考图三，性能测试工程师的发展参考图五。

2.2.4 安全测试的发展

安全测试是在软件产品的生命周期中，特别是产品开发基本完成到发布阶段，尤其是金融类产品，对产品进行检验以验证产品符合安全需求定义和产品质量标准的过程。以发现安全隐患为主。安全测试的分类请参考图三，安全测试工程师的发展方向请参考图五。

2.3、其他

如图五所示，除了众所周知的“管理”与“技术”之外，自动化测试者还可以考虑以下几方面作为自己的职业发展方向。

2.3.1 职业培训师

职业培训师以讲授培训课程为职业，通过教授专业知识而获得报酬的人。根据工作性质不同，可分为：为公众服务，为特定人群服务，为企业服务三大类。

分类：PTT，TDP，TTT，ETT

要求具备的素质：以始为终；不断学些更新；教学相长的共赢思维。

2.3.2 职业规划师

是以职业人的个体利益出发，结合专业知识和相关资源，给与客户有关职业的适应，发展等方面的专业咨询、辅导、判断、和解决办法的专业人才。

最高门槛认证：CCDM

优点：高薪，稳定，自由度高

要求具备的能力：超凡的沟通能力和必要的评测技术。

2.3.3 自由职业者

自由职业者的最大好处就是可以自己掌控自己的时间，有选择自己想做什么工作的自由，同时拥有赚钱的潜力。弊端就是需要自己寻找业务。

2.3.4 创业

创业之路可能是职业生涯中最不确定但却最具挑战的选择。作为创业者的两大优势：完全的自由和完全的不封顶的赚钱潜力。你完全可以自由，随心所欲，也对自己的未来负全部责任。但是创业者必须考虑做什么才能赚钱，可能是最艰难，最冒险的职业抉择。同时还要学习销售、市场营销，以及商业和理财等诸多方面的技能。

三、自动化测试的职业规划

明确了可参考的职业发展方向，本节主要针对管理和技术方向的规划给出抛砖引玉的建议。根据著名的 10000 小时定律理论：即一个人想要成为某个领域的专家，需要经过一万个小时的锤炼。按此比例结算，如果每天有效工作时间为 8 小时，一周工作 5 天时间计算，那么成为一个领域的专家需要 4-5 年的时间。

3.1、职业规划的方法

职业规划可以通过两步走，一是职业方向定位，二是职业核心能力评测。

管理路线还是技术路线？其实自动化测试在“技术方向”和“管理方向”也是可以互换，交叉发展的。不管走纯技术路线还是纯管理路线最后都会进入瓶颈期。因为自动化测试的特殊性，管理者如果不懂技术，就不能很好的有效评估自动化测试的重点、难点，不能很好的做计划，控制项目进度。做技术的如果不懂管理，就不能很好的理解项目的价值，项目的目标和成本。容易陷入“唯技术论”中。所以不管选择做管理还是做技术，也同样完全取决于个人。我们可以通过核心能力评测来参考自己适合做什么。

职业核心能力是在人们工作和生活中除专业岗位能力之外取得成功所必需的基本能力，它可以让人自信和成功地展示自己、并根据具体情况如何选择和应用。

职业核心能力又分为三大部分：

- 基础核心能力：职业沟通、团队合作、自我管理
- 拓展核心能力：解决问题、信息处理、创新创业
- 延伸核心能力：领导力、执行力、个人与团队管理、礼仪训练、五常管理、心理平衡

当你有很高的职业发展期望时， 我们可以通过职业核心能力评测进行“胜任力”评估，用以支持你制定的职业目标并树立一个能力提升的方向与标准。

3.2、管理方向

关于如何成为自动化测试管理人才，首先一定要对测试理论，测试流程熟悉，其次要有一定的技术功底，亲自参与过 UI 自动化测试/性能测试/安全测试的工作。同时，尽量选择一个领域（比如金融，人力资源管理等）持续做下去，成为这个领域的业务专家。

发展建议：工作前三至五年积累经验，沉淀技术，学习测试理论，有意识的锻炼自己的管理技能。然后逐步向 Leader，PM 发展。

3.3、技术方向

关于如何成长为自动化测试专家，不管是 UI 自动化，还是性能跟安全。首先要了解系统的架构，然后再选择合适的测试工具，最后要保证多实践。

对于工作中没有机会接触性能测试跟安全测试工作的，可能通过自学或者参加培训来提高自己的能力。

四、结束语

本文简单介绍了自动化测试的概况，从职业方向和职业规划两个方面介绍了自动化测试的职业发展。读者可以参考前述章节尝试规划自己的职业发展。最后，衷心感谢奥博杰天软件的谈益平（博士）的命题，感谢 Kevin 从资料收集到论文修改过程中的帮助及建议，感谢 Gary 和 Martin 的意见和建议，感谢大曹的校对!

参考文献:

《中国软件测试现状调查报告》: http://www.51testing.com/html/38/n-3727438.html

大数据 “杀熟”前传

◆作者：侯 峥

相信很多小伙伴发现在某个 APP 上用自己账号搜索出的商品价格和用爸妈的账号搜索出的商品价格不一样，有的时候价格相差得还很大，这是什么原因呢？是 APP 出现了 Bug，还是自己眼花了？两者都不是，实际上这个现象就是大数据杀熟。不止是购物方面，线上的很多订单都存在杀熟的影子。

互联网的兴起已经有很多年了，这些年来沉淀下很多用户数据，商品信息数据，用户行为数据等等，这些数据对于商业营销是非常宝贵的资产，如何研究这些数据，并从中挖掘出其中的价值已经是互联网商业发展的必备技能。

如果说，互联网底层技术是互联商业发展的第一梯队，大数据分析就是互联网商业发展的第二梯队。当然，随着以后的发展，互联网商业也会遇到其他的发展挑战。大数据分析不止适用于线上商业，实体商业也同样需要大数据的分析结果作为运营基础。

大数据乍听起来好像是深不可测的样子，感觉离自己很遥远，实际上接触了之后会发现大数据离自己还是很近的。举个简单的例子，十一假期你准备出去玩，但是不知道去哪里好，作为一个互联网时代的人，我们的第一反映就是去网上搜索“十一旅游圣地”，然后再查评价，查攻略，查注意事项等等，然后再根据经验啊、直觉啊、喜好啊等等判断搜索出的信息的可靠程度，最后决定十一假期去哪里完。这一套下来就是整个数据分析的雏形。

小伙伴们理解了吗？当然大数据分析可不是上面说的那么简单，里面涉及了很多技术上和算法上难点。这段时间接触了一些大数据方面的需求，简单来讲大数据只有几个步骤：数据提取、数据清洗、数据存储、数据展示、数据分析。对于测试来讲，大数据方面方面主要涉及的环节是数据提取、数据清洗、数据存储、数据展示四个阶段，数据分析也有涉及但主要是协助，数据分析需要机器学习和数学的知识。

数据提取

说到数据提取不得不说的就是数据来源。赵本山的小品里有一句话：“我不想知道我是怎么来的，我只想知道我是怎么没的。”但是，在数据提取里我们一定要知道数据是怎么来的。

数据是怎么来的一般需要知道几个问题：1、数据是从哪里来的？2、数据是通过什么方式来的？3、数据提取的类别是什么？这几个方面直接影响数据提取的准确性和数据提取的效率。

1、数据是从哪里来的？

- 数据由公司系统采集提取；
- 数据由合作公司处理后传输

2、数据是通过什么方式来的？

- 由业务人员管理平台手动输入到数据库中；
- 通过插码或者其他技术方式存储入库；
- 接口传输数据入库；
- 通过文件传输数据，解析文件内容入库

3、数据提取的类别是什么？

1）增量提取

2）全量提取

这些数据被称为源数据，提取成功后会存储到指定的数据库中。在提取的过程中，如果数据不符合规范或者数据量级超出系统能够承受的阈值，数据提取将会有误或者失败。

数据清洗

数据清洗就是把脏数据清洗掉，提高数据质量。 在数据提取的过程中会应为某种原因导致存如数据库中的数据质量存在问题。这些数据在后续的分析中没有任何实际的意义，甚至会导致数据无法进行分析。

数据清洗一般包括数据分析，定义和执行清洗规则，清洗结果验证等步骤。数据清洗一般会检查拼写错误、去掉重复的（duplicate）记录、补上不完全的（incomplete）记

录、解决不一致的（inconsistent）记录。确认数据不存在数据内容和数据属性的问题后，会根据业务规则或者数据分析人员的要求重新整合数据。

数据存储

整合完毕的数据需要存储到目标数据库中，在数据存储过程中有些数据的格式、长度不符合数据库表的规范，这些数据便会入库失败。

截止到 2012 年，数据量已经从 TB（1024GB=1TB）级别跃升到 PB（1024TB=1PB）、EB（1024PB=1EB）乃至 ZB(1024EB=1ZB)级别。由于大数据的量级如此庞大，因此在数据存储时不仅仅要考虑数据存储的准确性，还需要考虑数据存储的效率以及数据存储发生异常时数据的回滚问题。

数据展示

大数据的量级决定了数据库表的结构会比较复杂。一般会根据业务按照月份、省份建表。也就是说同一个业务报表中的数据会从不同的表中读取。因为数据的来源不同，页面的展示就会有很多意外的“惊喜”。比如数据展示不正确，或者数据根本就不展示。常见原因有以下几点:

- 页面报表的横向和纵向内容与数据库的表结构不符
- 数据库表中没有数据
- 页面调度数据脚本没有执行
- 前端定义错误，页面报错

数据经过以上的处理之后就会发送给大数据分析工程师，对用户的分析（“杀熟”）就开始了。很多人对于大数据“杀熟”都很不喜欢，但是这是一种商业的运营模式，只不过是由之前的线下“杀熟”变成的现在的线上“杀熟”。这也是人们生活习惯的变化，也体现了技术的进步，习惯就好。对于大数据测试来说，测试的对象变得更加抽象，对于技术的要求则更高。但是这也不代表着大数据测试时高不可攀的，大数据测试的根本还是测试，基本功扎实，再补充大数据方面的知识，这些问题都不是事。

一种更好的报告性能测试结果的方法

◆ 作者：枫　叶

摘要：报告功能测试的结果相对简单，因为这些测试有一个明确的通过或失败的结果。报告性能测试的结果要更加微妙得多，而且显示这些价值的方法有很多，但是迈克尔·斯塔尔认为这些方法都不是特别有效。他提出了一种使性能测试结果一目了然的报告方法。

有效的汇报测试结果是我们专业的圣杯之一。如果正确操作，它将提高项目的质量，并帮助我们关注真正的问题。但是如果做得不好，它会增加混乱并降低测试人员带来的价值。

报告功能测试的结果相对简单，因为这些测试有一个明确的通过或失败的结果。报告性能测试的结果要微妙得多。

让我们从一个定义开始：出于这篇文章的目的，我使用“性能测试”这个术语来表示执行度量的任何测试，其一系列数值范围都被认为是可接受的结果。它可以是功耗的测量，网站并行服务的用户数量，可以从硬盘读取数据的速度，等等——任何一个非功能性需求的测量。

性能测试的第一个挑战是确定什么是“通过”。这在需求定义阶段经常被忽略。我看到过很多需求被解读成这样：“从数据库提取数据时间必须少于 10 毫秒”，或者“处理一个视频文件的速度应该至少为每秒 100 帧（fps）”。这些需求是不完整的，因为它们没有包含我们想要达到的实际目标。我们只知道我们允许容忍的最坏的结果，但仍然通过产品。这儿有两个问题。

首先，让我们假设我执行一次测试，发现处理视频文件在以 101 帧每秒的速度完成（回想需求是“至少 100 帧每秒”）。看起来很好，对吗？但是这是否意味着我们已经接近边缘（也就是说，产品难以满足需求），或者一切都是好的？假如需求定义得很好，它应该包含目标和最小值——例如，目标：120 帧每秒；最低：100 帧每秒。有这样的

需求，101 帧每秒的结果很清晰地表明了产品难以满足需求。

其次，当测试稍微失败时(例如，99 帧每秒)，产品经理为了“灵活”就会面临压力，并接受产品的现状。我们多少次听到 “确实，我们都低于最小值，但是我们几乎通过了，那么我们可以判定这是好的”？假如完整的需求是可用的（目标：120 帧每秒），那么就很清楚了，结果距离目标有多远，而且产品有一个真正的问题。

为了完整起见，我将提到，非功能性需求不仅必须指定目标和最小值，而且还必须指定测试方法，因为测试方法影响测试结果。举个例子，当度量 CPU 使用率，取决于我们如何执行度量，结果会变化很大。我们是否测量记录的最大值？一次持续多久？我们算测量的平均值吗？一秒测量几次？我们的测试中还有其它什么并行运行在 CPU 上吗？

从理论上讲，报告性能测试结果根本不是一个问题。只呈现出结果并且指出通过或者失败。但是再者，我们不仅想要知道结果；我们想知道结果和目标之间的关系。编写一份不太复杂但仍能提供完整状态图的报告是一项平衡工作。

我们可以使用一个表格:

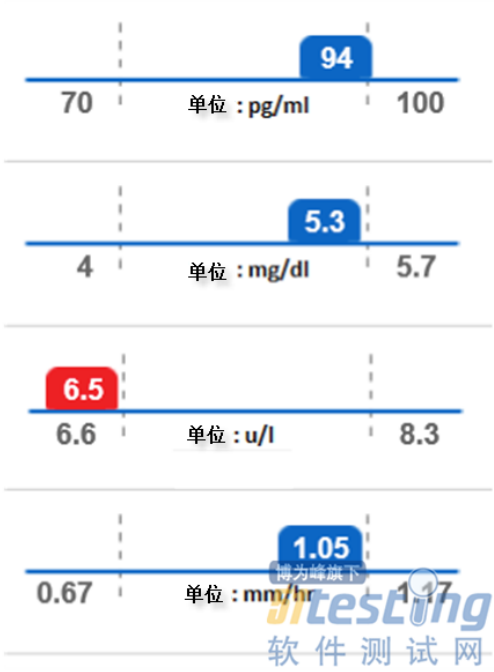
需求	目标	最小值	结果
视频处理速度（帧每秒）	120	100	101

但是，因为多数产品都有很多性能需求，所以我们最终会得到一个很大的表，其中充满了数字。它将难以快速看出哪里出了问题。我们可以使用颜色来提高可读性:

需求	目标	最小值	结果
帧处理速度（帧每秒）	120	100	101
CPU 占用率（%）	7	10	8.55
性能消耗	1.5	1.9	1.34

但是这带来更多问题。帧处理速度和 CPU 使用率得到相同的颜色代码有意义吗?一个几乎失败，当另一个则在可接受范围内。那么可能是用红色来处理色框？但是那么我们会使用什么颜色表示失败呢？一个绿色的结果我们考虑要多久才能变成黄色？更不用说由于一些人有色盲而可能出现的困难。

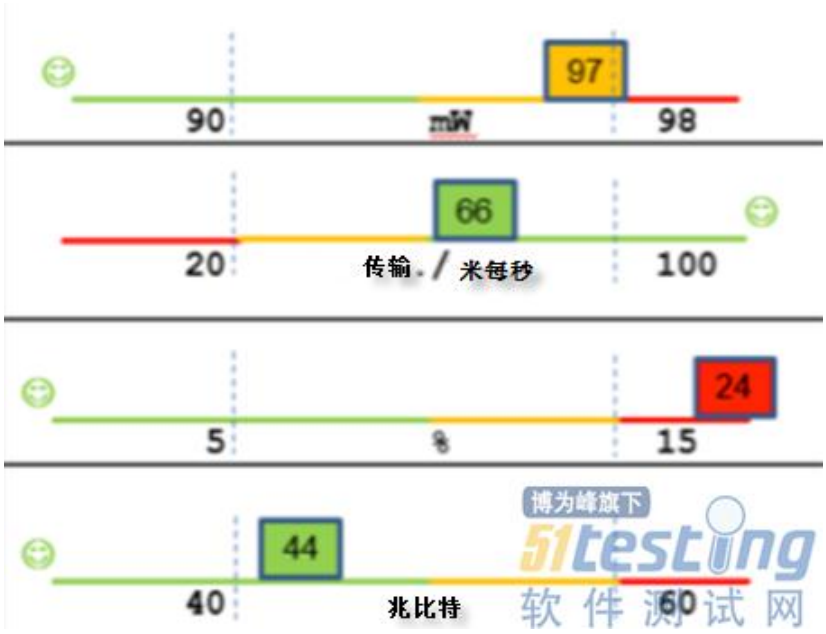
当我的医生派我去做每年一次的血液检查时，我正在考虑这个问题。无论如何，来自实验室的结果包含了一个以这种格式显示的几十个数字的列表：



即使我不是医生，我也能马上分辨出哪些结果是好的，哪些是次要的，并且哪些是我应该与医生讨论的事。

我脑子里闪过一个念头：为什么不使用这个方法来报告性能测试呢？我拿出一些数据点，并且用幻灯片做了个实验：

特性	分数
能量消耗	
传输/米每秒	
CPU 占用率	
内存使用	



请注意，我仍然使用颜色，但是坐标轴以独立于颜色的方式解释了颜色的选择，并确定了哪个高的更好，哪个低的更好。阅读器可以清楚地看到每个测量在允许范围内的位置;这些颜色主要用于在有麻烦的地方集中注意力。制作这样的报告需要一些时间，但它可以自动化。

我还没有在实际项目看见这个想法的实现——我仍然在研究这个想法——但是假如你确实使用这个想法，我将会高兴地了解到您的经验和您的组织的反应。

用按键精灵实现 APP 自动化

◆ 作者：枫　叶

简单介绍下应用背景：测试 app 时多次重复点击某一按钮的时候会出现报错，开发修复并提交测试时，如果采用手动点击按钮，效率不高，在开发经理提示下使用按键精灵实现自动操作。

一、安卓手机按键精灵 APP 录制（免 root 版）

适用于安卓 7 及以下系统的手机

1. 手机从应用市场下载并安装按键精灵 app;
2. 电脑安装按键精灵手机助手;
3. 在将手机连接 USB 前，请先找到开发者选项，打开“开发者选项”并勾选“USB 调试”;

4. 用数据线将手机连接电脑，连接后，电脑屏幕右下角会提示开始安装驱动，安

装完后会出现提示成功安装。

5. 电脑打开按键精灵手机助手，如果未识别设备，点击右上方“尚未连接到手机”，点击“重新扫描”，此时会搜索发现手机，选择开始连接。

添加上手机后，可以断开电脑的数据线，在 app 上进行操作了。

6. 按键精灵手机助手与手机连接成功后，会显示手机型号，同时也说明手机已成功获取到 root 授权。

7. 手机成功获取 root 授权后，就可以在手机端按键精灵上进行点一点、录制或运行脚本等操作；

8. 打开手机上的按键精灵 app，选择“录制”选项，点击录制图标，打开选择应

用页，选择要录制的 app

9. 点击保存的录制，可以进行循环设置，设置运行次数或者重复运行至手动停止

10. 录制时按音量+号开始录制，音量-号停止录制。

二、苹果手机按键精灵 APP 录制

适用于 iOS10.0.2 系统

1. 电脑下载并安装爱思助手 7.0;

2. 数据线连接电脑和手机;

3. 先将手机越狱：

还原——还原所有设置，再打开爱思助手———一键越狱；

注意将苹果手机备份（这个没有操作，因为测试手机不需要备份），并恢复出厂设置；

iOS10.0.2 越狱注意事项：

1）在越狱之前，最重要的是备份自己的设备。

首先将设备与电脑进行连接，使用爱思助手进行备份。越狱过程中可能出现种种不确定因素，进行备份可以将越狱的风险降低，同时越狱速度也会快一些。

2）恢复出厂设置后再进行越狱。

为了让越狱过程稳定而顺利的进行，建议大家将设备恢复出厂设置后再进行越狱。这个过程需要在手机上进行操作，在设置-通用-还原里面选择抹掉所有内容和设置，这个操作会让你的设备恢复出厂设置。

3）设备上显示“存储容量几乎已满”的问题。

这是由于越狱程序写入了系统目录导致的警报，第一次运行 Cydia 完成移动目录后

就不会再出现该提示。

4）越狱进程中，请确保 iTunes 已经关闭。

5）越狱整个过程大概历时 10 分钟，期间设备会自动重启，我们无需操作、只需等待。

越狱后，大概需要 10 分钟，手机会自动重启;

按提示安装 yalu102，再安装 Cydia;

打开 Cydia，添加 url: http://apt.mobileanjian.com/，软件源中搜索按键精灵，进行安装;

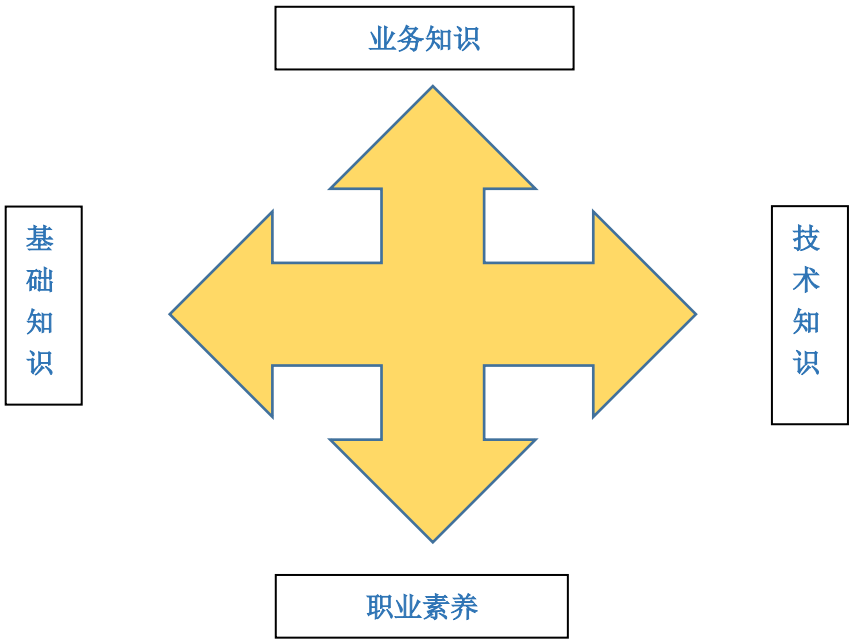
经过以上操作，可以打开手机端按键精灵开始录制，音量+号开始录制，音量-号停止录制了。

小白如何入门软件测试？

◆作者：咖啡猫

题记：前几天和同学小聚，有个同学抱怨道：“我们公司马上要改制了，之前是国企以后就不是了，待遇也大不如前哎。本来毕业的时候想着是做文职类工作轻松些而且国企也比较稳定，现在看来真得另谋出路了。咖啡猫，我想转行做软件测试咋样啊？你给我提点建议啊？”我相信我这位同学的经历并非个例，有很多朋友都因为各种各样的原因想转行进入软件测试行业，今天就单独写一篇文章，聊聊小白如何入门软件测试吧。

作为一个拥有四年多软件测试经验，参与过大大小小十来个项目，涉猎范围包括嵌入式项目、大数据项目、APP 项目、人工智能项目，做过功能测试、性能测试和自动化测试，也参与过测试人员的招聘的测试工程师，我认为入门软件测试需要四个方面的知识 or 技能，它们是：业务知识、职业素养、基础知识、技术知识。



这四者之间的关系见上图，职业素养是一切的根基，因为人在职场，就必须拥有必

要的职业素养，软件测试工程师也不例外。基础知识和技术知识是两大支柱，它们共同为业务知识服务。毕竟对于一个公司来说，盈利是主要目的，所以业务是根本目标。笔者记得有一年腾讯的面试官就说过一句话：“你的个人技能如果能给公司业务带来价值，那么你的个人价值就越高”

一、职业素养

职业素养包含的内容很多很多，这里只聊聊和测试强相关的几点吧。

首先是认真的工作态度。因为软件测试本来就是个细致活儿，不认真仔细地全心投入是很难发现潜在 BUG 的，一旦这些 BUG 流到最终用户手上，给公司带来的损失就是不可计数的。所以认真的工作态度非常非常重要！

其次是善于沟通和团队协作。现在的软件开发模式都是 team work 形式。所以善于沟通就显得非常重要了，比方说发现一个疑似 BUG，你如果和程序员小哥说：“我又发现了个 BUG，你快来看看”对方八成会说：“你自己环境配置有问题吧，我本地是可以运行的”然后就没有然后了，但是如果你和程序员小哥说：“老铁，有空吗？不知道是不是我的测试环境配置有问题，导致你开发的某某模块运行不了”程序员小哥八成会说：“操！不会又出 BUG 了吧”然后立马帮你排查问题。平时和团队成员沟通的时候，学会同理心，多站在别人角度思考问题，有个好人缘的话开展工作起来也相对容易些。

最后是较强的学习能力。因为 IT 业是一个技术更新迭代非常频繁的行业，可能今年流行的框架到了明年就过时了，所以需要从业者具备较强的学习能力。

二、基础知识

基础知识是入门 IT 行业的通用基础，类似大学里学过的公共课一样。只要是 IT 行业从业者，就必须要掌握。其中包括：

1、软件工程的相关知识。比如软件的定义、软件的生命周期、软件开发的模型、软件开发流程、计算机组成原理、操作系统分类等等。

2、软件测试的基本知识。比如软件测试的概念、软件测试的目的、软件测试的原则、产品质量模型、软件测试的分类、软件测试的基本流程、测试方案由哪几部分组成、BUG 的闭环流程是怎样的、测试用例怎么写、怎么高效提 BUG、软件测试处于软件开发周期中的哪个环节、相关利益方都有谁、测试报告的格式是什么等等。

3、英语。由于现在很多公司都会涉猎海外业务，而且很多开源技术文档都是英文的，所以学习英语其实对软件测试工程师来说也是非常重要的。所以平时多积累一些单词量、多听 VOA 音频、多看英文原文文档对英语学习是非常有帮助的。

三、技术知识

1、数据库基础。因为数据库是最常见的数据存储工具，所以掌握它非常有必要。比如数据库的概率、分类、实体联系模型、三大范式、数据的备份、存储过程、SQL 语句怎么写（增删改查）等等。

2、LINUX 基础知识。现在很多大型服务器都装 linux 操作系统，所以我们要熟悉 Linux 的文件结构、基本命令、shell 脚本相关知识。这里建议大家可以自己在电脑上装个 linux 虚拟机，再装 mysql。这样一来，就可以在上面联系 Linux 指令和 sql 语句了。

3、编程语言基础知识。可以学一门高级语言，比如 java、python 等。虽然测试工程师在编程上的要求可能略低于开发工程师，但是了解编程语言能让你成为一个 level 更高的测试。比如在 code review 的时候，你就可以站在测试的角度上对代码结构提出自己的建议；又比如高段位的测试开发工程师、自动化测试工程师等都需要编程能力，所以打好语言基础很重要。

四、业务知识

一切技术都是为业务服务的，所以当你进入一家公司一个项目组的时候，首先需要了解这家公司是做什么的，这个项目组在这家公司里处于什么地位。比如你去了一家银行，你所在的项目组负责银行自助 APP 开发，那么你就需要学习银行开户、存款、转账、销户等一系列业务需要遵循的流程，一般产品原型图里都有流程图，好好学习这些流程图对设计测试用例很有帮助。又不如你去了一家智能交通领域的公司，你所在的项目组负责开发城市道路交通违法行为自动抓拍的工具。那么你就需要了解一下道路交通法规，有哪些违法行为，如何界定这些违法行为，车牌号码的编制规则等等。一般来说，测试人员都可以从产品经理/项目经理/需求工程师处获取包含这些业务知识的文档，自学即可。有不懂的再请教其他同事。

后记:

写到这里，小白入门软件测试所需要的知识基本就介绍完毕了。可能有些朋友们要问，如何获取这些知识呢？你可以买书、买视频资料利用业余时间自学；你也可以关注

一些测试方面的公众号，利用零散时间学习；条件容许的话你还可以报名一些培训课程，在老师的指导下系统地学习软件测试知识。总之，网络时代获取学习资料不是问题，最最关键的是执行力！JUST DO IT！笔者见过太多买了一大堆学习资料，百度云盘里都存满了，但是从来没点开看的人了。所以，只有坚持学习才能成功转型，毕竟机会总是留给有准备的人！

TestNG 的依赖注入详解和使用场景分析

◆ 作者：王 练

摘要：软件设计方法中的依赖注入是比较晦涩的概念，采用这种方式能够解耦类之间的依赖，提高系统的灵活性。作为当今最为流行的自动化测试框架 TestNG，为了增强系统的灵活度，为我们提供了依赖注入的实现，给我们提供了很大的便利。作为 TestNG 使用者，我们可以不用理解过于晦涩的依赖注入的概念，而很便利的得到由此带来的好处，甚至我们可以不知道这种设计是依赖注入。

本文从依赖注入的概念说起，首先给出依赖注入的简介和入门级示例。之后介绍 TestNG 的两种依赖注入使用方式：原始依赖注入和外部依赖注入。对于每种方式，都给出简介和示例，同时对何时使用该种方式也做了初步的建议。本文的目的，一方面说明 TestNG 依赖注入设计的来源和使用方法，从而深入理解 TestNG，避免囫囵吞枣的使用，提高遇到问题的解决能力；另一方面，通过实例说明 TestNG 依赖注入的应用场景，防止理论无法落地实践，毕竟能够解决实际问题的方案才是有价值的。

一、依赖注入简介

1.1 依赖注入简介

详细介绍依赖注入需要很大的篇幅，由于本文不是专门介绍依赖注入，所以这里只做简单的说明。依赖注入，英文是 Dependency Injection，简称 DI，是软件架构设计中的一种技术。简单理解就是我们定义的类（A）如果需要依赖其他的类（B）来完成工作，通过依赖注入的设计，使得类 B 的实例化不在类 A 中进行，而通过第三方的实例化后传递给类 A，从而实现解耦的目的。这个第三方就是 DI 容器，也称 DI 框架。简单的说就是类之间的依赖关系，不再类之间直接引用，而是通过 DI 容器进行适配管理。

这样做的好处就是实现解耦，即类 A 虽然调用了类 B，但类 A 要与类 B 的实现解耦，这与设计模式中的依赖倒转原则一致：高层模块不依赖于底层模块，他们都应该依

赖于抽象，抽象不应该依赖于具体实现，具体实现应该依赖于抽象。

1.2 依赖注入示例

依赖注入的实现方式有很多，常用的有三种方式：构造方法注入（constructor injection），setter 方法注入（setter injection），接口注入（interface injection）。下面以构造方法注入作为示例，说明依赖注入的实现机制。

1.2.1 不使用依赖注入

考虑这样的需求，一个测试执行类，在测试完成后需要生成 Html 格式测试报告，一个简单的实现如下。

```
//无依赖注入的示例
public class TestRunner {
    //该类引用的外部类
    private HtmlReportService report;
    //构造方法
    TestRunner() {
        //指定特定的实现类,无依赖注入
        report = new HtmlReportService();
    }

    //业务逻辑中使用外部类方法
    public void run() {
        System.out.println("进行测试");
        System.out.println("生成如下格式报告： " + report.getReportType());
    }
}
```


HtmlReportService 实现如下。

```
public class HtmlReportService{

    public String getReportType() {
        return "HTML";
    }

}
```

这是没有使用依赖注入的情况的实现。由于在 TestRunner 类中直接使用了具体的报告实现类 HtmlReportService，所以当 HtmlReportService 发生变化或者需要增加其他类型报告服务时，都需要修改 TestRunner 代码。两个类的没有解耦。

1.2.2 构造方法依赖注入

对 TestRunner 类做如下修改，将 HtmlReportService 的引用修改为对接口的引用，同时不再需要 TestRunner 本身实例化报告服务，修改为通过构造法方法传递引用。具体代码如下。

```
//构造方法依赖注入的示例
public class TestRunner {
    //该类引用的外部类
    private IReportService report;
    //构造方法
    TestRunner(IReportService report) {
        //保存传递过来的实例
        this.report = report;
    }
    //业务逻辑中使用外部类方法
    public void run() {
        System.out.println("进行测试");
        System.out.println("生成如下格式报告： " + report.getReportType());
    }
}
```


HtmlReportService 实现变更如下。

```
public class HtmlReportService implements IReportService {

    public String getReportType() {
        return "HTML";
    }

}
```

IReportService 实现如下。

```
public interface IReportService {
   String getReportType();
}
```

简单的依赖注入组装，可以通过手工的编程方式完成，具体如下。

```
public class Injector {

    public static void main(String[] args) {
        //首先构造依赖
        IReportService report = new HtmlReportService();
        //通过构造方法方式注入依赖
        TestRunner testCase1 = new TestRunner(report);
        //使用对象
        testCase1.run();
    }

}
```

通过依赖注入的方式，完成了 TestRunner 与 HtmlReportService 的解耦，在确保 IReportService 接口能够兼容旧版本的的情况下，修改 HtmlReportService 实现，或者增加 IReportService 实现类，对 TestRunner 都没有影响。需要修改的是依赖注入的组装过程。

当然可以通过更灵活、强大的第三方框架完成依赖注入的组装，就是上文提到的 DI 容器。我们平常所说的依赖注入学习，除依赖注入本身的概念和知识之外，更重要的是依赖注入框架的学习。上述例子可以通过 Spring 框架进行组装。

```
public class InjectorBySpring {
    public static void main(String[] args) {
        //组装对象
        BeanFactory beanfactory = new ClassPathXmlApplicationContext("Beans.xml");
        TestRunner testCase1 = (TestRunner) beanfactory.getBean("TestRunner");
        //使用对象
        testCase1.run();
    }
}
```

Beas.xml 编写如下。

```
<?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="report" class="com.demo.injection.wiki.nodi.HtmlReportService">
    </bean>

    <bean id="TestRunner" class="com.demo.injection.wiki.constructor.TestRunner">
        <constructor-arg ref="report" />
    </bean>
</beans>
```


Spring 框架还支持通过注解的方式对依赖注入进行组装，方式更为灵活。

1.3 依赖注入与控制反转

提到依赖注入就不能不提控制反转。控制反转（Inversion of Control，简称 IoC），概念也是很晦涩难懂，这里不能详细介绍。简单来说，所谓控制就是使用，反转就是控制权的反转。将使用的对象的控制权，从使用类本身移交给第三方，也就是 IOC 容器控制，就称为控制反转。

从这个角度来看，依赖注入与控制反转实际是一个概念的不同表述，由于控制反转

概念比较含糊（可能只是理解为容器控制对象这一个层面，很难让人想到谁来维护对象关系），所以 2004 年大师级人物 Martin Fowler 又给出了一个新的名字：“依赖注入”，相对 IoC 而言，“依赖注入”明确描述了“被注入对象依赖 IoC 容器配置依赖对象”。也有人理解依赖注入是实现控制反转的方法，也是有一定道理的。

二、TestNG 中的依赖注入

从上述描述中，我们可以看到，依赖注入是一种软件架构的设计方法。对于 TestNG 来说，TestNG 的依赖注入实际是提供了使用该框架的人一种依赖注入的能力，也就是在 TestNG 框架下能够很容易的实现依赖注入。具体来说，TestNG 支持两种依赖注入的方法：原始方法(TestNG 框架自身实现的)，外部方法(通过类似 Guice 等依赖注入框架实现的)。

2.1 原始依赖注入方法

2.1.1 原始方法简介

TestNG 的原始依赖注入方法，是 TestNG 框架已经实现的 DI 容器完成了特定对象的依赖，当使用者需要是，按照特定的方式说明注入即可。TestNG 的 DI 容器会根据使用者的需要将特定对象注入到特定位置。更具体的，当使用 TestNG 框架管理测试时，可以在方法中添加额外参数，这些参数是我们需要的某个对象的声明。当 TestNG 发现我们如此配置时，会对该对象赋予正确的实例，供我们使用。这些方法是 TestNG 注解修饰的方法，这些对象就是需要注入的 TestNG 内部对象。目前支持依赖注入的方法和对象如下:

1）任何@Before 方法或@Test 方法都可以定义一个 ITestContext 参数。

2）任何@AfterMethod 方法都可以定义一个 ITestResult 参数，该参数传递的刚刚运行的测试方法的结果。

3）任何@Before 方法和@After 方法(@BeforeSuite 和@AfterSuite 除外)都可以定义一个 XmlTest 参数，这个参数传递的是当前<test>标签内的信息。

4）任何@BeforeMethod 方法和@AfterMethod 方法都可以定义一个 java.lang.reflect.Method 参数。对于@BeforeMethod 该参数传递的是即将执行测试的方法，对于@AfterMethod 该参数传递的是刚刚运行的测试方法。

5）任何@BeforeMethod 方法都可以定义一个 Object[]参数。该参数传递的就是即将执行的测试方法的参数列表，这个参数或者是 TestNG 为我们注入的（就像 Method 一样），或者是来自@DataProvider 的。

6)任何@DataProvider 方法都可以定义一个 ITestContext 参数或 java.lang.reflect.

Method 参数。后一个参数将接收即将被调用的测试方法。

在方法中使用 @NoInjection 注解可以关闭依赖注入。

下表总结了全部可以进行依赖注入的注解修饰方法和相应的参数:

Annotation	ITestContext	XmlTest	Method	Object[]	ITestResult
BeforeSuite	Yes	No	No	No	No
BeforeTest	Yes	Yes	No	No	No
BeforeGroups	Yes	Yes	No	No	No
BeforeClass	Yes	Yes	No	No	No
BeforeMethod	Yes	Yes	Yes	Yes	Yes
Test	Yes	No	No	No	No
DataProvider	Yes	No	Yes	No	No
AfterMethod	Yes	Yes	Yes	Yes	Yes
AfterClass	Yes	Yes	No	No	No
AfterGroups	Yes	Yes	No	No	No
AfterTest	Yes	Yes	No	No	No
AfterSuite	Yes	No	No	No	No

2.1.2 原始方法示例

使用原始方法进行依赖注入非常简单，只要根据上表在期望注入的注解方法中添加相应形参即可。比如下面的例子，在@AfterMethod 注解的 onAfterMethod 中添加了 Method，ITestResult，Object[]三个对象，含义就是通知 TestNG，该测试类需要在该方法注入如上的三个对象。这样 TestNG 就是将，这三个对象的实例传递给 onAfterMethod，具体这三个对象是如何实例化的，如何维护的，测试类是不需要关心的。当然，TestNG 能够确保对象的正确性，比如 Method 传递的就是刚刚运行的测试方法名称。

```
@AfterMethod
public void onAfterMethod(Method method,ITestResult result,Object[] data) {
    System.out.println(method.getName()+",进行如下测试数据请看:"+data[0].toString());
}
```

2.1.3 场景 1:BeforeMethod 中使用测试数据

默认的@BeforeMethod 给定的 onBeforeMethod 函数是没有参数的，如果需要在 onBeforeMethod 函数中对测试环境根据测试数据进行处理时如何实现呢？可以如实现如下的测试类。

```
public class DiDemo14TestData extends DemoBaseTester{
    @Test(dataProvider="Test-Data")
    public void testModifyUser(HashMap<String, String> data) {
        System.out.println("测试如下测试数据(old_desc-new_desc):"+data.get("old_desc")+"-"+data.get("new_desc"));
    }
    @BeforeMethod
    public void onBeforeMethod(Object[] data) {
        System.out.println("进行如下测试数据准备:"+data[0].toString());

    }
    @AfterMethod
    public void onAfterMethod(ITestResult result,Object[] data) {
        System.out.println("进行如下测试数据清理:"+data[0].toString());
    }
}
```

利用 TestNG 的依赖注入机制，在 onBeforeMethod 声明使用对 ITestResult result,Object[] data 两个对象的依赖，TestNG 会根据上表自动为我们注入这两个类的实例。上述示例中 DemoBaseTester 为统一读取 excel 的基类，实现略。编写 testng.xml，制造测试数据后，运行结果如下。

```
[TestNG] Running:
  E:\1[redacted]AutoTest\Source_Code\[redacted]_WebUI_AutoTest\testng4debug-wld2.xml

进行如下测试数据准备:{new_desc=修改为的描述1, old_desc=需要修改的描述1}
测试如下测试数据(old_desc-new_desc):需要修改的描述1-修改为的描述1
进行如下测试数据清理:{new_desc=修改为的描述1, old_desc=需要修改的描述1}
进行如下测试数据准备:{new_desc=修改为的描述2, old_desc=需要修改的描述2}
测试如下测试数据(old_desc-new_desc):需要修改的描述2-修改为的描述2
进行如下测试数据清理:{new_desc=修改为的描述2, old_desc=需要修改的描述2}
进行如下测试数据准备:{new_desc=修改为的描述3, old_desc=需要修改的描述3}
测试如下测试数据(old_desc-new_desc):需要修改的描述3-修改为的描述3
进行如下测试数据清理:{new_desc=修改为的描述3, old_desc=需要修改的描述3}

===============================================
demo
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

2.1.4 场景 2:AfterMethod 中获取配置文件参数

在 AfterMethod 中能够获取很多内容，通过默认的 ITestResult 实例已经可以完成很多功能了，使用 TestNG 依赖注入 ITestContext 实例，能获取更过能力。如下示例给出了获取配置文件中 parameter 定义的内容。

```
public class DiDemo14TestResult extends DemoBaseTester{
    @Test
    public void testModifyUser() {
    }
    @AfterMethod
    public void onAfterMethod(ITestResult result,ITestContext context) {
        System.out.println("测试参数driver_host为:"+context.getSuite().getParameter("driver_host"));
    }
}
```

配置文件 testng.xml 编写如下。

```
<suite name="demo" parallel="tests" thread-count="1" >
    <parameter name="driver_host" value="192.168.20.200"/>
    <test name="guice">
        <classes>
            <class name="com.demo.DiDemo14TestResult" />
        </classes>
    </test>
</suite>
```

运行结果如下。

```
[TestNG] Running:
  E:\1[redacted]UIAutoTest\Source_Code\[redacted]WebUI_AutoTest\testng4debug-wld2.xml

测试参数driver_host为:192.168.20.200

===============================================
demo
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

2.1.5 其他场景

其他场景与上面的两种情况类似，通过查阅 TestNG 给定的依赖注入表格，能够知道在每个注解方法下，TestNG 提供给我们的注入能力。

具体根据业务逻辑进行选择。比如在 BeforeClass 中注入 ITestContext 实例，就可以通过这个接口获取非常多的资源，对逻辑处理、日志打印都是非常有帮助的。

2.2 外部依赖注入方法(Guice)

2.2.1Guice 简介

Guice 是 Google 开发的一个轻量级，基于 Java5（主要运用泛型与注释特性）的依赖注入框架(DI)。对比 Spring 框架，Guice 非常小而且快。支持构造方法，属性，方法（包含任意个参数的任意方法，而不仅仅是 setter 方法）进行注入。Guice 采用 Java 加注解的方式进行托管对象的配置，充分利用 IDE 编译器的类型安全检查功能和自动重构功能，使得配置的更改也是类型安全的。Guice 提供模块对应的抽象 module,使得架构和设计的模块概念产物与代码中的 module 类一一对应，更加便利的组织和梳理模块依赖关系，利于整体应用内部的依赖关系维护，而其他 IOC 框架是没有对应物的。

2.2.2Guice 示例

依然使用上文的例子进行说明，可以对比一下与 Spring 框架 DI 的异同。

首先 HtmlReportService 和 IReportService 的实现不变。

TestRunner 的实现增加@Inject 注解并实现一个新定义的接口 IRunner。

```
public class TestRunner implements IRunner {
    private IReportService report;
    @Inject
    TestRunner(IReportService report) {
        this.report = report;
    }
    public void run() {
        System.out.println("进行测试");
        System.out.println("生成如下格式报告: " + report.getReportType());
    }
}
```

IRunner 实现如下。

```
public interface IRunner {
    public void run();
}
```

Guice 的依赖注入组装需要通过继承 AbstractModule 父类，并实现 configure 方法，具体如下。

```
public class TestRunnerModule extends AbstractModule {
    @Override
    protected void configure() {
        bind(com.demo.injection.wiki.nodi.IReportService.class).to(com.demo.injection.wiki.nodi.HtmlReportService.class);
        bind(com.demo.injection.guice.application.IRunner.class).to(com.demo.injection.guice.application.TestRunner.class);
    }
}
```

TestRunnerModule 中定义了接口与实现类之间的关系，共 Guice 后续组装使用。

如下示例展示了如何完成业务逻辑的编程，在这里 Guice 会通过 TestRunnerModule 查找依赖的具体实现类，并根据实际业务逻辑的@Inject 注解将构造函数中指定的对象进行注入。

```
public class InjectorByGuice {

    public static void main(String[] args) {
        //组装对象
        Injector injector = Guice.createInjector(new TestRunnerModule());
        IRunner testCase = injector.getInstance(IRunner.class);
        //使用对象
        testCase.run();
    }
}
```

综合来看 Guice 是更轻量的依赖注入框架，能够通过简单的注解和接口实现完成依赖注入的需求。通过 Guice 在 TestNG 中进行依赖注入的定制，有 3 中使用方式，一种

比一种更为灵活。

2.2.3 场景 1:固定注入

在 TestNG 中使用 Guice 进行依赖注入，最简单的方式是通过实现 Module 接口，固定注入对象。这种场景适用于需要使用依赖注入完成测试类与依赖类的解耦，并且注入的对象时固定的情况。

考虑下面的场景，测试类 GuiceTest 需要依赖接口 ISingleton 的实现类 GuiceExampleModule 完成测试工作，但考虑到后续的扩展性，需要将 GuiceTest 和 GuiceExampleModule 通过依赖注入完成解耦。实现 GuiceTest 如下。

```
@Guice(modules = GuiceExampleModule.class)
public class GuiceTest{

  @Inject
  ISingleton m_singleton;

  @Test
  public void singletonShouldWork() {
    m_singleton.doSomething();
  }

}
```

通过@Inject 注解通知 Guice 为我们注入该对象，其中 ISingleton 实现如下。

```
public interface ISingleton {
    void doSomething();
}
```

这里面涉及到单例的问题，这只是测试类对该接口的实例化要求，与依赖注入本身并无关系。我们可以不考虑这个因素。作为 ISingleton 接口的实现类 ExampleSingleton，实现如下。

```
public class ExampleSingleton implements ISingleton{
    public void doSomething() {
        System.out.println("可爱的ExampleSingleton通过Guice注入，完成工作");
    }
}
```

通过上面 Guice 的示例，我们知道为了让 Guice 正常工作，需要定义接口与实现类的关系，即继承 AbstractModule 父类，实现 configure 方法。

```
public class GuiceExampleModule extends AbstractModule {
    public void configure() {
        bind(ISingleton.class).to(ExampleSingleton.class).in(Singleton.class);
    }
}
```

与示例不同的是增加了.in(Singleton.class)的调用，就是上文提到的 Guice 在注入时确保了该实例的单例性。

TestNG 配置文件编写如下。

```
<suite name="demo" parallel="tests" thread-count="1">
    <test name="Guice1">
        <classes>
            <class name="com.demo.injection.guice1.GuiceTest" />
        </classes>
    </test>
</suite>
```

运行结果如下。

```
[TestNG] Running:
  E:\1-[redacted]Source_Code\[redacted]UI_AutoTest\testng4debug-wld2.xml

可[redacted]的ExampleSingleton通过Guice注入,完成工作

===============================================
demo
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

在这里隐含着一层含义，Guice 的依赖注入组装是没有显示调用的，换句话说是 TestNG 完成了依赖注入的组装，也就是 Guice 对象 Injector 的使用。这也是 TestNG 说明可以通过 Guice 完成依赖注入的意义，即 TestNG 帮助我们完成了 Guice 的管理。

2.2.4 场景 2:灵活注入

上述方式注入的对象是固定的，如果有需要根据情况进行不同对象的注入时，应该如何完成呢？TestNG 整合 Guice 有灵活的方式，还是上述的例子，测试类的 Guice 注解使用 moduleFactory 修饰，具体如下。

```
@Guice(moduleFactory = ModuleFactory.class)
public class GuiceModuleFactoryTest {

  @Inject
  ISingleton m_singleton;

  @Test
  public void singletonShouldWork() {
    m_singleton.doSomething();
  }
}
```

含义是通过 mouduleFactory 完成注入类与接口的绑定，这个 mouduleFactory 是 ModuleFactory 类，实现如下。

```
public class ModuleFactory implements IModuleFactory {

    @Override
    public Module createModule(ITestContext context, Class<?> testClass) {
        Module moudle = null;
        if(context.getName().equals("Guice-Inject-1")){
            moudle = new GuiceExampleModule1();
        }
        if(context.getName().equals("Guice-Inject-2")){
            moudle = new GuiceExampleModule2();
        }
        return moudle;
    }

}
```

需要实现 IModuleFactory 接口，具体为 createModule 函数，函数提供了 TestNG 的 ITestContext 对象和当前测试类对象。createModule 函数可以自己的业务逻辑，通过这两个对象完成 Module 的创建。这里的例子，是根据<test>标签的名字完成不同的绑定，当为"Guice-Inject-1"时注入 ExampleSingleton1.class，当为"Guice-Inject-2"时注入 ExampleSingleton2.class。当然，需要定义这两个 module，具体如下。

```
public class GuiceExampleModule1 extends AbstractModule {
    public void configure() {
        bind(ISingleton.class).to(ExampleSingleton1.class);
    }
}
```

```
public class GuiceExampleModule2 extends AbstractModule {
    public void configure() {
        bind(ISingleton.class).to(ExampleSingleton2.class);
    }
}
```

需要注入的对象 ExampleSingleton1 和 ExampleSingleton2 的实现如下。

```
public class ExampleSingleton1 implements ISingleton{
    public void doSomething() {
        System.out.println("Test:Guice-Inject-1对ExampleSingleton1通过Guice注入,完成工作");
    }
}

public class ExampleSingleton2 implements ISingleton{
    public void doSomething() {
        System.out.println("Test:Guice-Inject-2对ExampleSingleton2通过Guice注入,完成工作");
    }
}
```

TestNG 配置文件编写如下。

```
<suite name="demo" parallel="tests" thread-count="1">
    <test name="Guice-Inject-1">
        <classes>
            <class name="com.demo.injection.guice2.GuiceModuleFactoryTest" />
        </classes>
    </test>
    <test name="Guice-Inject-2">
        <classes>
            <class name="com.demo.injection.guice2.GuiceModuleFactoryTest" />
        </classes>
    </test>
</suite>
```

运行结果如下。

```
[TestNG] Running:
  E:\I-[redacted]AutoTest\Source_Code\[redacted]UI_AutoTest\testng4debug-wld2.xml

Test:Guice-Inject-1对ExampleSingleton1通过Guice注入,完成工作
Test:Guice-Inject-2对ExampleSingleton2通过Guice注入,完成工作

===============================================
demo
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

工厂类会接收到 ITestContext 和测试类两个由 TestNG 初始化好的实例，我们通过实现 createModule 方法返回一个 Guice Module 对象的实例，该实例给出了测试类依赖的对象绑定。通过 ITestContext 和测试类的实例，我们可以获取测试环境的全部信息，比如在 testng.xml 中定义的参数等。

2.2.5 场景 3:注入抽象

TestNG 整合 Guice 除代码层面外，还支持 testng.xml 中的配置。使用 parent-module 和 guice-stage 参数，对<suite>标签进行修饰能够获得更多的扩展性。通过 guice-stage 的定义我们可以选择创建父 Injector 的阶段。默认为 DEVELOPMENT，可以配置为 PRODUCTION 或者 TOOL。这是 Guice 对软件运行阶段的定义，根据不同阶段的定义，Guice 提供的能力有所不同，TOOL（最小代价，有些功能会无法使用）

DEVELOPMENT（快速启动，但不会做校验）PRODUCTION（异常检查与性能，启动会比较慢）。TestNG 使用示例如下:

```
<suite parent-module="com.example.SuiteParenModule" guice-stage="PRODUCTION">
</suite>
```

对于如上配置的测试套件，TestNG 只会实例化一次该 Module。使用这样的配置来获取指定的 Guice Module 和 Module 工厂，之后再给创建每个测试类的 Injector。使用这种方法，我们可以在 parent-module 中声明所有公共绑定，也可以在 Module 和 Module 工厂中注入在 parent-module 中声明的绑定。这样就实现了注入的抽象，示例如下。

首先定义 parent-module 类，实现如下。

```
public class ParentModule extends AbstractModule {
    protected void configure() {
        bind(MyService.class).to(MyServiceImpl.class);
        bind(MyContext.class).to(MyContextImpl.class).in(Singleton.class);
    }
}
```

其中绑定类 MyService 和 MyServiceImpl 实现如下。

```
public interface MyService {
    void serve(MySession session);
}
```

```
public class MyServiceImpl implements MyService{
    public void serve(MySession session) {
        System.out.println("parent-module中指定MyServiceImpl绑定");
    }
}
```

其中绑定类 MyContext 和 MyContextImpl 实现如下。

```
public interface MyContext {
    MySession getSession();
}
```

```
public class MyContextImpl implements MyContext{
    public MySession getSession() {
        System.out.println("parent-module中指定MyContextImpl绑定");
        return new MySession();
    }

}
```

之后定义需要被注入的 Module。

```
public class TestModule extends AbstractModule {
      private MyContext myContext;
      @Inject
      TestModule(MyContext myContext) {
        this.myContext = myContext;
      }
      @Override
      protected void configure() {
        bind(MySession.class).toInstance(myContext.getSession());
      }
    }
```

最后测试类定义如下。

```
@Test
@Guice(modules = TestModule.class)
public class TestClass {
  @Inject
  MyService myService;
  @Inject
  MySession mySession;

  public void testServiceWithSession() {
    myService.serve(mySession);
  }
}
```

配置文件 testng.xml 编写如下。

```
<suite name="demo" parallel="tests" thread-count="1" parent-module="com.demo.injection.guice3.ParentModule">
    <test name="guice">
        <classes>
            <class name="com.demo.injection.guice3.TestClass" />
        </classes>
    </test>
</suite>
```

运行结果如下。

```
parent-module中指定MyContextImpl绑定
[TestNG] Running:
  E:\1-[redacted]AutoTest\Source_Code\[redacted]_WebUI_AutoTest\testng4debug-wld2.xml

parent-module中指定MyServiceImpl绑定

===============================================
demo
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

可以看到，ParentModule 类中定义了 MyService 和 MyContext 的绑定。之后 MyContext 使用构造方法注入的方式，注入到了 TestModule 中，同时 TestModule 也定义了 MySession 的绑定。之后通过 testng.xml 中的 parent-module 关键字设置 ParentModule 类，这样启用了对 TestModule 的注入。之后的依赖注入使用中，测试类 TestClass 有两

个注入：MyService 的注入来源于 ParentModule 的贡献；MySession 的注入来源于 TestModule 的贡献。这样可以保证所有测试获取的 session 示例是一致的，同时 MyContextImpl 对象在测试套件中只创建一次，这样我们可以将该测试套件的全部测试配置在一个公共的测试环境中。

三、总结

依赖注入的概念晦涩难懂，但对于 TestNG 的使用者，我们完全不需要过于关心依赖注入的细节，这也是 TestNG 提供依赖注入特性的初衷。我们可以不知道什么是依赖注入，就可以很容易的在注解方法内注入我们需要的对象，但前提是这些对象是 TestNG 承诺在此可以注入的；同时，为了更好的扩展性，TestNG 还支持 Guice 框架进行依赖注入的增强。后者就需要我们对依赖注入有一定的理解了。

总体而言，对于大部分场景，原始的依赖注入已经功能非常强大了，所以学会查阅 TestNG 原始依赖注入表格是关键。当在测试类之间需要使用依赖注入解耦，就需要增加自定义的依赖注入机制，使用 TestNG 本身支持的 Guice 是不错的选择。

本文使用的 TestNG 版本：6.9.9。

参考文献

1.https://testng.org/doc/documentation-main.html#dependency-injection
2.https://en.wikipedia.org/wiki/Dependency_injection