目 录

上了"保险"后 Windows XP 到底可靠吗？

◆ 作者：郭盈、周润松

近日，根据 Stat Counter 的统计数据显示，Windows 10 在国内的市场份额迄今不过 5.22％，落后于 Windows 7 和 Windows XP，Windows XP 甚至一度反弹上涨。Windows XP 退市 1 年多，仍然占据着 windows 领域不容小觑的比例。据调查，有的购买了预装正版 Windows 7 的顾客在用了一段时间之后，还是主动回来要求装回 Windows XP。在 XP 已经停止补丁更新很长时间，针对其加固的产品也是发展的蒸蒸日上，作为用户，特别是大型企业用户如何甄选真正可靠、安全的 XP 加固产品呢？中国软件评测中心总结了如下几种招数来判断。

漏洞防护测试方法

1、有效理论分析在漏洞利用上，分为两块。一块是漏洞本身，另外一块是针对这个漏洞的利用方法。漏洞本身数量很多，不重合或收敛。漏洞的利用方法数量有限，收敛度较高。很经常的出现多个漏洞使用同样的利用技巧的情况。所以理论上是没法穷举漏洞，但可以穷举所有漏洞利用方法的。漏洞本身和漏洞利用方法不同的组合可以产生出非常多的漏洞利用样本，为测评和防护提供了理论依据。

选取 24 个高危 Windows XP 漏洞的抽样分类分析，分析漏洞的类型和利用方式。

MS-ID	CVE-ID	漏洞类型	利用方式
MS13-008	CVE-2012-4792	释放后使用	HeapSpray/ROP
MS13-037	CVE-2013-1311	释放后使用	HeapSpray/ROP
MS13-037	CVE-2013-2551	整数溢出	HeapSpray/ROP
MS13-	CVE-2013-	释放后使用	HeapSpray/ROP

038	1347		
MS13-059	CVE-2013-3184	释放后使用	HeapSpray/ROP
MS13-069	CVE-2013-3205	释放后使用	HeapSpray/ROP
MS13-071	CVE-2013-0810	远程 DLL 加载	远程 DLL 加载
MS13-080	CVE-2013-3893	释放后使用	HeapSpray/ROP
MS13-080	CVE-2013-3897	释放后使用	HeapSpray/ROP
MS12-063	CVE-2012-4969	释放后使用	HeapSpray/ROP
MS12-077	CVE-2012-4787	释放后使用	JIT Spray
MS12-063	CVE-2012-4969	释放后使用	HeapSpray/ROP
MS12-037	CVE-2012-1876	堆溢出	heapspray/ROP
MS12-037	CVE-2012-1875	释放后使用	HeapSpray/ROP
MS12-037	CVE-2012-1876	释放后使用	HeapSpray/ROP
MS12-027	CVE-2012-0158	栈溢出	直接跳转/ROP/SEH 覆盖
MS11-081	CVE-2011-1999	释放后使用	HeapSpray/ROP
MS11-081	CVE-2011-1996	释放后使用	HeapSpray/ROP
MS11-050	CVE-2011-1256	释放后使用	HeapSpray/ROP
MS11-006	CVE-2010-3970	栈溢出	直接跳转/ROP/SEH 覆盖
MS10-090	CVE-2010-3962	释放后使用	HeapSpray
MS10-087	CVE-2010-3333	栈溢出	直接跳转/ROP/SEH 覆盖
MS10-081	CVE-2010-2746	堆溢出	HeapSpray
MS10-056	CVE-2010-1900	栈溢出	直接跳转/ROP/SEH 覆盖

影响 Windows XP 漏洞可以分为下面几种类型：

（1）释放后使用（Use-After-Free）：由于某些对象在释放之后仍然被使用，导致内存可以被复用并执行代码；

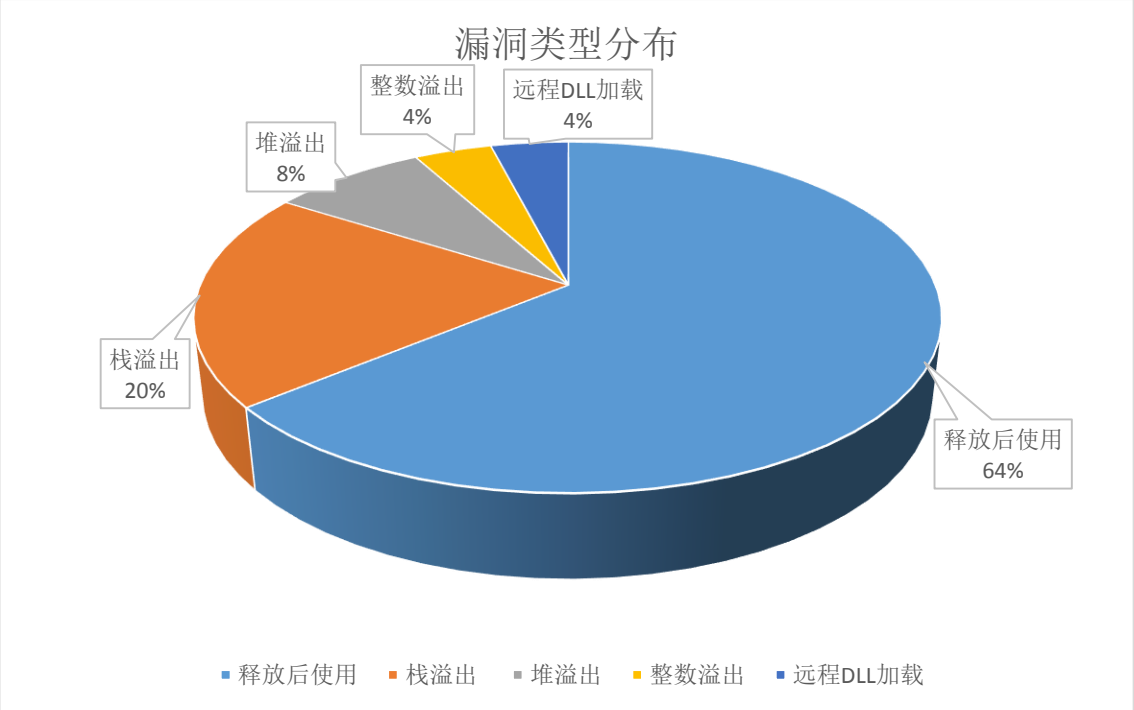
（2）整数溢出（Integer Overflow）：由于某些未知的整数溢出导致可以篡改内存，导致代码执行；

（3）栈内存溢出（Stack Overflow）：由于没有检查栈上的变量长度就直接赋值，导致覆盖了敏感内存位置导致代码执行；

（4）堆内存溢出（Heap Overflow）：由于没有检查堆上的变量长度就直接赋值，导致覆盖了敏感内存位置导致代码执行；

（5）远程 DLL 加载：应用程序可能直接加载远程的 DLL 并执行其中的代码。

各漏洞的分布情况如下图：



从另一个维度来看，成功的漏洞利用方式也是有限的：

（1）直接跳转执行：直接控制代码的执行流程跳转到恶意代码；

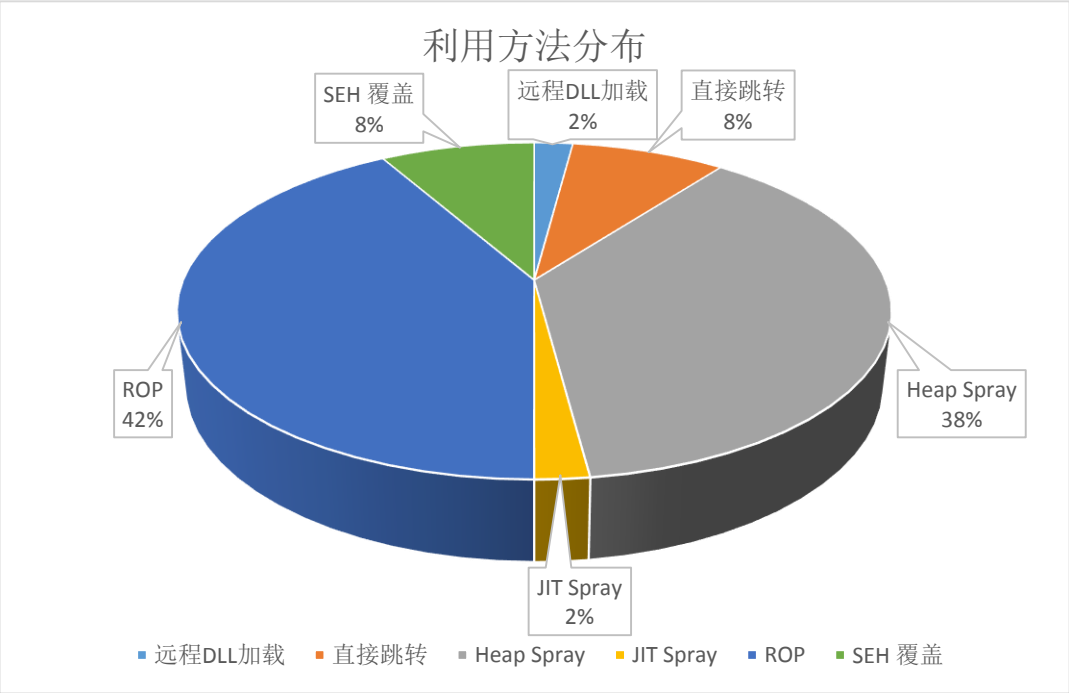
（2）堆喷射（Heap Spray）和其他类型的喷射（如 JIT Spray）：通过在堆上大量分配对象占据内存，覆盖刚刚释放的区域使代码得到执行；

（3）面向返回编程（Return Oriented Programming，ROP）：通过寻找已有的模块当中的可执行的内存块，串联成一个调用链并使代码得到执行；

（4）远程 DLL 加载：利用 Windows 的某些机制漏洞，直接加载远程的 DLL 并执行；

（5）SEH 覆盖：利用 Windows 的 SEH 机制，跳转执行流程。

在取样范围之内的 24 个漏洞的分布情况如下：



由于关注在漏洞分类和漏洞利用方式的防范上，对于未知的漏洞，只要漏洞类型或漏洞的利用方式仍然落在这样一个已知范围之内，XP 加固仍然能够防范。因此上述的漏洞验证方法是有效的。

2、测试方法

为了验证安全加固方案，我们采用使用已有的漏洞验证的方法。制定了如下的 Windows XP 安全加固方案的测试方法：

测试环境

Windows XP 安装后，安装 SP3 补丁包。同时将 IE 升级为 IE8，并安装微软办公软件 Office2003，并安装 Office2003 SP3 补丁包。将此作为 XP 安全加固软件产品的原始基准环境使用。

Windows XP 安装后，安装 SP3 补丁包，同时选择性的安装后续补丁包，但不打全。同时将 IE 升级为 IE8，并安装微软办公软件 Office2003，并安装 Office2003 SP3 补丁包。将此作为 XP 安全加固软件产品的补丁基准环境使用。

测试样本

选取 2008 年 4 月 16 日，即 SP3 补丁包发布之后，在上述测试环境当中可以生效的漏洞利用样本。同时，需要预先整理样本的攻击目的和攻击行为并进行分类，如试图进行样本窃取的样本、试图下载程序并执行的样本等。

测试实施

在没有安装 XP 安全加固软件产品的上述两种基准测试环境下，使用测试样本进行攻击，攻击能够成功；同时，用成功生效的测试样本在安装 XP 安全加固软件产品的上述两种基准环境下进行攻击，判别攻击是否可以成功。如成功，判别 XP 安全加固软件产品漏洞防护失败，如不成功，判别 XP 安全加固软件产品漏洞方式成功。

攻击成功与否的判别依据如下，其中以下几种情况算作攻击失败:

- 测试样本当中的恶意代码没有执行;
- 测试样本当中的恶意代码虽然执行了，但其试图做的，有意义的恶意行为没有成功（例如启动进程、联网下载、窃取文件等）。

效率测试方法

1、时间特性测试方法

（1）XP 安全加固软件安装前后的系统启动时间对比

测试环境同 3.1 节的漏洞防护测试环境，通过秒表的方式进行统计系统启动的时间，从系统开机到系统全部启动，鼠标等待图标消失为止。

（2）XP 安全加固软件安装前后的 IE 浏览器浏览网页的时间对比

测试环境同 3.1 节的漏洞防护测试环境，通过 www.numion.com/Stopwatch/的记时功能对安装 XP 安全加固软件前后的系统 IE 浏览器访问主流网站的展现时间进行统计。

（3）XP 安全加固软件安装前后办公软件对不同大小文件打开时间对比

测试环境同 3.1 节的漏洞防护测试环境，通过秒表的方式进行统计 XP 安全加固软

件安装前后办公软件对不同大小文件打开时间，文件大小设置为 1M、10M 和 100M。

2、资源利用率测试方法

（1）XP 安全加固软件的 CPU 利用率

通过资源管理器的进程模块查看 XP 安全加固软件在安静模式下的 CPU 利用率、处理漏洞攻击防御时的 CPU 利用率。

（2）XP 安全软件占用的内存大小

通过资源管理器的进程模块查看 XP 安全加固软件在安静模式下所有 XP 安全加固软件进程所占用的内存大小、处理漏洞攻击防御时的所有软件进程所占用的内存大小和防御成功后所有软件进程所占用的内存大小。

（3）XP 安全加固软件安装后占用的磁盘空间

在资源管理器中找到 XP 安全加固软件的安装目录，统计目录所有文件在硬盘上所占用的磁盘空间大小。

兼容性测试方法

1、Internet Explorer 浏览器兼容测试方法

浏览器，用于显示网页服务器或档案系统内的文件，并让用户与此些文件互动的一种软件。除了浏览器网页外，同时还提供收藏夹、打印等功能。对浏览器你的兼容性测试主要围绕其核心功能进行展开。

（1） 访问网页的兼容性测试

测试方法：安装 XP 安全加固软件后，打开浏览器，访问受大众用户欢迎的网站，以及行业性的重要网站，政企常用的基于浏览器的工作系统的网站，网站访问和使用应该正常，即网页可正常打开显示，并且加载项可正常显示。

以下列出参考的网站类型，在测试过程中可依据实际情况进行筛选。

受大众用户欢迎的网站：

- 导航类：hao123 导航、2345 导航、百度网址大全等
- 新闻类：新浪、网易、腾讯
- 搜索类：百度、谷歌

- 社交类：微博、人人
- 电子商务类：淘宝、京东、天猫
- 视频类：爱奇艺、优酷网、土豆网、搜狐视频、乐视

行业性网站：

- 政府网站，国家部委、公共服务门户
- 企业网站，重点 IT 企业

基于浏览器的办公系统：

通达 OA、协众 OA、互动 OA、易捷 OA 等

（2） 使用兼容性

测试方法：安装 XP 安全加固软件后，IE 浏览器的一些常用功能是否正常。其中主要包括：

文件下载：从任意网站下载一个文件，文件应该能够正常下载；

文字输入：在地址栏、搜索栏输入文字，应该能够正常切换输入法、正常输入中文文字等

浏览器收藏：可以方便正确的收藏网页、访问收藏夹，从收藏夹中打开已收藏的网站。

2、办公软件兼容测试方法

办公软件通常用于编辑文档、制作报表、制作幻灯片的作用。对于一些行业来说，还有一些基于办公软件的编辑平台和经过二次开发的行业软件。而 XP 停止服务后，Office2003 也随之停止了服务支持。办公软件兼容测试主要围绕 XP 安全软件对 Office2003 的兼容问题。具体测试方法如下：

（1） 测试环境

Windows XP 安装后，安装微软办公软件 Office2003，同时安装 Office2003 SP3 补丁包和文档转换模块（用于打开 OOXML 格式的文档）。将此作为 XP 安全加固软件产品的办公软件的原始基准兼容测试环境使用。

Windows XP 安装后，安装 SP3 补丁包，同时选择性的安装后续补丁包，但不打

全。安装微软办公软件 Office2003，并安装 Office2003 SP3 补丁包和文档转换模块（用于打开 OOXML 格式的文档行）。将此作为 XP 安全加固软件产品办公软件的原始基准兼容测试环境的补丁兼容基准环境使用。

（2） 测试实施

Office2003 SP3 在上述两套基准环境下，打开 WORD 类型的文档 doc 和 docx 文档，打开 EXECL 类型的 xls 和 xlsx 文档，打开 PPT 类型的 ppt 和 pptx 文档。重点关注文档转换模块的功能是否可以正常运行，对比两种基准环境下功能正常运行情况。

Office2003 SP3 在上述两套基准环境下，打开 WORD、EXECL 和 PPT 文档，并插入对象（例如文字插入、图片插入等），对比两种基准环境下对象操作正常运行情况。

Office2003 SP3 在上述两套基准环境下，打开 WORD、EXECL 和 PPT 文档，并进行公式编辑、宏运行和自定义动画设置，对比两种基准环境下公式、宏和动画功能操作功能正常运行情况。

3、常用软件兼容测试方法

由于 XP 安全加固软件产品的特性，很可能会影响系统常用软件的兼容性。受影响的软件从技术上说通常是和系统结合比较紧密，或规模较大的软件，其中一些软件不乏广泛的使用。由于安全软件的排他性，安全软件之间产生的兼容性问题不在考虑之列。常用软件兼容性上重点考虑系统自带的 Media Player、PDF 阅读器 Adobe Reader 软件、第三方输入法、各种大型游戏。具体测试方法如下:

系统自带的 Media Player 软件：应当可以正常打开并播放包括 mp3、avi、wmv 格式的音频、视频文件;

PDF 阅读器 Adobe Reader 软件：应当可以正常打开从网络上下载的 PDF 文件;

第三方输入法软件：应该可以在各种软件当中正常输入;

各种游戏，尤其是英雄联盟、魔兽世界等大型网络游戏，应该可以正常登录，正常进入游戏。（打开客户端）

向自动化迈进：风险，意料不到的困难和承诺

◆译者：于 芳

敏捷宣言或其 12 条准则中没有提到任何要团队必须平衡测试自动化的工作以达到成功的说法。但是，如果你曾经做过任何形式的敏捷或适配性开发工作的话，你肯定知道他有多重要。敏捷促进了反馈循环更快进行。测试自动化，当有技能地去做，恰恰提供了这一点。重复地跑自动化测试用例给你的程序的质量提供了心脏的跳动，告知每个人当你从代码库向程序中添加或移除功能点时你程序的稳定性。对自动化的极乐世界是有一条路通往那里的。它将你导向你的团队去建设有价值和可靠的自动化。不幸的是，它是设置在双方中，其中的问题可能会将勇敢的努力变成一堆沮丧和白费力气。学习怎样避免将会在你实现测试自动化过程中挡路的几个障碍的发生和出现。

视野的缺乏

毫无疑问，测试自动化最有危险的隐患是缺乏视野。一个连贯的视野产生一幅详细的地图---某种脆弱的你的团队可以开始剖析和理解的东西。没有那幅共享的理解，团队们经常在一个项目的生命周期里表现出漫无目的的情形。尽管他们可能坐得很近，但是一个团队可能跟下一个团队的策略不同。其他团队可能在做重复工作。另外一个可能完全在往错误的方向发展。更糟糕的是，团队们可能在建造一些最终会被忽略或放弃的不可维护的东西。洞察力应当是简单的并且清晰地阐述它的目标。主要的目标是去创建一个回归套给予你的团队信心当他们在重构程序的时候能够发现缺陷。

重构旧的代码没有一个自动化安全网会是很危险的。它可能会有不可预见的质量后果，尤其是当处理那种过于复杂的不愿用手“摸”的代码时。开发团队和他们的产品拥有者需要有平静的大脑让他们无所顾忌地不用负面影响顾客体验地重构程序。另外一个重要的目标是你的自动化测试套应当将你的测试人员从重复手工执行回归用例中解放出来。这种解放会允许你的团队将精力集中在更有深度的探索性测试中。如果你曾经多次手动跑过回归测试，你就知道这最终是一个重复性的会麻木你的知觉和大脑的活动。

自由的测试人员是开心的测试人员，而开心的测试者会发现更好的缺陷。一个稳固的自动化策略应当将你对单元，功能（或服务）和 UI 测试的预期细节化。因为每一种类型都有其自己的成本和投资回报，对你团队应当对每个类型贡献多少时间具体化。马克.科恩的自动化三角形象地呈现了每个测试类型的合理比率。不要低估一个有粘聚性的可实现的策略的价值。

它会成为你在专业，奉献和团队合作之旅中的技术指导。

整合聚集

我曾见过三家机构的团队测试自动化配置。第一和最有毁坏性的是孤狼方式。创建和维护测试的责任降落在一个单一的个人，通常是团队里最有技术能力的 QA 工程师。这个人跟其他每个人一样对手动测试有额外的责任。孤狼方式中使用的自动化工具很可能是由管理层购买的。第三方工具产生录制-回放自动化用例，且能快速地创建，但是那些测试通常本质脆弱。他们无法摆脱地将用户界面元素绑定到测试逻辑上，因此保证任何在用户界面上的小的变动意味着级联的测试失败。对这种等级的测试破坏意味着很多测试将不得不重新录制，而没有足够的时间来解决这些问题因为有其他的测试责任。所以，用户界面回放测试停留在一个失败的状态。

因为对结果缺乏信心，回归测试套最终将会被暂停使用。该机构将会把用户界面测试自动化与失败关联在一起。在不同方面让团队信服将会是一场费劲的战争。下一个方式是有一个小的，分离的自动化测试团队与其他团队分开的方式。简单的团队不会将他们最好的做法与组织剩下的人员社交分享。这些团队也会发现随着他们的测试数增加，维护工作变得不可持续。不是去开发新的自动化用例，他们去修复测试问题并且给框架打补丁。他们没有这种网络带宽来创新或者将他们的知识和专业与其他团队共享。甚至更危险的是，他们的分离可能导致在所有测试自动化工作上的一种域拥有感和微妙的不愿分享的情绪。给测试自动化支持的最有效率的方式是让每个人都负起责任来。

在敏捷中，有一种成熟度标准来定义工作是否被认为是完成的。整个团队同意而且为这个完成情况负责。一个例子是，团队需要手工测试他们开发的功能。测试人员给精心安排所有的测试成员测试活动，包括开发人员。将测试自动化编织近你的完成度标准中是一种让所有人对产品质量负责的好方式。非技术测试人员可以帮忙编写前段测试用例，而开发人员或者测试“自动化人员”编写实际的自动化代码。目标是让整个团队感受到所有权。每个人分享开发，维护和修复测试的责任增加了测试的效率，移除了职员

的平静。它鼓励合作和共享最好的做事方法。对自动化策略有一个统一的洞察和普遍存在的理解是使用这种团队为基础的方式的前提。

银弹

测试自动化对非技术的管理经理们是一个黑盒子。他们对它的理解是有限的，而他们对调查解决方法的时间几乎是不存在的。市场上有产品生成可以提供最终测试自动化解决方案—银弹。它的设想是非技术测试人员可以从理解使用该产品开始，然后快速不费力气地建造一个健壮的自动化套。管理者们可能有一种感觉是因为没有必要的特殊的培训，软件的成本是合理化的。营销这类产品的网站生成要移除测试自动化的复杂性换之以一种直观的用户界面。别弄错，测试自动化是复杂的。当心提防那些声称可以戏剧化简化创建自动化测试的产品。

随着专利测试数目的增加，团队们变得跟自动化工具更加紧密地连接起来。如果你已经创建了成千上百个测试用例却发现这个工具不能实现他承诺的功能怎么办？对对成本有要求的机构，围绕是撕还是替换的变动的讨论会很困难因为已经投入的沉没成本。关心设想的经理们不可能去承认失败，而团队被留在一遍无所选择只能继续讲他们自己推往墙角。这需要有勇气的领导力来承认在工具选择流程上的错误。从一个技术角度来看，选择一个第三方测试工具需要完全采用他们的架构。你想要一个增强吗？提交一个功能请求，然后跟用户库的其他人排在一条线上。同时，你在非法访问计算机测试系统来让他们工作。增加的技术债务将会导致广泛的测试脆弱性。当脆弱测试越来越多时，维护他们的成本变得不可持续。如果你决定转用另一个测试工具，没有一个良好的退出策略。这不会像将你的测试用例从一个工具中到处然后导入到另一个中那么简单。

数字游戏

那些对测试自动化直观价值不熟悉的人尝试从一些度量中收集获取这种信息。他们将注意力放在数个编写好的测试用例且注意自动化测试发现的缺陷数。而那些度量可能很有意思，他们只阐述了故事的一小部分。没用自动化的测试团队每次回归测试用例手工运行时都会支付一轮循环成本。随着代码库的增大，回归测试集也会随之增加。结果，那个循环成本继续增加。一个有意义的度量是“我们使用自动化工具每周节约了多少人-时？”。随着团队可以执行更多探索性测试用例，他们可能发现更多测试缺陷。有多少缺陷是在使用探索性测试发现而使用手工运行回归测试用例不可能发现的？这是个有趣的问题，但是不可能有答案。更困难的是分配一美元可以展示投资回报的价值。自

动化带来价值，但是据我的经验很难衡量。

一个有着可靠的安全网的团队很大胆。产品所有者可能不止会冒险还会允许团队去对重构的代码做更多冒险的活动。这种重构导致的结果可以增加产品质量，甚至它还能改善该程序的用户体验，然后可能增加登录和转变。更多的用户就会带来更多收入。而用这种假设的情形来决定投资回报率是不可能的。它强调了一个观点---不是所有有价值的东西都容易去衡量。关注于错误的度量的领导者会激励错误的事情。如果一个团队将关键表现指数作为总测试数的话，队员将会更可能会写更多测试用例。总体测试数只代表自从引进测试自动化所走的距离；它并不告诉你离目的地有多近。而高测试总数并不等同于更好的覆盖率；它只意味着团队已经找到一种方式去游戏/测试系统。有意义的测试自动化度量的一个例子是手工回归测试集被自动化的百分比。那给团队成员一个对进度的理解。它也回答了那个还需要多少工夫才能完全自动化手工回归测试的问题。

不能快速传送价值

没有一个合理合适的架构的框架的自动化是毫无价值的。框架这个词包括所有支持性的帮助你的测试有价值的代码。开发一个合适的框架构架需要时间。它是一个与你团队的需求不断演变的一个适应性的过程。总是有要加强的代码，有要更新的组件和要执行的重构工作要去做。由于软件开发的节奏的原因，编写测试自动化的团队迫切需要尽可能快地传送价值。如果你的公司有冲刺评审，展示你所取得的进展。不要等到你的测试框架成熟到开始编写测试用例；你可以使用现有的框架骨架来跑测试用例。

从价值的角度，你能做的最差的事是花费数轴来开发一个没有任何测试进行的框架。一个没有测试的框架-不管架构多么优雅，提供的价值微乎其微。结果，投资商可能开始怀疑他们的投资。从一开始，你的焦点应当放在自动化手工冒烟测试上。使用如 Jenkins 这样的工具，将这些测试用例的运行绑到一块部署代码上然后部署到你的测试环境上。你的下一个目标应当是自动化整个手工回归测试集，从高价值的组件开始。类似于注册，转变，登录，下单和支付的功能是高价值目标，需要被透彻测试。采用一种自动化策略后，至少在首次冲刺末尾承诺发送一小部分测试结果。通过展示你的测试结果在团队附近某处的信息发射器（如挂在墙上的电视）最大化透明度。然后定期间隔地，反思你的进展然后做过程改正来增加测试程序集的价值。

遗漏价值提议

任何团队的自动化努力目标都是去创建有价值的测试用例。有价值的测试用例是稳定的，快速的，恰当关注的和易于维护的。这些测试类型是由已经长期自动化测试的团队编写的。随着时间的推移，队员们对什么该自动化，什么不该自动化有了理解。有经验的测试团队视自动化为一把扳手，而不是锤子。由于增加的维护成本和扩展的整体执行时间他们很小心地添加测试用例到程序中。当评价新功能是否能自动化，成功的团队问自己，“这个果汁值得挤榨吗？”这个测试被自动化更好还是留作手工测试更好？这个自动化太复杂、在我们的一个冲刺里给定的有限的时间里耗费时间编写吗？正在开发的功能做自动化已经饱和了吗？自动化的执行依赖与某些我们无法控制的外部变动的代码块吗？

如果对这些问题的任一个回答是是的话，找其他的去自动化吧。成功的团队会快速删掉不提供价值的测试。他们甚至更快速递删除那些没有明显原因就随机失败的测试。那些测试需要高维护成本。信息展示台的观察者可能视这些停留在失败状态的测试为不成比例的时间量。最终你的机构会成长为不再相信你的自动化测试，然后对其提供的任何价值打折扣。

测试自动化:

一个成功的测试自动化，就像软件开发，并不简单。有很多重要的成本涉及到。有很多潜在的错误的转变和错误的开始。要知道每个你编写的有价值的自动化测试用例，你的团队变得更加解放。更重要的是，不要忘记授权成员做事。允许他们围绕一个视野，提供给他们一个明晰的策略让他们自我组织。给他们自由去探索新的技术，然后决定他们想要拥抱的自动化工具。成为决定过程的一部分意味着他们会更多投资在他们自己的成功上。你的团队可能刚刚发现了一个如果隐藏不被发现就会导致上百万损失的缺陷。这种投资回报率怎样？

敏捷测试模式之 Scrum 及其实践

◆ 作者：王 东

摘要： 本文首先简单介绍了敏捷开发模式和 Scrum，之后详细介绍了在测试团队的具体实践以及作者的经验总结。

一、敏捷开发模式简介

敏捷是近年来软件研发领域很火的一个词，采用敏捷开发模式的研发团队是越来越多了，尤其是敏捷模式中的 Scrum 更是佼佼者大行其道，这表明敏捷模式确有其好处，能给企业带来效率的提升和成本的降低。

让我们看看大名鼎鼎的敏捷宣言，如下图所示:

大家对这段敏捷宣言都有自己的理解，我理解的其核心观点就是敏捷：能够快速，灵活的对变化做出响应，能够去掉繁文缛节，做到身轻如燕，专注于目标并在短期内产出成果物。

其实敏捷开发模式的提出和壮大也是被现实所迫。近年来软件需求变化越来越迅速，如何快速响应这种变化及时推出满足市场需要的产品是对软件从业者的巨大考验，而敏捷开发模式就是一种较好的解决方案，可谓是银弹。

二、Scrum 简介

Scrum 很多研发团队都在用了，Scrum 是当前最流行的一种敏捷开发模式，原因我认为关键是该模式简单明了，给出了具体的实践方式，可操作性很强。

Scrum 由一组迭代周期组成，每个迭代(sprint)为 2-4 周，在每个迭代中都依次有 planning (计划)，daily standup meeting(每日站会)， review(评审) 和 retrospective (回顾)会议组成，由 ScrumMaster (Scrum 负责人)来召集这些会议，当然由团队负责人担任 ScrumMaster 也是比较常见的；有些团队也采用成员轮流担任 ScrumMaster 的方式，这样可以增强每个团队成员的主人翁和责任意识等。

三、敏捷测试模式

敏捷开发模式大家都耳熟能详了，但敏捷测试模式还是比较新颖的名词，不能说是本人的创造，但本人在当初组建测试团队时就有运用敏捷开发模式的想法，团队成立后便着手实践该模式并坚持至今，自认为是取得了良好的效果。

当然肯定有人会反驳我的观点，他们认为敏捷开发模式每个迭代都包括设计，开发和测试，无需单独出来一个敏捷测试模式。这种观点当然是正确的，但无论什么模式都要应用到实际工作中去。我所在的公司测试部门是独立的部门并独立开展测试工作，不属于某一具体的开发团队，也就是说开发团队并没有人进行专门的测试工作，在职能架构上研发部和测试部也是并行独立的。

在这种情况下，测试团队应该根据产品计划合理的开展工作，采用敏捷测试模式就是一种不错的选择。

四、Scrum 在测试团队的具体实践

下面具体介绍下我所在的团队的敏捷测试模式实践。

我们采用的仍是业界流行的 Scrum 作为具体的敏捷测试模式，一般情况下每两周一个 sprint(冲刺，也就是迭代的意思)，每个迭代由计划，每日站立，演示和总结共四类会议组成。

在计划会议中由 ScrumMaster(ScrumMaster 负责人，由部门负责人也就是本人兼任)根据产品计划列出本迭代的 backlog (待办事项，有的地方也称为 User Story 既用户故事)，之后再将每个待办事项细分为多个任务，每个任务都会指定具体的负责人和预估的花费时间，这个时间以小时为单位，一般最小单位为 2 小时，最大为 16 小时。太短的时间会导致任务过多，可通过整合到其他任务的方式处理；太长的时间则导致任务较粗放，难以把控进度情况和出现的问题。每个任务的预估时间不一定准确，这需要 ScrumMaster 的经验或者采用团队每个成员进行估算然后取平均值的做法来设定。需要注意的是：请把这个会议的时间控制在一小时内，经常见到这个会议超长的现象，主要原因是没有提前计划，在会议中才开始列出待办事项，我认为待办事项在开会前 ScrumMaster 就应该已经胸有成竹了，在这个会议中只需要把待办事项具体分解为多个任务即可，甚至于有些待办事项也提前分解好了，在这个会议中只是让团队成员认领任务或者直接分配给团队成员即可。

让团队成员自己认领任务还是进行分配需要根据团队的具体情况进行处理，我所在的团队一般情况下都是由 ScrumMaster 来分配的，因为自由认领的情况下很可能会出现一些任务大家去争抢，一些任务没人愿意去认领的尴尬情况。当然分配任务也是根据每个团队成员的所长和发展方向作为依据的。

之后每个工作日早上上班一刻钟后召开每日站会。站会，顾名思义，就是站着开会，这样会议的时间不会太长，大家也能集中精力，会议一般控制在十分钟以内。在这个会议中，每个团队成员分别回答三个问题，这三个问题依次是:

1、上一个工作日做了什么？

2、今天计划做什么？

3、遇到什么问题或者困难或者说需要什么帮助？

其他的就不要说了，如果真的需要花较多的时间，就应该另外安排时间进行。

在每日站会中，注意团队成员是在彼此进行沟通和承诺，以便让大家都对任务做到心中有数，而不要搞成是向 ScrumMaster 汇报工作，这两者是不同的。

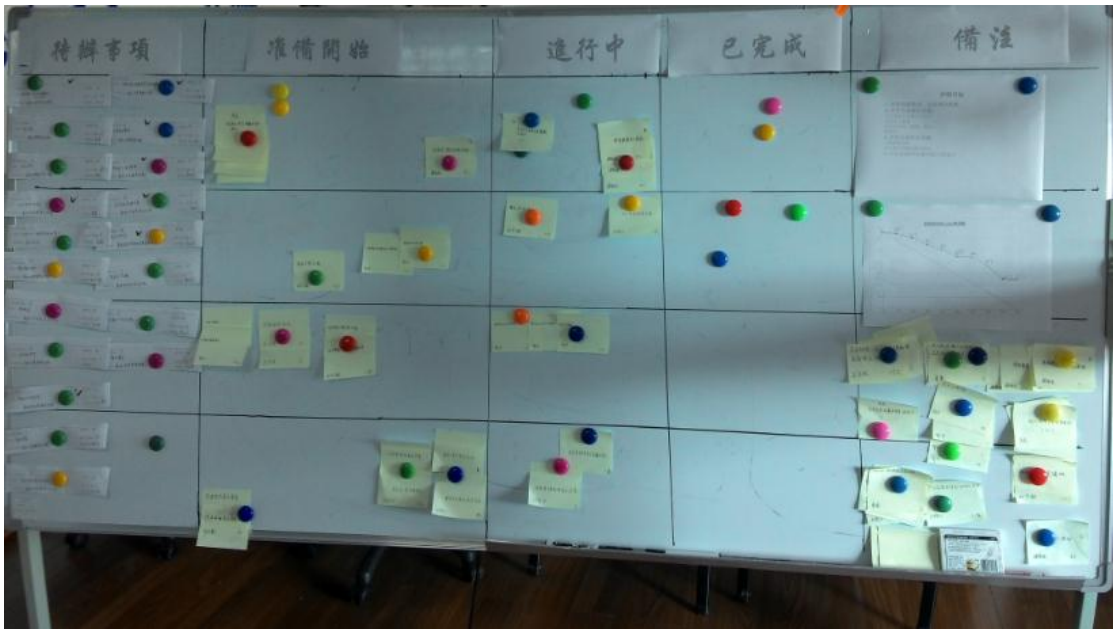
在开始实施 Scrum 的时候，强烈建议准备一个较大的白板和一些即时贴。在这个白板上先划出 4 列，这 4 列的列名分别是：待办事项，准备开始，进行中和已完成；再划出横行，可由团队成员数加一来确定横行的数目。在计划会议中，就可以把确定的任务

写到即时贴中，一个任务用一张即时贴，可用 A4 纸打印出待办事项的内容，待办事项的格式可如下图所示。

待办事项编号：	优先级：
名称：	估计工作量：
演示方式：	所有者：

待办事项的优先级越高，则应该贴在越靠上的行的待办事项列中，之后将任务即时贴都贴在白板的待办事项行对应的准备开始列中。

在开每日站会的时候，团队成员都面对白板围成一个半圆。当团队成员说完其上个工作日完成了什么和今天准备做什么之后，就可以分别移动对应的任务即时贴从进行中到已完成和从准备开始到进行中了，如下如所示。



这里如果第三个问题的答案是肯定形式的（遇到问题或者困难等），则可能无需移动即时贴了，也就是说遇到路障了，则可以在对应的即时贴中特别标注下原因，ScrumMaster 和团队相关的其他成员此时应该对路障高度关注，搞清楚原因并试图给出解决方案，如果需要较长时间才能找到原因或者给出解决方案，则需要另外安排时间，不要占用过多的会议时间。

当迭代进行了一段时间，大家都对 Scrum 有了比较深入的理解后，就可以用更方便的电子白板来取代白板和即时贴了，常见的有 https://www.pivotaltracker.com，这个比较

方便灵活，微软的 TFS 也自带了 Scrum，支持中文，使用起来很方便。不过使用电子白板后，可能就不是所有人都站着开会了，我们是 ScrumMaster 坐在电脑前移动卡片，其他团队成员站在其周围形成一个半圆。还有的团队是到会议室去开每日站会，大家都是坐着的，不过此时大家都知道这个会议不会开太长时间的，就不必要拘泥于必须要站着的形式了。

一般来说一个 Scrum 团队的成员数应在 5-9 人之间，如果团队成员多于 9 人，建议将其拆分为两个小的 Scrum 团队分别实施各自的 Scrum。

到了迭代结束的那天，可以在下午召开演示和回顾会议，我们一般将这两个会议放在一起召开，当然一定是先演示后回顾。

有人要问了，研发有产出可以演示和评审，测试演示什么呢？这个问题问得好。我们是有需要演示的就演示，没有就不演示。主要是对新的测试方法和技术的演示，比如某个成员采用了一种新的测试用例设计方式，那就可以让其演示下具体的设计和编写思路和做法；又比如某个成员掌握了一种新的自动化测试技术或者方法，也可以演示下整个过程，还可以让成员分享自己的经验等等。演示会增强演示者的自信心，同时团队中的其他成员得到了宝贵的经验和学习的机会。

这个会议的时间控制在一小时以内，会议时间太长了大家都有疲惫感，所以演示者在会议召开前可以适当准备下，比如制作一个简单的 PPT，准备好要演示的环境等。

最后的回顾会议其实就是总结会议，对本迭代进行回顾总结，目的就是 CMM 中所说的最高级别持续改进：保持做得好的，改进做得不够好的。这个会议最多半小时就足够了。可以采用给每个成员发放一张卡片，正面写出本迭代自己认为做得好的至少三个方面，反面写出本迭代自己认为做得不够好的至少三个方面。在开始阶段，大家都能写出很多，但当迭代了多次后，可能大家写不出来了或者写的和以前的迭代的内容差不多，这时候可采取针对本迭代内的每个待办事项逐一进行回顾和总结，这样更具有针对性。如果大家还是实在写不出来做得不够好的，那可能说明团队确实很优秀了，那也可以往后减少甚至不用开总结会议了。

ScrumMaster 收集大家的卡片并将其在白板中列出，如果总计做得好的或者做得不够好的数量超过三个，则可以用举手投票的方式选中得票数最多的前三个，之后对做得不够好的前三个，需要大家分别讨论下解决方案，最后由 ScrumMaster 整理出最终的解

决方案并指定解决方案的跟进负责人，这样到下个迭代开总结会议的时候再来看看上个迭代的这三项，看看究竟有无改进，长期坚持下去，必定能使团队朝着更好的方向发展起到好的作用。

五、总结

总得来说，采用敏捷模式开展测试工作，使得每个团队成员

都非常清楚自己的目标和当前工作状态，每日站会使得大家能够有效沟通并尽快得知和解决出现的问题，同时报告自己的工作进度也无形中产生了一种压力，避免了有些员工前几天比较懒散，到了最后几天才匆忙赶工的情况；报告当天的工作计划也是一种承诺，会督促团队成员尽量按时完成任务。迭代的方式使得团队成员对自己的工作有了强烈的节奏感，这种节奏感我认为对于工作效率的提升是至关重要的，因为团队成员清楚的知道自己每天应该做什么同时也知道团队中的其他成员正在做什么，大家都为同一个目标而努力，到了迭代结束的日子大家进行演示和总结，这样会促进彼此的成长和团队凝聚力，使其不断的持续改进，长期坚持下去，这样的团队必定是一个战斗力很强工作效率很高的团队。

参考文献

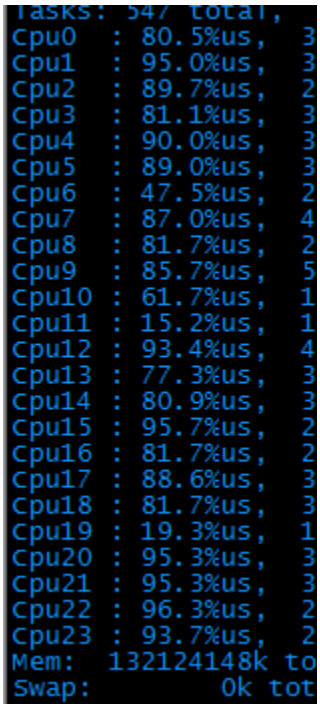
1、www.baidu.com
2、www.mountaingoatsoftware.com/scrum
3、www.scrumalliance.org
4、www.controlchaos.com

性能测试瓶颈定位
—磁盘 IO 和线程切换过多

◆ 作者：搜狗测试 曹承臻

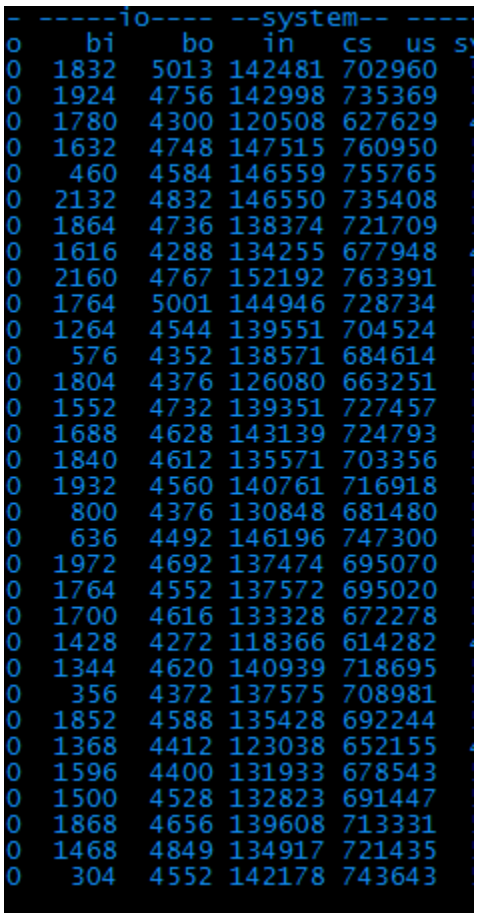
近期在一个性能测试项目中遇到了一个调优的过程。分享一下给大家。

1、第一次打压时，发现 A 请求压力 80tps 后，CPU 占用就非常高了（24 核的机器，每个 CPU 占用率全面飙到 80%以上），且设置的检查点没有任何报错。



2、了解了一下后台实现逻辑：大体是这样的：服务器接到请求后，会再到另一台 kv 服务器请求数据，拿回来数据后，根据用户的机器码做个性化运算，最后将结果返回给客户端，期间会输出一些调试 log。

查了下，kv 服务器正常，说明是本机服务服务器的问题。具体用 vmstat 命令看一下异常的地方。



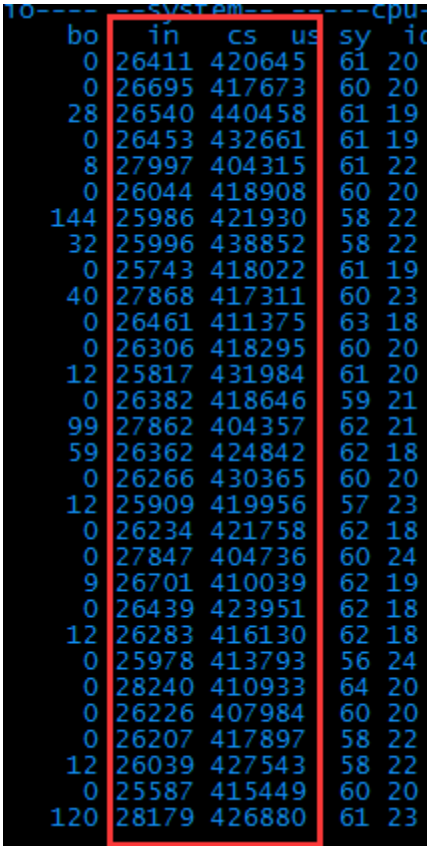
```
- -----io---- --system-- ----
o     bi     bo    in    cs  us sy
0   1832   5013 142481 702960
0   1924   4756 142998 735369
0   1780   4300 120508 627629
0   1632   4748 147515 760950
0    460   4584 146559 755765
0   2132   4832 146550 735408
0   1864   4736 138374 721709
0   1616   4288 134255 677948
0   2160   4767 152192 763391
0   1764   5001 144946 728734
0   1264   4544 139551 704524
0    576   4352 138571 684614
0   1804   4376 126080 663251
0   1552   4732 139351 727457
0   1688   4628 143139 724793
0   1840   4612 135571 703356
0   1932   4560 140761 716918
0    800   4376 130848 681480
0    636   4492 146196 747300
0   1972   4692 137474 695070
0   1764   4552 137572 695020
0   1700   4616 133328 672278
0   1428   4272 118366 614282
0   1344   4620 140939 718695
0    356   4372 137575 708981
0   1852   4588 135428 692244
0   1368   4412 123038 652155
0   1596   4400 131933 678543
0   1500   4528 132823 691447
0   1868   4656 139608 713331
0   1468   4849 134917 721435
0    304   4552 142178 743643
```

3、从图中可以直观的看出，bi、bo、in、cs 这四项的值都很高，根据经验，bi 和 bo 代表磁盘 io 相关、in 和 cs 代表系统进程相关。一个一个解决吧，先看 io。

4、用 iostat –x 命令看了下磁盘读写，果然，磁盘慢慢给堵死了。

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rsec/s    wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00     1.00    0.00    3.00      0.00     24.00     8.00     0.00    0.33   0.33   0.10
dm-0              0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
dm-1              0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    3.00      0.00     24.00     8.00     0.00    0.33   0.33   0.10
dm-3              0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
dm-4              0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
sdb               0.00    88.00  202.00  558.00  26608.00   6344.00    43.36     1.50    1.98   1.00  75.70
```

5、看了下过程，只有写 log 操作才能导致频繁读写磁盘。果断关闭 log。重新打压试下。



bo	in	cs	us	sy	id
0	26411	420645	61	20	
0	26695	417673	60	20	
28	26540	440458	61	19	
0	26453	432661	61	19	
8	27997	404315	61	22	
0	26044	418908	60	20	
144	25986	421930	58	22	
32	25996	438852	58	22	
0	25743	418022	61	19	
40	27868	417311	60	23	
0	26461	411375	63	18	
0	26306	418295	60	20	
12	25817	431984	61	20	
0	26382	418646	59	21	
99	27862	404357	62	21	
59	26362	424842	62	18	
0	26266	430365	60	20	
12	25909	419956	57	23	
0	26234	421758	62	18	
0	27847	404736	60	24	
9	26701	410039	62	19	
0	26439	423951	62	18	
12	26283	416130	62	18	
0	25978	413793	56	24	
0	28240	410933	64	20	
0	26226	407984	60	20	
0	26207	417897	58	22	
12	26039	427543	58	22	
0	25587	415449	60	20	
120	28179	426880	61	23	

6、Bi 和 bo 降到正常值了，说明磁盘的问题解决了。但是上下文切换数竟然达到了每秒 40 万次！好可怕~

7、只知道上下文切换数很大，怎么知道是在哪些进程间切换呢？

到网上搜了一个脚本，这个脚本用来统计特定时间内进程切换的 top20 并打印出来。

```
#! /usr/bin/env stap
#
#
global csw_count
global idle_count
probe scheduler.CPU_off {
csw_count[task_prev， task_next]++
idle_count+=idle
}

function fmt_task(task_prev， task_next)
```

```
{
return sprintf("%s(%d)->%s(%d)",
task_execname(task_prev),
task_pid(task_prev),
task_execname(task_next),
task_pid(task_next))
}

function print_cswtop () {
printf ("%45s %10s\n",  "Context switch",  "COUNT")
foreach ([task_prev,  task_next] in csw_count- limit 20) {
printf("%45s %10d\n",  fmt_task(task_prev,  task_next),  csw_count[task_prev,  task_next])
}
printf("%45s %10d\n",  "idle",  idle_count)

delete csw_count
delete idle_count
}

probe timer.s($1) {
print_cswtop ()
printf("------------------------------------------------------------\n")
}
```

保存成 cs.stp 后，用 stap cswmon.stp 5 命令执行下。

Context switch	COUNT
discover_agent_(12250)->swapper(0)	70395
swapper(0)->discover_agent_(12250)	70370
discover_agent_(12250)->swapper(0)	64330
swapper(0)->discover_agent_(12250)	64303
discover_agent_(12250)->swapper(0)	61493
swapper(0)->discover_agent_(12250)	61466
discover_agent_(12250)->swapper(0)	33717
swapper(0)->discover_agent_(12250)	33706
discover_agent_(12250)->swapper(0)	32897
swapper(0)->discover_agent_(12250)	32881
discover_agent_(12250)->swapper(0)	19274
swapper(0)->discover_agent_(12250)	19265
discover_agent_(12250)->swapper(0)	17311
swapper(0)->discover_agent_(12250)	17306
discover_agent_(12250)->swapper(0)	13720
swapper(0)->discover_agent_(12250)	13714
discover_agent_(12250)->swapper(0)	6985
swapper(0)->discover_agent_(12250)	6984
discover_agent_(12250)->swapper(0)	5184
swapper(0)->discover_agent_(12250)	5179
idle	356852

8、发现是 discover 进程在反复和系统进程进行切换。从此消耗了大量资源。

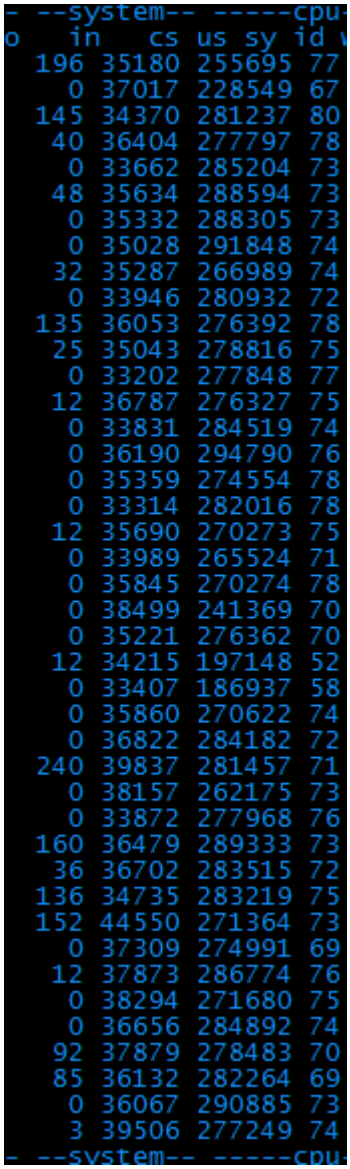
9、从网上查了下减少切换进程的一些方法:

Context Switching(內文切換)

1. Def：
   當CPU從一個process切到另一個process執行之前，OS必須保存原有process的執行狀態(eg. pc, CPU register store in PCB)，同時載入New Process的執行狀態(eg. Load PC, CPU register值from its PCB)稱之Context switching
   而 Context switching是一**系統負擔**，其時間長短，幾乎完全取決於Hardware因素
2. 如何降低context switch之負擔
   - 法一：多加利用registers
     - Def：如果registers數量夠多，則每一個process可有(prirate)自己的registers set
       所以在context switching時，OS只要切換pointer指向不同的registers set即可完成
     - 優點：負擔最小(速度最快)
       (∵避免Memory store/Load之時間)
     - 缺點：不適用於register數量少之情況
   - 法二：利用Thread(light-weighted process)來取代process
     得以降低context switching負擔
     ∵同一個process內的Threads彼此共享code section, data section, open file, OS resources
     ∴私有資訊不多，context switching時不需大量的memory access，Store/Load量少
   - 法三：System process擁有自己的register set
     當user process與system process之間的context switching，OS只要改變Register Set指標即可

开发随后改了下：将线程数开大了一倍，控制在一个进程中。

重新打压了一下。发现上下文切换数降低到 25 万次左右。



此时的性能数据可以达到每秒 260 次左右，远远高于之前的 80 次。已经达到可以上线的需求。

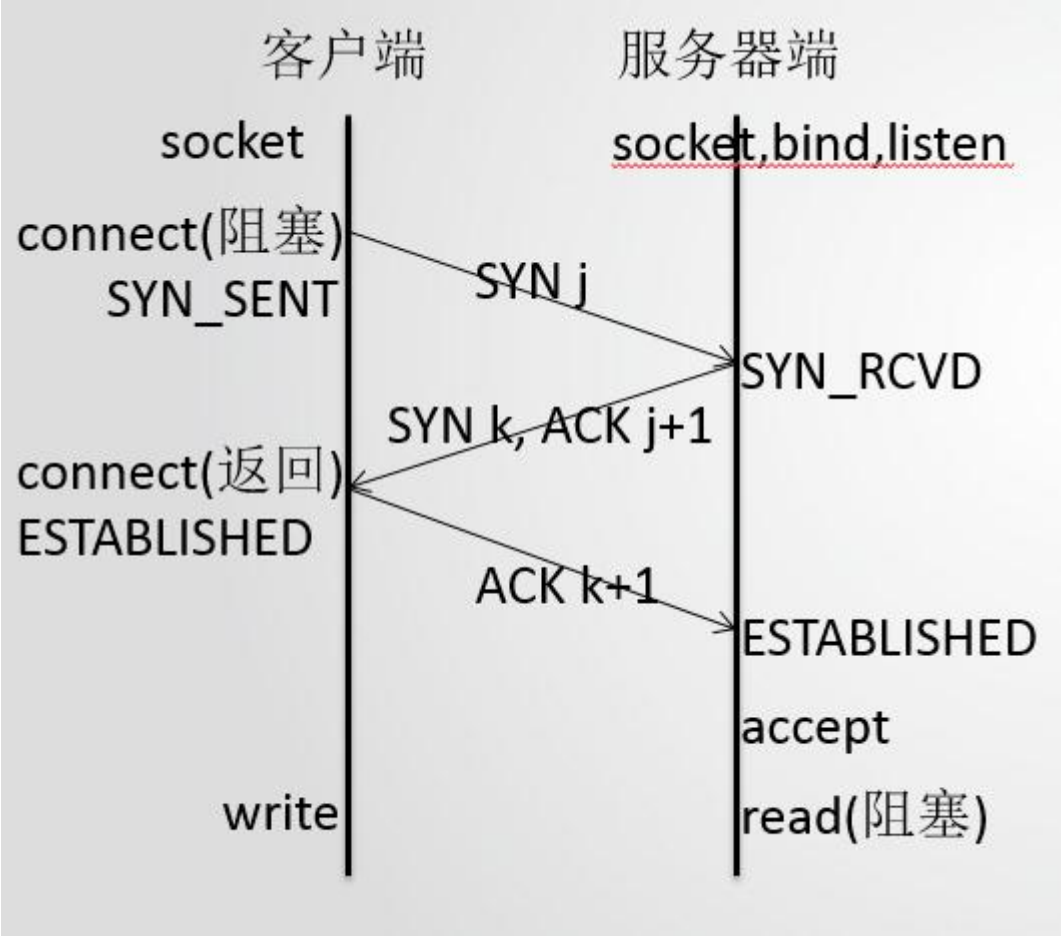
但是由于页面中断书和上下文切换数还是很高，后续还是需要优化~

TCP 协议使用操作详解

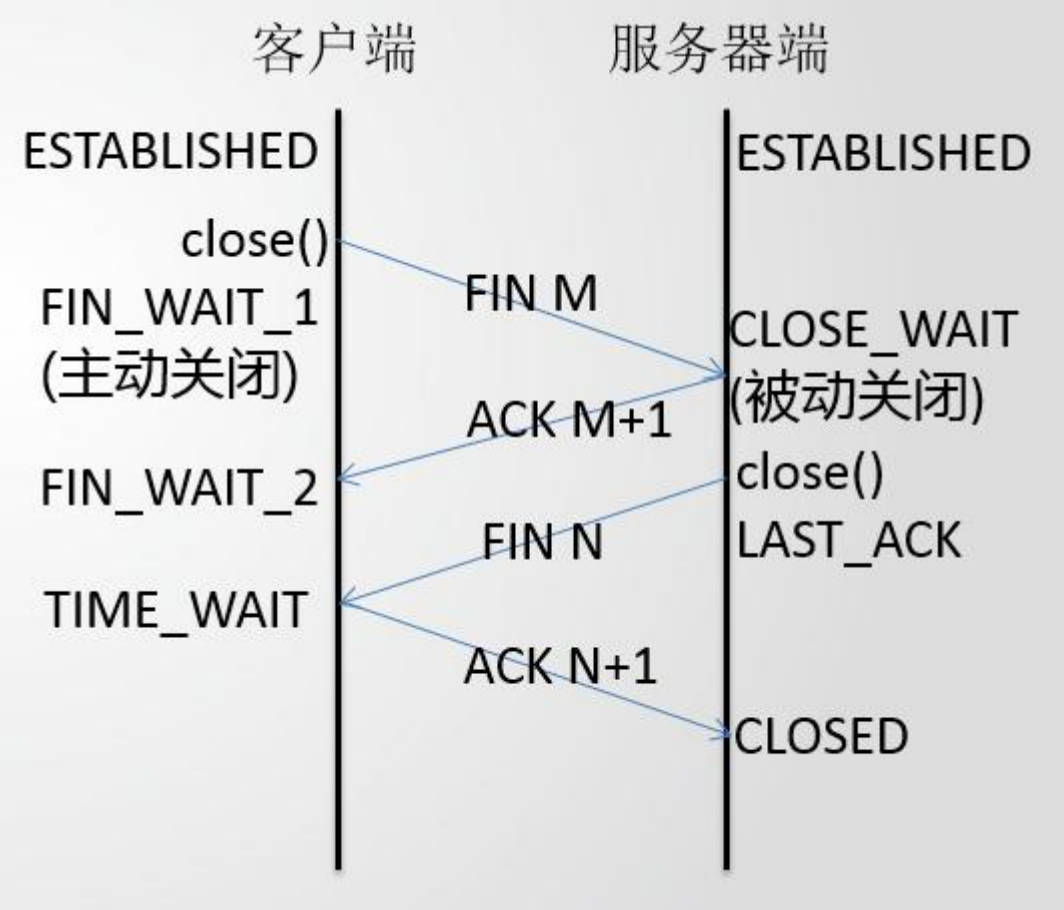
◆搜狗测试：陈之歌

TCP 简介

TCP 是面向连接的，在传输数据之前要先和对端建立一个连接，建立连接的过程我们通常叫做 3 次握手。下面通过图片来给大家展示 3 次握手的过程



在数据传输完之后我们可能不再需要这个连接，那么就需要把连接断开。断开连接的过程被称之为 4 次挥手。



关于三次握手和四次挥手的过程，有不明白的可以移步：http://www.51testing.com/html/67/n-3708167.html，本篇文章主要讨论以下几种情况：

试图与一个不存在的端口建立连接

服务器端口还没有监听，我们的客户端就调用 connect，试图与其建立连接。这是会发生什么呢？没错，这符合触发发送 RST 分节的条件，目的为某端口的 SYN 分节到达，而端口没有监听，那么内核会立即响应一个 RST，表示出错。客户端 TCP 收到这个 RST 之后则放弃这次连接的建立，并且返回给应用程序一个错误。正如上面所说的，建立连接的过程对应用程序来说是不可见的，这是操作系统帮我们来完成的，所以即使进程没有启动，也可以响应客户端。

试图与一个不存在的主机上面的某端口建立连接

这也是一种比较常见的情况，当某台服务器主机宕机了，而客户端并不知道，仍然尝试去与其建立连接。根据上面的经验，这次主机已经处于未启动状态，操作系统也帮

不上忙了，那么也就是连 RST 也不能响应给客户端，此时服务器端是一种完全没有响应的状态。那么此时客户端的 TCP 会怎么办呢？据书上介绍，如果客户端 TCP 没有得到任何响应，那么等待 6s 之后再发一个 SYN，若无响应则等待 24s 再发一个，若总共等待了 75s 后仍未收到响应就会返回 ETIMEDOUT 错误。这是 TCP 建立连接自己的一个保护机制，但是我们要等待 75s 才能知道这个连接无法建立，对于我们所有服务来说都太长了。更好的做法是在代码中给 connect 设置一个超时时间，使它变成我们可控的，让等待时间在毫秒级还是可以接收的。

Server 进程被阻塞

由于某些情况，服务器端进程无法响应任何请求，比如所在主机的硬盘满了，导致进程处于完全阻塞，通常我们测试时会用 gdb 模拟这种情况。上面提到过，建立连接的过程对应用程序是不可见的，那么，这时连接可以正常建立。当然，客户端进程也可以通过这个连接给服务器端发送请求，服务器端 TCP 会应答 ACK 表示已经收到这个分节（这里的收到指的是数据已经在内核的缓冲区里准备好，由于进程被阻塞，无法将数据从内核的缓冲区复制到应用程序的缓冲区），但永远不会返回结果。

我们杀死 server

这是线上最常见的操作，当一个模块上线时，OP 同学总是会先把旧的进程杀死，然后再启动新的进程。那么在这个过程中 TCP 连接发生了什么呢。在进程正常退出时会自动调用 close 函数来关闭它所打开的文件描述符，这相当于服务器端来主动关闭连接——会发送一个 FIN 分节给客户端 TCP；客户端要做的就是配合对端关闭连接，TCP 会自动响应一个 ACK，然后再由客户端应用程序调用 close 函数，也就是我们上面所描述的关闭连接的 4 次挥手过程。接下来，客户端还需要定时去重连，以便当服务器端进程重新启动好时客户端能够继续与之通信。

当然，我们要保证客户端随时都可以响应服务器端的断开连接请求，就必须不能让客户端进程再任何时刻阻塞在任何其他的输入上面。比如，书上给的例子是客户端进程会阻塞在标准输入上面，这时如果服务器端主动断开连接，显然客户端不能立刻响应，因为它还在识图从标准输入读一段文本……当然这在实际中很少遇到，如果有多输入源这种情况的话开通通常会用类似 select 功能的函数来处理，可以同时监控多个输入源是否准备就绪，可以避免上述所说的不能立即响应对端关闭连接的情况。

Server 进程所在的主机关机

实际上这种情况不会带来什么更坏的后果。在系统关闭时，init 进程会给所有进程发送 SIGTERM 信号，等待一段时间（5~20 秒），然后再给所有仍在运行的进程发送 SIGKILL 信号。当服务器进程死掉时，会关闭所有文件描述符。带来的影响和上面杀死 server 相同。

Server 进程所在的主机宕机

这是我们线上另一种比较常见的状况。即使宕机是一个小概率事件，线上几千台服务器动不动一两台挂掉也是常有的事。主机崩溃不会像关机那样会预先杀死上面的进程，而是突然性的。那么此时我们的客户端准备给服务器端发送一个请求，它由 write 写入内核，由 TCP 作为一个分节发出，随后客户阻塞于 read 的调用（等待接收结果）。对端 TCP 显然不会响应这个分节，因为主机已经挂掉，于是客户端 TCP 持续重传分节，试图从服务器上接收一个 ACK，然而服务器始终不能应答，重传数次之后，大约 4~10 分钟才停止，之后返回一个 ETIMEDOUT 错误。

这样尽管最后还是知道对方不可达，但是很多时候我们希望比等待 4~10 分钟更快的知道这个结果。可以为 read 设置一个超时时间，就得到了一个较好的解决方法。但是这样还是需要等待一个超时时间，事实上 TCP 为我们提供了更好的方法，用 SO_KEEPALIVE 的套接字选项——相当于心跳包，每隔一段时间给对方发送一个心跳包，当对方没有响应时会一更短的时间间隔发送，一段时间后仍然无响应的话就断开这个连接。

服务器进程所在的主机宕机后重启

在客户端发出请求前，服务器端主机经历了宕机——重启的过程。当客户端 TCP 把分节发送到服务器端所在的主机，服务器端所在主机的 TCP 丢失了崩溃前所有连接信息，即 TCP 收到了一个根本不存在连接上的分节，所以会响应一个 RST 分节。如果开发的代码足够健壮的话会试图重新建立连接，或者把这个请求转发给其他服务器。

总结

这里尽可能详细的介绍了在 TCP 建立连接、传输数据、关闭连接过程中代码层面以及内核层面的所有工作，可以更好的理解 TCP 协议的工作机制。列出的这 7 种情况大部分是我们在平时工作中可以遇到的，也是我们作为测试工程师需要关注的，有些情况是

TCP 协议自动完成，有些情况是需要在代码层面完成。书上提到几种情况都非常经典，也可以帮助我们在测试中拓展思路，这里我又加上了几种自己想到的情况一起分享给大家，如果有问题可以一起讨论，一个人的思考总是片面的，大家一起思考，可以得到更深入、更成熟的见解。

如何进行 BUG 三方评估？

◆ 搜狗测试：李德栋

一、Bug 三方评估场景:

Bug 三方确认的触发点一定是在对于 Bug 的处理意见上，开发、测试、产品三方或任意两方有不同意见时:

1、开发与测试意见不统一

- 开发把 Bug 置为不是问题打回;
- 开发把 Bug 置为开发难以实现打回;
- 开发把 Bug 置为需求如此打回;
- 上线前开发修改一个 Bug，测试认为改动影响较大;

2、测试与产品意见不统一

- 产品认为此 Bug 不影响上线，测试认为问题严重;
- 产品认为此 Bug 需要修改，测试认为改动影响较大最好暂时不做修改;

3、开发与产品意见不统一

- 产品回复上线计划邮件，明确此 Bug 需要修改，开发持不同意见;

二、Bug 三方评估的方式:（几种场景适合的方式）

- cynthia: 成本最低，效率也最低，适用于不紧急的场景;
  - 需求未测完，距上线还有充分时间;
  - 用户报的线上问题，测试评估过影响较小;
- 邮件: 适用于中等紧急的问题

标题:【Bug 三方评估】

收件人：开发、产品、setest

类似上线计划邮件中的中优先级 Bug 评估;

测试过程中有争议的 Bug 偏多，需要推产品和开发尽快处理;

- 口头：效率最高，成本也最高，适用于处理重要且紧急的 Bug
  - 代码已冻结，高优先级 Bug 未解决;
  - 代码已冻结，开发要改 Bug;
  - 代码已冻结，线上用户反馈了影响严重的 Bug;

三、评估原则:

- 三方共同评估，非一方或两方决定;
- 实事求是、只说事实、不说自己的推断;
- 明确都是以“保证产品质量”为最终目的;

四、评估内容:

1、Bug 现象

2、影响是否严重

- 影响严重
  - 系统所提供的功能或服务受到明显的影响
  - 主要功能丧失，数据被破坏，系统崩溃、悬挂，死机，数据不能保存;
  - LOGO 不对，有错字，提示信息不正确
- 影响不严重
  - 系统所提供的功能或服务没有受到明显的影响
  - 版本信息、文件描述、签名、changelog 存在明显勘误，导致公司口碑受到影响的问题

3、重现路径是否常用路径

- 常用路径
  - 常见环境：网络，系统，硬件
  - 常见场景：例如用户经常访问的网站，使用用户常用的第三方软件
  - 常用数据
  - 常见操作：使用全屏模式（习惯于使用全屏模式的用户会频繁使用）

4、是否有很高的重现几率

5、线上版本是否已经存在

上线在即，开发要求修改一个底层函数，目的是修复线上一个长期存在的问题；需三方评估

6、改动方法

浏览器一款插件在 XP 下显示异常，开发的修改方案是，对 XP 用户不显示；需三方评估

7、改动范围

开发为了修复插件在副屏显示异常的 Bug，重构了扩展模块；需三方评估

8、实现难度

9、影响范围

改动影响所有 win7 用户

10、风险

开发改动一笔数据库操作相关的底层函数，回归了主路径功能未见异常，但总觉得不放心；

11、开发修改需要的时间

开发修改一个 Bug，且跟测试说改动很小不用回归，但修改了很长时间，或者改动多次仍未改好；

12、改动验证需要的时间

开发要更改浏览器的签名文件，此时需要长时间做冲突测试，及签名相关的功能验

证;

13、上线时间点

上线目的是为了解决线上急需解决的崩溃，必须尽快上线，此时有开发想搭车解决其它 Bug;

14、其它经验

新入职 1 周的开发修了一个 Bug，跟测试说不用回归

开发大神修了一个 Bug，跟测试说不用回归;

五、评估结论:

1、三方意见统一

2、意见不统一

经过一系列的三方确认，最终意见仍然不统一；上报各自 Leader 确定最终解决方案

六、最终结论:

1、是否推迟上线

2、下一版本修改

3、带着风险上线

性能测试过程分享

◆作者：搜狗测试 曹承臻

一直在做性能测试，在此把性能测试过程和遇到的问题和大家分享下，纯手打~

从接到性能测试的测试需求，到最终测试完成，都需要经过哪些阶段呢？在这里结合我具体项目，总结分享一下。

接到性能测试需求阶段需要做的事:

首先要评估测试需求是否合理，并不是所有的性能测试需求都需要直接来安排测试，而是评估下是否需要做本次性能测试。产品提出需要做性能测试是基于用户的考虑，可能并不了解实现。如果了解具体实现了后，有些看似请求量大的测试，可能并不需要做。举个例子：假如一个请求，每次用户开启应用时都会发送到服务器，服务器则会返回给客户端本账号在好友中的积分排名情况。从产品的角度认为，每次应用启动，都会触发服务器查询一次数据库。这样会数据库会造成很大压力。而测试再了解了具体实现后发现：针对每个用户机器码的排序数据是从 redis 服务器返回的，而 redis 服务器会每隔一小时请求存储的 mysql 服务器来更新账号排名信息，这样看来 mysql 服务器请求频率很低，没有任何压力。由于 redis 服务器的性能之前已经测试过类似的，没有性能问题，所以这次并不需要对 mysql 服务器做压力测试。

其次，如果确定要进行性能测试，就需要评估性能测试的方案。包括环境搭建、逻辑了解、数据准备、脚本编写、测试过程、问题定位、修改优化、回归、出报告的时间。需要强调的是，性能测试开始的时间一定是功能测试已经通过了。否则进行性能测试会存在修改功能逻辑导致性能发生变化，性能测试就没有任何指导意义了。

环境搭建:

这里主要指测试服务器的搭建和打压环境的搭建。测试环境可以有开发来搭建。原则上测试服务器配置不能高于线上服务器的配置，且测试服务器部署的服务要尽量接近线上服务器，比如说线上服务器上运行了 3 个服务，那么测试服务器也要运行同样类似，包括打压脚本上的设计也要考虑（后面会讲），这样得出的结果才具有指导意义。

再就是 linux 打压机的部署。具体部署方法就不在此赘述了。

逻辑了解:

这里主要开始着手了解整个性能测试的业务逻辑。一般需要了解请求个数，请求参数含义等。除此之外，在这里要强调几个新手容易忽视的问题：就是打压测试服务器时，要和线上服务器做明确隔离。不要简单认为所有的请求都是指向测试服务器，就认为是只向测试服务器打压。主要分为三种情况:1、测试请求会经过跳转链接到线上服务器。2、测试请求的 js 会异步请求线上资源。3、测试服务器会经过逻辑处理后返回给客户端，而这里的逻辑处理可能包括到线上服务器查询数据。

数据准备:

这里主要指性能测试有效数据的准备。为什么说是有效数据呢？举个例子：加入你手动写完脚本后，跑一下脚本，发现服务器返回没有报错。都是 200 的 response。这是否就说明是有效的打压呢？未必！应该回放脚本时，通过抓包工具抓包看下，对比真正使用场景中返回的 response。看下内容格式是否一致。如果你的打压脚本返回的是空的 200 请求或者仅仅有关键字，但是内容都是空的，而真正场景返回的是大小为 15K 的 json。这说明你的打压场景是有问题的。应该结合服务器逻辑和询问开发。构造出可以让服务器认为是正常的请求来处理。从而返回正常的数据。

脚本编写:

脚本编写是整个性能测试过程最重要的部分，脚本编写关系到性能测试是否能够模拟用户需求，是否真正可以对服务器构成压力。从数据准备中可知，不同的数据可以导致服务器不同的处理逻辑，对服务器的压力也不同，所以整个脚本编写的目的就是能够尽量模拟上线后用户对服务器的压力情况。举个例子：在有登陆、发帖、点赞、退出登录的网站测试脚本时，如果脚本对于登陆、发帖、点赞、退出登录的过程是等同比例的。这显然不符合用户的真正使用场景。正常登陆和退出是等比例的，而点赞和发帖应该是存在一个倍数关系。这样就需要用 block 块技术来拆分每个过程，对不同过程设置不同的迭代次序。对于脚本编写，新手可以先学着录制脚本，然后再优化脚本。具体常用的方法有使用事务、关联，添加 header、添加 cookie、ip 欺骗、block 技术等。具体情况具体分析。对于 LoadRunner 如果要用的顺手，势必要学习 LoadRunner 的语言——C 语言。这样写起脚本来会轻松一些。

测试过程:

这里主要指打压的过程了。打压过程并不是放那里不管，需要时刻关注被测服务器的性能指标，结合 LoadRunner 场景的曲线来动态判断是否存在瓶颈。LoadRunner 场景曲线主要关注 HPS、TPS、responsetime。服务端主要监控 CPU、内存、磁盘 IO、网络 IO。然后从这几方面再层层深入查找问题。另外，值得强调的是，打压过程不仅需要关注被测服务器的指标，也需要关注打压 agent 的性能指标是否正常。比如说如果并发数一直上不去，可能是打压机本上连接数受限导致的。也可能是打压机自己出口带宽满了导致的。

问题定位&修改优化&回归:

问题定位是一个比较考验个人综合技术能力的地方。也是整个性能测试的核心所在。需要强调的是，不是看到某个参数遇到问题了，就直接可以断定服务区的瓶颈就在这个地方。比如:

大量的页调入请求导致内存队列的拥塞网卡的大吞吐量可能导致更多的 CPU 开销大量的 CPU 开销又会尝试更多的内存使用请求大量来自内存的磁盘写请求可能导致更多的 CPU 以及 IO 问题具体是哪个部分的问题，需要再根据具体的逻辑实现和推断来逐步缩小范围。主要的过程可以参考一下步骤层层深入。服务器硬件瓶颈→网络瓶颈（对局域网，可以不考虑）→服务器操作系统瓶颈（参数配置）→中间件瓶颈（参数配置，数据库，web 服务器等）→应用瓶颈（SQL 语句、数据库设计、业务逻辑、算法等）
注：以上过程并不是每个分析中都需要的，要根据测试目的和要求来确定分析的深度。对一些要求低的，我们分析到应用系统在将来大的负载压力（并发用户数、数据量）下，系统的硬件瓶颈在哪儿就够了。

最后就是开发优化，测试回归，这里需要强调一点：测试回归时，如果开发优化时修改了功能逻辑，则需要根据改动回归功能逻辑是否正常。不能只盯在性能指标满足了就 OK 了。

性能报告:

最后就是总结性能测试报告了，性能测试报告没有固定的格式，但是大体需要包括以下几个方面:

- 整体结论：本次性能测试的整体结论。是否符合预期，是否可以上线，可以抗

住多少常规用户数。

- 具体推算方式：主要指给出整体结论的参考数据、计算公式等。
- 具体性能图标：服务器 CPU、内存、网络 IO、磁盘 IO、TPS、responsetime 等指标，涉及到数据库的还需要给出与服务器相关的指标。

测试的核心价值和能力是什么？

◆ 作者：刘琛梅

最近发生的几件事情让我开始思考这个问题——测试的核心价值和能力是什么：

故事 1: 前几天帮别的项目组面试招人，候选者是一位拥有 8 年测试经验的资深人士，简历很漂亮。但交流了一会，我就发现了一个问题，他熟悉的业务和他现在申请的职位的业务相差甚远，但是他除了对业务非常熟悉之外，几乎没有任何思考和总结，对测试，甚至连最基本的概念都模糊不清。不过更触动我的是，和我一起面试的另外一位面试官，面试完后私下给我说，感觉有点不舒服，害怕自己 8 年后会同这位面试者一样。

故事 2: 也是帮别的项目组面试。候选者有 4 年左右的工作经验，简历很漂亮，已经在大公司里面做测试项目管理了。面试时，他对他当前所在公司的流程，研发管理方法侃侃而谈，全部都很 high level。但当我问到他一个在具体的场景下如何识别和处理项目风险，深入问他一些细节的时候，发现怎么聊都深入不下去，原来对他而言，“风险”就等于“项目延期”，一旦出现了“项目延期”，就发邮件给领导，按照流程去处理就可以了，而且他认为，这就是测试管理，这就是测试策略。

正是这两个故事，引得我思考这个问题：什么是测试的核心能力？

作为测试人员，掌握业务是必须的，但是业务知识不等于测试能力，并不是测试的核心价值更不是测试的核心能力。测试流程固然重要，但它只是固化下来的经验、管理的手段和方法。测试流程也不应该是测试的价值，测试的核心。

那么对测试而言，究竟什么是测试的核心价值？哪些是一位优秀测试应该具备的能力？作为测试者，我们又该如何去提升、突破自己？

我理解，测试的核心能力有两个：对需求的理解和把握和对产品失效规律的把握。——我们对需求进行分析，得到产品的测试范围，并确定我们的测试目标（验收标

准）；结合设计，得到产品的测试重点、测试难点，测试深度和广度。同时我们还需要结合我们对产品失效规律的把握，基于风险来进行测试。我们所有的测试，要“测什么”，“怎么测”，都是围绕上面来进行的，我理解这才是最核心的测试技术——定好测试策略。

认识到测试策略在测试的核心地位后，对我的测试职业发展之路产生了巨大的影响。记得我曾经就职的一家公司会要求测试代表在项目开始写《总体测试策略》，在项目过程中做“缺陷分析”，做“产品质量评估”。我一直觉得那玩意很虚，对这些技术不以为然，认为有这些分析思考的时间，不如去看看开发的实现代码，觉得只有编码才是真技术。但自从我被逼着开始试着认真的去写测试策略，去学习怎么把握需求，怎么确定测试的深度和广度，学习怎样做缺陷分析才最有效，怎样评估质量时，我突然好像进入了一个全新的世界，我越来越觉得测试很难，真的很难，但随着不断的思考和总结，我也越来 越感到测试的魅力。

我也逐渐理解到，对测试来说，核心价值，就是能够利用自己对产品的失效规律的把握，对需求的理解，来预防缺陷，和其他角色一样来保证产品的成功。

例如，测试参加需求（story）的确定讨论，往往能够从系统、用户的视角提出很多很有价值的问题，能够帮助团队快速澄清问题，达成一致。

测试参加开发的设计讨论，往往能够从功能之间的联系、非质量属性、容易出现 Bug 的地方提出很多开发在设计上容易忽视的地方，而这些问题点到了测试阶段，其实都是 Bug，这就起到了预防缺陷的作用。

是不是我们做到这步就够了呢？事实上，“失效规律的把握”也好，“需求的把握”也罢，归结起来还是和具体业务息息相关。测试者一旦换了产品，面对的是一些你不熟悉的业务，你掌握的失效规律，掌握的需求，特别是对用户的深入理解，可能就失效了。所以我们需要把这些知识，从测试的层面再提炼总结一下，成为测试能够“通用”的知识、技能、或者模型。

总结提炼的方法可以概括为：

1、 我们需要反复思考和理解各种基本概念，这是能够将其“通用化”的基础。

最简单最基础的，往往是最难的。举一个最简单的例子，你是否能够一句话说清楚“模块测试”，“功能测试”和“系统测试”的差别？能否说清楚“功能测试”和

“场景测试”的差别？能否说清楚“可靠性测试”和“稳定性测试”的差别？能否举出实际的例子？这个例子能否给别人说清楚让别人可以理解？

2、对一些流程性的，固化性的东西，要去理解，流程为什么要这样安排，为什么要在这个时间让这个角色来做这件事情，还要问自己，这样真的好吗？更重要的是，先不要急着否定，先问问自己是不是真的理解了。

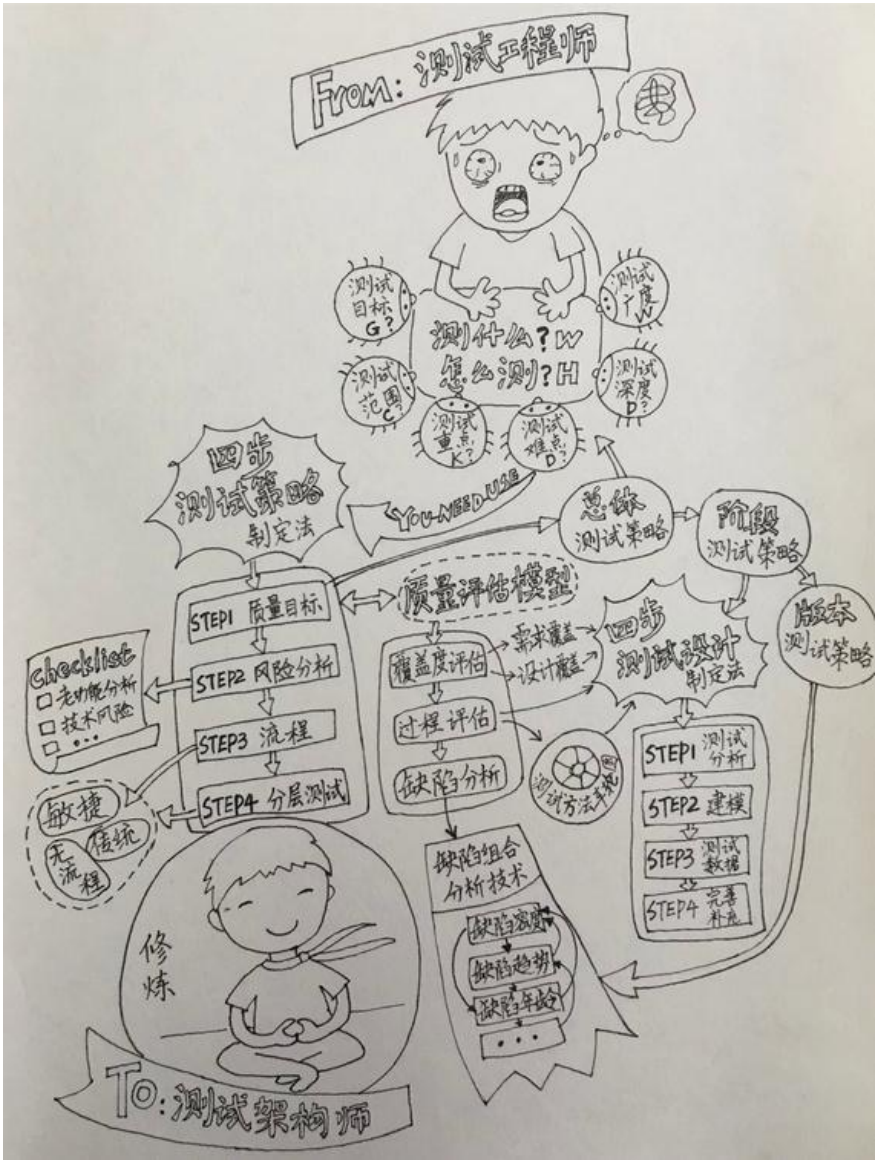
3、训练自己归纳总结，并将其“模型化”的能力。一项知识或技能，最初我们不了解它，我们不能对它描述太多，慢慢的我们可以对他描述很多，但是到最后，直到我们能够用简单概括的话能够把它说清楚了，所谓的“把书读薄”了，才是真正掌握了它。“读薄”，就是“通用化”的，“模型化”就是“通用化”的最好方法。

测试策略、测试设计技术、缺陷分析技术、质量分析技术、自动化测试技术.....只有我们能够把这些技术“通用化”，建立了模型，建立了科学的思考方法，这些东西才会真正变成我们自己的东西，我们才可以把这些技术和这些技术背后的经验用到其它的领域。

最后附上我总结的一些模型方法：

- 测试策略：四步测试策略制定法。
- 测试分析：车轮图分析法。
- 测试设计：四步测试设计法。
- 产品质量评估模型。
- 缺陷分析：缺陷分析模型。

并附它们之间的关系图一份：


FROM：测试工程师
测什么？W
怎么测？H
测试目标G？
测试广度W？
测试范围C？
测试深度D？
测试重点K？
测试难点D？
四步测试策略制定法
YOU NEED LUCK
惠普测试策略
阶段测试策略
版本测试策略
STEP1 质量目标
STEP2 风险分析
STEP3 流程
STEP4 分层测试
质量评估模型
覆盖度评估
过程评估
缺陷分析
需求覆盖
设计覆盖
四步测试设计制定法
STEP1 测试分析
STEP2 建模
STEP3 测试数据
STEP4 完善补充
测试方法参考
缺陷组合分析技术
缺陷密度
缺陷趋势
缺陷年龄
Checklist
□ 无功能分析
□ 技术风险
□ ...
敏捷
传统
无流程
修炼
TO：测试架构师

测试女巫之 Python 学习回顾篇

◆作者：王平平

一、前言:

测试女巫陪伴着大家已经 9 期了，从 2014 年 1 月份到 2016 年 1 月份一晃都过去两年了，女巫这里总结一下：前两期是“自动化即嵌入式自动化和路由器自动化”；后七期均是“统计学如何应用到实际工作上”。每一期都是实战经验！回顾这两年，非常欣慰：所幸我们一直在努力学习，而且真的在不断在进步。现在，两年了，我觉得是时候了，是时候做个总结了，尤其对于一直希望“do something”的一个不大不小的 Team Leader 来说，是时候想想你希望将自己以及自己的团队引导什么方向了，如何思考呢，首先就要对我们已经掌握的知识进行梳理，且需要环顾公司迫切的需求是什么？我们目前拥有的知识哪些需要进一步加强，哪些是公司需要，但是我们还不具备的知识。这些更需要迫切的规划，制定 Schedule，慢慢填补这些或大或小的漏洞。

其实无论对于知识，还是对于职场的心路历程都需要总结，就像一个在茂密的森林中一直在辛勤打猎的猎人，每天都在辛苦打猎，打了一屋子猎物，塞得满满的，这时需要做个整理，需要做个分类，才能知道，我们比较擅长打什么猎物，我们打的这些猎物是否符合市场的需求，如果发现价值较低的猎物比较多，价值较高的猎物比较少，我们就要反思为什么，在打猎之前我们做规划了吗？是我们的技艺需要提高吗？打了猎物后如何推销自己的猎物，如何与其它猎人竞争？如果猎人与领导有各种各样的冲突，是应该怎样化解这些冲突……

所以决定了，从这一期开始，开启总结模式，相信总结完毕后总会有不一样的收获，第一次先总结我们做自动化开发的语言：Python 学习历程，期间也会将在学习期间的一些心路历程进行一些有价值的总结。

学而时习之，不亦乐乎^_^

二、Python 学习经历回顾

1、申请学习经费

2012 年初女巫有一个想法：希望对于待测物可以实现部分功能自动化。

有了这个想法就开始上网找资料自己学习，虽然找到了 Python 这个语言也自学了一点，但是总觉得学习很难有突破，虽然能实现一小部分功能自动化，但是代码的健壮性不强，且对于复杂一些的功能就无法实现自动化，网上的资料永远都是铺天盖地的“入门经典”，如希望进一步提高，就很难找到系统化的资料，所以有了能否在外面找培训老师，可以给我们开阔思路，进而让我们的自动化开发有更大的突破？

于是请教行政部是否可以申请经费请老师给我们上课。女巫所在公司并不是很大方的公司，连女巫的领导都告诫女巫：“测试人员在公司的地位不高，学习经费？你还是别想了，那都是留给“贡献度高”的开发部门！”。幸好那时的女巫比较倔强并没有知难而退，依然递上了申请书，被一级一级不同的主管打回 4 次，一次次的说明，一次次的解释，像游戏通关一样。

印象最深的是我们一位研发处长找我面谈：“你知道吗？公司不是学校，我不能因为你勤学就给你经费让你去学习，你要首先告诉我，你的学习能给公司带来多大的价值？我需要衡量你可能带给公司的价值到底值不值得我给你批准的这个经费，另外请给出 Schedule，什么时候能实现你所承诺的这个价值。”

谈过话之后，真的觉得太不容易了，就申请一些培训经费，还需要我承诺因为花费了这些经费必须为公司创造非凡的价值，还要我提供什么时候可以提供这些价值。Oh My God！当时我真的怀疑他是让我知难而退，压根不想给我这笔费用。没关系，女巫不会被你吓倒，女巫整理了下图两个图表如【图 1】和【图 2】，当然这两个图表只是部分内容，真正给老板汇报的表格会更详细，第一个图表说明 2012 年靠我们自学已经产生的效益，以及通过培训后，我承诺可以产生的新效益，且产生效益的时间就是在当年 2013 年可以显现出来。另外说明一下：所谓的效益也就是节省人力，即相同的人力可以做更多的工作。

年份	測試類型	節約人力說明
2012 年已產生的效益	1. Smoke test 2.Boundary test 3.Pressure test	每輪測試可以節省 20 人天
2013 年預期產生的效益	1.加強已有腳本的健壯性	我們的目標是每個人從每人 1 天完成 150 個測項，達到每人 1 天完成 250 個測項
	2.增加 pressure test 的覆蓋率	
	3.增加基本功能的覆蓋率	
	4. 3 個 class 中的各個 Function 細節功能的學習	
	5. Regression test	

【图 1】

第二个图表是详细说明预期产生效益的状况

The Monkey runner study plan of 2013		
Import Tasks	The current Status	The plan of 2013
Boundary test	1. SMS*MMS容量满时基本功能操作 2. Email容量满时基本功能操作 3. 联系人容量满时基本功能操作 4. Caledar容量满时基本功能操作 完成Boundary test中50%的操作	a. 完成目前所有脚本的模块化 b. 已经有计划2013年完成的部分： 1. Alarm添加全满时基本功能操作 2. callog全满时基本功能操作 3. camera拍照全满时基本功能操作 4. Camcoder 摄像全满时基本功能操作 c. 需要进一步规划将所有可以进行 Boundary测试的功能列出，这个需要Wendy制定计划，并安排大家共同完成。
Pressure test	1. 连续拨打与接听电话1000通以上 2. 连续发送与接收SMS&MMS 1000笔以上 3. 连续发送与接收Email 1000笔以上 4. 连续拍照和摄像1000次以上 5. 连续添加PHB资料1000次以上完成了pressure test中的40%的部分	a. 完成目前所有脚本的模块化 b. 已经规划的计划2013年完成的部分： 1. 连续添加Alarm1000笔以上 2. 连续播放音乐1000次以上 3. 连续切换网络1000次以上 4. 连续播放音频1000次以上 5. 连续编辑图片1000次以上 6. 连续开启和关闭飞航模式1000次以上 c. 需要进一步规划将所有可以进行压力测试的功能列出，这个需要Wendy制定计划，并安排大家共同完成。
Normal test	0%(目前还未实现此任务)	a. 所有脚本需要通过模块化的方式实现 b. 计划如下： 1. Mary和Wendy根据3个方向审核我们目前的测试用例，将较机械可以通过自动化脚本测试的内容提炼出来，并制定计划 2. 将Senior engineer 与Junior engineer通过自愿的方式成为各个小组，初步商议分为："多媒体小组"，"通讯小组"，"其它小组"，每个小组推选一个小组长，然后按照Mary和Wendy制定的计划完成自动化脚本的产出

【图 2】

有了“承诺”和“schedule”这“双剑合璧”终于盼望已久的经费批下来了，女巫眼角湿润了，耳边响起了刘若英的“给十五岁的自己”这首歌中的一句歌词：“我们都不要放弃，都别说灰心，永远听从刻在心中那些声音……”

事实证明对于“地位不高”的测试部门，申请学习经费确实是很不容易，但是并不是完全不可以，永远不要放弃，尤其在你没有做完所有你能做的事情前坚决不能轻言放弃！还有要以老板的思维考虑问题，老板不在乎你想学什么，能学什么，在乎的是你学的东西对公司能产生什么效益？且什么时候能产生这些效益？如果产生效益的时间太

长，老板是没什么耐心慢慢等的。无论是写报告还是面谈都要仅仅围绕上面两个问题，你就会发现原本很难沟通的老板会变得容易沟通很多~

2、跟随老师后面当个有想法的小白兔

经费有了，找到了老师，老师是与我年龄差不多的一位湖北帅哥，我们对于自动化并不是没有任何基础，所以我和老师协商，达成以下共识:

（不得不说，这位湖北帅哥被我烦得够呛，他后续告诉我，没有一家被培训的公司有这么多要求的^_^）

1) 我们的费用一共可以上四次课，分四次上，每两次之间的间隔是 1 个月，这一个月留给我们消化第一次课的内容，并找出疑问在下次上课时请老师给予解决。

2) 每次上课前我会发出课程需求，希望他根据我的需求，给我们定制化课程，并在上课两周前将课程内容发给我确认。

3) 我们会利用上课前两周时间先自学老师发过来的课程内容，找出问题点，在上课时每个学员带着问题听课，我们会在上课前 1 周将这些问题发给老师，我们的目标是上课时必须把这些问题解决。

4) 老师需要给我们一个优质框架的代码 Demo，上课的时候需要讲解。我们也会在课后根据老师框架搭建属于自己的自动化优质框架，搭建框架期间遇到的问题，需要记录下来，优先邮件与老师沟通，如果邮件说不清楚，则下次上课老师需要当面讲解。

5) 每次上课结束后，我们会汇整上课的问题，发给老师，麻烦老师给我们解答。同上一步，如邮件可以解答的，就邮件解答，如邮件无法解答清楚的，则下次课程需要当面讲解

呵呵我真的把能想到的都要求了，上课期间，我请老师吃午餐，老师向我感叹说:“你们真的太认真了，我经常给那些很大公司的员工上课，每个人都是一副很牛很厉害的样子，而且培训时间都在工作时间，即使这样他们也不认真听，也不提任何问题，总是用着一副:“您谁啊？”的态度斜眼看着培训老师，但是同时我上课也轻松，轻轻松松培训费就挣到了，可是你们呢？上课不能再工作时间，必须要在周末，还一个个这样认真，我哪敢怠慢？”所以我明白了不要埋怨培训老师不认真，首先被培训这的态度要非常端正，要给出非常认真的态度，制定详细的学习计划，培训才能达到你想要的效果。所以我也感谢申请经费的曲折过程，如果非常容易就能申请到了，我也不会如此珍

惜每次培训。

我们一共找老师培训 4 次课程，共花费大约 5W 人民币。在最后一次课程结束后，老师坦诚得告诉我，其实你们主动性这样强，后续可以自学，因为 Python 是开源的，完全可以自学，而且他已经把他知道的都讲给我们听了，后续即使我们能申请经费他也没什么可教给我们的了~

我目前反思这 5W 人民币，我们学了什么，其实就两方面的知识:

1) Python 两个模块系统的学习即 MonkeyRunner 和 Selenium，

2) 如何搭建一个比较优质的可持续添加代码的框架。

其实最关键的是框架，这个知识在网上是无法找到的，对于模块实际上在官网上都可以找到相关的资料，如果想通过 Python 尽可能多得实现自动化，就要学习更多的模块，这些模块的信息在网上都可以找到，所以我们完全可以自学，好吧，那我们慢慢开启了“自学模式”。等开启了自学模式我们慢慢发现，其实靠自己能做的事情真的很多~~

3、开启自学模式的小白兔

OK 小白兔开始了摆脱培训老师的进程，同时真心感谢培训老师的提醒（估计也是被我烦到实在受不了^_^，问题太多，赚我们的钱太不容易了）

为了介绍我们的自学的方法，我随机选择一个模块 Pywinauto 为例介绍

我们的自学方法，希望可以给大家一些启迪^_^。

1) 确定我们的需求

首先我们要确认我们的需求是什么：我们的需求是可以控制运行在 Windows 系统中的应用程序，例如测试工具之类的。

2) 在网上寻找资料

写出关键字 Python+空格+你的需求，就能找到相关的 Pywinauto 的相关介绍，建议大家一定不要怕麻烦，一定要看官网，虽然全是英文，但是官网的逻辑性比较强，且内容比较全面，比在网上找那些片断式的知识点，学习效率高的多，所以不要被一大堆英文吓住，想起周星星的一句台词：“你吐啊吐啊也就习惯了^_^”所以啊，想搞好自动化，英语的底子还是很重要的，谁让那些官网都是英文呢？根据女巫的经验：官网基本

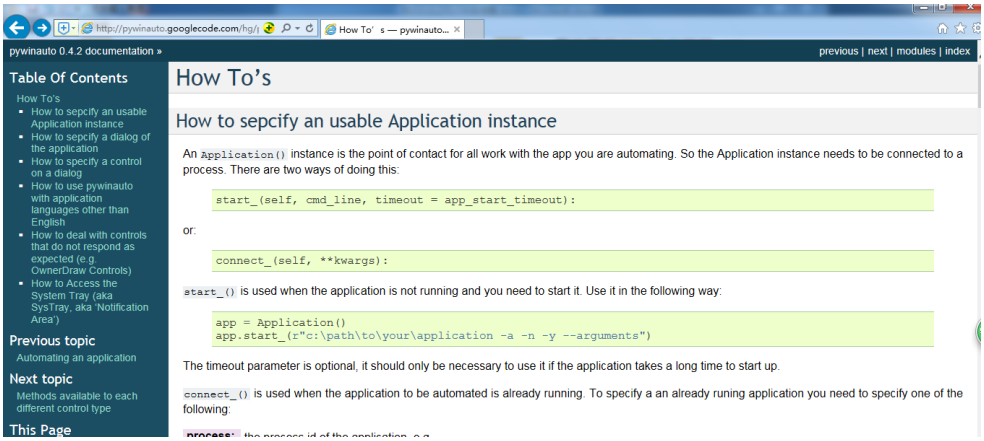
可以满足用户的各种需求，Pywinauto 的官网请参看下面的链接:

http://pywinauto.googlecode.com/hg/pywinauto/docs/

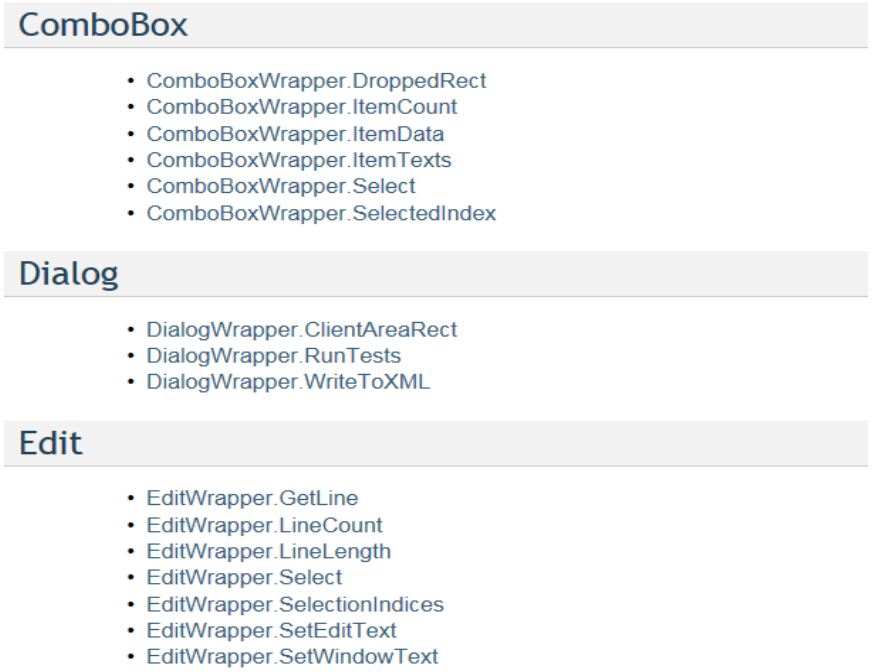
如下【图 3】和【图 4】

【图 3】说明如何学习 Pywinauto，即学习 Pywinauto 的逻辑顺序是什么。根据官网推荐的方式学习，比自己像个无头苍蝇到处乱撞好太多了~

【图 4】说明如何控制各个控件，即控制各个控件的函数。



【图 3】



【图 4】

3) 找到学习的资料后，一定要注意，不要妄想把所有的函数都学过一遍，一定要先

确认自己的需求，然后学习那些能满足自己的需求函数即可。

4) 最后一步非常重要，一定要实践，实践再实践，编程这个东东就是实践科学，看10000 遍不如自己动手试一遍！

4、已学习内容的总结以及学习新知识的规划

根据“自学模式”中第一步到第四步，我们从 2013 年以来根据自己工作中的需求整理了学习的成果，请看【图 5】,【图 6】,【图 7】，分成三部分如下:（为了满足工作中各种各样的需求，就需要学习各种各样的 Module，所以除了安装和基本语法以外，我们主要精力都是放在学习各种各样的 Module 上）

1) Python 的安装

2) Python 的语法

3) 各个 Module 的介绍：其中黄色底色标出的是，我们新的需求，等待学习中，这些新的需求已经归为某一个 Module 了，需要实现这些新需求，就要进一步学习这些 Module 中的类或者函数。

a. String&Random（自动化测试有随机产生字串以及处于字串的需求时需要这两个 Module）;

b. Unittest（需要搭建单元测试框架时，用到此 Module）;

c. PyMouse&Pykeyboard（需要控制鼠标和键盘时，用到此 module）;

d. OS and SYS（需要调用 DOS 界面，且在 DOS 界面输入相关命令时，用到此 Module）;

e. Selenium（需要对浏览器实现自动化时，用到此 Module）;

f. Pywinauto（需要对运行在 Window 系统上的应用软件进行自动化控制时，用到此 Module）

g. wxPython/BOA（需要搭建 UI 界面时，用到 wxPython，如需要使用 wxPython 图形用户界面的构建器，即可视化的框架建立，就可以使用 BOA，当然因为 BOA 在 2007 年 7 月 5 号就停止了更新，所以 Bug 是相当的多，但是并不会阻碍其功能的使用。）

h. Thread（如果需要用到多线程处理，或者需要暂停，继续，终止，结束线程，就

可以使用这个）

Knowledges	Submenu	Documents	Document Address	Tools	Tools Address	Notes
Python Install		None	None	python-2.7.8.msi	[redacted]	Python安装完毕后要按照左图所示环境变量中加入Python安装的路径
Python Syntax Study	Basic syntaz	Python syntax Introduction	[redacted]	None	None	None
	HTML	HTML Foundation	同上	None	None	None
Module Study	String &Random	Python技术_random&string	[redacted] study	None	None	None
	Unittest	Unit test Study SOP	同上	None	None	None
		单元测试的框架和自动产生report的方法	同上	None	None	None
		单元测试运行方式总结	同上	None	None	None
	Pymouse &Pykeyboard	Pymouse基础知识学习	同上	pyHOOK	[redacted] y\Auto [redacted]	None
		Pykeyboard基础知识学习	同上	pywin32	同上	None
		获得windows中应用程序中的特定参数	同上	None	None	None

【图 5】

OS and SYS类	DOS命令获得结果打印到文本中	[redacted]	None	None	None
	Command Automation	同上	None	None	None
	SDK Automation				
Selenium	Selenium study	[redacted]	None	None	None
	selenium+Python建立环境和录制脚本	同上	setuptools for 2.7	[redacted]	None
			pip 1.3.1	同上	None
	Web Driver API study	同上	None	None	None
	单元测试的框架和自动产生report方法	同上	None	None	None
	Selenium验证测试结果的方法	同上	None	None	None
	selenium配置Firefox profile	同上	None	None	None
	单元测试运行方式总结	同上	None	None	None
	Time Function Test SOP(字符串处理方式)	同上	None	None	None

【图 6】

wxPython/BOA	one frame invoke another frame	Ongoing	Ongoing	Ongoing	Ongoing
	API_MenuBar 如何通过menubar调用另一个frame				
	API_Button				
	API_ListBox （新增string,设置鼠标位置，得到listbox中有多少数值，删除一笔，插入一笔，reset listbox等）				
	API_CheckBox				
	API_Combox				
	API_Static text				
	如何将listbox中的值导出到txt中				
	如何将txt的值导到listbox中				
	各个控件的参数如何加到注册表中				
Thread	Python thread introduction	[redacted]	None	None	None
	如何解决暂停当前执行的thread,暂停后还可以继续执行当前的thread				
	如何解决退出当前执行的thread				

【图 7】

整理学习内容的方法也很重要，如【图 5】【图 6】【图 7】

第一、对于学习的内容进行第一步分类：即分成 Python 的安装，Python 的语法和 Module Study。如【图 5】第一列 Knowledge。

第二、然后对于第一步分类进行细分，例如 Module 可根据 Module 名称分成细分。如【图 5】第二列 Submenu。

第三、根据你的需求挑选 Module 中的 Class 或者函数，来实现你的需求。如【图 5】第三，第四列 document 和 document Address。

当然这里所说的 document 是你的学习成果，即自学并实践成功后自己总结的文档，这里请放文档所在的路径位置，后续复习时方便查找。

第四、对于一些 Module 需要另外安装第三方软件，需要将这些软件也要搜集起来，并放到相应的路径位置下，后续复习时方便查找。如【图 5】中第五，第六列 Tools 和 Tools address

三、总结

这一次总结了 Python 的学习历程，以及如何总结我们已经学习的内容，和如何规划后续工作中会用到的新知识的学习。Python 的学习历程从申请经费的曲折过程，总结出来了对于“需要花钱”的申请，该如何与老板沟通的方法。对于 Python 的培训，也总结出如何能组织一场真正有效果的培训的几大步骤；接着，大家注意了，接下来就是非常重要的“开启自学模式”，这里总结的步骤虽然不多，但都是精华，我们根据这些步骤真的学了 11 个新模块，要知道，亲们，我们可是花了 5W 人民币才学了两个模块啊，所以啊，自学才是硬道理，找出适合自学的方法是最最重要的。最后就是到了金秋十月收获的季节了！我们累积了 2-3 年的学习内容，就可以对我们的学习进行分类整理，后续如果我们忘记了学习过的内容，通过我们整理的表格可以很快找到相关的资料和工具，更重要的是，我们能掌握我们目前学习知识的脉络；最后要对以后的学习进程做出规划，首先要写出我们的需求是什么，再将我们的需求与 Python 的 Module 做一个对应关系，这样就对后续的学习内容非常明确了。

参考文献:

有关 Pywinauto 的官方的网站，其中有学习的逻辑以及相关的类和函数:
http://pywinauto.googlecode.com/hg/pywinauto/docs/controls_overview.html

Loadrunner 学习笔记（一）

◆ 作者：脚踏实地

一、LoadRunner 11 安装

1. 文件准备

（1） 下载镜像文件读取工具 Daemon_Tools;

（2） 下载 LoadRunner 镜像文件 LoadRunner-11.iso;

（3） 下载两个 LoadRunner_dll 文件，分别是：lm70.dll 和 mlr5lprg.dll;

（4） 下载注册表清除工具 lr_Del_license(regedit).exe。

2. 安装读取镜像文件的工具

点击 Daemon_Tools 工具的 exe 程序进行安装，本人使用的是 Daemon_Tools_5.01.407_2，具体版本及安装路径根据需要进行选择。

3. 安装 LoadRunner 11 的镜像文件

进入 Daemon_Tools，从 Daemon_Tools 工具中找到 LoadRunner-11.iso 的所在路径进行添加。打开镜像文件，进入安装向导，点击 LoadRunner 完整安装程序按照提示进行操作。

二、破解 LoadRunner

一般下载安装的 LoadRunner 只有十天的试用期，需要我们替换新的证书，让软件能长期使用。

1、替换 dll 文件

“lm70.dll”，“mlr5lprg.dll”这两个文件复制并粘贴到 LR11 安装目录下的 bin 文件夹下，替换掉原有的两个文件。

2、删除注册表

1）启动 LoadRunner，从 configuration 中选择相应的证书“LoadRunner license”，打开会发现有试用的证书;

2）选择“New License”，添加新证书，输入“AEAMAUIK-YAFEKEKJJKEEA-BCJGI”，会出现弹框说已有证书，且试用期为 10 天;

3）关闭 LoadRunner，运行 lr_Del_license(regedit).exe，使用注册表清除工具将已有证书删除;

3、添加新的证书

1）重启 LoadRunner，此时注册表是空的，说明已有证书已删除;

2）选择“New License”，首先输入 globa-100 的注册码：AEAMAUIK-YAFEKEKJJKEEA-BCJGI，继续点击“New License”，输入 web-10000 的注册码：AEABEXFR-YTIEKEKJJMFKEKEKWBRAUNQJU-KBYGB。

3）重启软件，破解工作结束，软件能长期使用。

三、安装及破解注意事项

1、替换 dll 文件时需要将 LoadRunner 关闭之后进行替换，否则会出现复制出错;

2、先替换 dll 文件再删除注册表，颠倒顺序容易出错;

3、运行注册表删除程序时，必须先关闭 LoadRunner;

4、在带有 IE7、IE8 的系统内安装 LoadRunner，IE8 以上浏览器需要一些工具才能兼容，方便往后的录制学习。

四、录制脚本

1、录制前准备:

1）开启 start web sever，方式开始-所有程序-HP LoadRunner-samples-web-start web sever

2）将 IE 浏览器设置成默认浏览器;

3）确认 IE 浏览器版本为 IE7 或者 IE8，LoadRunner 只对 IE8 以下的浏览器兼容。

2、录制过程

1）启动 LoadRunner，点击 Create/Edit Scripts，点击“+”进行新脚本创建;

2）根据具体需要选择协议，一般选择的是 Web（HTTP/HTML）协议，点击 create;

3）根据选择应用类型，IE 浏览器的安装路径一般是默认的（如果有改动需要添加相应的正确路径），以及 sample 地址的路径 http://127.0.0.1:1080/WebTours/;

4）设置 options 选项，一般脚本选择 Recording 下的基于 HTML 下的 URL 模式;

5）点击 ok，开始进行录制，此时 events 数一直在不断递增。

3、录制结束

录制完成后，点击结束按钮，此时会生成相应的脚本，脚本就录制成功了。

五、注意事项

1、录制过程中，若 events 数始终为 0，说明脚本没录制成功，需要检查以下几个方面:

（1）是否开启服务;

（2）是否是兼容的 IE 浏览器;

（3）是否选择的是正确的 IE 浏览器的安装路径;

（4）是否将 IE 浏览器设置成默认浏览器;

（5）是否设置了正确的 samples 页面的 URL。

2、录制脚本的过程最好有针对性，比如注册过程、登录过程等等，其他不相关的动作最好不要录制，这样方便我们理解脚本。

3、考虑脚本的简洁性，可以将脚本当中的停顿时间进行删除，不会影响脚本的回放。

4、选择的基于 HTML 模式下的 URL 模式是自由度相对较高，几乎无耦合，若采用别的方式，代码会相对繁琐，建议使用此模式。

移动开发和测试如何影响其余行业

◆译者：枫 叶

我们付出更多的工作在某个事物上，我们将会得到更好的。随着应用开发和测试逐年增长，我们能够从网络中学到每一个东西，并将它运动到下一个技术波。我们如何在网络事物上测试和开发，更重要的是，移动设备，在它到来之前的被获知了。

测试和开发们在手机出现之前的更传统时期学到的不同的技术和技能如今仍然起了一个重要的作用，但是不是去简单地看过去如何预告未来，重要的是观察未来如何帮助我们在很多东西出来前更好地测试和开发。

所以，我们如何从移动开发课学到的并把它们改进到其他行业，比如网页？普罗拉普.登迪，电子云开发公司的副总裁，证明我们所学的应用到的主要有三课。

而且它都始于用户体验。

没有一样能比移动应用的用户体验重要性和敏捷的时间价值更人性化。登迪告诉顽固主义者。这些天来你们得到的，正确地，40 秒或者其他，作为行业基准，从某人下载并打开你的移动应用那时起，它有些儿得到了价值，最初的价值，出现的，并且因此，用户体验是最重要的。我们曾看到，很快速地，网页和其他应用开发者观察到的他们设计的用户体验的重要性。”

第二课围绕着基于云服务的应用。因为移动的速度和使这些时期越快完成越好的需求，基于云的服务优先于安装软件或预付软件。这种云第一的心态已经被移植到其他领域，比如企业市场和其他非移动应用。

最后，可能是最显著的一点，是速度。人人正趋向敏捷，移动成为在敏捷市场的大玩家。

“移动，被定义为，最短的发布循环，人类自然地使用敏捷技术。我们已经看到，甚至那些落后在企业的，很快地移向自适应敏捷，并且开发错误程序今天，”登迪解释。“那三件事...移动是第一个跳向他们的。我们现在正看到其余行业也开始去适应。”

那种采纳只能支持开发和测试在非移动项目能做的。在这里“如果它不破裂，别固定它”会将容易被接受，但是如果我们不期待新行业里什么在起作用，不久后旧事物将被尘封。

测试提供给软件的价值是什么？

◆ 译者：枫 叶

首先它看起来像是一个很简单的问题，对吗？测试员，当然，测试并评估提供给他们的代码——找出错误，衡量质量，甚至评估风险。测试是一个被实践的真实的过程，我们利用它为了想要得到一种生活，但是因为我们继续扩大设备和我们正测试的应用程序的范围和复杂度，实际测试角色将改变多少？

它是詹森.阿尔邦，AppDiff.com 的首席执行官，向测试团队提出的一个问题。他阐述测试不是关于平台。它不是一件你是否正在笔记本，网页，手机上，或利用周围的计算处理的事情。

对于阿尔邦，我们第一眼瞥见看起来仍然有一些糊涂的问题：我们如何量化质量？

“我认为是时候收回脚步并离开，将我们的最佳实践在从一个平台搬到下一个到下一个，”阿尔邦对顽固思想作解释。“我们逐渐地用 Appium 看到，对吗？事实上考虑什么是我们要完成的最基础的活动。测试是什么？我们如何量化质量？我们为什么测试？那些是我觉得在我们的行业里依然是未被解释的，反过来说。”

很难不被所有散布着的围绕新的设备或者技术清扫掉。有如此多要测试的新事物而不是加强核心的测试能力，我们趋向于刚适应已经知道的以至于它为其他不相近的平台工作。

就像阿尔邦说的，我们正取得以前工作过的不同事物，为一个新电话，新写字板，不同的智能手表，或者现在连接到因特网的家庭设备而自然向上调谐它们。那是一个短期的解决方法吗？当然。如果我们继续更新坚持的模式，我们会曾看到基础的测试问题被解答和固有的形态被改进吗？可能不会。

假想我们测试所有能想到的那是有点独断。当我们继续被科技进化，测试能做的最

重要的一件事是往后退一步，呼吸，然后发现测试的价值——同时更好的方法去测试而不是过去我们在其他平台上所工作的。

“那是我想让人们关注的，”阿尔邦说。“它确实不关于手机，网页或是桌面程序，一些特定平台事物。它是关于我们作为测试者想到的，我们对软件做了什么价值？”

软件测试适合谁做？

◆作者：Linda

在我本人想继续写完这篇文章之前我们来讨论一个问题？

软件测试适合什么样的人干？

回答各式各样:

A: 男生

B: 女生

C: 有耐性不足的人，沟通能力不好，抗打击能力不强的，逻辑太混乱，没有思路的人，比较懒的人，学习能力不强而又不积极主动的人

D: 认真，负责，仔细，有恒心，能加班等等...

那么究竟是什么样的人适合呢，软件测试到底是不是只是女生的专利，带着这些疑问我们进入到软件测试的真实世界中，来看下什么是软件测试:

我们定义它是帮助识别开发完成（中间或最终的版本）的计算机软件（整体或部分）的正确度(correctness)、完全度(completeness)和质量(quality)的软件过程；是SQA(software quality assurance)的重要子域。

在了解到软件测试的真正定义之后我们就会再次回到我们前面的提问上，到底什么人适合做软件测试，软件测试是不是只是女生的专利，软件测试女生的专业发展方向，男生就不适合做软件测试了？

一、软件测试更适合女生做？

那么从生活中经常会有这样的问题抛出，女生更适合做软件测试，因为原因见下或者更多:

（1） 女性做软件测试优势明显

软件测试行业将要改写女性在 IT 行业就业比例“不公平”的历史，软件测试职业

给女性进入 IT 行业提供了更大的便利和更多的机会。

在现在看来，从一定意义上讲，软件测试这一职业对从业者的特性有着特殊的要求。比如做软件测试的人需要有耐性、心细、敏感、逆向、设问、怀疑、举证、韧性、安静等特征，在以上特性上，耐性心细、韧性、敏感、安静等特征要求与女性的生理个性气质比较吻合，所以从这个角度来看，女性从事软件测试工作似乎有着得天独厚的优势。

目前在做软件开发工作时，不难发现我们周边基本上是清一色的男性，工作氛围更显得“严峻”。转到软件测试岗位后，或者现在入职一批女生作为软件测试人员，那么在均衡的性别比例使得工作压力也缓释了不少，“男女搭配，工作不累”，这也是软测岗位的特色。

（2） 企业倾向录用女性测试人才

我们在做了那么多年软件测试之后，不难发现一个现象就是目前参加软件测试培训的学员中，或工作现场的情况来看，软件行业男性与女性的比例已接近了 10 比 2。这的确是个有趣的现象，据统计，很多测试培训学校的女生出来找工作也不怎么费事，录用了女学员的企业还表示，只要测试技术好，他们更愿意录用女性测试人才。

软件测试是一份要求细致与耐心以及善于表达和沟通的“精细活儿”，软件测试不仅男性可以胜任，女性也是非常符合这一工作要求的。

另外，相对于软件开发来说，做软件测试更适合“女性选择工作强调稳定性好、风险小、不过度劳累”等传统观念。现在我们很多人认为 IT 行业绝对还是一个对我们的就业有着巨大吸引力的行业，不管什么专业，毕业时大家都希望能进入到这个行业中去。可是这个行业从就业开始就存在着激烈的竞争，如果你没有一技之长，别说在这个行业站住脚，就是想踏进这个行业都不是件容易的事。通过参加一些相关技能的培训，先保证自己能顺利进入到 IT 这个行业中，再通过不断地学习提升自己，争取更好的发展空间，这也是我选择软件测试职业培训的最核心的原因。

（3） 软件测试行业成女性就业首选

日前市场分析指出，性别歧视普遍存在于职场中和很多各种类型的公司，主要原因在于，很多职业存在着较大工作压力与工作强度，加之很多岗位存在临时加班、出差等突发情况，女性职员难以承受，所以很多企业对女性职员往往敬而远之。其次，一些工

作对体力要求比较高，男生明显比女生有优势。IT 是一个男性主导的行业，女性很难迈进这个圈子。其实，这是一个误区吧。一些软件企业在招聘时并没有这些门槛，尤其是 IT 行业的‘新贵’软件测试，从职业选择的角度讲，是最没有性别偏好的岗位。因为软测主要负责对软件质量进行度量，对求职者的耐心、细心、质疑、沟通等素质能力的要求较高，而这些要求又恰与女性自身的个性气质相吻合，因此，女性从事软件测试往往具有得天独厚的优势。

本人从 2008 年毕业到现在观察到一个很明显的问题，现在公司整个测试部门，很尴尬的一个问题就是在一个办公室软件测试的男性比例与女性的相比总的来说和之前软件开发行业的男女比例是刚好相反的，所以会存在一个有趣的问题就是公司成员办公室恋情会相对比较多。女性普遍具有温柔、细腻的性格，感知能力较强，形象记忆较好，想象力较为丰富，沟通能力较强。这些特点使她们易与环境融合，讲究处理问题的方式方法，在工作中具有较多优势。很多人可能会认为 IT 是男性主导的行业，女性很难迈进这个圈子，其实这是一个误区。“软件企业看中的不是性别而是能力，尤其像软件测试这种没有性别偏好的岗位。”测试是软件质量的把关人，是使软件符合客户使用习惯，女性耐心、细心的特点符合测试工作的要求，较好的沟通能力使其更易于和客户协调，从事软件测试具有得天独厚的优势。另外，测试行业重视积累，随着从业经验增长，个人发展也可获得较大的空间，“从测试员到架构师、客户经理，只是经验多少的问题。这个行业是越‘老’越吃香。”

职业寿命与职业生涯发展，特别是与长期发展直接相关，是职场人应该认真考虑的问题。对女性而言，就业有劣势，就要懂得回避，应当充分认识个人优势，做好长期职业规划，选择呈上升趋势、热中有稳的职业，及时培养相应的技能，随着职业发展使自己成为行业专家。如果感到职业寿命受到威胁，就应考虑未来的发展方向，使自己顺利完成转变。

也有人说，对大多数女孩做软件测试会比干软件开发好。软件开发太辛苦了，不适合女孩子，要干好软件测试或是通过软件测试面试，至少要具备计算机知识和技能，以及软件测试思维能力。如果还具备简单的编程能力，则求职把握更大，好工作机会更多。

女生更受企业欢迎。有人这样戏剧化的比喻软件测试领域，“如果说软件开发行业是男子单打，那么软件测试行业就是混合双打。”软件测试从业人员往往更偏好认真、

细心、耐心、敏感等.

二、男性做软件测试怎样?

作为软件这个这个行业来说如果你想进入这行，你一定要考虑清楚，你不是择业而是定向。软件测试人员一般要细心认真有耐心、有责任心、有团队精神、时时保持怀疑态度、具有一定编程经验、沟通能力、有上进心。

当然，男生也可以做测试，而且也有不少优点，测试的高级人才应该男生居多吧。

做软件测试的男生不多，但是比较吃香。因为男生做测试，可以出差，可以加班。女孩子如果加班太晚，一是体力问题，二是安全问题。所以，很多单位还是很喜欢招聘男生做测试。现在男生做测试的待遇也比同样经验和能力的女孩要高一些。当然不排除个别人员。测试的强度一般要比开发低一些，所以，待遇也有一定的差距。因为测试和开发所面对的情况不一致，所以，需要的只是也不太一样，不存在知识面宽的问题。

软件测试适合男生去学吗？在 IT 业，竞争异常激烈，人们每天要面对大量不同工作压力，尤其是软件开发工作，需要很强的逻辑推理能力，在高强度的工作压力下，更是对人们脑力、体力的双项考验，因此，许多用人单位对于这一职位的招聘更偏向于男性。而软件测试工程师相比之下，工作的压力不是太大，不需要天天去思维创造，工作重点是在注重经验的积累上面，同时，软件测试工程师需要调节软件开发人员、项目经理和自身的关系，因为是在找软件编程人员的漏洞，所以男生当然适合学习。

软件开发多运用正向思维的方式，而软件测试工作则要根据原始需求、验证编码，在思维上更多使用逆向方式，根据已知的方向进行发挥，从中找出并预见到软件开发中的不完善之处。男生比较稳重，抗压能力也比较强，所以做软件测试当然可以。

根据软件测试岗位工作的实际要求设计而成，以实际应用场景为核心，配以实际

测试项目和测试工作流程，注重学习的系统性、教学的渐进性及学员的参与性，使学员能够用最少的时间掌握测试工作中最实用的必备职业技能，具备测试岗位需求的工作经验和综合素质，从而具备较强的竞争力。我国软件企业人才结构将日趋合理，这无疑有利于我国软件行业整体品质的进一步提升。当前，我国软件测试人才正处于一个“双高”地位，即地位高、待遇高，职场前景非常广阔，因而，近两年来，软件测试工程师也成为了 IT 行业最新的亮点。

但由于女性在职场上的非持久性（如生儿育女、照顾家庭等）以及劳动强度承受能

力不及男性，她们的职业寿命也面临着较大威胁，这也是女性择业时更关注职业寿命、工作压力等因素的原因，所以很多大型公司在往上发展过程中男性作为软件测试人员的管理

三、综合阐述:

说女生更容易是指的在人们的观念当中，都是感觉女生更仔细一些，当然在做黑盒测试的时候这样发现的问题也会多一些，其实测试一仅仅要细致的人才能做好，要关联很多方面，从个人自身素质方面就要有很强的分析能力，逻辑思维能力，可扩充性的思考能力。

在自动化测试中要有自动化测试脚本的编写能力，在性能测试中要有编写性能测试脚本的能力，语言编程的能力就在测试中越来越扮演着重要的一面。现代的测试男女都一样做，也有很多男生比较仔细的。不管是男女，都要从综合素质考虑这个人是否适合做测试这个职业！

敏捷测试自动化

◆ 译者：于 芳

一、概要

这篇文章描述了一种适合中小型软件项目的敏捷自动化测试方法。考虑下创立一个小的测试工具团队，他们会将敏捷开发规范应用到测试自动化问题中。这样做意味着，他们会每天跟测试人员比对，见证测试策略的展开，然后发现方法去应用丰富的技术（很多是便宜或者开源的技术）到特定测试人员需要的特定事上。他们会在可衡量的，易维护的用不到一周甚至更少工夫就可以完成的迷你项目上实施这个做法。这些工匠们也将会跟开发一起共事来确保产品有合理的可测试特性。他们将把技术应用到一系列测试需求上，不仅仅是测试自动化。

二、测试自动化的原则

在考虑如何采用系统化和有效率的测试自动化方法时，这或许能帮助将以下的原则牢记在头脑里。他们在过去 15 年里被从很多测试自动化工程师和经理们的经验中剔除。其中有一些我曾经在我的一篇文章--测试自动化蜗牛油中详细讲过，在我的书《从软件测试中学到的东西》里也谈过。

1) 测试自动化不能复制手工测试人员。

手工测试人员，即使是那些没有特殊技术和培训的人，也能够去做和发现自动化测试能做或发现的想得到的事情。即使有其局限性，也要承认自动化有很大的价值。但是，通常认为自动化作为手工测试的延伸更有效率，而不是取代手工测试。有效率的自动化努力因此从有效率的思考测试开始。缺席的好的测试策略，自动化自然成为很多极少发现缺陷的重复性活动。

2）测试自动化不只是测试执行。

大多数人听到测试自动化立即会有一种想法“测试在我们睡觉的时候进行”换句话说，他们想让计算机来跑测试用例。这的确是一种有用的测试自动化。但是，还是更多的东西。举例来说，下面的清单中的每个都可以在某种程度上自动化，即使其他项目保持手工流程。测试产生（数据和脚本产生器）。工具可能会创建特定的数据例如随机的邮件信息，或者构成数据库，或者产生我们想要去覆盖测试的参数组合。

- 系统配置。工具可能保留或者复制系统参数，将系统设置为一个特定状态，或者创建或者还原“复原的”硬盘驱动。
- 模拟器。工具可能模拟次级系统或还不能用来测试的环境条件，或者手头上去提供真实的环境比较贵的情形。
- 测试执行（道具和测试脚本）。工具可能自行运行软件，或者是通过 GUI 模拟用户工作，或者通过绕过 GUI 使用替换的可测试的接口。
- 侦探器。工具可能能使对人类来说不可见的东西变成为可见。他们可能统计性地分析一个产品，从语法上分析一个日志文件或者监控系统参数。
- 神谕。一个神谕或者看到的东西是任何一种我们发现失败或成功的机制。工具可能自动探测到某种产品中的故障情形。
- 活动录制和覆盖分析。工具可能监视测试活动当他发生且回顾性报道什么在测，什么没有在测。他们可能记录下动作为以后其他测试做回放。
- 测试管理。工具可能跟测试结果打交道，组织测试思想，或者呈现度量方式。

3）测试自动化易于即时作废。

软件项目围绕生产代码做解析。测试代码并不是生产代码。因此一个典型的软件项目的优先级允许生产代码来改变及时当这种行为会破坏测试代码。这是正常的现象，而通常来说是合理且经济实惠的做法。

设想产品会变化，你基本上有两个选项：投资测试代码让他更坚固地面对这种变化，或者让测试代码不那么昂贵和不易修复如果你不在意产品变化会破坏测试代码的话。

4）测试工具有很多且变化多样。

大多数人，有其是经理们，认为测试工具是市场所卖的“测试工具”。他们倾向于

很昂贵。但是，事实上，几乎任何东西都可以作为测试工具，而且很多为其他用途售卖的工具是有其有益于测试的。有些工具是免费的，有些是提供在库里给开发者用的。

我使用的资源包括:

- MSDN 统一库
- 任何微软的开发工具（他们常常包含一些有用的工具）
- 微软兼容性工具箱和其他免费工具（www.microsoft.com）
- 基于 Web 的测试资源（HTML 检查工具，可访问性分析工具）
- Windows 资源箱（在微软网站上可以找到）
- 脚本语言（例如，Perl，Ruby，TCL）和相关联的库。
- 共享件库（www.download.com）
- 操作系统监控工具（www.sysinternals.com）。
- 开源测试件（www.opensourcetesting.org）
- 监控探索性测试的侦探器（www.spectorsoft.com）
- 靠近门的小室（经常是领先几步的人已经创建了必要的工具）。

5）测试自动化依靠于产品的可测试性

有些测试自动化种类只有在被测试的产品被设计为便利测试或者至少不被设计为特别难测的情况下才具成本效益。

可测性的两个大支柱是可控性和可观察性。GUI 相比较于命令行，批处理或者编程接口来说不那么好测，因为后者提供更大的可控性。类似地，可观察性当日志文件，侦探器或者可查询的数据库能够通过 GUI 增加信息可见度时增加。

6）测试自动化可以使你从好的测试中分神。

有时候技术对测试自动化的关注可以导致一种自动化者跟软件测试任务切断，生产很多工具和脚本看起来不错但是就对商业来说说得通的连贯性测试策略价值甚微的情形。

三、敏捷测试自动化

在过去十年里，“敏捷开发”的运行已经从模糊晦涩中浮现出来，开始对在美国，有其是商业性软件公司的软件开发的文化产生重要的影响。自从上个世纪 80 年代末，我就参与到这种运动中。我的大多数写作来自于敏捷的视角（看我 1994 年的文章—疯狂世界里的流程演变）。敏捷开发的一种形式是“极限编程”，这是一个 12 个原则的集合，由沃德·库明汉姆和肯特·贝克推动。

敏捷思维的一种表达被包括在内如果有所谓的敏捷宣言的话。在敏捷开发的元素内，是一个以实践者为中心的技术，跟顾客非常紧密接触，无数小的里程碑，和一种对变化欢迎的态度。敏捷开发给管理和顾客在保留技术工作的创新性和效率的同时更多控制开发进程的控制权。敏捷测试自动化是将敏捷开发规范用到测试自动化问题中。

四、敏捷测试自动化原则

我提起这些作为从测试自动化获取强壮和不断发展的结果的运行原则。

- 测试自动化意味着对测试项目的所有方面提供工具支持，不仅仅是测试执行。
- 测试自动化是由测试人员主导的。
- 测试自动化当受到清心贡献的编程者（工匠们）的支持时才会向前进。
- 测试工匠们为可测试性功能和生产能够使用这些功能的工具倡议。
- 测试工匠们收集和使用各种各样的工具来支持测试。
- 测试自动化组织起来实现短期目标。
- 长期的测试自动化任务需要引人注目的商业案例。

在缺乏用心工作的工匠的情况下，这些规范仍然适用，你只需要将他们用到一个业余工匠那里。

“短期”来看，我意思是 40 个小时或更少。长期是指任何超过短期的时间。一个较长期的任务可能在短期增叠时期中完成，只要每个这样的增叠是一件可发送的，有用的工作单元。

“由测试人员主导”我的意思是测试自动化的进度应当以它怎样为测试人员和测试经理解决问题来衡量。测试工匠与一个有问题的且需要帮助的测试“客户”结对。工匠的即时目标是提供对测试来说可接受的帮助。

五、敏捷自动化任务

测试工匠做以下事情:

- 快速响应从测试者发出的帮助请求
- 寻求测试效率问题
- 调查与测试有关的可能的解决方案。
- 应用技术来改善测试流程。
- 倡导产品中的具体可测试功能。
- 研究可用的工具学习怎样使用他们。
- 收集开发人员或测试人员生产的工具。
- 评审将要到来的产品计划来评估自动化的可能性。

六、敏捷自动化团队的组成

最简单的，一个团队可以由一名单独的工匠，与一大群测试人员共住并给他们提供支持。但是如果有很多测试人员的话，会有一大堆积压待办的请求，而且多数错过测试效率提升的机会。

换个角度说，如果有很多自动化人员而只有一个测试人员，那不见得是个问题只要工匠们也做好准备愿意并且能够当做测试人员用（例如在一个特定产品，子系统或功能集中承担主要责任，做一切必要工作来测试）。如果没有足够重要的自动化任务，而测试又需要，那么让工匠去做测试。现在，将工匠转为测试是个糟糕的主意如果你有一个大的自动化项目你期望他们在一个特定的时间框里做。但是如果自动化任务是比较短期聚焦且快速执行的，那就不是问题。

避免那些只坚持做程序员的工匠手，他们在测试方面没有培训或技能经验。自然的问题来了：为什么不把测试团队的每个人变成工匠手？有两个原因：好的工匠手是稀少昂贵的商品，而测试团队从丰富多样的技能和背景中获益而不仅仅是编程背景。

敏捷工匠不是真的跟测试人员分离的，他们是团队的一份子。他们是有着特定技能的测试人员，那些特定技能使得他们更能帮助其他测试人员测试得更快和更好而不是他们自己为测试一个产品承担主要责任。

七、敏捷自动化怎样开展的

测试自动化从以下方面合理化自身:

- 让新的有用的测试种类成为可能
- 让已经完成的有用的测试种类变得更有效率，更快，更可靠
- 或者从别的方面给更成功和更少耗成本的测试项目做贡献

测试自动化是不对那种不需要开始，不值得投资或者在需要时不能发布的测试进行合理自身的。

敏捷自动化根据这个循环进行：理解测试是怎样做的，识别一些在测试人员角度能重大改善测试的某些技术，在一周时间内发布解决方案，重复。这个循环有点简单，因为在现实中，一个发布好的解决方案通常需要在事实后微调和改善。

这个过程可以由 5 个清单管理，加上一些背景人物。这些清单应当是清晰可见的，或许在一个网站上。这些清单如下:

- 请求清单。这是一个从顾客（测试人员）那里得来的新请求。
- 分配清单。这是每一个工匠现在被分配要做的任务的清单。
- 发布清单。这是一个目前由测试团队使用的解决方案的清单。清单上的每个项目应当包含一个简短的描述和有关该解决方案对测试效率的积极影响的声明。
- 维护请求清单。这是一个需要改善的解决方案的清单。它应当分为两个部分:关键维护和改善。
- 障碍清单。这是仍保留未解决的测试中的生产问题清单因为他们需要昂贵的新工具，大量可测性改进或者在短期内不可能完成的更多的任务。

背景任务:

- 跟测试人员结对来理解产品是怎么测试的。
- 评审即将到来的产品说明书和技术来理解技术上的测试问题。
- 跟测试人员和测试经理一样工作来发现合理的评估和汇报效率的方法。

八、测试人员怎样跟工匠共事

工匠维护测试团队的可见性。任何团队里的测试人员，即使他没有亲自跟一名工匠共事，也会看到他们跟团队里的其他人共事。

一个测试人员可能在任何时候向测试自动化组长请求帮助。在实际工作中，自动化组长应当安排定期跟测试人员谈话来核查工匠们可以为测试人员做什么。相替换地，测试经理可能指导特定的测试人员跟自动化团队合作来解决一些问题。

请求也许包括如下的东西:

- 我怎样测试这个新东西?
- 我怎样去看产品内部在发生什么?
- 我怎样知道这个测试成功还是失败了?
- 有没有一种方式我能够自动运行这个操作?
- 有没有一种方式让这个缺陷更加容易重现?
- 帮我调查这个缺陷。
- 这里有个测试我想执行。你能帮我做 1000 个变量吗?
- 我对这个产品的覆盖率做到多好的程度?
- 我想对这个产品做压力测试。有没有什么工具能做到?

做了一个请求，一个测试人员应当期望自动化团队及时响应。测试人员被期望能与工匠共享工作结果。他们作为一对来解决问题。

九、开发人员与工匠怎样共事

开发人员也是测试自动化团队的顾客。当一个开发人员以私人意图创建了一个特殊的工具，这个工具可能也会测试人员有用，那是开发人员的兴趣来把他给工匠部署为一个测试工具。工匠于是开始对那个工具做维护和技术支持工作。

开发人员可能会看到工匠比其他测试人员多。工匠有技术能力学习产品的内部工作方式，用有效率的方式讨论可测试性的功能。他们可以比每个测试人员用一种类似的深度谈话来寻求跟开发人员说话的方式更有效率和更连续地将那些功能抛给测试人员。

十、管理层怎样跟工匠共事

管理者对用最好的办法做事有着强烈的兴趣。测试自动化工作是从测试移除掉的一步，而测试本身就是从收入流移除的一两步，因此自动化必须就他怎样改善了测试来证明自身价值，或者从其他方面改进了业务，例如他值得那个所付的价钱。

管理者跟工匠一起工作主要是通过请求和及时获得能够回答特定标准的问题的报告：

- 我们从测试自动化中获得了什么？
- 他让我们花费了多少钱？
- 考虑大多成本，我们从测试自动化得到的东西足够吗？
- 如果我们暂停对测试自动化的进一步投资会发生什么糟糕的事情？

上面提到的任务清单被设计来部分是为回答这些问题提供一个基础。另外一个测试可能做的事是生产一个平衡的度量板可以精确传达测试效率的有意义的印迹。

这里是一些潜在的度量和其他特定的观察表可以帮助阐明测试效率的问题，因此弄清楚自动化投入的价值。

- **可修复的缺陷发现率**（排除不能重现，测试人员错误，不是缺陷，和其他噪音类别）。很多因素影响这个数字。这是单独使用极为危险的度量。然而，有时候测试上的一个大的改善可以通过这个度量的重要跳跃来证实。
- **发现率钉鞋。**这是由于一种部署的新型测试带来的缺陷发现率里的短期跳跃。这里要看的一个关键因素是，不管发现的缺陷看起来什么样，他们要没有新的测试方法时不会被发现的。
- **修复率。**增加发现的缺陷被修复的比率也许告诉我们一些有关缺陷质量，汇报他们的及时性和开发与测试之间关系的质量的一些事。所有这些可能都会因为测试自动化带来积极影响。
- **测试使得不可能成为可能。**测试自动化使得某些通过手工测试做不到的事成为可能。这种“度量”是真的工匠们做周期性管理评审的清单。管理层必须要问的问题是“这种可能的测试实际上在发生吗？”和“这种可能的测试在产生结果吗，例如增加自信或增加缺陷发现率？”
- **噩梦浮现。**这是一个因为测试自动化的使用应当从整体或部分来避免的严肃问

题的清单。这会包括设计来避免一些早期灾难的重复的测试活动。

- **测试循环时间。**需要多少时间来测试，假设没有发现重大缺陷来收集足够的证据表明产品可以发行？减少这些时间提升整个开发流程的敏捷性。
- **开发/测试循环时间。**在实际工作中，测试是和开发绑定到一起的。我们想要看到对比于在一个新的产品开发上放的员工投入相比整体时间有个减少。
- **管理层的信心。**这不是一个那么能作为度量的因素。测试的目的是给管理者提供他们需要的信息来做合理的商业决策。管理者从测试提供的质量信息上看到改善吗？有时候一个前后对比的比对帮助理清观点。
- **技术支持成本。**最终，评估测试投入的一个关键要素是检查我们先于发布的质量意识的好坏程度与之后的以经验为准的产品体验相比较如何。这是为什么技术支持很重要的原因。设置一个系统来定期跟开发项目沟通那些问题正在被技术支持看到。看看跟顾客数量相关的电话数【一般指服务电话】。看看电话时间可能怎样会通过给予更好的技术支持诊断工具例如测试工具可能生产的工具来减少。
- **顾客保持。**如果质量更好，很有可能，顾客会更开心。
- **名誉。**这可能看起来很稳固，但是并不是如此。名誉通过公司里其他人对测试团队的故事来传达。名誉是由故事驱动的，隐私你需要尽可能频繁地取得可见的，积极的和令人兴奋的胜利。同意，自动化团队和测试团队对请求帮助说越多“是”，他们的名声就越好。

十一、敏捷自动化的风险

自动化可能不会显著改善测试效率除非测试人员知道怎样去测试。

效率度量例如每天创建或执行的测试用例数可能会严重误导，而且会导致在无用的测试上做大笔投资。

测试人员可能不想跟工匠一起工作。敏捷自动化的成员必须是搞笑的顾问：可找得到，可合作，且资源丰富，否则这个系统将很快崩溃。

工匠们可能会提议和发布需要太多与提供的价值相关的在进行中的维护的测试解决方案。

工匠们可能缺乏理解和发布有效解决方案的专业知识。

工匠们可能很成功，他们解决完了重要问题，因此转向不重要的问题。

十二、行动中的敏捷自动化

- **一个测试人员的请求。**一个团队完成了安装测试效率急度改进-300%的在整个团队里的包的吞吐上的提升。这从他们的测试人员开始抱怨将麦金塔电脑做重影像测试时需要的时间开始。测试组长带进来一个咨询的编程人员，之后他跟测试人员一起工作来在两周时间内重新设计流程。这个顾问看了几个可能帮上忙的不通的工具，然后看了看 windows 的边和麦金塔的边。这个团队最终雇佣了一个工具组合，一些是购买的，一些是从仓库里拿来解决问题。这个团队一起工作将解决方案弄出来，但是外面的工匠师才是将他们集到一起的催化剂。值得说明的是该团队考虑使用 Rational Robot(在 window 那边)来自动化整个流程，但是决定它对这项工作是个错误的工具。我采访了测试人员有关这个创新怎样帮助他的日常工作，他表示对这个新系统充满了极度的热情和自豪。这个流程是被集中聚焦且由测试人员的需求做驱动（事实上，是商业的需求）和引入一个程序员和测试人员来帮助工作。

- **安装检查。**我在自己的系统上安装了一个产品。它几乎立即请求许可来更新自己（如期所料）和从网上下载最新的版本。不幸的是，我每次尝试启动他的时候都会连续地崩溃。几次重启后，我终于再次求助于网络重装了它。这个问题才消失。这引出了一个问题：有没有一个产品能允许测试人员来告诉一个产品是否恰当地安装在系统上？在这个案例中，答案是好像没有。好吧，写一个这样的工具也不是什么难事。而这么做会使得对产品的安装测试变得快一点和可靠一点。同时，每当测试人员经理任何一个方面的奇怪的中断失败，他们可以运行安装检查器来确保产品没有被毁坏。调研间歇的缺陷的难度是好的工匠要长期解决的事情。敏捷测试自动化可以提高重要缺陷被及时汇报的概率。

- **数据库卸载工具。**在一次咨询一个项目时，我听说那个产品使用了微软的 Access 数据库后端。我突然想到我可以写一个 Perl 脚本来截下数据库的图，可能那能用作一个测试侦探器。通过从 Perl 食谱借来代码，我将脚本写好，然后一两分钟内跑完。我发现数据库里的某些数据是以二进制形式打包，而我的 Perl ODBC 库不能读二进制数据，但是大部分功能我能轻易访问。我跟我的客户展示了这个脚本，展示了它可以怎样用来在测试中创建一写诊断性的截图，或者可能进入一个测试神谕来自动检测数据库的

测定种类的损坏。

现在，这是一个正当的极好的测试主意，但是仅仅是工匠师的美好幻想直到，除非他被用来测试这个产品。仍然，这展示了敏捷自动化的三个方面：一个好的工匠师有对可能帮助的工具的广泛的知识（像通过 ODBC 对数据库的脚本化的访问），敏捷自动化起始于很简单的解决方案（如一分钟的脚本），有时我们需要开发的支持（如一种解码数据库中某些记录的二进制呈现）。

- **客户测试覆盖率。**在一个涉及到发送邮件的项目，我在看一些手工测试脚本。测试脚本参考了描述各种发送和接受的消息矩阵。我注意到一个数据库卸载工具脚本类似上面提到的可以修改来提供一个说明测试矩阵的那个部分可以被实现的报告。那被称作覆盖率分析工具。覆盖率分析工具帮助将测试人员从测试纸面工作上解放出来，通过自动汇报那些东西被测来实现。那样的话，测试人员和测试经理不用去担心某个功能会意外没测了。

因为测试人员跟我探索了这种可能性，我想到了另外一个主意：可能我们能自动化发送大量信息，从需求上，选择去接收一千个不通的信息。那样导致一个在测试实验室里跟测试消息库一起创建一个消息发送服务器的想法。我们估计了这个可能会在几天之内启动和运行。

这是一个工匠师跟测试人员一起工作怎样产生了一些好主意的例子。也是一个从宽广角度思考测试自动化的利益（覆盖率分析；自动化消息发送的便利工具）而不是仅仅是简单的“测试用例”。

十三、跟每个测试人员十分钟谈话

我坐下来跟每个测试人员一起看他是怎样测试产品的。我发现没有使用工具来检查正确的文件是否被安装很难开始。因此，我给他看 InCtrl5，一个免费固件工具能够对系统在某些特定事件之前或知乎进行截图，然后汇报文件和注册设置的变化。我们用了这个工具。

然后他告诉我他想去深挖的一个缺陷。显然有一个窗口应当在前面的却被另一个窗口挡在了后面。他的结论是哪个在前面的窗口正在接受某种设焦的信息，这让他把窗口在错误的时间放在了前面。他问我是否知道任何能够帮助测试那种假设的工具。我立即想到了 Spy++，一个用来监控 windows 消息的工具。它要先装载 Visual 9 C++才能使

用。（一个工匠师必须投资一定比率的时间来获得所有种类工具的基本知识及到哪里去获得，这里它得到好处了。）我们围绕 Visual C++争着寻找了一会。最终，那个测试人员找到一个开发人员说服他来展示给我们 Spy++。我们不仅得到了那个工具，还跟那个开发人员对这个工具可能会或不可能对我们的调查有所帮助进行了一场有趣的谈话。

这表示敏捷自动化有时候是调查研究解决方案，而那个过程是怎样的一个活动将吸引开发人员来帮助测试事业。

转变为敏捷测试自动化:

（1） 建立和宣布测试自动化团队人员给开发人员和测试人员

（2） 为自动化团队建立一个汇报系统和发布系统

（3） 在公司内对已知测试件制作一个发明

（4） 对公司外可得到的测试件做个快速的调查

（5） 让工匠师跟测试人员坐一起，理解他们是怎么测试的，并识别出测试自动化机会

（6） 给测试人员创造制作请求并看着它快速被授权的体验

（7） 以不低于每两周一次的频率来评审自动化工作的进度

（8） 持续提升对测试人员的培训工作

UFX 全流程自动化测试的设计与实现

◆作者：姜志晨

摘要：随着一款软件功能的不断强大，测试的工作量将逐渐增加，在测试资源不变的情况下，以人为驱动的手工测试将会遇到瓶颈。为了节省人力、时间等资源，提高测试效率，便出现了自动化测试的概念。本文主要介绍针对接口软件 UFX（United Finance Exchange，统一金融接入系统）的全流程自动化测试方法的设计与实现。

一、前言

作为软件生命周期中的一个重要环节，测试是确保软件质量必不可少的工作。随着一款软件功能的不断强大，测试的工作量将逐渐增加，在测试资源不变的情况下，以人为驱动的手工测试将会遇到瓶颈。为了节省人力、时间等资源，提高测试效率，便出现了自动化测试的概念。

本文主要介绍针对接口软件——UFX（United Finance Exchange，统一金融接入系统）全流程自动化测试方法的设计与实现。

二、UFX 相关背景

（一）UFX 介绍

UFX（United Finance Exchange，统一金融接入系统），是由恒生电子开发的公司级的统一接入平台。一方面，这个平台可以提供公司级的企业服务总线，将业务系统通过这条总线进行消息传递、数据共享。对于接入的用户，无论是客户（投资者）还是员工（业务技术人员）都通过统一的协议和接口访问，简化终端系统的开发，优化公司数据的访问，最终实现信息系统的协作和共享。另一方面，这个平台可以提供各类接入协议的统一管理，包括不同的通信协议和不同的业务协议（如 C/C++/Java/C# API、Fix 等），其技术标准规范完全满足证券行业互联网接入的安全要求，从而支持互联网上各类专业、机构投资者个性化的接入需求。

（二）UFX 测试的主要工作

作为一款金融软件，UFX 主要提供了多种业务相关的委托、撤单、查询以及基础数据委托功能。针对这些功能，UFX 测试的主要工作包括以下三点:

- 确保输入字段与输出字段与接口文档定义一致;
- 确保对数据库的增加、删除与修改操作结果正确;
- 确保推送的消息内容与接口文档定义一致;

（三）UFX 测试遇到的瓶颈

随着 UFX 功能的不断强大，目前已经提供了上百个接口，支持了现货、期货、期权、基金、融资融券等多种业务。测试的工作量正不断增长，在测试资源有限的情况下，原有工作模式的弊端逐渐显露出来:

UFX 之前也有做过自动化用例，但是用例编写维护复杂、结果比对需要人工参与、用例往往无法直接循环执行需要手工切换回初始状态，这样的自动化测试相比手工测试并没有多少优势，没有受到测试人员青睐，自动化推行受到阻碍。

面对对接多个服务端的上百个接口，每一次枯燥而繁重的回归测试对于使用手工测试的测试人员而言都是一场灾难，限于人力和时间限制，回归测试无法深入到每个接口的细枝末节，回归测试质量有待提高。

之前的工作模式下日常只会针对有修改的接口进行测试，导致由于一个接口改动而引入的另一个接口的缺陷往往只能在回归测试的时候才有机会暴露出来，对发版计划造成影响的同时也增加了软件质量的风险。

作为一款接口软件测试工作，除了升级维护接口软件的测试程序，还需要升级维护对接服务端的测试程序，长此以往日常环境维护也是一笔不小的开销。

综上，手工测试无法满足需要，自动化又推广不起来，程序质量不断受到威胁，开发新功能的同时还要耗时间来修改之前遗漏的问题，导致新开发的程序也没法有较高的质量，变成了一个恶性循环。针对上述问题，UFX 测试遇到的僵局需要被一种易使用、日常化、全流程的自动化模式来打破。

三、UFX 全流程自动化测试遇到的问题及解决方案

本章主要通过介绍 UFX 自动化设计过程中遇到的主要问题的解决方案，描述一下 UFX 全流程自动化的设计过程。

（一）如何提高自动化用例的可维护性？

为了便于自动化用例的编写、评审以及日后维护和推广，自动化用例需要做到易读、易写、易管理，为了达到这个目标，解决方案主要包括以下几点：

鉴于 Excel 的强大的编辑功能以及清晰的结构，我们选择使用 Excel 表格对测试数据进行管理。

支持自定义脚本简称，使用的命令更容易记忆，用例的读写简洁化。

放弃之前同一接口一份测试用例的模式，将一个接口的用例根据测试点拆分成多个子用例，便于单独执行，也便于后续的用例补充。

用例存放的目录接口层次简清晰并且简洁。

需要制定统一的命名规范、数据格式规范、目录结构规范、备注书写规范，UFX 的测试人员都需要遵守这些规范来编写用例。

（二）如何提高自动化用例的可重复性？

由于 UFX 接口调用后，数据库往往会产生变化，再次执行需要手工切换虚拟机快照才能保证结果一致。这种模式下用例的可重复性就很低，为了解决这个问题，解决方案包括以下几点：

用例开始执行需要有一个统一的“起点”，只有在“起点”相同的前提下同一份用例执行完毕之后的结果才有可比性，对于 UFX 测试工作来说，这个“起点”就是一个统一的 oracle 标准库。

每个用例执行前需要对这个用例可能会涉及到的表进行备份，该用例执行完毕之后再对这些变动过的表进行恢复，确保用例执行完毕之后又回到之前的“起点”，不会影响到该用例的再次执行或者另一个用例的执行。其中备份表与恢复表可以通过事先实现的存储过程来实现。

（三）如何提高自动化用例的独立性？

解决了用例不可重复的问题，还需要解决用例的独立问题。之前 UFX 一个接口只有一份用例，导致这个用例数据内容过多，不方便维护。另外，多个接口的用例串联执行的时候之间也会相互干扰，当一个用例执行失败时，也会因为多米诺骨牌效应导致后续用例全部失败。针对用例独立性的解决方案包括以下几点：

一个接口根据不同的测试点将用例拆分为多个子用例，可以根据需要对需要执行的子用例进行挑选，结合子用例的比对结果也可以更快速的定位问题原因。

同一接口的不同子用例之间以及不同接口的用例之间使用本章第二节中提到的备份与恢复数据库的方法来排除干扰，确保相互之间的独立。

结合上面提到的重复性，做到多个用例可以不受执行顺序和执行个数的影响，多次循环执行。

（四）如何提高自动化用例的缺陷敏感性？

之前的 UFX 自动化用例在检查返回结果和数据库变化时，每个字段都需要单独写出期望，由于这部分工作很耗时，往往自动化用例就仅对重要的字段做了检测，这样自动化用例对缺陷就比较迟钝，遗漏率也就比较高。为了使自动化测试用例对缺陷更敏感，解决方案包括以下几点：

参考 UFX 测试的主要工作内容（见第二章第二节），自动化测试工具新增功能：支持将接口返回结果导出为 Excel 文档；支持通过触发器监控数据库变化，将变化导出为 Excel 文档；支持订阅推送的消息，将结果导出为 Excel 文档。

自动化测试工具需要支持 Excel 文档的比对。首先，首次执行用例后，通过人工测试导出的 Excel 文档，确保导出结果正确后将这些 Excel 作为“标准数据”；其次，用每次执行完用例得到的“导出数据”与“标准数据”进行比对；最后，根据比对结果得到测试通过或者测试失败的结论。

由于测试软件的正常变动，也有可能导致比对不一致，这时需要及时更新“标准数据”。

自动化测试工具支持过滤不比对字段，便于调整自动化用例对于缺陷的敏感度。

如何使自动化用例再保证覆盖率的情况下更精简？

虽然全流程的接口自动化测试目标是在无人看守情况下运行，测试的执行效率就不是主要关注点。但是冗长的测试数据会导致用例不便于编写与维护，对于测试资源也是一种浪费。如何确保自动化用例在覆盖率尽可能大的前提下尽可能精简，解决方案包括以下几点：

- 测试用例根据等价划分原则进行精简，无需做所有字段的全组合覆盖。

- 通过自动化用例评审以及 GCOV（GCC Coverage，一个测试代码覆盖率的工具）代码覆盖率来确保用例的覆盖率。

（六）如何准备自动化用例的执行环境？

为了确保自动化用例顺利编写与执行，对于环境有以下几点要求：

- 自动化测试环境要独立。由于自动化测试用例的执行对数据库依赖性比较高，因此自动化测试不能与手工测试共用同一个环境，而且两个测试人员不能同时使用同一个自动化环境执行测试用例。
- 自动化测试环境要易恢复。由于自动化用例编写调试的过程中，可能会导致数据库数据混乱，这时需要自动化环境可以方便的切换到之前备份的状态，如使用虚拟机的快照切换功能。
- 不同 UFX 自动化测试环境都是用同一份标准库。这样可以保证同一份用例可以在多个 UFX 自动化环境执行，便于用例的移植复用。

（七）如何实现的 UFX 自动化测试的无人值守？

完整的 UFX 测试流程除了用例的执行，还包括测试包的同步与升级这两步环境维护工作，如何将这两个步骤也通过自动化执行，解决方案包括以下几点：

- 使用 AllwaySync 文件资料同步软件，将 FTP 服务器上最近更新的测试包同步到自动化测试环境。
- 使用后台升级工具，对自动化测试环境进行升级。
- 使用自动化测试工具执行测试用例，并将执行结果通过邮件告知测试人员。

编写脚本将上面三个步骤串联执行，再通过 windows 定时功能来执行这个脚本，做到全流程自动化执行。

四、UFX 全流程自动化测试的实现

上面一章主要介绍了针对 UFX 全流程自动化遇到问题设计的解决方案，本章主要介绍一下目前 UFX 自动化测试实现的流程关系。

（一）UFX 自动化测试用例执行流程

结合第三章中的解决方案，最终按照如图 1 所示的流程图实现了 UFX 自动化用例

执行（该流程图假设已经通过了首次执行，并得到了“标准数据”），称为 “五步法”：

图 1 UFX 自动化五步法流程图

其中五个步骤分别为：

第一步，准备阶段。该步骤主要是进行数据库的备份，并根据具体需要进行准备工作，包括：监控数据库变化的触发器创建，主推消息订阅，以及相关数据准备等。

第二步，用例执行。该步骤主要是将事先编写好的用例数据顺序执行。

第三步，导出数据。该步骤将用例执行中得到的返回结果、数据库变化信息、消息主推等内容导出为 Excel 文档。

第四步，数据比对。该步骤将“导出数据”与“标准数据”进行比对，得到比对结果。

第五步，恢复阶段。该步骤主要是将变动后的数据库恢复为备份时的状态，便于之后的用例正常执行，此外将执行结果通过邮件发送给测试人员。

（二）UFX 全流程自动化测试流程

UFX 全流程自动化测试，除了作为主体部分的用例执行，还包括测试程序的获取与

升级等操作，具体流程图如图 2 所示：

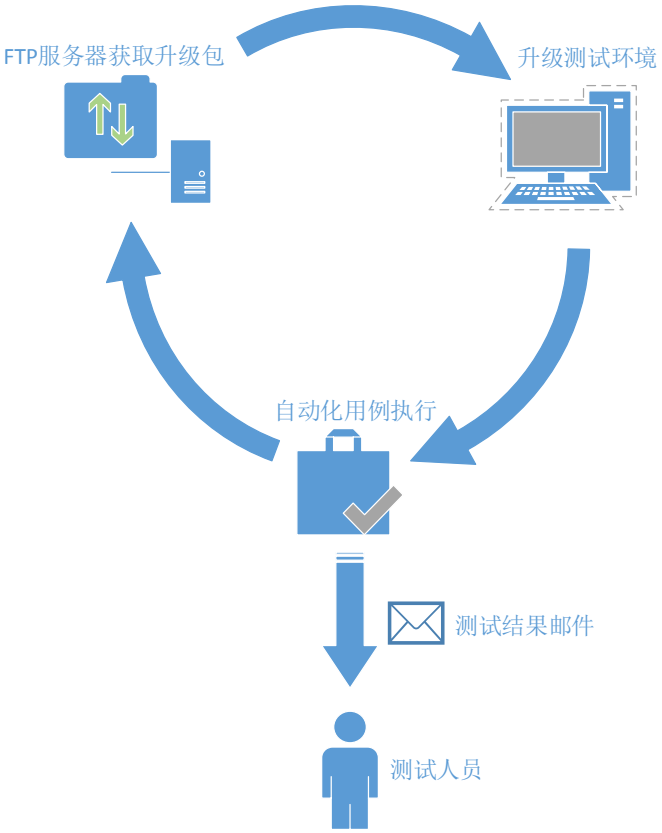


图 2 UFX 全流程自动化流程图

其中主要包括：

- 使用 AllwaySync 文件资料同步软件，从 FTP 服务器上获取最新的测试程序。
- 使用后台升级工具，对自动化测试环境进行升级。
- 使用自动化测试工具执行测试用例（前面一节介绍的“五步法”），并将执行结果通过邮件告知测试人员。

编写脚本将上面三个步骤串联执行，再通过 windows 定时功能来执行这个脚本，最终实现 UFX 全流程自动化循环执行。

五、总结

至此 UFX 全流程自动化的设计与实现已经介绍完毕，目前这种自动化模式已经初见成效，第二章第三节中提到的瓶颈得到解决：

目前 UFX 日常测试已经实现了全自动化执行，新增的程序修改会直接补充自动化

用例来进行测试，此外所有自动化用例按照全流程的模式日常执行，确保所有接口新引入的缺陷能在第一时间发现。

相比手工测试，自动化测试对于工作效率的提升也在一次次的执行中体现出来，有效缓解了之前的困境，解放出来的测试资源用于优化工作过程、完善测试用例，使得UFX 测试工作进入良性循环。

借助自动化测试提升产品质量和工作效率这两大优势，UFX 最终实现在 2015 年全年无异常，全年准时发版。

此外自动化的“五步法”上手简单，新入职的员工也能在学习文档一天之后独自完成简单的自动化用例。目前自动化测试已经在其他测试小组慢慢推广开来，被更多的测试人员接受。

我们并不会止步于此，测试人员在朝着如何使自动化用例覆盖更广努力，同时自动化工具开发人员也在朝着提供更完善的功能努力。在自动化测试上的持续投入过程中，会面临新的挑战，但当自动化测试成为测试人员可靠的武器时，测试工作也将进入一个新的阶段。