目 录 (下 篇)

解读日常工作中开发和测试的关系

◆ 作者：小鱼儿

摘要：本文是自己在日常测试中在处理和开发关系的一点感知和想法，结合了自己工作当中的一点经历，希望和大家共同分享。

“小冤家，你干嘛，像个傻瓜？我问话，为什么，你不回答，

你说过，爱着我，是真是假，说清楚，讲明白，不许装傻”

大家应该都能唱这首歌吧，这是电视剧《情深深雨蒙蒙》里面赵薇的歌词，当时赵薇唱的时候感觉真的是活泼又有趣，我今天也把自己作为测试人员在与开发沟通过程中的有趣小事叨唠叨唠。

以前去面试的时候不知道面试官是一种惯性思维还是真的想知道这方面的答案，面试过程中总会有这样一个问题：

在你以往的工作当中你有没有与开发就某一个问题存在分歧甚至是争吵，这时候你是怎么解决的或者你觉得自己处理的比较好的经验有没有？

这时候我该怎么回答呢？总不能说我没有遇到过，我这个测试人儿和开发相处的很愉快!（O(∩_∩)O 哈哈~……）

一般我的官方回答大致如下：

“这种其情况其实经常会遇到，但是首先我们得摆正一个态度:大家的共同目标都是一样的，都是为了产品，为了客户;在这个前提下，首先认真倾听开发的想法，找出开发和自己的出路，然后将自己的想法合情合理的用一种开发比较容易接受的方式表述出来，当然原则是必须要坚持的”。

其实细细回想自己几年的工作经历，那些和开发头疼脑热的对话，针尖对麦芒的沟通，回想着和开发的相处模式，然后和上面的官方回答一对比，突然有种偷笑的冲动，

想着想着思绪就涌出来……

我的第一份工作（3 年）：系统整合测试工程师（SIT）

系统测试的东西会比较的广而大，要测试硬件也会测试软件，系统是连接硬件与软件的载体，没有操作系统那也就没什么软体供用户使用了（当然这里排除那种牛逼的程序员可以命令行操作了），硬件搭载在系统下供普通用户使用，所以当时我所在的那个部门接触最多的开发部门或者可以说叫开发吧，当时来说应该是：HW 和 BIOS,这 2 个部门即为 Hardware 和 Basic input and output System,如果有在代工厂或其他类似有这种性质公司呆过的同僚应该理解起来会比较的容易，HW 简单来说就比较偏硬件了，BIOS 就是管程序输入输出的，反正就是 2 个比较牛的部门（哈哈，希望听到我这样说工作在这 2 个部门的同僚会比较高兴，下回见到我手下留情一点点……）.

当时只知道每天沉浸在无数片服务器主板的拆装中，一般我们系统部门会有自己一套常规的测试用例，供我们日常项目的测试使用，大致就是基本的功能测试，性能测试，兼容性测试这几块。项目前期最经常碰到的问题就是开不了机或者即使开机了还是没法看到系统里面的东西，或者是 Reboot 跑不过，或者是网速上不去，或者是系统读不全硬件部件等等，找的最多的部门就是 HW 和 BIOS，HW 经常会把里面一片 cpld 的芯片抠出来拿去他们那里检测，BIOS 部门也会去读 BIOS 芯片，当然这只是比较大的问题才会直接去动芯片里面的东西，一般的问题 HW 和 BIOS 看一下或者直接刷个分位也就解决了，可能这中间开发测试的分界线并不是很明显，大家的矛盾冲突不是很直接，项目前期测试问题很多，部门直接接触比较频繁，有时候会有点尴尬，怎么又要跑到别的部门去，哈哈……要是那个部门帅哥美女比较多还愿意去一点，要是……还真是不愿意烦他们，有时候大家一天打交道多了，真是中午吃饭都不愿意碰到，真怕影响食欲，因为在食堂坐在一起吃饭肯定又是围绕那个 Bug 讨论了，就感觉边吃饭边开会一样……唉…吃饭都不让人好好吃了。

那段时间对于我们来说最麻烦的就是自己定位问题,然后去找相关的部门解决，我们当时还只是普通的 tester，上面会有 PL(Project leader)，自己发现的问题首先需要定位清楚是什么导致的，然后反复确认这个问题的确存在，同时仔细验证在同机型上面是否存在同样的问题，还是只有这个机型才有这个问题，一般写 bug 的时候要将这些对比写清楚，以便 PL 确认，定位清楚找谁解决，供后面修复的人员分析，有时候 PL 很忙没有时间弄清楚你的问题，你还需要耐心和 PL 解释清楚你发现的 bug 产生的过程，你还得自

己控制自己工作 plan，确保自己能在规定的有时间测完自己的任务。问题来了，有时候你自己定位了问题，HW 和 BIOS 可能没时间解决，就会相互推脱，推来推去，来回折腾，要是测试人员是实习生，哎，人微言轻(此处省略一百字……)

第一段经历让我明白了，做事情需要有条理，同时自己的工作自己需要有个规划，和其他部门的沟通自己如果实在人微言轻就需要寻找中间媒人，这个媒人可能是你的 leader，可能是你的 boss，也可能是你在公司玩的比较好的兄弟姐妹，当然前提还是你自己得条分缕析，别让帮你的人为难，给老大增加烦恼，同时掌握沟通的艺术。本着将事情理清楚弄明白的原则，本着解决问题的态度---心要细，嘴要甜的去解决问题!

我的第二份工作（1 年）：软件测试工程师

作为系统整合测试工程师，使我接触了硬件也接触了软件，可是那将近 3 年的工作经历在我后来面试其他公司的时候被定位为硬件测试工程师，说句实话我自己都不是很明白，难不成是我的简历描述的让大家这样误解了，反正那段经历在我后来找第二份工作的时候实在……硬件测试的标签让我四处碰壁，最后好不容易找到了一份软件测试的工作，这正是我现在从事的工作。

接触第二份工作之后我才正式接触流程，懂得了测试开发明确的分界线，也知道了什么是 kick off，case review，code review，QA……并且自己也切切实实的参与其中，这中间和开发的矛盾才显现的比较明显。

就来讲讲中间的趣事一二件吧。

进入这家公司是以 PL 的角色，但是这个 PL 很虚，主要是下面没有人，自己既是 pl 也是 tester，自己的项目自己测试，如果项目忙自己弄不过来，老大才会给我这个项目派个人，这时候才会感觉自己是 PL（呜呜呜~~~~(>_<)~~~~ ）。

25:8 条 bug-Close with “not defect”

记得测试第一个项目的时候，上了 25 条 bug，并且这其中有 8 条都回答成 not defect，当时自己部门课长很是惊讶，产出 bug 量这样大，还不是修复的，not defect? 什么鬼？真的是用惊讶来形容，立马就组织开了个会，会上就直接说这次的 bug 都有哪些，并且让我一个个解释现象，报 bug 的原因，还好自己的专业素养在那，不慌不忙，娓娓道来，后来老板发现大部分的问题都是开发人员的不仔细，工作马虎导致的。

会议结束，开发部门负责这个项目的暂时就称呼她为 N 姐就过来，一来那就是一顿

指责啊，说什么我乱报 bug，什么问题都往上面报等等，反正就是枪药味十足，说句实话当时真是被她的气势吓住了，我也是工作了小 3 年，真真的是没有见过这样吵架的，当时一句话说不出来，唉，那感觉就像，就像那啥……后来我老大和我说：“N 姐那人就那样，不用理会她，保证自己测试的没有问题，上的 bug 的确是存在的，只有自己没有错误那就不用怕！”现在想想真是感激!!! 当然这是其中见到的比较奇葩的。不能代表其他开发人员的素养!

监视器—>项目 delay

最近开发自己设计了个监视器，这个监视器用来监控内部服务器状态，并且定时发邮件给指定人员，这个项目年前就开始了，是个内部的项目，刚刚开始就一拖在拖，后来好不容易发版本了，可是基本功能一直没有实现，然后开发就说“就是这样的，是不是你测试的有问题啊，要不你在测试一遍我看看，我自己测试都是好的……”搞到后来开发测试都没有继续下去的信心了，其他项目也要人力测试，这个项目感觉遥遥无期，人力投入和实际产出不成正比，有点入不敷出，可是开发那边最终貌似找到原因了，想发 Final2 了，大致意思就是我发了总归会有人测的。反正这种事情就是比较头疼，schedule 只能 delay，只能说计划赶不上变化，人员也只能重新安排。

将近一年的软件测试使我懂得，将其中的有争议的原因弄明白，毕竟开发知道代码内部的东西，了解白盒，了解程序内部的具体实现，理解了开发的想法，找出了这其中的不同，同时明白自己发现的 bug 产生的原因，自然就知道怎样去说服开发解决自己报的问题了。很多时候最害怕的还是自己不理解原因，发现了问题，这只是表象，开发觉得你不懂他的代码就只知道指手画脚，让他们改来改去，他们经常不耐烦的回你，“本来就是这样的……”但是如果你耐心去听开发人员的心声和想法，然后用他们能够接受的方式去表达你想说的，这样他们也会愿意和你沟通，这样你既理解了开发想法，也懂得自己的测试，这样一来事情不就好解决啦!

开发与测试怎么说呢，就好比一对小夫妻，一对小冤家，这关系如同夫妻一般微妙，也如同夫妻关系一般让人头疼，他们共同为了一个家庭的和睦和发展，但是他们也会有分歧和真吵，分工不同，目标一致，掌握二者的艺术，和谐自然来。

接口测试之 Jmeter 数据+关键字驱动实践

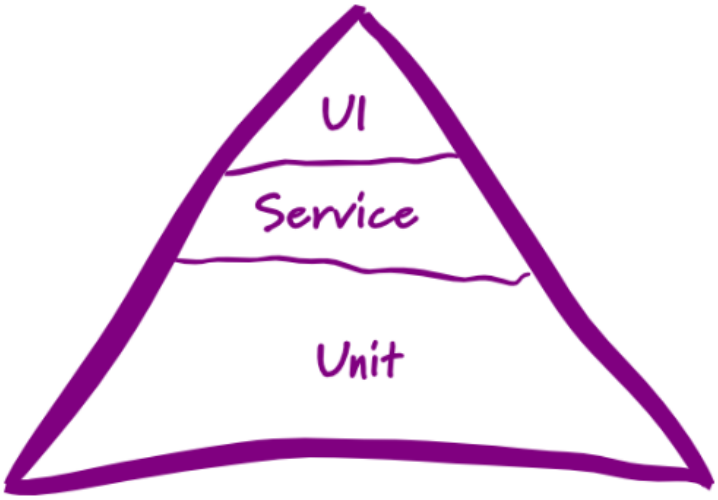
◆作者：宫磊

本篇文章介绍利用 Jmeter 实现数据+关键字驱动的一种接口测试方案，主要想和大家讨论接口测试用例如何设计及使用何种测试工具或方法才能更有效的进行接口测试。

什么是接口测试?

我们这里说的接口是指程序之间提供服务的软件接口。接口测试是测试系统组件间接口的一种测试，主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。测试的重点是要检查数据的交换，传递和控制管理过程，以及系统间的相互逻辑依赖关系等。

那为什么进行接口测试呢？大家可能听说过测试金子塔，最底层是单元测试，中间层是接口测试，最上层是 UI 测试。相对于 UI 测试，接口测试能更早的发现问题，它比 UI 测试粒度更细，更能发现底层问题，它发现和解决问题的效费比更高，所以进行接口测试是有必要的。



数据驱动框架:

利用 Jmeter 建立数据驱动测试是比较简单的，首先使用 Jmeter 根据自己业务建立

测试计划.然后将测试计划中所有业务流程的“Hard Code”测试数据，全部参数化实现。在这里我们使用 Jmeter 中“CSV Data Set Config ”配置元件从 CSV 文件中获取数据。

举例：假如我们有一个接口，输入一本书的 ID 信息，返回书名及这本书的价格:

HTTP 请求的数据:

```
{
 "bookid": 1
}
```

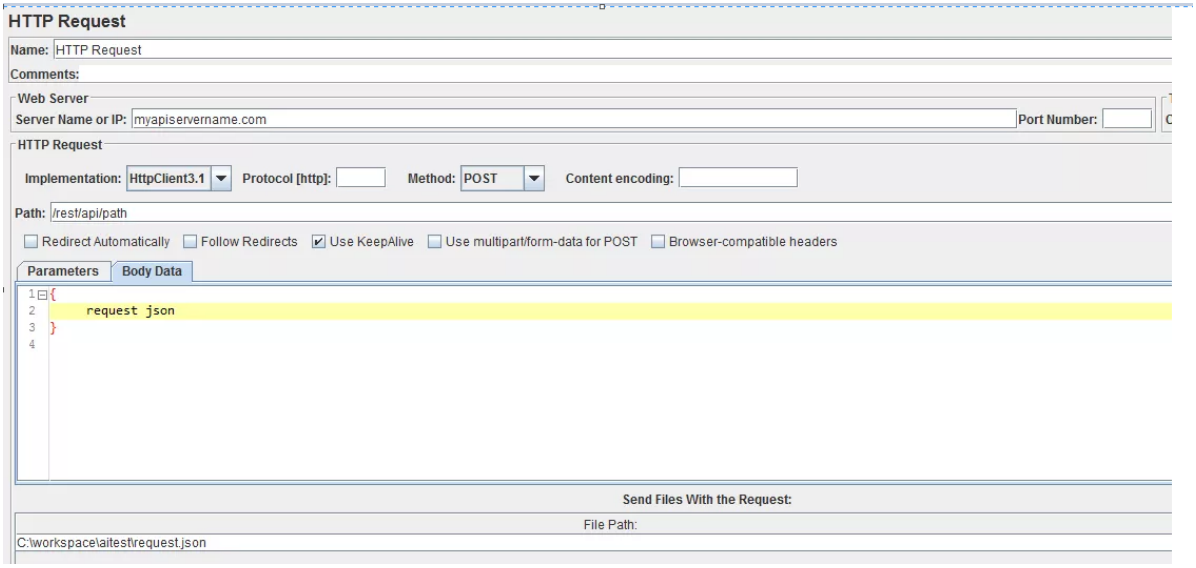
服务器返回的响应数据:

```
{
    "bookname": "Jmeter"
    "bookprice": 23.5
}
```

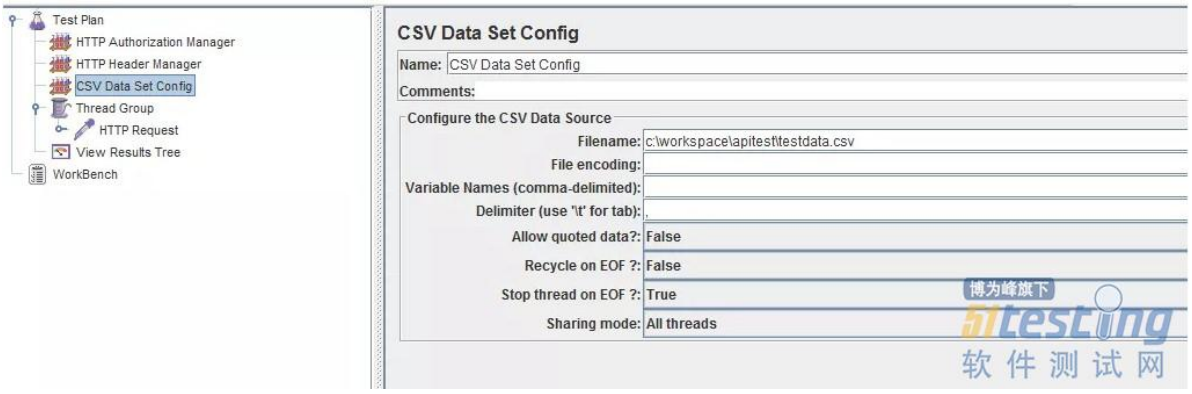
那如果我们想用不同的数据测试 API 的话，那就需要对请求数据进行参数化设置。

Jmeter 实现步骤如下:

1、首先在测试计划中添加一个线程组－然后右击线程组添加一个 Sampler (HTTP 请求)



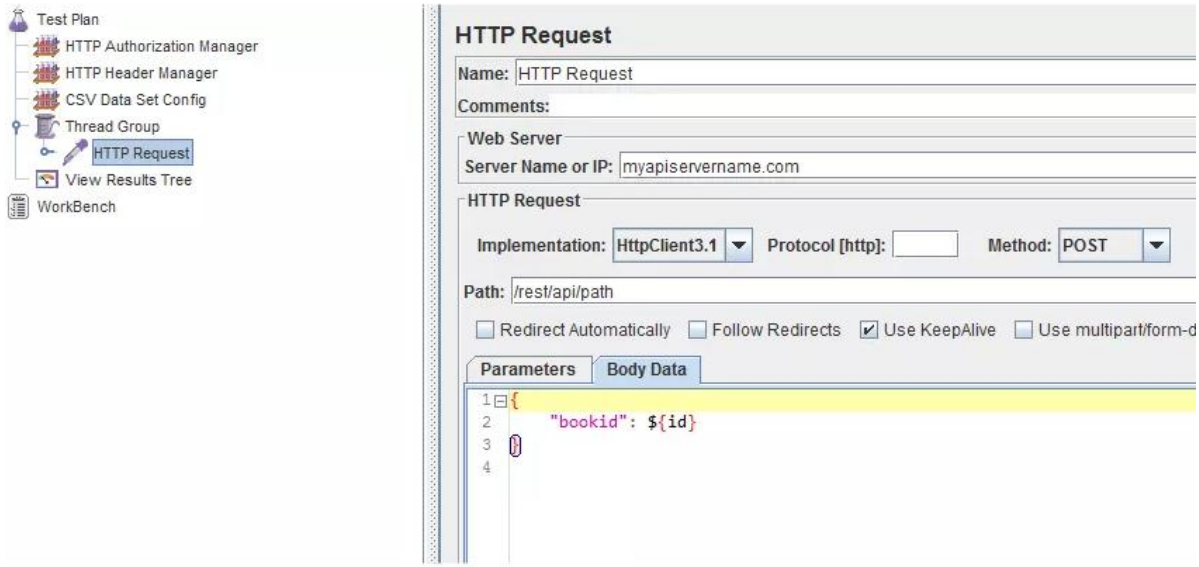
2、再建议一个 CSV Data Set Config 配置元件如下图:



CSV 文件里数据格式如下:

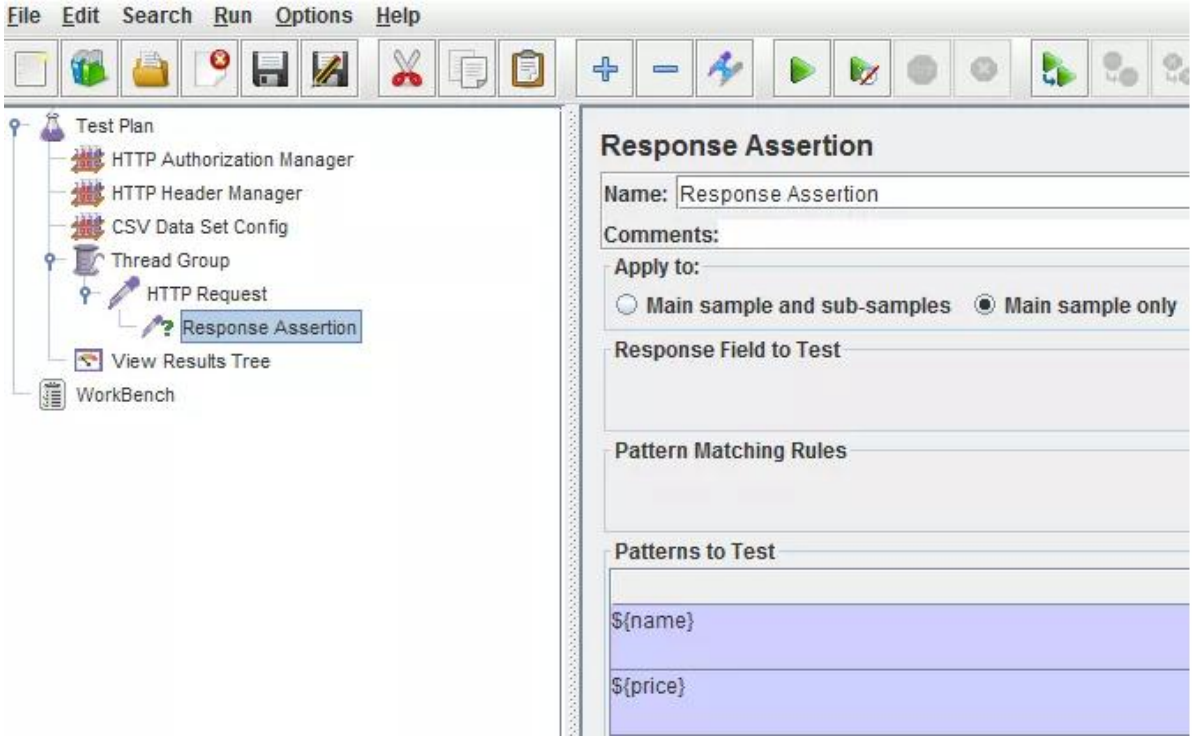
	A	B	C	D	E
1	id	name	price		
2	1	Awesome Jmeter	23.5		
3	2	REST API	11		
4	3	Hello world	16		
5	4	Developer Soft Skills	33.5		
6	5	Hope to Die	22		
7	6	Divergent	34		
8					
9					
10					
11					
12					
13					
14					

3、参数化相应的请求数据：请 ID 号这时用变量代替



4、那请求后，我们还需要验证服务器返回的数据是否正确，那就应该再添加一个

断言机制。同时断言也需要进行参数化处理。



经过上面配置后，Jmeter 将会根据 CSV 文件里的数据依次发 6 个 HTTP 请求到服务器。

关键字驱动框架:

主要思路是可以为每一个关键业务流程建立一个关键字。Jmeter 利用关键字分别调用不同的业务流程，从而实现对相应的业务逻辑或接口的测试。

举例：假如我们有一个订飞机票的 Web 程序，在这个程序里用户可以注册一个账户，然后利用这个账户去订票，查看相应的飞机票及取消订票等操作。所以我们可以根据上面需求为下面不同的接口定义一个关键字，每个关键字实现了相应的功能。

- Register New User
- Login
- Logout
- Book Ticket
- Edit Ticket
- View Ticket

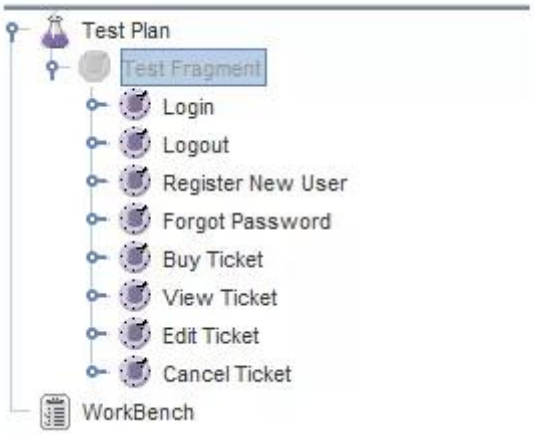
- Cancel Ticket
- Forgot Password

下面我们就可以设计相应的测试用例，让 Jmeter 直接读取这些测试用例的关键字，从而实现对相应的业务流程覆盖。如下图用例 TC001，需要注册一个账户并用此账户订票后退出，那 Jmeter 就需要调用 Register New User,Book Ticket,Logout 三个关键字从而实现了对此用例的执行。

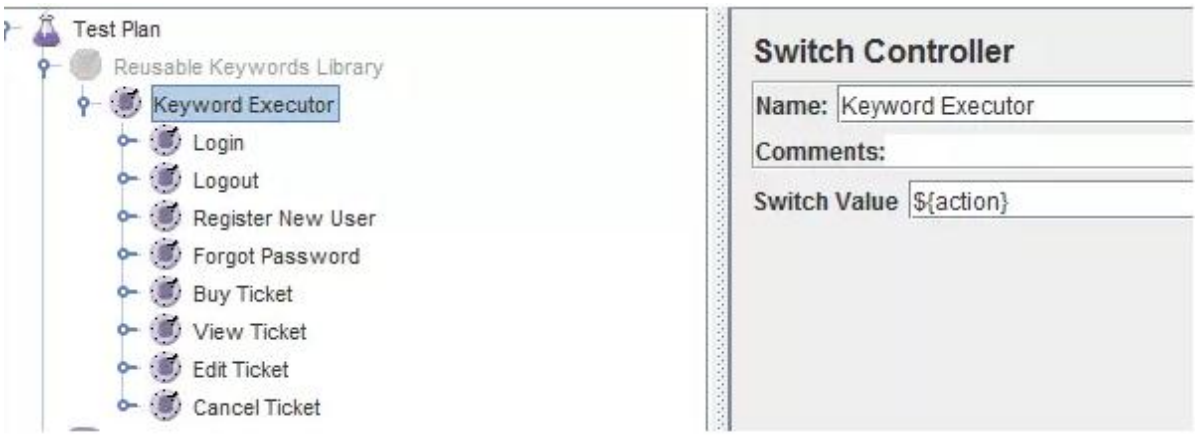
TestcaseID	Description	Keywords to be called
TC001	Check for successful creation of a new account by booking a ticket	Register New User Book Ticket Logout
TC002	Login as an existing user & view the ticket details	Login View Ticket Logout
TC003	Login as an existing user & book a new ticket using VISA CC	Login Book Ticket Logout
TC004	Login as an existing user & book a new ticket using MasterCard CC	Login Book Ticket Logout
TC005	Login as an existing user & book a new ticket using Discover CC	Login Book Ticket Logout
TC006	Login as an existing user & book a new ticket using AMEX CC	Login Book Ticket Logout
TC007	Login as existing user, book a new ticket & edit	Login Book Ticket Edit ticket Logout
TC008	Login as an existing user & cancel an existing ticket	Login Book Ticket Cancel ticket Logout

Jmeter 实现步骤如下:

1、首先设计相应的可复用的模块：我们可以将每个关键的业务流程都单独编成一个 Sampler，Sampler 名字取业务流的名字，然后放在 Test Fragments 下面。



因为每个业务流都会通过“关键字”随机的被调用，所以在这里建立一个 Switch Controller，通过向“Switch Controller”传递不同的业务名做为“关键字”，从而实现对不同的业务接口的调用。

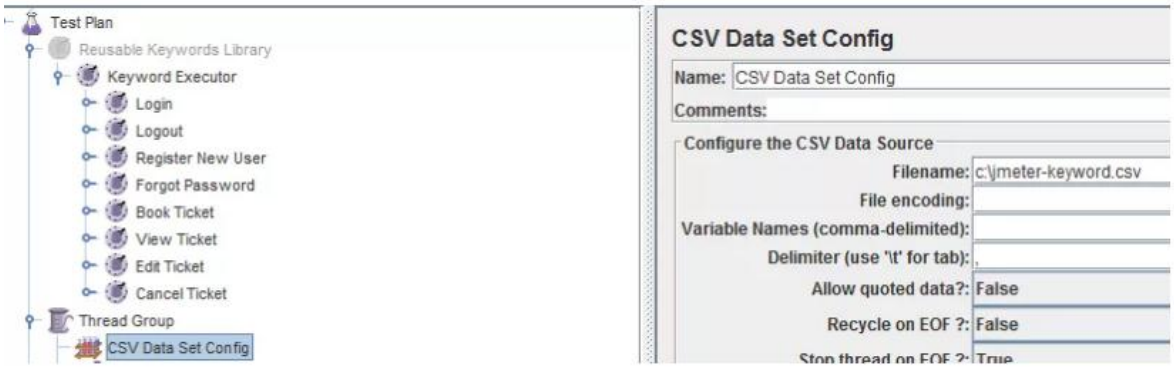


2 建立测试脚本：前面我们已经建立了可复用的模块。现在我们需要根据业务流程编写相应的测试用例。每个测试用例通过组织不同的关键字，实现一个业务流。Jmeter 会调用相应的测试用例，解析关键字后再去调用相应的 Sampler，从而实现对业务流程接口测试。另外，如果后期想增加什么业务流程，只需要修改相应的测试用例即可。

testcaseid	description	keywords	cc
TC001	Check for successful creation of a new account by booking a ticket	Register New User;Book Ticket;Logout	Promocode
TC002	Login as an existing user & view the ticket details	Login;View Ticket;Logout	
TC003	Login as an existing user & book a new ticket using VISA CC	Login;Book Ticket;Logout	VISA
TC004	Login as an existing user & book a new ticket using MasterCard CC	Login;Book Ticket;Logout	MasterCard
TC005	Login as an existing user & book a new ticket using Discover CC	Login;Book Ticket;Logout	Discover
TC006	Login as an existing user & book a new ticket using AMEX CC	Login;Book Ticket;Logout	AMEX
TC007	Login as existing user - book a new ticket & edit	Login;Book Ticket;Edit Ticket;Logout	
TC008	Login as an existing user & cancel an existing ticket	Login;Cancel Ticket;Logout	
TC009	Use Forgot password to reset credentials & Login	Forgot Password;Login;Logout	

3、建立相应的“关键字”解析引擎：

Jmeter 线程组会执行“关键字”解析引擎，所以建立一个 CSV Data Set Config 去读取相应的测试用例。

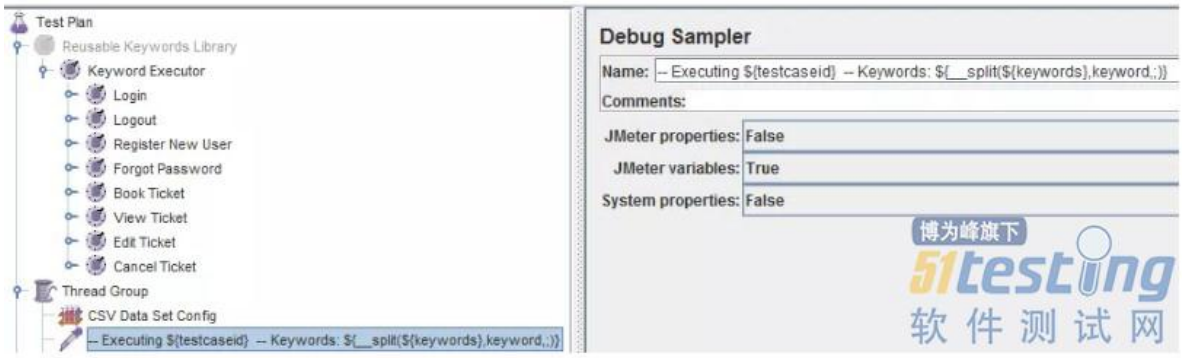


用例 1 中关键字有“Register New User ,Book Ticket,logut”，当从 CSV 文件中读取时，会存放在一个变量中，但这里面的每个关键字都会被执行，所以需要读取后分解他们并存储到不同的变量中，然后才能去调用相应的“Action”。在这里可以使用“Split”函数 ${__split(${keywords},keyword,;)}，使用这个函数将测试用例中的业务流“Register New User ,Book Ticket,logut”，分解如下：

keyword_1=Register New User

keyword_2=Book Ticket

keyword_3=Logout



然后增加一个 ForEach 控制器去循环每一个关键字。Module Controller 通过 SWitch Controller 去调用相应的关键字，从而实现相应的业务流。

接口测试小结:

根据上面的介绍大家很容易开发出一个基于“关键字”驱动的接口测试方案，我们可以为要测试的每个接口编写成一个 Sampler 放置于上面介绍的可重用的模板下，然后通过关键字解析引擎去调用相应的关键字，从而实现对相应的接口进行测试。

目前的关键是设计相应的接口测试用例。我的理解是应该为每个接口设计不同的用例，比如可以设计接口中的不同参数，正常的参数还有非法的参数等，另外还可以用不同的接口组合成不同的业务流进行用例设计。

这篇文章仅是对使用 Jmeter 测试接口的一种探讨。欢迎大家一些学习交流。

❖ 拓展学习

- Jmeter 性能及接口测试项目实战：http://www.atstudy.com/course/explore/Performance

当测试 APP 的消息推送通知时考虑用户体验性

◆ 译者：枫 叶

摘要:

一个信息发送应用的推送通知的功能性被看成是相同的不管被使用的设备或者操作系统。这篇文章讨论了如何测试这些推送包含了铭记设备和应用的不同状态，并且为什么功能配置审核员正如物理层的一样重要。

配置管理帮助我们证明和证实了软件和系统满足了共同的意图并且，更重要的是，我们的客户使用的需求。实体上的，配置管理的技术细节可以是相当有挑战性的，但是同样重要的是用户体验，一些配置管理分析师没有意识到是他们工作要求的一部分。在配置管理术语里，这被称为功能配置审核。

这篇文章分析信息发送应用表现在我们心爱的手持设备的方式。大多数我们每天使用发信息应用来交流，不管是以文本，照片，音频，或者视频，因此留心用户的体验是很重要的。

考虑应用、设备和操作系统的状态。

一个应用的推送通知被移动设备的操作系统便利化并被应用使用。当开发者为应用和它的通知创造代码时，他们使用特定操作系统内置的库去在代码里实现特性。所以不论他们是否在安卓，窗口操作系统，黑莓，或者苹果设备上运行，信息应用被设计成与推送连接为了实施让我们保持规律基础上更新的的功能性。

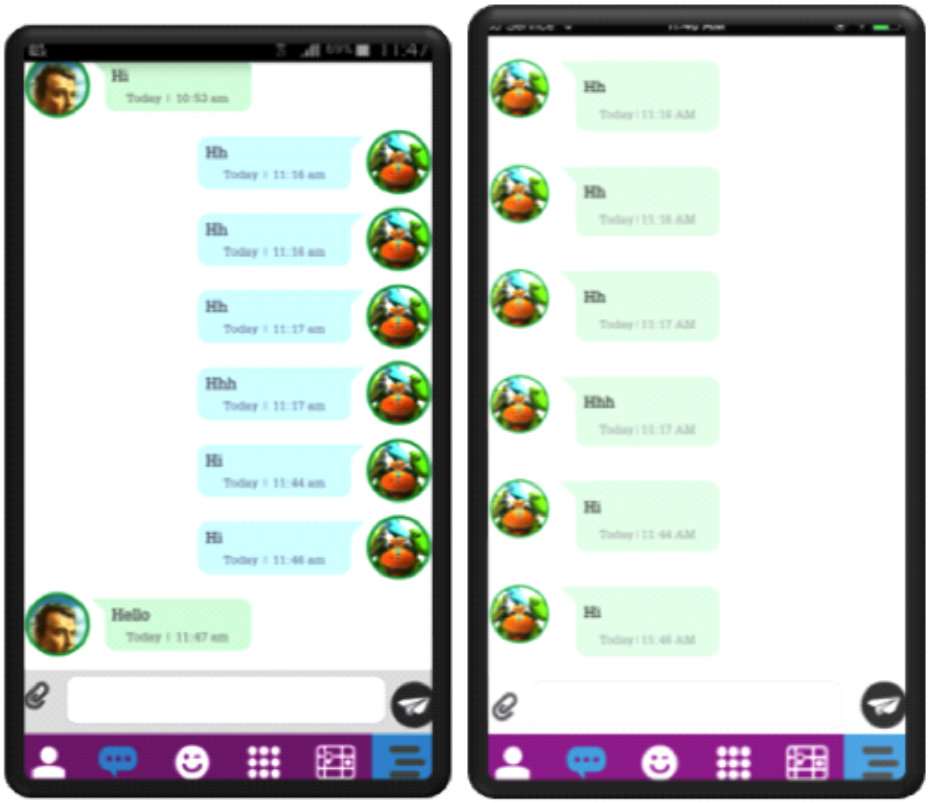
当开发者们为移动网页应用和推送通知编码时，他们不得不记住在不同操作系统上的用户必须看到相同的东西。质量保证部门必须在所有平台和所有可能的应用状态上测试一个新的或者已更新的应用并且设备不得不被作为因素考虑进。

这里，我将分析当应用在前台或者被推倒后台使用，且当移动设备被锁住，离线，

或者关闭状态。

当应用在前台使用

当我们在一个与某人的聊天线程里，说，用户 A，并且他们给我们发送另一条信息，不论它是否在一个苹果或者安卓设备上，我们不应该受到任何推送。我们将简单地看到一个新消息在我们的聊天线程里。



但是，如果在与用户 A 的一个聊天线程里，且用户 B 发送我们一条信息，在安卓设备上，我们将送到音频和可视通知。即使我们在相同应用的另一个屏幕，我们将收到警示。无论如何，一个苹果设备只发送音频警示。

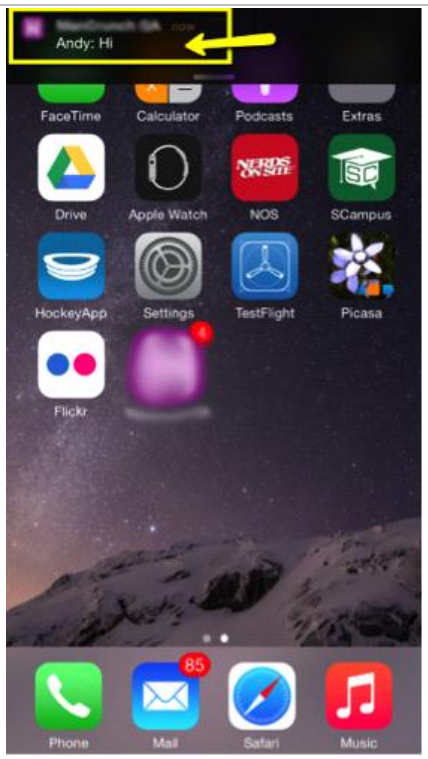
这是一个在质量保证团队在测试一个消息应用如何执行功能时不得不认识到的操作系统库之间的不同。

当应用在后台使用

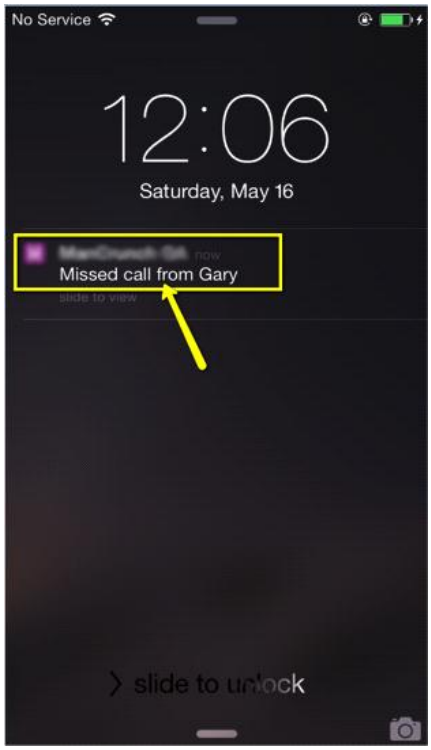
在安卓设备，当我们收到一条信息，我们能看到发送者的名字和在推送面板上的信息。如果一条信息很长，我们能看到的只是它最初的部分。直到我们进入信息，我们将看到在通知面板的应用图标，并且当我们双击它，我们会被带进应用的聊天线程里。

超过一条收到的信息会会将被显示成，比如，“从 2 个对话来的 2 条信息。”他们可能是从相同的发送者或者从多人来的。

无论如何，在一个苹果设备上，我们看到持续数秒的信息且然后它消失了，所以如果我们不在周围，我们可能不知道警示知道我们进入应用。另一个不同是每一条信息被分别显示在苹果设备上，即便它们是从相同的发送者来的。



假如我们的设备在锁定的状态，或者它是一个安卓或者苹果，它将收到音频和可视的通知。在安卓设备上，我们将不能看到警示直到解锁它；在苹果手机上，我们有选项能设置屏幕开/关这样以来当我们一收到它们时我们就能在它自己锁屏的设备看到警示。

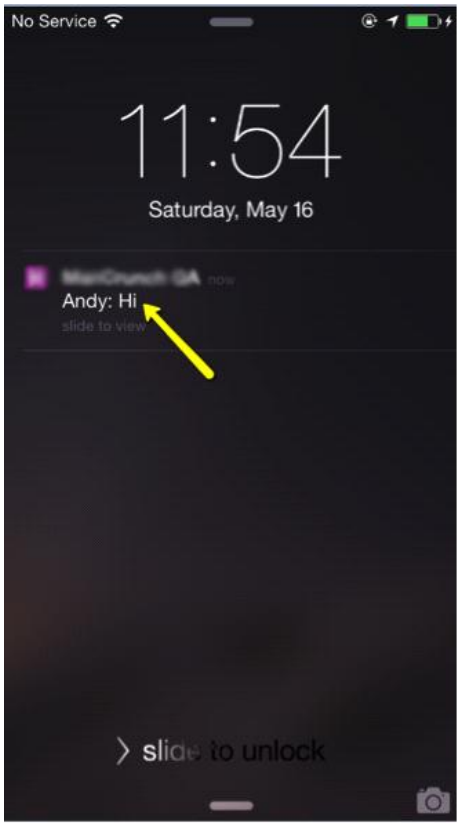


当设备离线或者关闭

假如我们的设备离线或者我们把它关了，我们将想要我们一回来就尽快被信息警示

通知。一旦我们在线，我们能看见在锁屏上的通知。

在一个苹果设备上，当我们离线时如果同一发送者已发了超过一条信息或者给我们打电话，然后只有最后一条信息或者错过的电话通知，就像情况可能是，在锁屏上可见。当我们不在时，假如超过一人已发送信息或者打电话，然后只要我们一在线，我们将只看到最近的信息。



在安卓设备上，无论如何，收到的信息数量将会被显示，举个例子，“从 2 个会话来的 2 条新信息。”

用户体验的重要性

测试推送通知包含考虑应用和设备所有的可能状态在任何既定时间。假如应用设计需要在这样一个颗粒级别上工作，只考虑测试努力所需的所有可能场景的测试。对于测试团队去计划它们的测试策略以至于没有一个场景被留与偶然发生使得应用工作得很好来说，它是一个挑战和机会。

理解用户体验性对于交付有质量的软件是首要的，这样会助于创造忠诚的客户基础。确保你理解你的配置审核物理层和功能两个方面。

我在国企做测试的日子

◆作者：张云斌

2016 年 11 月，我从一家互联网公司，跳槽到一家大型国有企业的信息部，开始了测试生涯的一段激流勇进之旅。

我从一毕业开始就从事软件测试工作，到现在已经有 7 年时间了，一开始做 web 自动化测试 3 年，到后来做 app 自动化测试 3 年。俗话说，三十年河东，三十年河西，于我而言，是三年 web，三年 app。在软件测试行业里，不管做 web 还是 app，都离不开软件工程的流程，也跑不掉项目的需求、设计、开发、测试、上线阶段。从最开始的使用 QTP 做录制回访，再到 Robotium 搭建的自动化测试框架跑 app 的 UI 自动化，都是在纯粹的自动化测试领域下，不断地自学、探索、提问、投入大量的时间和精力，去完成每一次版本的迭代和回归测试。这些或许都是测试生涯里的一次次里程碑，但是，在现在的单位里，我不断地审视自己，真的只是一个会做自动化测试的专项测试人员吗？

在现在的工作环境下，我的岗位依然是软件测试工程师，但是，由于我们的项目大部分外包给第三方团队开发、测试，所以，在我的工作中，很少参与到具体功能的测试中，基本都是由外包测试人员执行。在初入单位时，我让自己更多地参与到项目的排期会，通过每周的需求会，第一时间掌握产品的需求，并通过自己的理解，将功能需求用思维导图做了一次分解，并把初步的设计方案同步给测试人员。然后我会召集外包团队的开发和测试人员开会讨论本次的需求和用例设计方案，做一次用例的评审。

由于外包团队和本部团队对工作的参与精神是不一样的，外包团队可能会认为，我只要按照需求做好功能并交付就可以了。所以，在这之前我们的产品始终是一款只满足内部需求的产品，而在用户体验、交互设计上始终没有任何突破和创新，这是我在国企工作中发现的一个很深刻的问题。所以，我在入职的初期，尽可能地通过使用公司的多款产品，整理出产品在设计、交互、界面、使用上存在的一系列问题和改进点，并反馈给项目负责人。通过这种形式，让我这个新人能够受限于潜移默化的惯性思维，跳出传统的界限，提出富有创新的思路。

在这里，我的定岗依然是自动化测试工程师，所以，在入职的时候，我也开始着手搭建基于 selenium 的 web 自动化测试框架。由于在进入这家公司之前，我的三年时间里，都是在做移动应用的自动化测试，所以，在自动化测试工具的选型上，我也是慎之又慎，生怕自己在不熟悉的工具上耗太多时间，又怕做出来后没有任何的产出，还好架构师和导师认可了我的能力，并给了我充分调动外包资源的权利，所以最后我们选择了 selenium 这个成熟的 web 自动化测试工具。并用 3 个月的时间，将基础框架搭建并完成部分功能的自动化回归测试。我也第一次将 PO 模式带入到实际的测试代码编写中，由于对 webdriver 协议的不熟悉，也不断地在网上发帖寻求帮助，搜索相关材料。

在国企和互联网公司的最大区别就是，互联网公司的工作自由，不需要太多规则约束，而在国企，你需要有严谨地表达，准确的措辞，良好的计划导向，整体的逻辑思维。所以，每做一项任务，你需要有相应的规划。不过，部门经理在一次谈话中，也说过沿途下蛋的故事，就是在做一件事情的时候，你不能一股脑儿扎进去，而需要在每一个阶段有产出。领导这边看中的是不定期地输出成效，而对于之前我在互联网企业做测试，我可能更多的是关注专项测试，并可以自由地按照自己的期望去做，领导最人员的培养可能没有全面细致的要求。

在这里，很多工作上的事情，需要不断地沟通、不断地开会确认、不断地逐级审批，一开始我会认为这些繁琐而复杂的程序，束缚手脚，让自己放不开，去做自动化的工作。后来，当我认真想想，其实，这何尝不是一次次先苦后甜的经历啊。也许，你在沟通中出现障碍和分歧了，也许你在开会时别人提出异议了，也许你在提交某项事务时，领导会用预算、成本、投入产出比低等理由驳回你的任务。很多事情在着手开展之前，确实需要经过谨慎思索，并经过沟通讨论后，输出的方案才能准确有效，才能避免后面出现的返工，尽量将问题暴露控制在最低的范围内。其实，领导可能会相对保守，但站在公司整体战略的角度想想，最终也就会释然了。

在国企的日子里，不但是自己的测试管理能力提升了，而且也将对知识的学习由深度优先变为广度优先，不但要精通专项测试，而且需要知道网络搭建、运维流程、服务器管理、信息安全系统，除了要有良好的沟通表达能力，还需要能全局掌控信息部的体系结构。

在国企的日子里，我想我要继续扮演的角色，是一位让大家都离不开的专项测试工程师，可以在网络测试、服务器性能测试、信息安全测试里，给其他各岗位人员提供强

大有力的技术方案和质量保障，让自己哪一天不在岗位上时，他们仍然需要自己的协助。这才能实现自我价值，也许这方面我还需要慢慢打磨和突破自我。

这是我在国企的日子里，从一个专项的测试工程师，努力蜕变成一个全能型的工程师。接下来的日子里，还需要在安全测试领域、自动化测试领域，更进一步投入到实际的工作中，获取更多的知识，来实现一次次伟大的飞跃，希望可以跟伙伴们一起加油！

Web 前端性能测试平台开发(Flask)

作者：晴 空

开篇先打个小广告，在《牛刀小试-LR 性能测试》那篇小文中我有说到性能测试要做到性能的原子化，这样我们把性能可以分为前端，网络，中间件，APP(应用)，操作系统，数据库等，今天，我们来一起开发一个专门对 Web 前端性能自动化平台(后续可以在该版本的技术和基础上完善其他功能，比如说：接口的自动化和接口性能以及对其他层的监控数据做可视化)。

一：要啥自行车？奢侈！

大家都懂敲代码之前有很多事情要做并且是最重要的事情(能敲代码的人多的是，最重要的是能产出发现问题并给出解决策略的 idea)，what’ s this? 当然是需求分析。

我们的愿景：

实现 Web 前端性能测试(自动遍历所有页面) 监控每个页面加载时间段的耗时，并且统计每个页面中附加的资源(css/js/img/XmlHttpRequest)最后利用精美的图表作展现。

技术可行性分析：

自动遍历所有页面？没问题啊 webdriver 是这块儿的利器啊。

如何统计页面加载时间呢？performance.timing 绝对靠谱。

哪儿有精美的图表？百度 Echarts 团队为你分忧解难。

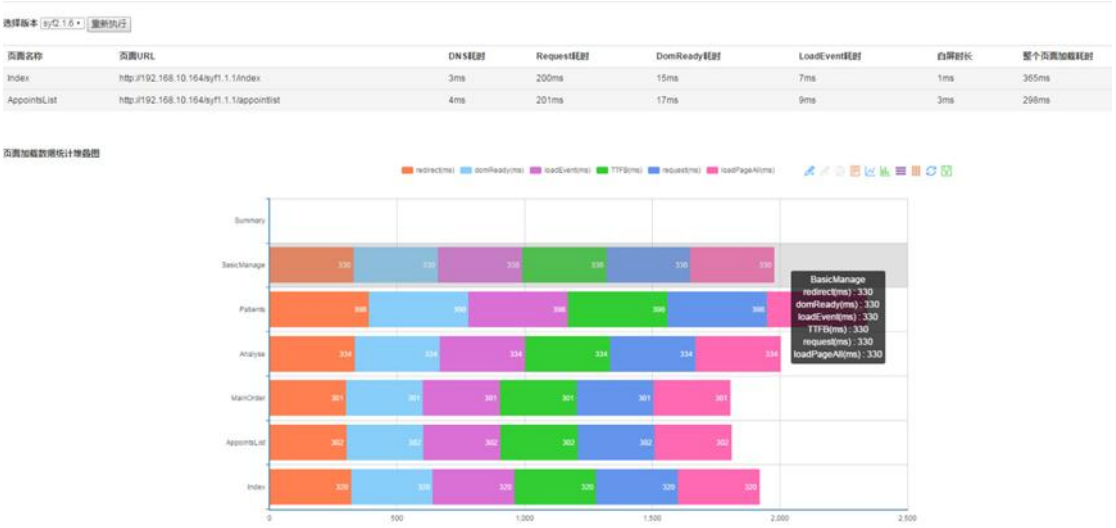
技术选型：

这是个非常值得探讨的问题，有些 Leader 是综合团队现状和实施成本(新技术是需要花时间学啊)；有些知乎党就不说了，啥最新用啥，只用最新的，不用最合适的；当然还有一些人完全是靠自己的兴趣拍脑袋决定的。

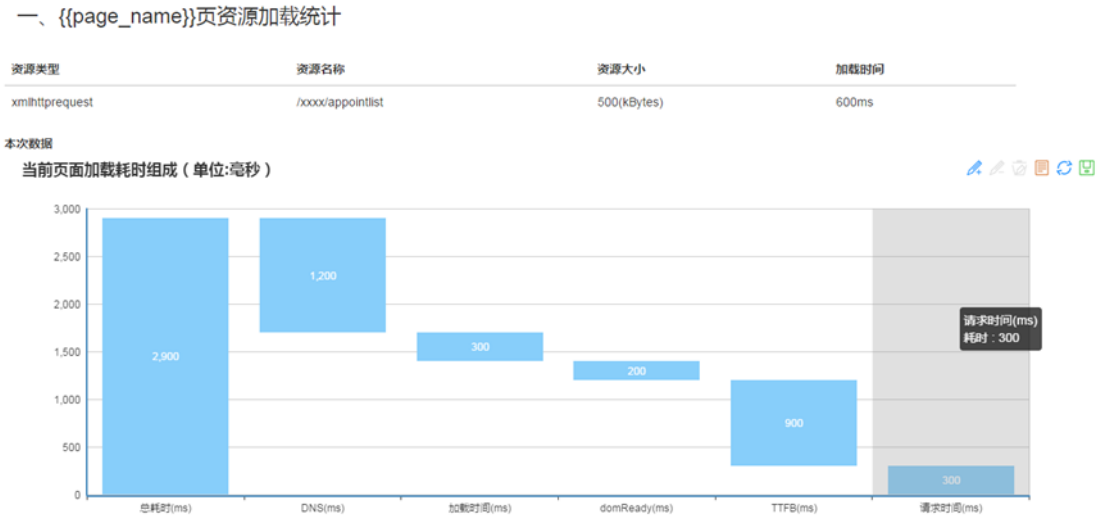
我们这里选择 python+webdriver+flask+sqlite+bootstrap+jquery 来完成我们这个小平台的开发，至于为什么会选择这几种技术，学习成本低，开发效率高，总而言之一句话 ROI 高！

好嘞~ 童鞋们可以先脑补下自己想要什么样的交互页面，我这里给出一个最简单的嘿嘿~

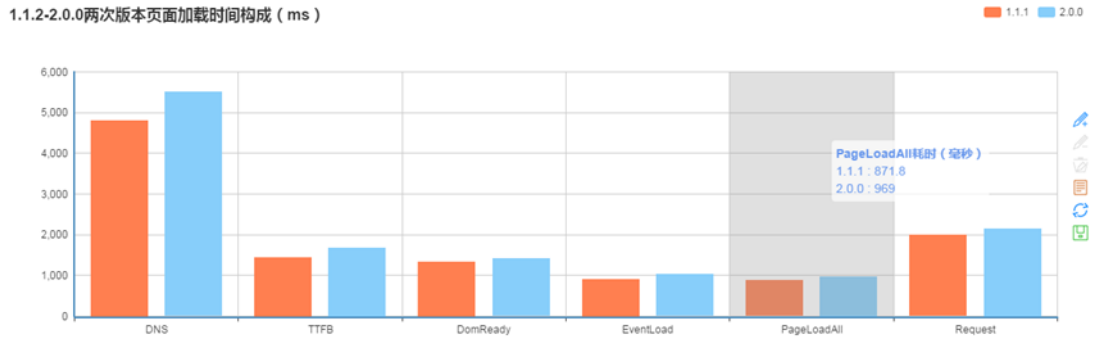
主页面：(展示统计到的页面信息并以堆叠图方式展示性能数据):



页面详情页面：(展示页面中附加资源的组成以及该页面历史版本的数据对比图)



二、 历史版本比较

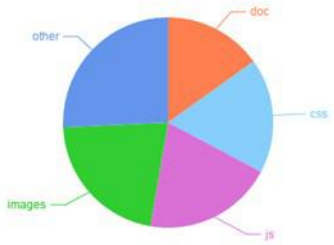


三、资源类型统计

资源类型	请求数	请求Size(kb)	加载时间(ms)
doc			
css			
js			
images			

Top3名称	资源类型	加载时间(ms)	百分比
	css		
	doc		
	imgs		

请求数量分布



请求大小分布

（ps：对技术选型这块儿同学们可以根据自己的需要做改变

如果你不想用 flask 可以换成 Django 框架

如果你不想用 sqlite 数据库 可以换成 mysql 或者其他的 NoSql 类数据库

如果你不想用 jquery 那你的选择就更多了 什么 React,Nodejs,Vuejs,Angularjs 还有啥 backbone.js 眼花缭乱。

我个人喜好小而精致的东西 就拿 flask 来说吧 它是个微型框架远远没有 Django 重，但是 flask 丰富的插件可以供我们快速地完成任务。所以嘛，要啥 Django 要啥自行车，别太奢侈。

技术没有 Low 不 Low 之分！ 只有适合不适合。）

二：这个轮椅是专门为你设计的

这里是对应软件工程里的概要设计阶段。经过前面的梳理，我们很清楚地知道自己想要的是什么了。嗯~接下来咱把硬邦邦的需要转化为软件工程里的东东吧。

数据库：我们建两张表分别存放所有页面的统计数据就叫 WebPages，还需要建立

一张表来存放每个页面的详细信息 我们叫 PageDetail。

建表语句如下:

```
CREATE TABLE [PageDetail] (
  [project] text,
  [resourceName] text,
  [requestType] text,
  [pageRequestCount] INTERGER,
  [resourcetSize] INTERGER,
  [resourceLoadTime] INTERGER,
  [createTime] datetime,
  [pageName] text,
  [pageLoadTime] INTEGER);
```

```
CREATE TABLE WebPages (project text, name text ,url text  ,direct INTERGER ,domReady INTERGER, loadEvent INTERGER ,request INTERGER ,ttfb INTERGER,loadPageAll INTERGER ,ifredirect INTERGER, createTime  datetime);
```

接口:

我们重新执行脚本来统计前端性能的话，需要调用接口。

我们就命名一个接口吧，python-flask 里接口的定义很简单哦~

```
# 重新执行测试脚本
@app.route('/redo', methods=['GET', 'POST'])
def redo():
  if request.method == 'GET':
      return u'该接口不支持 GET 方法访问'
  else:
      # 向测试环境发生 get 请求来验证测试环境是否存在
    if env_test_code == 200:
      # 重新执行测试脚本
      Else:
      Return  测试环境不存在
```

如果我们选择不同版本的话，我们也需要调用接口来返回所选版本的测试数据:

```
@app.route('/index', methods=['GET', 'POST'])
```

```
def index():
    if request.method == 'POST':
        # 从 ajax 请求中取参数 selected_version 取不到则为 null
        version_text = request.form.get("selected_version", "null")
    else:
        version_text = '默认版本号'
    # 接下来巴拉巴拉从 sqlite 中取数据并返回
```

三：走两步~走两步！

让我们开始敲代码吧？好啊！来吧！

接下来我们先完成前端部分的开发工作然后再搞后台部分的任务，bootstrap &flask 学起！

3.1、flask 环境搭建和基础知识

啥是 flask?

Flask 是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2 。

Flask 也被称为 “microframework”，因为它使用简单的核心，用 extension 增加其他功能。Flask 没有默认使用的数据库、窗体验证工具。然而，Flask 保留了扩增的弹性，可以用 Flask-extension 加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

环境搭建:

先用 pip 安装 virtualenv 库~ (这样做是为了单独拉一个 python 环境出来以便我们一台服务器上可以跑多套服务)

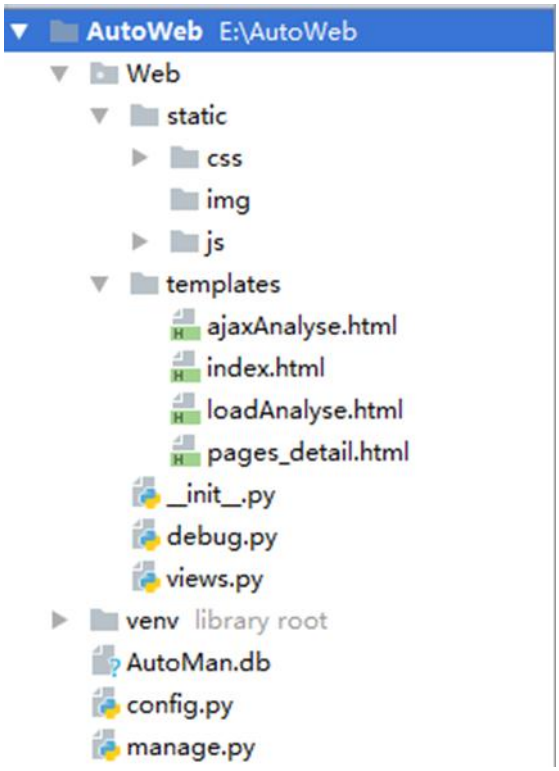
Dos 窗口里敲命令:pip install virtualenv

然后，新建一个目录 我们就叫 AutoMan 吧，Dos 窗口里切换到 AutoMan 目录后执行：virtualenv venv(创建虚拟环境目录)

我们如果要在 virtualenv 里安装需要的包，需要先激活虚拟环境(切到 venv/scirpts 里执行 activate)然后再执行 pip 安装即可。

我们这里需要安装的环境有 flask，selenium，requests。我们可以分别执行安装，也可以将 flask，selenium，requests 放到 requirement.txt 文件里然后执行 pip install －r requirement.txt。

上一张我们项目的目录结构图



目录说明:

1: Web 我们的项目目录

1.1: static 存放静态文件的目录，它下面又分为 css,img,js 目录分别存放样式文件图片和 js 文件(下载 jquery.js 放到 js 目录下，bootstrap.css 放到 css 目录下)。

1.2: templates 目录存放的是我们的 html 静态页面。

1.3: views.py 里定义了接口以供前端发起请求。

1.4: __init__.py 里初始化了 Web 这个服务。

2: 和 Web 平级的 venv 是我们 flask 虚拟环境的目录，发布的时候不需要它。

3: AutoMan.db 我们用到的 sqlite 数据库文件。

4: config.py 文件里是我们用到的配置信息，比如说页面信息，统计性能的 js 方法

等。

5: manage.py 是我们程序的执行入口。

我们先来初始化我们的应用:

在__init__.py 文件里输入:

```
# -*- coding: utf-8 -*-
import sys
reload(sys)
sys.setdefaultencoding("utf-8")

from flask import Flask

app = Flask(__name__)
from app import views
```

接着，我们可以尝试下在页面输出一个字符串是什么感受，嗯啊，感受!

在 views.py 文件里输入:

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

import requests, config
import sqlite3, time
from selenium import webdriver
from app import app
from flask import render_template, g, redirect, url_for, request
import sys
reload(sys)
sys.setdefaultencoding("utf-8")
```

定义路由

```
@web_app.route('/')
def root():
return 'Hello Flask'
```

在 manage.py 文件里启动我们的应用:

```
from Web import web_app
web_app.run(host='0.0.0.0', port=5566)
```

启动服务(我用的是 pycharm 调试)，在浏览器里输入 http://localhost: 5566

是不是很神奇~~~~我们继续补充下 flask 中的路由知识。

Flask 中 route() 装饰器把一个函数绑定到对应的 URL 上，灰常简单！

比如下面的

```
@app.route('/ajaxAnalyse')
def ajax_analyse():
return render_template('ajaxAnalyse.html')
```

我们通过浏览器访问 localhost:5566/ajaxAnalyse，flask 路由会指定该地址并返回 ajaxAnalyse.html 模板交给浏览器渲染后呈现给我们。

```
@app.route('/resource_<name>_<version>')
def resource_analyse(name, version):
    print name
    return render_template('pages_detail.html')
```

如果我们想在路由中使用变量，<>包含起来即可，上面这个例子中我们使用两个变量，分别是 name(页面名称)和 version(项目版本)。

好嘞~ 接下来就简单了，我们开始构造 url 路由。

首先是首页，首页的话，我们想展示所有页面的统计信息和堆叠图。所以下面的代码即可实现（我们从数据库中读取数据并和 html 模板一起返回让浏览器渲染）。

```
# 主页显示所有页面前端信息
@app.route('/index', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # 从 ajax 请求中取参数 selected_version 取不到则为 null
        version_text = request.form.get("selected_version", "null")
    else:
        version_text = 'Syf1.1.1'
    g.db = sqlite3.connect(config.db_dir)
```

```
            all_page_load_data = {}
            page_time_detail = []
              # 从数据库中查询所有页面汇总信息
            for value in ['direct', 'domReady', 'loadEvent', 'request', 'ttfb', 'loadPageAll', 'name']:
                sql = 'select ' + value + " from webLoad where project='" + str(version_text) + "'" + ' order by
name'
                current_data = g.db.execute(sql).fetchall()
                current_list = []
                if value == 'name':
                    for data in current_data:
                        current_list.append(str(data).strip("(u'").strip("',)"))
                    all_page_load_data[value] = current_list
                else:
                    for data in current_data:
                        current_list.append(int(str(data).strip('(').strip(')').split(',')[0]))
                    all_page_load_data[value] = current_list
            # 查询每个页面的时间详细
            page_detail_sql = "select name, url, loadEvent, domReady, request, ttfb, 0 as request_count, 0 as
request_size " \
                              "from webLoad where project='" + str(version_text) + "'" + ' order by name'
            page_detail_data = g.db.execute(page_detail_sql).fetchall()
            for data in page_detail_data:
                page = {}
                page['name'] = data[0]
                page['url'] = data[1]
                page['loadEvent'] = data[2]
                page['domReady'] = data[3]
                page['request'] = data[4]
                page['ttfb'] = data[5]
                page['request_count'] = data[6]
                page['request_size'] = data[7]
                page_time_detail.append(page)
            g.db.close()
```

```
        return render_template("index.html", all_page_load_data=all_page_load_data,
page_time_detail=page_time_detail)
```

咦？这里我们貌似引入了一个新词—模板，没错 flask 中使用 Jinja 作为模板。

我们一起来看下 Jinja 中一些常用知识点。

0：表达式

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
<title>My Webpage</title>
</head>
<body>
<ul id="navigation"> {% for item in navigation %}
<li><a href="{{ item.href }}">{{ item.caption }}</a></li>
{% endfor %} </ul>
 <h1>My Webpage</h1> {{ a_variable }}
</body>
</html>
```

两种分隔符{% … %} 和 {{ … }}. 前者用于执行表达式，后者打印表达式的结果.

1、变量

```
#这两种表达方式是一样的，如果属性不存在默认的设置是返回空字符串
{{ foo.bar }} {{ foo['bar'] }}
```

2、Filters(过滤器，我们也可以自定义过滤器)

对变量操作的函数，以|的形式写在后面

```
#几个例子
attr(obj, name)
foo|attr("bar") <=>(等价于) foo["bar"]
2.batch(value, linecount, fill_with=None)
#从左边取 3 个值，fill_with 为不够的情况下的填充值
{%- for row in items|batch(3, ' ') %}
```

3、注释 Comments

#写在{# ... #}之间

{# note: disabled template because we no longer use this

{% for user in users %} ... {% endfor %} #}

4、空白 whitespace control

没有配置 trim_blocks 和 lstrip_blocks

 {% if True %} yay {% endif %} -> yay

配置了 trim_blocks 和 lstrip_blocks

 yay

手动开启 lstrip_blocks

 {%+ if something %}yay{% endif %}

手动删除空白

{% for item in seq -%} {{ item }} {%- endfor %}

5、转换符 Escaping

使用{{}} ，或者 raw block {% raw %}

{{ '{{' }} {% raw %} {% for item in seq %} {{ item }}

{% endfor %} {% endraw %}

6、Line Statements

配置后可以 mark a line as a statement

for item in seq:

{{ item }} ## this comment is ignored

endfor

这种方法同 {% for item in seq %} {{ item }} {% endfor %}

7、模板继承 Template Inheritance

#1.父类模板 Template

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">

<html lang="en"> <html xmlns="http://www.w3.org/1999/xhtml">

<head> {% block head %}

<link rel="stylesheet" href="style.css" />

<title>{% block title %}{% endblock %} - My Webpage</title>

{% endblock %}

</head>

<body> {% block content %}{% endblock %}

 {% block footer %} © Copyright 2008 by you. {% endblock %}

</body>

#2.子类模板继承 Child Template

{% extends "base.html" %}

{% block title %}Index{% endblock %}

{% block head %} {{ super() }}

<style type="text/css"> .important { color: #336699; } </style> {% endblock %} {% block content %} <h1>Index</h1>

 Welcome on my awesome homepage. {% endblock %}

#3.还可以用 self，引用 block 的变量

<title>{% block title %}{% endblock %}</title>

<h1>{{ self.title() }}</h1> {% block body %}{% endblock %}

Super Blocks 引用 parent block 的内容

{% block sidebar %} <h3>Table Of Contents</h3> ... {{ super() }} {% endblock %}

#4.block 结束符也可以加上名字

{% block sidebar %} {% block inner_sidebar %} ...

{% endblock inner_sidebar %}

{% endblock sidebar %}

#5.block 中的变量引用

{% for item in seq %} {% block loop_item %}{{ item }}{% endblock %} # 打印为空，因为 block 内部不能引用外面的 item 变量 {% endfor %}

需要改变为 {% for item in seq %} {% block loop_item scoped %}{{ item }}{% endblock %} {% endfor %}

#6.If a template object was passed to the template context you can extend from that object as well

{% extends layout_template %}

#7.转换符 HTML Escaping

可以使用自动或者手动转换

Automatic Escaping 自动转换会转换所有字符，除非在应用中设置了 safe 的或者后面加了|safe

Manual Escaping 手动转换：加|e filter: {{ user.username|e }}. 这是 escape 的缩写

#8.控制结构

1）、For 循环

```
<h1>Members</h1> <ul> {% for user in users %}
<li>{{ user.username|e }}</li>
{% endfor %} </ul>
```

2）、if/else 结构

```
if {% if kenny.sick %} Kenny is sick.
{% elif kenny.dead %} You killed Kenny! You bastard!!!
{% else %} Kenny looks okay --- so far
{% endif %}
```

#9.宏 Macros

1）定义一个宏

```
{% macro input(name, value='', type='text', size=20) -%} <input type="{{ type }}" name="{{ name }}" value="{{ value|e }}" size="{{ size }}"> {%- endmacro %}
```

2）使用

```
<p>{{ input('username') }}</p> <p>{{ input('password', type='password') }}</p>
```

3）类似一个宏里调用另一个宏，使用 call block

```
{% macro render_dialog(title, class='dialog') -%}
<div class="{{ class }}"> <h2>{{ title }}</h2>
 <div class="contents"> {{ caller() }}
</div>
</div>
{%- endmacro %}
{% call render_dialog('Hello World') %} This is a simple dialog rendered by using a macro and a call
```

block. {% endcall %}

带参数 {% macro dump_users(users) -%} {%- for user in users %} {{ user.username|e }} {{ caller(user) }} {%- endfor %} {%- endmacro %} {% call(user) dump_users(list_of_user) %} <dl> <dl>Realname</dl> <dd>{{ user.realname|e }}</dd> <dl>Description</dl> <dd>{{ user.description }}</dd> </dl> {% endcall %}

#10.Filter sections 过滤器块

可以对一个 block 使用 filter

{% filter upper %} This text becomes uppercase {% endfilter %}

#11.赋值

{% set navigation = [('index.html', 'Index'), ('about.html', 'About')] %}

{% set key, value = call_something() %}

#12.Include(包含)

把其他 tempalte 的内容加进来

{% include 'header.html' %} Body

{% include 'footer.html' %}

{% include ['special_sidebar.html', 'sidebar.html'] ignore missing %}

#13. import（导入）

1）如 forms.html 定义了两个宏

{% macro input(name, value='', type='text') -%}

<input type="{{ type }}" value="{{ value|e }}" name="{{ name }}">

{%- endmacro %}

{%- macro textarea(name, value='', rows=10, cols=40) -%}

<textarea name="{{ name }}" rows="{{ rows }}" cols="{{ cols }}">{{ value|e }}

</textarea>

{%- endmacro %}

2）两种 import 的方法

{% import 'forms.html' as forms %} <dl> <dt>Username</dt> <dd>{{ forms.input('username') }}</dd> <dt>Password</dt> <dd>{{ forms.input('password', type='password') }}</dd> </dl> {{ forms.textarea('comment') }}

{% from 'forms.html' import input as input_field, textarea %} <dl> <dt>Username</dt> <dd>{{ input_field('username') }}</dd> <dt>Password</dt> <dd>{{ input_field('password', type='password') }}</dd> </dl> {{ textarea('comment') }}

Jinja2 模板的用法非常多，我们只需要掌握常用的即可，遇到不会的直接去官网搜索吧。

下面贴上我这边 Index.html 页面源码中所用到的 Jinja:

基类模板内容:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
  <title>{% block title %} {% endblock %}</title>
  <link rel="stylesheet" type="text/css" href="../static/css/bootstrap.min.css">
  <script src="../static/js/jquery.min.js"></script>
  <script type="text/javascript" src="../static/js/echarts-all.js"></script>
  <script type="text/javascript" src="../static/js/bootstrap.js"></script>
  <script type="text/javascript" src="../static/js/events.js"></script>
  {% block head %}{% endblock %}
</head>
<body>
  <nav class="navbar text-info" role="navigation">
        <ul class="nav nav nav-tabs panel-title">
            <li class="dropdown">
                <a class="dropdown-toggle" data-toggle="dropdown" href="#">前端性能<b
class="caret"></b></a>
                <ul class="dropdown-menu">
                    <li><a href="/index">Clinical 项目</a></li>
                    <li><a href="#">GeStorage 项目</a></li>
                    <li><a href="#">Training 项目</a></li>
                    <li><a href="#">PerformanceTiming 学习</a></li>
                </ul>
            </li>
            <li class="dropdown">
                <a class="dropdown-toggle" data-toggle="dropdown" href="#">UI 自动化测试<b
class="caret"></b></a>
```

```
                    <ul class="dropdown-menu">
                        <li><a href="ui_auto">Clinical 项目</a></li>
                        <li><a href="ui_auto">GeStorage 项目</a></li>
                        <li><a href="ui_auto">Training 项目</a></li>
                        <li><a href="ui_auto">RobotFrame 知识库</a></li>
                        <li><a href="ui_auto">UI 自动化脚本设计</a></li>
                    </ul>
                </li>
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">接口自动化测试<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li><a href="service_auto">Clinical 项目</a></li>
                        <li><a href="service_auto">GeStorage 项目</a></li>
                        <li><a href="service_auto">Training 项目</a></li>
                        <li><a href="#">接口测试知识库</a></li>
                    </ul>
                </li>
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">兼容性自动化测试<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li><a href="#">Clinical 项目</a></li>
                        <li><a href="#">GeStorage 项目</a></li>
                        <li><a href="#">Training 项目</a></li>
                        <li><a href="#">兼容性实施原理</a></li>
                    </ul>
                </li>
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">性能测试<b
class="caret"></b></a>
                    <ul class="dropdown-menu">
                        <li><a href="#">Clinical 项目</a></li>
```

```
                    <li><a href="#">GeStorage 项目</a></li>
                    <li><a href="#">Training 项目</a></li>
                    <li><a href="#">性能测试实施细则</a></li>
                </ul>
            </li>
            <li><a href="per_auto">全局变量配置</a></li>
            </ul>
    </nav>
        <div id="content" class="container">
          {% block content %}
          {% endblock %}
        </div>
</body>
</html>
```

我这边 index.html 内容如下:

```
{% extends "base.html" %} //继承基类模板
{% block title %}平台主页{% endblock %}
{% block content %}
        <div class="row">
          <label>项目版本</label>
          <select id="project_version" onchange="optionChange()">
                {% for option in project_version.Clinical %}
                <option value={{option}}>{{option}}</option>
                {% endfor %}
          </select>
          <button id="redo" onClick="ajaxReRun()">重新执行</button>
        </div>
        <br>
        <table id="pages_summary" class="table table-hover">
          <th>页面名称</th>
          <th>页面 URL</th>
          <th>DNS 耗时</th>
```

```
                <th>Request 耗时</th>
                <th>解析 DOM 耗时</th>
                <th>DomReady 耗时</th>
                <th>LoadEvent 耗时</th>
                <th>白屏时长</th>
                <th>整个页面加载耗时</th>
                {% for page in page_load_result %} //Jinja 中的循环体
                <tr class="active hover">
                    <td class="active"><a
href="javascript:{{page.page_name}}()">{{page.page_name}}</a></td>
                    <script type="text/javascript">
                        function {{page.page_name}}()
                        {
                            var href = $('#project_version').get(0).selectedOptions[0].text + "_"
+"{{page.page_name}}";
                            location.href = href
                        }
                    </script>
                    <td class="active">{{page.url}}</td>
                    <td class="active">{{page.dns}}</td>
                    <td class="active">{{page.request}}</td>
                    <td class="active">{{page.dom_parser}}</td>
                    <td class="active">{{page.dom_ready}}</td>
                    <td class="active">{{page.load_event}}</td>
                    <td class="active">{{page.whitewait}}</td>
                    <td class="active">{{page.loadall}}</td>
                </tr>
                {% endfor %} //Jinja 中循环体的结束标识
            </table>
        <br><br>
            <h4>所有页面加载数据统计堆叠图</h4>
        <div id="page_summary_chart">
            <table id="data_chart" width="100%">
```

```
            <tr>
                <td align="center">
                <div id="chart" class="text-align:  center" style="width: 1250px;height:650px;">
                    <script type="text/javascript">
                        // 基于准备好的 dom，初始化 echarts 实例
                        var myChart = echarts.init(document.getElementById('chart'));
                        // 指定图表的配置项和数据
                        option = {
                        tooltip : {
                            trigger: 'axis',
                            axisPointer : {type : 'shadow'}
                        },
                        legend: {
                            data:['dns(ms)', 'request(ms)', 'dom_parser(ms)', 'dom_ready(ms)',
'load_event(ms)', 'whitewait(ms)', 'loadall(ms)'],
                        },
                        toolbox: {
                            show : true,
                            feature : {
                                mark : {show: true},
                                dataView : {show: true, readOnly: false},
                                magicType : {show: true, type: ['line', 'bar', 'stack', 'tiled']},
                                restore : {show: true},
                                saveAsImage : {show: true}
                            }
                        },
                        calculable : true,
                        xAxis : [
                            {
                                type : 'value'
                            }
                        ],
                        yAxis : [
```

```
        {
            type : 'category',
            data :    {{page_chart_data.page_name|safe}}
        }
    ],
    series : [
        {
            name:'dns(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.dns|safe}}
        },
        {
            name:'request(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.request|safe}}
        },
        {
            name:'dom_parser(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.dom_parser|safe}}
        },
        {
            name:'dom_ready(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.dom_ready|safe}}
```

```
        },
        {
            name:'request(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.request|safe}}
        },
    {
            name:'load_event(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.load_event|safe}}
        },
        {
            name:'whitewait(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.whitewait|safe}}
        },
        {
            name:'loadall(ms)',
            type:'bar',
            stack: 'Time',
            itemStyle : { normal: {label : {show: true, position: 'insideRight'}}},
            data: {{page_chart_data.loadall|safe}}
        }
]
};
// 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);
```

```
                </script>
            </div>
            </td>
        </tr>
        </table>
    </div>
{% endblock %}
```

(PS: 在一个表格中循环查询到的结果并输出到页面)

下面贴出来我这边 views.py 文件里的全部内容供同学们参考:

```
# -*- coding:utf-8 -*-
import sqlite3
import sys
import time
import requests
from flask import render_template, g, redirect, url_for, request, send_file
from selenium import webdriver
import config
from Web import web_app
reload(sys)
sys.setdefaultencoding("utf-8")
# 定义根路由
@web_app.route('/')
def root():
    return render_template('login.html')
@web_app.route('/login_success')
def login_success():
    return render_template('login_success.html')
# 定义路由 index
@web_app.route('/index', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        version = request.form.get("selected_version", "null")
```

```
        else:
            version = 'syf1.1.1'
        g.db = sqlite3.connect(config.db_dir)
        summary_data_sql = 'select page_name, url, dns, request, dom_parser, dom_ready, load_event,
whitewait, loadall '\
                           + " from WebLoad where version='" + str(version) + "'" + ' order by
page_name'
        page_load_result = query_db(summary_data_sql)
        g.db.close()
        project_version = config.project_version_info
        # 将页面信息转换为 Echarts 使用的数据
        page_chart_data = {}
        page_name_list = []
        page_dns_list = []
        page_request_list = []
        page_dom_parser_list = []
        page_dom_ready_list = []
        page_load_event_list = []
        page_whitewait_list = []
        page_loadall_list = []
        for result in page_load_result:
            page_name_list.append(str(result['page_name']).strip("u'"))
            page_dns_list.append(result['dns'])
            page_request_list.append(result['request'])
            page_dom_parser_list.append(result['dom_parser'])
            page_dom_ready_list.append(result['dom_ready'])
            page_load_event_list.append(result['load_event'])
            page_whitewait_list.append(result['whitewait'])
            page_loadall_list.append(result['loadall'])
        page_chart_data['page_name'] = page_name_list
        page_chart_data['dns'] = page_dns_list
        page_chart_data['request'] = page_request_list
        page_chart_data['dom_parser'] = page_dom_parser_list
```

```
            page_chart_data['dom_ready'] = page_dom_ready_list
            page_chart_data['load_event'] = page_load_event_list
            page_chart_data['whitewait'] = page_whitewait_list
            page_chart_data['loadall'] = page_loadall_list
            return render_template('index.html', project_version=project_version,
page_load_result=page_load_result, page_chart_data=page_chart_data)
        # 定义重新执行脚本的路由
        @web_app.route('/redo', methods=['GET', 'POST'])
        def redo():
            if request.method == 'GET':
                return 'route does not support get'
            else:
                # 向测试环境发生 get 请求来验证测试环境是否存在
                env_for_test = request.form.get("selected_version", "null")
                current_env = config.hosts['164'] + env_for_test
                env_test_code = requests.get(current_env).status_code
                if env_test_code == 200:
                    # 定义页面名称列表
                    all_page_data = {}
                    page_urls = config.page_urls
                    timing_js = config.timing_js
                    # 统计页面加载时间的 6 个维度 登录页专用
                    index_data = {'dns': 0, 'request': 0, 'dom_parser': 0, 'dom_ready': 0, 'load_event': 0,
'whitewait': 0, 'loadall': 0}
                    # index 页面资源的信息 登录页专用
                    page_index_info = []
                    # 将统计数据保存到 sqlite
                    connection = sqlite3.connect(config.db_dir)
                    cursor = connection.cursor()
                    # 先删除页面资源细分表 webDetailLoad 所有数据
                    clear_resource_sql = "delete from PageDetail where project='Clinical' and version='" +
env_for_test + "'"
                    cursor.execute(clear_resource_sql)
```

```
                    # 先清理上次的数据
                    clear_webload_sql = "delete from WebLoad where project='Clinical' and version='" +
env_for_test + "'"
                    cursor.execute(clear_webload_sql)
                    connection.commit()
                    # get 到 clinical 登录页(登录页需要单独收集其页面加载性能指标)
                    driver = webdriver.Chrome()
                    login_url = current_env + '/login/index'
                    driver.get(login_url)
                    driver.maximize_window()
                    time.sleep(5)
                    time_line = driver.execute_script(config.timing_js)
                    # 将页面加载时间存入 all_page_data
                    for key in index_data.keys():
                        index_data[key] = time_line[key]
                    # 将 index 的数据存入 all_page_data
                    all_page_data['Index'] = index_data
                    # 获得 index 页面所有的资源实体
                    entries = driver.execute_script(config.get_entries_js)
                    # 遍历登录页面所有资源
                    for items in entries:
                        # 以页面名：数组形式保存页面信息
                        page_index_resource = {}
                        # 资源名称
                        page_index_resource['name'] = items['name']
                        # 资源类型分类
                        resource_type = str(items['name']).split('.')[-1]
                        if resource_type in ['jpg', 'png', 'gif', 'jpeg']:
                            page_index_resource['resourceType'] = 'img'
                        elif resource_type == 'js':
                            page_index_resource['resourceType'] = 'js'
                        elif resource_type == 'css':
                            page_index_resource['resourceType'] = 'css'
```

```
                else:
                    if items['initiatorType'] == 'xmlhttprequest':
                        page_index_resource['resourceType'] = 'xmlhttprequest'
                    elif items['initiatorType'] == 'script':
                        page_index_resource['resourceType'] = 'js'
                    else:
                        page_index_resource['resourceType'] = 'other'
                # 资源大小
                page_index_resource['transferSize'] = items['transferSize']
                # 资源耗时
                page_index_resource['duration'] = items['duration']
                # 将登录页资源信息存入列表
                page_index_info.append(page_index_resource)
                # 将登录页资源列表信息存入字典对象 对应 key 为 Index
                save_resource_sql = 'insert into PageDetail ' + "values('Clinical','" +
str(env_for_test) + "'," + "'" + str(
                    'Index') + "'," + "'" + str(page_index_resource['name']) + "'," + "'" + str(
                    page_index_resource['resourceType']) + "'," + "'" \
                                        + str(page_index_resource['transferSize']) + "'," + "'" + str(
                    page_index_resource['duration']) + "'," + 'CURRENT_TIMESTAMP)'
                cursor.execute(save_resource_sql)
                connection.commit()
            # 登录系统后 对其他所有页面进行遍历以收集其页面加载性能数据
            driver.find_element_by_id('txtUserName').send_keys('30048')
            driver.find_element_by_id('txtPassword').send_keys('5436')
            driver.find_element_by_id('btnConfirm').click()
            time.sleep(3)
            # 循环登录后的所有页面
            for page in page_urls.keys():
                # 将页面加载时间统计出来存入 all_page_data
                current_page_timing = {"dns": 0, "request": 0, "dom_parser": 0, "dom_ready": 0,
"load_event": 0, "whitewait": 0, "loadall": 0}
                current_page_url = current_env + (str(page_urls[page]))
```

```
driver.get(current_page_url)
time.sleep(10)
durations = driver.execute_script(timing_js)
for item in current_page_timing.keys():
    current_page_timing[item] = durations[item]
# 将当前页面的 name 和统计到数据存入到 all_page_data
all_page_data[page] = current_page_timing
entries = driver.execute_script("return window.performance.getEntries()")
# 遍历当前页面所有资源
for items in entries:
    current_resources = {}
    current_resources['name'] = items['name']
    # 根据资源类型进行统计
    resource_type = str(items['name']).split('.')[-1]
    if resource_type in ['jpg', 'png', 'gif', 'jpeg']:
        current_resources['resourceType'] = 'img'
    elif resource_type == 'js':
        current_resources['resourceType'] = 'js'
    elif resource_type == 'css':
        current_resources['resourceType'] = 'css'
    else:
        if items['initiatorType'] == 'xmlhttprequest':
            current_resources['resourceType'] = 'xmlhttprequest'
        elif items['initiatorType'] == 'script':
            current_resources['resourceType'] = 'js'
        else:
            current_resources['resourceType'] = 'other'
    # 资源大小
    current_resources['transferSize'] = items['transferSize']
    # 资源耗时
    current_resources['duration'] = items['duration']
    # 保存资源
    save_resource_sql = 'insert into PageDetail ' + "values('Clinical','" + str(
```

```
                                    env_for_test) + "'," + """ + str(
                                    page) + "'," + """ + str(
                                    current_resources['name']) + "'," \
                                                                 + """ + str(current_resources['resourceType']) + "'," +
""" + str(
                                    current_resources['transferSize']) + "'," + """ + str(
                                    current_resources['duration']) + "'," + 'CURRENT_TIMESTAMP)'
                                cursor.execute(save_resource_sql)
                                connection.commit()
                        # 退出 chrome driver 进程
                        driver.quit()
                        # 更新 WebLoad 表中的数据
                        for name in all_page_data.keys():
                            page_time = all_page_data[name]
                            if name == 'Index':
                                insert_sql = 'insert into WebLoad ' + "values('Clinical','" + str(env_for_test) +
"'," + """\
                                                                      + str(name) + "'," + """ + str(current_env) + "/login/index'," +
str(page_time['dns']) \
                                                                      + ',' + str(page_time['request']) + ',' +
str(page_time['dom_parser']) + ',' + str(page_time['dom_ready']) \
                                                                      + ',' + str(page_time['load_event']) + ',' +
str(page_time['whitewait']) + ',' + str(page_time['loadall']) + ',' + 'CURRENT_TIMESTAMP)'
                            else:
                                insert_sql = 'insert into WebLoad ' + "values('Clinical','" + str(env_for_test) +
"'," + """ \
                                                                      + str(name) + "'," + """ + str(current_env) +
str(page_urls[name]) + "'," + str(page_time['dns']) \
                                                                      + ',' + str(page_time['request']) + ',' +
str(page_time['dom_parser']) + ',' + str(page_time['dom_ready']) \
                                                                      + ',' + str(page_time['load_event']) + ',' +
str(page_time['whitewait']) + ',' + str(page_time['loadall']) + ',' + 'CURRENT_TIMESTAMP)'
                            cursor.execute(insert_sql)
                            connection.commit()
                        connection.close()
```

```
                return 'success'
            else:
                return 'environment is not exist'
    @web_app.route('/<project_version>_<page_name>')
    def page_detail(project_version, page_name):
        g.db = sqlite3.connect(config.db_dir)
        # 页面资源信息
        resource_sql = "select resource_name, resource_type, resource_size, resource_duration from PageDetail where version='" + str(project_version) + "' and page_name='" + str(page_name) + "' order by resource_duration desc"
        history_sql = "select version, dns, request, dom_parser, dom_ready, load_event, whitewait, loadall from WebLoad where page_name='" + str(page_name) + "' group by version"
        composition_sql = "select resource_type, count(resource_type) as counts, sum(resource_size) as size_count, sum(resource_duration) as duration_count from PageDetail where version='" + str(project_version) + "' and page_name='" + str(page_name) + "' group by resource_type"
        top_sql = "select resource_name, resource_duration, resource_size from PageDetail where version='" + str(project_version) + "' and page_name='" + str(page_name) + "' order by resource_duration desc limit 3"
        resource_composition = query_db(composition_sql)
        resource_summary = query_db(resource_sql)
        top_resources = query_db(top_sql)
        # 对历史数据比对来说 若只有一个版本需要添加默认的比对数据
        history_info = query_db(history_sql)
        g.db.close()
        if len(history_info) > 1:
            pass
        else:
            default_data = {'version': 'syf-next-version', 'dns': 0, 'request': 0, 'dom_parser': 0, 'dom_ready': 0, 'load_event': 0, 'whitewait': 0, 'loadall': 0}
            history_info.append(default_data)
        version_list = []
        for data in history_info:
            version_list.append(str(data['version']).strip("u"))
        resource_type_list = []
```

```
            for item in resource_composition:
                resource_type_list.append(item['resource_type'])
            return render_template('page_detail.html', resource_summary=resource_summary,
history_info=history_info, page_name=page_name,
                                   version_list=version_list,
resource_composition=resource_composition, resource_type_list=resource_type_list ,
                                   top_resources=top_resources)
        # 全局环境变量配置
        @web_app.route('/variable_config', methods=['GET', 'POST'])
        def variable_config():
            if request.method == 'GET':
                return render_template('variable_config.html')
            else:
                return render_template('variable_config.html')
        # 以字典形式返回查询结果
        def query_db(query, args=(), one=False):
            cur=g.db.execute(query,args)
            rv=[dict((str(cur.description[idx][0]), str(value)) for idx, value in enumerate(row)) for row in
cur.fetchall()]
            return (str(rv[0]) if rv else None) if one else rv
        @web_app.route('/ui_auto')
        def ui_auto():
            return render_template('ui_auto.html')
        @web_app.route('/service_auto')
        def service_auto():
            return render_template('service_auto.html')
        @web_app.route('/case_detail')
        def case_detail():
            return render_template('case_detail.html')
        @web_app.route('/cap_auto')
        def cap_auto():
            return render_template('cap_auto.html')
        @web_app.route('/per_auto')
```

```
def per_auto():
    return render_template('per_auto.html')
```

3.2、jquery 发送 ajax

这块儿的是很简单的，jquery 发送 ajax 有固定格式滴~

```
    function test(){
  $.ajax({
            //提交数据的类型  POST GET
            type:"POST",
            //提交的网址
            url:"接口地址(对应 flask 中的路由 url) ",
            //提交的数据
            data:{Name:"sanmao",Password:"sanmaoword"},
            //返回数据的格式
            datatype: "html",//"xml", "html", "script", "json", "jsonp", "text".
            //在请求之前调用的函数
            beforeSend:function(){$("#msg").html("logining");},
            //成功返回之后调用的函数
            success:function(data){
          $("#msg").html(decodeURI(data));
            }     ,
            //调用执行后调用的函数
            complete: function(XMLHttpRequest, textStatus){
                alert(XMLHttpRequest.responseText);
                alert(textStatus);
                 //HideLoading();
            },
            //调用出错执行的函数
            error: function(){
                //请求出错处理
            }
        });
  }
```

我们这里用到两个 ajax 请求，1: 重新执行脚本 2: 下拉框选值的版本

重新执行脚本时，我们发送 ajax 的源码是这样的:

```
// 重新执行测试脚本
function ajaxReRun(){
   var select = document.getElementById("project_name");
   var options = select.options;
   var index = select.selectedIndex;
   var version_text = options[index].text;
   $.ajax({
      //提交数据的类型 POST GET
      type:"POST",
      //提交的网址
      url:"/redo",
      //提交的数据
      data:({"selected_version": version_text}),
      //返回数据的格式 "xml", "html", "script", "json", "jsonp", "text"
      datatype: "text",
     //ajax 请求成功后的事件
      success:function(data){
            if (data=='Env Is Not Exist'){
                  alert("亲~  这个版本的测试环境去哪儿了？ ");
            }
            else{
                  window.location.href =  window.location.href;
                  alert("所选版本的前端性能测试执行完毕");
            }
      },
      //调用出错执行的函数
      error: function(XMLResponse){
            //请求出错处理
          alert(XMLResponse.responseText)
      }
```

```
    });
}
```

我们从下拉框中选择版本信息时 **ajax** 源码如下:

```
// 刷新数据
function optionChange(){
    var select = document.getElementById("project_name");
    var options = select.options;
    var index = select.selectedIndex;
    var version_text = options[index].text;
    $.ajax({
        //提交数据的类型  POST GET
        type:"POST",
        //提交的网址
        url:"/index",
        //提交的数据
        data:({"selected_version": version_text}),
        //返回数据的格式  "xml", "html", "script", "json", "jsonp", "text"
        datatype: "json",
       //ajax 请求成功后的事件
        success:function(data){
            $('#weirtable').html($(data).find('#weirtable').html());
            $('#data_container').html($(data).find('#data_container').html());
        },
        //调用出错执行的函数
        error: function(XMLResponse){
             //请求出错处理
            alert(XMLResponse.responseText)
        }
    });
}
```

我们可以把上面这两个方法写到 Index.html 里也可以单独写到一个 js 文件中然后在 Index.html 文件中导入 js 文件即可。

关于 Jquery 其他的东西 我暂时没用到，同学可以自己扩展这块儿的知识以完善体系。

3.3、bootstrap 基础积累

官网上有教程，自己去搜!

3.4、发布走起~

想在 windows 上发布 flask，如果你不怕坑太多，那就去鼓捣吧。

我是在 CentOS 上部署的，下面是部署过程的详细步骤，仅作参考哦~

1、pip install virtualenv

```
virtualenv venv                                  # 产生虚拟路径 venv
source venv/bin/activate          # 启动虚拟环境（deactivate 可退出虚拟环境）
pip install -r requirements.txt                  # 安装依赖包(将依赖到的包全部写到 requirements 文件中方便管理)
```

2、虚拟路径下安装 uwsgi:

(1) 编辑 uwsgi 配置文件: [uwsgi]

```
#########config.init start##########
[uwsgi]
# uwsgi 启动时所使用的地址与端口
socket = 127.0.0.1:6000
# 指向网站目录
chdir = /opt/usr/AutoMan
# python 启动程序文件
wsgi-file = run.py
# python 程序内用以启动的 application 变量名
callable = surveillance
# 处理器数
processes = 2
# 线程数
threads = 2
# 记录日志（自己添加)
daemonize = /var/log/auto_man.log
```

```
# clear environment on exit
vacuum                = true
#状态检测地址
stats = 127.0.0.1:9191
#########config.init end###########
```

(2) 安装运行 uwsgi

```
                （venv） #: pip install uwsgi
                （venv） #: uwsgi config.ini
```

安装 nginx:

```
    # yum install nginx
    只需在/etc/nginx/conf.d 创建一个 AutoMan.conf 配置即可:
server {
    listen          7000 default_server;          # 默认的 web 访问端口
    server_name   192.168.31.61;       #公网 ip
    #charset koi8-r;
    include /etc/nginx/default.d/*.conf;
    access_log   /var/log/nginx/access_airflask.log;
    error_log   /var/log/nginx/access_airerror.log;          #错误日志
    location / {
        include          uwsgi_params;       #导入的 uwsgi 配置
        uwsgi_pass       127.0.0.1:6000;    #需和 uwsgi 的配置文件里 socket 项的地址
        uwsgi_param UWSGI_PYHOME /opt/usr/AutoMan /venv;    #(虚拟环境下)
```

四：车门关好了没？大家一起发车

相信各位同学对 flask 有了初步的认识，我个人对这个平台的后续扩展构想如下:

1: 添加接口的自动化测试模块，可以在 web 页面上设计接口用例，所见即所得。

需要用到的技术：requests 库，flask-celery-基于后台的作业执行。

2: UI 自动化测试模块，目前设想 UI 这块儿不作为重点，使用 RF 的 ride 来设计编写用例，然后将 test-suite 文件上传到该平台，继而在该平台上选择 suites 并在后台执行 pabot 即可。

需要用到的技术：flask 中实现多文件上传。

3: 安全测试模块，安全测试的水 不是一般的深，敬请期待个人关于安全测试学习的文章。

4: 性能测试过程中的监控，这块儿的东西实现起来也是蛮方便的，目前只想到 tomcat，os，db 的监控。

5: 使用 python 的多线程技术做接口的性能测试。

期待高手和前辈们的指正和沟通~~~~~

Python 自动化测试番外篇—接口测试

◆作者：lamecho 辣么丑

1.1、概要

今天将推出一篇 Python 在自动化测试应用的一个番外篇，接口 API 测试。

1.2、接口测试理论

我们先来认识一下接口是什么，什么又是接口测试。由于我的文章一贯秉承实战为主，对于理论概念性的东西我也尽量讲的通俗易懂一些，比如说接口，我不会把纯理论的含义复制粘贴过来完事。好，闲话说到这，我举个简单的例子来说明什么是接口，对于我们测试人员来说怎么样做好接口测试。比如说某个公司的一个网站有个功能是用来查询一个城市的天气情况，而这个公司也有对应的 APP 应用同样可以做城市的天气查询，那么程序实现上不会在网站开发一套天气查询功能，手机短 APP 也去开发一套城市查询功能吧。理解到这里大家应该明白，我们只需要在服务器端开发这个功能，只要我们的服务器提供一套方法让外部的程序去使用就好了，不管是 Web 还是 APP 你只要去使用服务端提供的天气查询功能就好了，后面就是各自前台展示的问题了。所以这里服务器端提供出来的这套方法就是接口，通俗讲就是方法，只要知道接口的地址及参数的传递，谁都可以来调用这个方法来获取天气情况。

那么我们再思考一下，平时我们做的黑盒测试不就是在网站或者 APP 上去执行天气查询的功能吗，根据具体的功能再去考虑测试用例编写。而接口测试就是在网站和 APP 还未能完成前端的功能展示时，我们提前针对天气查询这个功能做测试，和黑盒测试区别就在于我们不是测试眼睛看到的实实在在的界面操作，而是通过程序调取后台的方法（接口）完成测试而已。那么我们的接口测试原则来说和黑盒测试的测试思路是一样的，在做黑盒测试时你要设计什么测试用例，在做接口测试时也同样的要去这样去设计测试用例，只是没有了 UI 界面方面的测试内容。比如在做 APP 端的测试时，我们是不是要设计一条输入空内容的案例去做天气查询，看看我们的 APP 怎么去处理，是弹出提示框提示输入内容为空还是任何反应都没有，反正只要程序不报错崩溃就行了。然后我

们看看具体在做接口测试时，要怎么做呢？同样的我们也要在接口传递参数时，设计一条这样传递空字符串的接口案例，再去检查服务返回内容是什么，是服务返回 400 报错还是返回 200 正常，而正常的返回的内容又是什么。

所以综述上面讲的，只要我们大家能够做好黑盒测试，再理解了什么是接口，不存在怎么做接口测试的问题。

最后再说说接口的请求一般有 4 种，但是我们一般用到的最多的就是 POST 和 GET 两种方式。区别在于 GET 是不传递参数的请求，POST 是需要传递参数的请求。

1.3、Python 实现的接口测试

本节我们就进入到接口测试的实战，今天我将利用网上现成的两个不同功能的接口去给大家讲解 python 在接口方面的应用。其实自己公司的就有一套但接口测试脚本，但我还是要顾及一下公司的感受的。

一般来说做接口测试，我们应当手上能够拿到后台开发提供的接口文档，但是我今天给大家找的是网络上的案例，我也尽可能的讲的详细一些。

首先我们先来看看这两个接口实现的功能，第一个是

http://www.webxml.com.cn/WebServices/WeatherWebService.asmx

大家可以在浏览器里打开这个网址，没错是一个做天气查询的接口。

第二个

http://www.gpsso.com/WebService/Dream/Dream.asmx?op=SearchDreamInfo

呵呵，一个周公解梦的，声明不是在这里宣传迷信。而这个我之前的程序有实现了解梦的功能。

话不多说，我就直接先上一段 python 代码，先感性的认识一下。

我们点开页面里的 getSupportCity，字面意思去理解获取支持的城市，显而易见是一个查看系统支持哪些城市的接口方法。我们在新页面的滚动到最下面看到如下图的内容，看到没，刚才讲到的 GET，POST 出现了。我们先来看看 GET，介绍了上面是请求，下面是服务器的返回。如果你请求对了服务器会返回给你 200 OK，然后是具体的结果。

HTTP GET

以下是 HTTP GET 请求和响应示例。所显示的占位符需替换为实际值。

```
GET /WebServices/WeatherWebService.asmx/getSupportCity?byProvinceName=string HTTP/1.1
Host: www.webxml.com.cn
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns="http://WebXml.com.cn/">
  <string>string</string>
  <string>string</string>
</ArrayOfString>
```

HTTP POST

以下是 HTTP POST 请求和响应示例。所显示的占位符需替换为实际值。

```
POST /WebServices/WeatherWebService.asmx/getSupportCity HTTP/1.1
Host: www.webxml.com.cn
Content-Type: application/x-www-form-urlencoded
Content-Length: length

byProvinceName=string
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns="http://WebXml.com.cn/">
  <string>string</string>
  <string>string</string>
</ArrayOfString>
```

看看我们的 Python 代码中的内容

```
# -*-coding:utf8-*-
import json
import urllib
import urllib2
url='http://www.webxml.com.cn/WebServices/WeatherWebService.asmx/getSupportCity?byProvinceName='
ret=urllib2.urlopen(url)
print ret.read()
```

执行我们的脚本后，看看打印的返回结果

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmln
  <string>北京 (54511)</string>
  <string>上海 (58367)</string>
  <string>天津 (54517)</string>
  <string>重庆 (57516)</string>
  <string>香港 (45005)</string>
  <string>澳门 (45011)</string>
  <string>哈尔滨 (50953)</string>
  <string>齐齐哈尔 (50745)</string>
  <string>牡丹江 (54094)</string>
  <string>大庆 (50842)</string>
  <string>伊春 (50774)</string>
  <string>双鸭山 (50884)</string>
  <string>鹤岗 (50775)</string>
```

结果还很长只截取了开头的部分。看到这样的结果表示我们的接口请求成功了，并获得了支持的城市名称<string>城市</string>，城市后面括号里是对应的城市代码。回到我们的脚本中看看代码的实现。import urllib2 表示我们在 python 中导入了 urllib2 这个包，当然后面就是通过 urllib2 进行的接口访问，具体的执行也很简单 3 行。第一行拼接我们需要访问的接口地址，那么我们来看看接口地址怎么得到的呢？回到我们的网页里找到 GET 里的介绍：

GET /WebServices/WeatherWebService.asmx/getSupportCity?byProvinceName=string HTTP/1.1

Host: www.webxml.com.cn

自然我们的 url 拼接起来就是 host+get 里边的内容，而?byProvinceName=string 后面的 string 是要我们传递的参数，这里的传递参数要区别一下 post 传递参数，这里是直接在网址里去传，一般是？后面就是参数了。最后的 string 就是具体我们要传的内容，比如我们要查询是不是支持北京这个城市，就把 string 替换成北京。当然我们的脚本里是什么都没传，也就是表示传的是个空值，在接口说明里提到了这样的话：“输入参数：byProvinceName = 指定的洲或国内的省份，若为 ALL 或空则表示返回全部城市；返回数据：一个一维字符串数组 String()，结构为：城市名称(城市代码)。”如果我们传递的是 ALL 或是空的话是返回全部的城市。那么我们如果要传递一个具体的城市名称呢？比如我所在的西安，我们要这样写

url='http://www.webxml.com.cn/WebServices/WeatherWebService.asmx/getSupportCity?byProvinceName='+u'西安'.encode('utf-8')

执行以下看看结果

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <string>查询结果为空！这地区暂时不被支持</string>
</ArrayOfString>
-------------------------

Process finished with exit code 0
```

返回的结果也正常，不过居然不支持我大西安，鄙视一下。说到这里，我们回过头再谈谈我们的接口测试方法上，我们在这个接口上目前为止实现了两种类型的参数传递，并得到了正常的返回结果。那我们如果放到实际的工作或项目上，是不是类似我们在做黑盒的功能测试呢，展开点你可以把设计的功能测试案例的思路方法运用到接口测试上来。

以上是针对天气预报支持的城市查询做了一个脚本，现在我将对它的另一个接口“城市天气预报查询”在 python 里实现。

```
# -*-coding:utf8-*-
from appium import webdriver
import urllib
import urllib2
city=u'北京'.encode('utf-8')
url='http://www.webxml.com.cn/WebServices/WeatherWebService.asmx/getWeatherbyCityName'
data={'theCityName':city}
data=urllib.urlencode(data)
ret=urllib2.urlopen(url,data)
print ret.read()
```

看看我们的脚本，是不是和上一篇的脚本不一样了。是的，这样是我们提到的 GET 请求的写法。我们先来看看这个接口的详细信息的截图。

HTTP POST

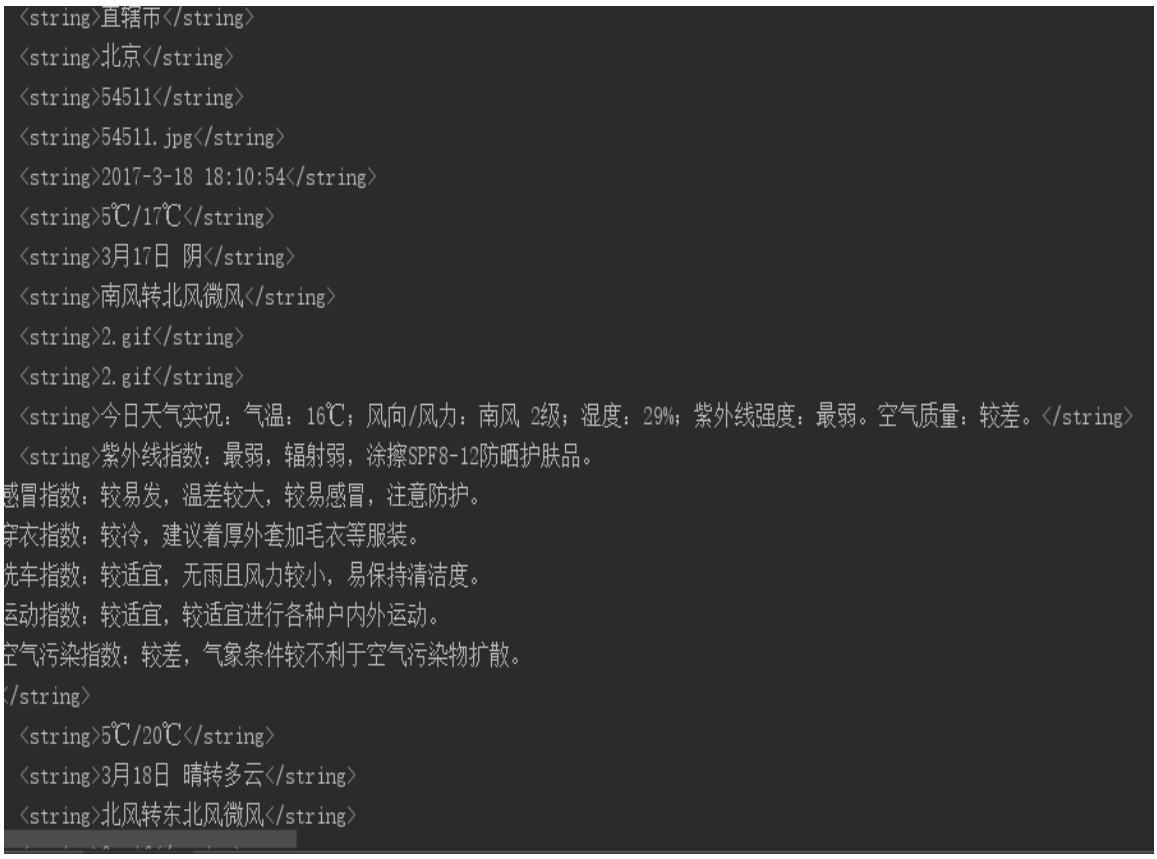
以下是 HTTP POST 请求和响应示例。所显示的占位符需替换为实际值。

```
POST /WebServices/WeatherWebService.asmx/getWeatherbyCityName HTTP/1.1
Host: www.webxml.com.cn
Content-Type: application/x-www-form-urlencoded
Content-Length: length

theCityName=string
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns="http://WebXml.com.cn/">
  <string>string</string>
  <string>string</string>
</ArrayOfString>
```


首先拼接我们的 url，大家应该会了吧。那我们就来看看脚本中新出现的 data 变量，在 python 里它被定义为一个 dict 字典类型，key 值就是我们要传的参数名称 ‘'theCityName'’，那么对应的 value 值是什么呢，自然就是我们要传递的具体参数内容城市名称了。但是我们不能直接传“北京”这个内容进去，会报错。由于我们的后台采用的编码是 utf-8，这里可以从后台的返回示例里看出来（content-type），所以我们要将中文的内容转化为 utf-8 的格式传递进去。处理完这些后再看这句 urllib.urlencode(data)，为什么要再去对我们的 data 值做这样的操作呢，是因为我们请求的 content-type 类型是 x-www-form-urlencoded，所以我们要将 data 转化为系统识别的格式传递进去才行，如果不去遵守规范自然程序是不认识的。

大家理解了这里，在遇到其他类型的时候，自然就要思考怎么去把传进去的数据转化为系统要求的格式了。最后还是用我们的 urllib2.urlopen(url,data)方法去请求服务端得到返回。看看我们的返回结果是什么，可以看到返回结果正常返回了北京的具体天气情况，没有 pm2.5 差评。



好了，原本我还要介绍一个解梦的接口，其实实现方式都已经告诉大家了，有必要给大家留份作业。在解梦这个接口中，先解读它的接口详情介绍，是属于什么类型的请求，然后请求的地址是什么。透露个小惊喜，其实这里还是埋了一颗雷的。

最后感谢大家耐心读完本篇文章，其实接口的测试过程中不止我文中讲的那些，这两篇自动化测试的番外文章只是针对接口测试的一个引子，希望大家在实际的工作多去实践。

拓展学习

- 如何使用 Python 独立搭建基于网页和基于第三方软件测试的自动化测试框架
  http://www.atstudy.com/course/158

如何查看日志以及根据日志来构造数据

◆ 作者：范训海

摘要

我们经常碰到这样的情况：明明数据库里面已经有数据了，但是当我们去对应页面去查询的时候，却发现是空的，这时候我们就需要去后台查看日志了，然后根据日志的 sql 语句来构造数据了。

1、在前台页面执行查询操作

登录测试环境的前台页面，然后进入测试页面，点击页面右上角的“condition”按钮，然后在弹出页面设置 Month from 为：2015-01，再点击“OK”按钮，此时我们可以看到查询结果为空。

2、在数据库里面执行查询操作

登录测试环境的数据库，然后分别查询 BI_DW_RTS_COUNTRY_D 和 BI_DW_RTS_COUNTRY_M 这 2 个表（这 2 张表分别对应日表和月表），查询结果如下：

```
select * from BI_DW_RTS_COUNTRY_D
select * from BI_DW_RTS_COUNTRY_M
```

	CALL_START_DT	CUST_CARRIER_CD	COUNTRY_CD	BIZ_PKG	BIZ_LINE	ROAMER_TYPE	NO_OF_CALLS
1	20150101	BGDWT	LTU	1	1	1	16
2	20150101	BGDWT	LTU	1	1	3	16
3	20150101	BGDWT	LTU	1	3	1	16
4	20150101	BGDWT	LTU	1	3	3	16
5	20150101	BGDWT	LTU	4	1	1	16
6	20150101	BGDWT	LTU	4	1	3	16
7	20150101	BGDWT	LTU	4	3	1	16
8	20150101	BGDWT	LTU	4	3	3	16

可以看到这 2 个表里面都是有数据的

3、在后台查看日志

登录测试环境的后台，输入 logs 进入：/opt/mcb/app/tomcat6/logs 这个目录下，然后输入：tail –f catalina.out.xxx 查看当天的最新日志信息，截图如下:

```
com.ibatis.sqlmap.engine.execution.SqlExecutor - 执行的方法是：iBatisExecuteQuery
com.ibatis.sqlmap.engine.execution.SqlExecutor - select count(*) from (select COUNTRY_C
_DW_RTS_COUNTRY_M where 1=1 and CUST_CARRIER_CD = ? and BIZ_PKG = ? and BIZ_LINE = ? and
START_MONTH >= 201501 and CALL_START_MONTH <= 201701 group by COUNTRY_CD )
com.ibatis.sqlmap.engine.execution.SqlExecutor - 0=ACCLK; 1=1; 2=2; 3=2; 4=1;
```

4、把查询语句拷入数据库并修改

将日志里面的 sql 语句拷到数据库里面，然后把里面的?用后面的参数代替，再执行查询操作，此时可以看到查询结果是空的，说明和页面的查询结果是一致的。

```
com.ibatis.sqlmap.engine.execution.SqlExecutor - 执行的方法是：iBatisExecuteQuery
com.ibatis.sqlmap.engine.execution.SqlExecutor - select count(*) from (select COUNTRY_C
_DW_RTS_COUNTRY_M where 1=1 and CUST_CARRIER_CD = ? and BIZ_PKG = ? and BIZ_LINE = ? and
START_MONTH >= 201501 and CALL_START_MONTH <= 201701 group by COUNTRY_CD )
com.ibatis.sqlmap.engine.execution.SqlExecutor - 0=ACCLK; 1=1; 2=2; 3=2; 4=1;
```

说明一下：0 表示第一个参数，1 表示第二个参数，依次类推。

5、构造数据

根据日志里面的 sql 语句可以判断出该页面主要涉及到的表是:DW_RTS_COUNTRY_M,所以我们需要编辑这个表，将 CUST_CARRIAD_CD 的值改为 ACCLK,BIZ_PKG 的值改为 1，依次类推，修改过后的结果如下图所示:

	CALL_START_MONTH	CUST_CARRIER_CD	COUNTRY_CD	BIZ_PKG	BIZ_LINE	ROAMER_TYPE
1	201501	ACCLK	IRQ	1	2	2
2	201501	ACCLK	IRQ	1	2	2
3	201501	ACCLK	IRQ	1	2	2
4	201501	ACCLK	IRQ	1	2	2
5	201501	ACCLK	IRQ	1	2	2
6	201501	ACCLK	ITA	1	2	2
7	201501	ACCLK	ITA	1	2	2
8	201501	ACCLK	ITA	1	2	2

6、检查前台页面与数据库记录是否一致

数据构造完了以后在数据库里面再次执行查询语句，此时可以查到记录数不再为 0 了，然后进入到对应的前台页面去执行查询操作，此时也能查到数据了。

说明一下：如果在前台页面执行查询操作的时候，后台日志报错了，很有可能是代码写的有问题，要找开发确认一下。

从烧烤摊主到测试主管只用了 4 年，过程却曲折离奇

◆作者：王有缘

摘要:

本文通过“酷爱游戏”为切入点，歪打正着进入测试行业，又以“创业失败”，“面试碰壁”，“频繁跳槽”，“深陷赌博”传达出作者虽有不甘平庸之心，却又无法逃避现实弱肉强食法则。最终找到目标定位，脚踏实地工作从而迈出人生困境，步入重拾自由，爱情的健康之路。

我 92 年出生，籍贯湖南邵阳，毕业之后无一技之长，想出去找工作却不知道自己能做什么，而当年酷爱游戏，整天沉迷在网吧上网玩游戏，当时闪现了一个念头，要是我会游戏开发就好了，在游戏里我想要啥就有啥。

于是，我就在搜索网上面输入关键字：游戏开发，一排排广告出现在眼前，最终选择到长沙雨花区的一个有名培训机构。可让我大跌眼镜的是，我和我爸一起从邵阳坐绿皮车到长沙，对培训机构进行实地考场，原以为是培训游戏开发，结果这只进行测试工程师培训。可我当年根本没听说过测试工程师这个职业，更不清楚这个职业是干什么的。经过她们各种讲解测试工作如何轻松，前景如何好，待遇如何高，反正听完后我也不知道是干嘛，但我还是交了学费，因为认为测试活少轻松待遇高。

一年后，测试培训就读完成，我迷茫了，而且比没学之前更迷茫，更感觉上当受骗了。为什么呢？因为我在培训机构不仅没学到任何实质性技术，反而花费了家里巨额费用（在长沙吃、住、行和学费一年共花了几万块，对当年农村家庭而言这就是一笔巨额）我不知道怎么去面对家里人，也不知道下一步该做什么，又只能回邵阳继续泡在网吧玩游戏，开始逃避现实生活。可不工作终究没有收入来源，穷的供不起自己网费和餐费，怎么办呢？我当时就想了个主意，约堂哥去长沙创业摆烧烤摊，生意再小好歹也是个老板嘛。主意已有，就是不知道堂哥愿不愿意和我一同去创业呢，万万没想到，我的想法和他一说，结果一拍即合。我俩每人出资 5 千创业金，共计 1 万，开启了创业梦！

来到了长沙，才发现之前只是有创业想法，并没有创业计划，如何开头没有一丝丝头绪，连住房头等大事亦没有解决，没有提前看好房，如果在当天租不到房就只能在网吧过夜或流落街头了，两人如果选择去网吧或街头过夜，身上金钱，证件，行李外露绝对是不安全所以行不通，但如果选择去住宾馆一晚上 100 多，虽然一次可行，可要是几天都租不到合适的，连续开房绝对是承受不起的，怎么办，想来想去都没有想到好的点子。然后我决定和堂哥先去网吧上网，边玩边想法子，堂哥性格特别随和，反正我说啥他都答应，比较没主见吧。

到了网吧开好机，我 QQ 一登，哎呀我去，这人呢真是只有想不到，没有做不到的事，然后我就想起了我之前玩游戏，然后结识了几个长沙本地朋友，而且互相加了 QQ。我赶紧在 QQ 上面看看谁在线，正好有两个在线的，我就找了一个感觉平常聊天比较投机的人，给他发消息说明我和堂哥此次来长沙目的，然后交待还没找到房，可不可以先在他家借住几天，只要找到房子，即刻搬走。这个朋友听完来意之后没有拒绝，还赞赏了我几句，说有创业之心的人，值得敬佩，并且愿意提供我俩住宿。当天和堂哥就赶到他家，把行李放好之后，他还请我们吃了一餐饭。那时心里真是暖暖的，觉得世界还是好人多，在家靠父母，出门靠朋友果然没错。

几天后，烧烤摊终于落实下来，也正式营业了起来，早出晚归，早出是要去菜市场，冷冻市场去进烧烤材料，晚归是因为我们开的并不是大排档，而是用烧烤推车摆在外边售卖，到凌晨 1 点后才回家。我在复印店打印了菜单，拿单子跑网吧一个一个问人要不要点些什么，堂哥就主要负责路上的客人，我跑腿，他烧烤。反正累死累活做了 1 个多月，然后进行算账时，我俩有点心凉，因为除去所有成本开销，才赚了 1000 来块钱，平分之后一人就 500 多，感觉还没进厂轻松挣得多，虽然有点不开心，但还没至于就此放弃，毕竟做生意本来就不易，加上我俩没有任何经验，没亏有的赚就已经不错啦，接下来的事却让我俩彻底放弃了这个烧烤摊，我俩还是和往常一样，烧烤摊摆在那个路边，去了同样的网吧，结果遭遇到恶霸把我们的摊子都要给搞烂，然后我和堂哥赶紧推车躲起来放在小路上，黑灯瞎火，我跟堂哥说外边不能摆了，那就不做路人了，我专跑网吧，也还是有点生意的，然而祸从不单行，此次遭到网管的拒绝不允许我在里面进行推送，我问他为什么，之前不是可以吗。网管说没有那么多为什么，说不行就是不行，出去吧！我沮丧的出了网吧，拿着单子垂头丧气的回到了堂哥身边，说：恐怕我们这个摊子搞不下去了，恶霸也赶，网管也赶，有可能遭到某同行报复了。又租不起店

铺，也没有更多钱周转，只能各自去找份工作生活下去，堂哥依旧听从了我的意见，没有提出自己想法，就此，我和他各奔东西踏上寻找工作之路。

我来到了深圳，堂哥去了苏州。我暂时无业游民，堂哥已是工厂员工。我来深圳因为有培训机构的同学在这边已有测试工作，我想他在测试这块稍微能帮下我，毕竟我也学过稍微指点下就能理解。他白天上班的时候我就在各个招聘网站投简历等面试电话，晚上等朋友下班回来，就问他一些面试技巧和测试相关知识，他也力所能及的帮我解答。投了几天简历，一个电话都没接到，有点着急，朋友下班回来的时候我就问他这是为什么，朋友让我把简历给他看，他看完摇了摇头，我知道这简历估计是不合格，然后朋友就在简历这块给我修修改改，让我接着投，果不其然，第二天就接到了面试电话，我十分欣喜，约好面试时间后，我更抓紧时间在搜索网站上找各种测试笔试题和面试技巧，然后在自我介绍这一块也加强了锻炼。在去面试的路上，我内心激动，但也焦虑。激动是因为终于可以争取到上班的机会，焦虑是简历上工作经验作假，我没有任何测试工作经验，加上这是第一次进公司面试，对整个面试流程一无所知。进入公司开始面试时，我还没等面试官开口，我说了句：你好，我简历上经验是假的，不然没经验不会有人给我打电话约面试。面试官脸刷一下就拉下来，说：你这不是在耽误大家时间吗？好了，那你回去吧，这个项目是需要有经验的才行。我原本以为坦诚相待会加分，然后能打消我的顾虑，顺利面试。结果直接当头一棒把我打醒，好不容易有一次面试，结果自己虎头虎脑啥也没面就被遣回家，心有不甘，继续投简历。

由于经历过一次失败面试，再次接到面试电话时，就没有最初的那份激动，紧张，不安。面试路上，我把之前准备的那些面试技巧和笔试题在脑海里再过了一遍，胸有成竹的等待着新一轮面试。自古真情留不住，总有套路伤人心。并没有笔试题，也没有问我知道搜索的那些鸡毛蒜皮问题，偏偏对我简历上的经历进行询问，无奈这个简历根本不是自己做的，朋友帮忙的，上面那些东西我根本就没仔细看过，包括最基本的入职时间，离职时间，做了什么项目，在这个项目过程中工作职责是什么等等都不知道，然后我开始答非所问，显然被再次暴露简历作假，然后又被谴回家……

面试这关我足足花了 2 个月，收到了 100 多封面试邀请函，面试了 60 多家终于跌跌撞撞入了职，入职还需感谢前面几十家的各种盘问，各种刁钻，学到不少知识，因为每一次的面试失败我都会回家进行总结，再把面试时没有回答出来的问题进行搜索解答，最终在面试上面练就一副对答如流的才艺，十分顺利终于通过首家公司面试获得入

职函。可我上班了，对于测试工作职责我依旧不懂，而且特别怕在工作上啥也不懂最后幸幸苦苦面试通过，试用期都过不了立马被开除，担心归担心，真入职后，公司有一个老员工带新人，我心中的那块石头也终于放了下来。

此后从事黑盒测试工作 2 年，虽然只有短短的 2 年时间，我却换了 6 家公司，原因很简单，我觉得在公司没学到任何东西，而且工资低，和工厂上班没什么区别，因为我也完全是在做一些重复性工作，一直在编写测试用例，然后根据测试用例去执行测试，连测试计划，测试报告，甚至最基本的测试流程我却一窍不通，更谈不上具有技术性的自动化测试，性能测试了。我除了会简单的使用测试管理工具，以及会简单的操作 Linux 系统和数据库基本的增删改查，其它啥也不会，就因为自己水平太低又没有深入到测试中，才感觉测试如此枯燥乏味，没有半点技术含量。然后公司换来换去工作职责依旧没变，而我，觉得在工作上仍然学不到有用的技术。所以工作当中继续做一天和尚撞一天钟，上面分配什么任务，就做什么事，做完了就聊 QQ，看新闻。

日复一日，我又这么悠闲的又混过了一年，但在测试工作的第三年，我染上了赌瘾。在群里听说了一款叫某某彩的玩意，十分钟开一期，说钱来的非常快，正好闲来无事，反正可以直接在网上购买，十分方便，中奖也直接到账，很是刺激，于是我就先充了 500 元试玩一下，结果运气挺不错，瞬间拿 500 就博得了 3000 多元，我一下子就被沉迷上了，想当时我一个月工资才 5000 来块，而玩它不到 2 个小时时间，竟然可以赚这么多，加上我自以为是的以为自己运气就是最好的那个，就一心想一口吃个大胖子，就加大了投注，想着赢个十来万就不赌了，赶紧买车装 B，于是深陷泥坑，无法自拔。之前下注都是几十几十，每次开奖实在太快了，输赢也快的不敢想，短短 2 天时间，就输了 2 万多，输红了眼的我，再也控制不住，想扳本，赢回本就再也不碰，可是越输越多，然后投注几百上千的砸进去，可无奈这是个无底洞，把之前上班挣得钱和房租、生活费的钱统统赔了进去，要是这样松了手，我到也轻松，至少认真工作每月按时领工资，还是能正常生活下去，可我因输了这么多钱，早已无心上班就干脆辞了职专门在家研究时彩，还把手里的几张信用卡加上临时额度全部套出，结果全部输光，总共赔进去十五万余元，对于上班每月拿 5000 的人来说这笔 15 万元的帐是天文数字，以我目前的能力，完全偿还不起这个债务，可是欠银行的钱，躲到天涯海角也能被逮住，不想进监狱那就得还钱。思来想去没找到解决办法，这个事情又不敢和家人交待，要是父母知道我赌博输了这么多钱，气坏了身子怎么办，要打死我怎么办，几天茶饭不思，憔悴的看

上去老了四五岁，女朋友也因为我赌博的事和我分了手，生活费，房租费用我都没办法自理了，也没去处，当时只有一个念头，既然还不起，又不想去坐牢，更不想看到父母对我的伤心，失望！我选择轻生！当这个念头产生后，浑身充满了戾气，感觉这个世界再无阳光，整个世界都是黑暗，整个世界已与我无关，我只想与它划清界限，逃离现实，可还没等我下定决心怎么轻生时，赌博事情最终没瞒住，因为欠了银行这么多钱，临时额度到期是不可以最低还款，需要全额付清，而我已经快 2 个月没还银行 1 分钱，银行打我电话联系无果，就私自联系了我家里人，最终我的事情全部被捅了出来，父母听闻此事，感到无比惊讶与震惊，原本只是以为我调皮不听话，但也从不去参与赌博，我爸没说话，那时也不想和我说一句话，我妈说我爸气的头疼，妈在电话里面没有骂我，问我怎样才能解决这件事，我说如果家里一次性拿不出这么多钱，那就能拿多少是多少，先把临时账单给还清，其它钱我再慢慢工作去还，也不至于抓进去，我还这么年轻，要是进去坐牢这辈子就毁了。最终家里凑了 3 万元给我，然后幸亏遇到贵人，前公司美术主管龙哥，我和他说了这件事之后，他也不想我在年纪轻轻，也被这样给毁了，而且知错能改，善莫大焉。于是借了 5 万给我及时把账单给清了，银行事件最终得以解决。

至此之后，我发誓再也不碰赌博，差点让我家破人亡。如果我当时不负责的去轻生了，我死了到是解脱了，可丢下老爸老妈就没法过了，事情过后还心有余悸。虽然银行账单是解决了，可人情账单却没清，家里的钱是找亲人借的，龙哥的 5 万也是借的，这样算来我只是换了个好说话的债主而已，并没有真正解决欠款的事，可每月 5000 还款无济于事呐，在深圳吃住行除掉，所剩无几。这 8 万元我得还到何年何月？因为这些事情我终于找到人生正确的方向，即抛开所有的消极情绪，重新迎接新人生，脚踏实地认真工作还清债务，然后过着自由，快乐平淡的日子就好。又给自己定了个短期目标，那就是当上测试主管，提升工资，尽快还清债务。于是在家自学各种测试理论，学习测试管理，自学脚本编写与性能测试等等，为了更充实自己，还买了练字帖和单反在家练字、摄影。由于之前测试经验，加上后期的努力自学，鼓足了勇气去面试主管一职，功夫不负有心人，面试了两周即坐上了测试主管一职，也就是在测试第四年，我终于坐上了测试主管这个位置。而当上管理后，发现技术可能不是最重要的，但对工作流程，进度把控，为人处事，以及与同事之间、部门之间的沟通必须轻车熟路。

想想这大难不死必有后福，或许没有那一次失败，我还只是一个小职员得过且过，

过着不求上进的生活，人生没自信，做事不稳定东蹦西跳，经历了这么一个人生低谷，反而激发了我的斗志，在工作上积极负责，敢于担当，在生活上充满自信，也十分乐意与人打交道，因为我能感受到每一天的自由，都值得我去珍惜。日子虽然过的平凡，却无比自在。

现在，当测试主管已有 2 年，龙哥 5 万元的债务与家里的 3 万元债务已全部还清，身边还养了只边境牧羊犬，还特别感谢狗狗，因它结缘让我认识了现在的女朋友，如今交往了一年，明年俩人打算领证结婚，自赌博事件以后我的生活一切都比较顺利，也算是因祸得福吧，虽然那件事到如今只有 2 年，可我却比 2 年之前多了份稳重，多了份成熟，多了份担当，少了些轻浮气躁，少了些幼稚，少了些逃避。我觉得，无论身处何职，只要在当前找到自己人生定位，方向和目标，就一定能成功，活的潇洒，自在。最怕的不是当前这段路有多艰难，而是在这段路上跌倒后再也无法爬起来继续前行！

以男性思维获取测试需求，以女性思维设计测试用例

◆ 作者：小王子

今天，办公室的测试小妹热泪盈眶的讲：“世间最遥远的距离，不是生与死，不是天各一方，也不是我就站在你面前，你却不知道我爱你，而是在第十轮测试报告入库的路上，你却告诉我发生了第十一轮需求变更，尤其是这个项目还不支持自动化测试!”讲完这段话，她悲壮的走向了第十一轮测试!

作为一个过来人，讲真，我满怀同情，但更多的是想说：“小妹，现在流的泪，都是当初做测试需求分析时遗漏的水!”小妹负责的项目在短短两个月内进行了十几次的需求变更，要知道，该项目的定位是需求稳定型项目，按照套路来，迭代发生概率不大。**那么，这样一个稳定型项目，为什么它的需求却充满了不稳定性呢？**

原因有两点：一、这个项目的项目经理刚刚从其他项目组转过来，对项目的前世今生以及未来都理解不透彻，制定了一个残缺的需求规格说明；二、测试小妹严格遵守公司相关规章制度，以规格说明作为唯一需求源，继承产生了一个残缺的测试需求分析结果。最终导致陷入了不断加／变更需求的坑!

抛开项目经理的问题，作为测试人员，最根本的问题在于测试需求分析严重不充分。也许在未来的测试生涯中，小妹会懂得：**以男性思维去获取测试需求，以女性思维设计测试用例!**

一、男性思维偏于理性，女性思维偏于感性。对于软件测试来讲，理性，感性分别意味着理性认识和感性认识，也就是说，根据理性认识来进行测试需求分析，根据感性认识进行测试用例设计!

引用一段名词解释，感性认识：指客观事物直接作用于人的感觉器官而产生的，它反映的是事物的具体特征和外部联系，具有直接性和形象性的特点，是对事物现象的认识。而理性认识是对感性认识材料的抽象和概括，它具有间接性和抽象性的特点，反映

的是事物的本质。

看完两者的定义，想必是恍然大悟吧，没错，测试需求分析过程需要对产品功能、性能、可靠性等等特性的测试点进行抽象和概括，而测试用例设计的过程实际上是对测试执行过程的预演，是对测试用例的具体特性和外部联系进行直接形象的描述。

二、男性思维偏于看整体，女性思维偏于看细节。作为一名测试需求分析人员，在进行测试需求分析的过程中，你需要站立于高山之颠（整个产品体系之上），来分析你的测试需求，只有从项目整体去看，才能更全面、精准的把握用户需求，如果在测试需求分析阶段，以女性思维去纠结更多细节，那么，后果就是，细节很完美，需求却总有遗漏，进入日复一日迭代的死循环！

而在测试用例设计阶段则刚好相反，我们需要更多的把握细节，不应该大而化之的去设计测试用例，需要针对已有测试需求点的每一个细节，设计详尽的测试用例，并对测试用例的输入输出、前置条件、运行环境等细节作出清晰准确的描述，确保测试用例的可用性、准确性以及测试需求的覆盖率！

三、男性思维相对受情绪影响要少，而女性思维情绪化相对较明显。这一差异说的是什么，相信大家都深有体会，那么在软件测试过程中，该如何利用这种差异呢？

同样，在测试需求分析过程中，我们应当尽量的采用男性思维特征，即减少“情绪”的影响（注意：这里说的情绪不是指喜怒哀乐，而是指测试人员的个人观点），也就是说，在测试需求分析时，应尽量实事求是，通过行业标准、需求规格说明、用户访谈、竞争对手现状等真实存在的具体的需求作为依据，进行分析。应尽量克服“情绪化”，不要将未经确认的测试人员个人的猜测、想象等擅自加入测试需求，这将有可能导致测试需求分析结果错误。

而在测试用例设计时，则可以较多采用女性思维方式，尽可能的去怀疑、猜测，通过这种“情绪化”的方式来设计测试用例，有利于更多的暴露软件问题！

最后，我想说，开头那个测试小妹就是我……曾走过的弯路，愿未来不再重复！

怎样更好地做好测试工作？

◆作者：CandyHe

2014 年之前，我是一个标准的程序媛，2014 年之后，偶然的机会让我成为了一名互联网测试人员。快 3 年的测试生涯，从当初的一个测试小白鼠，到如今，lead 一个小 team，个中感悟颇多。昨天和 team members 聊了聊，发现大家普遍有着危机感，不知道怎样才能最大限度地发挥测试的作用，让开发们不再觉得测试的工作很 low。其实，我也在摸索中，希望我的只言片字对大家能有所帮助。

现状

现在招纯手工测试的人基本不多，一般都会要求 automation 和 manual testing。但是只有比较大的公司，才会有专门的测试团队；甚至有些大的公司，也没有测试比如 Facebook。几乎所有的公司都会要求开发能够做更多的测试工作。那么，这是不是意味着，越来越少的公司会需要测试人员？

如果一个行业能够生存下去，那么必然有它存在的理由。对于测试行业，也是一样，我们怎样让自己有更多的存在的理由，怎样才能发挥测试人员的作用呢？

我们能做的

听过一个演讲，大意是任何事情，都包含我们能为之努力的，也包含一部分可遇不可求的。对于我们来说，应该关注在我们能为之努力的，而对于那可遇不可求的，天知道。所以对于测试来说，更多的是需要去 improve 我们自己的 skills，这样，不管时代怎样发展，总会有我们的用武之地。

Coding skill

就像前面说的，现在招纯手工测试的公司不多了，那么我们首先能做的，就是做一些 coding 相关的工作，可以从 automation 开始。写过 automation 的人都知道，只要有了开始，你会发现，automation 是最简单的任务。但是 automation 简单，写好又是一件很不容易的事。Coding skill，怎样把 automation 的 code 写的易于维护，经常性地 review

test points 是否 cover 齐全等等，我们能做的事情很多。其实老板不会在意你做的工作是简单还是复杂，只要你能做好他想要你做的事情，这猜中最重要的。

如果你觉得 automation 已经满足不了你想提高 coding skill 的要求了，你可以去 review 开发的 code，从小的 bug 看起，你会得到很多不一样的点。

当然，你也可以着手去改一些小的 bug，前提是要跟开发多些沟通，一般我们的程序猿程序媛们都是很乐于帮助你的。

测试知识的积累

测试和开发一个很大的不同就是，开发要求对某个点有很深入的研究，对于测试来说，更多的是广度方面的提升。拿互联网测试人员来说，有些知识是必须的，比如说最基本的 cookie／cache／session，还有 SEO，网站上的广告植入等等，并不一定说你要各个点都要精通，最起码你要有基本的了解。

还有就是 domain knowledge 了。我们最近招了很多新人，人员流动也比较大，这个时候往往就会有很大的 risk，这个时候其实对于测试人员来说，是一个挑战，也是一个机遇。挑战的是，怎样确保新人做的东西，不会破坏以前的功能。与此同时，这也是你展示测试的重要性的一个机遇。

其实任何职业的人，一般都会有危机感，其实这也是一件好事，说明你是积极向上的。不管测试也罢，其他职业也罢，把事情做到极致，是很重要的一个职业素养。我觉得我做的很成功的一个点，就是产品经理任何时候，在 production 上发现了一个问题，都会让我去做一下 investigation。这个时候，其实我们能做的事很多，简单的处理就是，告诉他这是个 bug，然后注明 bug 让开发去研究解决。而我，会做的稍微多一点，比如说把原始的 feature ticket 是怎样的贴上去，可能的原因会是什么，怎样能够复现等等。时间久了，就会赢得别人对你的尊重。

软件测试中的系统网络图

作者：Sarah xl

系统网络图——额，不清楚实际上有没有这个概念，不过我个人是这么称呼这种关系的。来源是网络拓扑图，如果系统分得比较清晰的话，实际上他就是一个网络拓扑图。大家都知道网络拓扑图描述的是物理布局的关系。系统网络图描述的是各软件系统布局关系，如果每一个软件系统占用一台物理机（服务器）那么他就是网络拓扑图。之所以分开来讲，因为有些公司测试环境为节省资源有多个系统部署在同一个服务器上面，因此跟网络拓扑图又有些区别。

那么，在软件测试过程当中要如何应用这个网络图呢？看这个问题之前，我先告诉大家画这个图的好处。好处有四：一、明确测试分析要点。二、准确定位问题。三、减少沟通时间。四、加深对业务逻辑、程序逻辑的认识。

明确测试分析要点。画网络系统图起码有 50%是应用于此。如何分析？请看例子：

eg1：需求（1）：功能 A 可用显示并可点击，不可用时隐藏。由后台参数 a，b，c 决定功能 A 是否可用。此功能已上线，由于用户误会需要更改需求。需求（2）：功能 A 可用显示并可点击，不可用同样显示并不可点击。

分析需求：此需求并没有涉及到后台逻辑的更改只是需要前端系统把原有的不可用隐藏更改为不可用显示并不可点击。首先，我们根据系统网络图知道前端系统从后端系统获取值来判断功能 A 是否可用。需求由隐藏更改为显示置灰不可用，决定功能 A 是否可用在后台判断，此时后台逻辑不需要更改。

测试分析：分析前端调用后台接口情况。

（1）如果后台做逻辑判断，直接传值给前端系统。返回 1 则功能 A 可用，返回 0 则功能 A 不可用。那么按照等价类的测试手法只需要测试功能 A 可用或者不可用即可。不需要覆盖参数 a，b，c 的设置参数值。

（2）如果前端系统做逻辑判断。设置 a 参数给前端系统返回 1，不设置返回 0。设置 b 参数给前端系统返回 1，不设置返回 0。设置 c 参数给前端系统返回 1，不设置返回 0。由前端系统逻辑判断是否全部返回值为 1 时才可用，否则不可用。那么测试分析需要覆盖 a，b，c 三个参数。对比两种情况，若情况（1）覆盖 a，b，c 参数设置容易造成用例冗余，减低测试效率。若情况（2）不测试 a，b，c 参数设置，则造成漏测。

准确定位问题。

eg2:问题（1）: 前端页面操作功能 A 时报由于参数 c 的原因该功能不可用。

分析问题: 分析可能原因（1）前端读取后台参数 c 时，没有相应的处理。此时报 bug 给前端开发同事。可能原因（2）后台关于 c 参数的逻辑不正确，返回错误信息给前端。此时应当报 bug 给后端同事。分析问题的依据需要参考日志信息，所以一定要建议开发同事打印日志信息。除此之外，对开发接口文档信息需要有一定的了解。

我碰到的情况是，测试同事看到前端报错直接把问题分配给前端同事。前端同事根据描述检查各项逻辑，最后也没找到问题的所在，只能重复找测试同事重现问题。比较熟悉系统网络图的前端同事可能会根据接口的信息定位是否后端返回错误。再把问题转给后端同事。无论是哪一种情况，都浪费大量的人力资源在问题的定位上面。更不用说那些复杂系统了，A 系统开发推给 B 系统的开发，B 系统的开发推给 C 系统开发，C 系统开发找测试重现，然后重新找 A 系统、B 系统开发再次确认。因此准确定位问题后，直接把日志信息附在 bug 报告上面，这样可以避免大量的确认时间。

减少沟通时间。

减少沟通的时间，一方面是 bug 定位沟通上。另外一个是对业务逻辑的理解。如 eg1 例子说明，后台参数 a，b，c 由后台做逻辑判断。那么请教前端同事无疑是浪费时

间的行为，而且也得不到正确的答案。因此，请教前端同事的时候可以这么问他：功能 A 是否可用是由后台返回参数决定的？还是前端逻辑判断的？再根据系统关系图找到对应的系统开发来确定问题。这样有助于测试分析，也有助于后期 bug 定位，更加减少与开发的沟通时间。

加深对业务逻辑、程序逻辑的认识。

根据系统关系图，了解每个系统大概的逻辑判断，加深对业务逻辑的理解。最后，关于系统网络图一定要自己去画，一边画一边理解。单纯看别人画的图，是比较难理解其中的意义的。当然如果你是天才，或者超高技术的心理学家，可以不理会此建议。

我看过很多关于测试文章，一直在强调需要正确的策略，要有方案。不否认他们讲得很正确，但是看完又像是什么都没看。策略需要什么样的策略？方案，需要什么样的方案？如果你也不懂，那么不如从最碎片的这里开始。先画一下系统网络图，再对应去理解每个系统的逻辑关系。

文中举例是最简单的例子，请大家加深理解并扩展到实际工作当中去。如果你们的测试系统有 5 个或者以上这个关系图的应用又当如何呢？因果图、判定表、等价类等等等的测试方法与网络系统图又如何结合使用呢？

此文献给初、中级测试工程师们！受限于个人的技术以及能力，以上观点仅代表个人观点不保证正确性。请大家实践见真章。

《51 测试天地》（四十五）上篇 精彩预览

马上阅读