谈谈我在自动化测试中遇到的坑

首先感谢您愿意付费来看我的文章。

这篇关于自动化测试的文章，可能和你看到的大多数自动化的文章有所不同。我不是一位专职的自动化测试工程师，没有开发过自动化的工具或者框架，用的自动化的工具也不多，也没有做过开发，所以我讲不出那些现在很多人很看重的“很深”的东西。我也不想去讲某个流行的自动化的工具要怎么使用什么的，我觉得这些东西并不是我的，而且也是可以很容易获取的。

那么在自动化这个很大的领域来说，我是什么呢？**我是自动化技术的使用者，要在团队中做自动化，还是脚本的编写者、管理者和运行者。**我想大多数测试朋友和我做的事情是一样的把。我想在这篇文章中，给大家分享一下我这些年实践自动化的经历，特别是那些不是那么成功的经历，希望能够给你带来一些思考和共鸣。

我的自动化历程

初次接触自动化测试

初次接触自动化测试：我发现光靠工具和热情是做不好自动化测试的。

我是自动化测试的簇拥者。记得刚做测试那会，一听到“自动化测试”这个概念，就觉得好神奇，当时就把“手头的工作都自动化了”。我能把这些内容都自动化，不是我厉害，而是新员工手里的工作不多，又很简单，而且当时公司已经研发了一些自动化平台，我的这些自动化测试的原理就是捕捉到一个windows的窗口然后往里面发送字符串，连测试结果都还不能做到自动检查，还要自己去看日志或者截屏。尽管做得非常粗糙，这也极大的鼓励了我，我每天跑着这样的脚本乐此不疲，想象着下一步这些脚本会变得很智能。

GitChat

接下来我就开始向公司的自动化测试前辈（本部门、外部门）学习，自己开始搞自动化。这时我又换了个产品，新的leader当时并不赞同做自动化（现在非常能够理解他当时为什么不赞成做自动化），我很沮丧，但我想公司已经有了现成的自动化测试的平台和工具了，我只要学会了用这个工具，自己就可以写脚本了，自动化测试不就做起来了嘛，不就是个工具吗，能有多难呢。于是我决定加班来学习脚本语言和学习使用工具。我的进展不错，但很快我就开始感到，自动化测试并不像我想象的那么美：

- 一个非常简单的功能，写好再调通，花费的时间并不少。别人5分钟就能做好的事情，我要花1个小时。
- 脚本执行时一旦发现问题，排查起来花费的时间也不少。一般来说跑出问题了，我会再反复跑几次，先确认是不是真的有问题，再加各种打印或者等待来运行脚本定位问题（别笑，当时真的是这样的）。 我还记得当时我对这个问题，是这样安慰自己的，没事，自动化的优势是体现在反复执行上的。但是很快我就发现：
- 界面、环境稍微有点变化，脚本就不能用了。这点让我感到反复执行好像也不是那么好使，有点崩溃。
- 由于我们的产品经常会定制，版本的分支也很多，我发现如何把这些脚本管理起来，便于在不同的测试场景下测试也是个问题。

这两个问题让我有些崩溃，大家都说的自动化测试反复执行是强项，为什么到我这里就不灵了呢。

我开始意识到，**自动化测试不是靠一个工具，然后靠一腔热情加个班就可以完成的事情。**除了工具，如何设计函数，如何检查脚本的运行结果，如何做版本管理等等每一件事情的工作量都不小，需要有策略有规划，一步步的来完成。当然，如果你只是想写几个脚本玩玩除外。

第二次进行自动化测试

第二次进行自动化测试：没有做好自动化的准备，盲目追求自动化率。

第一次的自动化测试就这样以失败告终了。但我也成长为了一名测试基层小leader，有了些可以“做主”的小权利。我认真总结了上次的经验，显然问题主要出在没有规划和设计自动化上，我想只要我做好了规划，加强设计，再做一次自动化一定行，于是我决定和我的小伙伴一起，再来做一次自动化。

当时我所在的公司的做事方式是做事情必须要有个目标，要写个承诺，年底还会拿这个承诺来考核你。所以我开始思考自动化测试的目标。我发现无论是公司内部还是公司外部，只要说到自动化，都是说定位于回归测试，好吧，既然大家都这样说，那一定有大家的道理，那我的自动化目标也是定位在回归测试自动化好了。另外既然是一个团队都来做自动化，肯定要从简单的地方开始入手，这样我们的自动化的目标就变成了从简单的回归测试开始自动化，完成100%的回归测试。

这个目标看起来似乎没有任何毛病，但具体执行起来的时候，在"简单的内容先自动化"的思想的指导下，我们做了很多测试边界的脚本。（什么叫测试边界值的脚本呢。比如一个接口的配置是允许输入（1，5），边界值就是0，1，5，6，边界值的脚本就是测试数据为0、1、5、6的脚本）。

由于我们的目标是要100%的回归测试，但我们当时并没有一个标准的回归测试用例集。那些简单的边界值脚本就自然而然的都成为了回归测试用例集。后来项目压力压下来，做自动化的时间变少了，为了达到自动化率的目标，我们甚至发展到把一个测试边界的脚本，拆成多个脚本（比如上面那个例子，测试数据为0、1、5、6，本来一个脚本就可以测试完，我们却偏要写4个脚本），这样，我们"很聪明"的达到了自动化测试的目标。

但这样的自动化，我们自己都不不太想去运行，因为我们自己心里清楚，这样的脚本，运行不运行又有多大的意义呢。

这次经历让我对自动化测试有了新的思考：

1. **自动化的脚本要开发哪些内容，不应该在自动化开发的时候才来决定，而应该是事先就确定好了的。**换句话说，测试用例是自动化的基础，**有明确的测试用例才能保证自动化测试的内容符合预期目标。**
2. 没有考虑项目进度会影响到自动化测试这个风险，也没有考虑自动化实现时会不会有什么问题或困难，就轻易承诺100%的自动化率，**盲目追求自动化率，使得最后大家花精力开发出来的自动化脚本没有太大的作用。**

归根到底，还是没有做好自动化的准备。

我们在做自动化测试的时候，很容易只是盯着自动化，仅从自动化这个方面去思考，把自动化当成了一种很高级的测试，去设计自动化的框架，组织等，却忽**视了自动化测试的本质——自动化测试就是一种测试执行的方式。我们在手工测试时要如何准备测试执行，自动化测试的时候也需要考虑。**

GitChat

第三次自动化测试

第三次自动化测试：自动化脚本的误判。

第二次测试就这样也算是失败了（反正脚本几乎是都废了）。有了这两次的失败经验，俗话说事不过三，所以我准备再来一场。

这时团队的测试能力已经有了长足的提升，我们已经有了一些测试用例，为了做好自动化测试，我专门组织大家把需要自动化测试的用例筛选出来了。既然目标是回归测试，用例的筛选标准也很明确，就是那些基础的，需要手工反复执行的用例。

然后我又对这些用例逐个进行了分析，把当前的自动化测试技术暂时不能支持的用例也标记出来了，告诉大家不用担心，这些用例可以下一次再执行，我们就算是要追求100%的回归测试率，也是我们真正应该执行的，并且现在可以自动化测试的那些用例。

我还记得当时我们的自动化测试平台也升了次级了，平台也更稳定了，提供的功能也更多了，大家的干劲也很足，所谓天时地利人和，我对这次的自动化测试实践充满信心。

很快，脚本被一批批的开发出来了，之前不能自动化的测试的用例也随着自动化测试技术的突破而变得可以自动化了，一切都在向着好的方向发展。但很快我们就发现新的问题出现了，自动化脚本结果出现了误判！

什么叫自动化脚本的误判呢，就是自动化脚本在自动化平台上显示的结果和真实的结果不一致。比如脚本A在自动测试报告中显示的结果为PASS，但实际的功能却有可能有问题。在自动化测试报告中显示的结果为失败，但实际可能却是受到环境的影响造成的，功能却没有问题。

换句话说，我们无法相信自动化测试的结果。

这真是要把人折磨死的节奏啊。

我们想了很多办法，比如每一轮自动化测试，同一个脚本都反复执行几次（如执行5次），然后设置一个脚本执行失败的“容错值”（比如设置容错值为2，即执行5次这个脚本，脚本失败只要不超过2次就都算通过，OMG这个想法也真是见招拆招，见洞补洞，丧心病狂啊）。想办法保存所有的测试执行记录，然后再手工再从测试记录里面去抽检一定比例的测试脚本的执行结果，看抽检的结果和脚本运行结果的差异，再以此来决定脚本出现误判的概率（OMG，我都服了我们小组的惊人的数学功底，但这真的是完全跑偏了好吗）…….

其实这些问题，**归根到底就是脚本的check部分写得有问题。**

如果我们把自动化测试比作一个机器人。让自动化测试来模拟执行某个功能并不难，这就像是机器人的手一样。**难的是机器人的“脑子”，如何让自动化脚本来聪明的确认脚本的执行结果就变得非常重要，这才是自动化测试真正的难点。**

首先我们要梳理自动化check的使用规范，根据的业务的实际情况和使用的自动化工具来确定要怎样进行check才不会遗漏，来最大程度的保证自动化测试在结果检查上的准确性。

对high level的自动化测试来说（对low level的自动化测试，如接口、单元测试来说，这个问题并不明显），**无论是UI界面的，还是CLI（命令行）的用户接口的，都隐含了一个情况，就是自动化测试只能check到预期有的东西，却不能check到预期外的东西。**

以下面这个web页面为例。假如我的自动化判定的是“秒杀”，但实际“秒杀”后面却多了一些别的东西，比如多了个“：）”。在手工测试下，我们很容易发现这个问题，但自动化测试却往往测不出这个问题。

GitChat

CLI也是同样的道理。比如我想得到的是arp的回显。如果系统在给出了arp回显的同时，又打出了“this callback has not been registed”，手工测试很容易就发现了这个问题，而自动化却往往测不出：

```
NF> show arp vrouter-id 0
this callback has not been registed.
```

第四次自动化测试

有了前三次的失败经验，第四次自动化测试顺畅多了。这时我们的自动化率也就是在10%的样子，但我们一共有接近1w个用例，所以脚本总数还行。团队里有的人觉得还是可以的，感到不管怎么样总是有了些东西，也有的小伙伴有些质疑，我自己是表面上表现得很支持，干劲很足，内心却总是觉得有些别扭。

其实对我的产品来说，基本功能部分的质量还是有保证的。在没有自动化测试的时候，即使是做回归测试，我们不会去测试那些很基础的用例，而会把几个功能组合起来做场景的回归测试，这样如果基本功能有问题，也可以第一时间发现。

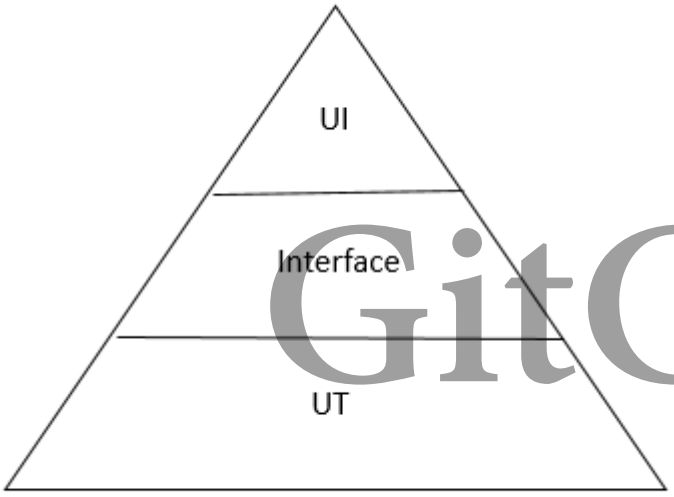
自动化后，变成了先跑这些基础的回归测试用例，通过了，再去做场景回归。无论自动化测试的结果如何，都要投入一定的人力去运行维护自动化的环境，确认结果。虽然这时我们的自动化已经稳定多了，但自动化的乌龙事件还是比较多的。

我没有感受到自动化的便捷，相反，我感觉它成为了一个负担。

那时正是自动化测试在行业中被热捧的时期，一方面是外面的专家老师对自动化测试的各种赞美之情，另一方面我却觉得我那么努力，却一直没有达到应有的预期，我感到十分迷茫。

我想过是不是回归测试的用例写得不对，我试过将手工执行的场景自动化，但是这对high level的自动化来说特别不现实。此时我的自动化测试仿佛进入了一个僵局，我不知道现在我做的事情的意义在哪里，是该停下来还是继续走，还有关键是怎么走。

这段时间看了很多自动化测试的材料，看到这个著名的金字塔：



GitChat

出自Martin Fowler的博客http://martinfowler.com/bliki/TestPyramid.html

我感到有点释然了，我觉得这真是道出了一个自动化测试的真相：

- 自动化测试也需要分层
- UI的自动化（也就是high level）的自动化本来就做不高

这个模型虽然没有解决我的问题，但让我不再纠结。我想试试把自动化往下走，做接口的自动化或者单元测试。当时我们项目的情况是，单元测试开发在做，但我侧面了解到此时开发做单元测试做得很敷衍。没有做接口测试，我再一了解，我们的设计，就没有接口一说，接口的改动也非常随意，根本就没法做。

此时我们的项目也很紧张（其实工作了这么多年，这个行业好像就没有不紧张的项目），产品开发和测试的压力都很大，都在拼命加班，很多同学都觉得现在做的所有事情都是符合公司要求的，也做顺了，结果看起来也还不错，也许之前自动化测试大家心里多多少少还是有些质疑的，所以大家都不太想再做接口的自动化或者再去改进UT的自动化，当然，也不知道这部分该怎么做。

在我感到自动测试可能就是在金子塔尖徘徊徘徊的时候，又发生了一件事情打破了我之前对自动化的认识。公司的大领导突然非常重视自动化，整体上加强了对自动化的投入，成立的专门的自动化小组。大领导直接拍了一个很高的自动化目标，自动率要达到30%，不到一个月又更新为要达到40%，然后要达到60%.....

这就意味着自动化测试不仅仅是做回归测试，还要做新功能的测试。要做新功能的测试，并且还要让脚本在新功能提交的时候就可以测试，就需要提前把脚本写好，而且界面、CLI都要尽量没有变化，当时我觉得是不可能做到的，觉得领导就是在那里拍脑袋瞎指挥。而且Martin这样的大师都说了，自动化测试要做成金字塔的样子，我们去把金字塔的塔尖做大做平，真的有意义吗？

事实时，自动化测试组的leader在老板的支持下，真的做到了。在老板的支持下，他把流程改了，他把自动化测试明确的放在了流程中进行考虑。我们的产品是有CLI和UI的。以前CLI和UI是功能设计后期才输出，现在改为了在需求确定后就要输出相关的设计。对CLI要输出确定的界面回显。而且一旦评审通过，不允许随便修改。如果要修改，必须要通知自动化团队。自动化团队在CLI接口设计完成后，就会立即封装自动化的函数（我们内部叫AW，action word），自动化团队基本能够在用例输出的时候就可以完成所有的AW封装。产品团队可以在用例设计完就投入脚本的编写。然后我们真的做到了用脚本来做新功能的接收测试，功能测试！

由于这个自动化团队是一个拉通了所有产品的资源部门，他们还尽量考虑了AW在不同产品的复用（在AW复用之前，我们的测试用例就已经复用了），后来进化为了脚本的复用。比如A产品有“扫描”这个功能并且已经做了自动化了，B产品也准备做“扫描”这个功能，B产品不仅可以直接用A产品需求相同部分的用例，还可以直接用A产品的脚本！脚本让原本分散在不同产品不同版本中的测试人员，形成了一种合力，大大提升了测试效率。我感到这时的自动化，才是真正可以替代手工执行的自动化，我第一次真真实实的感到了自动化测试的好处。

这段经历给我的触动是巨大的。

- **我感到自动化测试，不是测试单方面能够搞定的事情，需要开发的理解和配合，更需要领导的支持。**
- **自动化测试的技术不是最重要的。我从感到自动化测试是种负担，到实实在在感到自动化的作用，我们的自动化测试平台，脚本语言，脚本的编写方式都没有变化。变的只是配合方式和做这件事情的时机，也就是自动化测试的策略。**我们现在能够获得技术的途径太多了，是不是技术越牛就可以做得越成功呢？显然不是这样的。做成一件事情的关键是能够审时度势，去解决当前问题。
- **如何看待权威：Martin讲的金字塔，是Martin基于他的视野提出来的，是有上下文的，忽视上下文无异于断章取义。**
- **自动化测试除了脚本，还有很多上下游相关的工作，比如用例，比如需求。如果你的需求是乱的，用例是乱的，你觉得自动化会不乱吗？**

GitChat

至于自动化测试真的可以提高效率吗？我觉得不行。我觉得这是对自动化测试意义的最大的误解。那我们为什么又要做自动化测试？**自动化测试最大的意义在于，对测试人员的能力的固化。脚本可以代表测试人员的测试方法，通过脚本就把在原来在人身上的能力，固化为组织的资产。不同的团队及时没有懂这个功能的人，也可以通过脚本来分享这种能力，这才是自动化的意义。**

后记

故事已经讲完了，非常感谢你可以一直看到现在。

现在我换了家公司。在新公司里，我实践着我的自动化理念。我花了两年的做用例基线，用例已经整得不错了。最近又开始重头做自动化，从找工具，整合库，设计业务框架，设计关键字，设计脚本的管理规则开始。现在公司对自动化的关注还算不错，但依然没有多少资源。现在我的自动化做得很慢。是的，我真的宁愿做慢一点。

后来我在新公司里有幸作为评委，参加了一个测试优秀实践的评选会。当听到分享者讲他们加班加点做了2000个脚本，却只发现了2个问题时。我就问了个问题：“这些脚本的测试内容，没有自动化手工执行的时候，你们会做吗？”

分享者静默了几秒，说，不会。

然后我就没有再继续问了。还给了她们比较不错的分。

补充

对自动化测试，除了前面故事里的提到的一些理念，我还想我还想补充一些。

自动化的基本要求应该是什么？

我认为基本要求其实就两点：

1. 测试者能够信任自动化的结果。
2. 可连跑、可维护、可移植。

不要小看这两点，这两点也正是自动化测试的难点。要做好这两点，考虑的东西会非常多。我认为评价一个团队自动化做得好不好，看这两点就完全够了。

如何评估当前团队的自动化测试水平

想自测一下当前团队的自动化程度处于什么水平，我觉得可以参考下表的内容（摘自《H3C软件测试经验与实践.pdf》）：

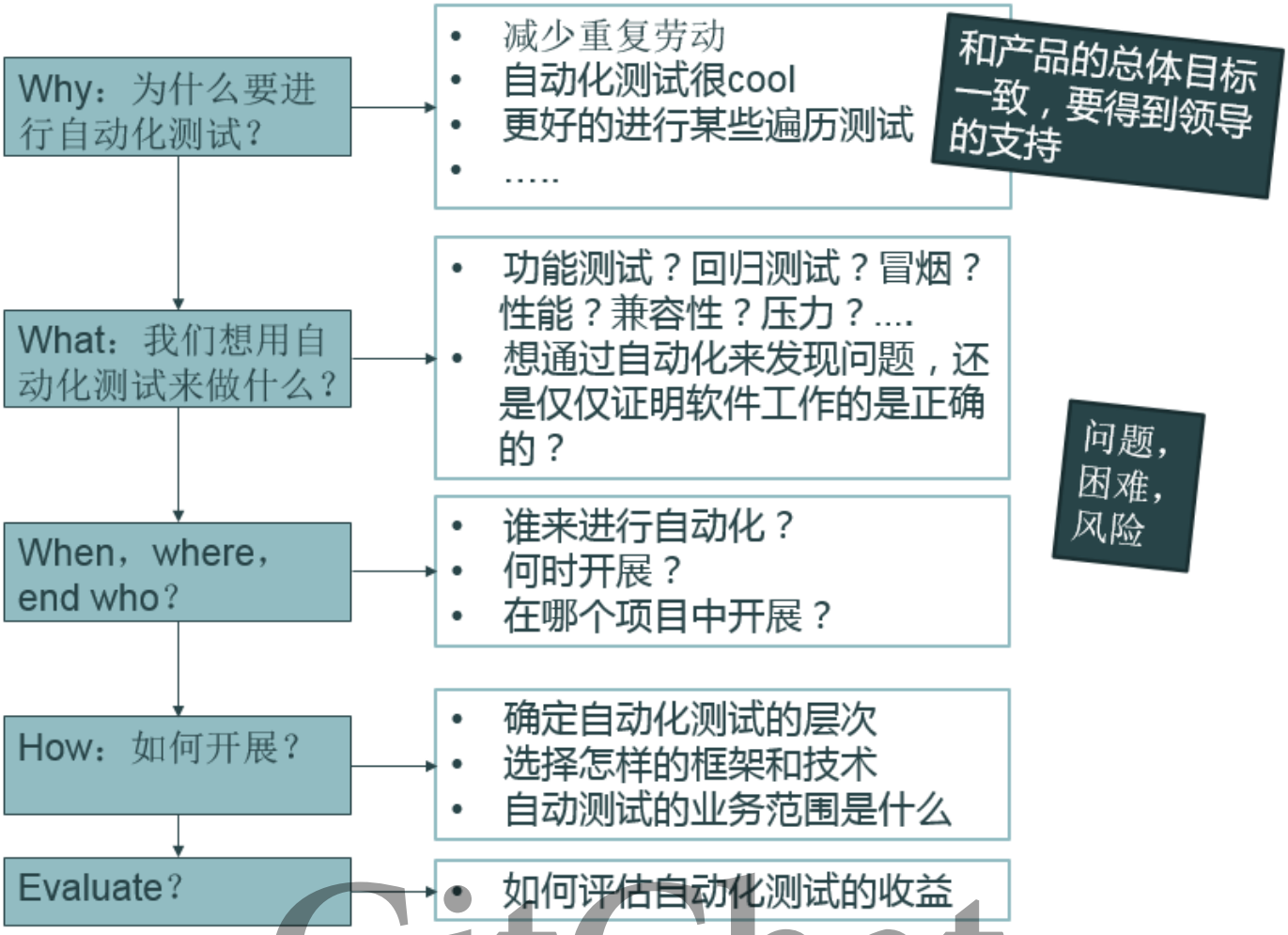
	第一代	第二代	第三代
定义	以捕捉/回（Capture/Playback）工具为中心的 自动化。	以脚本（Script）为中心的自动化。	以自动化测试平台为中心的自动化。
特点	○以捕获和回放作为主要的自动化手段，主要用于GUI系统测试； ○提供简单的脚本自动生成和开发功能；脚本语言简单，编程要素少；脚本可维护性差； ○对被测系统变更的容忍度基本为零，被测系统的细微改动都可能导致脚本无法运行。	○整个团队采用了统一的适合本技术领域的的完备的脚本语言； ○测试工程师基于自己的测试环境编写自动化脚本，可移植性差，质量也参差不齐； ○测试工程师的自动化成果工作不能形成合力，难以持续有效积累。	○统一的自动化平台框架，统一的脚本架构和风格，统一的脚本质量； ○对测试操作进行了高度的抽象概括，形成了完备的Action Word通用lib； ○脚本的重用性，可维护性，鲁棒性有了保障。
示例	QaRun测试工具；SNMP Tester工具	早期VRP测试组工程师完成的tcl测试脚本。	基于ATF自化平台开发的TestBladeV1、V2的脚本。

GitChat

文章是篇老文章。其实在这些年里，自动化测试技术已经有了长足的进步，但是自动化的水平却没有太大的提升。很多团队只是技术升级了，水平并没有升级。如何才能升级水平，我也不知道，也许大家可以在chat中讨论一番。

在团队中该如何开展自动化

我们为什么要做自动化测试，或者说做自动化测试的动机是什么？是因为手工测试没有前途了，需要转行到自动化测试吗？所以我们要决定在项目中开展自动化的测试，最好先做一个“自动化测试的可行性分析”——5W1H1E法：



GitChat

下面是一些可供大家参考的经验：

- 自动化测试并不廉价
- 自动化测试不是单靠测试就可以搞定的事情
- 对被测对象来说，不变的、需要重复执行的内容优先自动化
- 可以从“半自动化”开始
- 自动化测试的步骤：可以先模拟测试者的手，再模拟测试者的眼睛，再模拟测试者大脑
- 自动化脚本往往不那么可靠，要花最多的时间在脚本对测试者大脑的模拟上

附录

视觉测试工具：Pix-Diff，applitools，viff，phantomCSS，BackstopJS，DPXDT