作者：陈雷 (Jackei)

Blog：http://jackei.cnblogs.com

鉴于这个系列写的内容是希望帮助“大多数 2-3 年工作经验、急切盼望提升自身能力的 tester 找到捅破‘测试自动化’窗户纸的办法”，所以木有高深内容，高手们请直接飘过，呵呵。

1. “关键字驱动”的由来

说到“关键字驱动”和“测试自动化”，就不能不提到 Mosley Daniel 的《软件测试自动化》一书，这本书 03 年引入国内，04 年市面上开始有卖，书中有两个相信能吸引到很多 tester 的话题：

（1）脚本应该录制还是编写？——想知道答案的自己下载电子书看吧：）

（2）“数据驱动”和“关键字驱动”。

彼时的我，刚刚经历了一次不成功的自动化实践，虽然 Rational Robot 提供了类似 UI 库管理，数据池管理之类的强大功能，但是痛苦依然。对测试自动化理解的不够深入，不知道该如何应对 RAD 模式下 UI 和业务快速调整，以及对 C/S 下非标准控件的识别等问题，导致了无法快速维护脚本、replay 次数不够，回放通过率不够，最后的结果是 ROI 无法满足要求，自动化项目宣告结束。

带着很多疑问的我原本想从那本书中找到些答案，遗憾的是那时功力实在太差，居然没有看懂，唯一留下印象的就是作者在 80 年代就开始探索自动化回归的技术，并且在 90 年代已经尝试了“数据驱动”和“关键字驱动”的技术，想来当时 Robot framework 之类的都还没有出现，所以我相信“关键字驱动”的技术源自这书的作者和他的朋友们。

2. 从“数据驱动”到“关键字驱动”

所谓的数据驱动，原本没有什么特别的，无非就是把 hard code 在脚本中的数据参数化出来，之所以算是 Robot、WinRunner 甚至 QTP 时代测试工具的卖点，其实主要是因为那个年代大多数 system tester 不懂开发，总需要有个功能来帮助自己完成参数抽取、数据维护、自动替换之类的功能。

而关键字驱动，则进一步在技术上把 tester 分成了完全不懂技术的和懂点技术的，前者只能根据格式填写一下 excel 表格，后者对工具/框架内置的所谓关键字库进行增补或二次开发。

找些例子来看看吧。

（1）简单的数据驱动。

如下面代码，定义了一个 class 并实例化了几个对象，用来存放不同的用户登录信息；而在负责完成登录操作的 do_login_as() 方法中，把 send_keys 原来向表单中填充的数据参数化，根据每次调用 do_login_as() 方法时传入的不同对象来实现用不同帐号登录的效果——虽然我对以“登录”为样例介绍自动化测试脚本深恶痛绝，不过这里为了表述简单，还是用了。（下面的代码只是一个示例，不是直接用来运行的。）

```python
class Account(object):

    def __init__(self, username='', password=''):
        self.username = username
        self.password = password

def do_login_as(user_info):
    browser.find_element_by_id("username").send_keys(user_info.username)
    browser.find_element_by_id("password").send_keys(user_info.password)
    browser.find_element_by_id("submit").click

big_boss = Account(username='admin', password='123')
guset = Account(username='guset', password='321')

do_login_as(big_boss)
do_login_as(guset)
```

其实数据驱动本来也没有什么技术含量，写过代码的谁还不知道什么是变量啊？不过在早期的商业工具中，的确把这个作为了一个卖点，毕竟 10 年前的测试工具设计思路也与现在不同。早期的工具始终都假设”系统测试工程师不懂开发“，所以他们在开发测试脚本时需要借助类似录制回放+编辑调试的模式；另外，对于数据驱动所需要的测试数据，最好也是通过工具内置的 data pool 管理，通过表格编辑，甚至读取 csv、excel 文件之类的。如果我们依照这个思路去实现自己的测试框架，那肯定是商业工具很有价值，毕竟自己实现 data pool 管理、外部数据文件读取之类的功能，代码量也不小，要处理好那些数据格式和内容校验之类的，貌似跟我们所需要完成的自动化也没啥太大关系。

可问题在于，为什么一定要有 data pool+外部数据文件来实现”数据驱动“呢？

（2）传统的“关键字驱动”

还是先用个例子来看看传统的“关键字驱动”长什么样子。

New Test Case		
open	www.google.com	
type	q	关键字驱动
clickAndWait	btnG	
verifyTextPresent	关键字驱动	

上面是一个 Selenium 0.x 时代 Core 模式下（什么是 selenium 的 core 模式和 RC 模式请自行 google，不过话说现在已经全民 webdriver 时代了，不知道也就不知道吧）的例子（QTP 的实现效果也类似），简单说就是第一行是 test case name，下面 3 列开始进入正题，第一列表示你想干啥，第二列是被操作对象是谁，第三列是你对它干了啥。06 年刚刚开始尝试 web 测试自动化的时候，我们一度认为这是一个比之前的测试工具要智能的多的东东，不过在尝试了很长一段时间之后，还是发现了这种模式下的问题，又逐渐转回了编写脚本的方式。

对比上面的第一个脚本，可以看出关键字驱动进一步屏蔽了底层的实现细节，例如，你只需要 clickAndWait 那个 btnG，而不需要知道 btnG 到底是个啥，以及它在哪里，你只需要填写表格。等你把一个个 test case 变成了一个个表格，把一个个 step 变成了一行行表格中的内容，基本上你的自动化测试就搞完了。——真的是这样吗？

现在回想一下，关键字驱动之所以会出现，可能初衷还是为了降低自动化实施的门槛——因为 tester 都不太懂开发嘛，所以开发能力强的人把框架实现出来，原来那些只会写 excel 的 system tester 填写表格就可以了。也许现在还有很多公司会倾向于这个方案，美其名曰“分工协作，人尽其能”。不过关于这个问

题，暂时先不展开讨论，先看看这种传统的关键字驱动模式会遇到什么问题。

首先，页面对象的识别问题。这是任何一种基于 UI 实现回归测试自动化的框架都会遇到的问题。在 C/S 时代，就已经出现了很多定制化 UI 组件难以被工具识别的问题，可好歹总逃不出 windows 的手心。到了 web 时代，前端实现技术百花齐放，而前端代码的编写也更是一个开发人员一个习惯，如果缺少统一的编码规范，对于那些没有使用 id 或者 name 之类属性的页面对象，传统的关键字驱动框架就没有例子中看起来那么美好了。当然，有人说 xpath 可以解决一切问题。嗯，xpath 是很强，但是一个没有开发经验的 tester 想掌握它其实也不会比学会一点基本的编码技术容易多少；另外，xpath 并不是万能的，在实际中还是有些它处理不了的情况。下面再给个例子（如果 tester 看不懂这个例子，估计学会 xpath 也有点困难）：

上面这个图中是一个表格，id 列（第一列）和需求名称列（第二列）下显示的内容，都是可以点击的超链接，最后的“操作”一列中的一个个小图标也各自指向一个链接（分别是“变更”、“评审”、“编辑”、“建用例”），通过查看第一行记录的 html 代码，我们发现：

```
<tr class="a-center" style="background-color: rgb(255, 255, 255); background-position: initial initial;
background-repeat: initial initial; ">
    <td>
        <input type="checkbox" name="storyIDList[341]" value="341">
        <a href="/story-view-341.html">341</a>
    </td>
    <td></td>
    <td class="a-left nobr" title="登录不成功">
        <nobr>
            <a href="/story-view-341.html">登录不成功</a>
        </nobr>
    </td>
    <td class="a-right">
        <a href="/story-change-341.html" target="" class="icon-green-story-change" title="变更"> </a>
        <a href="/story-review-341.html" target="" class="icon-green-story-review" title="评审"> </a>
        <a href="/story-edit-341.html" target="" class="icon-green-common-edit" title="编辑"> </a>
        <a href="/testcase-create-8-0--0-341.html" target="" class="icon-green-testcase-createCase" title="建用例"
        > </a>
    </td>
</tr>
```

1) 假如想要编辑一条记录，没有可利用的 id 或 name 属性，唯一可用的是 link；

2) link 指向的 url 中包含了这条记录的 ID 信息，因为这里是第一行记录的代码，所以都是 341，而后面的每一行记录的代码都是各自的 ID。

上面这个例子是企业应用中常见的场景，关键问题在于数据记录 ID 是一个不可控因素，而数据记录的 title/name 之类的是可控的，为了提高 test case 的可维护性和相互独立，我们肯定不会依赖 ID 去模拟页面操作，而是根据 title/name 之类取回 ID 信息，再拼装回所需要操作的链接。这种情况下，xpath 恐怕也搞不定了。（如果这里再涉及到要在几个 iframe 之间跳来跳去，恐怕写脚本的人就要崩溃了。）

当然，有人会说关键字驱动框架一般也可以定义 function 来实现类似的操作啊。唉，都到了编写自定义 function 的地步了，干嘛还非要纠缠在关键字驱动上啊。

第二，脚本的可维护性问题。如一开始的例子，传统的关键字驱动是一种纯“面向过程”的脚本组织方式（就像 C/pascal），表格中填写的是一个操作序列。如果多个 test case 都涉及到某个页面，基本上就会在多个 case 中都看到类似 clickAndWait btnG 这样的内容，而一旦页面中 btnG 改名叫 buttonG 了，或者

clickAndWait btnG 与 verifyTextPresent 之间增加了一个 clickAndWait XXX 的 step，那基本上每个 case 都需要修改。

嗯，问题来了，当你的脚本数量从 1 增长到 100、1000 的时候，当 UI 的变动无法避免的时候，当你发现 100 个 case 回归执行只有 90 个通过，执行失败的 10 个需要逐个检查错误日志和查看截图，再挨个修改的时候，当下次回归测试又发生这种问题的时候——基本上这就是一个死循环了。如果解决不了根本问题，前期投资可以舍弃了，别纠结了。

当然，有人说关键字驱动已经进化了，可以跟最新的 webdriver 结合起来，提升关键字的封装层次，解决这个可维护性的问题。好吧，这个问题等下再详细说。

第三，脚本的可扩展性问题。在大规模实施自动化的过程中，脚本一般都会简单的是一行行的 browser.find_elmenet_by_id(xxxx).click() 这样的模式，根据各种条件来判断执行的分支，进行各种异常处理，第三方类库/包的调用，主机环境的访问，诸如此类，这些对于所有的 3GL/4GL 来说其实都很容易实现，但对于传统的关键字驱动来说，嗯，也可以实现，大概是下面这个样子【摘自 robot framework（此 robot 非 rational robot）】：

下面是一个在 keyword 表格里面实现的 FOR 循环：

:FOR	${var}	IN	@{SOME LIST}
	Run Keyword If	'${var}' == 'EXIT'	Exit For Loop
	Do Something	${var}	

这是获取时间，当然也是写在表格里面的：

${time} =	Get Time			
${secs} =	Get Time	epoch		
${year} =	Get Time	return year		
${yyyy}	${mm}	${dd} =	Get Time	year,month,day
@{time} =	Get Time	year month day hour min sec		
${y}	${s} =	Get Time	seconds and year	

import 外部的库:

Import Library	MyLibrary			
Import Library	${CURDIR}/Library.py	some	args	
Import Library	${CURDIR}/../libs/Lib.java	arg	WITH NAME	JavaLib

读取外部文件:

Import Resource	${CURDIR}/resource.txt
Import Resource	${CURDIR}/../resources/resource.html

正则表达式:

${escaped} =	Regexp Escape	${original}
@{strings} =	Regexp Escape	@{strings}

重复执行 5 次 goto pre page 操作:

Repeat Keyword	5 times	Goto Previous Page		
Repeat Keyword	${var}	Some Keyword	arg1	arg2

执行一个 keyword 然后期待能捕获一个异常:

Run Keyword And Expect Error	My error	Some Keyword	arg1	arg2

${msg} =	Run Keyword And Expect Error	*	My KW	
Should Start With	${msg}	Once upon a time in		

有人可能会说“你看，关键字驱动框架也可以扩展的很强大啊！”。是，在 programming 的世界中，没有什么不能做的，不过都弄到这个份儿上了，学习这一套东西跟学习一个标准的编程语言还有什么差别吗？先不说这样的框架越扩展越难维护，可靠性也就越差，单单这些关键字的用途被局限在自己的框架中，你所积累的知识和经验无法重用到其他测试代码的编写中这一个理由，就应该彻底放弃这种方式了。

如果要说的直白一些，传统的关键字驱动框架的时代在前几年就已经开始远去（是 had been，不是 have been），我们感谢上一代 tester 的努力探索和实践，但最终历史证明这是一个不算成功的尝试，一个框架如果不具备开放性，一切都自给自足，那么有一天这也会成为限制自己发展的最大原因。

（3）穿马甲的“关键字驱动”

时代在进步，关键字驱动也在进步，这个领域中的代表 robot framework（此 robot 非 rational robot）也在进步，于是，test case 变成了下面这个样子。

Test Case	Action	Argument	Argument
User can create an account and log in	Create Valid User	fred	P4ssw0rd
	Attempt to Login with Credentials	fred	P4ssw0rd
	Status Should Be	Logged In	
User cannot log in with bad password	Create Valid User	betty	P4ssw0rd
	Attempt to Login with Credentials	betty	wrong
	Status Should Be	Access Denied	

依旧不变的是“表格”，改变的是填写方式——其实这背后的，是关键字定义终于被开放出来，tester 可以自己定义 keyword 然后“注册”到框架中，而那些依然没有学会基本编程技能的 tester，继续用这些 keyword 重复上个时代的事情——填写表格。

其实相对于最初对关键字驱动的定义，这个真的已经不是关键字驱动了，如果非说它是，那么只能说上个时代的关键字驱动中，test case 表格的每行都是一个页面操作，而“新的”关键字驱动中，test case 的每行都已经是一个完整的业务操作，以上面的“Create Valid User” step 为例，robot framework 希望的实现方式是 tester 通过 python 等 4GL 实现一个同名的 function，这个 function 接受两个参数，分别是“fred”和“P4ssw0rd”，再把这个 function 注册到 robot framework 中。而“Create Valid User”内部的实现，可以类似于一开始“数据驱动”中的那个例子，充分利用 4GL 的特性和已有的其他第三方组件（例如 webdriver），来实现各种复杂的基于 UI 的操作，这样也就解决了刚刚“传统的关键字驱动”所遇到的问题。

最后，当完成了这个 function 的开发并在 robot framework 中注册后，做手工测试的 system tester 就可以很容易的把原本 excel 中的一个个 case 转变为自动化脚本了。

其实这个思路有它的优点，例如：通过分工协作降低实施门槛，可以一开始就编写符合 robot 所需格式的 manual test case，等到 keyword 开完全了以后这些 case 就可以直接导入执行了；不再自给自足，而是保持一定的开放，并利用其他第三方组件的特性。这样很大程度上解决了自动化项目实施遇到的人员能力问题和可维护性、可扩展性的问题。

另外，新的关键字驱动还有一个更加先进的“近亲”BDD 作为参照，很容易把它的一些实践也一起融合进来。

一切看起来都很美好，不过问题也还是有。

1) 表格化的 test case 毕竟不同于编写代码，调试就变成了一个问题，如果写错了关键字的一个字母，要及时发现并定位到问题就不那么容易。当然，可以再开发一个 web 平台，让编写 case 的人仅能从一个 list 中选择已经定义好的 keyword，不过这个成本恐怕就不是一般研发团队能承受的了。
2) 作为一个软件，易用性和复杂度总是成反比的，当框架提供了方便的表格化编写 case 功能时，也相对的增加了底层的复杂度（虽然没看过 robot framework 的代码，但是相信底层代码的分层也应该比较复杂），对于想要真的掌握框架的团队来说，无形中增加了一道门槛。另外，复杂度与可扩展性也是成反比的，就像我可以用木头做一辆车，也可以把木头车拆了做些别的东西，但是我没法把一辆汽车拆了弄成别的东西——前两年广东美院那位把解放卡车拆了做成关公像的牛人除外。当然，最终实施自动化时到底如何进行框架选型，就要团队自己在易用性/复杂度/可扩展性上进行评估了。
3) 把 excel 里面的 manual test case 通过新的关键字驱动直接变成可执行的脚本是最好的方法吗？这似乎只是一个传统 system test 的惯性思维在作怪，为什么没看过开发人员把 unit test 也写到一个个表格里面？为什么 manual test case 就一定要先写在 excel 里面，而不是一开始就是代码？
4) 如果仅仅是考虑把 step 组装起来，再把 case 组织成 suite 执行，其实代码实现上可以说毫无技术含量，但是对于一个没有开发经验的 tester 来说，这毕竟是一个跟 coding 简单亲密接触的机会，可以让 tester 从低难度的代码开始培养兴趣和信息。而 keyword，无论新的还是旧的，却剥夺了这个机会；当 tester 希望学习框架的时候，会发现表格的层面跟下层框架之间的不是楼梯，而是一道沟。

3. “关键字驱动”的未来

我们如今所处的环境总是在变化着，今天与 10 年前相比，最大的变化就是测试行业获得了极大的发展，大多数企业都认可了测试工作的重要性，并且开始思考如何提升测试工作自身的质量和效率，而且不同规模的企业都在探索着合适自己的研发流程和技术；而 tester 们的技术能力也在不断增强，至少能写代码的人比 5 年前多了很多。当然，还要感谢开源世界带来的众多框架、组件，让自动化的门槛不断降低。

就像传统的关键字驱动已经远去一样，新的关键字驱动未来会如何，大家讨论吧。:-)

接下来，准备讨论下“Page object model”技术，或者如何构建一个轻巧但功能完备的 web 测试框架。