目 录

LoadRunner 中事务和集合点的前后顺序

◆ 作者：曹承臻

在用 LoadRunner 打压时，经常会用到事务和集合点这两个策略。那么问题就来了：插入集合点和插入事务的前后顺序应该是怎样的呢？

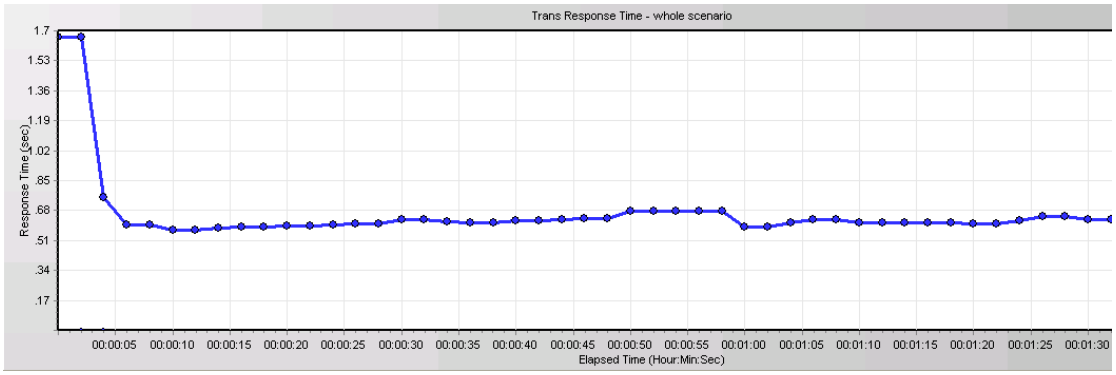
下面我们来做一个实验：

使用同一个脚本，集合点策略为集合 100 个用户后，对服务器进行打压。

1、集合点设置在开始事务代码后面：

```
lr_start_transaction("start");
web_reg_find("Search=all","Text=url",LAST);

lr_rendezvous("getlist");

web_url("getlist",
    "URL=http://[redacted]/discover_agent?myfav=1&cmd=getlist&hid=00C014EECC7850248B15539F6150F11E&c
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t15.inf",
    "Mode=HTTP",
    LAST);
lr_end_transaction("start", LR_AUTO);
```

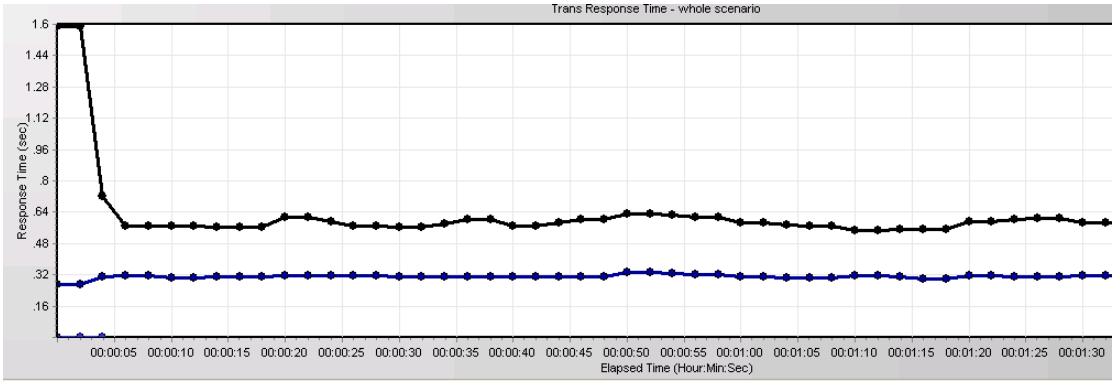
对应的 TPS 图表：（蓝色线为 start 事务，且 start 事务和 action 事务的时间曲线完全重合）



2、集合点设置在开始事务代码前面：

```
lr_rendezvous("getlist");

lr_start_transaction("start");

web_reg_find("Search=all","Text=url",LAST);

web_url("getlist",
    "URL=http://[redacted]/discover_agent?myfav=1&cmd=getlist&hid=00C014EECC7850248B15539F6150F11E&count=
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t15.inf",
    "Mode=HTTP",
    LAST);
lr_end_transaction("start", LR_AUTO);
```

对应的 TPS 图表：（蓝色线为 start 事务，且 start 事务和 action 事务的曲线没有重合）

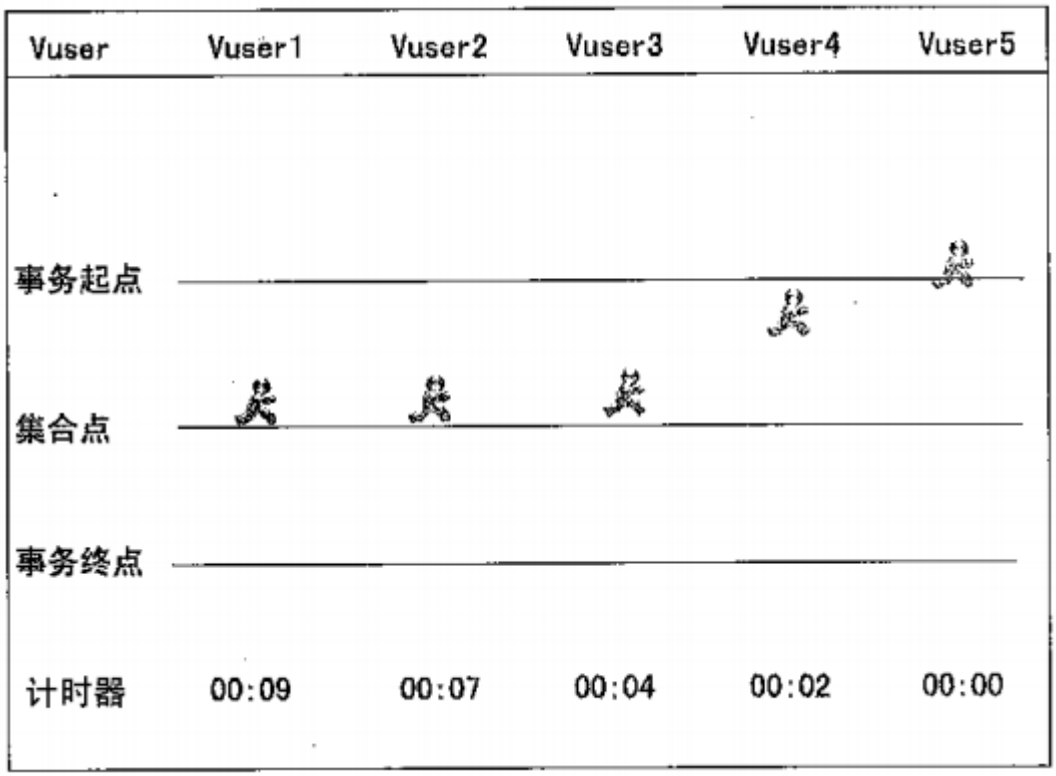


通过两张图表对比不难发现：

将事务设置在集合点之前时的时间比事务设置在集合点之后的时间平均值要多 0.2s 左右。

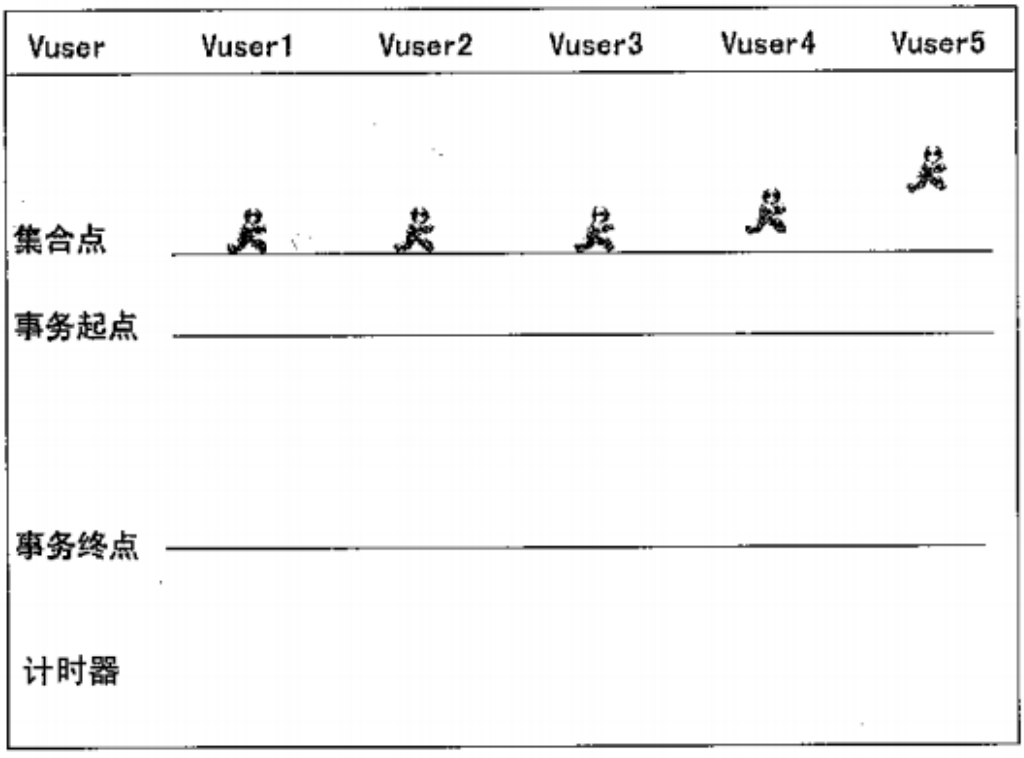
这是为什么呢？让我们来分析一下：

第一种情况，集合点设置在开始事务代码后面，其运行过程如下：



从图可以看到，当虚拟用户运行到开始事务起点时，事务就开始统计时间，但是由于集合点尚未集合完毕，由此可见，事务统计的时间多了集合的时间。

第二种情况，集合点设置在开始事务代码前面，其运行过程如下:



从图中可以看到，虽然 LR 也需要集合点集合完毕，但是由于事务此时还没有计时，所以统计出来的时间更能反映出真实响应时间。

综上所述：我们在插入集合点和事务时，需要将集合点插入在开始事务代码之前。

测试女巫之石头变宝石篇II

◆作者：王平平、仇丹华

摘要： 将统计学_6 sigma 应用到测试管理中，需要使用的 6 sigma 工具如下:
假设检定，双样本检定

一、前言:

测试女巫又来啦，大家还记的吗？前四次我们主要以“找到 bug 产生的原因”以及“提高 bug 产出效率”，“通过自动化节省人力”和“总结工作流程篇”为例介绍 DOE，柏拉图，主效应，交互作用，相关性，鱼骨图，Xbar Chart，One Way ANOVA，Two Way ANOVA，: QFD，DFMEA，量测系统，Johnson Transformation 这些方法，并着重介绍如何将这些方法应用到我们软件测试中。

我们在介绍上述工具时，除了用了一个比较完整的 DMADV 的流程改进方法来对我们的工作流程进行改进，还介绍了在实际工作中只需要对一个比较简单的事情进行分析，进而只用到几个简单的 6 sigma 的工具,这次我们继续介绍这些工具该如何应用到我们的实际工作中，上次我们主要解决“如何去搜集资料去拒接一些无理的工作需求”。

从第 37 期的文章开始我们将从 DMADV 的改进流程转移到“思维”上来，这次我们将继续“将思维进行到底”，这次我们将“思维”转移到我们的测试对象；即研究测试产品和如何客观的判断重要问题测试结果上来:

对于很多公司来说很多产品是共用一个芯片的，如果我对于每个产品根据芯片进行分类，在对每个产品进行测试后，搜集相关的数据，并使用统计学的方法分析总结后得到此款芯片的特点，这样对下一个使用同样芯片的产品会有很大的借鉴意义。例如通过分析我们可以得到这款芯片的软肋在哪里，针对这些软肋，下一个产品在我们在规划测试计划安排测试策略的时候，应该先集中火力测试那些芯片的“弱点”做到有的放矢，知己知彼百战不殆。

对于一个非常重要的测试任务如何客观的给出结论呢：例如在产品规划阶段有根据测试结果决定使用哪款芯片，对于此测试任务，我们测试完之后如何确保我们最大程度

的摒弃了主观因素导致的错误呢？当然对于我们测试人员来说最希望不要增加我们的工作任务的基础上，能够改进我们的工作。这个能做到吗？

原来一个产品做完测试后，我们使用统计学的知识进行分析后可以得到这么多的附加价值；其实这些附加价值才能让我们像一个“闪闪发光宝石”；就像上一期我们说过的，我们要做一个“贪婪”的“资本家”，不断的反复“压榨”我们的项目；不断抽离出宝贵的可量化的经验，将“一堆平淡无奇的石头”变成“一堆闪闪发亮的宝石”；好吧，测试女巫又开始搜集“宝石”了，大家准备好了吗？

奇妙旅程即将开始：Here We Go!

这次我们主要用到的工具为：DOE，主效应，柏拉图，假设检定，双样本检定。在第三十四期的杂志中已经介绍了前 3 个工具的原理和用法，此份文档中也会用到这些工具，因为之前已经介绍过了，这里就不再赘述。对于假设检定和双样本检定是我们新学习的一个新工具和新理论，在这里我们将进行详细的说明，并给大家演示如何应用到实际工作中。

二、6 sigma 常用工具基础知识介绍

1. DOE 实验设计方法

此工具已经在第 34 期“51 测试天地杂志”中的“测试女巫”此文章中有详细的介绍，这里就不再赘述。

2. 主效应分析方法

此工具已经在第 34 期“51 测试天地杂志”中的“测试女巫”此文章中有详细的介绍，这里就不再赘述。

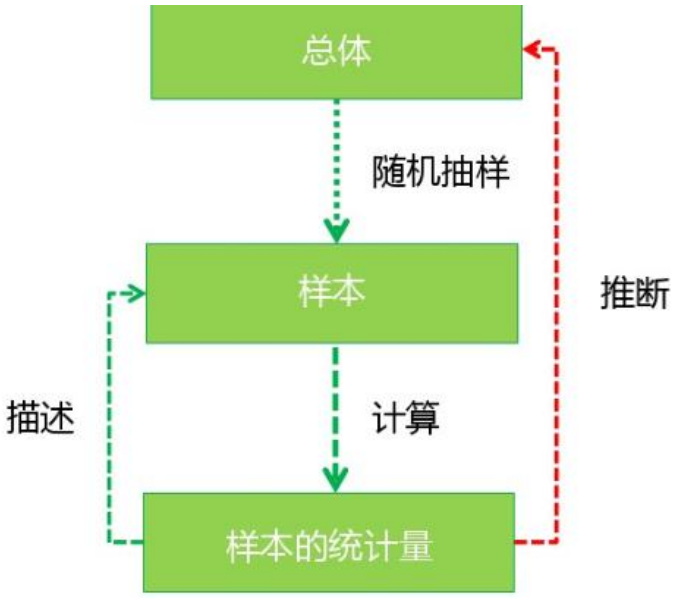
3. 柏拉图分析方法

此工具已经在第 34 期“51 测试天地杂志”中的“测试女巫”此文章中有详细的介绍，这里就不再赘述。

4. 假设检定

1) 定义：对总体进行假设，这个假设是以一定的理由为基础的，但这些理由通常不完全充分，这时就要从总体中进行抽样，对样本进行计算分析，得到结论，通过样本的结论来反推总体的结论。

如【图 1】



【图 1】

2) 假设检定的名词定义

分析方法即为统计学的方法：“假设检定”，它是一个客观的方法，首先它需要先进行假设，然后确定好“能承受决策错误的机率”，根据这个机率来制定可接受的结论。

对于假设检定有两种假设:

即虚无假设（H0），对立假设（H1）。

- 虚无假设（H0）: 通常是研究者想推翻的结论，也称零假设
- 对立假设（H1）: 通常是研究者想证明的结论，也称研究假设

决策结果与实际情况的对比请看【图 2】

决策结果	实际情况	
	Ho为真	Ho为假
未拒绝Ho	决策正确	第II类错误β
拒绝Ho	第I类错误α	正确决策

【图 2】

我们主要研究的是两个决策错误的风险，而在实际操作中我们主要研究的是第 I 型错误 α;注意,这个 α 在统计学中是一个固定的值,例如在我们的工业生产中它可以为 0.05;即能承受产生 α 的风险机率为 0.05; 在不同的领域它的数值会有变化，例如在航空或者医疗领域这个数据会严苛很多。

P-Value: 它的意义在于使用统计学的专业工具 minitab 对于抽样的数据进行计算，算出第 I 型错误实际的机率是多少

3) 使用方法

a. 首先我们对于我们希望研究的问题进行假设，注意这个假设是针对母体即总体的数据进行假设。

b. 确定好“能承受决策错误的机率”一般来说是固定值即 0.05

c. 然后对于母体的数据进行随机抽样，对于样本进行计算 P-Value 值

d. 对比 P-Value 与第 I 型错误 α 的大小，如果 P-value 小于第 I 型错误 α 则拒绝虚无假设，否则接受虚无假设

5. 双样本检定（TWO Sample t-Test）

通过假设检定的方法来比较两组数值，首先定义虚无假设和对立假设是什么，然后使用 minitab 中的“two sample t-Test”来计算此两组样本的 P-value 是什么，然后比较 P-Value 与第 I 型错误 α 的大小; 后得到结论。

三、应用到实际工作中_总结工作篇

根据之前做项目的经验，首先第一步需要安排测试计划和测试策略，刚开始的测试策略均是对各个功能做一次完整的测试，后续在总结前 1 次或者前几次的测试结果，改变测试策略，即对于一些功能进行更为深入的测试，而对于另外的比较稳定的功能则可以减少一些人力，物力的投入。但是我们并没有对于之前做过 的项目按照芯片进行分类，接下来我们以分析一个 WIFI 芯片和 Modem 芯片为例来说明如何总结之前的测试结果。

1. 搜集 A 项目的 WIFI Bug

注意虽然都是属于 WIFI Bug，在搜集数据时要注意对这些 bug 要进行细化分类，以

方便后续我们制定策略，如【表 1】我将 bug 分成 4 类

1) Connection ： 与 WIFI 连接相关的 bug

2) Display ： 显示问题

3) Setting: WIFI 中一些子项目的功能

4)WPS: 与 WPS 相关的 bug

Bug Category	Bug numbers
Connection	37
Display	33
Setting	39
WPS	13

【表 1】

1) 进一步以严重程度进行搜集数据

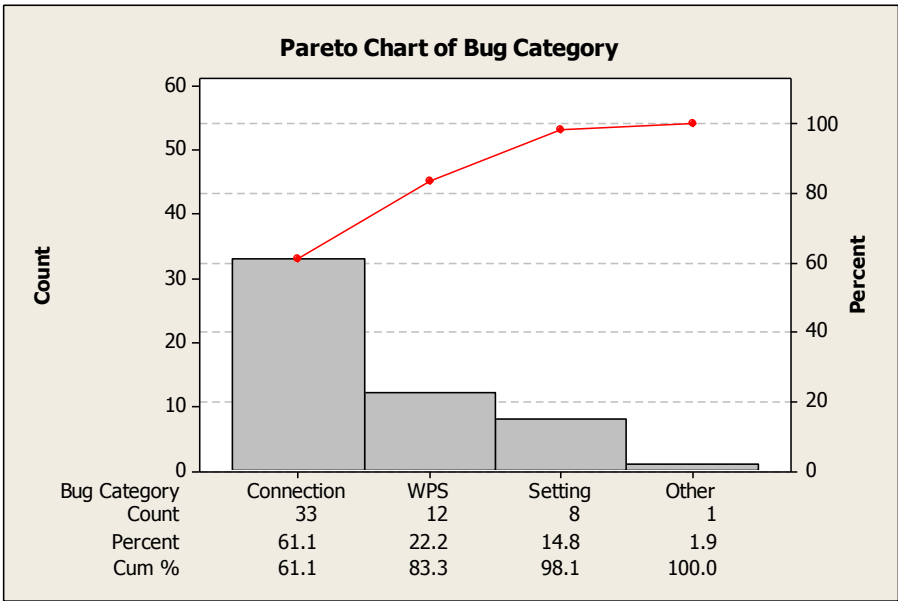
一般我们关心最多的是严重度比较厉害的 bug，所以我们进一步考虑严重程度高的 bug 的分布状态，如【表 2】

Bug Category	Bug numbers
Connection	33
Display	1
Setting	8
WPS	12

【表 2】

Bug 的分类同上

使用柏拉图进行分析如【图 1】，从【图 1】可以看出 Connection 和 WPS 这两方面的 WIFI Bug 已经占了所有 WIFI Bug 的 80%以上；尤其是 Connection 占用了 60%。



【图 3】

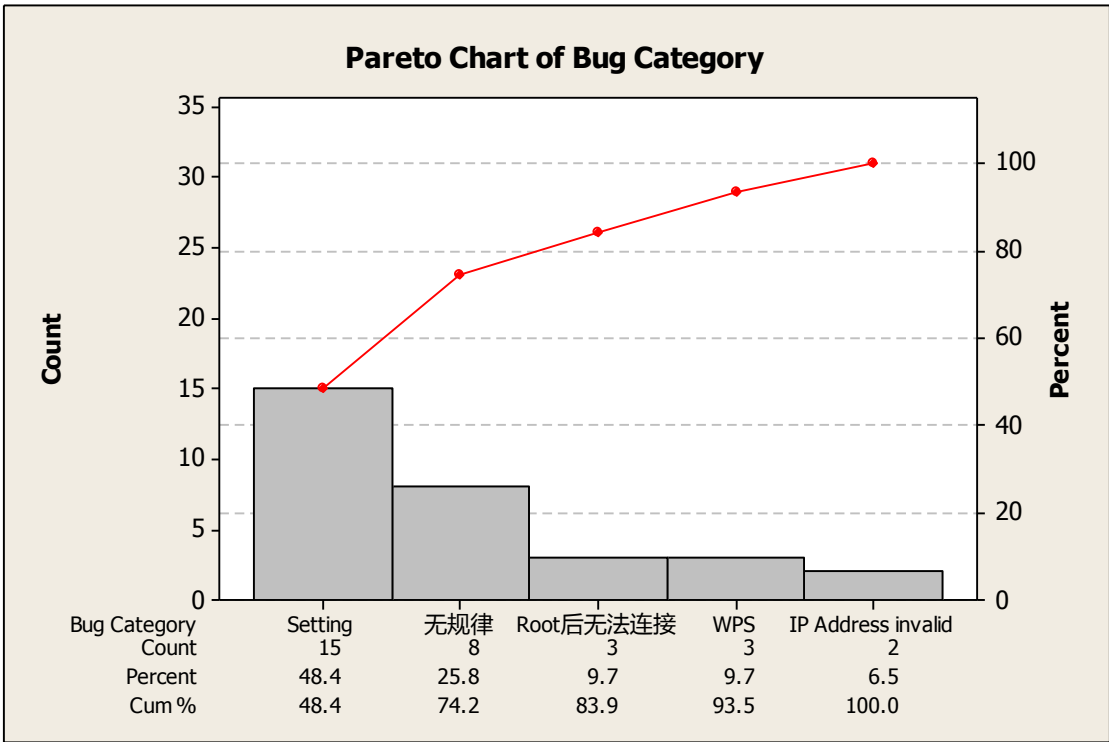
2) 对于 Connection 进行进一步分类，如【表 3】

- 无规律的无法连接：即找不到特定的操作步骤可以复现此问题
- Root 后无法连接：重启待测物后无法连接
- IP Address invalid：客户端获取的 IP Address 是非法的地址
- Setting：对 WIFI 的一些子菜单进行设置后发现的无法连接
- WPS：对 WPS 功能进行操作后发现的无法连接

Bug Category	Bug numbers
无规律的无法连接	8
Root 后无法连接	3
IP Address invalid	2
Setting	15
WPS	3

【表 3】

使用柏拉图进行分析如【图 4】，从【图 4】可以看出 Setting 和无规律占了所有 Connection 的 bug 的比率将近 80%



【图 4】

从上述的分析结果可以得到以下结论:

a. 对于 A 款 WIFI 芯片，最突出的问题是 WIFI 连接性的问题；所以后续我们制定测试策略对于这款 WIFI 芯片首先要将人力和物力向 WIFI 连接性这方面倾斜

b. 对于 WIFI 连接性问题几乎一半以上的问题都是更改 WIFI 子菜单某些功能导致的。所以我们的测试策略可以进一步细化优先测试更改 WIFI 子菜单功能。

2. 总结方法

1) 首先我们要确认我们要研究的芯片是什么

例如上面的例子我们需要研究的芯片是 WIFI 某款芯片

2) 我们需要确认研究的项目是哪个项目，并确认好与芯片相关的功能是什么，以确认我们需要搜集的资料对象是什么

例如上面的例子，我们确认研究 A 项目 WIFI 的 Bug

3) 然后对这些 bug 进行细化分类，一定要注意分类的原则是：尽量对这些 Bug 进行抽象，找出类似的内容，以方便后续制定测试策略。

4) 对于这些数据进行柏拉图分析，找出 bug 分布的“二八原则”，即找出 80%的 bug 分布。

如果 bug 第一次分类不够细化，对于第一次找到的问题，还可以进行再一次的细化，然后再进行柏拉图分析，这样找出的 bug 的分布会对后续制定测试策略有很大的意义

5) 因为芯片和项目是一对多的关系，所以对于一个芯片的研究绝对不能只研究一个项目就宣告结束了，应该尽量多的搜集多个项目的数据，对这些数据进行分别分析，这样才能比较全面的反应出一个芯片的特点。

四、应用到实际工作之如何客观的判断测试结果

1. 实际工作中需要判断的例子

在项目规划初期，经常会有这样的需求：即选择使用什么芯片或者什么平台，举一个简单的例子：有两款 WIFI 的芯片 A 和 B；B 的芯片比 A 要贵一些，对于 Cost 就是竞争力的市场来说，我们当然希望使用芯片 A。但是我们不能全部从 cost 出发来决定问题，需要做测试告诉相关的部门和主管，芯片 A 和芯片 B 在性能上是否有区别。

将这个实际工作中的例子转换为“统计学理论”就是：

虚无假设是：芯片 A 的性能与芯片 B 没有区别

对立假设是：芯片 A 的性能与芯片 B 有差别

我们对于芯片性能的测试主要是吞吐量的测试

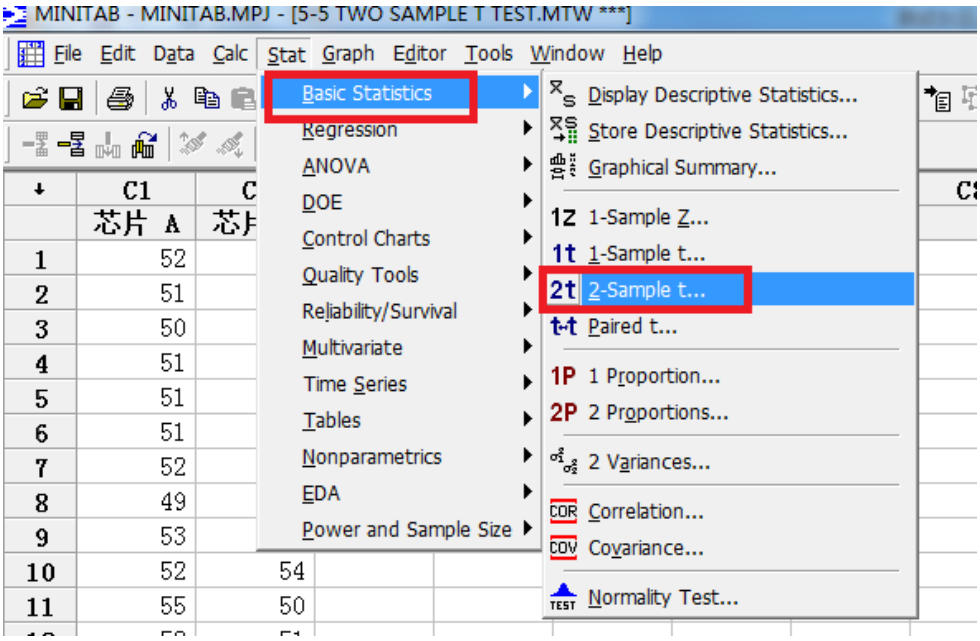
将测试的结果搜集如下【图 5】

芯片 A	芯片 B
52	56
51	52
50	51
51	49
51	50
51	50
52	58
49	48
53	50
52	54
55	50
52	51
50	48
51	50
52	52
53	51
50	54
53	51
47	51
49	52
48	52
52	53
51	47
51	52

【图 5】

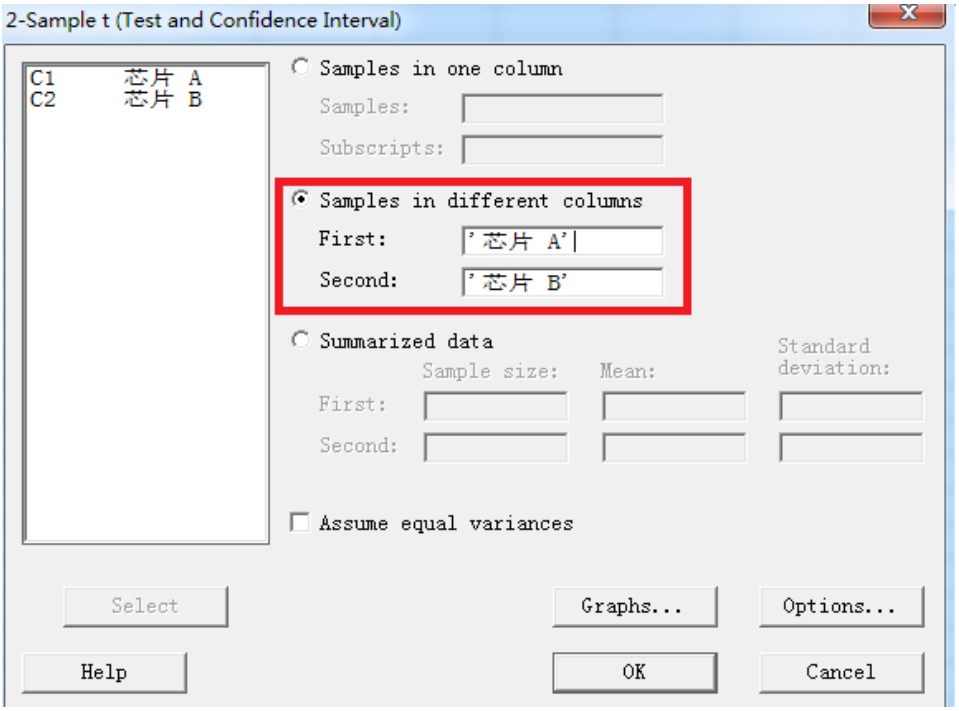
2. 分析步骤

1) 进入 Minitab 的菜单:Stat->Basic Statics->2-Sample t-Test 如【图 6】



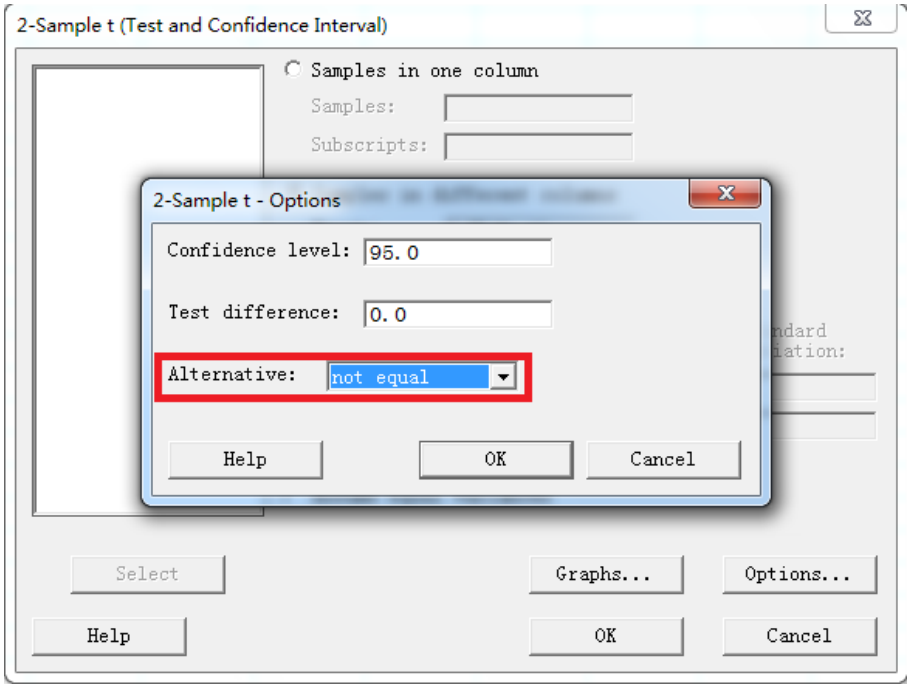
【图 6】

2) 选择两个 sample 到如下栏位如【图 7】



【图 7】

3) 选择 Option，对立假设设置为“不等于”如【图 8】



【图 8】

3. 结论

结果 P-Value>0.05 如【图 9】，所以不能拒绝虚无假设，即虚无假设成立，结论是：A 芯片与 B 芯片并没有明显的差异

```
Two-Sample T-Test and CI: 芯片 A, 芯片 B

Two-sample T for 芯片 A vs 芯片 B

          N   Mean  StDev  SE Mean
芯片 A   24  51.08   1.74     0.36
芯片 B   24  51.33   2.48     0.51


Difference = mu (芯片 A) - mu (芯片 B)
Estimate for difference:  -0.250000
95% CI for difference:  (-1.499096, 0.999096)
T-Test of difference = 0 (vs not =): T-Value = -0.40 P-Value = 0.688 DF = 41
```

【图 9】

五、总结

我们在这一期中介绍了两个重要的知识点，

第一个是在每个项目测试结束后我们需要从芯片的角度去将已发现的 bug 进行分类，然后进行柏拉图分析，得到这些 bug 的分布状态，由于时间原因这里我只分析了一个项目的一个芯片；还是再次强调一遍，大家如果真的想“搜集到闪闪发亮的宝石”，必须持之以恒的对每个项目的每个相对独立的芯片都要做这样的搜集数据以及分析。最后形成一个表格，这个表格就是真正属于我们的“宝石”，且随着时间的推移和我们所经历项目的增多，就会发现我们能找到这些项目中所用芯片的特点，如后续有新的项目，我们在没测试前，就有很多资讯供我们参考，以便我们制定最合理的测试策略。

第二个是项目在进行前的专项测试需求。这个专项测试需求虽然比较简单，但是我们的测试结果却是老板们重大决策的依据，例如我是选择比较贵的芯片还是比较便宜的芯片，如果选择了比较便宜的芯片对我的产品会有多少不良的影响。我们引入的统计学的方法，并没有额外增加我们的工作量，只是将我们的测试结果“翻译”为统计学的语言（假设检定），然后使用统计学的工具，给出一个非常可观的结论。

为什么要这样做？举个例子，一个功能是否有改善，有的人认为成功率提高 1%就算有改善，但有的人认为提高 10%以上才叫有改善。这个例子说明，如果没有统计学的方法，即使是相同的测试结果，不同的人就会有不同的结论，而“假设检定”这个理论以及运用这个理论的“Two sample T-Test”最大程度的降低了主管因素，且这个理论和方法是国际上通用的方法和理论，用来说服老板或者客户，是非常有效的!!

还是那句话：对于 6 sigma 可以带来的奇妙旅程，我将不懈的坚持探索，只有不断

的使用它们才觉得原来很多很有深度的内容并没有彻底的理解，所以“路漫漫其修远兮，吾将充满欢喜的上下而求索”！

参考文献:

1) 有关 DOE 基本概念介绍的网站：http://wiki.mbalib.com/zh-tw/DOE
2) 有关介绍主效应的网站：http://baike.baidu.com/link?url=Yh8AQxA9lNSPCbV3kZAGJs3Zg48QycNDy6iWzElHL6t5QhqnAlE6g5SDhe0xmPt72qHgFHdEWemezR90Ct9_cq
3) 有关介绍柏拉图的网站：http://www.pinzhi.org/thread-15643-1-1.html
4) 有关假设假定的网站：http://wenku.baidu.com/link?url=Ey7MfQW3SuYRt0NPxER9LzwHAPqFl0GJEJeFbSr0USKrbPO1rKcylY2hlvWeIweM75yKPDZrZ8C_5L29k4uLy-A99vDms21r8nppzNeea8_

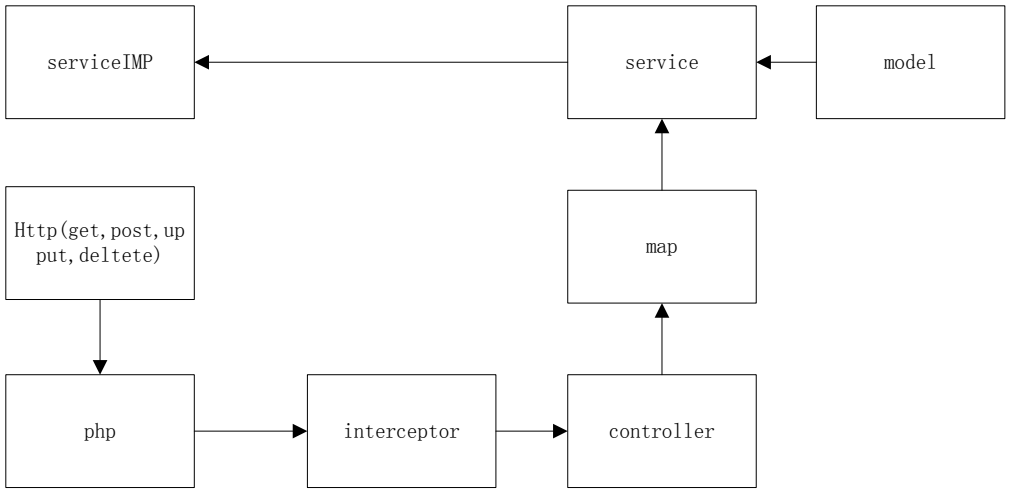
基于互联网金融平台的测试框架设计与分析

◆ 作者：潘杰

目前互联网金融火的一塌糊涂，基于互联网金融平台的自动化测试的项目也是如火如荼的进行。笔者手头上负责一个 p2p 项目的测试框架开发，因此如何设计一套有效的测试框架也成为工作所需和互相交流测试经验的必须。

这个网站的后台主要是 php 和 java， 也就是说，一些基础的服务，如充值提现，投标起息还款，是采用 spring mvc 的框架来写的，然后 php 来调用 java 的 API，java 平台通过 intercepter 将 php 传递过来的 http 请求映射到对应的 controller,controller 再通过 map 映射到对应的服务和实现。

简单来说网站框架就类似如下:



网站一些基本的业务如注册登录，用户中心，投资、红包等等由用户前台触发 php 调用，一些活动、红包、礼品券等等由后台触发 php 调用，充值提现投资起息还款由 java 平台实现。

基于这样的平台的自动化测试框架选型的时候，笔者考虑过以下几种:

一种是采用基于 selenium，集成 thinkphp 的框架来写。主要的原理就是利用 selenium 的 firefox 插件来录制，页面上的 html 元素和 javascript 脚本，然后做 2 次封装将这些录取到的元素和 js 封装成一个个的标准对象，保存到标准对象库，然后再添加一些数据库的数据准备和数据清理函数，以及数据库增删改查语句。

然后在引擎脚本中，引用和调用这些对象的方法，类似如 edit,type 等等，然后就在页面跳转的时候加上驭循环和一些判断，检测页面元素的值是否存在，或者是检测一些方法返回值，或者是采用断言来处理数据库查询到的结果和页面上返回的结果做匹对，可以在以 selenium 为基本的框架的时候，引入 thinkPHP 或者 YII 框架加快开发脚本速度。

另外一种是采用 QTP 方式来使用，其基本原理也和采用 selenium 原理大同小异，唯一的差别就是 QTP 提供了一个很好和强大的基本类库，以及一个很好的对象识别机制 obeject Spy,QTP 的基本类库里面基本什么都有，java,.net, web,乃至于 dephi……

在识别对象的时候可以直接通过 java 里面的类来映射，也可以直接用 web 相关类库里面的类，甚至是 windows 平台类……QTP 提供了多种识别方式以帮助定位对象，这样只用修改少量的对象属性，在回放脚本和编辑核心代码的时候，就可以起到事半功倍的效果，但是 QTP 也有缺点，就是对于 Js 的处理不方便，还有就是时刻需要启动 QTP，而不能和一些开源测试框架特别是 java 开源项目，来实现自动部署测试用例和自动打包的集成。

ruby +watir 的原理和 selenium 也是类似，也是一种基于 WEB GUI 的自动化测试框架，笔者研究甚少，也就不多言了。

但是由于 p2p 行业的特性，这些基于 web 的自动化测试框架有很多不适合的地方，跟投资充值提现相关的这些相关的功能，比较在乎的不仅仅是页面上的一些元素功能的显示，更关心的是这个数据的正确性。

如果采用基于 GUI 的方式做自动化测试，个人感觉针对页面元素的识别和校验往往并不能反映数据正确性，而在处理页面元素异常的时候，也往往无法针对复杂业务逻辑和数据做较强的处理和效益，而且 GUI 是模拟人工处理，在执行效率上面，也是效果较差，而且如果出现某个页面元素无法识别或者异常，有可能中断整个页面的处理，在分析代码覆盖率的时候，基于 web 的方式也不是那么容易分析。

目前的这个项目，java 向 PHP 平台提供的主要是基于 HTTP 协议的 restful 应用，之所以采用 restful，而不用 webservice 来处理传输数据相关，是因为 webservice 即便是采用 json 而不是 xml 来处理传输数据，相比较 restful 也显得较重，json 还需要加密解密解析、序列化什么的，而在 restful 里面直接就可以通过 http 请求对资源进行操作。

因此笔者觉得还是从 controller 层直接进行接口测试比较直接有效，又考虑到 spring 框架里面提供了 mock http 请求的方法，而 web UI 的正确性相对于后台业务数据正确性的优先级就没有那么高了，而 spring 的测试框架里面虽然可以通过断言 controller 层返回的 ModleAndView 对象校验 controller 的正确性，即通过接口测试来效益结果，但是如果 controller 层后面的对象 太多的话，一旦出现问题也不便于排错。

因此基本的测试框架思路就是采用 spring mvc 提供的 mock restful 的工具类，然后引入断言机制和数据库处理，来逐个 controller 分析业务逻辑的正确性和数据正确性。而 sping MVC 本身也提供了一套测试框架，可以通过服务端测试和客户端测试分别来测试。

服务端测试在使用 spring mvc 测试框架之前，可能采取类似如下代码:

```
@Test
  public void serverSample() {
     MockHttpServletRequest request = new MockHttpServletRequest();
     ModelAndView mav = new sampleController.function(parameters );
     ModelAndViewAssert.assertViewName(mav, user/view);
     ModelAndViewAssert.assertModelAttributeAvailable(mv, user);
  }
}
```

采用服务器端测试后，可以采用如下两种方法:

standalone:

```
public class ServerTest {
  @Autowired
```

```
    private MockMvc mockMvc;
    @Before
    public void init() {
        SampleController sample = new SampleController();
        mockMvc = MockMvcBuilders.standaloneSetup(SampleController).build();
    }
}
```

➢ 集成:

```
public class ServerTest {
    @Autowired
    private WebApplicationContext wac;
    private MockMvc mockMvc;
    @Before
    public void init() {
        mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
    }
}
```

➢ 测试:

```
@Test
public void testSample() throws Exception {
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders.get(/user/1))
            .andExpect(MockMvcResultMatchers.view().name(user/view))
            .andExpect(MockMvcResultMatchers.model().attributeExists(user))
            .andDo(MockMvcResultHandlers.print())
```

```
            .andReturn();


    Assert.assertNotNull(result.getModelAndView().getModel().get(user));
}
```

以上是采用 spring MVC 的服务端测试方法，至于客户端则有几种方法，

1.通过 jetty 启动容器，真实映射到 controller 层实现;

2.使用 spring boot 测试

3.使用 mock service server 测试，第三种方式基本上属于使用 resttemplate 来测试客户端比较好的方法;

即先通过 MockRestServiceServer 创建 RestTemplate 的 Mock Server，然后添加客户端请求断言，判断客户端请求的断言是否正确，3、添加服务端响应，检查服务器端相应是否正确。

客户端相关代码在网络上也有很多资源，因此也就不再赘叙，这里主要是提供一种基于 spring mvc 框架和基于 restful 应用如何测试 controller 层的思想。

前台自动化测试方案

◆作者：刘利方

一、自动化测试概述

自动化测试主要应用到查询结果的自动化比较，把借助自动化把相同的数据库数据的相同查询条件查询到的结果同理想的数据进行自动化比较或者同已经保障的数据进行不同版本的自动化比较，减轻人为的重复验证测试。多用户并发操作需要自动化模拟来保障大量用户的执行操作，减少对影响资源的依赖。自动化在迭代 1 开始进行搭建，在迭代 2 能够具备自动化能力。

二、测试目的

本文档主要描述 NPB 的自动化测试粒度、原理及操作流程等。为以后开发人员开发测试用例提供指导。基于敏捷测试驱动开发(TDD)的原理,自动化测试主要达到以下三个主要目的:

1、测试驱动开发。先写出针对测试用例，然后进行功能模块开发。该方式可能加大了开发人员在开发前期的工作量，但是就总体来说，此方式会驱动开发人员更进一步熟悉业务需求，提早预知开发过程中可能出现的各种情况，为后期进入编码测试提供便利。

2、减少或者避免由于模块代码更改，功能扩展等因素带来的重复测试工作。

3、指导开发人员能够更好的对代码进行架构设计，为以后的测试用例的书写，功能的扩展提供方便。

三、测试对象

NPB 项目前台部分的测试对象是业务层(service 层)，测试粒度为 service 层类的所有的或者主要的核心方法，铺盖粒度为：语句覆盖(即开发人员提供的测试用例要能够走通每一行代码)。

四、测试环境

4.1 外部环境

测试环境指的是测试用例的运行环境。测试环境与开发环境共用一个平台。开发代码和测试用例代码分属于不同的 source file 中，测试类与被测试类的包名称相同，类名称不同，这样使得测试类和被测试类的.class 文件位于相同的目录中，测试类可以任意调用与测试相关的开发代码而不会产生耦合或者依赖关系。借助测试工具(如 Junit)进行自动化测试。

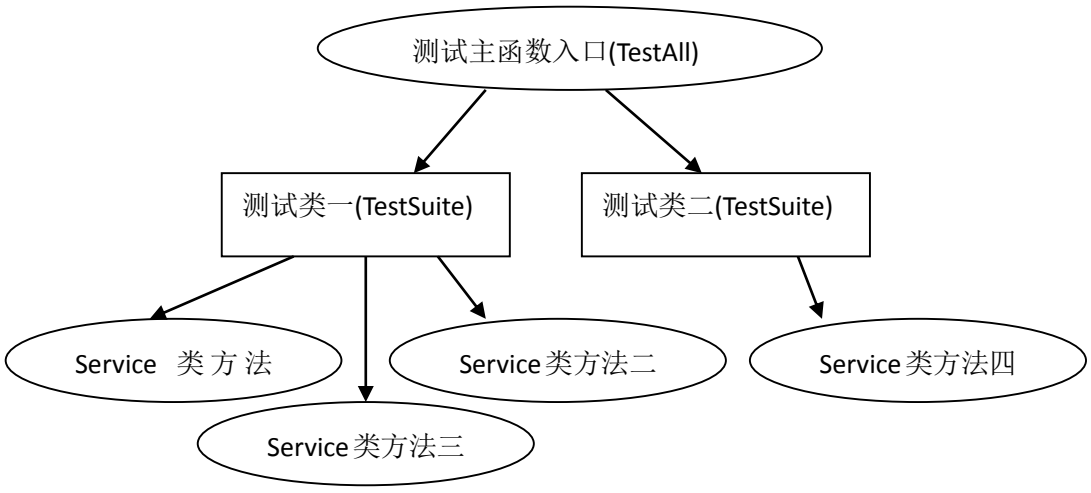
4.2 测试工具

NPB 拟采用 Junit 工具进行测试。Junit 是一种单元测试工具，能够实现自动测试，对于一个要测试的方法，我们输入其所需要的参数（自己构造），然后查看其返回是否符合我们的要求，用 Assert 的方法来比较返回的结果是否正确。

五、测试操作

测试操作主要包括流程图和操作描述两部分。以图文的方式介绍一个测试流程。

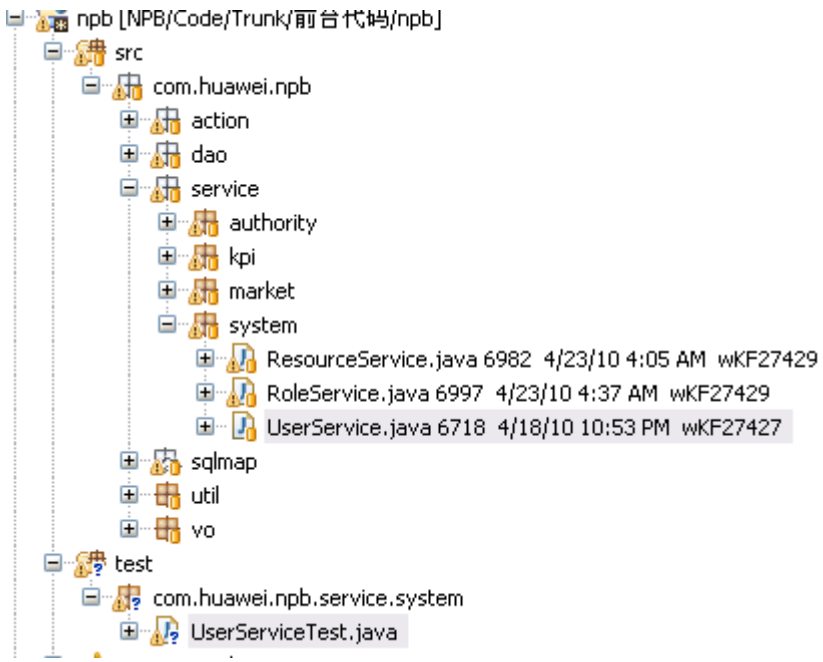
5.1 流程图



5.2 操作描述

目录结构:

首先针对 service 层的每个类，写一个 Junit 测试类，测试类要继承 Junit 的 TestCase 类，测试类位于独立的测试包中。目录结构如下图所示:



针对 service 层指定类的每一个方法写一个测试方法或者多个方法对应一个测试方法。此方法要提供能够覆盖所有方法语句的入口参数，同时针对每一组入口参数要列出理想输出，然后用断言的方法来比较输入是否与预想相符。

测试模块:

NPB 项目会基于系统的核心模块及实际开发情况进行测试，如系统维护(system management)等。

六、测试原理用例示例

下面以简单示例的形式展现自动测试的原理:

现有一 service 层类 UserService，其包含多个方法如：insertUser()、updateUser()和 QueryUserList()等 。

首先我们先了解下基于 iBATIS 持久层开发此方法的返回值问题，insert 操作会放回插入对象的 ID。而 update 操作和 delete 操作会返回更新或者删除操作的条数。

测试用例的命名原则:

测试类的命名以被测试类名后添加 Test 命名。

基于 junit3.8 的测试方法要求是 public 的，无返回值(void)，并且每个测试方法的名字以 test 开头。

UserService 类测试函数如下所示：

```
package com.huawei.npb.service.system;

import java.util.ArrayList;

import java.util.List;

import junit.framework.Assert;

import junit.framework.TestCase;

import com.huawei.npb.vo.system.UserVo;

public class UserServiceTest extends TestCase {

  private UserService us = null;

  //初始化被测试类

  protected void setUp() throws Exception {

        us = new UserService();

  }

  //此方法的代码会在每个测试方法执行完成后执行

  protected void tearDown() throws Exception {

        System.out.println("passed");

  }

  //测试用户列表

  public void testQueryUserList() {

        //从数据库中查出用户总数

        int retInt = 100;

        List list = null;

        UserVo uv = new UserVo();

        try {
```

```
            list = us.queryUserList(uv);
        } catch (Exception e) {
            e.printStackTrace();
        }

        Assert.assertEquals(retInt,list.size());
    }
    //测试用户更新
    public void testUpdateUser() {
        //更新操作理想返回值
        int intExpected = 1;
        //随意定义一个数据库中已经存在的更新对象
        UserVo vo = new UserVo();
        vo.setUserId(100);
        vo.setUserName("test1");
        vo.setDescription("testDes");

        //更新操作会返回更新条数
        int intOutput = 0;
        try {
            intOutput = (Integer) us.updateUser(vo);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

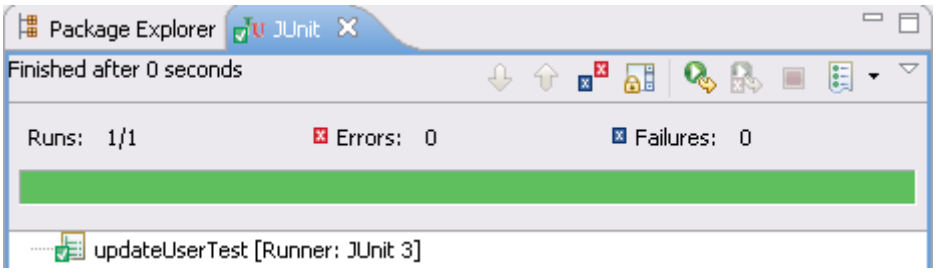
```
        //判断 update 操作返回值是否与理想返回值一致
        assertEquals(intOutput,intExpected);
    }
    //测试插入用户
    public void testInsertUser() {
        //插入理想返回值
        int intExpected = 1000;
        //定义一个插入对象，其 ID 与理想返回值对应一致
        UserVo vo = new UserVo();
        vo.setUserId(100);
        vo.setUserName("test1");
        vo.setDescription("testDes");

        int intOutput = 0;
        try{
            intOutput = (Integer)us.insertUser(vo);
        }catch(Exception e){
            e.printStackTrace();
        }

        //判断 insert 操作返回值是否与理想返回值一致
        assertEquals(intOutput,intExpected);
    }
```

```
//测试查询用户
public void testQueryUserByKey() {
    //查询理想返回值（采用用户 ID）
    int intExpected = 1000;
    //定义一个插入对象，其 ID 与理想返回值对应一致
    UserVo vo = new UserVo();
    vo.setUserId(1000);
    vo.setUserName("test1");
    vo.setDescription("testDes");
    int intOutput = 0;
    try{
        intOutput = ((UserVo)us.queryUserByKey(vo)).getUserId();
    }catch(Exception e){
        e.printStackTrace();
    }

    //判断 insert 操作返回值是否与理想返回值一致
    assertEquals(intOutput,intExpected);
}
//测试删除用户
public void testDeleteUserByKeys() {
    //删除操作理想返回值（删除条数）
    int intExpected = 2;
    //随意定义一个 List 该 List 中包含要删除的已经存在的 UserVo 对象的 ID
```

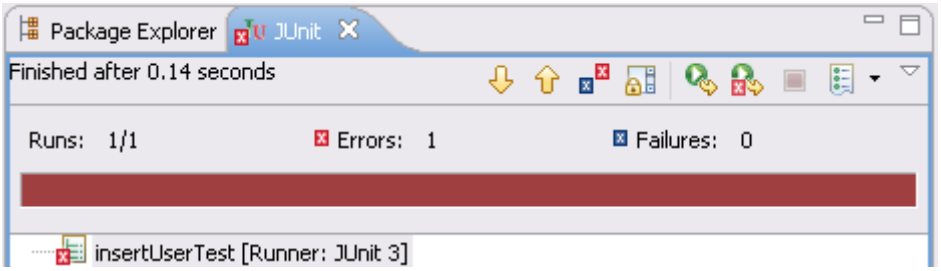
```
        List list = new ArrayList();
        list.add(1000);
        list.add(10);

        //删除操作会返回删除条数
        int intOutput = 0;
        try {
            intOutput = (Integer) us.deleteUserByKeys(list);
        } catch (Exception e) {
            e.printStackTrace();
        }
        //判断 update 操作返回值是否与理想返回值一致
        assertEquals(intOutput,intExpected);
    }
}
```

测试类的正常运行的前提是引用 MyEclipse 自带的 junit.jar 文件，引入的时候要注意 junit 对应的版本有两种，分别为 junit3 和 junit4 两种。我们现在采用 junit3。

方法测试通过显示如下(此图显示测试类的 updateUserTest()测试方法通过):

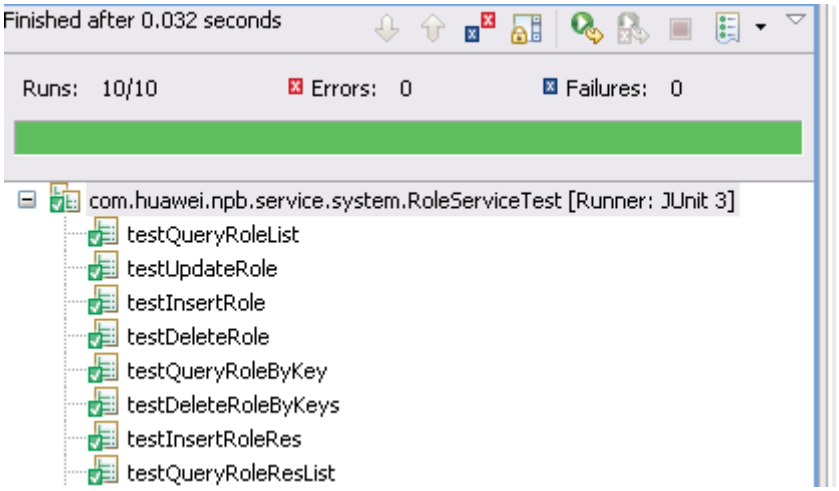


测试不通过显示如下(此图显示测试类的 insertUserTest()测试方法没有通过测试):

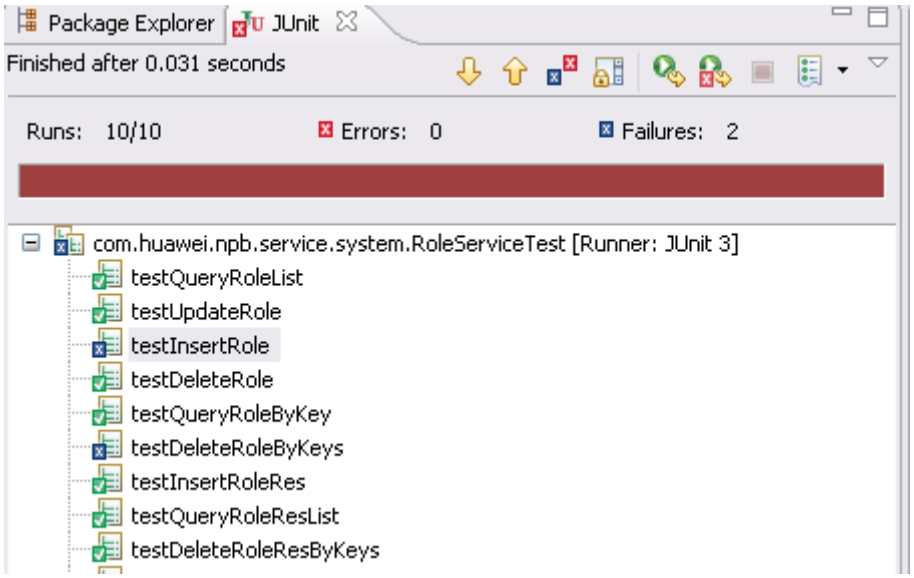


以上显示的是一个测试类中单个方法的运行显示情况。当一个测试类中包含多个方法时，逐一运行每个方法就显得臃肿繁琐。此时我们可以一次性运行整个测试类，来观察每个测试方法的通过情况。

类测试通过显示如下(此图显示测试类的所有测试方法通过):



类测试不通过显示如下(此图显示测试类的 testInsertRole()方法和 testDeleteRoleByKeys()方法不通过):

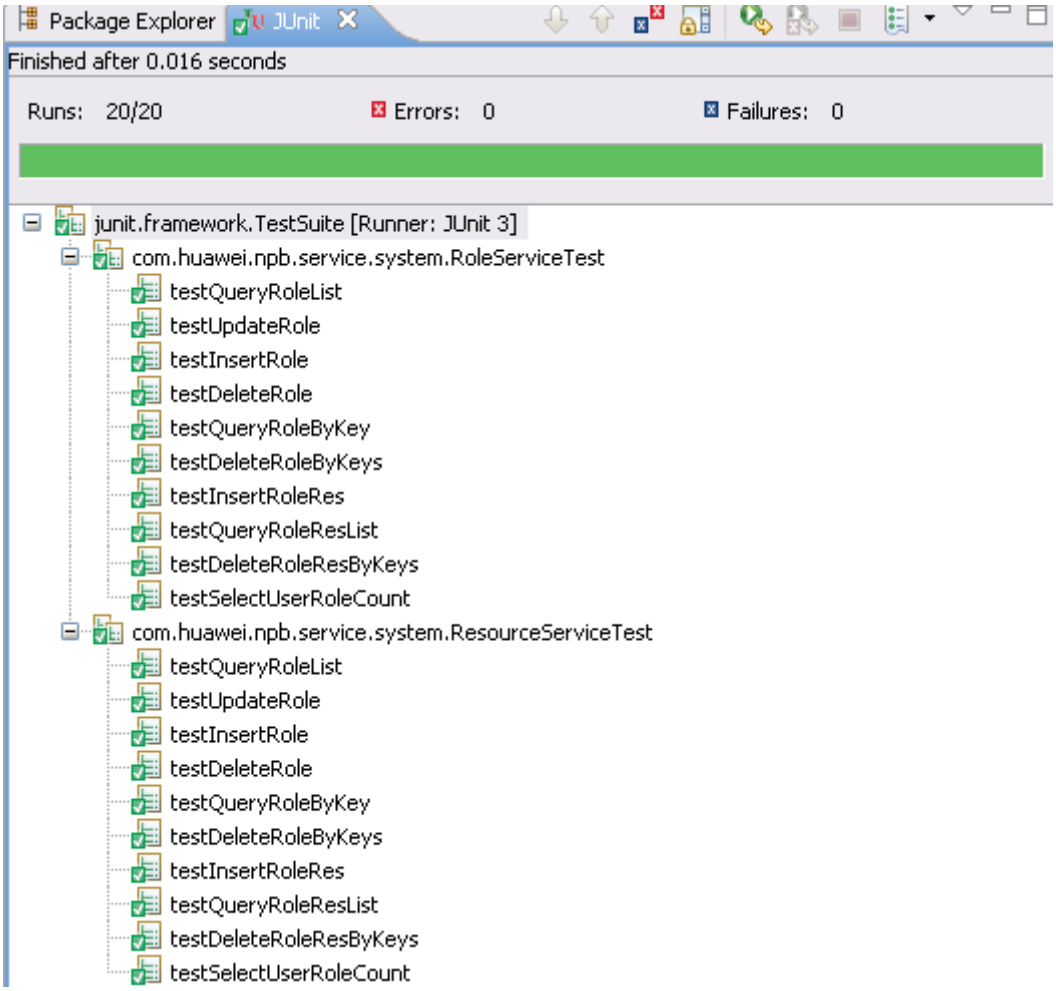


随着开发的不段推进，测试用例会不断的增加，用例繁多，且有一定频率的回归测试时采用以上方法对每一测试类进行逐一测试时，任务量大且极其不便。为此 junit 提供了进一步的测试封装方案。封装类如下所示：

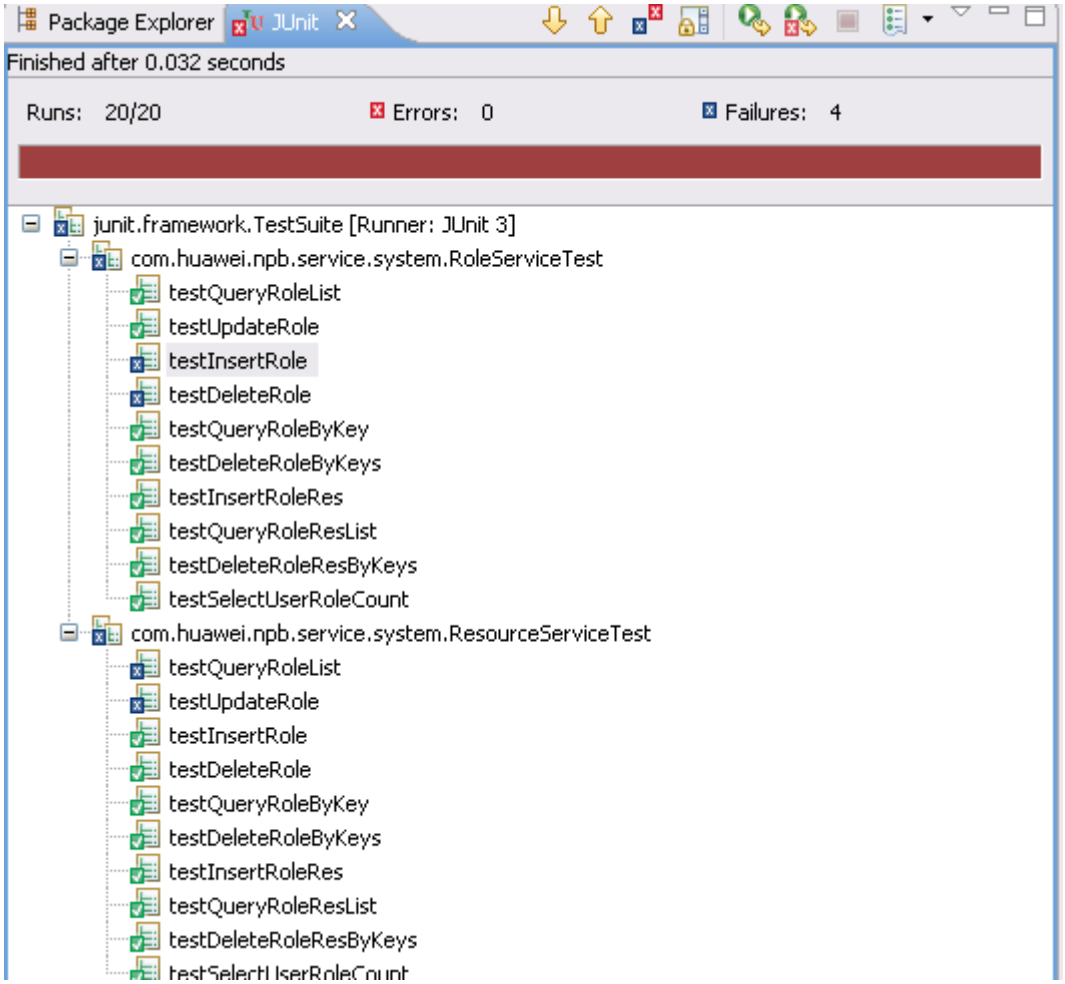
说明：TestSystemAll 类封装了系统维护模块所有的针对 service 层的三个测试类。运行此类，可以查看到系统维护模块所有测试类的每个测试方法的运行情况。

```
package com.huawei.npb.service.system;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;


public class TestSystemAll extends TestCase

{

  public static Test suite()

  {

        TestSuite suite = new TestSuite();

        //包含 UserService 类的测试类

        suite.addTestSuite(UserServiceTest.class);

        //包含 RoleService 类的测试类

        suite.addTestSuite(RoleServiceTest.class);

        //包含 ResourceService 类的测试类

        suite.addTestSuite(ResourceServiceTest.class);


        return suite;

  }

}
```

所有测试类通过测试后显示效果如下图所示:



所有被测试类有部分测试方法不通过测试后显示效果如下图:



七、风险描述

NPB 系统自动化测试存在的风险主要表现在以下几个方面:

1、NPB 项目前期一些辅助性功能的编码已经基本完成。如果针对已开发完的代码再逐一写测试用例有一定的工作量，需要一定的时间，而此时间无法得到根本保障。

2、大部分开发人员对自动化测试比较陌生，没有驱动测试开发经验。在时间紧，任务重的情况下很难达到预期效果。

3、编码之前写测试用例，基本上相当于约 1/4 倍的开发量。在制定的时间内对开发计划和时间安排具有很大的考验。

浅谈测试过程风险管理

◆ 作者：琦少

测试过程风险管理，是个比较大的命题，大家都可以在这方便有自己的思考，今天先分享下我近来的感悟和思考。欢迎大家就这个话题进行讨论和交流。

对于公司内部的测试部门，从项目本身的角度（除人员变动外）常见的风险，我总结为三大类:

1) 时间风险

2) 质量风险

3) 体验风险

下面将分别展开论述，大家不妨将其作为参考点，结合自己的实际项目，进行发散的思考。

一、时间风险

【说明】

由于不合理评估工作量，不合理的测试计划，问题解决不及时等因素，造成的是时间上的违背预期。

【管理方案】

1）和项目经理进行充分的沟通，以便他更好评估进度。

2）监控需求和开发过程的效率，避免压缩测试时间。

3）积极参与，快速决策，避免效率引发的时间风险。

二、质量风险

【说明】

由于需求理解误差造成实现和设计不符，或由于测试疏忽造成质量不达标。

【管理方案】

1) 对需求的把控，不在于评审会开了多久，而在于关键的问题是否解决。

例如:

>新需求是否提出了现有架构或技术条件下，根本无法达到的功能？

>新需求是否和现有功能明显的违背或冲突？

>新需求量是否基本合理？（在项目的预定周期内可以开发测试完成）

不要严重超标，否则考虑拆分版本。

>新需求是否存在不可测试的因素？

在实验室中不好模拟的，可以通过埋点收集数据。

目标:

第一时间暴露需求阶段存在的问题，将需求阶段的风险减最低。

2) 对研发质量的把控，可能来不及一行行的读代码。重点应在方向保证。

例如:

>检查 svn 提交代码后，是否有漏掉的需求点。

>审查是否有太不严谨的逻辑。（通过灰盒测试，或者技术评审）

>审查是否实现和设计冲突。（单元测试，或持续集成测试阶段）

目标:

避免研发过程中的明显误差。

3) 对测试用例的审核，关键不在于是否大而全，而要重点关注，覆盖率，思路，可执行性。

例如:

>特别敏捷的项目中可以没有测试用例，但是要有 checklist。

>多用动宾短语，清晰易懂。

>尽量少解释，描述；突出驱动步骤，和校验结果。

目标:

保证校验环节的效率和严谨。

4) 对 bug 的管理，切忌半途而废，切忌不了了之。

例如:

>开必有因

一定要清晰描述 bug 的复现条件，包括，测试环境，步骤。

最好有截图，有日志（如果需要）

>关必有果

尽量减少 wontfix 关闭的时候，注明分析的原因。

>不姑息

不要让 bug，飘荡太久，任何 bug 都有“保鲜期”。

要有责任心，定期 review 自己的 bug。

>不二过

痛定思痛，分析 bug 产生的原因，看是否能将经验固话。

可以考虑自动化，持续集成。

三、体验风险

【说明】

一些创新，或者探索性设计本身具有未知性，要随准比好应对较低的用户接受度。

【管理方案】

1）一些见仁见智的设计方案，要做好风向把控，发布前说明可能的体验风险。

2）做好云端配置，做好回滚机制。 新产品体验不好的情况下，及时回滚。

四、结语

本着言简意赅的指导思想，以上均是提纲挈领的阐述。当然只是提出测试过程风险管理这个话题，供大家交流讨论。如果有好的想法希望交流，依然非常欢迎。

专项测试之 APP 耗电

◆作者：琦少

一、前言

随智能手机的兴起，移动互联网的发展，优秀的 APP 层出不穷。然人们对优秀 APP 的要求也越发的“挑剔”。从起初的新颖，到后来的稳定，再到现在的流畅，省电等。随着用户日益增长的 APP 质量需求，一个优秀 APP 的背后必然站着一个优秀的测试团队，一个优秀的测试团队必然会针对用户的痛点建立专项测试。这是一个新兴却快速发展的测试领域，也是一个移动互联网 QA 迟早会涉及的领域。今日暂对 APP 耗电测试做些简单的讨论。

二、耗电场景

如果说耗电测试是一个完整的生命，那耗电场景就是它的灵魂。

当然每个 APP 的功能不同，谈业务场景真可谓万万千。但我们适当的抽象一下，或者换个角度思考一下，不难看出有些通用的方法。

软件之所以会耗电，是因为使用了手机的某种资源。手机各个模块的耗电情况是不同的。

如果想了解详细，谷歌官方有统计资料，不难搜索到，我总结的几个典型的耗电场景如下:

1) 定位，尤其是调用 GPS 定位。
2) 网络传输，尤其是非 Wifi 环境。
3) 屏幕亮度
4) cpu 频率
5) 内存调度频度
6) wake_locker 时间和次数

如果您的应用程序涉及到以上的一点或几点，就有耗电风险。

您可以尝试用自动化的方法构建一些典型的场景，并关注电量消耗。

三、检测方法

这部分是今日讨论的重点，关于电量获取的方法有三种:

1) 专业仪器，电表

这个就不展开去说，大致就是把电表连接在电池两级，靠硬件检测电流，电压变化。

通常都有 PC 上的配套软件，能有些图表输出。如果实在觉得抽象，可以想象下心电图。

这种方式需要一定的成本，且环境不熟不太容易。选用的公司不是很多。优点就是精准。

2) 安卓 API（耗电检测 APP）

通过监听 BroadcastReceiver 中的 batterInfo 广播就能了解到电量信息。

特点就是能及时获取，能获取的信息解释如下:

关键信息	类型	解释
status	int	充电情况，充电中还是充满，或者未充电等
health	int	电池健康情况，是正常或者过热等
level	int	电池剩余容量
scale	int	最大容量，通常是 100
temperature	int	温度
voltage	int	电压
technology	String	电池技术，例如锂电池

3) adb 命令（PC 端）

能很方便的打印出一段时间的电量信息。

>在安卓 5。0 之前，只有 dumpsys batterryinfo

>安卓 5。0 之后，有 dumpsys batterrystats

相比之下，在信息全面性，精准性方面都做了升级。

下面以 batterrystats 信息为例简单的说明下:

【命令】

```
adb shell dumpsys batterystats
```

【结果分析】

第一步:

搜索我们应用的包名（例如 com。ksmobile。launcher），定位到如下段落

记住其 UID（下图中左上角 u0a201）

```
u0a201:
  Mobile network: 45.49KB received, 38.51KB sent (packets 195 received, 287 sent)
  Mobile radio active: 25m 46s 331ms (2.6%) 16x @ 3208 mspp
  Wake lock Config Service fetch: 1m 42s 281ms partial (1 times) realtime
  Wake lock *alarm*: 128ms partial (10 times) realtime
  TOTAL wake: 1m 42s 409ms partial realtime
  Sensor GPS: 5m 4s 969ms realtime (6 times)
  Foreground activities: 33m 37s 953ms realtime (58 times)
  Foreground for: 15h 56m 57s 809ms
  Proc com.ksmobile.launcher:BaiduLocate:
    CPU: 510ms usr + 740ms krn ; 0ms fg
    6 proc starts
  Proc *wakelock*:
    CPU: 2s 280ms usr + 4s 230ms krn ; 0ms fg
  Proc com.ksmobile.launcher:wallpaper:
    CPU: 2s 320ms usr + 1s 460ms krn ; 0ms fg
  Proc com.ksmobile.launcher:
    CPU: 43s 650ms usr + 26s 230ms krn ; 54s 20ms fg
  Apk com.ksmobile.launcher:
    Service com.baidu.location.f:
      Created for: 0ms uptime
      Starts: 0, launches: 6
    Service com.ksmobile.launcher.userbehavior.UserBehaviorLogService:
      Created for: 0ms uptime
      Starts: 0, launches: 1
```

第二步:

搜索关键词 Estimated power，定位到如下段落

找到我们程序对应的 Uid 的耗电情况

```
Estimated power use (mAh):
  Capacity: 2800, Computed drain: 2880, actual drain: 980-1064
  Cell standby: 1271
  Uid 0: 571
  Uid u0a12: 353
  Uid 1000: 229
  Screen: 77.0
  Uid u0a198: 65.2
  Uid u0a35: 62.1
  Uid u0a201: 60.2
  Wifi: 49.5
  Idle: 48.7
  Uid u0a138: 43.0
  Uid u0a61: 19.5
  Uid 1001: 13.8
  Uid 1013: 4.46
  Uid u0a199: 3.18
  Uid u0a52: 1.89
  Uid u0a20: 1.11
  Uid u0a3: 0.626
  Uid u0a49: 0.582
  Uid u0a185: 0.448
  Uid 1027: 0.404
  Uid u0a32: 0.298
  Uid u0a64: 0.292
  Uid u0a102: 0.196
  Uid u0a36: 0.193
  Uid u0a40: 0.181
  Bluetooth: 0.133
  Uid u0a91: 0.130
```

四、项目实践

1) 检测电量的安卓 APP，实战

第一步：建立安卓工程，Activity 中监听电量广播

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    batTxt = (TextView) findViewById(R.id.battery_info);
    registerReceiver(mBatInfoReceiver, new IntentFilter(
            Intent.ACTION_BATTERY_CHANGED));
}
```

第二步：解析监听信息

【充电状态信息】

```
private BroadcastReceiver mBatInfoReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        // 监听电量变化，电量发生变化时触发
        String action = intent.getAction();
        if (action.equals(Intent.ACTION_BATTERY_CHANGED)) {
            // 1) 状态
            String statusString = "";
            switch (intent.getIntExtra("status", 0)) {
            case BatteryManager.BATTERY_STATUS_UNKNOWN:
                statusString = "未知";
                break;
            case BatteryManager.BATTERY_STATUS_CHARGING:
                statusString = "充电中";
                break;
            case BatteryManager.BATTERY_STATUS_DISCHARGING:
                statusString = "未充电";
                break;
            case BatteryManager.BATTERY_STATUS_NOT_CHARGING:
                statusString = "充不进去";
                break;
            case BatteryManager.BATTERY_STATUS_FULL:
                statusString = "满电";
                break;
            }
```

【电池健康信息】

```
// 2) 健康
String healthString = "";
switch (intent.getIntExtra("health", 1)) {
case BatteryManager.BATTERY_HEALTH_UNKNOWN:
    healthString = "未知";
    break;
case BatteryManager.BATTERY_HEALTH_GOOD:
    healthString = "挺好";
    break;
case BatteryManager.BATTERY_HEALTH_OVERHEAT:
    healthString = "过热";
    break;
case BatteryManager.BATTERY_HEALTH_DEAD:
    healthString = "废了";
    break;
case BatteryManager.BATTERY_HEALTH_OVER_VOLTAGE:
    healthString = "过载";
    break;
case BatteryManager.BATTERY_HEALTH_UNSPECIFIED_FAILURE:
    healthString = "未知错误";
    break;
}
```

【充电方式】

```
// 3) 充电方式
String acString = "";
switch (intent.getIntExtra("plugged", 0)) {
case BatteryManager.BATTERY_PLUGGED_AC:
    acString = "充电器充电";
    break;
case BatteryManager.BATTERY_PLUGGED_USB:
    acString = "数据线充电";
    break;
}
```

【总电量，剩余电量，稳定，电压，电池工艺等信息】

```
// 4) 剩余
int level = intent.getIntExtra("level", 0);
// 5) 总刻度
int scale = intent.getIntExtra("scale", 0);
// 6) 电压
int voltage = intent.getIntExtra("voltage", 0);
// 7) 温度
int temperature = intent.getIntExtra("temperature", 0);
// 8) 电池工艺
String technology = intent.getStringExtra("technology");
```

第三步：设置 TextView，显示信息。

【布局文件】

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.batteryapp.MainActivity" >

    <TextView
        android:id="@+id/battery_info"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

【Activity 中】

```
        // 显示
        batTxt.setTextColor(Color.BLUE);
        batTxt.setTextSize(28);
        batTxt.setText("1)状态: " + statusString + "\n" + "2)电量: "
                + level + "%\n" + "3)刻度: " + scale + "\n" + "4)健康: "
                + healthString + "\n" + "5)充电情况: " + acString + "\n"
                + "6)电压: " + voltage + "(mv)\n" + "7)温度: "
                + temperature / 10 + "(度)\n" + "8)电池工艺: " + technology);
    }
```

第四步：运行效果如下：

2) 谷歌开源电量分析工具 battery-historian 使用介绍。

第一步：去 github 下载 jar 包，地址如下：

https://github。com/google/battery-historian

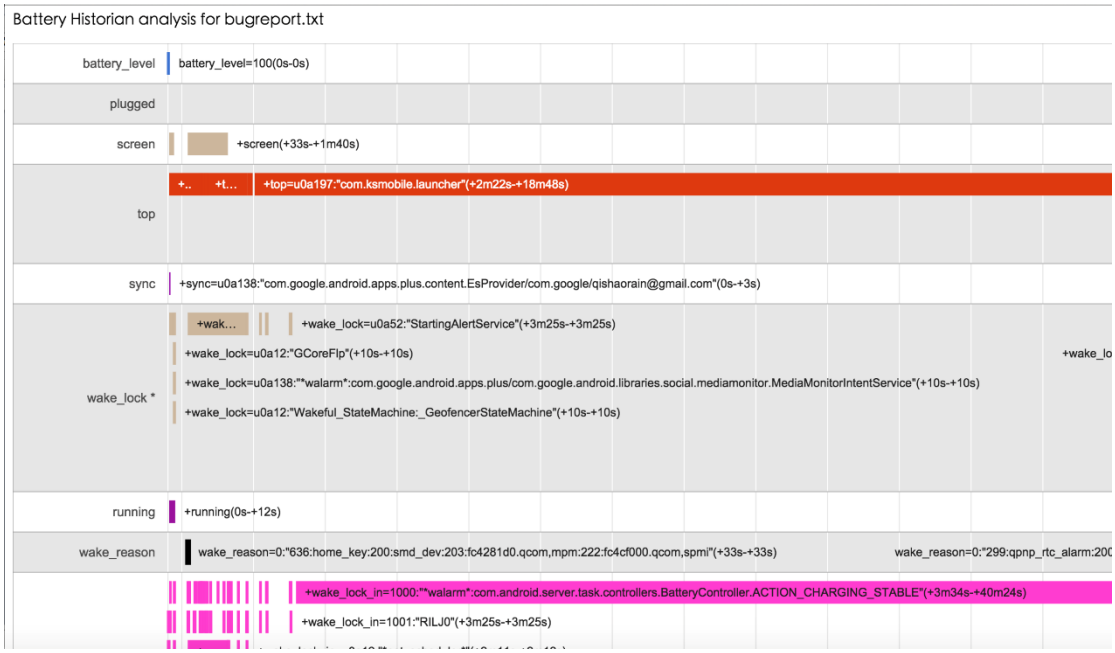
第二步：运行以下命令（确保您电脑上已经安装了 python）：

adb shell dumpsys batterystats > result。txt

python historian。py -a result。txt > result。html

第三步：打开 html 查看结果，部分截图如下：

具体信息量很大，在此就不展开讲解，细致研究有很多奇妙的内容。



3) dumpsys batterystats 结果解析工具，实战。

第一步:

将 dumpsys batterystats 结果输出到 a。txt，以此为分析源

第二步：java 程序去实现解析逻辑

【解析 Uid 和耗电值的对应关系】

```
// 找出Uid<=>耗电值，对应关系
public void readFile() throws IOException {
    // 读取一行
    line = reader.readLine();
    while (line != null) {
        if (line.contains("Uid") && !line.endsWith("x")) {
            hash.put(line.split(":")[0].split("Uid")[1], line.split(":")[1]);
        }
        // 拼接全内容
        contentArray.add(line.trim());
        // 读取下一行
        line = reader.readLine();
    }
    // 关闭
    reader.close();
}
```

【解析包名和耗电值的对应关系】

```
// 解析包名<=>耗电值，对应关系
public HashMap<String, String> show() throws IOException {
    HashMap<String, String> result = new HashMap<String, String>();
    Iterator<Entry<String, String>> iter = hash.entrySet().iterator();
    while (iter.hasNext()) {
        System.out.println("-------开始---------");
        Entry<String, String> entry = iter.next();
        System.out.println("Uid:" + entry.getKey() + "-----" + "耗电: "
                + entry.getValue());
        String tag = entry.getKey();
        System.out.println("#######找包名########");
        String part = contentArray.toString().split(tag + ":")[contentArray
                .toString().split(tag + ":").length - 1];
        String k = "";
        if (part.contains("Proc")) {
            System.out.println(part.split("Proc")[1].split(":")[0]);
            k = part.split("Proc")[1].split(":")[0];
        }
        result.put(k, entry.getValue());
        System.out.println("-------结束---------");
    }
    return result;
}
```

【结果】

```
应用: com.sec.android.app.music<==>耗电: 0.0134 (毫安时)
应用: android.process.media<==>耗电: 0.0493 (毫安时)
应用: com.sec.android.gallery3d<==>耗电: 0.00788 (毫安时)
应用: qmuxd<==>耗电: 0.0141 (毫安时)
应用: com.google.android.apps.plus<==>耗电: 0.130 (毫安时)
应用: com.sec.android.app.videoplayer<==>耗电: 0.00315 (毫安时)
应用: com.sec.android.Kies<==>耗电: 0.215 (毫安时)
应用: com.samsung.dcm<==>耗电: 0.0504 (毫安时)
应用: com.sec.android.cloudagent<==>耗电: 0.0450 (毫安时)
应用: com.samsung.android.app.galaxyfinder<==>耗电: 0.0879 (毫安时)
应用: com.android.systemui<==>耗电: 0.0181 (毫安时)
应用: com.samsung.indexservice<==>耗电: 0.0751 (毫安时)
应用: com.ksmobile.launcher<==>耗电: 3.18 (毫安时)
应用: com.android.nfc<==>耗电: 0.0283 (毫安时)
应用: com.google.android.gms.persistent<==>耗电: 0.0767 (毫安时)
应用: com.cleanmaster.mguard<==>耗电: 0.00343 (毫安时)
```

五、展望

专项测试，刚刚起步，我相信它会把软件测试推到前所未有的高度，同时也令我们这些测试人员面临前所未有的机遇和挑战。以上的一些分享仅仅是这个领域的冰山一角，有太多的东西需要推进和完善，我觉得最好的做法就是以项目实际需求为导向，以兴趣和责任为动力。不断的去学习，实践，一步步的提高自己，提高整个团队的实力。

测试飞虎队是怎样炼成的

◆ 作者：琦少

一、前言

提到飞虎队，大家第一印象就是精英。相信绝大多数公司都希望能组件出一支优秀的测试队伍，来支撑自己的业务，很多公司都喊出了精英化战略。既然如此，就命中一个话题——测试人才战略。

有几个问题是不得不面对的:

1) 如何选拔人才

2) 如何培养人才

3) 如何使用人才

一个公司只有回答好以上三个问题，才算真的准备好去组建优秀测试团队。同样，一个公司里有能解决好以上三个问题的测试主管，才算是为这支“飞虎队”安排了一个合格的主帅。当然这三个问题也是去一个公司的面试者比较关心的问题，除非你在这个公司不图发展，或者不喜欢测试行业。

二、关于选拔人才

人才选拔是测试主管和 HR 深度合作的环节。现在企业，总喜欢搜索简历中的“自动化”，“测试开发”，“白盒测试”，“敏捷测试”等关键词。认为这些就是测试人才的标准。但实际经验来看，仅凭这点筛选出的人才合格率并不高。对于测试这个行业，往往存在如下问题:

1) 面试者有意在简历中描述关键词，但实际相关技能水平不高甚至没有。

HR 可以适当了解相关的技术，通过一些问题加强第一关的筛选。例如:

>如简历中提到自动化测试

可问下应聘者自动化测试的条件，自动化运用的项目。

>如简历中提到白盒测试

可问下应聘者白盒测试的常用方法，在项目周期中如何组织开展白盒测试。

>如简历中提到敏捷测试

可问下应聘者，采取哪些有效的方法，保证敏捷测试顺利展开。

如果，无法回答自动化测试的适用条件，无法回答采取何种措施保证敏捷测试，只能说明应聘者临时抱佛脚，投其所好而已。这种情况要适当避免，以提高面试的效率和合格率。我相信这是 HR 或者 HRBP 可以做到的。

2) 对于测试行业，性格态度略重于技术水平。

如果通过第一关，证明有一定的技术基础，接下来面试官，一方面深入考察业务需要用到的相关技能，更重要的是考察应聘者的特质，是否适合测试。具体考察方法可以不同，但建议关注如下考察点:

>是否具有敏锐的观察力

>是否细心耐心，具备责任心

>是否善于描述问题，善于沟通

>是否有上进心，且积极主动

人才的特质，决定未来的可培养性，尤其是在招聘骨干员工的时候一定要慎重考虑。 分享些具体的场景: 面试官通常喜欢问优点缺点，如果一个应聘者说自己的缺点是马虎，孤高，不耐烦等建议就不要考虑了。还可以通过提问有怎样的学习目标？近期围绕这个目标做了哪些努力？取得什么样的阶段成果？这三连问能考察面试者是否真有上进心，是否真的积极主动。

3) 工作年限，和实际经验不成正比。

这是比较头疼的一个问题，也是我见过最多的一个问题。例如我们希望有个五年工

作经验的测试人员。

有些面试者写着四五年的工作经验，实际经验其实只有两年。也许他们并没有说谎，真的在一个公司工作了五年，但是后三年在重复前两年的工作内容和经验。这就给自己的发展设置了障碍。与之相反，有些人工作了也许只有三年，但三年中，积极主动的承担更多的项目，争取更多的锻炼机会，那么很有可能两年就达到四五年的工作经验和水平。

如果问我的选择标准，我会说实际点好，我相信绝大多数公司也会喜欢后者。这也提醒所有的测试人员，别把自己做废了，工作中尽可能积极主动，多承担，多锻炼。

三、关于培养人才

到了仁者见仁智者见智的环节，关于人才培养没有最好的方法，只有最合适的。今本着求同存异的原则，分享一些感想。培养人才如同练兵，一个测试团队也很像一直军队。

首先，自然是作战能力（相关业务领域的测试技能），通过面试的筛选已经保证入围者有一定的基础。不妨采取提纲挈领式的方法，不断的总结分享工作中的知识体系，让员工有的放矢，针对性的提高自己相关技能，必要时候给出即时的指导。

这样之后，只能说明团队能“打仗”，如何“打好仗”？这时候，几个核心词，就浮出水面，“凝聚力”，“合作意识”,” 团队精神“,” 效率“,” 执行力“。每个展开说都是一个话题，篇幅关系，在这不做细致探讨。总之，要做到一种效果，主帅有谋，将士用命，上下一心，斗志昂扬。自然就能攻必克，战必胜。话听着有点玄乎，仔细琢么确实是这个道理。要明白，要的不是个人英雄，不是侠客，是组织，是团队，这才才是目标。

分享三个典型场景的技巧吧:

1) 关于职业等级评定

细节不说，只强调一点，尽量让技术考核和实际工作结合。

2) 关于部门分享会

彻底废弃一个黑板下面一群人的听讲模式吧。

可以带大家实际演练，采取项目实战的方式。不懂？请借鉴下现在培训机构的

授课模式。

3) 关于兴趣小组

不可强求，真正凭兴趣聚在一起讨论一个问题，给自己定个目标，要有输出有成果。

四、关于使用人才

我看多很多关于使用人才的文章，也算是有点百家争鸣的味道，自然也不乏一些奇思妙想。今天我只想化繁为简，说些简单实际的话。

1）尽量让员工做自己擅长且感兴趣的事情。

兴趣是最好的老师，也是最强大的动力。可以，古人有诗：”衣沾不足惜，但使愿无违”。

你定的目标或者绩效要能充分的调用员工的兴趣和积极性。

2）少许诺，多兑现

言必信，行必果，大饼画多了，望梅止渴用多了，或者狼来了的故事讲多了

迟早会在某些时刻爆发出难以想象的后果。

3）因材制宜

面试诚然是选拔人才中重要的环节，但有句话，路遥知马力，日久见人心。

经过一段时间的工作后会对下属于比较清晰的了解。通常有以下几种情况:

【有才有德，供着用】

相信一些朋友看到这里会质疑，有才有德为何不可玩命用？其实道理很简单，尽管有些人能干肯干。但人的精力毕竟有限。有才有德之人毕竟是少数。他们是一支团队的榜样和标杆，人们不仅仅学他们如何做事，也必然关注他们有怎样的待遇。如果这样的人都用到累死，谁还希望自己成为这样的人？个中道理细品之下，大家都会明白。

【有才少德，激励着用】

这样的人就像是睡着的千里马。如果因为他睡着了，就否定它的能力，真的是很大的损失。想办法叫醒他，鞭策他。通常情况下，只要利益到位，鞭策恰当，或可成为一个团队的主要输出力，因为他们的格言就是有多少好处做多少事。并且只要愿意做，也

一定能做成。

【有德少才，培养着用】

憨厚型的典型代表，通常可以称之为“潜力股”。要有意识的多提供锻炼的机会，比如多参加项目，多去参加技术问题的解决，实践中不断提高个人的能力，能力是靠培养的，作为主管能给的就是合适的机会。

【无才无德，直接不用】

这种就不多说了吧。

五、结语

谈到这里，相信一些同胞颇有感触，想要做些什么，或改变些什么。那么“just do it”。

可能也有些人困惑了。说到这没有太多技术细节，也么有所谓的“大招”。对此我想说，授之以鱼，不如授之以渔。先传其道，再通其业，方可解惑。一旦，摆正心态，端正态度，领悟其中的道理，我相信，凭借自己的驱动力，每个人都能有很大的提高。

如何进行 web 服务的性能测试？

◆作者：小白松

摘要： 本篇主要围绕着性能测试要解决的问题，以及如何进行 web 服务的性能测试，为大家详细的讲解性能测试各阶段要做的那些事儿。

随着浏览器功能的不断完善，用户量不断的攀升，涉及到 web 服务的功能在不断的增加，对于我们测试来说，我们不仅要保证服务端功能的正确性，也要验证服务端程序的性能是否符合要求。那么性能测试都要做些什么呢？我们该怎样进行性能测试呢？

性能测试一般会围绕以下这些问题而进行:

1. 什么情况下需要做性能测试？
2. 什么时候做性能测试？
3. 做性能测试需要准备哪些内容？
4. 什么样的性能指标是符合要求的？
5. 性能测试需要收集的数据有哪些？
6. 怎样收集这些数据？
7. 如何分析收集到的数据？
8. 如何给出性能测试报告？

性能测试的执行过程及要做的事儿主要包含以下内容:

1. 测试评估阶段

在这个阶段，我们要评估被测的产品是否要进行性能测试，并且对目前的服务器环境进行粗估，服务的性能是否满足条件。

首先要明确只要涉及到准备上线的服务端产品，就需要进行性能测试。其次如果产品需求中明确提到了性能指标，那也必须要做性能测试。

测试人员在进行性能测试前，需要根据当前的收集到的各种信息，预先做性能的评

估，收集的内容主要包括带宽、请求包大小、并发用户数和当前 web 服务的带宽等

2. 测试准备阶段

在这个阶段，我们要了解以下内容:

a. 服务器的架构是什么样的，例如：web 服务器是什么？是如何配置的？数据库用的是什么？服务用的是什么语言编写的？；

b. 服务端功能的内部逻辑实现;

c. 服务端与数据库是如何交互的，例如：数据库的表结构是什么样的？服务端功能是怎样操作数据库的？

d. 服务端与客户端之间是如何进行交互的，即接口定义;

通过收集以上信息，测试人员整理出服务器端各模块之间的交互图，客户端与服务端之间的交互图以及服务端内部功能逻辑实现的流程图。

e. 该服务上线后的用户量预估是多少，如果无法评估出用户量，那么可以通过设计测试执行的场景得出这个值;

f. 上线要部署到多少台机器上，每台机器的负载均衡是如何设计的，每台机器的配置什么样的，网络环境是什么样的。

g. 了解测试环境与线上环境的不同，例如网络环境、硬件配置等

h. 制定测试执行的策略，是需要验证需求中的指标能否达到，还是评估系统的最大处理能力。

i. 沟通上线的指标

通过收集以上信息，确定性能测试用例该如何设计，如何设计性能测试用例执行的场景，以及上线指标的评估。

3. 测试设计阶段

根据测试人员通过之前整理的交互图和流程图，设计相应的性能测试用例。性能测试用例主要分为预期目标用户测试，用户并发测试，疲劳强度与大数量测试，网络性能测试，服务器性能测试，具体编写的测试用例要更具实际情况进行裁减。

用例编写的步骤大致分为:

a. 通过脚本模拟单一用户是如何使用这个 web 服务的。这里模拟的可以是用户使用 web 服务的某一个动作或某几个动作，某一个功能或几个功能，也可以是使用 web 服务的整个过程。

b. 根据客户端的实际情况和服务器端的策略，通过将脚本中可变的数据进行参数化，来模拟多个用户的操作。

c. 验证参数化后脚本功能的正确性。

d. 添加检查点

e. 设计脚本执行的策略，如每个功能的执行次数，各个功能的执行顺序等

4. 测试执行阶段

根据客户端的产品行为设计 web 服务的测试执行场景及测试执行的过程，即测试执行期间发生的事儿。通过监控程序收集 web 服务的性能数据和 web 服务所在系统的性能数据。

在测试执行过程中，还要不断的关注以下内容:

a. web 服务的连接速度如何?

b. 每秒的点击数如何?

c. Web 服务能允许多少个用户同时在线?

d. 如果超过了这个数量，会出现什么现象?

e. Web 服务能否处理大量用户对同一个页面的请求?

f. 如果 web 服务崩溃，是否会自动恢复?

g. 系统能否同一时间响应大量用户的请求?

h. 打压机的系统负载状态。

5. 测试分析阶段

将收集到的数据制成图表，查看各指标的性能变化曲线，结合之前确定的上线指标，对各项数据进行分析，已确定是否继续对 web 服务进行测试，结果是否达到了期望值。

6. 测试验证阶段

在开发针对发现的性能问题进行修复后，要再执行性能测试的用例对问题进行验证。这里需要关注的是开发在解决问题的同时可能无意中修改了某些功能，所以在验证性能的同时，也要关注原有功能是否受到了影响。

软件测试理念

◆ 作者：测试小公主

测试理念有多种，有一些理念，深藏于我的心中， 而这些理念，您或许偶尔想到，却没有说出，或许您感受到了，却因为工作生活的忙碌，没有将其背后的含义想具体， 在此我非常愿意和大家进行分享这些理念。

第一篇：测试工程师之间要善于发现闪光点

事实上，测试的过程也是人生的一部分，我们是在一个测试团体中进行测试的，就算只有一个测试工程师，那个测试工程师也可以作为一个测试部门的一份子，他（她）具有一切测试部门所必备的特性。

对于软件，我们可以把它当成一个机器，一个冷冰冰的机器，但是对于其他测试工程师，我们需要认清他（她），是一个人，一个地地道道的人，而不是机器。

我们绝对不能忽视我们之间的相互扶持，测试人员之间的相互扶持和鼓励是及其重要的。

是人就会有情感，他（她）之所以会在测试部门进行测试，那是因为爱测试，这份爱，支持着他（她）对于软件，对于测试部门，对于开发部门的一切活动的关注，那份坚持和执着，是需要每一个人的理解和支持的。

爱的付出需要有回报，测试工程师之间发现彼此的闪光点，是每一个测试工程师的首要职责。

互相讨论

测试人员之间的话应该是最多的，讨论 bug 的看法，讨论需求的理解，讨论测试用例的写法，讨论测试的形式，讨论测试的进度任务安排。我们互相之间会形成正确的认

识，形成测试的互补。

每天的讨论开始于每天早上的早会，早会是将大家准备做的任务陈述一下，时间是十分钟左右。确认任务都分配完整。同时也说明一下昨天测试遇到的困难，看是否有什么方法进行解决。而接下来的讨论，则在于对于在测试过程中的一些讨论。

这种讨论的价值是很大的，除了要注重讨论之外，也要注重时刻鼓励对方，比如说："你这次做的真的很不错!"，"非常感谢，我理解了。"

犯错的定义

犯错是很正常的事情，首先我们要知道每个人都是会犯错的，这种原谅他们所犯的错的心态，是极其珍贵的。如果测试工程师少发现一个 bug，是很正常的；如果测试工程师说错一句话也是很正常的，快速理解和纠正，会带来很多的好处，这种好处就在于，这种快速的理解和纠正，就可以以最快的速度改善软件的质量，同时也不会使得其他测试工程师丧失信心，有助于他（她）日后的测试热情保持高涨。

其次，这有助于对于犯错的特点一清二楚。一个测试工程师容易犯的错误，往往是，其他测试工程师也很容易犯，避免其他测试工程师犯同类错误，对于测试团体的集体质量提升带来很大的益处，大家能一起共同成长。

欣赏有价值的成果

如果一个测试工程师分享他(她)的测试经验，其他测试工程师应该以一种欣赏的态度去面对，而不是运用一种非常嫉妒的方式去面对，那么就能大大提高测试团队的信心。

不仅仅是在其他测试工程师主动分享的时候，值得欣赏，在平时的工作中，互相的交流，测试用例的互相学习，都可以作为欣赏。偶尔的说错一句话，写错一句话，全部忽略不计，只有呈现的最终大家都满意的测试文档，才值得学习和欣赏。

测试工程师之间的关系就如同食物链一般，任何一个人的缺失，事实上是缺失了一个食物链的一环，再招聘进来一个测试工程师，必定是不同的，需要很长一段时间的适应期，才可以重新构建一个完整的食物链。

并不是说新来的一个测试工程师不熟悉这个软件，而是，每个人的测试风格不同，使得其他测试工程师依赖于某个测试工程师的风格，而早已形成了某种互补的形式，一

旦要寻求一个相似的测试工程师，那就是一件比较困难的事情。

而这个适应期不会很长，因为只要大家互相理解和包容，就会很快的成为一体。

当一个完整的食物链构建好之后，通过测试人员的分工合作，就可以将任务完成到极致和最佳状态。

第二篇：测试分工要随机分配

在某些单位，每一个测试工程师负责一个模块，这种做法，其实并不可取，因为这样，这个模块的形式往往会只在某个人的掌握之中，软件的优劣程度，就会产生一个定势，而当这个测试工程师离职之后，再次找一个人替代他（她），就又需要从零开始。这段时间将会是一段非常困难的时期。

当这个测试工程师有事请假的时候，其他的测试工程师不能很好的代替他。

而当这个专属的测试工程师在工作的时候，他（她）仿佛在说，“我是不可替代的”，这种气势会有压倒一切的感觉，也会给研发部门带来负能量。

当这个测试工程师正在测试的时候，他的信息是封闭的，他很难感知到其他测试工程师的气息，他们的动态，他们的思维，而他的想法也无法传达给别人。在封闭的情况之下进行测试，然后还用着“我是不可替代的”气势，仿佛在让别人在惧怕他（她）。

而于此相反的测试组仿佛在说，我们是温暖团结，联系紧密的一组测试团队，高矮胖瘦都会有，我们会给软件带来全面准确的分析和测试。

假如一个单位有 3 个项目需要测试，有 2 个测试工程师，可以采取这个方法，对于每次的新版本，这两个测试工程师轮流负责，项目 1 的 a 版本，给测试 1，项目 1 的 b 版本，给测试 2；项目 2 的 a 版本，给测试 2，项目 2 的 b 版本，给测试 1；项目 3 的 a 版本，给测试 2，项目 3 的 b 版本，给测试 1，以此类推。

测试用例采用每个测试工程师各自写各自的测试用例的方式来进行，对于某些必要测试用例，也就是说每次的版本都执行的测试用例，可以放置在同一的地方，方便大家更新和补充。

测试的过程中，每个测试人员需要首先，互相沟通，然后在没有结果的情况下，再去问开发，这样不但增进了测试人员之间的友情互动，而且还能尽可能少的麻烦开发，而开发对于测试的询问也应该及时响应。

第三篇：软件测试要有真实性和准确性，富于情感

在描写测试用例的时候，尽可能全面而细致，寻找最为恰当的语言来描述测试用例，测试完毕之后，写上测试总结和体会，同样也用最为细腻的语言写出来，让人仿佛能身临其境。同时，也方便进行回顾和分析，这对于日后改善软件的质量是很有帮助的。

软件测试是一个长期的工作，对于同一个软件，今天测试的体会和明天测试的体会都会有不同，如果都能写上，那必定是一笔宝贵的财富，也许一年后，翻看以前之前我们写的测试报告，我们会发现自己的进步，会发现自己对于软件的了解更加的深入了，发现自己的测试范围扩大了，眼界更广了。 对于刚开始测试某个软件的测试工程师，需要一点一滴的积累对于软件的理解和认识。将它们写进测试用例，写进测试报告。平时，可以看一些简.奥斯丁的书，因为这位伟大的作家写的书最大的特点就是描写细腻准确，刻画心里活动细致入微。很适合用作测试用例以及测试报告的写作。

我认为我写的测试用例和测试报告还并不完美，用词还需更恰当，测试技法还需更丰富。而我的追求，就是能在日后写的越来越接近完美，我认为，测试用例和测试报告，应该力求达到，语句通顺自然，带有文学气息，细致入微，感情丰富细腻，让人能阅读了之后能感知到这个软件给人带来的最真实的体验。

这种真实体现在，对于测试用例的执行结果的真实准确，和对于测试分析的真实准确，如果说测试分析的真实准确需要文学的刻画，那么测试测试用例的执行结果的真实准确，是需要拥有耐心和细心的态度，将每个结果标注 pass 或者 fail。不能有随意性，不能有为了完成任务而进行标注的随意性。除了 pass 以及 fail，还可以有其他状态，用于特殊情况的标注，在 pass 和 fail 边上，最好能标注上原因和体会。

除此之外，这种真实性还体现在，测试工程师之间彼此沟通的真实性，及时性；和开发之间沟通的真实性，及时性，有任何信号都可以和开发进行沟通，不必拖拉。

我比较赞成自己执行自己的测试用例，因为对于自己的测试用例，自己是最为熟悉的。而且，由于自己对于软件有过一定的认识，才写出了测试用例，因此自己的执行效率是最高的。

执行别人的测试用例之前我也总是需要花费时间去研究一下软件，事实证明，一个对于软件有更加深入了解的测试工程师，在执行测试用例的时候，体会才是更为深刻

的。而这种花时间研究软件的过程，会被某些同行认为你在浪费时间。如果真的是这样的情况出现，我想应该责怪的是他们而不是你，因为他们不了解你，他们也不了解测试。

当这个测试组的成员还和我完全不熟悉的时候，又或许当我遇到一个随意编写测试用例的同事，会让我在执行测试用例的时候，有一种屈辱感。

但是由于善良的本质，会让我出于善心和责任心，加班再晚，都将测试用例一一执行完毕，并在边上标注上自己的意见和建议。

我知道为此我倾注了许多情感！

但是当我和别的同事之间产生的深厚的友谊，并且大家彼此之间都非常熟悉这个软件的时候，我想我执行别的同事的测试用例，就是另外一种感觉了。温暖，甜蜜，幸福，思维的碰撞，热情的燃烧都会接踵而来。

测试工作会占据我许多的时间和精力，在工作之余如果能再学习一些统计学的知识，一些数学知识，将对于软件测试带来更多益处。

您是否认为用以上这种测试方法，就能感受到测试组的同事们对测试的那份执着，是否也感受到一股正能量呢？

工具无关的自动化测试平台设计

◆作者：郝强

1. 问题的提出

最近几年来，我的自动化测试工具之旅大致是这样的，最早用的是 QTP,然后是 RFT(IBM 的功能测试自动化产品),之后也经历了 Selenium, Watir 等，再后还是一些商业工具主要是偏 web 自动化及移动自动化，如 sahi, appnium, Keynote DeviceAnywhere, SeeTest, HP UFT 等，这一系列的变化，让人痛苦的不是学习的过程，也不是各种编程语言的转换，最痛苦的是我们的自动化测试脚本要因为工具的变化而需要重写，因为无法重用，我们或是维护多种自动化工具脚本，或是将自动化测试脚本为最近使用的工具进行重写编写，有太多的 effort 花在这些事情上。

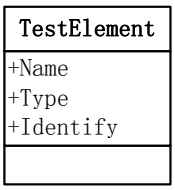
我们怎么解决这类问题呢？试想，如果我们能够有这样一个平台，如果提供统一的自动化编程 API,而且独立于某种工具，那该 有多好。所以本文的目的要设计 这样一个平台，能够对自动化测试人员提供统一的编程接口，能够适应测试工具的变化，而无须修改已经基于此平台编程好的自动化测试脚本。

2. 如何实现工具无关化

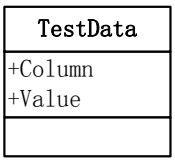
首先，我们要考虑工具无关化，如果要实现工具无关化，那么对于使用者（自动化测试脚本实现者）一定是使用一致的 api，一致的测试元素，一致的数据访问方式。那么我们先要考虑一下测试元素的一致性。

这里我们先假设我们未来的待测试应用都是 web 应用或是 mobile 应用，而 mobile 应用我们使用的都是 hybrid 应用。对于测试元素来讲，最重要的是如何能够识别它，我们在识别元素的时候，都会找到一个唯一 id 或属性来标识它。来看我们上边假设的应用，我们可以使用 xpath 来做为唯一 id 来识别元素。你可能根据自己实际场景来设计来定义它。除此之外，为了能够操作元素，我们需要知道它的名字，因为我们还需要为它命名。除此之外，我们还需要使用一个类型字段来区别不同工具之间可能对测试元素有些特别的要求，我们通常使用 type 字段来标识它。所以我们从逻辑上来看，一个工具无

关的测试元素大致看起来是这样的。

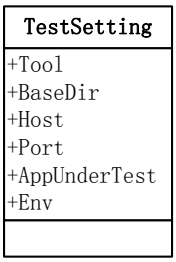


对于测试数据来讲，我认为，每个数据都是有一个列和一个值组成，所有数据看起来比较简单。

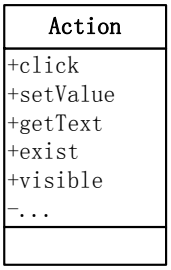


数据有一点需要注意一下，如果我们要实现数据驱动的自动化测试，我们就需要在此平台提供处理多行数据能力。

有了测试对象和测试数据之后，我们需要了解我们的待测试应用，我们的平台需要对待测试应用使用之前进行一些配置，使用时还要进行一些初始化等工作，使用完我们还会对些进行一些清理销毁等工作。如此一来，我们的平台需要考虑如何进行测试设置工作，因为未来我们平台面对可能各种不同的测试工具，那么在这里我们也需要在些考虑好一致的接口。所以此部分看来如下所示:



对于测试元素的操作，我们通常会使用类似 click, setValue 等一些点击，填值的操作，我们同时还会检查一些测试元素是否在页面中存在，也会检测一个测试元素是否展示在屏幕上。我们可以将些部分统一归结为 action. 所以对象的 action 看起来如下所示:



除了 action 之外，我们的平台还要提供 checkpoint 功能，此功能是为了能够让我们脚本有能力判断最近测试结果是通过还是失败。即有一个 checkpoint 失败了，整个测试脚本就是失败状态。Checkpoint 的功能使用起来极其简单，我们需要为其输入两个参数，一个为 expect,一个为 actual,二者进行比较并返回比较结果。这里需要强调的一点是，我们设计 checkpoint 时，要使其能够为其二个参数可以自适应到各种数据类型，因为我们实际应用时，有时会使用两个布尔值进行判断，有时可能会使用两个字符串进行判断，也可能我们可能直接将两个对象传过来进行比较。所以这里我们要让其能够自适应。

讲到这里，我们的平台还需要至少要有的一个功能是 report，我们的 report 要能够展示测试脚本最终是通过了还是失败了。同时能够记录每个步骤的状态，能够截屏。当然，如果 report 能够提供更多的 metric 数据就更好了。方便未来计算 ROI.

讲到这里我们都是讲的都是对外提供的统一的 API，从使用都 角度，这些已经基本够用了。但是对于工具来讲，我们要实现工具无关化，我们要讲起来可能会简单一些，但是实际做的时候还是比较麻烦的。因为我们需要针对我们平台支持的每一套测试工具编写接口，使其在外边看起来是一样的。所以平台这边说起来是简单的，但实现工作量还是比较大的，因为每套工具都有其复杂性，再次封装后并提供统一的简单易用接口并非总是那么容易。所以编写接口的人，必须 是对其封装的工具是极其熟悉的，并且有丰富的实际应用经验，因为这种你们才能为使用者写出他们真正所需的接口。

3. 总结

在上面我大致讲解了要实现一个工具无关的自动化测试平台所应该具有最少元素集合。在实际应用中，我们所做的工作远远大于这个集合。但是有了这个平台之后，我们自动化脚本的重用率有了很大的提升。对于团队中的自动化实现者来说，它们不需要再痛苦了学习和掌握每一种新的工具，或是因为工具的变化而重写自动化测试脚本了。

电子商务网站的高效测试方法

◆译者：于芳

一、概述:

测试对电子商务来说至关重要因为电子商务网站不仅业务关键同时对其用户可视度很高，任何的故障、错误都会立即造成昂贵的收入的损失甚至在更长时期里如果未受影响的用户寻找替代网站就是更昂贵的代价损失。当然电子商务世界存在的时间压力需要完全彻底的测试通常与业务关键性相关，因此需要一种新的方法来使得测试被集成到开发过程中从而确保测试不会是一个严重的时间负担。

对大多数该技术的熟悉意味着经试验检验正确的机制将要么合适要么可以修改为合适的机制。快速程序开发这种机制尤其建议使用一些有信誉保证的方法。电子商务必须找到它自己的方式并建立它自己的方法，尽管这跟大多数新企业一样。在本文中，我们建议使用几个已经经过时间考验的测试法则。

二、简介:

什么是电子商务？为本文做解释之用途来说，电子商务是定义为需要允许业务来单独草偶作或主要使用电子数据流的软件和业务流程。电子商务常常与 WEB 技术关联在一起，常常通过 WEB 入口来交易，但是电子商务比作为消费者界面的 WEB 页面供应的内容多得多。

集成业务流程的创立（企业资源规划），分散软件程序结合的集成，每个都是设计来方便看业务的不同角度（企业程序集成），软件和业务流程从而拥抱与供应商系统的交易（供应链管理），日益增加的在公共网络上进行交易的安全性的需要，电子商务网站上存在的潜在的流量要求都对电子商务的开发社区提供了新的和独有的挑战---这些挑战将需要新颖的和创新的解决办法，他们将需要在其被允许上线前完全的测试。

为什么测试在电子商务环境里很重要？首要和主要的原因在于电子商务，由于其本来的性质—业务关键。1998 年第三个季度，戴尔的电子商务网站每天的销售额超过 100 万美元；E*交易网站目前每天交易量超过 52,000 个，一天的网站故障的成本在 80 万美

元左右；欧洲的旅游业到 2002 年将价值 20 亿美元，根据数据监控的估值。顾客需求的及时满足，和其隐含的在快速送货和有竞争力的价格上的要求，及透明的对网站的访问要求，都组成了创建潜在庞大的对 WEB 网站和入口访问的要求。

第二个原因是电子商务是一个庞大的发展的市场但是需要大量前端投资以让进入网站成功。全球已经有 580 万网站，其中 250 万个是在今年创立的（1999）。国际数据公司（IDC）估计到 2003 年电子商务市场将从 1998 年的 50 多亿美元发展成 1000 亿美元。开发一个电子商务网站的 平均成本是一百万美元，伽特纳集团说，而这成本将在接下来两年里每年增长 25%。

第三个原因是因为电子商务开发的历史已经与昂贵的失败故障连累过，至少有些网站本可以通过在网站开放给普通大众之前经过更好的测试来避免的。（在电子商务的术语中，‘网站’ 意味着从供应商到后端系统和前端系统到客户的整个架构；它通常包括内网，网际之间和外网程序以及遗留系统和第三方插件。）

三、测试挑战

业务问题

一个成功的电子商务程序是如下这样的:

--可使用的。有问题存在的用户界面丢失客户。

--安全的。隐私，访问控制，认证验证，完整性和无拒绝能力是大问题。

--可扩展的。成功的网站会带来不断增加的要求。

--可靠的。失败对与一个业务关键的系统来说是不容思考的。

--可维护的。变化率大对电子商务是基本事项。

--高可用度。宕机的代价太高，不能容忍。

技术问题

电子商务网站的开发流程有独有的特征和某些相关联的风险。WEB 年被公共认知大概是两个月前。换句话说，一个可靠的更新测试可能需要产生每月粗略更新的电子商务网站。基于这个原因，快速程序开发（RAD）技术在电子商务环境里大占优势，而在某些情况下开发甚至直接在生产环境里做而不是在一个单独的开发环境里。快速程序开发技术不是新技术，不过大家认为这些技术在功能对用户可见的情况下作用最好—这样

WEB 网站开发会看起来是一个理想的程序区域。但是不幸的是，电子商务网站的其他方面至少与前端同等重要。端到端的业务流程集成和连续的严重的放在中间流程约束使他们比为快速程序开发所要的理想程序区域更少。

这些变化为测试人员增加了风险，创造了新的挑战，因为时间压力对在网站发布之前要花费更长时间测试发生了作用。与此同时，前端系统的技术环境正在快速第变化，因此变化被强加到电子商务网站上甚至当网站本身没有变化的时候也在被强加变化。这需要比在常规程序里预料更多的回归测试来保证网站在变化传到浏览器，搜索引擎和入口上之后能持续地正常运作。新问题也来到给测试人员，主要是交易的安全性和 WEB 网站在巨大负载条件下的性能。

前端系统

静态测试。一个电子商务网站的前端通常是一个需要在其自身上测试的 WEB 网站。这个网站必须在语法构成上正确，这是一个十分直接明显的问题，但是它也必须提供一个在一个或多个平台上可用水平的服务，并且在选择的平台间具有可移植性。应当在一系列浏览器上测试该网站，以确保在不同浏览器上看到的图片都是同样的质量。可用性是一个关键问题，测试必须使用用户的角度来测试。例如，一个页面上按钮的功能可能单独可接受，但是一个用户能容易地在网站上浏览，从网站上打印的信息是否在纸上显示较好？对网站的安全性获取信心也很重要。很多这样的测试可以通常创建和运行一个典型的用户交互文件来自动完成---对回归测试有用而且对检查基本功能节省时间。

动态测试。 连接在一个电子商务网站上的程序，要么是 CGI 编程要么是服务器扩展程序，将需要通过创建产生调用这些连接的程序的场景来测试，通过数据库搜索来举例。提供给用户的服务必须是在语法构成上探索过的，包括每个服务的转向时间机器总的服务器响应时间。这，也是必须要在不同替换的平台，浏览器和网站链接上练习过。电子商务程序主要是交易为中心的，建立在关键的业务流程上，并将需要在基于内网和基于外网的程序间高效的连接。

后端系统

电子商务系统的后端系统通常包括 ERP 和数据库系统。后端测试，因此是关于业务程序测试，并不会退出任何新的或业务角度上不易理解的问题，但是有潜在的新的技术上的问题，例如服务器负载平衡。幸运的是，客户-服务器系统测试已经教会测试社区很

多有用的东西，他们可以被应用到这种情况中。然而什么是主要的要应用关键的前端测试场景到后端系统里。换句话说，后端系统应当在同样的真实交易业务里驱动和使用将在前端测试里使用的数据。后端也许会证明是用户服务的瓶颈，因此在负载和扩展性下的性能就是要抛出的关键问题。安全性是其本身的一个问题，但是对性能也有潜在影响。

中间件和集成

集成是电子商务的关键。为了建造一个电子商务程序，下面的一个与或多个组件通常被集成起来:

- 数据库服务器
- 服务器-边程序脚本/程序
- 程序服务器
- 为用户界面提供的 HTML 表单
- 客户支付服务器上的程序脚本
- 与旧的后端系统集成的脚本/程序

开发一个电子商务网站的流程与开发一个 WEB 网站的流程大为不同---商务网站添加额外的复杂度。一个十分复杂的特性是集成的特性。

如果一个程序使用数据库服务器，WEB 服务器和不同供应商的支付服务器创建起来的话，连接这些组件要大量的工作，理解连接有关的问题并集成他们到一个单独的开发（可执行的）环境中都需要大量的工作。如果包含进来旧的代码，这就在该问题上添加了一个新的维度，因为需要投资时间来理解旧代码的接口和任何变化的可能的影响。

记住陡峭的与前沿技术相关联的学习曲线也很重要。与最新版本的开发工具和要集成的产品保持一致，他们与前面的版本的兼容性，调查所有的为性能创建优化的解决办法的新功能可能会是一项令人头疼的任务。WEB 上的电子商务程序也是一个相对新的现象，不可能有任何在类似的项目上的度量来帮助项目规划和开发。

安装和升级程序的维护任务也会变得涉及良多，自从他们需要专业知识在下面方面:

- 数据库管理。

- WEB 服务器管理。
- 支付服务器管理。
- 管理其他已经集成到网站的特别的工具。
- 技术支持也应当被记住。

正确地运作后端和前端系统不会对可靠的整体的功能或性能提供保证。端对端的完整的集成架构的测试，使用现实的交易是一个关键的主要的组件。

四、十大重要的高效电子商务测试法则

几十年来，信息技术在业务生涯里变成一个主要的因素，问题和挑战类似这样的现在被电子商务社区的已经被遇到并解决。关键的测试法则已经浮现，这些可以成功地应用到电子商务情形中。

法则 1、测试时一个风险管理流程。我们学到的软件测试最重要的教训是测试是管理不成功 IT 程序的风险的最好的机制之一。有效的测试采用了一种策略被适配到正在被测试的程序或服务类型上，适配到程序或者服务的业务价值，以及适配伴随故障一起的风险。测试规划的详细事宜和测试设计然后可以一致到一个业务为中心的添加真实业务值和提供某些对每个开发流程阶段出现的风险的 客观的评估的策略中。

计划应当包括风险衡量和价值以及集成测试和其他确保开发恰当地关注在开发出最大价值最小风险的质量相关活动。真实的项目可能不会达成计划的所有东西，但是度量至少会使我们能决定发布一个程序到现实中使用是否明智。

法则 2、知道在测程序的价值。为高效地管理风险，我们必须知道成功的业务价值以及失败的代价。业务社区必须涉入进来来设立让风险评估可以基于的值并致力于发布一个一致同意的质量水平。

法则 3、设立明确的成功完成的测试目标和标准（包括测试覆盖率衡量）。当测试一个电子商务网站时，对测试来说去简化成网上冲浪来说很容易，由于测试相关网站或另一个完全不相关的网站的容易性。这就是为什么测试程序必须要恰当规划进的原因，要和提供精确指令和期待结果的测试脚本一起。也会需要一些跨参照物回到修和目标，这样可以做一些有关多少需求在给定的时间内被测试了的评估。成功完成的标准建立在发布足够的业务值，测试足够的需求以对网站的最重要行为有自信，和最小化重大故障的

风险。这些标准---应当与业务社区达成一致---给我们需要的在决定是否准备好让网站被客户访问的决策给予关键证据。

法则 4、创建一个高效的测试环境。为电子商务网站创建一个完全代表性的 测试环境可能很昂贵，基于各种不同的平爱和因特网的使用作为一个通信媒介的情况。跨平台测试时，自然地，测试任何多平台软件程序的重要部分。在电子商务这种情形下，术语“跨平台”必须也能扩展到包括“跨浏览器”。为了确保一个网站恰当地加载和运行在所有的支持的平台上，应当尽可能地做压力和负载测试。作为一个绝对最小值，几个人应当能够同时登录和访问该网站，从支持的浏览器和平台混合中测试。压力和负载测试的目标，然而是将网站置在代表性的使用水平下。因此它会，对使用自动化工具比如 Segue 的 Silkperformer 或者 Mercury 交互的 LR 做性能或负载测试有益。

法则 5、尽可能早地在开发阶段进行测试。在软件工程社区里已经有这样一个共识，即缺陷越早发现，修正缺陷的成本就越低。在电子商务网站的案例中，货物运送后发现的缺陷将会被检测为由市场发现的该站点的故障，这个市场潜在地会大到同因特网的用户数量那么多。这增加了利润的损失，而且可能是顾客忠诚度的丢失，同时还有修复该缺陷的直接成本。电子商务开发快速，而且经常建立在不断变化的需求上的事实使得早测很困难，但是测试策略已经由 RAD 社区开发出来，而这些策略可以移动性地做支持。

也许 RAD 中最重要的思想是集合的开发团队，允许用户来跟开发者做交互，持续不断地从开发流程的开端验证产品的行为。RAD 使用由一系列严格控制的“时间盒”的产品原型，这种“时间盒”是让原型在固定的时间段内进行开发和测试的以保证产品开发没有偏移其最初的目的。这种 web 开发的方式使得测试成为开发流程的集成部分，从而加强开发周期里的风险管理。

法则 6、用户验收测试。客户或电子商务网站的最终拥有者应当在开发流程最后在需要供应商参与的地方进行域测试和验收测试。即使 RAD 是在使用其持续性用户测试方法，电子商务网站的一些属性也不会容易用这种方式来测试（或即使可能，也是在某些情况下）。有些能够发现如性能和安全问题的最终测试的形式需要被包括作为一个最终确认，因该网站可以在典型的用户交互中运行良好。RAD 用不到的地方，供应商的内部测试涵盖范围和用户验收测试涵盖范围应当在项目开发周期早期定义（在测试计划中）并在项目接近完成的时候重新访问来确保持续的目标和责任的一致性。但是用户验

收测试不应当被看作是一个在正式发布之前委托给场中用户的 beta 测试活动。电子商务用户正变得越来越不能容忍质量差的网站，而有关功能，性能和可靠性的技术问题已被认为是顾客抛弃网站的主要原因。早些将用户暴露到有问题的网站会增加他们发现该网站不可接受的可能性，即使开发人员继续在 beta 测试过程中改善该网站的性能。

法则 7、回归测试。变化了的应用程序需要做回归测试来确认该变化没有带来不想要的影响，因此这将会是任何一个电子商务测试策略的主要特征。引用外部链接的基于 web 的应用程序需要定期规律的回归测试，即使他们的功能没有改变，因为该环境是在持续不断地变化着的。如果有可能的话，回归测试应当做自动化在最小化该变化对测试时间表的影响。

法则 8、尽可能多地做自动化。这是一个有风险的法则因为测试自动化充满困难。据说一个傻瓜用工具还是一个傻瓜，而自动化一个不稳定的流程的后果是更快的嘈杂，这两者都是对的。然而，在紧凑的时间标尺内将电子商务网站的测试工作进行的充分足够又不用自动化的可能性十分微弱。解决方法是足够认真地对待测试流程，你可以将他们形成文档然后控制他们如此以来自动化成为一个灵活的可选项—然后你选择，购买和安全工具。这不会很快或很便宜---但是它可能只是避免了一个非常昂贵的故障。

法则 9、捕捉测试事件并使用他们来管理在发行时的风险。一个测试事件是任何期望值和测试的实际值之间的偏差。只有某些测试事件会与实际的错误有关，有些会是由错误的测试脚本，误解或者系统功能的细微变化引起的。所有的发现的事件必须通过一个事件管理系统记录，然后能被用来确认什么错误时系统中比较突出的，发行的风险又可能是什么。突出的事件可能是我们应用的一个完成标准，因此跟踪和评估事件的重要性的能力对测试管理是至关重要的。

法则 10、恰当地管理变化以避免毁坏所有的测试工作。电子商务开发中事情变化很快又频繁，变更的管理可能会成为一个瓶颈，但是测试软件程序的一个版本然后发布一个不同的版本没有太多意义；不仅是测试努力白浪费了，而且风险也没有减少。配置管理工具，如 PVCS 和 ClearCase,可以帮助最小化变更管理的总体费用，但是该法则是最重要的事。

五、总结

电子商务既是熟悉又是新颖的事物。一些技术相对的是新鲜的，将该技术应用到一

个完全的商业中当然是新颖的，但是在一个全新的环境中运行商业的业务流程的问题让所有的新颖在一些熟悉和难解决的问题下都相形见绌。有些矛盾的是，最严重的问题是在该技术更熟悉的领域里发生的，因为电子商务的出现给这个相对老、原本设计来做不同的用途的技术赋予新的和有挑战的需求。

测试对电子商务来说至关重要因为电子商务网站是业务重要并且对其用户高度可见的，任何故障都可能会立即导致收入的损失甚至长期的更昂贵的损失如果未受影响的用户选择其他网站的话。但是电子商务世界的时间压力不能够充分测试通常与业务重要性有关，因此需要使用一个新的方法来使得测试能够集成到开发流程中来保证测试不会呈现重要的时间负担。

对大部分该技术的熟悉意味着经尝试并检验正确的机制会要么合适要么可以修改以适合需要。快速应用程序开发 RAD，尤其建议使用一些有保证的方法。尽管像大多数新企业，电子商务必须找到它自己的方式并建立自己的方法。在本文中，我们已经建议使用一些测试法则，他们已经经受住时间的考验并且和一些从类似有挑战的开发环境学到的经验融为一体来给电子商务测试人员在他们的探索旅途上一个好的起点。