网站运维之道

关于可用性

这是前一段时间投稿给《程序员》的一篇文章。标题中的"道"有些大了，您可以理解为"门道"的"道"。一家之言，妄自言道，诚可笑也。

什么是网站运维(Web operations) ？运维，绝不是某些人眼中安装系统、做几根网线那么简单？除去应用开发和业务运营之外的保障网站能运转的事儿都可能是运维工作的职责范围。运维的工作包括(但不限于) 软硬件部署、网络管理、应用程序维护、安全、容量规划、故障修复等等。

运维，有别于"运营"。在中文的语境中，运营更多和业务结合在一起的。而运维，则是偏向技术层面。

任何一个成功的站点都离不开一只优秀的运维团队，尽管他们更多时候隐身在网站背后不为人知。

网站可用性

所谓网站可用性(availability)也即网站正常运行时间的百分比，这是每个运营团队最主要的 KPI (Key Performance Indicators ，关键业绩指标)。对于 Web 站点来说，传统的那个 24x7 的说法已经不是很适用了，现在业界更倾向用 N 个 9 来量化可用性， 最常说的就是类似 "4 个 9(也就是 99.99%)" 的可用性。看一下表 1 能更为直观一些。

描述	通俗叫法	可用性级别	年度停机时间
基本可用性	2 个 9	99%	87.6 小时
较高可用性	3 个 9	99.9%	8.8 小时
具有故障自动恢复能力的可用性	4 个 9	99.99%	53 分钟
极高可用性	5 个 9	99.999%	5 分钟

根据墨菲定理的推论，世界上没有 100% 可靠的 Web 站点(除非不运行)。业界网站的可用性都是多少？引人注目的 Web 新贵 Twitter (http://twitter.com)， 2008 年前四个月的可用性只有 98.72%，有 37 小时 16 分钟不能提供服务，连 2 个 9 都达不到，甚至还没达到"基本可用"状态。电子商务巨头 eBay 2007 年的可用性是 99.94%，考虑到 eBay 站点的规模与应用的复杂程度，这是个很不错可用性指标了。Web 应用类型决定了不同的站点对可用性的依赖性是不同的。要知道 4 个 9 的可用性实际上是很难实现的目标。至于 5 个 9 的 Web 站点，一半靠内功，另一半恐怕是要靠点运气。

(图 1 维基百科网站的一台数据库服务器的可用情况报告，由 Nagios 的监控得到的)

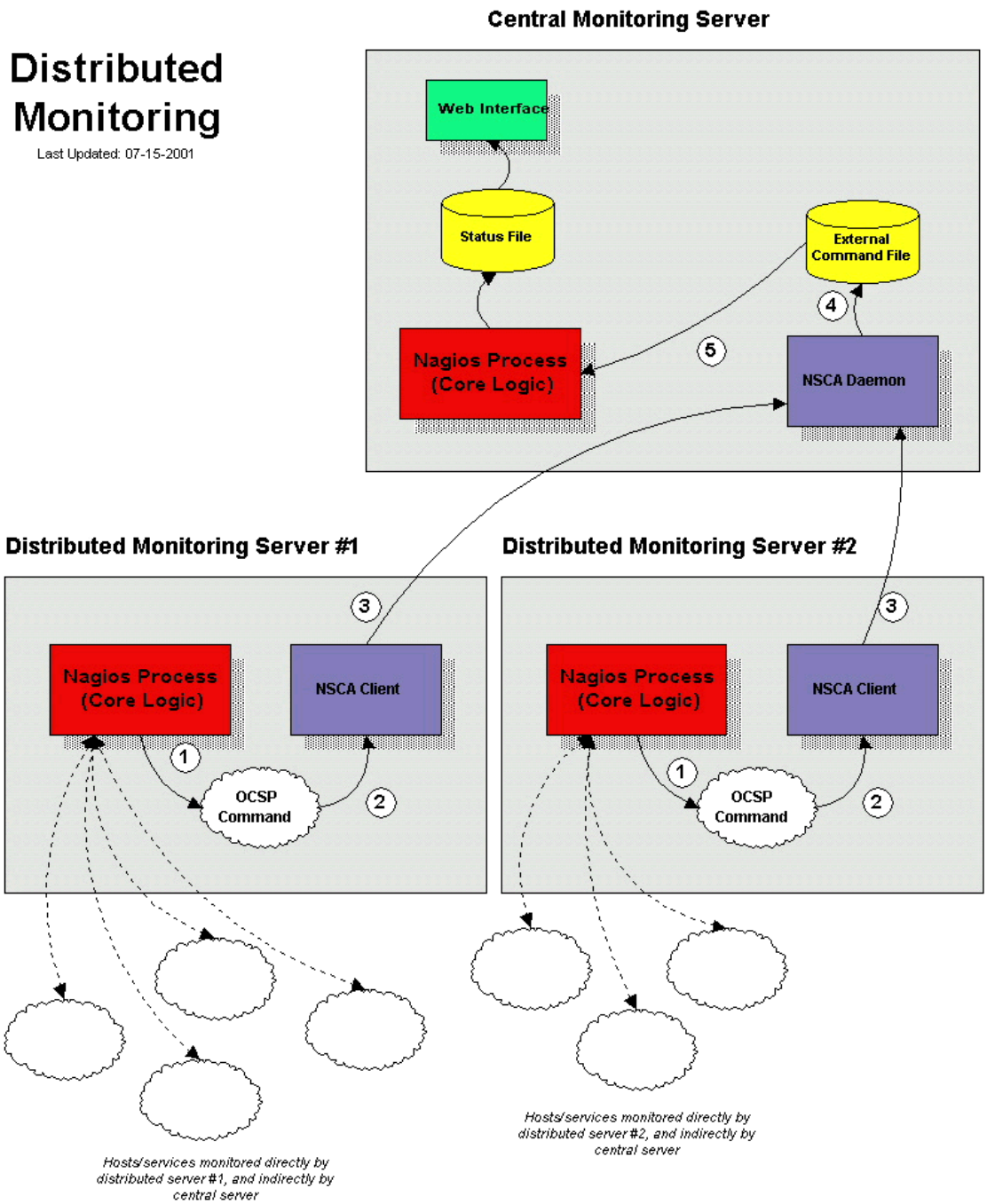
多数情况下，网站可用性会是 SLA （Service Level Agreement，服务水平协议）中的一个重要度量指标，也是运维团队向自己的客户(更多是公司老板)的正式承诺。可用性是能够持续改进的东西，KPI 制定者切不可狮子大开口，企图一步登天，拍拍脑袋提一些不太切实的指标。运维团队对可用性的承诺也不能开些空头支票，到头来两头难看。值得强调的是，如果是做第三方托管，更需要明确 SLA，明了第三方的服务能力，否则，费尽了九牛二虎之力终于保证了软硬件网络等环节都没问题了，IDC 却频繁断电或者 IDC 出口网络不可用，这也绝对做不到预期的高可用性。

提高可用性的一些常规策略有消除单点，部署冗余设备(或集群)，配置带外管理网络等，对可用性要求不高的网站这些可能足够了。如果要提供更高的可用性，比如 4 个 9 甚至 5 个 9，就不是简单靠硬件就能做到的事情，还需要建立完善的流程制度、建立变更机制、提升事故响应速度等。正所谓是"没有最高可用，只有更高可用性"。

一般来说，所有的网站运维人员都在追求网站的更高级别的高可用性，但是必须注意，这是以额外的软硬件投入、更多的人力成本为代价的。成本与可用性之间也请做到良好的平衡，盲目追求高可用性是不可取的。
(补充：Twitter 的可用性现在已经有了很大提升，但是可以看到，可用性不佳并非一个网站的杀手，只要产品对用户足够友好，足够有粘度，足够不可或缺，那么可用性并非是第一要追求的运维目标。有些运维人员被 Amazon 的某年圣诞节期间宕机所造成的影响埋下心理阴影，其实没那么可怕，如果真的觉得可怕，那么你可能被一些厂商销售人员洗脑了。)

网站运维之道 监控与报警机制

监控机制
定义了网站可用性指标，如何获取网站的可用值？监控工具该粉墨登场了。
多数网站都会倾向于利用开源软件自行搭建监控平台。笔者一向认为，即使网站有一台服务器，也应该搭建监控工具，这是保障网站能持续改进的基石。常见的开源监控工具有 Nagios（www.nagios.org）、monit（www.tildeslash.com/monit）等。Nagios 也可能是当前国内最被广泛采用的监控软件了，根据官方描述，Nagios 是开源的主机、网络、服务监控程序，从这个描述能看出，Nagios 的设计目标是很庞大的。依赖其强大的扩展性，通过分布式监控模式，管理上千台甚至更多的服务器也不在话下。而对于大型集群环境，Ganglia (http://ganglia.info/) 是个不错的选择。
另外商业化运作的比较好的开源监控工具或框架还有 Zenoss (http://www.zenoss.com/)、Zabbix (http://www.zabbix.com/) 、Hyperic (http://www.hyperic.com/) 、OpenNMS(http://opennms.org/) 等。这几个的定位都是"企业级"监控平台。当然，功能的确不比 Nagios 差，也有的弥补了 Nagios 的一些不足之处(比如 Zenoss 增强了对 Windows 服务器的监控能力)。但出于种种原因，在国内的流行程度并不广泛。



(图 2: Nagios 分布监控示意图
图片来源: http://nagios.sourceforge.net/docs/3_0/images/distributed.png)

如果要满足日趋灵活的 Web 监控需要就不得不提 Nagios 灵活的插件机制，最简单只需要几行 Shell 代码就能实现基本的插件功能。多数情况下，脚本捕获系统日志中的特定事件，通过 NSCA Client 发送给中心监控服务器即可。灵活性是衡量监控软件的一个重要标准，从这一点说，多数传统的商业网管软件怕是都不如 Nagios 这样胜任现在日趋复杂的网站环境。

提到网管监控，必然要谈到 SNMP。跨平台或者针对专有设备的监控离不开 SNMP，但有的时候 SNMP 的安全性也的确会带来严重问题。这就需要运维团队中的安全专家对监控系统机制的安全性做整体评估，或是提升运维团队的安全意识以避免在监控过程中引入更多的安全问题。

有些公司的运维团队喜欢自己写监控工具而不是利用已有的第三方开源工具。这种重复发明轮子的做法笔者认为是不可取的。这样做最明显的一个缺点是软件本身的维护成本可能会更高，而且团队人员变动的时候后续代码维护也是个潜在的问题。至于商业工具的选择，这里不作评价。

报警机制

光有监控而报警机制跟不上，不能及时把紧急情况下的信息传递给运维技术人员，那么监控形同虚设。现在报警信息发送途径主要有邮件、IM、SMS 三种(过去书籍中提到的传呼方式已是明日黄花)。

这几个途径中，邮件告警可能是最简单的，实现起来容易，一行命令即可做到，但因为邮件本身的异步属性和邮件服务器的延时问题，很难让运维人员及时得知信息。所以，如果比较严重的告警信息必须考虑其它实时性比较高的方法。至于发送到 IM，如果 IM 是支持 Jabber 的，实现起来并不难，可靠性也会有一定保障，而如果 IM 比较封闭，那么可行性就不大了，除非 IM 公司对你开放 API ，否则任何取巧的技巧来发送消息的方法其可信赖性都不强、SMS 是大家都比较倾向的一种方式，只是有很多人不知道具体如何实现，说白了也就是一层窗户纸。如果有电信服务提供商(SP)能够提供基于 Web 的调用接口给你，那么直接利用 Wget 或是 cURL 工具模拟浏览器处理表单信息即可，几行命令即可搞定。如果不具备这样的条件，不妨考虑一下短信 Modem，现在市场上这样的短信 Modem 很多，价格不贵，大多都提供二次开发的功能，简单的写点脚本即可实现目的。至于网上有人推荐的免费短信服务，因为实时性比较差，笔者是不推荐的。天下没有免费的午餐，这样的服务往往信息发送优先级很低，而且，短信到达率很难保障。

值得一提的是，报警服务器本身也需要监控的。建议定期发送测试邮件、测试短信来验证告警功能处于正常状态。尤其是在节假日来临前更要反复确保该功能是正常可用的。

网站运维之道 之容量规划

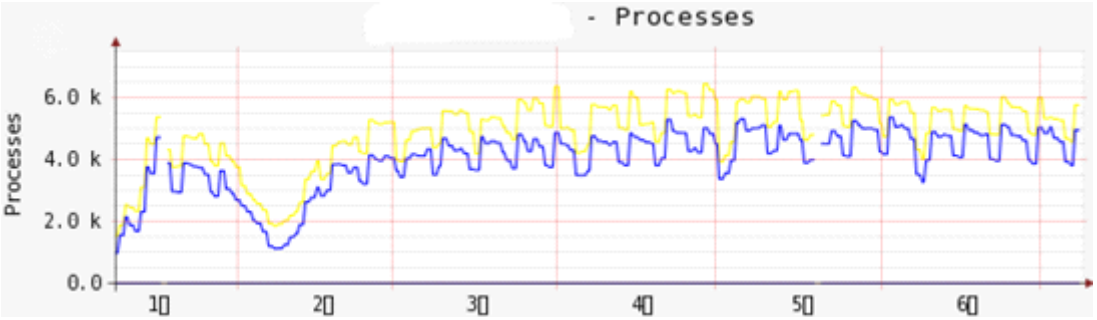
容量规划

有效的监控能够避免绝大多数问题的扩大化，但是还是做不到防患于未然。监控告警机制完善后，就需要着手考虑容量规划(Capacity Planning)的问题。

所谓的容量规划，也就是一个公司为了满足商业目标的需求而决定生产能力的过程。俗语说，"人无远虑，必有近忧"，容量规划，需要的是"远虑"。对应到运维的工作上来，一方面是**商业目标带来的容量需求**，一方面是**针对相关历史数据的分析带来的预测**。这里的历史数据，是需要运维团队采集、整理的。(从这个角度上说)，容量规划是一个长期的过程。

相关的数据保存和图表生成，基本上都会采用 RRDtool （http://oss.oetiker.ch/rrdtool/）来做。 RRDtool 也已经是业界的事实上的标准，但毕竟 RRDtool 只能算是一套引擎。而规模化的数据管理工作则需要求助其它工具，则不能不提 Cacti （http://www.cacti.net/）这是现在相当通用的做法。老牌的 MRTG 已经很少有人用了。

利用 Cacti，很容易得到一段时间内某项数据指标的变化趋势(比如网络流量的增长趋势、服务器负载的趋势等)。这是运维过程中最主要的参考数据之一，缺乏此类数据而做决策是不可想象的。



如上图，可以发现被监控的服务器上进程数量半年内的增长趋势，在 2 月份间的进程数并不高(春节期间)，随后的几个月突破 4000 个进程，对于普通的服务器来说，这是比较危险的。尽管当前系统运行可能比较平稳，但运维技术人员绝对有必要考虑中期解决方案。

容量规划中的另外一个重要参考维度是 Web 访问日志的趋势图。对于中小网站来说，Awstats 足以胜任，更大一点的规模或是对统计要求更高的站点或许只能自己写统计工具了，还没听说有什么针对大型网站而且性价比好的商业工具。这里笔者要强调一下的是，商业站点尽量不要用第三方的流量统计工具，这样很容易泄漏比较关键的商业信息。

补充后记

容量规划其实远远不止这些，比如应用服务器容量规划方面、数据库容量规划，主机容量规划、存储容量规划等等，把整个架构拆成各个组件，每个组件的容量规划都是值得大书特书的一块内容。

另外一个关键点是团队的"容量规划"，团队成长这一方面如果跟不上也很容易成为瓶颈。

未完待续，下一篇谈一下关于《流程规范》的事儿。

网站运维之道 之流程规范

流程规范

对于相对正规的网站维护工作，所有网站的所有变更必须能做到有记录，可回溯。如果是单枪匹马作战，那么要实现这个目标并不是很难，只需要把好习惯培养起来就成了，可如果要面对一个团队，那么就必须要依赖流程规范来进行约束。

所谓"流程规范"，在初期也可以拆开来对待：流程 + 规范(废话!)。

关于流程(Process)，直白的说就是"**把大象放入冰箱需要几步?**"的问题。比如上线一台服务器，那么可能要经过至少前期的选型规划、基准测试、压力测试.....等等诸多步骤。如果跳过某个环节(比如缺少基准测试)而直接上线，遇到问题的时候几乎就会因为缺乏对比数据而走弯路。

关于规范(Norm)，在运维的过程中是个范围比较大的话题，因为 Web 站点环境因为各种原因而不可复制，在另一个公司可用的规范照搬到另外一家公司未必管用。如果能够意识到并且尽早抽象出来标准化组件，并着手推进，那么规范必然会逐渐丰富起来并完善。比如

Web 服务器配置规范、Linux 主机配置规范、SAN 存储系统测试规范，都是可以尽早抽象出来并且可具体化的东西。

流程规范建立容易，但是**如何确保执行**却是一个很有挑战性的问题。从这一点来说，对于运维团队的领导的要求还是比较高的。如果要成功管理一个运维团队，起码要有足够的技术经验(当然，也容易看到外行领导内行的运维团队)，而且要有足够强的执行力。

在流程规范的建立过程中，往往容易陷入为了规范而规范的误区，或是生搬硬套 ITIL(Information Technology Infrastructure Library，"信息技术基础架构库") 那一套大而无当的东西进来(这里不是说 ITIL 不好，但最合适自己的才是最好的)，必须明确，**规范的最终目的是为了运维团队更快而不是变成束缚**，所以，千万要避免技术人员对规范的抵触。

在运维团队发展的某个阶段，推行"流程规范"所引入的 ITIL 等事物是一把双刃剑，运用得当会很好的促进团队成长，运用不好则会阻碍一部分激进成员的积极性，这一点需要注意。

补充一点，对于流程规范，不是死的东西，必须具备不断**反馈、改进、进化**的能力，运维团队也应该定期修正流程规范的有关内容。有一句耳熟能详的话是：遵守流程而不拘泥于流程，这里的"不拘泥"切不可变成钻空子的借口，要知道我们生活中很多无形成本就是钻空子引起的。

未完待续，下一部分谈一下关于《知识管理与知识积累》等方面的内容。

--EOF--

网站运维之道 之知识管理与积累

所谓"知识管理与知识积累"，其实有点绕，我们不如就说说"运维技术文档"的事儿吧，这样可能还直白一点。因为每次说起类似的话题，总有兄弟用不屑的语气说，不就是写写文档的事儿么?

运维友好的文档

不同的团队对文档要求可能都有不同的"风格"--更多的时候是运维主管要看着舒服。就运维来说，必须能够创建"运维人员友好"的文档。

一般来说，运维文档应该具备如下特点：

- **易读性** 便于阅读，便于技术人员阅读。尤其是内容不应引起歧义、转码等。
- **可搜索性** 针对具体内容便于查找，便于发现。
- **版本化控制** 这里不是普通的 V1.0，V2.0 之类的简单标识版本，而是要能够获取所有的内容改变过程，便于回溯。
- **通行格式** 能够适应不同的操作系统平台。
- **信息完备性** 具备足够丰富的交叉引用，反复保存的时候不会丢失信息等。

可能还有其他特性没在这里一一列出。有的网友看了上面的描述，这不就是 Wiki 嘛! Bingo! 基于 HTML 的 Wiki 页面，绝对是对运维友好的，尤其是网站运维团队。 我见过很多团队

用 Word 写文档，这是非常糟糕的事情。在版本化控制、可搜索性方面具备天生缺陷。或许书写运维报告用 Word 是好的选择，但是运维技术文档的积累绝对不能用 Word。

运维友好的 Wiki

你们的运维团队在用 Wiki 么?

一般来说，具备一项的语言背景可能更喜欢用该语言开发的工具(嗯，我说的是"一般")，有一定 Java 背景的程序员可能会喜欢用 Confluence 之类的 Wiki 工具。而对运维人员来说呢，什么是他们的语言背景? Shell ? No ! Perl/Python/PHP ，一般运维人员可能都熟悉三者之中的东西。

我个人多少喜欢一点 TWiki ，尽管我对 Perl 不那么熟悉。而很多中小 Web 网站，可能是 PHP 为开发语言，搂柴火打兔子，捎带脚让程序员帮着定制一些功能就成了。这是不是有点扯远了? 什么是运维友好的 Wiki 呢? 我的意见是要**能促进运维人员技能的 Wiki 软件**，比如选用了 TWiki，那么在维护的时候，Perl 背景技能就能派上用场并能进一步促进，多少有点以战养战的意味在里面。

此外，应该强制运维人员提交 Wiki 标记化的文档，而不是简单上传一些 Word 文档、PPT 甚至 HTML 附件。Wiki 编辑器里别直接粘贴从 Word 文档 Copy 来得内容。

如果团队足够大，应该有人专门定期检查文档质量，乃至对新人做一些简单的示例或者培训什么的。写一份好的文档甚至比写一大段好的代码更重要。

知识管理与积累

Wiki 上都记录什么? 最佳实践、技术心得、配置文档、软硬件信息 ... 乃至团队人员联系方式，随时记录是需要的，但**保持更新**更重要。

知识管理(KM, Knowledge Management)是干啥的? 这四个字说来话长，维基百科解释道：

```
... comprises a range of practices used in an organisation to identify, create,


 represent, distribute  and enable adoption of insights and experiences.
```

用我的土话说，要把信息沉淀下来并传递给更多的人用。一个人写的文档，团队其他的人要能看明白，要理解，要能拿着这文档做事情。没有知识管理意识的团队，成员之间的信息交流或许也有些不顺畅，可能会在人员的使用上存在很多瓶颈，遇到一点技术上的小事情，原来负责的人不在场，其他人可能搞不定，这是风险！

有些团队对待知识管理的态度上是"拿来主义"但缺乏分享精神，比如复制大量网络上的信息到内部，但是不愿意对外分享团队的心得，这样不好!

积累，意味着这是一件长期的事情。不是一窝蜂搞一下就结束不管的。一份运维文档应该贯穿网站建设的始终，逐渐丰富完善。

后记：如果还有兴趣，写写关于《自动化维护》的话题? 要不算了吧。今天这个话题写的有点匆忙，因为不是对媒体供稿，所以行文语气很散，有待于收到反馈后更新吧...

网站运维之道 之自动化管理

还是继续这个**网站运维**的话题吧。前面谈了知识管理与积累，这次谈一下运维过程中的自动化管理。

在进行这篇的扯淡之前，让我想起了《太平广记》里的一个《 板桥三娘子》的故事，姓赵的客商窥探到客栈老板娘三娘子在小箱子中取出小孩玩具大小的木头牛，木头人，喷口水，木头人、牛开始犁地耕作，撒一粒荞麦种子，木头小人种下，不一会儿，荞麦长成开花结实，木头人收割，乃至磨成面粉。然后三娘子把木头牛、人收入箱中，用得来的面粉做了数张面饼。多么好的一个自动化场景呀。

自动化的目的

自动化管理是网站规模化之后必须要面对的问题。为什么要自动化?肯定不是为了炫技，针对一个发展中的网站来说，自动化的主要目的还是为了节省维护成本，提升运维成熟度能力。另外一个经常被忽略的收益是能让运维工作更有趣味性一些，不那么无聊，不无聊的有益副作用是减少人为出错的可能。

自动化针对的范围大致可以分为安装自动化、部署自动化、软件发布自动化、升级自动化、监控自动化等几个方面。优化自动化? 别，这个稍微"高级"并且不靠谱了一点。

自动化要解决的问题是 N 次循环的过程，如果 N 不具备延续性，那么自动化未必有必要。比如某个过程可能只是短时间内需要临时进行几次，是否有必要将其自动化就有待于商榷。如果计划和开发自动化过程的成本高于非自动化成本就没必要了。

开发自动化过程

如果看过古龙的小说，他曾经描述过几个有趣的懒人，懒人造了一些木头人和机关来帮自己干一些不愿意做的事情。自动化多少就是"懒人"要做的事情，因为懒嘛，所以才会想办法节省时间和其他成本。一般来说，这个过程的开发者也是使用者，所以没必要一定要按照所谓的项目过程去走，但是开发者必须能够产出一份文档给同团队的伙伴(如果有的话)。

考虑到多数的网站运维可能都是在 Unix like 环境中的，而 Unix 的哲学思想之一就是 "Write programs that do one thing and do it well"，每个过程只做一件事情就很关键，"功能单一的自动化模块"是有必要的，把不同的模块拼装起来再去完成更复杂的需求。

Unix 相比 Windows 来说，天生具备可自动化能力。如 Shell/BASH(自动化日常操作)、CronTab(自动化任务调度) 、Expect (自动化交互场景) 、rsync(数据远程同步)等 啊都是一些需要注意的技术内容。

优化自动化过程

自动化过程一般要有个生命周期，定期升级、优化也是有必要的。面对不同的应用场景应该逐渐改进自动化的可用性。

示例：自动部署 Linux

对于批量的 Linux 安装，RedHat 提供有 Kickstart Installations 自动安装解决方案，不过该方案相对比较繁琐，前不久推出的 Cobbler 是让人眼前一亮的好工具(参见 hutuworm 的

介绍文章)。我一直怀疑 Cobbler 是中国人命名的项目，因为 PXE 发音为"pixie"(皮鞋)，而 Cobbler 的中文意思是"补鞋匠"。

OS 安装完毕之后的软件安装、更新是个麻烦事。**在一个 Linux 的环境中，SA 一定不要为软件相互依赖性浪费太多的时间**。什么 YUM、APT、YAST 啊，能用就用上。**别太迷信自己编译软件所能带来的优化收益**，实际上犯错的几率更大。达到某个规模后，本地建立、维护一个软件资料库(repositories)也是有必要的。

Linux 软件安装进化之路：

```
手工预编译-->RPM-->APT 等工具
```

已经进化到更好的阶段了，没必要还走着老路在原地折腾。

其他参考：Flickr 运维曾经采用 System Image 来自动化 Linux 相关的的运维工作。或许也可以尝试一下。

在系统配置管理(别混淆到另一个配置管理上去)方面，其实 cfengine 就挺好用的。更多类似工具参考这个比较列表。

标准化，减少后续维护成本是节省人力资源的一大法门。

自动化的一些风险

必须要承认的是，自动化有的时候是容易带来一些风险的，比如"冲掉"原有配置文件信息，不恰当的自动化脚本给系统带来额外负载等，在运维过程中需要不断总结经验。(又落入俗套)

这方面值得推荐的一本书是《UNIX 和 Linux 自动化管理》，借鉴一下其中的思路和方法。

对了，补充一下前面的《板桥三娘子》的故事发展，三娘子的面饼如果被客人吃下，则会变成驴..... 同样，自动化有的时候会把人陷进去的，运维人不要变成自动化的奴隶。

这个话题还需要继续下去么？我再想想 ...