目 录 (五十四期·下)

如果您也想分享您的测试历经和学习心得，欢迎加入我们~(*^▽^*)

投稿邮箱：editor@51testing.com

“软件测试已死”了吗

◆ 作者：多则惑少则明

摘要： 曾几何时，听到团队评价测试人员的工作就是页面上“点点点”；刚起步的项目团队，产品人员、开发人员也会参与页面上的“点点点”工作；难道测试人员的工作就果真这么没有技术含量吗？再这么发展下去，软件测试由项目中的其他角色人员担任，“软件测试已死”的观点不就真的应验了吗？但问题的根本在于，软件测试真的没有、或者不需要技术含量吗？下面就自己一些经验和感悟，聊聊自己的一些体会。

一、背景

相信很多人大概早就听过“软件测试已死”的论调了。放眼当前国内的互联网公司，无论是成熟的大公司、还是处于创业时期的小公司，对于大多数从事软件测试的同学来说，都面临测试普通技术含量低，大多数以功能测试为主，日常工作就是在页面进行“点点点”… 所以，对于工作了几年，却一直在做功能测试的人来说，不仅在公司内部没有什么竞争优势，几乎处于分分钟可以被换掉的处境中；也看不到未来的方向，想跳槽，去其他公司面试也几乎没有优势可言。

对于纯做功能测试的人员来说，由于本身即是含量低，入门门槛低，所以，有条件的公司都会将其包给外包公司来做，让 QA 同学专注于质量的其他方面了。但就目前大多数的功能测试而言，还是 QA 同学的本职工作。虽然很多公司招聘 QA 的时候，都将其职位标为：测试开发工程师。但实际 QA 的工作，还是以功能测试为主，或者说，QA 想要做些有技术含量的工作，还要自己“发愤图强”。

举个例子，A 同学负责部门的一个新业务，每天业务测试忙到吐血，几乎没时间搞其他东西。B 同学负责部门比较成熟的一个业务，项目压力不大，每天搞搞自动化、写写工具等等。到最后晋升、绩效评估的时候，大多数情况下 B 同学会比 A 同学有优势。究其原因，不外乎：

1、A 同学工作的技术含量不如 B 同学。A 同学的工作入门门槛低，几乎可以分分钟被其他同学接手；反观 B 同学的工作技术含量较高。

2、A 同学产出不如 B 同学。A 同学除了业务产生，再无过多其他，反观 B 同学，有业务、自动化、工具等产出

假设部门要裁员，大概率的情况下，相比 B 同学，A 同学走人的概率要大得多。那么问题来了，如何让自己的工作更加倾向于 B 同学，而非 A 同学呢？更进一步说，如何更大范围内，发挥自己作为测试同学/QA 同学的价值呢？

二、测试同学/QA 同学的价值高低如何体现？

如同其他工种一样，评价一位测试同学/QA 同学的价值高低，可以大致从以下范围考虑：

1、站在测试同学/QA 同学的角度，尽可能多的在测试岗位体现自己的价值

1）从业务层面来说，测试同学/QA 同学的价值就是：最大可能保证业务质量，避免业务的损失。

当然了，保证业务质量的手段就是——通过各种手段找 bug。这里的 bug 不是狭义范围内的找功能 bug，而是广义范围内的找业务 bug，包括功能 bug，性能 bug，一致性 bug,兼容性 bug 等等，也包括使用风险、业务风险等的风险类 bug，流程中的 bug，故障预防类 bug，监控类 bug，故障恢复类 bug 等等。广义范围内认为，一切影响/干扰用户使用的直接、间接问题都是系统的 bug。

2）提升自己的核心竞争力，提高自己被取代的难易程度，让自己不容易被取代

为了提高自己被取代的难易程度，就要提高自己日常工作的入门门槛，增加自己工作的技术含量。这样，一方面，对自己的晋升/绩效有直接的收益，另一方面，哪怕日后对自己跳槽也能成为自己的优势。

可能你会问，哪些工作是有比较高的技术含量的呢？对于这个问题，其实你可以完全自己找答案，找你自己需要的答案。具体的来源可以是招聘信息、高职级的具体要求、身边的牛人等等。当然了，有技术含量的工作除了有入门门槛外，还需要考虑自身的条件、职业发展等，自己有所取舍地选择适合自己的。

3）确保自己的专属价值，即除了你之外，其他人一般干不了。

专属价值的入门门槛更高。例如，那些专攻某个领域的技术牛人，往往能拿到 special offer，薪资是普通技术人员的 2 倍、甚至更多，究其原因，就是因为这些技术牛人有了自己的专属价值。

2、站在团队/部门的角度，尽可能多的影响其他人

只有你有更多的影响力，你才能对别人产生更多的价值，你对别人来说才是有用的。因而，从考虑影响更多人这个角度，可以考虑其他人到底需要什么，如何影响到其他人。

比如，你可以通过分享、提供工具给其他人用、解决普遍存在的一类问题、让自己的工作成为行业标杆等等，让其他人收益。

3、站在公司的角度，尽可能多的“为公司创造价值”

为公司创造更多价值，是提升自身价值、为团队/部门创造更多价值的结果。

当然了，非高管等职位，谈到为公司创造价值，往往会显得空洞，但可以从实际业务出发，来考虑。比如，所负责的业务直接/间接服务了多少用户、带来了多少收入、点击量等

三、测试领域对从业人员的要求浅谈

近几年互联网行业对于测试人员的招聘要求来看，各个公司对技术的要求是越来越高了！甚至在具体的面试中，对于有经验的测试人员，很少问具体的测试问题了，更多地在考察技术问题。目前不会写代码的测试人员，几乎找不到太好的工作了。圈中的同行也大多在讨论技术，比如，自动化、XXX 框架的编写、某某业务工具、各种辅助工具的开发等等。

从当前测试领域的趋势来看：测试领域对从业人员的要求越来越侧重技术能力了，而非测试思维。从测试人员的薪资也可以看出一二来，纯做功能测试人员的薪资，几乎不可能达到测试开发人员的薪资。因而，要想让自己在今后较长的一段时间，拥有竞争力，提升自己的技术能力，甚至达到一名资深开发人员的水平，是工作中的重中之重。

我曾经写过两篇文章:

1、《当公司今天找你谈话，明天让你走人时意味着什么》

2、《为什么互联网公司需要测试人员》

想来，测试领域的从业人员需要时刻保持警惕，避免陷入“温水煮青蛙”的舒适区中。努力提高自己的核心竞争力，让自己“分分钟”能找到更好的下家，这样的你，一方面即便公司裁员的时候，也不会轻易轮到你；另一方面，即便裁员轮到你，你也可以分分钟找到下一个落脚处。

四、写在最后

回到“软件测试已死”了吗的话题，个人认为，软件测试不会死，而且会一直存在。所谓专业的事情应该交给专业的人去做，软件测试领域中存在众多的技术方向，并且真正的软件测试应该是技术活，而非“体力活”。 如果你感觉一直、或经常在做体力活，那不妨，静下心来好好想想，是不是你目前努力的方向与测试领域的主流方向有所偏差呢？

测试领域中，每个从业人员职业生涯的长短并不是绝对的，也不是一成不变的，关键看从业人员能否跟得上主流方向的要求。假如今天的你，还在疲于奔命与“功能测试”，再无其他技术特长外，或许，可以看得见的将来，你的测试生涯也就结束了。但如果你始终围绕主流的方向开展测试工作，那么可以预见，测试生涯就是软件生涯！

往期作品：《如何证明你是一名优秀的测试人员》

测试环境管理：测试环境该填坑了

◆ 作者：侯 峥

随着公司业务的拓展和需求的增多，对于测试环境的要求也在越来越高。单一的测试环境已经满足不了现有的发展势头，扩展测试环境的规模，加强测试环境的管理已经成为摆在测试人员面前的重要问题。以当前的业务拓展形势和需求迭代频率，单一的测试环境以及配置策略不足以支撑完成测试任务，测试环境的实例数量层级和策略配置复杂度都需要想生产环境靠拢，以便完全模拟生产环境，及早暴露程序编码中出现的问题，但是这种环境又并不完全等同于灰度测试环境。

一、测试环境方面经常遇到哪些问题？

对于测试管理人员来讲，通常会遇到以下问题：

1、测试人员误操作，“rm -r *”删除测试实例

2、测试实例不够用需要扩充实例数量

3、虚拟机空间告警

4、部署操作机械频繁，需要规范及自动化操作

5、测试实例乱用，需要规范定义实例用途

对于以上的问题，有些问题可以提前规避，有些问题需要事后处理。无论处理方式是怎样的最终的目标都是解决问题，减小损失。这些问题的处理方式可以从技术上入手，也可以从工作流程上规范。

二、测试环境的问题会造成那些影响？

1) 测试人员误操作，“rm -r *”删除测试实例

对于误删这个问题，为大家多熟知的就是生产环境的删库跑路。实际上，误删现象

在测试环境上也多有出现，只不过是影响范围没有生产环境误删那么的严重，在比较之下，大家对于测试环境误删的事件就没有那么的重视。但是，实际上，由于测试环境误删所引发的蝴蝶效应也是很严重的。

蝴蝶效应是什么？这里就不必多说了。简单讲讲测试环境误删可能引起的问题。前不久，同事在测试环境执行“rm -r *”操作，删除了一台测试虚拟机上的全部实例，恰巧同事删除的是与其他厂商共用的测试环境，这个环境平时大多用于和对接厂商联调测试和客户演示用的。尽管误操作发生后，及时采取了补救，但是其他厂商和客户还是受到的影响。恢复环境的当天，收到了很多询问的电话。恢复环境一共花费了一天的时间，在这一天里联调的测试任务全部中断停滞，同时还要统一对外的解释口径。还好同事的账号只有本公司的权限，否则如果是删除了所有厂商的服务实例，恐怕再多的解释也掩盖不了问题。

2) 测试实例不够用需要扩充实例数量

对于测试任务的增加，需要增添测试实例，用于保证需求如期上线。这个问题在公司业务扩张的时候最容易出现。也是体现出了业务部门与技术部门的规划沟通不足。最近一段时间，需求的任务量加重，详细了解之后，才知道是业务部门增加了 3 个业务线条，需要对应的增加系统应用中的功能。但是对此技术部门并不知道，直到需求任务提出时，才意识到测试实例支撑能力不足，现有的测试实例支撑已经近饱和状态，再加上新增业务线条的需求马上捉襟见肘。

为了应对这个问题，临时搭建了几个测试实例，用于支撑新业务线条的需求功能测试。但是临时搭建的测试实例也存在一些问题，比如：与之前的测试环境规划有略微的出处，需要后期重复构建；新搭建的测试实例的策略可能有遗漏、测试数据不全，导致测试时任务中断。

3) 虚拟机空间告警

我们公司是为某电信运营商提供技术服务和支持。我们项目组的支撑业务系统是经营分析系统，系统的数据源传输，数据源处理，系统应用是由多个厂商共同协作处理。这种情况就使得我们需要一个共用的环境，这个环境用于测试联调、功能演示。在这种情况下，这个测试环境的总管理就需要甲方公司主理。因此，测试环境使用的健康情况就成为了甲方考核公司的一项指标。

由于需求版本的增加，提测版本的迭代，测试环境中的日志和备份内容也在增多。经过累积达到一定的数量会导致虚拟机空间不足。虚拟机空间不足会影响测试实例的性能，有些经验不足的测试、开发人员不能够快速的定位问题，更重要的是会影响甲方公司对于本公司的考核成绩。

4) 部署操作机械频繁，需要规范及自动化操作

公司部署服务是由测试人员在开发人员提测后，自行打包，再把打好的包部署到测试环境实例上，测试环境实例都是 tomcat 实例。由于测试人员打的版本包里的配置文件是开发本地的地址，如果直接部署的话，测试实例启动会报错，需要替换配置文件后重启服务。这些操作看似简单，但是如果操作不熟练或者不了解部署原理的，会很耽误工作时间。如果部署阶段出现问题，会致使花费大量的时间排查环境方面的问题。

备份是常听到的词。部署新的版本前，需要备份正在使用的版本，保证版本升级失败的版本回退操作。但是不同的人员的操作习惯和风格不一，这就使得测试环境的备份内容大相径庭，找不到规律。如果需要其他同事来恢复版本，在没有人员指导协作的情况下，不太容易找到需要恢复的版本。这样也不利于测试环境空间的高效使用，在清理空间时，也不能贸然清理。

5) 测试实例乱用，需要规范定义实例用途

在测试的过程中，有部分的 bug 是由数据问题导致的。这些 bug 并不是程序的缺陷，是垃圾数据导致的程序报错，实际生产环境并不会出现这样的数据。那么这些垃圾数据是如何产生的呢？敏捷开发是当下使用最广泛的开发模式，他的特点在于功能的小版本迭代，不需要等功能都开发完再提测，减少了测试的等待时间。但是对于程序来讲，按照功能点提测产生的数据就有可能成为下一个版本提测功能的垃圾数据，如果环境数据控制的不严格，会出现很多由数据导致的问题阻碍测试进行。排查这类问题花费的时间也占用了测试的总时间。

对于对外提供联调能力或者演示用途的测试环境，如果遇到由于垃圾数据导致的问题，客户或者其他厂商会直接定义为延误工作进度，他们并不清楚造成系统问题的原因。从客户或者其他厂商的视角，他们只关心系统能不能使用，并不关心造成系统不能使用的原因，如果遇到问题会直接记录考核。

三、针对测试环境问题的解决方案

针对测试环境问题的解决方法可以从技术上、规范上两个方面入手，但是倾向技术方案解决为主，规范方案解决为辅，减少人工干预以及认为操作带来的失误。

在测试环境内容误删方面，预防实例被删以便快速恢复实例，测试环境安装 dump 定期备份测试环境内容。

在测试环境空间告警方面，定期清理，创建 crontab 删除历史分支及日志文件。

在员工操作方面，编写部署脚本、生成测试数据脚本。脚本里定义备份方式以及备份路径，减少个人倾向，简化操作流程。

在测试规范方面，明确定义测试环境实例的用途，严禁实例乱用，制定对应的考核措施用以规范员工操作，细分员工操作权限。

在测试环境规划方面，与业务部门加强沟通，根据业务规划扩展测试环境规模，规划测试实例、路径及其用途，及时协调资源；与开发部门协商同一时段上线的需求内容在同一代码分支开发，提高测试环境的使用效率；测试环境实例预留支撑能力，以便应对需求突增的情况。

总结

对于测试环境的管理和规划相当重要，很多情况下，大家会忽视测试环境管理，很多的问题可以在测试环境上体现出来，但是由于测试环境管理的失责，致使问题上线了才暴露出来，这个时候解决问题的成本要远远高于测试阶段解决问题的成本。因此，加强测试环境的管理与维护也是保障质量的一种手段，也是保障质量的一种相对廉价的途径。测试环境规划的越合理，越与生产环境相似；越能够提高工作效率，越能及早发现问题。

往期作品：《由测试的历史追溯测试发展》

5 种常见的测试用例设计方法

◆作者：唐 米

摘要：一四年在 YX 公司带测试团队，一个用例评审的会议上，一不小心超常发挥，结果卡在了一个用例设计方法上，印象非常深刻，当时的业务场景是支付方式的选择和优惠方案。

再后来的工作中，也曾几次遇到需要选择合理的设计方法来写用例，不过每次在网上都是搜索了半天，也找不到令人满意的答案。很多简单的问题被复杂化，然后给出的解题思路更是令人百思不得其解。

网络资源下，任何一个问题都不缺答案，更多的时候缺的是个让人一目了然的答案。

测试前准备

作为一个测试人员，软件测试的流程首先是要非常熟悉的，何时何地都能脱口而出，避免一切翻车的可能。需要注意的是流程没有唯一答案，具体由项目决定。所以给出的只是一个还算通用的参考流程。

我们要熟知的测试流程：

总结一下：在测试流程中，有 6 个部分，其中 3 个部分涉及到了用例，可见写好用例的重要性。

所以，结合这些年吃过的亏，我来给大家缕缕，如何快速的 get 到测试用例的设计方法。

5 种常见的测试用例设计方法

一、等价类划分

1) 概念

某个输入域的集合，在这个集合中每个输入条件都是等效的，如果其中一个的输入不能导致问题发生，那么集合中其它输入条件进行测试也不太可能发现错误。

关于等价类划分的两个重要概念:

有效等价类: 有效等价类是程序规格说明有意义，合理的输入数据。

比如用正确的用户名和密码来登录系统就是有效等价类。

无效等价类: 无效等价类是程序规格说明无意义，不合理的输入数据。

比如用不存在的用户名和密码来登录系统就是无效的等价类。

2) 等价类法设计测试用例的步骤

为每个输入划分等价类，得到等价类表，为每个等价类规定一个唯一编号

设计一个测试用例，使其尽可能多的覆盖所有尚未覆盖的有效等价类。重复这一步骤，使得有效等价类均被测试用例所覆盖设计一个测试用例，使其只覆盖一个无效等价类。重复这一步骤使得所有无效等价类均被覆盖。

假设上面的文字你都没有看懂，那么做个题目就懂啦。

3) 案例来了

程序规定：输入三个正整数作为三边的边长构成三角形。请用等价类方法设计测试用例分别判断输入 3 个整数时的三角形为一般三角形、等腰三角形、等边三角形时情况:

提示:

需求提取:

1、三条边需求：整数/3 个数/非零数/正数

2、一般三角形的要求：二边之和大于第三边

3、等腰三角形：二二边相等且满足二边之和大于第三边

4、等边三角形：三条边相等

参考答案

测试项	测试点	详细需求	有效等价类	编号	无效等价类	编号
三条边	3个数	整数、非0数、正数	a	A01	a=0	B01
			b	A02	b=0	B02
			c	A03	c=0	B03
			a为整数	A04	a<0	B04
			b为整数	A05	b<0	B05
			c为整数	A06	c<0	B06
					a为小数	B07
					b为小数	B08
					c为小数	B09
构成三角形	符合三角形要求	两边之和大于第三边	a+b>c	A07	a+b<=c	B10
			a+c>b	A08	a+c<=b	B11
			b+c>a	A09	b+c<=a	B12
等腰三角形	符合等腰三角形要求	两边相等	a=b	A10	a!=b且a!=c且b!=c	B13
			a=c	A11		
			b=c	A12		
等边三角形	符合等边三角形要求	三边相等	a=b=c	A13	a!=b	B14
					a!=c	B15
					b!=c	B16

答案解析：符合的需求条件的即是有效等价类，比如，等腰三角形，那么要求至少有两条边相等，所有有效等价类就包括 a=b b=c a=c ，那么不符合条件的就是无效等价类包括 a!=b b!=c a!=c

二、边界值分析

1）概念

边界值分析方法的理论基础是假定大多数的错误是发生在各种输入条件的边界上，如果在边界附近的取值不会导致程序出错，那么其它的取值导致程序错误的可能性也很小。

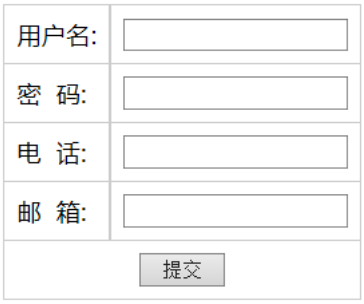
2）边界值分析法设计用例的步骤

- 分析输入参数的类型：从测试规格中分析得到输入参数类型
- 等价类划分（可选）：对于输入等价类划分方法进行等价类的划分
- 确定边界：运用域测试分析方法确定域范围的边界（上点、离点与内点）
- 相关性分析（可选）：如果存在多个输入域，则需要运用因果图、判定表方法这些输入域边界值的组合情况进行进一步分析

- 形成测试项：选择这些上点、离点与内点或者这些点的组合形成测试项

3）案例来了

假设存在以下的测试场景，某个网站的登录页面：



1、用户名：1—20 个字符，包括 1 和 20，其他不考虑

2、密码：6 个数字，其他不考虑

现要求用边界值分析法测试用户名和密码这两个输入框。

参考答案

边界值分析方法如下：

测试项	有效等价类	无效等价类	边界值
姓名	1-20个字符	/	0个字符 1个字符 2个字符 19个字符 20个字符 21个字符
密码	6个数字	/	5个数字 6个数字 7个数字

答案解析：密码这个字段的范围是闭区间【1-20】，用边界值设计用例，那么去找这两个数的左邻右舍 + 自己，1 则是 0 和 1 和 2，20 则是 19，20，21。

三、判定表

1）概念

判定表是分析和表达多种输入条件下系统执行不同动作的工具，它可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确 。

2）判定表法设计用例的步骤

- 列出所有的条件桩和动作桩
- 填入条件桩、条件项
- 填入动作桩、动作项
- 化简，合并相似规则
- 将每条规则转化为用例

3）案例来了

假设有以下逻辑:

条件	动作
如果觉得疲倦并且对书的内容感兴趣，不糊涂的话	回到本章重读
如果觉得疲倦并且对书的内容感兴趣，但糊涂的话	继续读下去
如果不觉得疲倦并且对书的内容感兴趣，但糊涂的话	回到本章重读
如果觉得疲倦并且对书的内容不感兴趣，但不糊涂	跳到下一章去阅读
如果觉得疲倦并且对书的内容不感兴趣，但糊涂的话	请停止阅读，休息
不疲倦，对书的内容感兴趣，书中的内容不糊涂	继续读下去
不疲倦，不感兴趣，书中内容糊涂	跳到下一章去读
不疲倦，不感兴趣，书中内容不糊涂	跳到下一章去读

运用判定表设计用例。

参考答案

		条件项							
条件桩	是否疲倦	Y	Y	N	Y	N	N	N	N
	是否对书的内容感兴趣	Y	Y	Y	N	N	Y	N	N
	是否让你糊涂	Y	N	Y	N	Y	N	Y	N
		动作项							
动作桩	重读	Y		Y					
	继续读		Y						
	停止阅读				Y	Y			
	跳到下一章						Y	Y	

答案解析：判定表的解题思路就是先列出所有条件，然后列出每个条件的取值，最后如上图，一列就是一条用例。

四、正交试验法

1）概念

正交试验设计(Orthogonal experimental design)是研究多因子多水平的又一种设计方法，它是根据正交性从全面试验中挑选出部分有代表性的点进行试验，这些有代表性的点具备了“均匀分散，齐整可比”的特点。

关于正交试验表的两个重要概念:

1、所有参与试验、影响试验结果的条件称为因子。

2、影响试验因子的取值或输入叫做因子的水平。

如何选择正交表:

1、考虑因子的个数

2、考虑水平的个数

3、考虑正交表的行数

4、取行数最少的一个

2）案例来了

有如下用户登录页面，三个登录条件：用户名、密码、验证码，考虑填写或不填写，用正交表设计测试用例。

参考答案

分析因子数，以及因子水平值:

3 因子 2 状态

因子	水平
用户名	填写
	不填写
密码	填写
	不填写
验证码	填写
	不填写

经过组合合并之后的对应用例

用例编号	用户名	密码	验证码
1	填	填	填
2	填	不填	不填
3	不填	填	不填
4	不填	不填	填

补充:

5	不填	不填	不填

答案解析: 正交试验法主要在于选取因子数和水平值，将两者结果列出再合并，工

作中用的不多，但是合适的业务逻辑下可以选择。

五、流程分析法

1）概念

流程分析法是将软件系统的某个流程看成路径，用路径分析的方法来设计测试用例。根据流程的顺序依次进行组合，使得流程的各个分支都能走到。

2）流程分析法设计用例步骤

1、画出业务流程图

2、设置功能路径优先级

3、确定测试路径

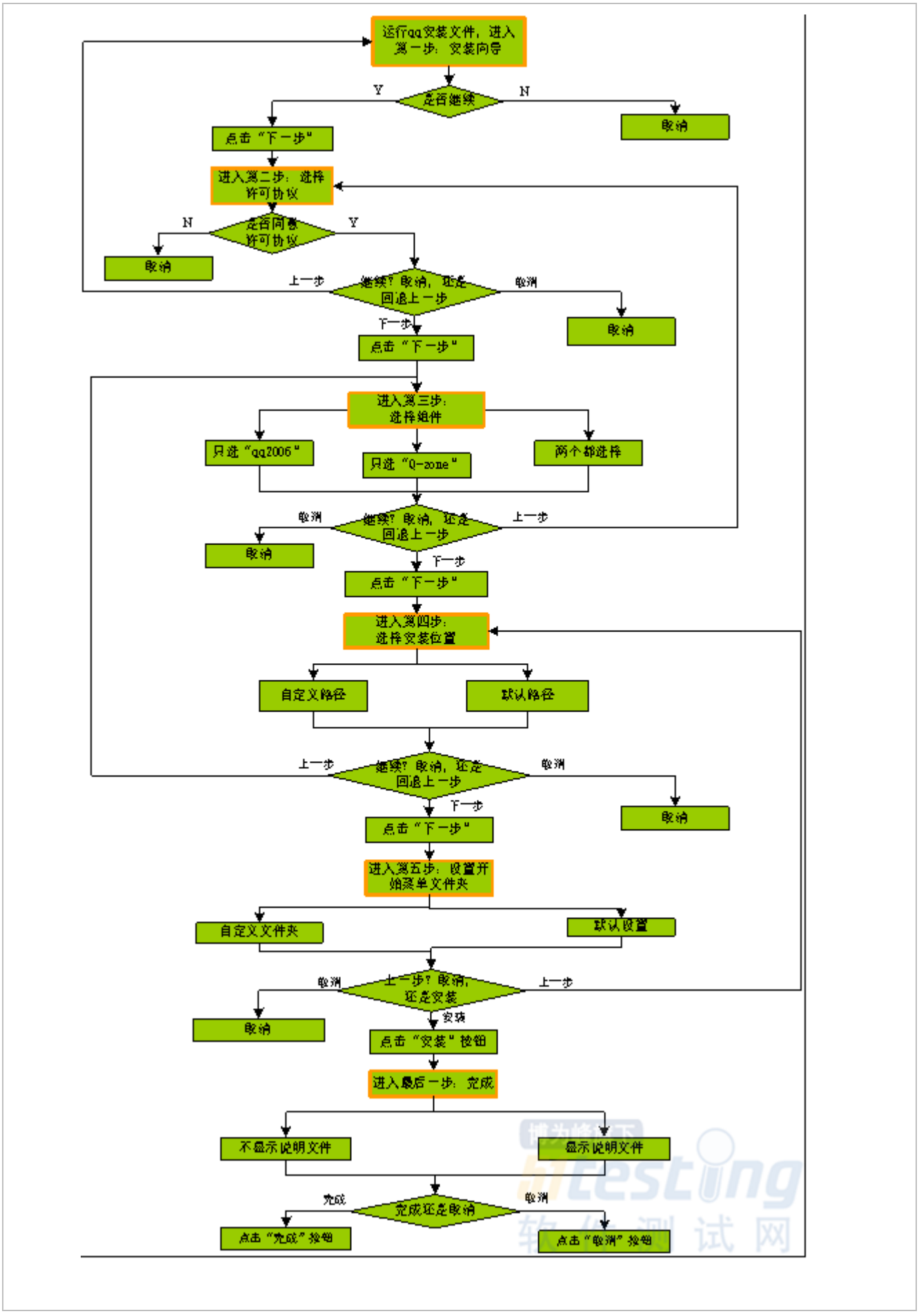
4、选取测试数据

5、构造测试用例

3）案例来了

案例：安装 QQ，安装系统之家版 QQ。

参考答案


运行qq安装文件，进入第一步：安装向导
是否继续
Y
N
取消
点击“下一步”
进入第二步：选择许可协议
是否同意许可协议
N
Y
取消
继续？取消，还是回退上一步
上一步
取消
下一步
点击“下一步”
进入第三步：选择组件
只选“qq2006”
只选“Q-zone”
两个都选择
继续？取消，还是回退上一步
取消
上一步
取消
下一步
点击“下一步”
进入第四步：选择安装位置
自定义路径
默认路径
继续？取消，还是回退上一步
上一步
取消
下一步
点击“下一步”
进入第五步：设置开始菜单文件夹
自定义文件夹
默认设置
上一步？取消，还是安装
取消
上一步
安装
点击“安装”按钮
进入最后一步：完成
不显示说明文件
显示说明文件
完成还是取消
完成
取消
点击“完成”按钮
点击“取消”按钮

对应的测试用例

用例编号	操作步骤	预期结果
1	1.运行qq安装文件，点击【取消】按钮 2.跳出弹框，点击【是】按钮	程序正常退出
2	1 运行qq安装文件，点击【下一步】按钮 2 选择【不同意】	1 程序进入许可协议设置页面 2 程序正常退出
3	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【取消】	程序正常退出
4	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【上一步】	程序回退到第一步“安装向导”
5	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【取消】	3 程序进入选择组件页面 5 程序正常退出
6	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【上一步】	3 程序进入选择组件页面 5 程序回退到第二步“选择许可协议”
7	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【下一步】 6 选择【默认路径】 7 点击【取消】	5 程序进入第四步“选择安装位置” 7 程序正常退出
8	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【下一步】 6 选择【默认路径】 7 点击【上一步】	7 程序回退到第三步“选择组件”
9	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【下一步】 6 选择【默认路径】 7 点击【下一步】 8 选择【默认文件夹】 9 点击【取消】	7 程序进入第五步“设置开始菜单文件夹” 9 程序正常退出
10	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【下一步】 6 选择【默认路径】 7 点击【下一步】 8 选择【默认文件夹】 9 点击【上一步】	9 程序正常回退到第四步“选择安装位置”
11	1 运行qq安装文件，点击【下一步】按钮 2 选择【同意】 3 点击【下一步】 4 选择“qq2006” 5 点击【下一步】 6 选择【默认路径】 7 点击【下一步】 8 选择【默认文件夹】 9 点击【下一步】 10 选择“显示说明文件” 11 点击“完成”	9 程序进入最后一步 11 程序安装完成
……	……	……

答案解析：看起来非常复杂的流程图和用例，设计的核心其实就两个方向，一个是正常流程，即安装安装向导一直点击【下一步】直到完成。另一个方向则是每个判断条件选择【否】的场景，那么这样的话，就会产生很多条其他分支的用例。

结尾篇

测试用例设计方法不止上面提到的 5 种，但是工作中遇到的业务场景基本可以通过上述的方法来得到解决。等价类划分和边界分析方法较为简单，很多时候可以结合起来一起用。正交实验表和判定表一般用在需要将多个输入组合起来测试的情况。流程分析法顾名思义就是存在不同分支流程的时候选用。

希望上述的方法能够帮助到大家。

为 Web 元素创建最终 XPath 的 20 种最佳方法

作者：桃 子

使用于任何 Web 元素类型的 XPATH 定位的前 20 种方法（XPATH 永远不会无效）:

Web 应用程序由不同类型的 Web 元素组成，例如用于单击按钮 Web 元素，输入以键入文本的 Web 元素，下拉列表，单选按钮等。

这些 Web 元素也称为标记或节点。

在自动化 Web 应用程序时，首先要编写一个自动化脚本，该脚本将找到 Web 元素，对其执行操作，例如，单击按钮，在输入框中输入文本，选择复选框，选择单选按钮，向上或向下滚动，最后验证操作的预期行为

找到一个元素就像在地图上找到某人的房子一样。我们在没有任何外部帮助的情况下找到朋友家的唯一方法就是我们应该有一张地图并知道要找什么（房子）。

在我们的案例中，地图就好像 DOM（HTML 标签，JavaScript 等），其中存在所有 Web 元素，以及我们想要查找的特定 Web 元素都在地图里

找到元素的唯一地址或路径后，自动化脚本将根据测试场景对其执行某些操作。例如，您要验证单击按钮后打开的页面的 URL 地址是否正确

但是，找到 Web 元素的唯一地址/路径并不能成功验证，因为可能存在类似的标记，相同的属性值，相同的路径，因为很难为称为“XPATH”的 Web 元素创建精确的唯一地址。

在这里，我们将深入探讨一些非常有效的技术，为任何类型的 Web 元素生成有效且独特的 XPATH

有时您可以使用浏览器扩展轻松创建 XPath，但在我的自动化测试职业生涯中，我

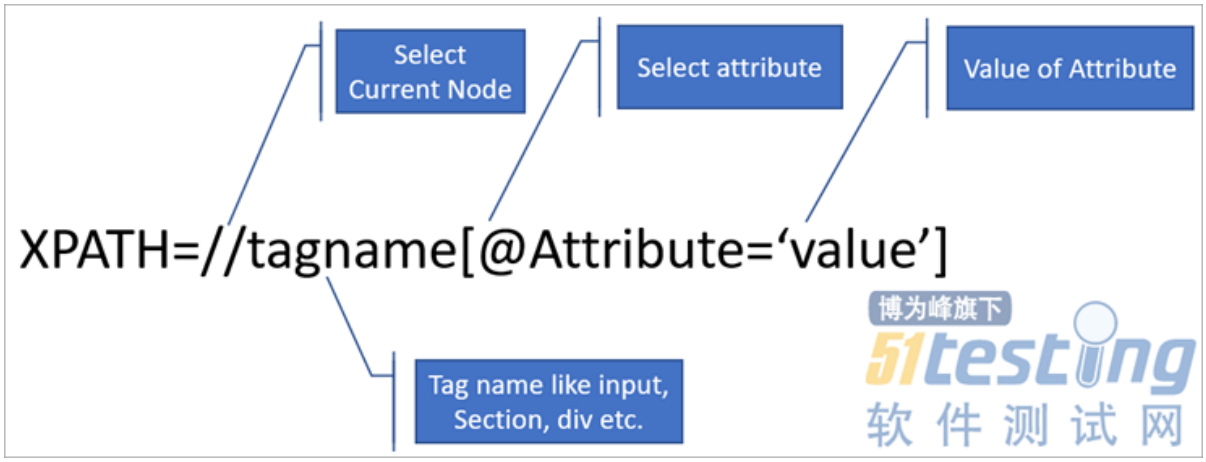
遇到了无数传统浏览器扩展无法工作的情况，您必须使用自己的创造力来提出自己的自定义 XPath。我确信你已经或将面临类似的情况。

在本教程中，我们将介绍如何为 Web 元素创建最终 XPath 的 20 种最佳方法，即使代码更改后，XPath 将始终保持有效（除非开发人员重写整个特征/模块）。

通过了解所有这些技术，您将成为编写自己的 XPath 的大师，并且能够编写杀手级 XPath，几乎没有机会变得无效。

首先，让我们首先理解 XPath 语法并定义其每个部分

XPath 的外观以及每个部分的描述



- //：选择当前节点，例如 input，div 等。
- **标记名：** Web 元素/节点的标记名
- **@：** 选择属性
- **属性：** 节点/特定 Web 元素的属性名称
- **值：** 属性的值

在这里分享一些技巧，80％的时间我的自动化测试脚本因 XPath 而失败。这是由于提供的 XPath 或 XPath 有多个 Web 元素无效或页面尚未加载。

因此，只要您的测试用例失败：

1、复制你的 XPath。

2、在 DOM 中的浏览器（F12 或开发人员工具窗口）中搜索它以验证它是否有效（参见下图）。

专业提示 1: 确保路径是唯一的，在 DOM 中搜索两次时，不会出现其他网络元素。

专业提示 2: 有时会出现计时问题，这意味着在脚本查找时您的网页元素/页面尚未加载，因此会增加一些等待时间并重新测试。

专业提示 3: 可以在搜索 Web 元素之前打印整个 DOM，这样，可以通过查看控制台来判断您的 Web 元素是否存在于 DOM 中。

在我们深入研究 XPath 之前，如果您可以直接访问开发团队，或者您的团队位于您所在的位置，那么请您的开发团队为您提供唯一的 ID 每个 web 元素或至少是您要用于 Automation 的元素，这将节省您的大量时间。

如果这种可能性无法实现，那么您可能需要使用您的创造力并提出自己的定制 XPath，这就是我们现在要学习的内容。

1）反向查找

假设你想点击一个按钮，就会有一个类似的按钮。两个按钮都具有 id 属性，但它们是动态的，并且两个按钮元素中的属性都不是唯一的。

在下面的场景中，我们要点击“Test Interactive”的“Setting”按钮。

代码如下：

```
<html>
  <head></head>
▼<body>
 ▼<widget id="rcTEST" connectivitytype="TEST" data-type="remoteConnection" data-path data-context="context" class="ng-isolate-scope">
  ▼<section class="module ng-scope">
   ▶<div class="pull-left statusMessage">…</div>
   ▼<div class="pull-right">
      <!-- ngIf: !showTESTOptions -->
      <!-- ngIf: showTESTOptions -->
    ▼<span ng-if="showTESTOptions" class="ng-scope">
       <!-- ngRepeat: TESTType in TESTSettings.TESTTypes -->
     ▼<div ng-repeat="TESTType in TESTSettings.TESTTypes" class="btn-group ng-scope">
        <!-- ngIf: !isConnected || TESTType.isReadOnly !== rcContext.isReadOnly -->
        <button class="btn ng-scope ng-binding" ng-attr-id="{{'connectBtn_'+TESTType.isReadOnly}}" ng-if="!isConnected || TESTType.isReadOnly !== rcContext.isReadOnly"
        ng-disabled="isConnectDisabled" ng-click="TESTConnectivityTool($event,TESTType)" id="connectBtn_false">TEST Interactive</button>
        <!-- end ngIf: !isConnected || TESTType.isReadOnly !== rcContext.isReadOnly -->
        <!-- ngIf: isConnected && TESTType.isReadOnly == rcContext.isReadOnly -->
       ▶<button ng-attr-id="{{'settings_' + rcContext.isReadOnly}}" class="btn dropdown-toggle" data-toggle="dropdown" ng-disabled="isConnectDisabled" id="settings_">…
        </button>
       </div>
       <!-- end ngRepeat: TESTType in TESTSettings.TESTTypes -->
     ▼<div ng-repeat="TESTType in TESTSettings.TESTTypes" class="btn-group ng-scope">
        <!-- ngIf: !isConnected || TESTType.isReadOnly !== rcContext.isReadOnly -->
        <button class="btn ng-scope ng-binding" ng-attr-id="{{'connectBtn_'+TESTType.isReadOnly}}" ng-if="!isConnected || TESTType.isReadOnly !== rcContext.isReadOnly"
        ng-disabled="isConnectDisabled" ng-click="TESTConnectivityTool($event,TESTType)" id="connectBtn_true">TEST View Only</button>
        <!-- end ngIf: !isConnected || TESTType.isReadOnly !== rcContext.isReadOnly -->
        <!-- ngIf: isConnected && TESTType.isReadOnly == rcContext.isReadOnly -->
       ▶<button ng-attr-id="{{'settings_' + rcContext.isReadOnly}}" class="btn dropdown-toggle" data-toggle="dropdown" ng-disabled="isConnectDisabled" id="settings_">…
        </button>
       </div>
       <!-- end ngRepeat: TESTType in TESTSettings.TESTTypes -->
      </span>
      <!-- end ngIf: showTESTOptions -->
    </div>
   </section>
  </widget>
 </body>
</html> == $0
```

如果查看“设置”按钮，则两个代码都相似。通过使用传统方式，例如 id，name，value，contains 等，它们都不会起作用。

// * [contains（text（），'Setting'）]，这将产生两个 web 元素。因此它并不是唯一的。

所以这是最终战略，

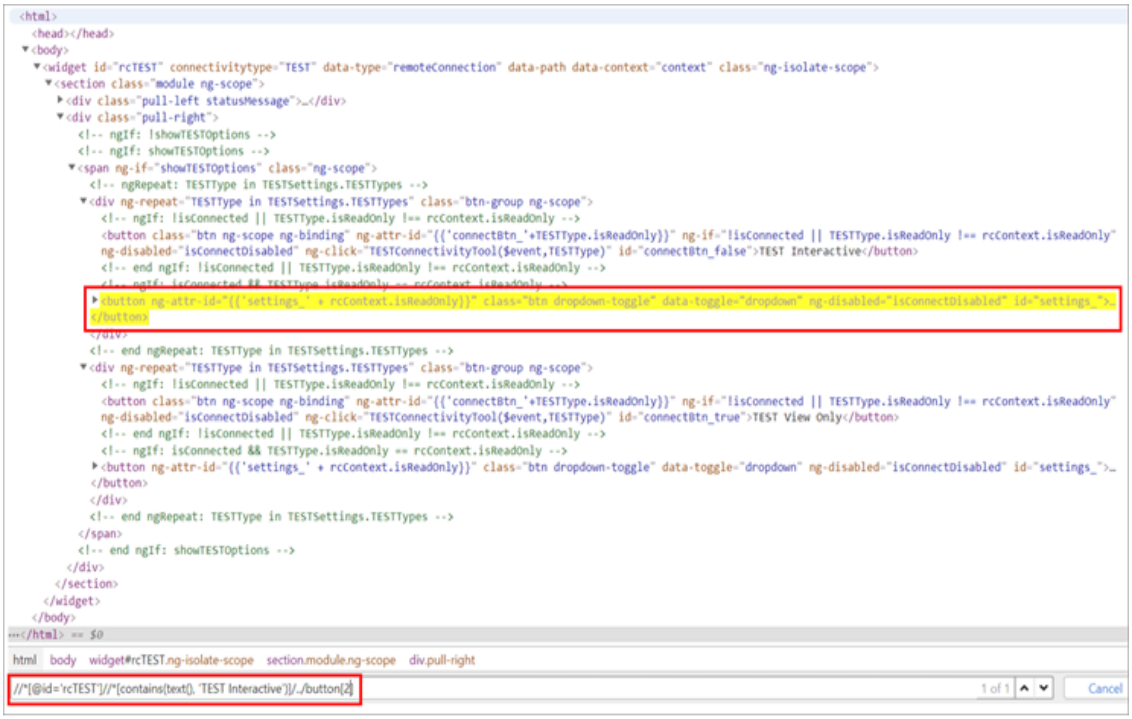
>>首先，找到最接近的唯一标签，在这种情况下，它是<widget id ='rcTest'.....>

```
XPATH : “// * [@ id ='rcTEST']
```

>>其次，找到最接近预期的 Web 元素的 Web 元素，在本例中包含（text（），'TEST Interactive'）。现在我们在<DIV>中存在'Setting'按钮但是要点击它，我们首先需要使用双点转到主<DIV>，如下所示。

XPATH : "// * [@ id ='rcTEST'] // * [contains (text ( ) , 'TEST Interactive')] / ..

>>如您所见，我们处于<DIV>级别，其中第二个 Web 元素为“设置”按钮。这个<DIV>有两个按钮，我们想要转到第二个按钮，即“设置”按钮。通过在末尾添加'/button [2]'，我们可以为“设置”按钮获取我们独特的 XPATH，如下所示



最终 XPATH:

```
"// * [@ id ='rcTEST'] // * [contains ( text ( ) , 'TEST Interactive' ) ] /../ button [2]"
```

如果您认为他们可能会将 Web 元素类型从“按钮”更改为其他内容，那么这是另一种生成方式。

“// * [@ id ='rcTEST'] // * [contains（text（），'TEST Interactive'）] /..//* [contains（text（），'Setting'）]”

或使用“跟随兄弟”

```
"// * [@ id ='rcTEST'] // * [contains ( text ( ) , 'TEST Interactive' ) ] / following-sibling :: button"
```

2）使用变量和自定义值

假设有一个 Web 应用程序具有上传/下载文件的 FTP（“文件传输协议”）功能，您可以通过单击下载链接下载特定文件的测试用例。

首先，我们可以将我们要查找的文件名定义为变量。

String expectedfileName = “Test1”；

现在使用 XPATH 我们可以找到实际的文件名

```
"String actualFileName = WebDriverAccess.getDriver().findElement
(By.xpath("//*"+fileName +"/tr/td[1]")).getAttribute("title");"
```

在上面的 XPath 中，.../ tr / td [1] .getAttribute（“title”）'将转到特定的行和第一列，并获取 title 属性的值。我们可以将实际文件名存储到另一个变量中。

一旦我们同时拥有预期文件名和实际文件名，我们就可以比较两者，如果两者都匹配，我们只需单击其下载链接即可

```
(if acutalFileName == expectedFileName) {
 WebDriverAccess.getDriver().findElement(By.xpath("//*"+fileName +"/tr/td[4]")).click();
}
```

我们还可以在每行中创建一个循环并继续验证文件名，直到找到它为止。

```
Loop(int count < 30)
{
String actualFileName = WebDriverAccess.getDriver().findElement
(By.xpath("//*"+acutalFileName +"/tr[" + count + "]/td[1]")).getAttribute("title");
(if acutalFileName == expectedFileName) {
WebDriverAccess.getDriver().findElement(By.xpath("//*"+fileName +"/tr/td[4]")).click();
}
Count++;
}
```

3）使用“XML”标签，“AND”等

我们可以使用自定义标签生成唯一的 XPATH 并添加其他条件。

例如，假设我们的主要 Web 元素存在于主<address>标记中，并且有多个地址标记，但您只想查找特定的地址标记。所有地址标签都有一个类属性，因此我们可以从中开始。

```
// address[@class='ng-scope ng-isolate-scope']
```

我们注意到我们的 Web 元素位于<DIV>标记中，其中包含一些名为“Testing”的文本。

```
// address[@class='ng-scope ng-isolate-scope']//div[contains(.,'Testing')
```

我们发现结果有多个 web 元素。因此，为了使它更独特，我们可以添加其他条件，例如“id”，它最终将我们指向我们正在寻找的 web 元素。

```
// address[@class='ng-scope ng-isolate-scope']//div[contains(.,Testing') and @id='msgTitle']
```

4）使用属性和表 XPATH

假设我们想要键入放置在表中的 Web 元素，并将表放在表单元素中。

我们可以在 DOM 中找到名为“myForm”的所有表单。

```
“//*[@name='myForm']”
```

现在在所有表单中找到 id 为'tbl_testdm'的表。

```
"//*[@name='myForm']//table[@id='tbl_ testdm']"
```

在表格中，转到特定的行和列。

```
"//*[@name='myForm']//table[@id='tbl_ testdm']/tbody/tr/td[6]/"
```

在单元格内，如果有多个输入，则找到一个输入，其中 value ='Open RFS'，这将为我们提供该字段的最终 XPath。

```
//*[@name='myForm']//table[@id='tbl_ testdm']/tbody/tr/td[6]/ input[@value='Open RFS']"
```

5）使用属性，表和文本

假设您的 Web 元素位于 Panel Table 中，并且有一些常用文本。

首先从具有唯一属性的面板开始，在这种情况下是'TITLE'。

```
//*[@title='Songs Lis Applet']
```

现在浏览所有表标签。

```
//*[@title='Songs Lis Applet']//table
```

在所有表中找到包含文本“作者”的列。

最终的 XPath 将是:

```
//*[@title='Songs List Applet']//table//td[contains(text(),'Author')]
```

6）使用嵌套属性生成 XPATH

也可以使用嵌套属性生成目标 Web 元素的 XPath。例如，在这种情况下，它将在 DOM 中查找特定属性，然后在其中查找另一个属性。

```
//*[@id='parameters']//*[@id='testUpdateTime']")
```

7）通过组合属性，分区和按钮生成 XPath

例如，在下面的 XPath 中，我能够通过使用 id（相对 XPath），一些 div 标签和一个按钮来找到目标 web 元素。

```
"//*[@id='MODEL/PLAN']/div[1]/div[2]/div[1]/div[1]/widget/section/div[1]/div/div[1]/div/div/button[1]"
```

8）XPATH 使用 CONTAINS，REVERSE LOOKUP 等生成

一旦我有一个没有直接识别的下拉菜单。我不得不使用 CONTAINS，REVERSE，DIVs 属性来提出最终的 XPATH，如下所示。

```
//*[contains(text(),'Watch Dial)]/../div/select[@data-ng-model='context.questions
[subqts.subHandleSubId]']"),
```

9）使用 Relative，CONTAINS，REVERSE，FOLLOWING SIBLING 等生成 XPath。

我遇到了应用程序显示图形的情况，每个图形值都必须经过验证。但是，遗憾的是，每个值都没有任何唯一标识，因此我提出了最终的 XPATH，如下图所示，其中包含一个图形值，它结合了相对，包含，反向，跟随兄弟和 div 标记

```
//*[@id='RESEARCH/PLAN']//*[contains(@id, 'A4')]/../../following-sibling::div[1]/div[1]/span[1]/span[1]
```

10）使用 Attributes，Contains，Reverse，Preceding-Sibling，Divs 和 Span 生成 XPath

```
//*[@id='ALARMDATA']//*[contains(@id, 'AFC2')]/../../preceding-sibling::div[1]/div[1]/span[1]/span[1]
```

11）使用属性，XML 标签等。

在下面的 XPATH，属性和 XML 标记中，序列用于提供 Web 元素的最终唯一地址。

```
//*[@id='RESEARCH/REVIEW']
//widget/section/div[1]/div/div[2]/div[1]/div[3]/div[1]//span[@class='details']
```

12）通过不查看整个页面而是查看所有链接并包含而生成 XPath

下面的 XPath 将仅查找整个页面中包含“参数数据手动输入”文本的链接。

```
//a[contains(.,'Parameter Data Manual Entry')]
```

13）使用包含和属性

```
//*[contains(@style,'display: block; top:')]//input[@name='daterangepicker_end']
```

14）使用属性，遵循兄弟姐妹和后代

```
//*[@id='dropdown-filter-serviceTools']/following-sibling::ul/descendant::a[text()='Notepad']
```

15）使用属性，遵循兄弟，后代和文本

```
//*[@id='dropdown-filter-service tools']
/following-sibling::ul/descendant::a[text()='Trigger Dashboard']
```

16）使用标题和文本

如果 web 元素是包含某些特定文本的标题，则 XPath 可能如下所示:

```
//h3[text()='Internal Debrief']
```

17）使用标题文本，兄弟姐妹，路径等

```
//h3[contains(text(),'Helium Level')]/following-sibling::div/label/input
```

18）使用属性，包含和前置兄弟姐妹

```
//div[div[p[contains(text(),'Status')]]]/preceding-sibling::div/div/span[3]/span
```

19）通过使用 Id 属性，一些特定文本和反向查找来查找下拉列表

```
//*[@id='COUPLING']//*[contains(text(),'COUPLE Trend')]/../div/select
```

20）组合“Id”属性并查找具有特定文本的链接

```
//*[@id='ffaHeaderDropdown']//a[contains(text(),'Start Workflow')]
```

结论

在编写杀手级 XPATH 时，它实际上取决于您对代码的理解和分析。您对代码的理解越多，您在编写有效 XPATH 时可以找到的方式就越多。

编写 XPath 的第一步是找到与您的目标 Web 元素最接近的唯一 Web 元素，并使用上面讨论的不同技术（如属性，DIV，跟随，包含等）保持接近。

最后，我们再次这样说，如果您要求开发团队在您感兴趣的所有 Web 元素中添加唯一 ID，那么它将真正让您的生活更轻松。

每当 sprint 周期或新需求开始工作并且团队与新模型共享时，我总是会经历所有模拟并考虑潜在的自动化测试案例，准备一份将使用的所有潜在 Web 元素的列表在自动化测试和准备我自己的 ID。

一旦完成了所有 Web 元素的列表以及我建议的 ID，我将事先将其分享给开发人员以用于开发代码。通过这种方式，我可以通过简化 XPATH 编写战斗来获得唯一的 ID。

下面是编写 XPATH 的不同方法的组合列表:

“//*[@id='rcTEST']//*[contains(text(), ‘TEST Interactive')]/../button[2]”

“//*[@id='rcTEST']//*[contains(text(), ‘TEST Interactive')]/..//*[contains(text(), ‘Setting')]”

“//*[@id='rcTEST']//*[contains(text(), ‘TEST Interactive')]/following-sibling::button”

“String actualFileName = WebDriverAccess.getDriver().findElement(By.xpath(“//*” +fileName +” /tr/td[1]”)).getAttribute(“title”);”

WebDriverAccess.getDriver().findElement(By.xpath(“//*” +fileName +” /tr/td[4]”)).click();

“// address[@class='ng-scope ng-isolate-scope']//div[contains(.,Testing') and @id='msgTitle']”

“//*[@name='myForm']//table[@id='tbl_ testdm’]/tbody/tr/td[6]/

input[@value='Open RFS']”

“//*[@title=’ Songs List Applet']//table//td[contains(text(),'Author')]”

“//*[@id='parameters']//*[@id='testUpdateTime']”)”

“//*[@id='MODEL/PLAN']/div[1]/div[2]/div[1]/div[1]/widget/section/div[1]/div/div[1]/div/div/button[1]”

“//*[contains(text(),'Watch Dial)]/../div/select[@data-ng-model='context.questions[subqts.subHandleSubId']”),”

“//*[@id='RESEARCH/PLAN']//*[contains(@id, ‘A4')]/../../following-sibling::div[1]/div[1]/span[1]/span[1]”

“//*[@id='ALARMDATA']//*[contains(@id, ‘AFC2')]/../../preceding-sibling::div[1]/div[1]/span[1]/span[1]”

“//*[@id='RESEARCH/REVIEW']//widget/section/div[1]/div/div[2]/div[1]/div[3]/div[1]//span[@class='details']”

“//a[contains(.,'Parameter Data Manual Entry')]”

“//*[contains(@style,'display: block; top:')]//input[@name='daterangepicker_end']”

“//*[@id='dropdown-filter-serviceTools']/following-sibling::ul/descendant::a[text()='Notepad']”

“//*[@id='dropdown-filter-serviceTools']/following-sibling::ul/descendant::a[text()='Trigger Dashboard']”

“//h3[text()='Internal Debrief']”

“//h3[contains(text(),'Helium Level')]/following-sibling::div/label/input”

“//div[div[p[contains(text(),'Status')]]]/preceding-sibling::div/div/span[3]/span”

“//*[@id='COUPLING']//*[contains(text(),'COUPLE Trend')]/../div/select”

“//*[@id='ffaHeaderDropdown']//a[contains(text(),'Start Workflow')]”

往期作品：《Appium+Python 实现 APP 启动页跳转到首页》

Docker 领路，走进压力测试的现代化

◆ 作者：王 练

摘要：

Docker 技术为软件开发、测试提供了非常便捷的功能，使用现成的镜像让我们的工作事半功倍。本文针对在项目中进行压力测试使用 Docker 进行说明，重点记录从手工搭建环境，到使用 Docker 提高测试效率，再对 Docker 脚本进行多次优化，最终达到测试效果的过程。

从中可以看到 Docker 带给我们的便利，学习到 Docker 在压力测试中的应用，同时也体现了技术精益求精、永不止步的趋势。

一、Docker 简介

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

对于目前项目的压力测试来说，看重是有很多现成的镜像供我们使用，还有就是在一个物理机上运行的多个容器可以实现隔离。

二、压力测试需求说明

需求说明：项目上需要对网络设备监管系统进行压力测试，测试的指标是系统能否完成 1W 设备的性能指标监控。

需求分析：实际网络运行时，需要对被管设备进行性能采集，指标包括设备的 CPU、内存、温度、端口流量、带宽利用率等指标。采集周期分为 5 分钟、15 分钟、24 小时。

目前的测试需求是：验证系统能否完成 1W 设备的 5 分钟性能指标采集。

当然，如果可以采集 5 分钟，15 分钟、24 小时也不在话下了。

三、从刀耕火种到现代化

在明确了测试需求之后，紧接着需要完成的就是测试设计和环境搭建了。由于无法使用 1W 的真实设备进行测试，具体原因如下:

找不到。1W 台设备需要非常大的精力去申请、借调，而且不是小数目。

不划算。任何研发事件都需要考虑性价比，进行本需求的测试，投入 1W 台实际的设备是不划算的。

没必要。从各种技术方案来看，都没必要使用实际设备进行测试。

所以肯定是需要使用模拟器进行测试。通过之前的技术积累，测试组对 SNMP Simulator 非常熟悉，该工具可以根据需要进行 SNMP MIB 配置，响应需要的 MIB 节点。在本例中，通过监控指标的确认，在单台实际设备上抓包后，即可配置与实际设备回应一致的模拟器。

然而，在实施压力测试的过程中，并不是一开始就想到使用 Docker 的，使用 Docker 也不是一开始就非常顺利的。经历了从刀耕火种到最终实现现代化的过程。

3.1 多台 Windows PC 的刀耕火种时代

当接收到该需求的时候，马上进行测试方案的设计和规划，核心的方法就是使用 SNMP Simulator 模拟实际的设备。具体分为如下步骤。

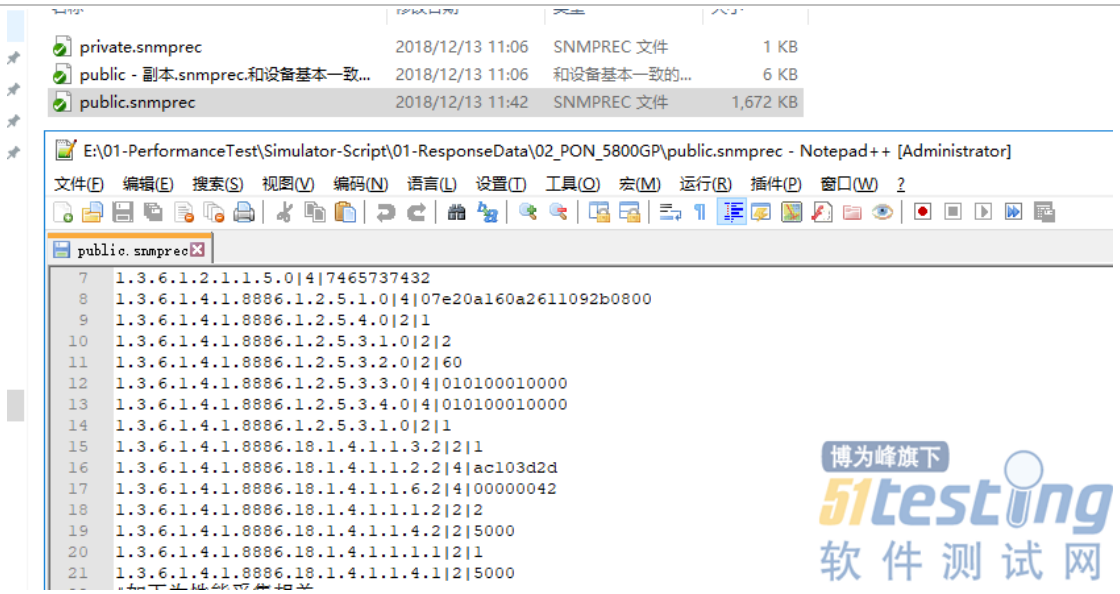
1）安装 Python 环境。使用 01-python-2.7.12.msi 进行 Python 环境安装，完成后添加 path 环境变量，如 C:\Python27;C:\Python27\Scripts。

2）安装 pip 工具。命令如下：cd /d xx\xx\pip-9.0.1，python setup.py install。

3）最后通过 pip 安装 pysnmpsim。命令为：pip install snmpsim，需要确保网络畅通。

4）对实际设备进行抓包。使用 Wireshark 工具对实际设备采集过程进行抓包，对其中的 SNMP 交互进行提取记录。

5）根据抓包结果配置 SNMP Simulator 响应报文。SNMP Simulator 工具会遍历指定的文件夹，搜索“.snmprec”后缀的文件，作为响应报文。且文件名为其读写共同体。

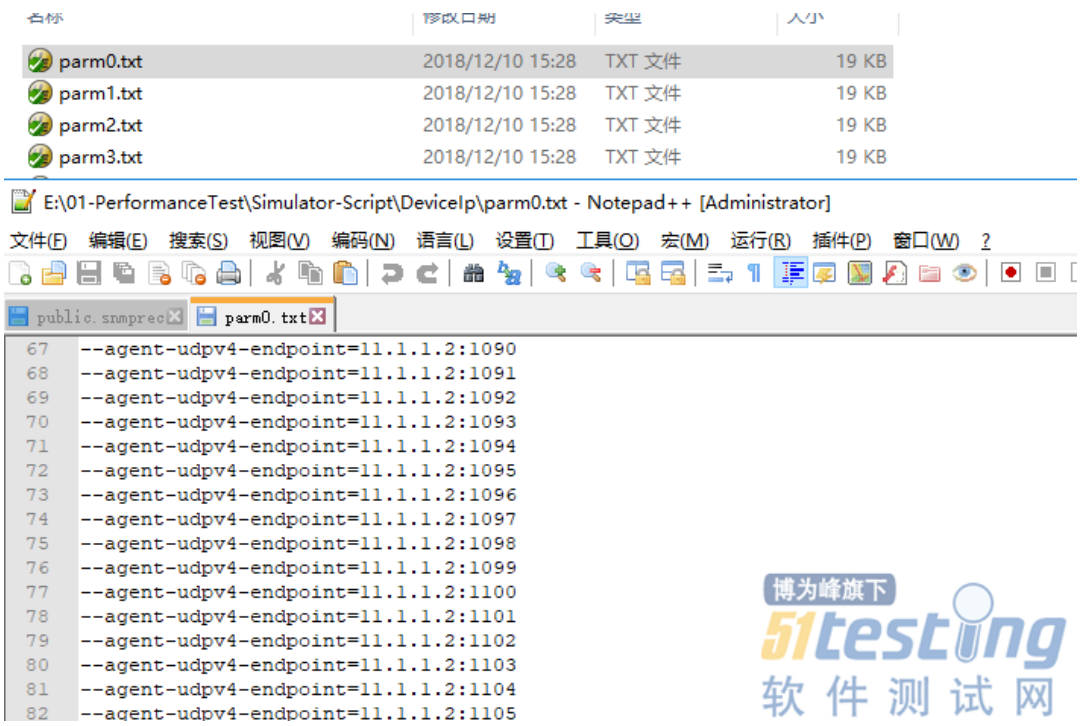


上述文件就是一个参考实际抓包结果进行配置的示例。以第一条为例，当 SNMP Simulator 接收到以“public”为共同体，且 OID 为“1.3.6.1.2.1.1.5.0”的 SNMP 报文时，会回应“7465737432”，其数据格式为“4”（表示 OCTET STRING）。SNMP Simulator 支持配置不同情况响应不同报文，本例不需要。

6）使用命令启动模拟器。由于需要启动非常多的模拟器，使用 SNMP Simulator 提供的文件命令格式，如下：

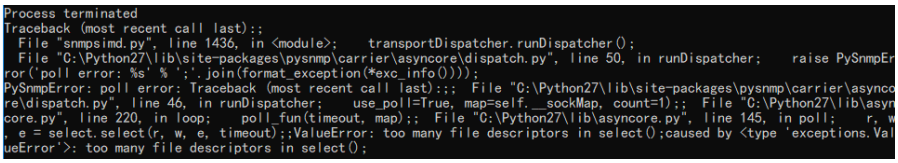
snmpsimd.py --v2c-arch --data-dir=xx\01-ResponseData --args-from-file=xx\Parm.txt

其中 01-ResponseData 目录中存放回应的 SNMP 报文数据，具体格式参考（5）步中的示例。Parm.txt 中存放需要模拟的设备 IP 信息，示例如下：



上述命令意味着在一个 SNMP 模拟器实例中模拟 Parm.txt 指定的 IP+端口的 SNMP 设备。

7）在多台 PC 上重复上述步骤。对于单 PC 模拟器的数目是一个非常重要的指标。实际发现，单进程中最多启动 500 模拟器。超过后会报如下错误。



这个有可能是 SNMP Simulator 的 BUG，但也无暇去解决了。通过开启多个 Python 进程规避。

8）在待测系统中使用数据库存储过程创建资源和任务。1W 设备和相关资源数据通过数据库脚本插入到库中，其中与资源对应的任务信息也一并插入库中。

9）执行测试分析结果。

最终在多个 PC 上完成了 1W 模拟设备的搭建，然而在实际的使用中有如下问题：

1）环境搭建太复杂。首先需要在 10 台 PC 上来回复制安装文件、响应报文文件，之后在 10 台 PC 上进行模拟器启动、查看。

2）环境不易维护。Windows 只能通过远程桌面的方式进行管理，对各个 PC 进行安

装、启动、调试、维护。一旦开启 10 个 MSTSC，就是手忙脚乱，生怕出现一点错误。可以说是一边胆战心惊，如履薄冰，一边是任务繁重，急火攻心。真是冰火两重天！

3.2 多台 CentOS 的原始社会

Windows 下由于远程桌面带来的弊端，直接想到的解决方案就是通过远程批处理脚本完成。然而理想很丰满，现实太骨感：Windos 下非常难以实现！或者用 PowerShell，或者通过编写工具对多个 PC 进行管理，或者在各个机器上安装 SSH。都是得不偿失的不归路。

既然想到了 SSH，那为什么不直接用 Linux 呢？所以最终决定使用多个 CentOS 替换掉 WindowsPC。

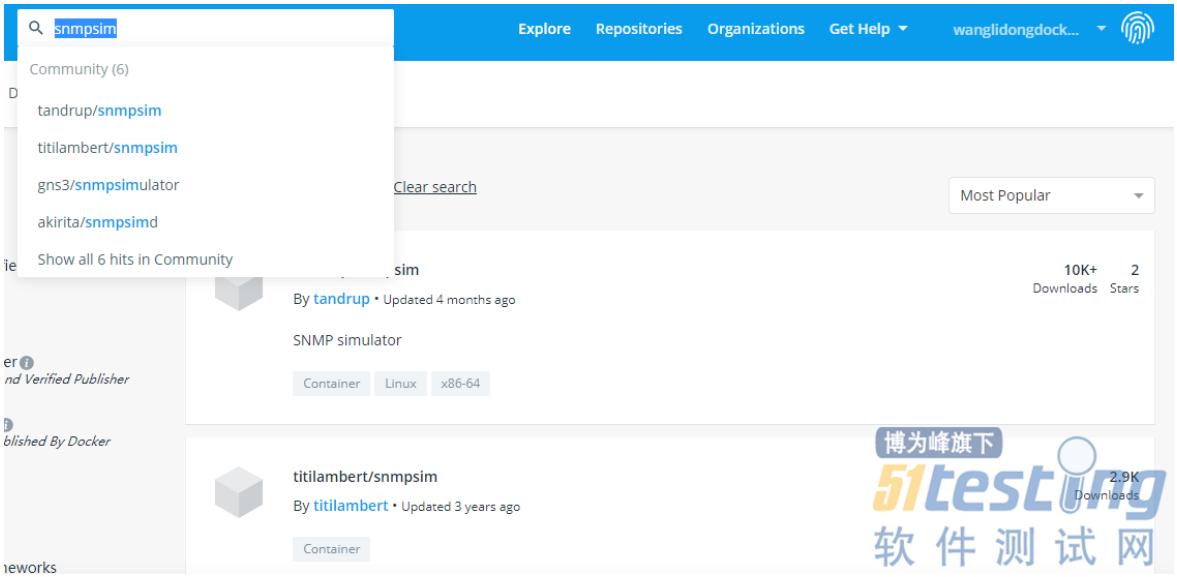
对于上述步骤，将操作系统换为 CentOS。这样通过 Linux 远程工具（SecureCRT　XShell PuTTY 等）可以很方便的对模拟器所在的 PC 进行管理。

事实上，这个方案并没有完全落地。因为一方面可以看到对于流程的改善并没有很多（只是方便了远程操作，装 Python、pip 等步骤都无法省略）；另一方面，Docker 也映入眼帘。

3.3 Docker 带领逐步走进现代化

Docker 进入视线是非常偶然的机会，其他项目组需要协助完成一个使用 Docker 的测试工具的改造，天然的就想到了编写 Docker File，增加功能。在完成该项目组的工作后，忽然受到了启发。此时才想起来还有 Docker 这个利器。既然别人可以用镜像作为测试工具，为什么不写一个 SNMP Simulator 的镜像解决模拟环境搭建的痛点呢？

写一个自己的镜像，多么具有诱惑力的事情呀？然而在编写的过程中，忽然想到是不是有人已经做了？做技术的同学一定要经受住技术本身的诱惑，以效率、务实为出发点。



在 Docker Hub 上搜索结果如上。既然有 10K+下载量的镜像，何必自己费劲呢？于是使用这个镜像开始工作。完整的步骤如下。

1）安装 Docker。参考相关网页安装 Docker，不在详述。

2）对实际设备进行抓包。与之前的步骤一致。

3）根据抓包结果配置 SNMP Simulator 响应报文。与之前的步骤一致，需要将文件拷贝到 CentOS 上。

4）使用命令启动模拟器（实际上是拉取、运行 Docker 的过程）。

```
docker run -d  –v xx\01-ResponseData:/usr/local/snmpsim/data

-p 25001:161/udp    -e EXTRA_FLAGS="--v2c-arch" --name snmpsim_711 tandrup/snmpsim
```

上述命令的含义是：使用 tandrup/snmpsim 的 Docker 镜像（如本地不存在，就在 Docker HUB 上拉取）启动一个名字为 snmpsim_711 的容器。该容器将本地的 xx\01-ResponseData 目录挂载到容器中的/usr/local/snmpsim/data 目录。以本地 25001 端口映射为容器的 UDP 协议 161 端口。容器启动的 SNMP 模拟器仅支持 V2C 版本（--v2c-arch）。

启动后查看如下。

```
[root@host1721661167 ~]# docker run -d -v /root/nview_performance/01_RAX711-L:/usr/local/snmpsim/data -p 25001:161/udp -e EXTRA_FLAGS="--v2c-arch" --name snmpsim_711 tandrup/snmpsim
fcbebe7e0ca185784831caae6c8f225a8823f4e79a8f68cdfee8af414a450f5c
d[root@host1721661167 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
fcbebe7e0ca1        tandrup/snmpsim     "/bin/sh -c 'snmpsim…"   3 seconds ago       Up 1 second         0.0.0.0:25001->161/udp    snmpsim_711
[root@host1721661167 ~]#
```

编写 Shell 脚本启动更多的容器，获得更多的模拟器。脚本如下。

```
snmp_port=25000
for ((i=1;i<=100;i++))
  do
    let snmp_port=snmp_port+1
    docker run -d -v /root/nview_performance/01_RAX711-L:/usr/local/snmpsim/data -p $snmp_port:161/udp -e EXTRA_FLAGS="--v2c-arch" --name snmpsim_rax711_$snmp_port tandrup/snmpsim
  done
```

5）在多台 CentOS 上重复上述步骤。

6）待测系统中使用数据库存储过程创建资源和任务。

7）执行测试分析结果。

总体步骤上节省了两步。不要小看这两步，带来如下的好处:

1、节省了两步，多台机器就是更多的节省。

2、避免了在多台机器安装 Python、pip、SNMP Simulator，也就避免了由于环境问题带来的困扰。这是 Docker 自带的光环，由于使用了现成的 Docker 镜像， 使用的基础环境都是一样的，就免除了这个困扰。

3、CentOS 通过远程工具更方便。无论是 SCP 拷贝文件，还是启动 Docker，都可以通过 Shell 解决，提高了环境的维护能力。

紧接着，需要对能够模拟的模拟设备个数进行分析。可以查看容器占用的资源进行分析。

```
[root@host1721661167 ~]# docker stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
fcbebe7e0ca1        snmpsim_711         0.02%               18.57MiB / 15.27GiB   0.12%               5.05kB / 4.83kB     19.1MB / 168kB      2
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
fcbebe7e0ca1        snmpsim_711         0.02%               18.57MiB / 15.27GiB   0.12%               5.05kB / 4.83kB     19.1MB / 168kB      2
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
fcbebe7e0ca1        snmpsim_711         0.02%               18.57MiB / 15.27GiB   0.12%               5.05kB / 4.83kB     19.1MB / 168kB      2
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
fcbebe7e0ca1        snmpsim_711         0.02%               18.57MiB / 15.27GiB   0.12%               5.05kB / 4.83kB     19.1MB / 168kB      2
```

单个容器占用内存为近 20M，通过限定容器内存的方式明确每台机器开启的容器数目。命令如下。

```
docker run -d -v xx/01_Response_Data:/usr/local/snmpsim/data
-m 20M --memory-swap -1
-p $snmp_port:161/udp -e EXTRA_FLAGS="--v2c-arch"
--name snmpsim_rax711_$snmp_port tandrup/snmpsim
```

其他信息均一致，增加了-m 20M --memory-swap -1，含义是最多占用 20M 内存，不限制 SWAP 空间。

将启动批量容器的 Shell 修改为限定内容的脚本，运行后效果如下:

```
[root@localhost nview_performance]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
b431db2ac4d1        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25100->161/udp    snmpsim_rax711_25100
a828bf841f65        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25099->161/udp    snmpsim_rax711_25099
780f3bc9948b        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25098->161/udp    snmpsim_rax711_25098
d8009e0f77a0        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25097->161/udp    snmpsim_rax711_25097
9fe7e93cb46d        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25096->161/udp    snmpsim_rax711_25096
33c8b9f5ed39        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25095->161/udp    snmpsim_rax711_25095
1684c8860a5c        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25094->161/udp    snmpsim_rax711_25094
b060e4bc1b18        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25093->161/udp    snmpsim_rax711_25093
ce1bbe6fccd6        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25092->161/udp    snmpsim_rax711_25092
8701a5c7ff52        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25091->161/udp    snmpsim_rax711_25091
69a680cf6a49        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25090->161/udp    snmpsim_rax711_25090
08c9b59924ba        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25089->161/udp    snmpsim_rax711_25089
fbd1f5787c1b        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25088->161/udp    snmpsim_rax711_25088
0d68b16b29aa        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25087->161/udp    snmpsim_rax711_25087
7e979f132869        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25086->161/udp    snmpsim_rax711_25086
a052db1b436c        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25085->161/udp    snmpsim_rax711_25085
73240464a1e3        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25084->161/udp    snmpsim_rax711_25084
ab2bf38209e0        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25083->161/udp    snmpsim_rax711_25083
62f1e355359b        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25082->161/udp    snmpsim_rax711_25082
6683698fdceb        tandrup/snmpsim     "/bin/sh -c 'snmpsim欽   4 days ago          Up 4 days           0.0.0.0:25081->161/udp    snmpsim_rax711_25081
```

查看资源占用情况如下：

```
[root@localhost nview_performance]# docker stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
b431db2ac4d1        snmpsim_rax711_25100   0.01%               17.2MiB / 20MiB     86.00%              5.74MB / 784kB      3.1MB / 17.7MB      2
a828bf841f65        snmpsim_rax711_25099   0.01%               19.45MiB / 20MiB    97.23%              5.74MB / 784kB      2.11MB / 8.53MB     2
780f3bc9948b        snmpsim_rax711_25098   0.01%               18.76MiB / 20MiB    93.79%              5.74MB / 784kB      2.29MB / 9.63MB     2
d8009e0f77a0        snmpsim_rax711_25097   0.01%               18.3MiB / 20MiB     91.50%              5.74MB / 783kB      2.74MB / 12.9MB     2
9fe7e93cb46d        snmpsim_rax711_25096   0.01%               18.86MiB / 20MiB    94.32%              5.74MB / 783kB      2.35MB / 11.3MB     2
33c8b9f5ed39        snmpsim_rax711_25095   0.01%               18.89MiB / 20MiB    94.47%              5.74MB / 785kB      1.34MB / 5.87MB     2
1684c8860a5c        snmpsim_rax711_25094   0.01%               19.25MiB / 20MiB    96.27%              5.74MB / 784kB      2.9MB / 18.3MB      2
b060e4bc1b18        snmpsim_rax711_25093   0.01%               18.75MiB / 20MiB    93.73%              5.74MB / 784kB      1.47MB / 8.73MB     2
ce1bbe6fccd6        snmpsim_rax711_25092   0.01%               19.53MiB / 20MiB    97.64%              5.74MB / 782kB      2.44MB / 2.52MB     2
8701a5c7ff52        snmpsim_rax711_25091   0.01%               19.87MiB / 20MiB    99.34%              5.74MB / 784kB      1.65MB / 308kB      2
69a680cf6a49        snmpsim_rax711_25090   0.01%               18.82MiB / 20MiB    94.10%              5.74MB / 785kB      2.55MB / 15.8MB     2
08c9b59924ba        snmpsim_rax711_25089   0.01%               19.9MiB / 20MiB     99.49%              5.74MB / 784kB      2.65MB / 8.88MB     2
fbd1f5787c1b        snmpsim_rax711_25088   0.01%               18.88MiB / 20MiB    94.39%              5.74MB / 783kB      1.53MB / 9.89MB     2
0d68b16b29aa        snmpsim_rax711_25087   0.01%               18.9MiB / 20MiB     94.51%              5.74MB / 785kB      2.17MB / 5.64MB     2
7e979f132869        snmpsim_rax711_25086   0.01%               18.69MiB / 20MiB    93.46%              5.74MB / 783kB      2.47MB / 5.68MB     2
a052db1b436c        snmpsim_rax711_25085   0.01%               18.79MiB / 20MiB    93.95%              5.74MB / 785kB      2.89MB / 9.96MB     2
73240464a1e3        snmpsim_rax711_25084   0.01%               18.98MiB / 20MiB    94.90%              5.74MB / 783kB      2.66MB / 17.3MB     2
ab2bf38209e0        snmpsim_rax711_25083   0.01%               19.82MiB / 20MiB    99.10%              5.74MB / 784kB      2.84MB / 11.9MB     2
62f1e355359b        snmpsim_rax711_25082   0.01%               19.04MiB / 20MiB    95.18%              5.74MB / 783kB      3.04MB / 11MB       2
6683698fdceb        snmpsim_rax711_25081   0.01%               19.08MiB / 20MiB    95.41%              5.74MB / 784kB      2.69MB / 16.2MB     2
df2cb84a5774        snmpsim_rax711_25080   0.01%               18.95MiB / 20MiB    94.77%              5.74MB / 784kB      3.14MB / 12.3MB     2
86bea83a72c7        snmpsim_rax711_25079   0.01%               19.82MiB / 20MiB    99.12%              5.74MB / 784kB      2.62MB / 8.35MB     2
```

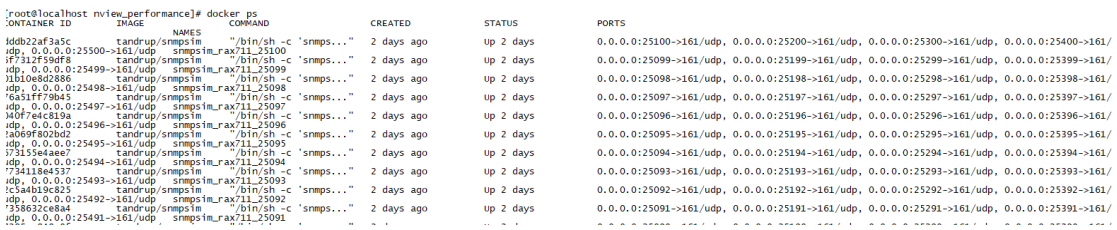
可见 20M 已经够了。

这样对于一台 8GCentOS，以有 6G 可用内存为例，可以启动容器为 307 个。对于 1W 的需求明显是无法满足的。我们需要更进一步。

其实上面的容器启动有个误区，一个容器中仅开启了一个模拟器。而通过在 Windows 上的经验，我们知道可以启动 500 个。我们不需要 500 这么多，10 个就可以解决问题了。所以在容器启动命令中增加多个端口映射。

```
docker run -d -v xx/01_Response_Data:/usr/local/snmpsim/data
-m 20M --memory-swap -1
-p 25001:161/udp
-p 25002:161/udp
…… ……
-p 25010:161/udp
-e EXTRA_FLAGS="--v2c-arch"
--name snmpsim_rax711_$snmp_port tandrup/snmpsim
```

其他内容都是一致的，增加了多个端口映射，意味着将 25001-25010 均开放给外部作为 SNMP 模拟器使用。相应的修改批量启动脚本，最终启动的效果如下。



这样一台 CentOS 上模拟上千台设备就成为可能。我们的压力测试也跟着 Docker 进入了现代化。之后再进行类似的测试，就不会像最开始那样大费周章，而且将会更稳定，更可靠，更易维护。

3.4 测试结果

为了简单起见，我们以 100 设备的采集情况为例进行说明。通过数据记录的查询确定采集数目的正确。在 19：05 查看数据库记录：select COUNT(*) from device_performance;

数目为：596583。

5 分钟后，在 19：10 会进行一次采集，期望的结果是增加 100 条记录，结果如下：596683，符合期望。

我们还可以通过编写查询语句将结果进行直接汇总记数，语句如下;

select count(*),COLLECT_TIME from device _performance GROUP BY COLLECT_TIME ORDER BY COLLECT_TIME desc;

count(*)	COLLECT_TIME
100	2019-04-22 19:20:00
100	2019-04-22 19:15:00
100	2019-04-22 19:10:00
100	2019-04-22 19:05:00

可见在上述 4 个周期，均完成了采集。对于 1W 设备的情况也需要进行相同的验证。

四、总结

当问题得以解决的时候，回顾总是简单的。正如我们向别人介绍某次技术攻关的经历，往往得到的回应是：这很容易想到呀。本次压力测试使用 Docker 的过程也是一样的，实际的过程远比总结出来的困难、不可捉摸。特别是当你沉浸在某一种自认为较为

先进的测试中时，对技术验证的渴望，对自身技术路线的自信，都会让你无暇顾及更优秀的方案。

虽然本次压力测试最终以一种看上去很好的方案解决了，留给我们的思考还在继续。对于走过的弯路、中间的收获，以及后续的计划，总结如下：

1）不要因为方案熟悉就直接上马。之所以在开始直接就上马 Windows 环境搭建，是因为在 Windows 上进行模拟器的搭建是轻车熟路，抓包、配 MIB、开启 SNMP 模拟器……都是非常熟悉的操作。所以非常容易掉进自己的技术温柔乡。

2）对即将解决的问题对比之前的问题进行估计。如何避免上述问题呢？需要对比熟悉方案解决问题和目前即将解决问题的异同。之前进行模拟器搭建是单台设备的模拟，进行 SNMP 接口的验证；而当前的问题是对大量设备的情况进行验证时，情况就复杂了。所以需要评估 SNMP 模拟器的适用性，对大量设备下的 SNMP 模拟器环境搭建难度进行合理评估。

3）不要懒惰。在单台 PC 到多台 PC 扩展的过程中已经意识到环境的费时费力，但由于没有时间，缺少尝新的意愿导致没能引入更先进的技术。这实际是两难的选择，一方面项目紧急需要测试结果，所以事急从权；一方面想想是不是应该先搞定搭建效率的问题，毕竟磨刀不误砍柴工。

4）拿来主义。从 tandrup/snmpsim 镜像的直接使用可以看到，如果有现成的就不要从零开始，毕竟站在巨人的肩膀上，起步会高一些。当然，在解决问题之后，可以去看一下 tandrup/snmpsim 镜像的 Docker File，为后续扩展做好准备。这点上编写一个自己的 Docker File 具有太强的诱惑力了，一定要理性看待技术的诱惑和项目的需要，以项目的进度和务实的效果为出发点，抵抗技术对你的诱惑。

5）精益求精，永不止步。技术和人一样，一定要保持进步才能有竞争力。当前的方案确实能够解决当前的问题，但可以从自动化、易用性上继续优化。同时，通过扩展 tandrup/snmpsim 镜像的 Docker File 增加个性化的需求，更能体现项目的自身特点。也满足了编写一个自己的镜像的愿望！

往期作品：《TestNG 的依赖注入详解和使用场景分析》

深入研究可持续性的测试自动化

◆译者：Alice

概述：当说到实现可持续的测试自动化时，拥有一个合适、恰当的测试自动化团队结构是应当采取的最重要的首要步骤。这篇文章有一些已经证实的适合一些不同的测试自动化场景的惯例--由自动化团队领导或回归测试团队带领，加上敏捷的适应性--已经帮助很多机构享受到了长期的测试自动化成功的成果。

启动新型测试自动化措施的公司--任何参与其中售卖给他们相关的工具的销售人员---倾向于认为他们的成功依赖于一次毫无瑕疵的发布。作为一名测试自动化顾问，我想提供一个基于在这个领域里看到的真实检查。初期的发布可能是一条崎岖的道路如果你没做准备的话，但是长期来看，那并不是将要促成或打破你的测试自动化的发布的第一步。

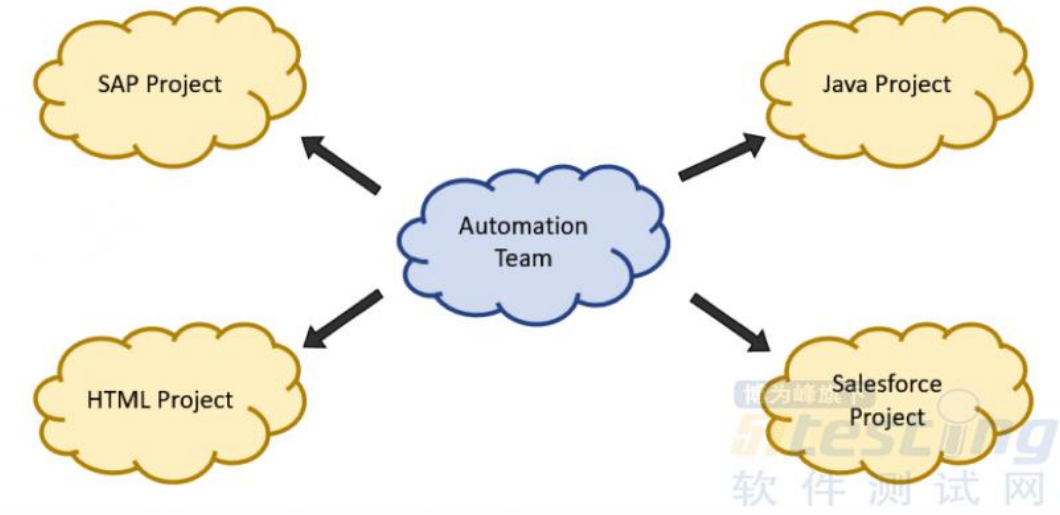
我最近目睹了几个开始不用或用很少准备在公司内部推动测试自动化的几个公司。采用这种“策略”的风险是测试人员可能基于以下原因抵制测试自动化：

- 他们感到没有自动化也挺好
- 他们不想或没有时间去学习新的工具
- 他们担心自动化太复杂，去追赶所花的时间和精力会超出预期的收益

很差劲的计划或者不做计划的推动显然不是启动测试自动化准备工作第一步的最好的方式。但是，我已经发现从困难重重的现有推动中复原比从不做恰当、合适的测试自动化团队结构来获取可持续的测试自动化总是更容易。

我这里分享些适用于好多不同的测试自动化采用场景很好用的做法，它们已经帮助很多机构享有长期的成功---甚至当初始推动并不理想。

自动化团队以项目为单位推出测试自动化



我能给出的最好的建议是从小事开始做起，然后再大做事。被证实好用的首个做法是从组建一个自动化团队开始。

这个队伍将在不同的项目上迭代，并从最重要的测试用例开始自动化。这让你能展示项目的进展，并给你一些程序变更如何影响关键的程序功能上有价值的见解。

我推荐从最有影响力的测试用例开始：那些覆盖了头条商业风险的用例。这为创建一个强大的回归模型创造了基础，好让其提供快速的对程序变更时候破坏了之前正常运行的功能的反馈。

自动化团队应当包括至少一名测试设计专家，做评审需求工作，和使用测试用例设计方法来选出应当进行自动化的测试用例。然后，测试设计专家或其中两三个自动化专家可以评估是否需要测试数据管理，然后他们可以合作来共同定义测试用例。同时，测试设计专家将着手下个项目。

取决于你希望测试的软件和希望多大程度的个性化，可能会需要一个自动化工程师。他们会确保个性化控制的创建以及后续的维护。

队伍的大小将随着时间推移而增加，取决于此队伍必须处理的项目和其中的测试用例个数的多少。这些建议只适用于起步阶段的启动。一旦项目推动开始启动，你会希望尽快招募更多的人加入队伍。

当测试用例自动化后，他们可以在夜间运行在测试虚拟机器上，接着以天为基础汇报执行结果。将自动化测试程序启动和运行最重要的事就是实际去跑测试用例。如果你没在跑测试用例，你就不会从他们那儿接收到任何有价值的东西，不管你的测试集准备

设计得多好，也不管它们如何充分地覆盖了首要的商业风险。

取决于你公司的结构（敏捷与否，有测试队伍还是独立的测试人员），自动化队伍或许要带头教会每个人怎样应用自动化最佳做法以及当他们开始自己的测试自动化时应当考虑的项目变量。

在测试人员开始运用测试自动化之前，必须做个决定：谁为这些测试用例负责任？这很关键。如果自动化团队或者也许是后期的回归团队来负责，那么他们必须与测试人员和业务方的增多沟通。如果是测试人员的责任，他们需要时间确保所有的测试用例正在运行并且符合业务预期结果。每个机构需要决定什么是最好的，取决于他们的结构和工作风格。

不管你选哪个方向，一定保存足够的沟通时间。一方面，新的测试用例需要由回归团队评审然后启动来执行。另一方面，回归测试结果尤其是失败的测试用例需要分享出去以便负责的团队成员或测试人员可以评审他们并依据需要进行更新。至少，确保测试人员独立检查回归测试结果，并做出合适的调整这样回归团队能够推送从共享文件夹推送新的测试用例然后独自评审这些用例。

每个项目的回归团队领导和带教测试人员

我见到的最有效率的启动方法是建立一个初期自动化团队，将回归测试团队引入进来，当测试自动基础建立好以后其他人就准备好去向前执行即可。

此回归团队仍然对在项目初期创建测试用例负责，维护已有测试用例，跑测试用例，护卫测试基础设施和忽略更广的测试自动化准备活动和项目负责。你可以称这样的团队是出色测试中心。

有这样的准备工作，就有可能让项目和大的变更通过严格的测试设计专家的评审，他们将着手用正确的格式和提供一个项目应当看起来是什么样子的轮廓。他们像软件架构师或者在我们这里是测试架构师。

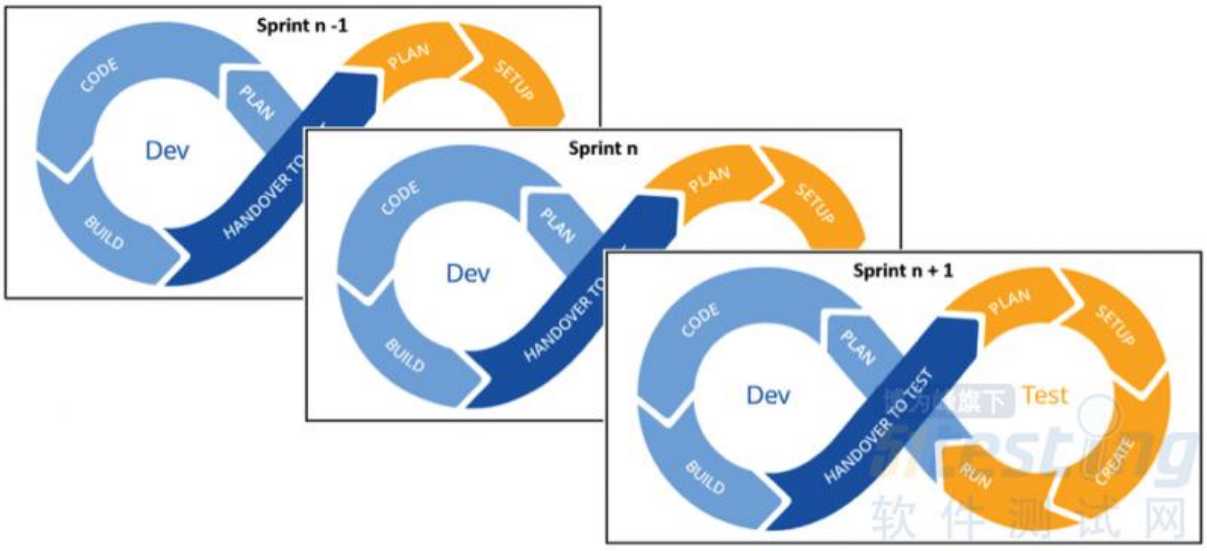
这个回归测试团队即为“吸纳组”如果你有相关问题或者你想驱动新的创新的话。所有权和责任应当是回归测试团队因为他们对整个测试模式有最好的概览。

从资源的角度来看，你需要回归测试团队里有高级技能人员，但是你也可以不让不太有经验的成员去做测试。如果测试人员需要比培训所能提供的知识更多的知识时，他们总可以询问回归测试团队，他们可以分享他们已经构建起来的知识技能来帮助他们。

适应敏捷环境

对于敏捷环境来说，测试自动化是在团队内由敏捷测试人员所驱动的。这个人应当比自动化专家还要技能高级--他们需要理解测试用例设计，需求，测试用例创建和执行及核心测试最佳惯例，并且他们需要对测什么和怎样有效地测试有较好的理解。

我推荐开发和测试的持续启动，从一个冲刺到另一个冲刺：



当开发工作在冲刺 n-1 的功能点完成，测试工作开始。同时，开发工作可以继续在冲刺的功能点 2 上继续进行，等等。只要确定留出时间来修复缺陷和做进一步的改进完善即可。

我们将开发和测试分开来了吗？

最终，我想说几个问题，他们经常出现在有关结构的情况下：是我们的开发人员做测试，还是应当将开发工作和测试工作分开来做？应当在测试工作中使用软件开发工程师？

理论上来说，你可以节省时间和金钱让开发人员也去做测试工作，因为开发人员最懂代码。但是还是诚实一点：开发人员永远不会有足够的时间来完成所有的每个人想要他们完成的开发任务。如果他们也负责做测试自动化的话，那将意味着他们没有更多时间来完成这些开发任务。甚至如果说开发人员最懂代码，你确定他们将检车边缘条件吗？预测风险，并试着去问正确的问题？探测系统并将之推送成一个败品？

我的看法是不写代码的人更适合做这项工作。而他们中的其他人可以聚集在实现实际的程序上的业务目标之上，并尝试去思考每个可能导致系统中的错误的操控的可能

性。从我在此行业里所看到的来说，那并不总是开发人员的优势。

总结

所有这些建议是建立在我个人的帮助用户采用测试自动化的经验之上。我知道每个读者可能对此有不同的看法---所以我鼓励你留言分享与之有关的想法。什么对你帮助最大？你们公司是怎样建立测试自动化的？遇到过什么挑战？

压测必经之路 ,解读 JMeter 分布式

◆ 作者：王 练

摘要： JMeter 是当前 Web 性能测试中应用最为广泛的工具，简洁强大的界面，开源免费的授权，以及广泛的插件扩展，使得 JMeter 能满足几乎所有 Web 场景的性能测试。然而，单机性能的限制，是 JMeter 一直以来最大的诟病。由于采用 Java 多线程进行并发用户的模拟，使得线程数的增加自然增加了测试机的资源消耗。一边是被测系统并发数的日益提高，一边是 JMeter 单机性能的掣肘。测试人员仿佛是走钢丝的杂技演员，平衡木的一边是并发数，一边是测试机的资源，战战兢兢、小心谨慎的想找到其中的平衡。

实际上 JMeter 提供了一种分布式压测的方法提高并发能力。本文介绍如何配置一个 JMeter 的分布式测试环境，并对单机和分布式的测试机资源占用情况进行对比。最后，对分布式的线程数、并发机制以及影响分布式 TPS 的采样信息回送模式进行说明。

一、引入 JMeter 分布式

JMeter 是 Web 性能测试中的利器，基本属于 Web 压测的事实标准。然而使用过 JMeter 的测试人员会发现，当并发用户增加后，JMeter 本身的性能也会急剧下降，导致无法对被测系统施加压力。

以一个实际环境为例，系统配置为：Windows 7 旗舰版，i5-4590 单 CPU4 核 4 线程，4G 内存。使用 JMeter3.1，Java8 进行测试。

当并发线程设置为 1000 时，资源占用如下图。

继续增加线程个数为 1500，此时运行出现错误如下。

```
Writing log file to: E:\07-测试工作\05-MSP相关\03-性能测试\01-测试工具\01-apache-jmeter-3.1\bin\jmeter.log
Creating summariser <summary>
Created the tree successfully using JMeter_Distributed_Demo.jmx
Starting the test @ Wed Jun 26 17:05:46 CST 2019 (1561539946908)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
Uncaught Exception java.lang.OutOfMemoryError: unable to create new native thread. See log file for details.
#
# There is insufficient memory for the Java Runtime Environment to continue.
# Native memory allocation (malloc) failed to allocate 32756 bytes for ChunkPool::allocate
# An error report file with more information is saved as:
# E:\07-测试工作\05-MSP相关\03-性能测试\01-测试工具\01-apache-jmeter-3.1\bin\hs_err_pid5700.log
errorlevel=1
请按任意键继续. . .
```

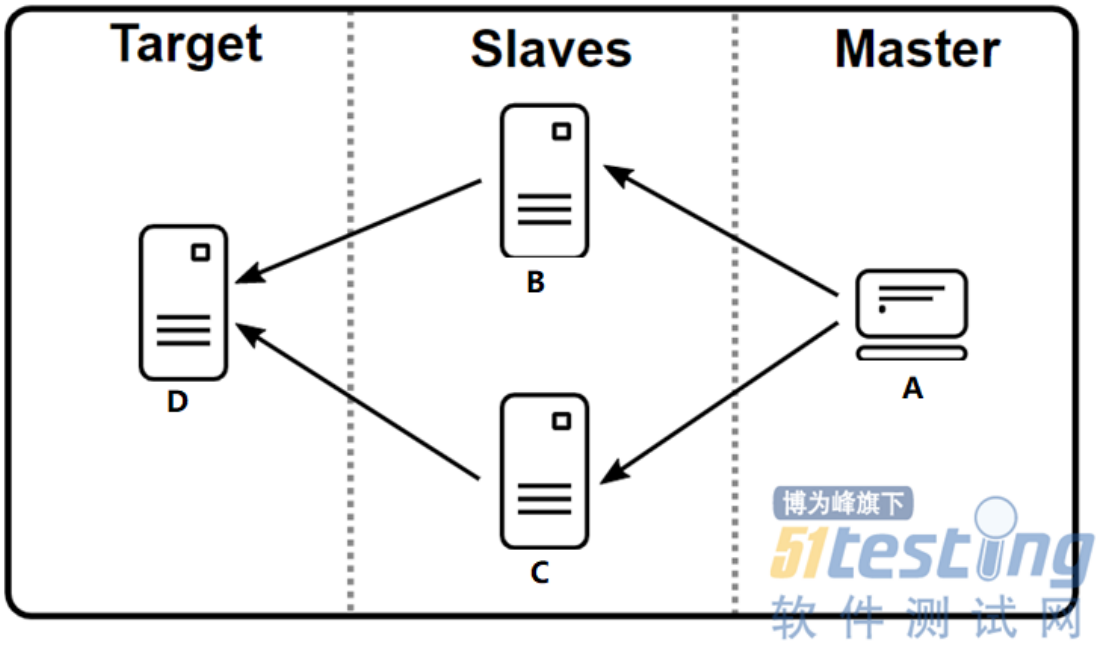
查看日志中的记录，可以看到由于内存不足，无法启动新的线程。

```
2019/06/26 17:05:47 ERROR - jmeter.JMeter: Uncaught exception:  java.lang.OutOfMemoryError: unable to create new native thread
    at java.lang.Thread.start0(Native Method)
    at java.lang.Thread.start(Thread.java:714)
    at org.apache.jmeter.threads.ThreadGroup.start(ThreadGroup.java:306)
    at org.apache.jmeter.engine.StandardJMeterEngine.startThreadGroup(StandardJMeterEngine.java:483)
    at org.apache.jmeter.engine.StandardJMeterEngine.run(StandardJMeterEngine.java:396)
    at java.lang.Thread.run(Thread.java:745)
```

如果我们需要更多的线程并发，此时就必须使用 JMeter 的分布式压测了。

二、环境搭建

遵照 JMeter 官网的方法，很容易搭建适合自己需要的分布式环境。以下图的实际环境为例，测试机 A 作为压测的协调主机（即上述的机器），B 和 C 作为执行机，实际完成压力的发送，被测系统位于 D 上。



具体配置过程如下：

1、执行机属性配置。配置执行机 RMI 服务的端口，修改\apache-jmeter-3.1\bin 下的 jmeter.properties，将如下属性修改为指定的端口：server_port 和 server.rmi.localport。本例中 B 均修改为 1039，C 均修改为 1058。如下图：

```
# RMI port to be used by the server (must start rmiregistry with same port)
server_port=1039
server.rmi.localport=1039
```

```
server_port=1058
server.rmi.localport=1058
```

2、启动执行机的监听服务。在执行机上启动服务，打开\apache-jmeter-3.1\bin 下的 jmeter-server.bat 即可。启动图如下。

```
E:\07-测试工作\05-MSP相关\03-性能测试\01-测试工具\01-apache-jmeter-3.1\bin>jmeter-server.bat
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: E:\07-测试工作\05-MSP相关\03-性能测试\01-测试工具\01-apache-jmeter-3.1\bin\jmeter-server.log
Using local port: 1039
Created remote object: UnicastServerRef [liveRef: [endpoint:[172.16.61.39:1039](local),objID:[7ed169f5:16b9350e0c2:-7fff
, 4383964726854532120]]]
```

```
D:\apache-jmeter-3.1\bin>jmeter-server.bat
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: D:\apache-jmeter-3.1\bin\jmeter-server.log
Using local port: 1058
Created remote object: UnicastServerRef [liveRef: [endpoint:[172.16.61.58:1058](local),objID:[-ed9ad0f:16b98ecc26b:-7fff, -2883123306425262410]]]
```

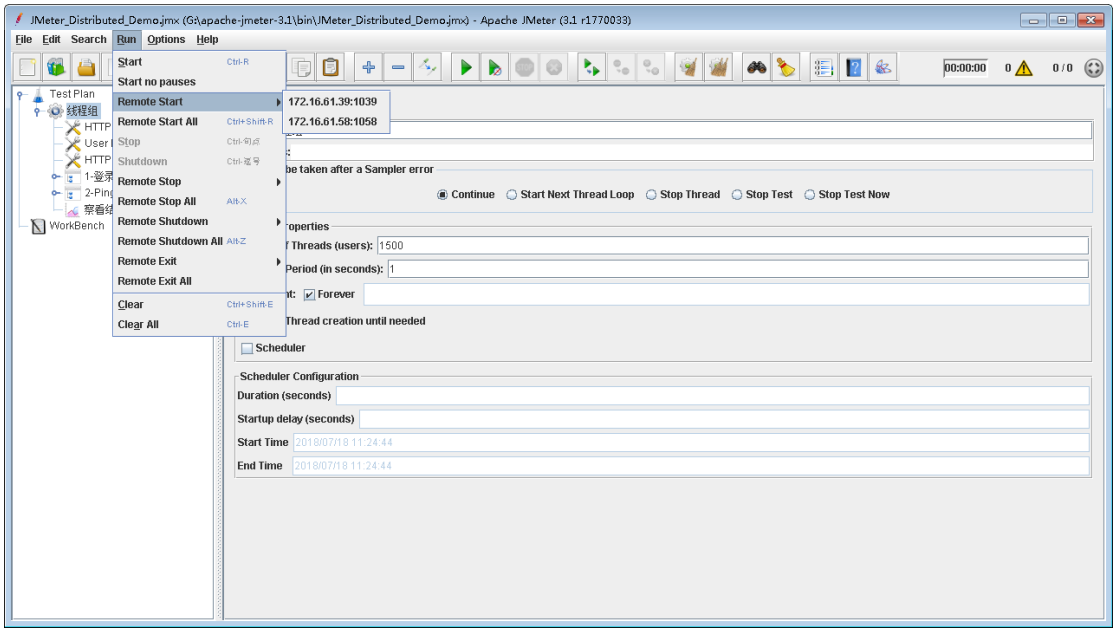
说明在 B 的 1039 端口，C 的 1058 端口均启动了执行的服务，等待协调主机的命

令。

3、协调主机属性配置。配置协调主机连接的执行机和端口，修改\apache-jmeter-3.1\bin 下的 jmeter.properties，将 remote_hosts 属性修改为需要连接的执行机 IP 和端口。如下图。

```
#remote_hosts=localhost:1099,localhost:2010
remote_hosts=172.16.61.39:1039,172.16.61.58:1058
```

4、执行测试。启动协调主机的 JMeter，通过界面进行启动。

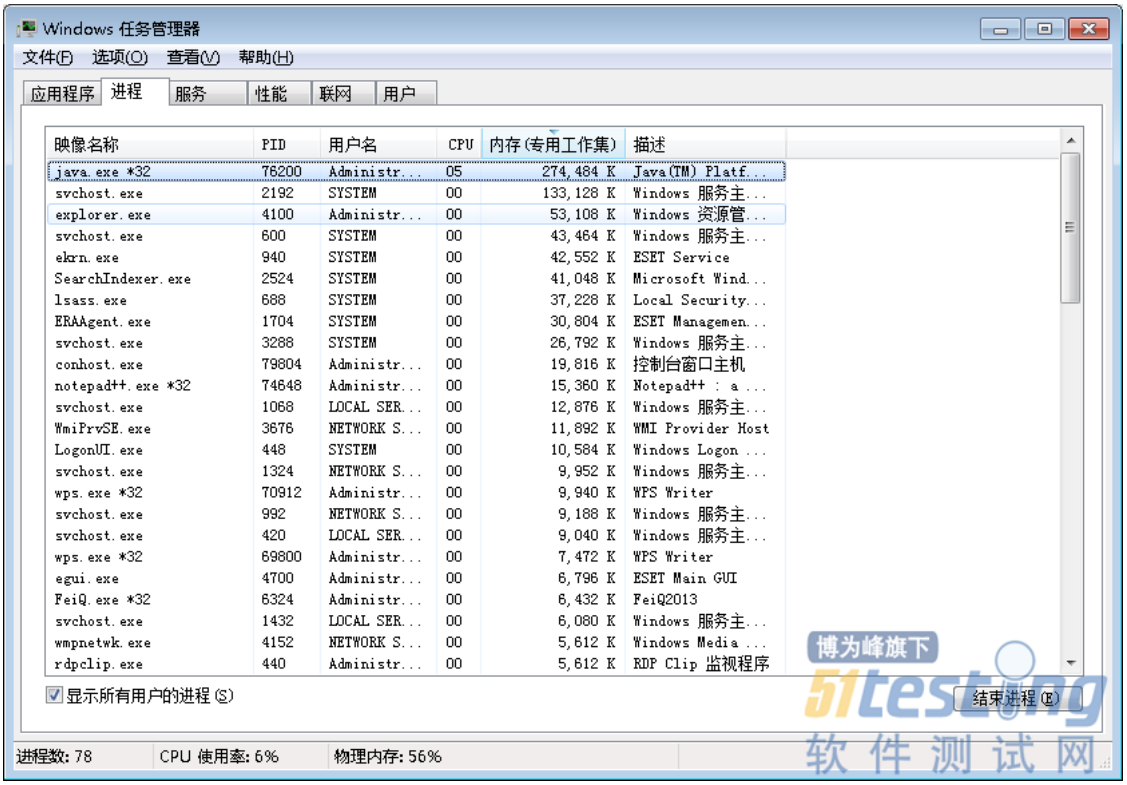


如上图，通过 Run->Remote Start 可以启动单个执行机，通过 Remote Start All 启动全部执行机。

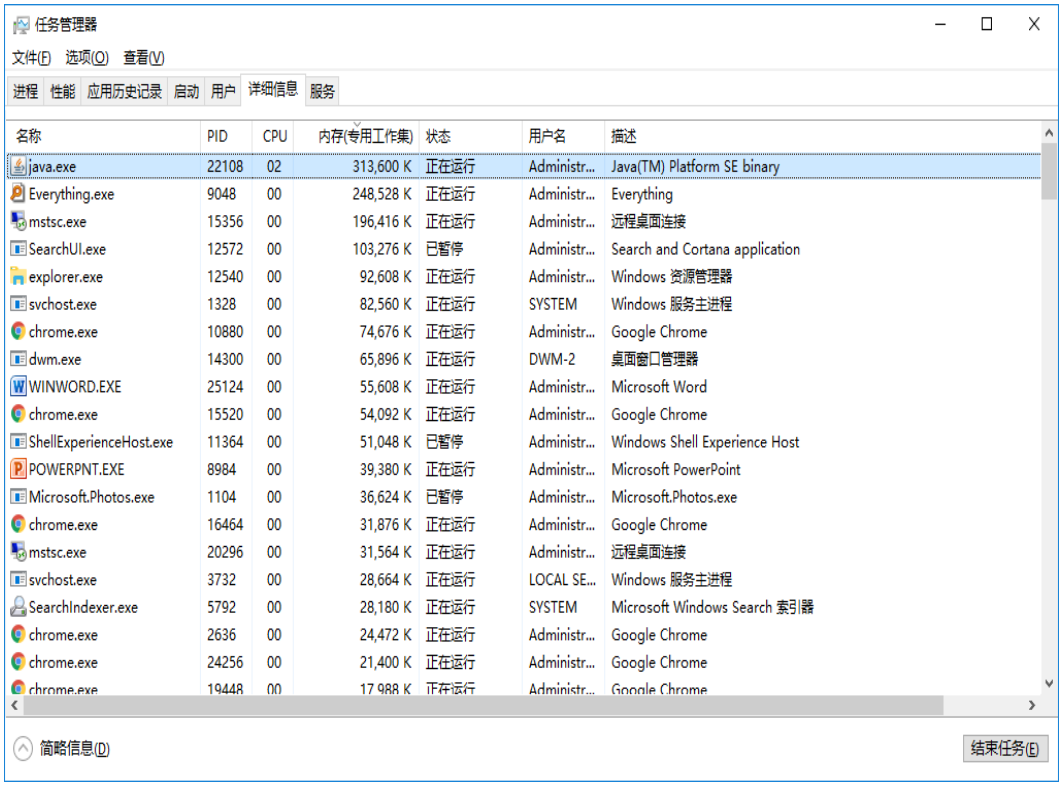
3、资源对比

在前面我们看到，单台如上配置的测试机在启动 1500 线程时会出现内存不够用的情况。那么分布式的情况如何呢？

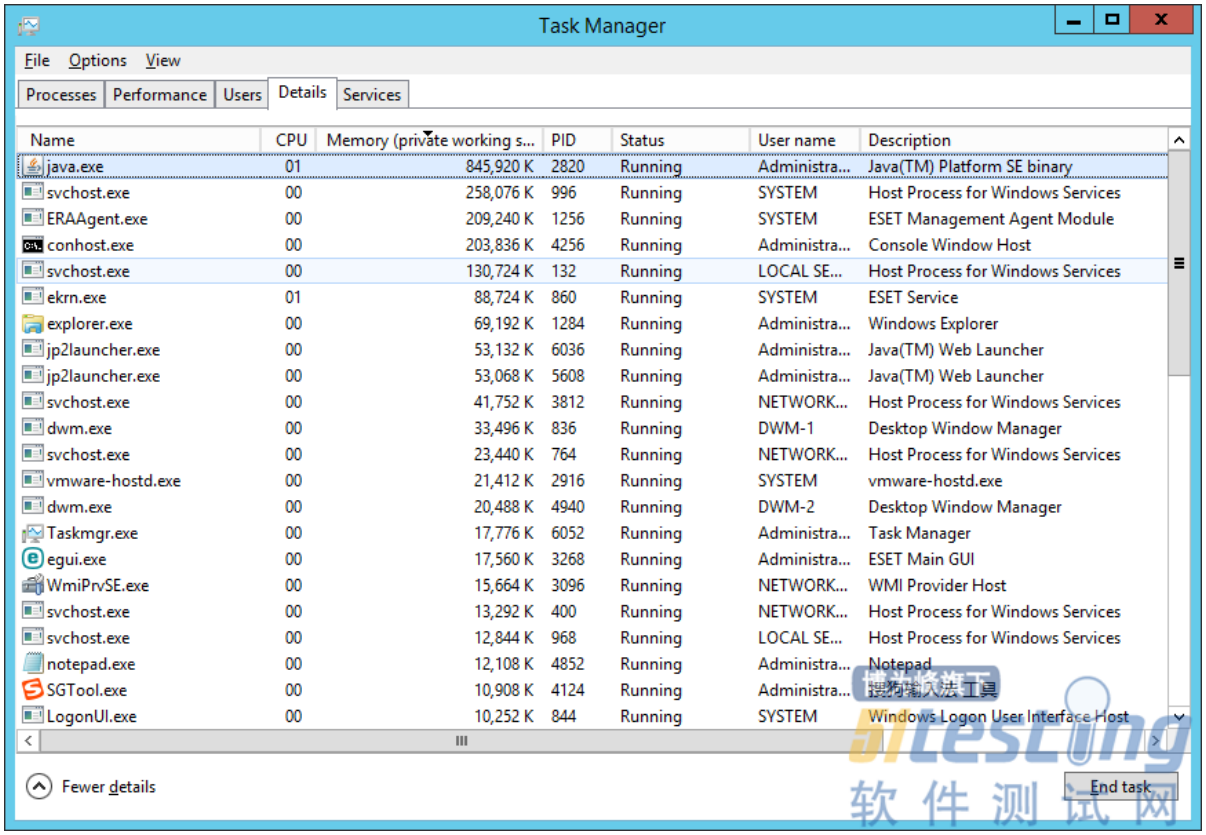
下图为协调主机的资源占用情况。



其中一台执行机的资源占用情况，执行机 B，Windows10 教育版，i5-6500 单 CPU4 核心 4 线程（3.20GHz），8G 内存。



另外一台执行机的资源占用情况，执行机 C，配置为 Windows Server 2012 R2 Standard，Intel Xeon 单 CPU6 核 12 线程。



4、注意事项

通过 JMeter 分布式进行压测，可以避免单机测试机资源的限制，提高压测的并发线程数。然而，还有一些细节需要注意。

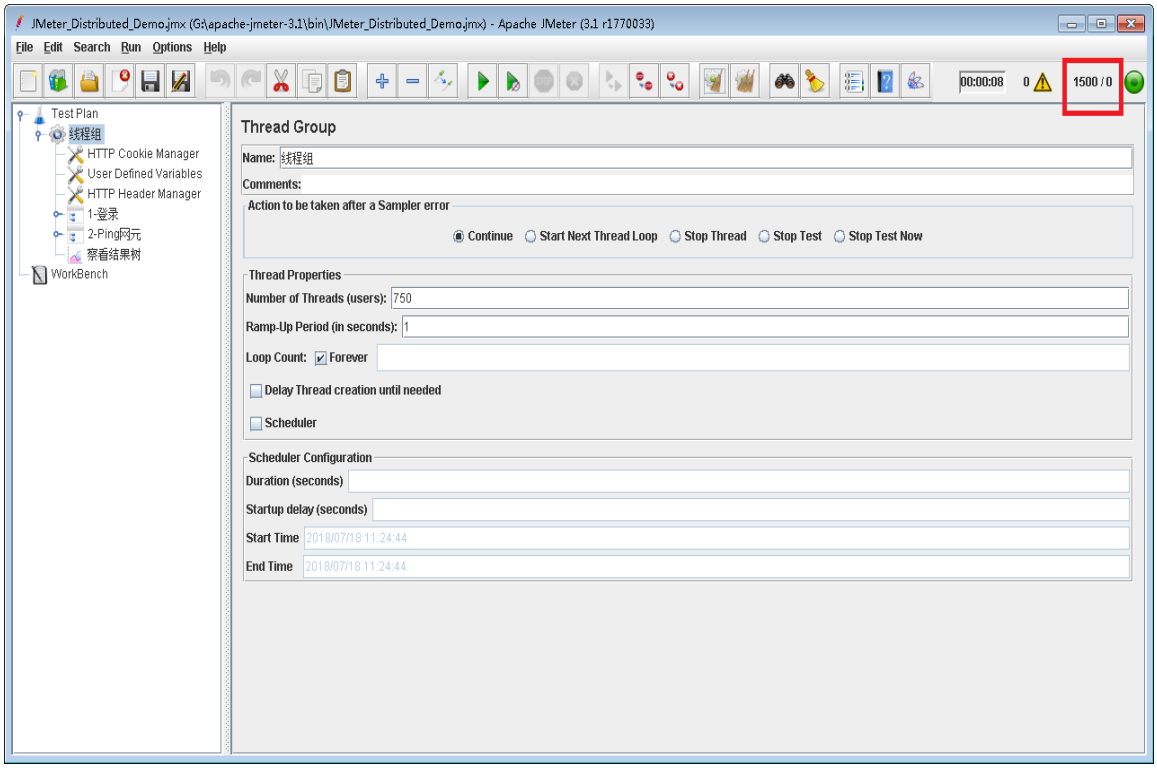
1、线程数。JMeter 分布式测试，是通过网络连接将协调主机载入的脚本分别传递（复制）给执行机，也就是说每个执行机拿到的脚本都是一致的，所以在每台执行机都会启动脚本中线程组指定的并发线程数。这样在设定脚本线程数目时，需要除以执行机个数，设定并发线程数。

以上述示例进行说明，期望完成 1500 用户并发，在单机测试时脚本设置并发线程为 1500，而在有两条执行机的情况下需要设定为 750，脚本修改如下。

```
<stringProp name="ThreadGroup.num_threads">1500</stringProp>
<stringProp name="ThreadGroup.ramp_time">1</stringProp>
<longProp name="ThreadGroup.start_time">1531884284000</longProp>
<longProp name="ThreadGroup.end_time">1531884284000</longProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.duration"></stringProp>
<stringProp name="ThreadGroup.delay"></stringProp>
```

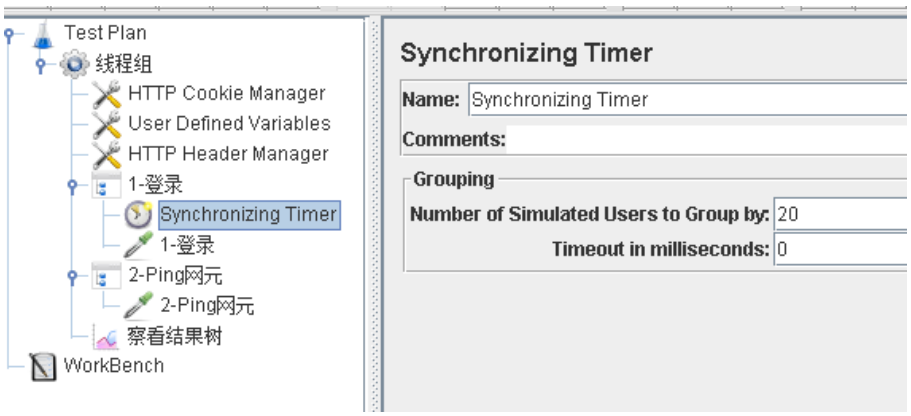
750

此时启动远程测试 Remote Start All 后，看到界面的显示如下。



其中的 1500/0，含义是一共启动了 1500 个线程，本机为 0。

2、同步定时器的使用。我们知道，同步定时器（Synchronizing Timer）的意义在于提供瞬间压力的测试，其原理是阻塞期望个数的线程（用户），在同时进行释放，尽可能真实的模拟并发用户同时进行某操作的情况。那么在分布式并发的情况是如何的呢？我们将上述例子简化为 20 用户并发，即每个执行机 10 个线程，同时增加一个同步定时器，期望 15 个线程瞬间压测。脚本修改如下。



执行远程测试 Remote Start All，监控结果树的情况。会发现没有任何请求被发送。这是因为，同步定时器仅在一个 JVM 中起作用，而分布式环境下属于两台机器的两个独立 JVM，同步定时器无法生效。此时对于每个执行机，均启动了 10 个线程，但获得的脚本中需要等待 20 个线程再释放，所以两个执行机均不会向下执行。

> Synchronizing timer blocks only within one JVM, so if using Distributed testing ensure you never set "Number of Simultaneous Users to Group by" to a value superior to the number of users of its containing Thread group considering 1 injector only.

JMeter 官网原文如上，含义就是同步定时器阻塞线程的机制仅在一个 JVM 中，所以在分布式的情况下，设定的阻塞线程数不能超过每个执行机的并发线程数。本例中就是不能超过 10 线程。

类似的，高斯随机定时器（Gaussian Random Timer），固定定时器（Constant Timer），均匀随机定时器（Uniform Random Timer），泊松随机定时器（Poisson Random Timer），BeanShell 定时器，BSF 定时器，JSR 定时器。由于是针对单线程的，所以不受分布式的影响。

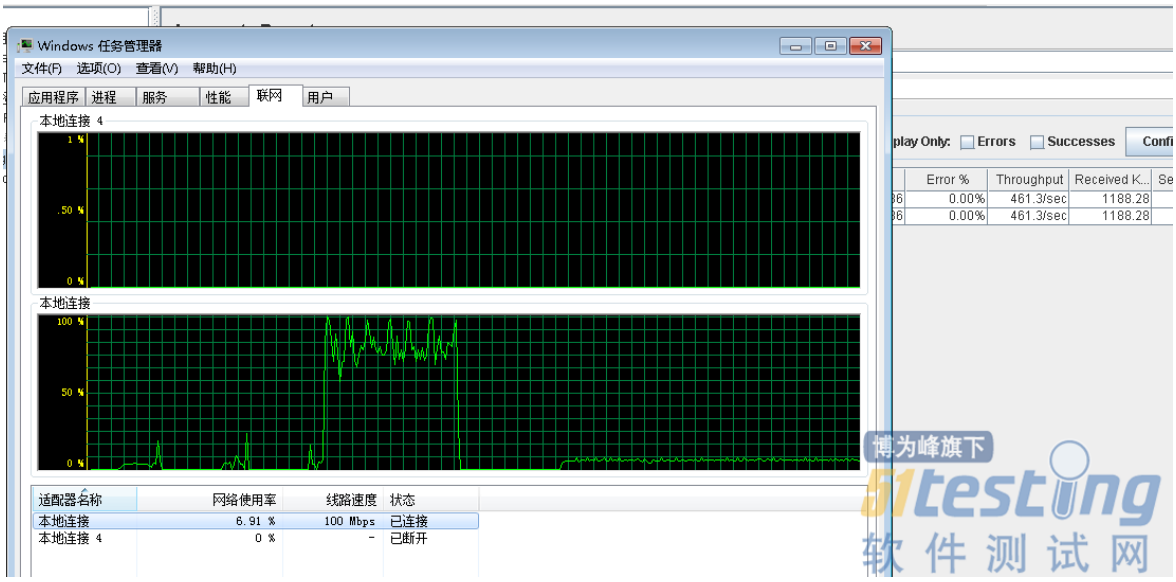
当然，吞吐量定时器也是对于每个执行机独立进行限制的。也就是说，设置的吞吐量限定值，由于脚本是分别在每个执行机进行运行的，所以限定的也都是当前作用的执行机。

3、执行机测试结果回送方式。JMeter 分布式的原理十分容易理解，在实际测试中就如同蝴蝶效应一样，一点细微的差别都会导致最终加压结果的巨大差异，从而导致最终测试结果的不确定性。在分布式测试中，执行机测试结果的回送方式影响非常大。

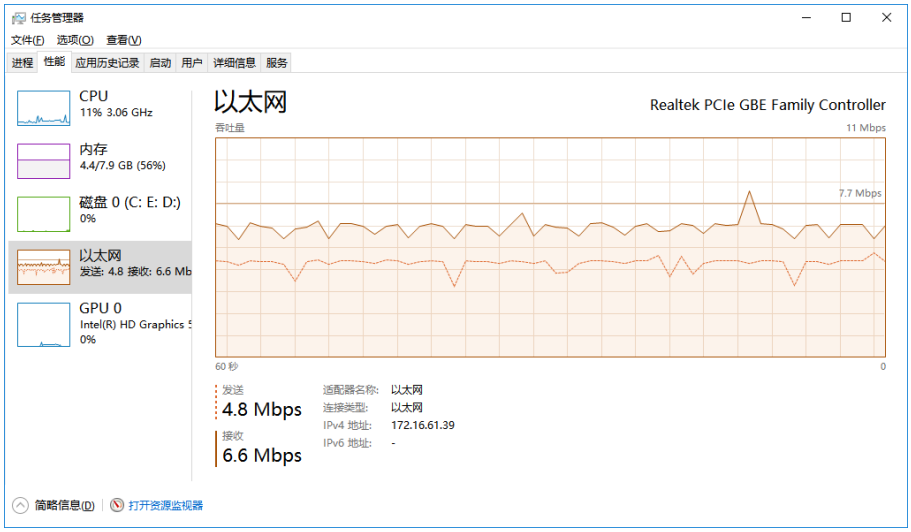
首先我们对比一下实际的测试结果。在协调主机和执行机上的配置文件\apache-jmeter-3.1\bin 下的 jmeter.propertie 中，修改 mode=StrippedBatch，如下截图。

```
#Set to true to key statistica
#key_on_threadname=false
#mode=Stripped
mode=StrippedBatch
#mode=org.example.load.MySampl
```

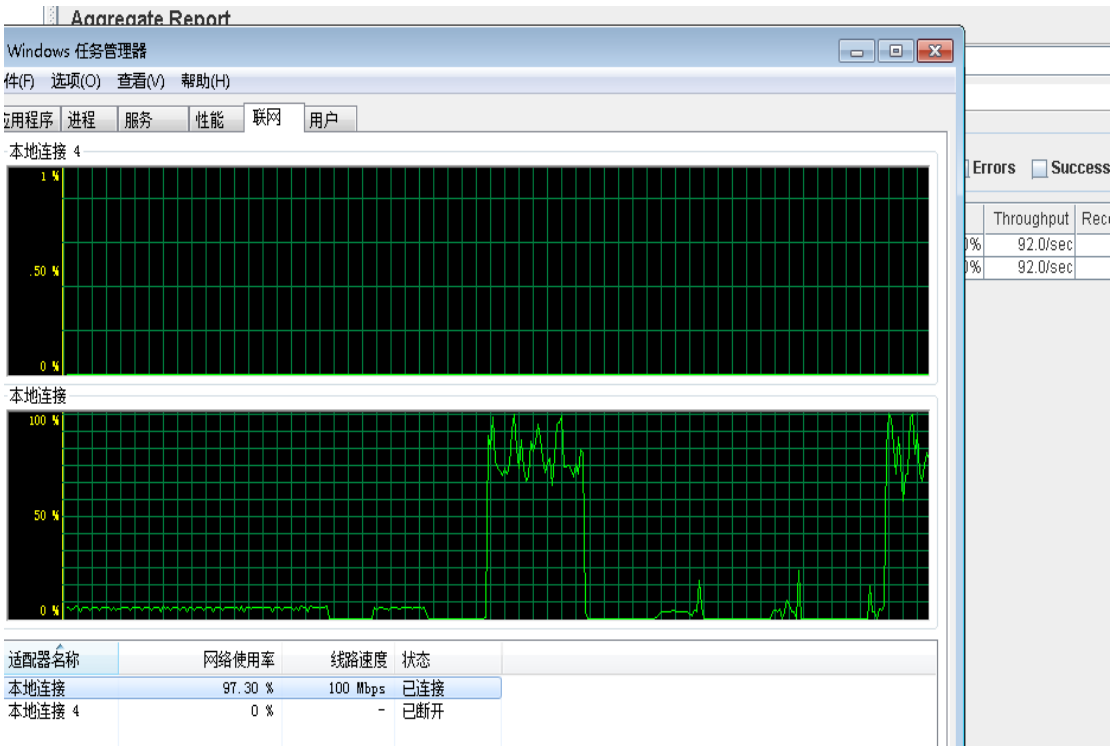
注意在协调主机和执行机上均要修改。运行测试，此时我们查看协调主机的网络使用率和测试的 TPS，可以发现对于百兆网卡使用了其中 7%的流量，对于被测系统的 TPS 为 461。



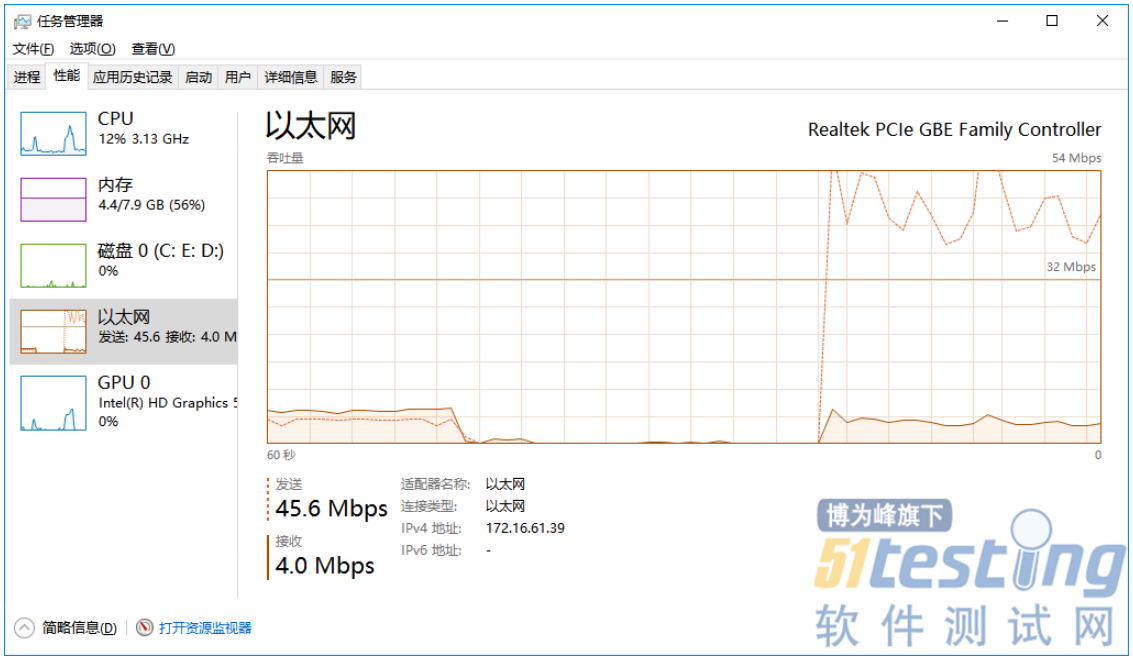
此时查看其中一个执行机的网络使用情况，也在可以接收的范围内。



接下来，修改 mode=Standard，进行同样的测试。观察协调主机的测试结果如下。



可以看到 100M 网卡已经被消耗殆尽，使用率达到 97.3%，而 TPS 也只有 92，可以想象，此时对被测系统压力是非常有限的。同样的，可以看到执行机的网络使用情况如下。



对比之前的结果，也是让人堪忧的。可以想象，以这样的测试环境对被测系统进行测试，得到的测试结果能有多大的可信度。

JMeter 官网有非常具体的描述，主要内容如下。脚本中的监听器会根据配置将测试

结果回传，默认情况下当结果产生后同步回传到协调主机。这会影响测试的最大吞吐量，JMeter 提供了配置属性对此进行干预，即 jmeter.propertie 文件中的 mode 属性。**该属性可以有如下取值:**

1）Standard: 测试中的采样信息产生的同时回传给协调主机。

2）Hold: 在测试结束前将采样信息保存在数组中。这种模式会占用执行机大量内存，不推荐使用。

3）DiskStore: 在测试结束前，将结果存储在硬盘文件（由 java.io.temp 属性指定）中。这种序列化文件会在 JVM 退出后删除。

4）StrippedDiskStore: 将成功的响应数据在采样信息中删除，之后使用 DiskStore 的方式处理。

5）Batch: 当采样信息超过指定的阈值后，同步将采样信息发送。阈值可以是采样个数阈值（num_sample_threshold）或者时间（time_threshold），这两个阈值可以在执行机的 jmeter.propertie 文件中指定。

- num_sample_threshold: 累加的采样个数，默认为 100。
- time_threshold: 时间阈值，默认为 60000ms（即 60 秒）。

后面的 Asynch 模式与该模式类似。

6）Statistical: 当采样信息超过指定阈值后，发送采样信息的摘要信息。根据线程组和采样名称进行表记。摘要包括如下字段:

- Elapsed time: 流逝时间
- Latency: 延迟
- Bytes: 字节
- Sample count: 采样个数
- Error count: 错误个数

其他字段会被丢弃。（5）中的两个属性也对该模式起作用。

7）Stripped: 将成功的采样在响应数据中删除。

8）StrippedBatch: 在响应数据中删除成功采样后，使用 Batch 模式。

9）Asynch：采样信息在本地队列中临时存储。启动一个独立的工作线程进行采样信息发送。这样能够允许测试线程得以持续执行而不用等待回送的结果返回。当然，当结果产出太快，导致临时队列被充满，也会使得测试线程阻塞，直到队列中一些数据排空。该模式可以用来削平采样信息的网络波峰。通过执行机 JMeter 的属性 asynch.batch.queue.size （默认 100）进行队列大小配置。

10）StrippedAsynch：在响应数据中删除成功采样，再使用 Asynch 模式。

11）自定义：通过配置一个特定的 Java 类，可以配置自行扩展的发送模式。扩展类必须首先接口 SampleSender，并且实现接收一个具有 RemoteSampleListener 类型变量的构造函数。

上述提到的 Stripped 类的模式，都会将响应数据中的成功信息进行删除，所以一些依赖前置响应数据的元件会出现问题。这个问题可以通过调整测试脚本得以解决。

往期作品：《JMeter RabbitMQ 采样器 AMQP 详解与实战》

词根字典用户测试

◆ 作者：狂 三

1、被测试对象

- 词根字典（http://cgdict.com）是一款结合了传统在线英文词典和英语词根词缀的综合查询系统。是一个全新的由年轻人组成的团队，创立于 2012 年 7 月，其团队致力于打造一款最实用的英语单词以及词根词缀查询系统。（词根作为英语单词的核心，不但对单词的形、义两方面起决定作用，对其发音也有重要的影响。有效掌握词根词缀，可帮助提高英语能力）
- 产品定位——为用户在学习或工作中提供单词、词根、词缀查询功能。

2、测试目标

找到用户在使用该产品时遇到的功能性问题，并提出修改意见来提高产品的可用性。

3、参与测试者信息

根据所选产品的功能特性，且项目进行时正好是四六级考试期间，所以最终选取了 8 位测试者（皆为大学在校学生），其中 4 位（测试者 1、2、3、4）在这段时间是经常使用单词查询软件或网站的，另 4 位（测试者 a、b、c、d）在这段时间不经常使用单词查询功能。分为两组来完成制定的测试任务。

4、测试任务

- 基本功能测试（二分式成功任务）:

目标 1：用户注册登录功能可用性测试

任务 1-1	前页面找到“注册”切换入口，完成注册	
任务 1-2	找到“登录“”按钮，进行登录。	
任务 1-3	页面右上角成功显示用户登录名	

目标 2：单词查询功能可用性测试

任务 2-1	在首页中输入一英语单词	
任务 2-2	快速且准确获取单词释义	

目标 3：词根词缀查询功能可用性测试

任务 3-1	于导航栏中找到“词根列表”，且点击进入	
任务 3-2	随机点击一词根或词缀	
任务 3-3	获取其选择的词根/词缀的具体用法	

- 次要功能测试（成功等级的任务）:

目标 4：生词本功能可用性测试（查询过程中积累的生词，帮助记忆）

任务 4-1	于首页搜索框中输入 counterproductive 一词	
任务 4-2	点击搜索按钮，获取 counterproductive 的释义	
任务 4-3	在释义页面找到并点击“+生词本”按钮 ,系统显示成功	

目标 5：用户信息完善功能可用性测试

任务 5-1	与首页找到“设置”入口，切至用户信息管理页面	
任务 5-2	完善个人用户基本信息后，并完成个人信息更新	
任务 5-3	切至会员头像页面，并切换和使用新头像，系统提示操作成功即可	

5、测试规则制定

（1）任务时间的计算规则

首先，每个任务给予一个“限定时间”，根据每个任务的价值与难易程度来主观评估，将限定时间一般设定为 5-10 倍熟练时间。以下表格为任务具体评判标准:

	熟练时间	限定时间	
任务 1	25s	60s	
任务 2	20s	30s	
任务 3	15s	20s	
任务 4	25s	45s	
任务 5	30s	60s	

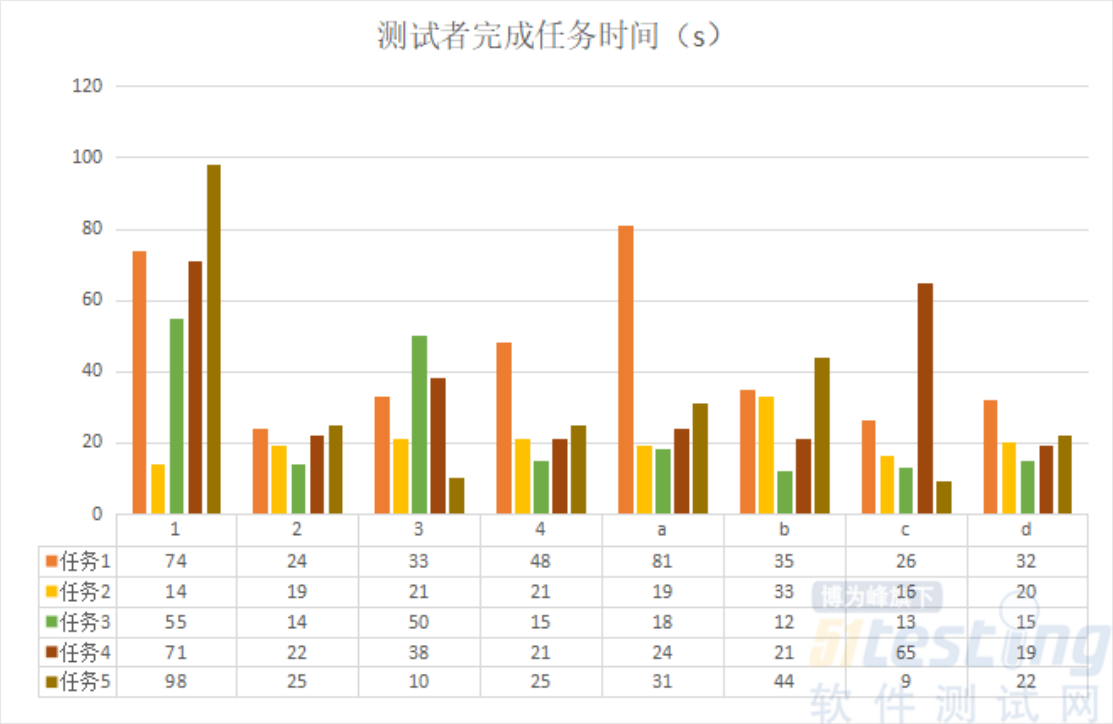
（2）任务完成度规则制定

任务的完成情况分为三种状态:

任务完成	任务部完成	任务失败	
测试者在限定时间内完成了任务要求动作	测试者在限定时间内只完成了一部分，没有完全完成任务要求动作	测试者完成不了任务自行放弃，或超过了限定时间仍没有完成任务动作	

任务完成效率是基于时间来测量，从测试者拿到任务开始计时，不等到测试者读完任务、开始操作时才计时。（可能有的人喜欢一边读一边做），在测试者宣布自己已经完成、或者限定时间到了的时候即结束计时。不是看到用户完成了就结束计时，而要测试者认为自己已经完成了，因为用户有时候会在做完操作之后去检查自己的操作是否成功了，这也应该算作任务用时的一部分。由于任务本身就存在误差，测试者在操作过程中多说了句话、或者系统响应速度慢，这些会影响任务的完成时间，所以对于记录的精确度意义并不是很大。

可用性测试_2 任务完成可视化结果及分析



	1	2	3	4	a	b	c	d
任务1	74	24	33	48	81	35	26	32
任务2	14	19	21	21	19	33	16	20
任务3	55	14	50	15	18	12	13	15
任务4	71	22	38	21	24	21	65	19
任务5	98	25	10	25	31	44	9	22

结果分析:

- 在基本功能测试任务 中，任务 1_用户注册任务中有 2 位测试者（1 和 a）完成时间超过限定时间，是为“任务部分完成”的情况，其余测试者顺利地在限定

时间内完成任务。数字组（经常使用）与字母组（不经常使用）的任务 1 测试结果无明显差异

- 任务 2_单词查询任务中，经常使用该功能的测试者都在限定时间内完成测试任务，成功率为 100%；在这段时间不经常使用该功能的测试者该任务的成功率为 87%，即有一位测试者在限定时间内完成部分任务。结果可以看出数字组测试者（经常使用）能够更好或者更熟练地使用该功能。
- 在任务 3_词根词缀查询测试任务中，数字组测试者任务完成成功率低于字母组测试者。不经常使用查询功能的测试者能够掌握词根词缀查询功能。
- 在次要功能测试任务中，任务 4_生词本添加功能测试任务成功率为 74%；任务 5_更新个人信息功能任务完成的成功率为 87%（任务成功率=(完全完成测试者数+部分完成测试者数*0.5)/测试者总数）。说明这两项任务测试中，功能 4 测试中遇到的问题比任务 5 要多。

可用性测试_3 通过综合多种因素的严重性评估进行问题总结

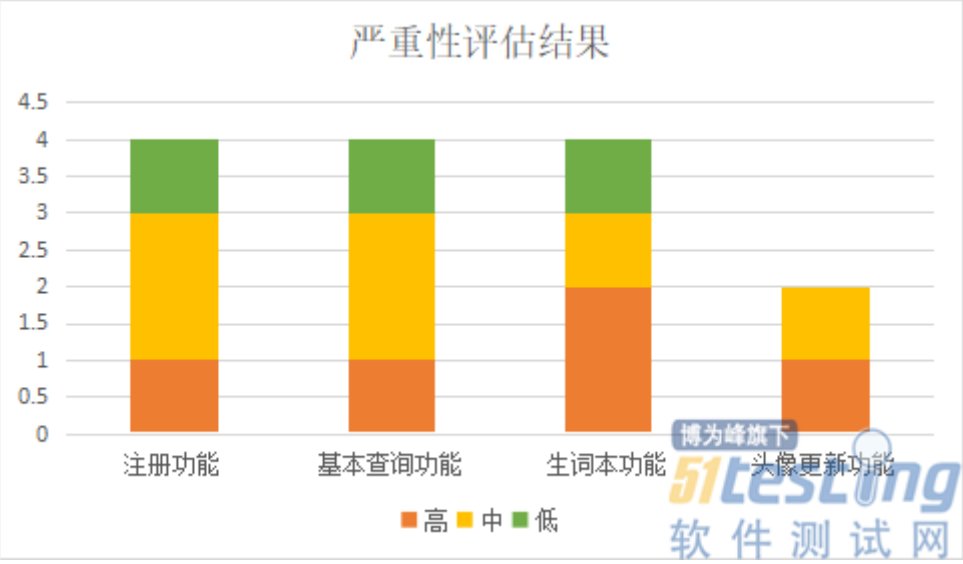
综合多种因素严重性评估定义

- 产品目标和用户需求的影响

低（0）	跟原定的产品目标没有偏离，会稍微影响用户体验时的心情，但不会影响用户需求	
中（1）	较为偏离原定的产品目标，会影响用户体验时的心情，甚至用户会因此放弃此次的使用	
高（2）	严重偏离原定的产品目标，用户完全放弃该网站的使用	

- 技术成本级别定义

	容易达成的技术修改，不需要花很多的人力物力即可完成。	
	不太容易能达成的修改和维护，需要花费一定的人力物力，以及一定的时间才能达成。	
	难以做到的技术修改，需要花费大量的成本，难以快速高效地达成，甚至有可能无法做到。	



可用性测试中发现的问题

1、在测试任务 1_用户注册功能测试中，测试者使用该功能的熟练程度一般，在限定内完成部分任务的原因有①因为注册时填写的用户名过短，经过系统提示需要进行重新注册。②填入的验证码错误，经系统提示后重新注册。

2、在测试任务 2 和 3 中，英语单词和英语词根词缀查询功能可用性良好，测试者能够较好理解其反馈的释义和掌握该功能。

3、在测试功能任务 4 中，生词本添加功能的使用操作简单且流畅，但测试中发现添加生词后，生词本中没有实时显示所添加的生词，等待系统生词反馈时间过长，所以生词本（用于记录和帮助用户记忆）功能的可用性较差。

4、用户的个人资料完善有利于网站社区用户的交流和发展，在用户更新个人资料并切换个人头像的测试任务 5 中，测试者操作熟练程度不高，且发现个人头像更换后，系统虽然有提示成功，但网页中并没有显示出更新后的图片，一直还是原始的灰色头

像，头像更换功能可用性差。

整体来说该产品各项基本功能都可以满足用户的需求，但还部分功能需要更好地完善，来提高用户体验的满意程度，同时需要突出该网站的竞争优势，加大宣传与推广，让更多人了解与使用。

可用性测试_4 词根字典客户旅程地图

阶段	单词查询	
行为	打开词根字典网站→在搜索框中输入单词 counterproductive，并点击查询按钮→获得需要查询的单词的意思和用法	
思考	想要知道 counterproductive 的意思，还有其词根的组成有哪些；既然名称为“词根字典”应该会有这方面的释义吧	
感受	（消极）一开始是对于获取新知识感到期待，但是得到反馈结果后发现并没有给出这个单词有哪些词根词缀组成，虽然知道其单词的意思和用法，但还是有点失望	
机会	希望能够完善单词词根查询功能	

阶段	进行用户注册	
行为	打开用户注册入口 → 依照要求填写个人信息和验证码 → 完成用户注册	
思考	需要我填入我个人邮箱，会发什么邮件给我吗？；这个应该不会花费很长时间吧；用户名是随便取的吗，有什么不可以取的吗？为什么一点提示都没有，等会儿有错误，岂不是又要再填一边？	
感受	不确定	
机会	填写个人资料时有相应的提示会比较好	

阶段	使用生词本功能	
行为	将刚才查得的单词加入生词本中，帮助自己记忆 → 点开生词本查看添加是否成功	
思考	一般的字典网站或者 APP 都有这个功能吧，恩没有错，这的确是必备功能啊；奇怪的是为什么生词本中没有显示刚才加的单词？网络问题还是网站本身问题？试着刷新也没有显示？！为什么退出这个页面再点进来还是没有显示？？这个添加过程需要多长时间？？我怎么知道这是我的问题还是它的问题呢？	
感受	（消极）对于一开始能够快速找到添加按钮感到开心和满足，但发现生词本功能还存在一些问题感到疑惑和失望	
机会	提高生词本单词显示加载速度，更好地完善这个必备功能	

阶段	更换头像	
行为	进入个人资料设置界面 → 选择头像设置 → 选取自己心仪的头像 → 完成头像跟新	
思考	希望能够有一个赏心悦目的头像在自己的社区中显示出来；头像图片来源有系统自供也可以自己上传图片，选择不限制	
感受	奇怪的是成功上传头像后（系统也提示成功更换了），首页中头像并没有任何变化，还是原来的灰头像，无论刷新多少次，都没有显示自己上传的头像，心情逐渐烦躁	
机会	头像没有作用，需要重点优化，也可以考虑删除该头像使用这一功能	

互联网时代测试女巫的自动化开篇

◆ 作者：测试女巫

测试女巫自动化进化论写了四期，这四期浓缩了测试女巫 5 年左右的工作进程，完事开头难，一般开了头就会一发不可收拾，其实在 C++为界面，Python 的脚本后，我们也发展了 C++为界面，C++为逻辑的另一个发展阶段；这个阶段是很多系统厂商的自动化开发 team 会用的模式，既然大家都了解的工作模式，我就不会再啰嗦；所以我萌生了总结近期，虽然只有 1-2 年的工作，但是真的是质的飞跃：突破的系统厂的限制，紧跟时代脉搏，但是不得不说这一两年真的非常的累，但是收获也非常的大；所以测试女巫将从这期开启另一个系列的文章。

如果你问，自动化已经很大程度上节省了人力，为什么还需要与互联网时代发生关联，其实回顾我们的自动化历程，本质上还是无法摆脱一个个独立工具的发展模式，对于这种模式其实很难有效率的控管，所以我们生活在互联网的时代，我们可以藉由互联网让我们的自动化脚本管理起来吗？这个想法其实是一个非常恢弘的想法，不是在做一个个工具的思维而是做一个完整的系统，可能还是会有人问：你这样做的意义是什么？举一个例子，在工业时代来临前，有一个非常著名的例子：火车与几十匹骏马比赛，拉相同重量的货物，看是火车跑得快还是骏马跑得快；工业时代发展到现在，还有人去做这个实验吗？现在的高铁，磁悬浮的速度，需要多少骏马呢？是的，人类总是在不断地追求效率，这个效率就是人类活动创造的能量要不断的大于每天消耗的能量。

因为担心与测试女巫公司存在争议，所以测试女巫会利用自己学习的知识，以及还需要为此系列学习一些新知识，搭建一个简单的新系统，为大家讲解；这个过程不会很轻松，但是女巫乐在其中^_^，准备好了吗，我们开始起飞了！

一、此系列的规划

我们的目标是建立一个可以将我们的 python 开发的脚本管理起来的系统。

开发环境的建立：python，Django，开发虚拟环境的建立

- Django 架构介绍
- Database 与 django
- Html+Bulma

二、开发环境的建立(以 Mac OS 为例进行讲解)

1、Python

2、Django

3、开发虚拟环境建立

正文:

一、Virtuallen

Virtuallen 是用来创建虚拟环境的软件工具，virtualenv 的一个最大的缺点就是，每次开启虚拟环境之前要去虚拟环境所在目录下的 bin 目录下，source 一下 activate，这就需要我们记住每个虚拟环境所在的目录。

二、为什么需要虚拟的环境

避免开发环境被作业系统等其它因素影响，开发环境需要一个纯正的不被其它因素干扰的环境，我们的发布是 docker 的沙盒的环境，开发和发布均在纯净的被隔离的环境下，后续厘清问题会比较容易。

其实可以这样理解，虚拟环境就是被隔离的环境，就是为了让 python 项目开发时不被全局变量所污染，而 Virtualenv 就是为单个项目创建独立的 python 虚拟环境。

三、virtualenvwrapper

virtualenvwrapper 是将所有的 python 项目虚拟环境环境都存放在一起，可以让我们更加方便的管理虚拟环境在使用 shell 配合小型开发工具就会非常方便。

所以 virtualenvwrapper 是一个基于 virtualenv 之上的工具，virtualenvwrapper 是管理 virtualen 创建的虚拟环境的工具。

四、安装 virtualenvwrapper 基于 MacOS

问题

因为 Mary 的 MAC，python3 安装的路径为 /library/Frameworks/Python.framework/Versions/3.6;

所以在设置 bash 文档时会出错.

解决方法

open -e .bashprofile 会打开.bashprofile 这个文档，在这个文档的最后添加以下配置参数:

```
export WORKON_HOME=HOME/.virtualenvs   #虚拟环境统一存储的路径
export PROJECT_HOME=/data/Devel #  定义创建虚拟项目的路径
export
VIRTUALENVWRAPPER_PYTHON=/library/Frameworks/Python.framework/Versions/3.6/bin/python3
  //定义 python3.exe 所在的路径
export
VIRTUALENVWRAPPER_VIRTUALENV=/library/Frameworks/Python.framework/Versions/3.6/bin/virtualenv
//定义 virtualenv 所在的路径
source /library/Frameworks/Python.framework/Versions/3.6/bin/virtualenvwrapper.sh//打开终端会自动启动这个 sh 文件
```

保存这个档案后再重新打开 terminal 就会自动运行这个.bash_profile 文档，使配置档案生效需要输入以下命令:

```
source ~/.bash_profile
```

建立方法

直接在 terminal 中输入 mkvirtualenv virtualname(user define)

查看 mac 的文件夹

可以看出会在 kr1510 文件夹下，建立一个.virtualenvs 的文件夹，其中有你刚刚建立的 myfirstvirtualenv 这个文件夹

```
kr1510deMacBook-Air:~ kr1510$ cd .virtualenvs
kr1510deMacBook-Air:.virtualenvs kr1510$ ls
get_env_details      postmkproject      premkproject
initialize           postmkvirtualenv   premkvirtualenv
myfirstvirtualenv    postrmvirtualenv   prermvirtualenv
postactivate         preactivate
```

postdeactivate predeactivate
kr1510deMacBook-Air:.virtualenvs kr1510$

build up virtual development environment to QA Cloud

使用 vs code 打开 qa cloud 的 code

此范例的前提条件是已经使用 mkvirtualen 建立了类似沙盒的隔离环境

第一步：打开 project 并查看沙盒

在 vs code 的 terminal 中输入 workon 就可以看到我们已经使用 mkvirtualen 建立的所有隔离的环境如下：

一共有两个隔离环境：testautomationframeowork and testqacloud

kr1510deMacBook-Air:qacloud kr1510$ workon
testautomationframework
testqacloud
kr1510deMacBook-Air:qacloud kr1510$

第二步：进入你想要进入的沙盒

在 project 的路径下，输入 workon testqacloud 就可以非常方便地进入这个沙盒环境

kr1510deMacBook-Air:qacloud kr1510$ workon testqacloud
(testqacloud) kr1510deMacBook-Air:qacloud kr1510$

第三步：在沙盒中 build 后端环境

pip install -r requirements.txt

注意：我们的 NJ_QAcloud 此 branch，已经将 psycopg 此包名改为 psycogp-binary，因为这个包的名称发生了变化，不更改安装会失败

已经做过实验，目前这个 NJ_QAcloud 的 branch 的 requirements.txt 是可以正常安装的。

安装完毕后直接运行：python manage.py runserver 即可以正常运行

build up virtual development environment on AutomationFramework

使用 vs code 打开 qa cloud 的 code

此范例的前提条件是已经使用 mkvirtualen 建立了类似沙盒的隔离环境

第一步到第三步均与 QA Cloud 保持一致

第四步：在沙盒中运行 **celery**

说明

new_master 这个 branch 的 requirements.txt 做了如下修改:

新增 django == 2.1.7,

Redis 指定版本:redis==3.2.0

五、总结:

对于 **virtualenvwrapper**，我们经常使用的命令是:

（1）创建沙盒: mkvirtualenv myproject

（2）查看 pc 中有多少沙盒: workon

（3）激活沙盒: workon myproject

（4）退出沙盒: deactivate

（5）删除沙盒: rmvirtualenv myproject

** 注意删除沙盒必须要退出当前的沙盒 **