

架构师

2月 ARCHITECT



特别专题

云计算的三个层面

探索有道云笔记工程师团队开发协作模式

基于AWS的自动化部署实践

Windows Server基础架构云参考架构

软件与敏捷

Scrum Master: 职位还是角色 ?

敏捷流畅度: 找到适合需求的敏捷方式

可视化Java垃圾回收

Backbone与Angular的比较

InfoQ
架构



每月8号出版

卷首语

所有的观察都存在偏见

有的观察都是带有偏见的

技术编辑总是会被各种新鲜概念冲击，结果就是我们对一项新技术或新产品的真
实被接受程度总会产生偏高的误判，而对老技术和老产品的被接受程度则产生偏
低的误判。

上周接到一篇投稿介绍DevOps的概念，作者Q君是一位有十多年从业经验的技
术经理，接触过Java、数据库、基础架构运维、敏捷、公共云、Hadoop等多个方面，技
术视野算是相当广阔了，但对DevOps却是刚刚接触。Q君说他们的团队在敏捷方面已经
有一些实践，刚刚发现DevOps这个概念，认为DevOps虽然仍处于初期阶段，但势头看
起来不错，会是Agile的一个很好的补充，写稿本来是为了给团队成员普及什么是DevOps，
顺手推荐了过来。

我迟疑了：在InfoQ上，介绍DevOps概念的文章在2010年、2011年左右已经发
过很多，搜索devops关键字可以搜出来一大把；到了2013年、2014年再谈
DevOps，我们基本会期待作者已经是DevOps的充分实践者，过来跟大家分享
实践经验的，这个时间段出来的介绍DevOps概念的文章，似乎多一篇不多，少
一篇不少。

但转念一想，我们的读者中还不了解DevOps概念的人绝对要比我想象中的多很
多，这方面的普及内容如果写的很好，仍然是价值很大的。一篇文章上或不上，
也许不该看它新或不新，而仅需要专注于其概念定义是否贴近正确、其实践是否
值得参考（不管是值得学习的亮点还是需要避开的坑）。

一个世界从不同的角度去看，总是不同的，所有观察都存在偏见。多试着从不同
的角度观察，并无法消灭这种偏见，但至少能够减少偏差的程度。这是我们作为
观察者需要努力的方向吧！

本期主编：杨赛



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 |

目录

人物 | People

虚拟专家座谈会：迈向云开发

观点 | View

2014年值得关注的9项技术

15个热门的编程趋势及15个逐步走向衰落的编程方向（上）

15个热门的编程趋势及15个逐步走向衰落的编程方向（下）

本期专题：云计算的三个层面 | Topic

探索有道云笔记工程师团队的开发协作模式

基于AWS的自动化部署实践

Windows Server基础架构云参考架构：硬件之上的设计

推荐文章 | Article

可视化Java垃圾回收

Backbone与Angular的比较

特别专栏 | Column

Scrum Master：职位还是角色？

敏捷流畅度：找到适合需求的敏捷方式

避开那些坑 | Void

从MVC在前端开发中的局限性谈起

ATDD实战

新品推荐 | Product

Intel 发布新版移动跨平台开发工具HTML 5 XDK

F#Tools 3.1.1增加对用于桌面和网络开发的Visual Studio 2013速成版的支持

12款免费与开源的NoSQL数据库介绍

Math.js：多用途的JavaScript数学库

Koa Web框架发布0.2.0版本

使用Cordova 3.3.0在Android或iOS上部署Chrome应用

QCon

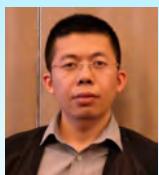
International
Software Development
Conference

全球软件开发大会

2014年4月25—27日 北京国际会议中心

北京站 2014

QCon北京2014 部分出品人团队



- 扩展性、可用性与高性能

杨卫华

新浪微博架构师



- 团队文化专题

段念

豆瓣网工程副总裁



- 尖端之上的Java

朱鸿（一粟）

阿里巴巴资深架构师



- 大数据应用与大数据处理技术

吴甘沙

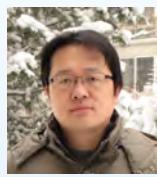
英特尔中国研究院首席工程师



- 移动应用案例分析

蒋炜航

网易技术总监



- 知名网站案例分析

池建强

瑞友科技IT应用研究院副院长



- 移动时代的前端

鄢学鹏

天猫前端团队负责人

大会门票持续热卖
2月19日前报名，享**8折**优惠

会务咨询 : qcon@cn.infoq.com

咨询热线 : 010-64738142 89880682

精彩内容策划中，讲师自荐/推荐，线索提供 : speakers@cn.infoq.com
更多经典专题 精彩内容 敬请登录 : www.qconbeijing.com

人物 | People

虚拟专家座谈会：迈向云开发

作者 [Richard Seroter](#)，译者 [王灵军](#)

开发者正在不断体验多种不同的云环境。当在云中工作时，开发者应如何改变他们的思考方式？是否有某些云环境更适合于刚准备入门的开发者？而那些目前尚未涉及云开发的开发者们又如何在此领域获得相应技能呢？

为了回答这些问题，InfoQ就云开发的现状、推荐工具和反面模式与三位意见领袖进行了交流。我们的专家组成员是：

- [Adron Hall](#), 通晓多种语言的码农和开发人员布道师。
- [Magnus Mårtensson](#), 微软MVP, 担任瑞典Active Solution顾问公司的云架构师。
- [Andy Piper](#), 热衷于推动Cloud Foundry的开发人员。

InfoQ: 是不是一位云开发人员的工具箱相比于普通开发者会有很大不同？如果是这样的话，那么在你们看来，云开发者更依赖于哪些传统的web开发者不会使用的工具呢？

Hall: 首先我会定义当听到“云开发人员”这个词时会想到什么。一名云开发人员就是负责这样的代码解决方案的人，解决方案是基于水平扩展的、分布式的、幂等的和异步处理，同时具有可伸缩、高度可用和弹性存储的特点。

我在回答这个问题时当然应完全根据这个定义。一名普通的开发人员经常是在某个传统的RDBMS数据库的基础之上构建应用，在此过程中他会使用某个框架或是其它基于此框架之上的工具，并受到垂直扩展的限制。这并不是不好的开发方式，但是对于云或其它任何可水平扩展的环境来说，以这种方式来构建应用或服务效率会非常低。一旦达到最大物理扩展极限，开发者就完全无能为力了，因为他再也没有办法使用任何合理的方式来提升性能。

一位云开发人员会横跨广阔的资源范围来构建应用，他经常将某个应用的功能拆分成更为具体的服务或模块。云开发人员也需要涉足于某些含有更多语言的工具包，这些语言包括从JavaScript到C#、Ruby或其它语言等。这样做的原因固然经常是出于必要，但在很大程度上也是为了在每个特定工作最适合使用的工具之间提供匹配。

所以，简而言之，云开发人员的工具绝对是不同的。

Mårtensson: 云开发人员需要用很多与非云开发环境不同的思考方法来武装自己。较之于其他寄宿 (host) 选项而言，当处于云部署平台时，你可以很容易地获取到很多东西。良好的云平台会提供一个工具集，这个工具集与你可能使用过的相比显得非常不同。它可拥有“无限的”存储空间，该空间同时具有自动备份、内建的缓存功能、强有力的服务总线和其他特性。

公平来说，你完全没有必要仅为了获得此工具箱而去100%买入云计算。如果需要的话，你完全可以只是使用来自云平台的如服务总线这样的某一项服务。与任何非云环境的寄宿类型不同的是，工具箱中拥有更多“工具”，比如快速弹性、内建容错、故障转移，以及你可以在任何时候按需优化费用的完全可度量的服务等。这些云特点常被引为[NIST云计算定义](#)。我认为，云开发人员有非常多的工具待去了解和运用。合适地运用这些工具可以助你构建那些以前看起来近乎不可能的、或者即使可能但也是代价昂贵的解决方案。如果这些特性被滥用的话，那么云开发人员则会冒着失去获得强大力量的希望的风险，甚至于冒着构建了更为昂贵的解决方案的风险。因为能力越强大，责任也越多。

最后，如果我们再多谈论点传统开发人员工具，其实对于云开发人员和其他类型的开发人员来说，这些工具都是极其一致的。按照我的经验，虽然在Windows Azure平台上使用PowerShell可以在云环境中获得很多收益，并且也自动化了构建和部署。但对于大多数其他寄宿情形来说，这种做法也可以获得同样的效果。我只是觉得在一个像云这样的内在的分布式环境中这样做是很自然的。对于任何一个想使用云来让自己能力更强大的开发人员，我的建议是去学习和了解云计算的真正力量。它们是你的新工具，将会助力你实现从一名传统开发者成为云开发人员的目标，你曾经为此有过一段痛苦的时光甚至于以前这只能是一种奢望！

Piper: 好问题，这也是我一直仔细考虑的问题之一。Adron在一件事上绝对正确，就是云带来的是规模——无论是就数据本身而言（同时到达的数据量，或存储的数据量）还是就你如何在云环境中为了获得可用性而扩展你的应用至跨地理位置的多个实例而言。也就是说，虽然很多专为这个新时代而创建的工具和技术（如Riak和其他的分布式数据库）纷纷出现，并且与以前比工具箱中出现了很多不同的工具，我总是倾向于认为，开发者使用新的云平台的最重要的一件事，就是忘掉他们过去的假设——换种方式来思考如何部署应用。

给开发人员提供一致的体验是构建能支持云应用的操作系统的目标之一。我们宁愿如此简单：用Ruby或node.js或随便某个工具来编写一个简单web应用，然后本地运行，或者运行在自己的单一服务器上，亦或运行在一个可缩放的弹

性的云环境中。所以我们构建一个抽象层来隐藏不同的云基础设施（AWS、OpenStack或无论什么）之间的差异，让开发人员能够很轻松地部署应用。这确实意味着他们需要意识到不能依赖于自己应用的位置、硬编码的IP地址或其他配置，这些都是不好的做法，而应是在编码时应尽可能将代码与数据库（它们有可能会改变）之间解耦。所以思考方式和架构都是不同的，不过最起码很多工具和代码都可以保持不变。

InfoQ：你们认为哪些IDE最适合于云开发？开发者应为些IDE添加哪些东西来增强其云开发的能力？你们对基于云的IDE有兴趣吗？

Piper: 很个人的说我是有潜在偏见的（作为一个Eclipse提交者），我很喜欢Eclipse，也是IntelliJ和RubyMine的粉丝。现在大部分成熟的IDE都拥有非常好的插件与云服务进行交互，比如有低级别的AWS Explorer，也有提供了平滑的构建/测试/部署体验Eclipse的Cloud Foundry集成插件。

我曾使用过像Eclipse Orion和Cloud9这样的基于云的IDE和编辑器，它们都非常方便。我很高兴地看到它们演进得也很快，从而可以利用最新的web特性。

当然很可能具有讽刺意味的是，我自己的开发者工具集根本不是以IDE为中心的。我将一天的大部分时间都花在Sublime Text以及命令行上，并且我也是Ubuntu的包管理和OS X的自制软件开源包管理器的大粉丝:-)。

Mårtensson: 听到Andy这么说很有趣，因为我有类似的背景，不过刚好相反。我是一个Microsoft/C#/Windows Azure的开发人员，这使得我在面对IDE环境和开发体验上极具偏见。最近Microsoft在Visual Studio中实现无缝的云开发体验上投入的努力是无与伦比的，而这是我在Visual Studio身上前所未见。它们确实做出了巨大的努力来支持快速和强有力的云开发。Visual Studio中的Server Explorer能够涉足任何云账户并和运行其上的任何服务进行交互。你可以管理它们，监控它们，并可以在云上运行类似于IntelliTrace和性能监视器这样的工具。然后你可以下载结果并在Visual Studio中进行分析。很自然地，你能在本地构建和测试任何云应用，包括最近加入的非超级管理员模式的计算模拟器（Express版本）。随后你可以将应用作为一个单元推送到位于Windows Azure上的自己的网站和云服务中。我在Visual Studio IDE所见到的用于云开发的（读作：Windows Azure开发）部分在Visual Studio历史上是无与伦比的。干了这杯Kool-Aid？是啊！大口地享受它吧！

Hall: 很明显有至上之选：Cloud9IDE。

虽然如此，在现在很多情况下IDE看起来有所妨碍。当某个人需要在具有很多种语言和工具的不同环境之间切换时，最容易的就是抓到Sublime或类似的某个东西并运行。

在重量级有像Visual Studio、Eclipse、IntelliJ、WebStorm和其他应用工具等这样的IDE。这些都很好，并且为一种或某几种语言提供了钩子以方便日常的运维编码工作。但是有可能当某一天快要结束时，某个不在那个IDE范围之内的语言突然冒出来，那就毁了你这一天。

另外一方面，如果一个团队能够专注于某个特定IDE，使用它来加载任何开发所需要的一切，并且IDE能够和首选的云环境协同工作，那么Visual Studio和其他几个IDE就会脱颖而出。一个很好的例子就是用于Visual Studio的AWS的.NET插件。这个特殊的插件是我所见过的仅有的没有将开发者绑定于实际云的工具集。它只是极大地简化了部署和查看云服务，这些都不用离开Visual Studio。对于.NET开发者来说，这是极大的好处，因为他们总是被鼓励并习惯于在云开发过程中一直呆在一种IDE中。

然而，在大多数web开发和云环境中，你会推送一些东西并使用像SSH这样的工具。在这种情形中Visual Studio和Windows通常都不是首选者（non-starter：比赛中不是首发的队员）。让Windows和Visual Studio与SSH和其他Linux或相关环境一起工作所花费的时间会非常让人沮丧。此时，非常戏剧性的是，像下面这样做反而容易得多，抓到一个终端，学会如何摆弄它，然后SSH连接到在这些终端，实际上就可以工作了。即使在使用Windows Azure，如果你并不打算在Windows上寄宿或也不想使用Visual Studio的至Azure的私有钩子，那么完全可以甩开基于Windows的开发工具转而使用运行于Linux或OS-X之上的来自于JetBrains的工具。从这些工具上获取好处，你的开发团队最终会感谢你。云毕竟是源自于Unix并在多年来已成为Unix技术世界一部分的很多理念的基础上继续演化着。

InfoQ：当开发者在构建云规模的应用时，应避免哪些反面模式呢？云提供商又如何避免你们犯这些错误，或反过来说，让你们做些错误的事情？

Hall: 在开发云分布式应用时，开发者会掉入很多巨大的陷阱之中。

我曾一次又一次看到的最大问题是，他们更愿意搭建单一的数据中心（比如AWS，一个区域等）。使用局限于某个数据中心的框架栈和故障转移所构建的某个应用仍然趋向于引起很多停机情况，比如“East 1 AWS Failure”，绰号为复活节故障。

另外一个很大的问题就是很多提供商——好吧，也许是所有的提供商——仍然还有很多SPOFs（单点故障）。Windows Azure在几个月前由于其证书问题而打破了大数据中心的故障记录。AWS也出现过由于数据中心的某个网络设备而导致的故障。Rackspace同样也有类似的问题，也鼓励了人们使用机架设备，这又在已有的其他故障点之上生成了更多的SPOFs。整个想法是想让云架构具有弹性，而这些情况适得其反。

从纯粹的开发者角度来看，最大的错误在于很容易在云的计算和存储环境中只是简单地构建一个传统的垂直堆叠在一起的应用。在云环境中使用按照传统架构信条所构建的应用会付出高昂的代价昂贵，并且效率很低。然而一次又一次，我看到人们只是重新实现某个垂直设计的应用；Sharepoint，WebSphere和其他所能想到的。他们通常只有单一的RDBMS，在此之上有个数据层，以及一个或者可能是几个应用节点，而这几个应用节点却位于某个有着SPOF的缓存层之上。

总的来看，云开发中有很多反面模式，而运维和开发总是很容易实现它们。

Mårtensson: 一个真正而又常见的反面模式就是将老的思考方式应用到新的范式上。比如认证；“我们想让顾客能够使用他们自己已有的Google，Yahoo！，Facebook和Microsoft帐号来认证我们的服务。当然我们也需要标准的用户名/密码登录方式。”你来找出其中的缺陷！基本上就是他们说想要设法变得很时髦，然后又想通过将自己的祖母带到聚会上来回归保守。如果这确实是你的开发和维护工作所要关注的，当然你可以运行并处理自己的用户名/密码认证服务。但如果你正在采用所有的方式来外部化自己的应用的认证，那么就需要通过将自己寄宿的认证服务（术语也称为安全令牌服务 STS）从你的应用中分离出来，然后才能真正实现这些方式。简而言之，你的应用不应关心任何认证。然而它应拥有一个可信任的、为你处理这个单独的关注点的标识符提供者。如果你不保存密码，则你根本不用保留此职责。最开初的表述是害怕由于不提供的标准的用户名/密码认证选项从而失去顾客。但这只会增加你的工作负荷，并且你将会失去使用云服务进行外部化认证的好处。如果你不能公正地对待这种新的模式，除了在开始就以完全错误的方式来使用它这种坏处之外，你最终还将会损害自己的业务。

Piper: 好吧，说到这里我已经很难在Adron和Magnus所阐述的常见反面模式之外再添加其他内容了——这些我都见到过。有个很明确的倾向就是以传统的方式思考并构建垂直而不是水平扩展的应用。在这样的情况下，当一个开发者“发现”像RabbitMQ这样的异步消息传递和像Redis这样的内存中的数据结

构，然后想到，哦，这是一种“全新”的做事方法时，我总是很吃惊。不，完全不是，这些概念已经出现了很长一段时间，而云平台比曾经任何时候都提供了一个幂等的、最终一致的服务模式。

在云提供商如何帮助你避免失误方面，就我自己来说，Cloud Foundry尽了其最大努力尽可能地来提供作为其环境一部分的有用配置信息，所以你可以编码来从配置信息中去查找值而不用硬编码端口、数据库设置等。这并不妨碍你硬编码，但是这样做确实不是一件好想法。

InfoQ：Redmonk的Stephen O'Grady曾经说过“云计算的最重要特征：进入的低门槛。”你们认为哪些云对于开发人员来说学习会更容易一些？不仅只是让开发者注册，而且还向开发者展示了该如何开始部署应用这些方面，谁工作做得好？对于开发人员仍然还存在哪些进入门槛？

Mårtensson：在瑞典我们说“*tala i egen sak*”，意思就是说你是偏颇地来表达自己关于事情的看法。再次说明，我是一名虔诚的.NET/Visual Studio/C#开发人员。当然就会偏向Windows Azure云。仍然……哦！不用暴露我的年纪，我在这个持续的消防比赛中已经有很好的数十年的经历，而且我还从来没有在Microsoft看见过像这样的事情。也许巧合的是，微软的云的气息到来适逢其时？也许只是因为这就是应当做的？但我还从来没有在Microsoft身上见过如此致力于推动技术变迁的情况。并不仅是VB和C#，还有很多。Microsoft为现在的Windows Azure的多种平台和IDE构建和维护了工具集和SDK。PHP、Java、Node.js、Ruby、Python & Visual Studio、Eclipse和开发一次并可给很多不同类型设备发送消息的能力；这些设备有iOS、Android、Window Phone和Windows 8。挑选你的毒药吧！今天谁是对Linux最大的开源贡献者？是的，这就对了，但是谁又曾知道这些呢？对于那些泡在这个空间的人以及那些知道Microsoft的人来说，这是一笔非常大交易！这是一个拥抱多元化的全新的Microsoft，并且它意识到许多公司为了日常运营将会使用很多服务和平台来构建自己的现实世界。这就是我们现在所处的情形，而Microsoft就在这儿。有哪个其他的栈/平台能匹配这个呢？

Hall：对于任何PaaS（平台即服务），进入门槛都尽可能如你所能达到的一样低。当我们深入探讨这些时，这会得到一些奇怪的比较结果。Window Azure据称首先是PaaS，然后才是IaaS（基础设施即服务），它以这种方式开始，然后努力推广它。AWS甚至都没有提到过PaaS这个词，即使他们拥有很多提供了PaaS风格的单一命令行（某些时候要点击）部署方式的服务和特征。而对于其他那些没有一个明确的PaaS说法的环境，门槛过多，而且很笨重，

并且我觉得不太值得提及。所以对于那些没有一个合理的PaaS说法的我将会在其他时候再讨论。

这带来了另外一个关键之处，在AWS上实际运行的所有PaaS服务都是怎么样的。Heroku、EngineYard、AppHarbor、AppFpg和几乎每个Cloud Foundry或OpenShift PaaS服务都安装在AWS上。所以很明显，如果某个应用包含了AWS的服务和AWS提供了PaaS服务的客户，那么它在可用性方面就大幅领先于其他人。即使我们回溯并只是看看首批AWS客户的安装和使用，这些客户在使用Beanstalk、EC2或S3，其安装和使用都极其简单。签约、检查认证机制或类似于通过邮件发送的编码令牌，然后安装好上面所提到的项目之一并启动，你就已经在运行一个应用了。

可以说最严重的障碍都在基于Heroku、EngineYard、Cloud Foundry和OpenShift的PaaS中被移除了。在上面这样PaaS环境中可以很容易地安装、使用和部署应用，这些恰好都是位于AWS中。

但是对于Windows Azure，Windows Azure团队和努力将这些云选项从一个“绝对不”移动到“嘿，这是非常容易的且功能丰富的”。Windows Azure，我不会再为回答这个问题而谈论其过去，它已经戏剧性地重新定义了如何更贴近部署，并积极地比几乎其他每一个PaaS或IaaS服务提供商提供了更多的部署选项和部署能力。另外其已转变为多语言的，在这场由AWS扮演兔子的龟兔赛跑中获胜。比如AWS节点Beanstalk功能。在Windows Azure能够极为容易地、优雅地和极好地在AWS上部署基于Node.js的应用差不多一年之后，AWS才实现。加上它可以围绕Node.js来定价，对于小型的共享的云寄宿模型来说，.NET、PHP和其他的应用已经为零。这对于开发者来说当然非常棒，他们可以在运营之前测试、调试大多数的应用。

总的来说，上面几乎涵盖了我个人所使用和一直关注的主要提供商的基本内容。Windows Azure、AWS、Cloud Foundry、OpenShlft、Heroku、EngineYard都是现在值得关注的主要公司，它们正在这个空间内做着繁重的创新工作，并正在逐渐地前进以移除更多的进入门槛。

Piper: 我喜欢Redmonk的这些家伙！他们都是非常聪明的人，并且我推荐那些任何不熟悉他们意见的人去寻找他们。开发者都是新的拥护者。

所以，是的，我完全同意进入的低门槛对于开发人员快速采用我们所讨论的云技术来说非常重要。实际上，这就是某个“啊哈时刻”，这可以让我从自己以前在IBM的角色转换到VMware的Cloud Foundry上来——本地编码应用的能力，无需任何改变地将其推送至云上，然后在几秒钟内将其扩展。鉴于我的角色，不用想就我可以说，很明显，Cloud Foundry进入门槛很低……但就如Adr

on所说，我想很多相似的PaaS提供商都是这样，如Heroku。我同样对为所见过的在Visual Studio和Azure环境中开发者所展示的工具集成所惊叹，在这个环境中开发者会觉得走向云的旅程十分愉快。我应该指出的是，很多云提供商，特别是PaaS提供商也拥有工作得很好的IDE插件。然而对于真正地低进入门槛，我仍然喜欢源自于Github命令行方式的克隆，本地构建和测试，然后从命令行推送至云的体验，Cloud Foundry和几个其他云提供商都提供了这种——这比需要一个IDE更轻量级。

InfoQ：对于开发者来说哪些事情在过去曾经是很难的，而现在由于云变得简单多了？而且，有没有某些事情在过去是简单地忽略掉或者根本不做，而现在变得简单明了的？

Mårtensson：这是一个大局观的问题。当我们开发时什么是重要的？我们会在某一天回忆起云之前的“黑暗时代”并问自己在那个时代我们是曾经如何让一切运转起来的吗？构建一个全球可伸缩的供几十万甚至上亿用户使用的解决方案确实不是我们大部分人都曾经做过的。但是我们确实可以做到这一点。如果我们业务比像Facebook那样构建自己的数据中心要小得多的话，那么能有机会看到过自己能做到哪一步吗？如果有一个了解了云平台的力量的像样的架构师的话，我就敢说再开发这样的解决方案甚至都不再是什么难事了。现在没有新成立的公司会说“好吧，我们现在获得一笔风险投资，让我们出去采购一些服务器回来。”如果我们展望未来并设想我们开发将会所使用的环境，我确信CPU的能力、内存的大小、存储容量、甚至互联网的速度对于我们构建应用的方式的重要性会越来越小。相反我们会开始依赖于总会有足够的容量给我们正在做的无论什么东西，以及能满足我们的服务所要求的无论什么样的使用模式的需要。当然事情在未来仍然会出现中断，并且某些时候服务会出现故障。但不应会出现这样的情形，即我们不得不恐慌地冲到市中心去买一个全新的硬盘驱动器。我们的服务将会是自愈的。在这个大局观中我们会使用新的模式来开发，这些模式从开始就会把所有这些问题都考虑进来，而且我们从来也不会再关心是否拥有足够的计算能力。

Hall：我将建议的是排在前三名的事物：

1. 已经影响到开发人员能如何来开发的最大的影响是能力，即仅需在这儿或那儿点击一些选项或某个脚本就可以让整个运行开发环境跑起来。服务器、测试服务器、UAT服务器等。在以前，即使在简单的虚拟化下这些在很多方式上都会受到限制。但是现在，在拥有AWS、Azure以及其他类似云环境所提供的云计算能力的情形下前面那些都完全不再是问题。以前需要耗时几小时

或几天甚至几个月的事情现在几分钟之内就可以完成，并且以一种能向前移动并保留全部努力的方式工作，而这种方式在6-7年前完全无法想象可以这样做的。

2. 跨地理边界的分发系统的能力，这在6-7年前，即便不用花费数百万美元资本投资，也会需要数十万。现在每个月仅只需要几百或几千美元，一个庞大的、跨越广为分散的多个国家的分布式系统在几分钟之内就可以安装并运转起来，并准备好投入使用。
3. 与垂直叠高的方法相比，分布式系统正在成为普遍的做法。随着这种改变，隐藏在幂等、弹性、自愈、异步、可伸缩、高度可用性系统背后的思想在大大小小公司的绝大多数程序员中出现。随着越来越多的开发者转向水平扩展的做法，垂直扩展背后的极端受限的设计逐渐地被扔到路旁。随后一系列很多用来增强利用这些功能的能力的语言和框架应运而生。这种观念模式和方法的变化一直在持续进行着，并日益增强着云计算的能力。

.....这就是在我脑袋中立刻能想到的排在前三的最重要的事物。现在已经发生那么多的变化，获得了很多的进展，以至于关于这个话题有人能写一本书了。

Piper: 对于开发人员来说，云让哪些变得容易呢？

1. 按需的、潜在的可自由支配的环境。实际上，像Vagrant这样的本地工具还一如既往是开发者的朋友，但是能快速提供和千篇一律的克隆环境以及能以通常很小的代价运行这些环境的能力也极具价值。这对于开发和运维活动一直是巨大的推动力——开发者不用再面对运营维护者的突发奇想，开发者曾经得去为他们提供新的环境。这并不是为了敲打运营团队——新的云工具和环境也为他们提供更多的敏捷性。“作为服务”是*aas缩写的关键部分。
2. 可伸缩性、可用性、弹性。我想Magnus和Adron对这部分已经说得很好。在这里除了将它们归结为这三原则之一之外，我无法再补充什么了。
3. 我认为有两件事——“大数据”和“物联网”——因为云的可用性在很大程度上变得更有能力。垂直扩展的大数据库这许多年来已经成为可能，但是具有海量存储容量、复制、MapReduce等特性的分布式数据库更倾向于与云联系在一起。传感器的连通性、数据的采集和对数据的响应一直以来都是只以单点为基础，但是现在的开发者已经可以构建复杂的能实现自己想法的系统，使用云结构的灵活性构建的系统只有零或很少几个故障点。
4. 上面只是一个快速总结。这是一个技术演变具有深远影响的领域。

InfoQ: 虽然很多开发人员都开始花费大量的时间来开发云应用，现实是还有很大一部分开发者在其日常工作中都没有理由去接触云。假定这部分开发者没有充足的自由时间来体验云环境，那么你们会推荐他们做什么以便与最新的云服务和战略与时俱进？你们自己又是怎样跟上这种不停向前流动的云空间的呢？

Hall: 对于那些没有接触过云/公共云或刚出现的私有云的开发者来说，我发现两种很有用的方式非常重要。

1. 学习一般的分布式系统。这些包括分布式数据库、分布式计算（网格计算等）、通过自动化或大量其他选项进行的跨分布式环境的网络管理。这段时间也出版了很多书籍，这些都能在这上面给予他们很多帮助，因为很多工作都已经极大地学术化了（对那些实际上正试图利用分布式系统的编码人员或运营人员并不是很有用），但其中很多东西在日常的开发和运营中基本上没用。
2. 当这样做有用时候，尽量尝试在日常的开发过程中小规模地引入它们。即使没有用到云，从分布式角度而不是垂直式角度出发来构建某个系统也可以拥有强有力、有用性和健壮性这样的特性。下面就是我这段时间来一直坚决鼓励的一个做法的要旨：除非应用只是临时性的，否则请不要开发纯粹的垂直式应用。如果某个应用的期望生命周期超过一年的话，那么请按水平扩展的、可伸缩的架构来构建该应用，这样它在一个分布式系统环境之中也能很好地工作。

Mårtensson: 向云迈出第一步很容易。首先我想提醒的是最大的问题是，从广泛和普遍采用云计算方面来看，我们目前处于哪个位置？我是一名云计算的倡导者和忠实信徒，并且我真的很想相信现在我们正准备促进云的大爆炸。这就是说很多即使不是所有的信号都在指着这个方向：培训公司在这上面的兴趣在显著增加，顾问们注意到更多关于云的喋喋不休，更多的客户正在激起兴趣而提供商的市场份额也在持续增长。我实在看不到一些关于正在快速增加地采用云技术的反面迹象。传统寄宿选项仍会继续挣扎求生并尽一切能力来显示自己仍是可替代的选项，但在我看来这只是延缓了它们无法避免的命运的到来。特别如果你认为混合场景也是一种真正的云场景的话，我想我们应也将看到一个快速应用需求。如果我们谈论技术采用生命周期，我认为我们在甲板上已拥有了早先的采用者，我们正站在深渊的边缘，深渊将早先的采用者们与其它分割开来。

云平台的提供商们正在尽力做到很容易就能采用并平滑地过渡到各自产品。比如Microsoft最近对于某些试用场合就取消了信用卡的要求。为了回答这个问题，开始与云计算同行将会实际上已经是很容易的了。你不需要花费大量时间来上手。你可以仅仅需用几分钟时间来注册并获得一个试用的可运行版本。实际上拥有一个已分配给自己的MSDN订阅的每个人都已在云中有一个个人的开发/测试环境。所需要的就是[激活MSDN订阅上的Windows Azure](#)，这大概需要2分钟左右。大量的在线指南可以帮助你将自己的第一个应用部署到平台。例如[我博客](#)上的视频演示。确实地如果你有15分钟的话，我敢打赌你肯定能

将自己的第一个应用在Windows Azure云上正式运行起来。这可能是一件简单得也许微不足道的事情，但它真的很酷很让人耳目一新。

Piper: 我想PaaS所提供的任何东西都是瞄着提供一个无阻力的部署表面——AppEngine、Cloud Foundry、Heroku、OpenShift以及Azure的涉及PaaS的很多方面等等确实都是这样。如果在自己所选的平台上没有尽力提供一种在其上部署应用的简单方式，那么很有可能你开始就没有做对。当你开始寻找自己应用中的可伸缩性和数据访问方面某些问题时，学习曲线上仍然还有很多要去学习。

个人来说虽然我发现像O'Reilly这样的出版社所出版的很多不错的语言和编程指南书籍常常都有好几年的生存期，但由于云领域的快速创新，“云”相关的书籍显然还没有老到有这种火候。只要等待一个月，AWS就会已引入一个全新的API或调降了价格，某个PaaS提供商就会宣布有了新的合作者、插件或功能！这就是说在博客中挖来挖去并跟随Twitter上的那些能推荐很好的链接的家伙会更有用。我发现两个特别的来源是很有价值的。Github活动feed让我能跟随自己的联系人所评级或创建的新的项目、apps和库。DevOps每周简讯（云和开发空间的每周汇总邮件）也是一个获取关于最新进展的摘要信息的有用途径。

作者简介



Adron Hall: 软件架构师、工程师、程序猿、码农、分布式系统的拥护者。他是位多产的开源贡献者，[积极使用Github](#)来贡献项目。你可以在[CompositeCode.Com](#)上了解他的思想，还可以在Twitter上的[@adron](#)跟随他。



随他。

Magnus Mårtensson: 就职于瑞典的Active Solution顾问公司，担任云架构师/开发者。他是Windows Azure MVP、Windows Azure的业内人士、Windows Azure顾问。你可以在[MagnusMartensson.com](#)上阅读其著述，还可以在Twitter上的[@noopman](#)跟



Andy Piper: 倡导Cloud Foundry的开发者。他的日常工作是包含以下内容的有趣混合：技术市场、业务开发、与开发者交谈、各种会议上的公开演讲、撰写文档和示例、向工程师抱怨其中断了事情、博客和推特、还有就是组织活动。从[这儿](#)可以了解他更多东西，还可以在Twitter上的[@andypiper](#)跟随他。

观点 | View

2014年值得关注的9项技术

作者 张龙

[Andrew C. Oliver](#)是一位专业的软件咨询师。他从8岁起开始编程，从Basic与dBase III+开始。他最为人所熟知的就是创建了POI项目，该项目现在托管在Apache上。在Red Hat收购JBoss之前，他还是JBoss的早期开发者之一。Andrew是Open Source Initiative的前董事会成员以及现在的顾问。除此之外，Andrew还是Open Software Integrators的董事长与创建者，这是一家专业的服务机构，分布在达拉谟、北卡罗来纳州以及芝加哥。近日，Andrew撰写了一篇[文章](#)，谈到了2014年值得关注的9项技术。

2014年充满了各种预测，我们无法控制世界未来的样子，不过我们可以成为自身命运的主人。下面就是2014年值得关注的9项技术，与你一同分享。

1. 文档数据库

很多IT系统基本上包含了将数据结构写到结构化存储这一部分，同时又要求高并发的性能。文档数据库出现已经有很多年了，其中就有Lotus Domino的Notes Storage Facility，不过NoSQL与大数据革命又燃起了新的领域，其中MongoDB与Couchbase就是其中的佼佼者。

2. 键值存储

有时，你有一张很大的表，可以放到内存中。如果是网格，那么你就可以将内存中的表分发到多个结点上以加快写的速度。如果是个读多写少的小表，那么你可以将其复制到所有结点上，这样读就是个内存中的事情了。无论采用哪种方式，键值存储都值得你好好学习一下。几乎所有的键值存储都可以创建自定义的缓存负载器或是缓存存储，从而实现对RDBMS或是其他数据源的读与写。很多键值存储采用了“稍后写”或是队列写来实现对数据库的写操作。这个领域的典型代表是Couchbase、Memcached、Infinispan与GemFire。

3. 图数据库

从推荐引擎到社交网络和地理分析，再到生物分析，图数据库都带来了极大的便利性。对于传统的RDBMS来说，朋友的朋友这种查询是非常低效的，即便利用最新的特性也没有太大的起色，这是因为其结构就不对。虽然图数据库已经出现了很多年，不过只是最近的数据爆发以及个性推荐等领域才使其变得更加流行。这个领域的典型代表是Neo4j与Apache Giraph。

4. Google Drive/Apps

Google Apps是个办公效率套件。我无法想象再回到原来的通过邮件发送附件的那种方式。最近，我们通过基于JavaScript的宏实现了越来越好的自动化。所有的一切都存储在云端，因此我们可以放心地睡觉。除此之外，可扩展性特性意味着只要我们能够访问云端，那么我们就可以将文档直接与其集成，反之亦然。

5. On-premises搜索

我现在还是能看到有很多人在编写着大量的and/or/like等SQL查询，其实这么做不仅会导致严重的性能问题，还会产生不清晰的代码以及不易使用的接口。这时可以看看Google的服务，无论是数据库、文档还是各种文件系统都行。Apache Solr值得你好好看看。

6. PaaS

无论是公有云还是自己的私有云，你都需要手工安装各种操作系统、应用服务器与应用，然后提前选择好将要部署的服务器与VM数量，这是十几年前的做法。PaaS是未来的趋势，能够做到实时伸缩，自动完成重复性的任务。我们所广泛使用的平台有CloudFoundry、CloudBees与OpenShift。

7. 云IDE

前不久，我们全家在打扫壁橱，我9岁的孩子不认识壁橱里面的一个很大的金属盒子到底是什么，那是什么呢？我们在隔壁房间看着我们的孩子。“我不知道，好像是个Dell的什么东西”。我们发现他确实不知道塔式机箱到底是什么，因为自从他出生以来，我们就一直在使用着笔记本（不过他看到过一台1U服务器，因为我们用它做过Hadoop的测试）。我觉得云IDE可以做到一点，那就是让下一代不知道笔记本到底是什么。为何要在硬盘上安装IDE呢？为什么不打开浏览器，然后就开始编码呢？比如说Codenvy或是Cloud9。

8. Hadoop

无论是使用MapReduce进行复杂的分析，抑或只是想做些日志分析和审计日志，Hadoop都是这个行业中最为火热的一个选择。如果你尚未使用Hadoop做过一些试点项目，那么今年就要考虑做做了。如果已经使用过Hadoop，那么我希望你能在今年对Hadoop有更好的了解与掌握。

9. 集群/分布式文件系统

从集群到HDFS，可伸缩性存储是关键。今年，你要重新思考SAN了。至少，如果还没有尝试过可以先做个试水。我预测会有很多混合方式出现。

观点 | View

15个热门的编程趋势及15个逐步走向衰落的编程方向（上）

作者 张龙

[Peter Wayner](#)是InfoWorld的一名特约编辑，也是一个多产的作家。除了Info World之外，他还经常为纽约时报和连线杂志撰写文章。近日，Peter撰写了一篇文章，谈到了未来15个热门的编程趋势以及15个逐步走向衰落的技术方向，该文发表之后在技术社区中引起了较大的反响，也希望文中的观点能给各位读者带来一些启示。

程序员们普遍对时尚界嗤之以鼻，因为这个圈子中的趋势就像风一样变幻不定。裙子忽长忽短、颜色变来变去、领结时大时小。不过在技术界，精确、科学与数学却统治着一切。然而，这并不是说编程没有趋势可言。差别在于编程的趋势是由更高的效率、更好的可定制性以及更棒的易用性来驱动的。新的技术会让旧有的技术黯然失色。下面我们就来介绍一下未来15个热门的编程趋势以及15个逐步走向衰落的编程方向。并非人人都会同意文中的观点，不过编程令人着迷之处恰恰就是快速的变化、激烈的争论以及即时的反馈。

热门：预处理程序

冷门：全语言栈

几年前，如果有人创建了新的编程语言，那么他不得不自己编写一些程序将语言的代码转换为二进制位。后来，有人发现可以利用现有的一些工具和技术做到这一点。现在，有想法的人只需编写一个预处理程序即可，它会将新语言的代码转换为已有的拥有大量库和APIs的语言。

喜爱动态类型的家伙创建了Groovy，这是一个简化版本的Java。那些想要修复JavaScript的人们创造了CoffeeScript。还有很多语言，如Scala和Clojure等，他们运行在JVM之上，已经有了这么好的JVM了，为何还要重新发明轮子呢？

热门：JavaScript MV*框架

冷门：纯粹的JavaScript文件

很久之前，人们学习JavaScript的目的就是为了弹出一个警告框或是检查表单中的email地址是否包含了@符号。现在，HTML AJAX应用已经变得异常复杂了，没有多少人会从头编写代码。更简单的方式是采用一个优雅的框架，编写一些胶

水代码来实现业务逻辑。目前有大量的JavaScript框架，如Kendo、Sencha、jQuery Mobile、AngularJS、Ember、Backbone及Meteor JS等等，这些框架都可以帮助你很好地处理Web应用与页面的事件与内容。

热门：CSS框架

冷门：纯粹的CSS

曾几何时，为了向网页中添加一点活力，你需要打开CSS文件并加入一些诸如font-style:italic之类的命令，然后再保存文件。现在的网页已经变得非常复杂了，几乎不可能通过这样几条简单的命令就能达成所愿。改变一处的颜色也许会导致其他地方的颜色显示不正常，老话说的好：万物皆有联系。这正是SASS及Compass等CSS框架的用武之地，他们提供了一些编程元素，比如说真正的变量、嵌套的块等编程结构，这些东西在编程领域不是什么新玩意，不过对于设计来说却是一个伟大的进步。

热门：SVG与Canvas

冷门：Flash

过去，Flash令人着迷，艺术家们都偏爱着Flash。其渲染的效果如此漂亮，很多聪明的艺术家都使用了大量的Flash创作了不少精致的效果。现在，JavaScript也可以做到过去只有Flash才能做到的那些效果，浏览器制造商与开发者们都为Flash的退出而欢呼雀跃。他们看到SVG (Scalable Vector Graphics) 等新格式与DOM层更好的集成。SVG与HTML由大量的标签构成，这对于Web开发者来说是很容易使用的。此外，Canvas对象也提供了大量的API进行绘制。这让我们相信，已经没有多少继续坚守Flash的理由了。

热门：大数据（不使用Hadoop进行分析）

冷门：大数据（使用Hadoop）

每个人都想成为风云人物，如果不，那他就会换个地方让自己脱颖而出。因此，“大数据”这个词儿的流行也就变得不那么让人惊奇了。有意思的是，很多问题并没有那么大，也没必要使用什么大数据解决方案。当然了，像Google或是Yahoo这样的公司会追踪人们的Web浏览；他们拥有的数据量是非常庞大的。不过对于大多数公司来说，他们所拥有的数据量可能一个普通的PC就装得下。肯定有公司需要使用不少机器，并行运行Hadoop，然后希望快点得到计算结果，不过很多公司其实并不需要这么做，他们只需要单台机器就行，根本没必要搞什么Hadoop。

热门：游戏框架

冷门：原生游戏开发

曾几何时，游戏开发意味着招很多开发者，从头开始使用C编写代码。当然了，这么做的成本也是相当高的，不过看起来好像不错。现在，没有人能够承担得起编写这么多代码的代价。大多数游戏开发者都开始使用Unity、Corona或是LibGDX等库来构建系统了。这样，他们就无需再处理这些细节信息了，可以将精力放在游戏情节、故事、角色以及艺术上了。

热门：单页面Web应用

冷门：网站

还记得通过URL访问只有静态文本和图片的网页时代么？那时，将所有信息放到网页上就可以做一个“网站”了。新的Web应用是包含着内容的数据库的前端。当Web应用需要信息时，它会从数据库中取这些信息，然后将其显示出来，没必要再像之前那样使用了。数据层与展示层和格式层是完全分开的。移动计算的出现是一个巨大的促进因素：单一的、响应式设计的网页看起来像个应用，同时还避免了App Store频繁提交审查的烦恼。

原文链接：<http://www.infoq.com/cn/news/2014/02/15-hot-programming-trends>

相关内容

- [15个热门的编程趋势及15个逐步走向衰落的编程方向（上）](#)
- [IT领域2014年发展趋势](#)
- [企业移动性的当前趋势](#)
- [编程：思考还是打字](#)
- [探讨编程语言语法](#)

观点 | View

15个热门的编程趋势及15个逐步走向衰落的编程方向（下）

作者 张龙

[Peter Wayner](#)是InfoWorld的一名特约编辑，也是一个多产的作家。除了Info World之外，他还经常为纽约时报和连线杂志撰写文章。近日，Peter撰写了一篇文章，谈到了未来15个热门的编程趋势以及15个逐步走向衰落的技术方向，该文发表之后在技术社区中引起了较大的反响，也希望文中的观点能给各位读者带来一些启示。

程序员们普遍对时尚界嗤之以鼻，因为这个圈子中的趋势就像风一样变幻不定。裙子忽长忽短、颜色变来变去、领结时大时小。不过在技术界，精确、科学与数学却统治着一切。然而，这并不是说编程没有趋势可言。差别在于编程的趋势是由更高的效率、更好的可定制性以及更棒的易用性来驱动的。新的技术会让旧有的技术黯然失色。下面我们就来介绍一下未来15个热门的编程趋势以及15个逐步走向衰落的编程方向。并非人人都会同意文中的观点，不过编程令人着迷之处恰恰就是快速的变化、激烈的争论以及即时的反馈。感兴趣的读者还可以参见本篇文章的第一部分。

热门：移动Web应用

冷门：原生移动应用

假如你有一个关于移动方面的好点子。你可以为iOS、Android、Windows 8，也许还有BlackBerry OS各编写一个应用。每个应用都需要单独一个团队，使用不同的编程语言完成。开发完成后，你还需要将应用提交到应用商店进行审查，最后才能被用户下载使用。此外，你还可以构建一个HTML应用，将其放到网站上，应用可以运行在所有的平台之上。如果需要做些修改，那么你无需回到应用商店，祈求能够快些通过审查。现在的HTML运行速度已经越来越快了，它完全可以与原生应用展开竞争，即便是那些复杂、交互非常多的应用也没什么问题。

热门：Android

冷门：iOS

几年前，Apple的App Store还是一家独大，不过时间改变了一切。虽然iPhone与iPad还是拥有非常多的粉丝，他们喜欢其精致、丰富的UI，但Android的销量却在节节攀高。有报告显示70%以上的智能手机销量来自于Android。

原因很简单，那就是价格。虽然iOS设备保持了一个比较高的价格，不过Android世界中有太多的竞争者，他们所生产的平板价格甚至只有iPad价格的1/5，省钱总是硬道理嘛。除了价格之外，开源也是一个不容小觑的因素。任何人都可以在市场中参与竞争，实际情况也是这样的。有大的Android平板，也有小的手机；有Android相机，甚至还有Android冰箱。

热门：GPU

冷门：CPU

在软件还很简单，指令可以在一行中清楚显示的时候，CPU是计算机之王，因为它做了所有繁重的工作。现在，视频游戏中有大量并行运行的图形计算，一块显卡的价格动辄就5、600美金，一些执着的玩家甚至会使用多块显卡。这甚至比很多一般的PC还要贵，除了游戏玩家外，计算机科学家们也将很多并行应用转到GPU上运行，速度比之前快了百倍以上。

热门：GitHub

冷门：简历

没错，你可以通过看书等方式来学习，不过，阅读实际的代码却更加直观和有意义。程序员是否写了足够好的注释？是否花时间将大的类拆分成若干各司其责的小类？架构是否还有扩展的空间？这些问题都可以通过查看代码得到答案。

这也是为何现在在找工作时有过开源项目开发经历会变得更加吃香的原因所在。从私有项目中分享代码是比较困难的，不过开源项目可以走进每个人的生活。

热门：租赁

冷门：购买

以前，公司会建立自己的数据中心、雇佣专门的人来维护他们所购买的计算机。时至今日，很多公司开始租赁计算机、数据中心、雇员，甚至按照小时数来租赁软件。这是个非常好的做法，也会为公司节省很多成本；同时，还会保证计算能力。

热门：Web界面

冷门：IDEs

很久之前，人们使用命令行编译器。后来，有人将其集成到了编辑器和其他工具当中，创造出了IDE。现在，IDE有被基于浏览器的工具所替代的趋势。在基于浏览器的工具中，你可以编写代码、创建系统。如果不喜欢WordPress的工作方

式，那么你可以通过它自带的编辑器修改代码并立即生效。你可以通过微软的Azure编写JavaScript胶水代码。这些系统基本上都没有提供很好的调试环境，而且在编辑生产代码时也存在着一定的风险，不过这个想法却是非常棒的。

热门：Node.js

冷门：JavaEE、Ruby on Rails及PHP

服务器的世界总是依赖于各种线程模型，不过这种方式会导致程序员所编写的各种低效、不负责任的代码影响到操作系统的效率。无论程序员编写的代码有多么差劲，操作系统总是会在各个线程间切换，从而平衡整体性能。

Node.js带来了JavaScript回调这种编程模型，代码运行速度也绝对够快。这种一开始只是用于弹出警告框的玩具语言的变化超出了很多人的想象。突然之间，创建新线程的开销变得很显著了，这时Node.js来了。如果程序员代码写的不好就会出现问题，不过让程序员清楚了解资源限制有助于他们编写出更快的代码。

Node.js的世界也因让浏览器与服务器之间保持和谐共处而获益匪浅。同样的代码既可以运行在浏览器端，也可以运行在服务器端，开发者可以在两端快速移动，也可以更好地完成功能。因此，Node.js已经成为互联网界最炙手可热的技术之一。

热门：Hackerspaces

冷门：大学

一个是4年250,000美金，一个是每月50美金，如果提前支付还有折扣。Hackerspaces正在不断驱动着创新，同时又没有大学那么多的开销。他们在创造着社交网络，影响着创业公司，没有官僚、没有政治。其课程不需要持续整个学期，这种特性非常适合于快速变化的技术世界。

原文链接：<http://www.infoq.com/cn/news/2014/02/15-hot-programming-trends-2>

相关内容

- [15个热门的编程趋势及15个逐步走向衰落的编程方向（上）](#)
- [IT领域2014年发展趋势](#)
- [企业移动性的当前趋势](#)
- [编程：思考还是打字](#)
- [探讨编程语言语法](#)

专题推荐语

云计算的三个层面

本期专题的三篇文章分别在云计算的不同层面：《Windows Server基础架构云参考架构：硬件之上的设计》最靠近底层，偏重硬件层面，面向需要自己搭建私有云基础架构的运维人员；《基于AWS的自动化部署实践》介绍了Autodesk基于AWS做出他们的SaaS服务的过程中如何解决运维的问题，他们的调研过程和实现思路无论对研发还是运维都会很有启发；《探索有道云笔记工程师团队的开发协作模式》是一篇针对有道云技术总监蒋炜航博士的采访，介绍他们这样一个客户端、前端、服务器端、存储系统的研发都在一起的团队是如何进行合作的，如果你也在做云服务/云产品的研发，也许会对你有一些参考价值。

Happy reading !



本期专题：云计算的三个层面 | Topic

探索有道云笔记工程师团队的开发协作模式

作者 杨赛

他是2013年QCon北京云计算专题的出品人，也是[2014年QCon北京](#)移动应用专题的出品人。他的团队往下做到云计算基础架构，往上做到客户端。他本人从80年代末起接触编程，在大数据处理、云计算、以及分布式系统等方面都玩过一圈。他是[蒋炜航](#)，网易技术总监，目前全面负责有道云笔记业务。

在这样一个团队中，后端和前端团队在同一个产品上工作，有紧密的协作关系。但是，底层的软件研发和应用层的软件研发毕竟有很大的区别，他们在代码提交和管理模式、测试机制、代码的交付周期、反馈和监控体系方面都有怎样的异同？在本次采访中，蒋炜航博士会介绍有道云笔记团队的一些实践。

InfoQ：您在去年QCon北京出品云计算专题，今年则出品移动App的专题。这也对应了有道云笔记在技术上的一个特点，就是从底层到客户端都是你们自己来研发、维护、运营的。能先简单介绍一下跟有道云笔记相关的技术团队都有哪些，各自负责哪些方面吗？

蒋炜航：我们底层有基础架构组，这个组在云笔记业务之前就建立了，服务的对象不单单是云笔记，还服务有道的很多业务。这个组也在我的团队，做的东西跟Hadoop（HDFS、HBase、MapReduce）这些差不多，是自己研发的一套系统。底层提供的东西比较基础，只是存储和计算服务，对所有业务通用，本身不包含业务逻辑。

业务逻辑的开发都在服务器端的团队，这个团队专门是负责云笔记的服务器端的，负责处理同步、多版本、提供API等服务。

客户端包括PC、Mac、iOS、Android、Windows Phone的客户端，以及web端和浏览器插件。基本上每一层的团队都是几个人来做，不是很多。

此外我们还有个精干的研究团队，这个部门做的是些三到六个月才有成绩的东西。方向主要是应用研究，研究的目标不是发论文，而是把成熟的技术做到产品中。比如编辑器，我们需要做跨平台的编辑器，过去大家比较熟悉的就是Office和Google Doc，而以前很多Web based的编辑器并不适合跨平台终端的需求。我们一开始也是用Web编辑器做起来的，但是并不是很适合。我们也不可能像Office那样用几百个人去做一个编辑器出来，所以研究部门做的事情就是，花一定的时间找到合适的方法来提升编辑器的体验。提高跨平台的体验

是一个很模糊的目标，不确定性很大，具体用什么做法要很多研究和尝试。这个团队还研究一些NLP和手写输入方面的项目。

InfoQ：有些软件产品的发展思路是求快，尤其在产品推出初期，要以最快的速度推出新的特性以验证产品的可行性；有些软件产品的思路是求稳，尤其在产品积累了一定用户量的时候，会更加关注软件的稳定性、安全性，升级的时候不要对现有用户造成负面影响。有道云笔记现在处于怎样的阶段，是偏重求快还是求稳？

蒋炜航：做互联网产品永远是要快的。同时，云笔记是个人的信息、知识管理工具，稳定性非常重要。要快，同时也要坚守产品质量，我们必须要两者兼顾。

可以这么说：我们的目标就是，最大化高质量产品的输出。我们无论做什么事，都要以优化它为目标来做。敏捷、测试、监控，都是很重要的手段，但相比之下，清晰的目标才是更加重要的。比如，什么叫做快？新版本发布多了就是快吗？引入新特性的频率高就是快吗？我觉得不是。新版本发布，可能会有很多bug；新特性引入，可能是用户根本不关心的，可能你80%的用户从来都不会用到。所以这样的快是没有意义的。

我们对快的定义是，要让有效的产品尝试的速度尽量的快。你发一个新版本来测试，是否获得了更多对用户的理解？新功能是否让用户在某个场景下的需求得到更好的满足？用户是否更加活跃了？有效的快速是满足业务目标的速度，这是通过不断为用户提供更合适的功能来实现的。

我们现在发布重要的版本，会先在内部做高保真原型，在内部试用，从内部非常快的得到有效的反馈。然后我们会做很多的小范围用户测试和AB测试，比如在不同的渠道分发不同的软件包，来验证新的交互、新的功能，改掉原有的问题。另外我们还有一个很简单的原则：我们客户端覆盖这么多的平台，但是一个新功能的引入可能先只在一个平台上做，这样我们可以很快通过用户分析，了解用户是否需要这个功能，确认有用了才在全平台铺开。否则，就不引入其他平台，这样其他平台可以有更多机会尝试其他的东西。

在这样的思路下，我们测试阶段的服务器端也可以做的很简单，比如我测试覆盖就几百几千人，那就先用Node.js搭一个服务给他们用就好了，性能问题可以以后再考虑。很多时候，做面向三五个专家用户（资深产品经理）的内测也已经足够了。

做软件开发不是做选择题，要么选A要么选B。只要目标明确，具体用什么手段都可以的。

InfoQ：你们后端研发团队和客户端研发团队在开发模式上的差异大么？

蒋炜航：本质上没有任何差别，大家都是为了业务的发展。实际上，我们有很多好的移动端研发工程师是从服务器端转过去的。很多应届毕业生是没做过前端和移动端，他们一开始就是对服务器端感兴趣。我们就会让他先做服务器，其中一些人就会转岗到客户端去。基本上，优秀的工程师在哪里都是优秀的，所以我们内部鼓励full stack工程师，可以从头到尾把高保真原型做出来，包括前端的JS、iOS、Android App、后台服务。我们所有的工程师都要有敏捷的思路，以及以用户为中心的认同感，这些方面是一致的。

当然，因为前后端软件的特性不同，肯定会有一些差异。比如，服务器端对性能更加敏感，所以会关注很多提高性能的手段。当然这些思路，客户端也可以引入。另一方面，敏捷在客户端是比较常见的，因为客户端是功能驱动开发，对交付速度要求更高。当然，服务器端也要有一定的敏捷思路，不能说要我提供十个接口，我就要做俩月。我完全可以用一周时间，用一些现成的技术，比如MySQL，把这个接口先提供出来，让客户端的开发能够用起来，之后再考虑性能问题，是不是要移到并行文件系统上面去，等等。

总之，在规定的Sprint之内，我们一定要做出一个完整的版本。无论是客户端工程师还是服务器端工程师，他们的工作质量都是很容易判断的。

InfoQ：两个团队的交付周期是一样的吗？

蒋炜航：交付周期都是一样的，我们的节奏是一个月，也就是每个月都会有新版本出来，只不过有些不对外发布。

我负责这个业务，具体来说我并不关心客户端团队是不是在满负荷工作，或者服务器端团队是不是在满负荷工作，我关心的是我们在这个周期内，产品经理想要展现给用户的价值是否高质量的完成了，是否能给用户提供一个高质量的版本，这是最重要的。

InfoQ：两个团队的代码提交方式、代码review机制、测试机制是怎样的？

蒋炜航：我们内部用SVN提交代码，每个项目提交到各自的repo里面。代码审查是结对review的方式。

测试方面，主要由工程师自己写单元测试，交到自动化测试系统里去跑。我们

有强制的测试覆盖率的要求。当然，服务器端和客户端跑的测试是不一样的，服务器端需要做更多的压力测试，而客户端在自动测试外还要做很多功能测试。

InfoQ：在监控和反馈机制方面，两个团队又分别是是如何去做的？

蒋炜航：服务器端主要是服务器健康状态的监控。客户端有我们自己写的crash report机制，还有一套自己的BI系统，用于收集业务方面的指标。

整个产品的日志跟功能是一样重要的，一起开发、测试。有健康的日志系统，才能了解用户的使用情况。这套日志系统检测的指标有几百项，其中一些比较重要的指标包括日活跃用户量、各个功能的使用率、用户满意度等等。其中用户满意度是关键指标，也就是NPS (net promoter score)，我们所有的产品中都有这个分数，这个分数体现了用户对产品的喜爱度。

InfoQ：客户端团队相当于是底层团队的客户，这两个团队之间是如何进行沟通的？

蒋炜航：一个一个sprint来走。我们每个sprint先开客户端的会议，事先跟server端沟通好，要做什么功能，可能有哪些接口，之后就是非常自主的形式了。比如客户端安排要做功能A，server端做接口A，那么这两边的人就会自己去沟通，讨论API该如何设计，协议如何做。

敏捷小组的特性就是自发性，我们没有规定谁是谁的客户，或者谁lead谁。我们就是设定了目标，分配任务到每个组，之后任何人都能驱动。比如很常见的，客户端说某个接口做的不对，会跟server端一起去改接口，不会说客户端不管API的设计，给什么用什么，大家肯定是一起来设计的。因为我们的工程师是full stack，所以这点是比较自然的。

另外，我们的产品、测试、运营也都跟研发团队是坐在一起的，这几个团队之间的沟通都很多。我自己现在是60%在产品上，20%盯研发进度，20%在运营和市场上。

原文链接：<http://www.infoq.com/cn/news/2014/01/full-stack-engineers-develop>

相关内容

- [文化与效能：豌豆荚如何通过文化建设来提升团队效能](#)

本期专题：云计算的三个层面 | Topic

基于AWS的自动化部署实践

1. 背景

作者 徐桂林

过去几年中，社交、移动和云计算深刻改变了整个互联网的格局。作为设计软件领域的全球领导厂商，Autodesk也于2009年正式开始从传统桌面设计软件提供商向在线服务、协作和移动端设计转型。在这次转型中，公司充分利用现代云计算的巨大优势给客户带来了大大超过传统桌面软件的处理能力、用户体验和性价比。其中AWS是目前公司服务的主要运行平台，每年在此投入千万美金级别。

1.1. 传统软件交付的挑战

在过去的30多年里，Autodesk拥有了非常多的桌面设计软件（如AutoCAD，Maya，3dsMax等）。由于设计软件经常需要处理超大的数据集合（如整个上海中心的设计模型）和极其复杂的运算逻辑（如阿凡达电影画面的绘制），其软件尺寸一般都比较大（GB级别）。以前，客户基本都是通过互联网下载、快递或者分销商获得软件安装包，整个过程比较耗时。另外，软件的升级、安装和维护也是一个非常大的工作量（一些大的设计公司要购买上千份软件拷贝）。尽管公司软件已经支持基于应用程序虚拟化的集中管理模式，但它还是有如前期基础设施建设成本大，服务能力缺乏弹性，仍需要专职运维人员等明显的缺点。所以，提升软件交付的用户体验成为我们必须面对的一个问题。

如大多数人所猜测，SaaS成为我们的第一个尝试方向。在2006年，Autodesk实验室开始尝试以SaaS提供设计软件服务的可能性，并且取得了不错的成绩。但是，目前SaaS应用仍然面临着浏览器能力限制、大数据传输慢等诸多问题，无法给专业设计师提供传统桌面软件一样的体验和设计能力。

1.2. 云计算带来的新可能

随着云计算的兴起，以AWS为代表的基础设施云提供商让我们有可能以一种全新的方法去解决这个问题。我们可以利用基础设施云的强大后台来帮助用户运行虚拟化的软件实例。用户无需下载、安装和维护这些软件，只需要链接到互联网上就可以在线使用我们的软件。而且按需付费以降低使用成本。基于此，Autodesk实验室在2009年开始这个尝试（注：该服务已于2013年成为公司云平台产品的一部分），并选择了AWS做作为我们的后台云服务提供商。选择AWS有下面的几个原因：

- 需要基础设施云（IaaS）。现在的平台云（PaaS）大部分都是为Web服务准备

的，并不适合我们运行虚拟化实例的要求。

- 需要强大的弹性运算能力。Autodesk设计软件对于计算的需求都很大，而计算能力的成本不低。所以，弹性计算能力能让我们很好的控制成本。AWS EC2在这方面非常符合我们的需求。
- 需要丰富的服务。除了运算能力，我们还需要给用户提供海量设计数据的存储，快速的数据访问，安全的访问控制等。AWS云服务在这方便服务非常完备，而且相互集成很容易。
- 需要稳定的服务。AWS EC2能够提供超过99.5%的弹性计算可用率，能为我们建立可靠服务提供坚实的基础设施。
- 需要全球化部署。Autodesk是一个在全球提供软件、服务的公司，所以我们希望基础设施提供商也能全球布局。而AWS已经在全球建立多个数据中心。

2. 自动化部署

在完成服务的基本实现并上线服务后，整个后台的维护和部署成本在不断加大。尤其考虑到我们需要高频（每个月更新一次）、多地（多个AWS数据中心）部署服务并同时需要维持高的服务可用性，构建一个自动化的部署系统成为必须要做的事情。

2.1. 设定目标

作为一个运行在AWS上的服务，我们在设计之初就开始思考AWS给自动化部署带来的新可能和挑战。在我们看来，针对AWS上服务的自动化部署需要特别关注到下面的一些特点：

- 基础设施的服务化。在AWS中，你可以利用类似Cloud Formation服务在很短时间（几分钟）获取你所要的所有基础设施（包括运算、存储、网络、IO等），而这些基础设施已经按照你的要求自动配置、连接好。所以，我们可以让自动化部署把基础设施的管理也包括进来，而这在传统的数据中心模式下很难实现。
- 支持弹性资源。因为需要支持弹性，整个服务要在运行时动态加入新的基础设施，如计算单元等。自动化部署系统需要能让新加入的基础设施立马投入服务。
- 保护数据更为重要。在传统的数据中心，我们可以让部署完全在内部网络完成，在确认好所有的安全配置后再上线。但是AWS是一个公有云服务，你的所有部署其实是在公有网络上完成的。所以，我们必须在自动化部署中充分考虑到数据安全的问题。

结合上面的特点、DevOps的普遍实践和项目的实际情况，我们给整个自动化部署系统定下了下面的目标：

- 一键式部署：必须尽可能的自动化所有部署过程，包括基础设施的创建和部署。
- 多环境支撑：必须能够适应于Production、Staging和Development环境。
- 无服务中断：必须能够无缝的进行服务升级、切换。

- 随时可回滚：必须可以很容易的回滚到前面的版本以处理意外问题。
- 安全性检测：必须在确认部署环境的安全设置已经满足条件后才开始做部署工作。

2.2. 整体架构

在确定目标后，我们进行了技术选型，希望能够找到既很好支持AWS相关自动化操作，又能集成DevOps优秀实践的方案（注：那时AWS还没有推出OpsWorks服务）。但最后并没有找到合适的方案，于是决定自己来实现整个自动化部署系统。经过几轮的改进和实现，现在的自动化部署系统整体结构如下：

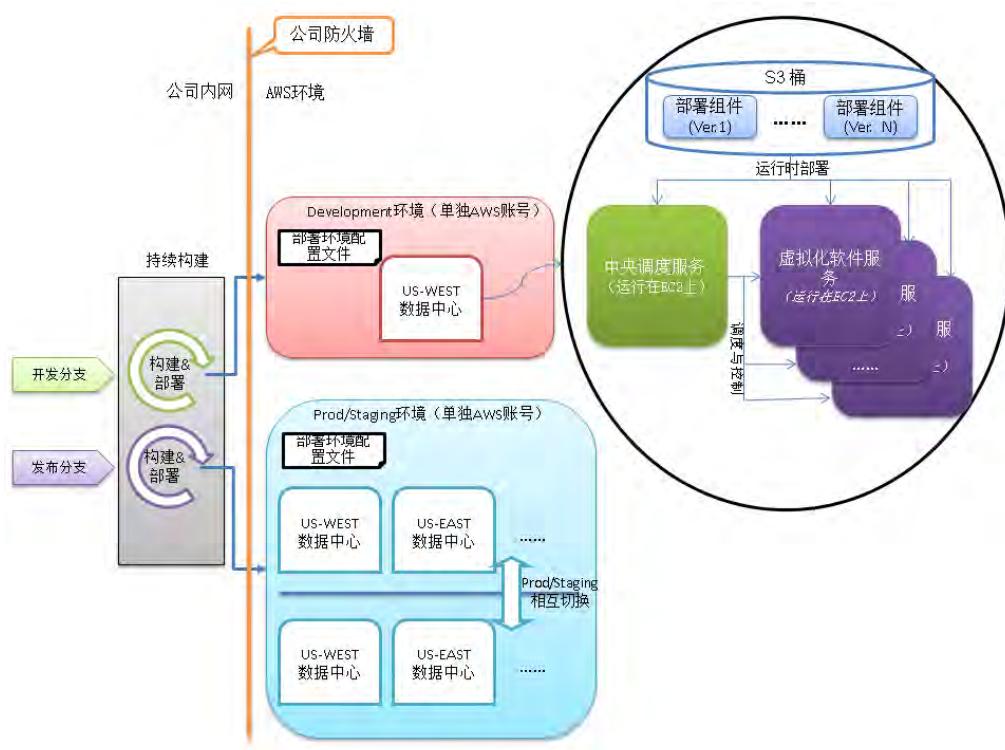


图1：自动化部署系统整体架构

类似于传统的自动化部署，我们也同样有开发分支和发布分支，并与持续构建系统对接。当开发人员提交代码后，相应的构建就会在代码所在的分支自动触发。在完成代码编译和集成的自动化测试（目前主要是单元测试）后，将产生相应部署组件并存放在构建系统中（目前，每个构建会产生所有的系统组件并拥有同样的版本号）。至此，整个构建过程完成。

为了支持多环境部署和安全隔离，我们使用两个独立的AWS账号来运行不同的环境。其中开发环境在一个AWS账户内，只部署开发分支的构建。而生产系统则在另外一个AWS账户中，其下运行Prod/Staging两组环境。发布分支永远只会向Staging环境部署并在完成最后验证测试后与当前Prod环境进行热切换，从而达到无服务中断的目的。

2.3. 一键部署流程

在完成自动化持续构建后，我们就可以部署其中的任意版本。当部署某一构建版本时（无论开发分支还是发布分支），整个流程如下：

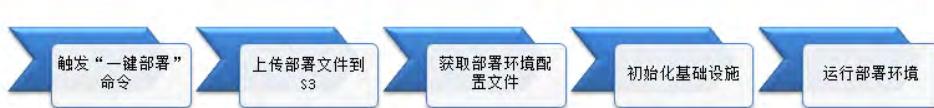


图2：一键式部署流程

1. 触发“一键部署”命令：在构建系统中选择好要部署的分支和版本，直接一键点击触发命令。
2. 上传部署文件到S3：如图1所示，部署组件是在运行时动态下载安装。AWS S3 提供在线存储并能够方便的下载到EC2实例中，所以把部署组件上传到S3中最合适不过。为了支持多版本并存和部署回滚，所有上传到S3的部署组件都按版本号分文件夹存储。
3. 获取当前部署环境的配置文件：部署环境配置文件存在相应AWS账户下的S3中。它定义了当前AWS账户下面的部署配置，包括需要部署的数据中心列表，每个数据中心下面的基础设施描述（Cloud Formation Stack）。由于Prod/Staging环境都在同一个AWS账户下，每个数据中心都会有两组Cloud Formation Stack配置（分别用于Prod和Staging环境）。部署系统会选择当前Staging环境的Cloud Formation Stack作为下一步的部署目标。
4. 初始化部署目标基础设施：根据当前选中的Cloud Formation Stack初始化整个基础设施，包括启动相应的EC2实例，绑定Elastic IPs等。
5. 运行部署环境：在整个基础设施运行起来后，EC2实例第一步就是自动做运行时部署（利用操作系统的启动脚本实现）。具体运行时部署细节如下：

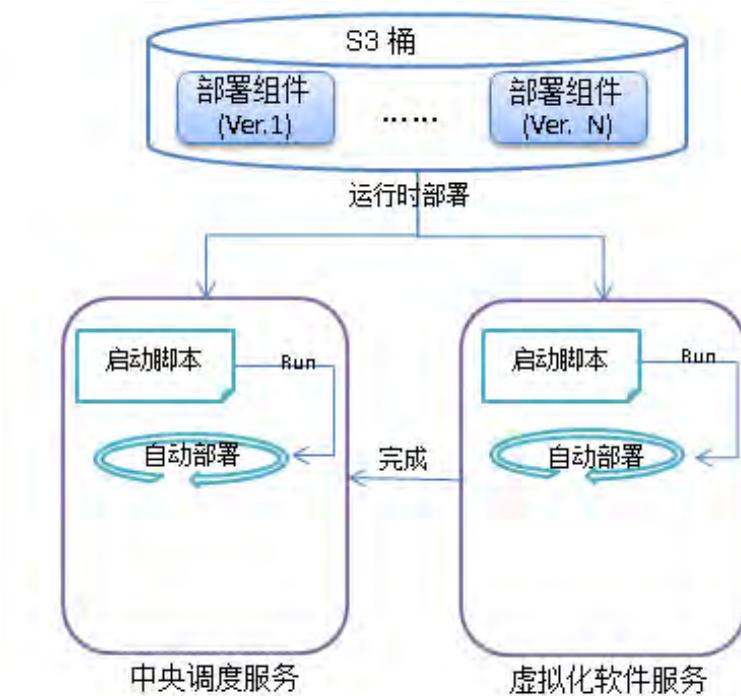


图3：运行时部署

- EC2实例在完成整个自动化部署中要及时进行有效性检测，如检测下载组件包是否完整正确，组件安装是否成功，期望的服务是否可以正确访问等。在完成所有部署和有效性检测后方加入的正式服务系统并处理用户请求，否则及时报告运维团队进行处理。

2.4. 为什么选择运行时部署

看到上面部署流程，相信很多人会问一个问题：AWS不是可以通过虚拟机镜像（AMI）来启动EC2实例，为什么不把上面的部署组件直接烧入AMI？这样在EC2实例启动后就直接使用而无需做运行时部署。其实，我们的最初解决方案就是把部署组件直接烧入AMI，但很快就发现了这种方案的局限：

部署组件的变化是非常频繁的，尤其是在发布前，一天都能够产生上十次的变化。这就意味着自动部署系统可能需要频繁产生AMI。而AMI的创建过程并不快（10分钟~1小时），并且过多的AMI也会造成管理问题和额外的存储成本。

作为一个平台项目，各个产品会在我们的平台上运行。我们希望提供给各个产品部门的基本AMI是平台版本无关的。这样产品部门在基本AMI基础上部署它们产品并重新生成的产品AMI也是平台版本无关的。于是产品AMI可以不做任何修改就能够快速采用最新的平台版本。具体如下图所示：

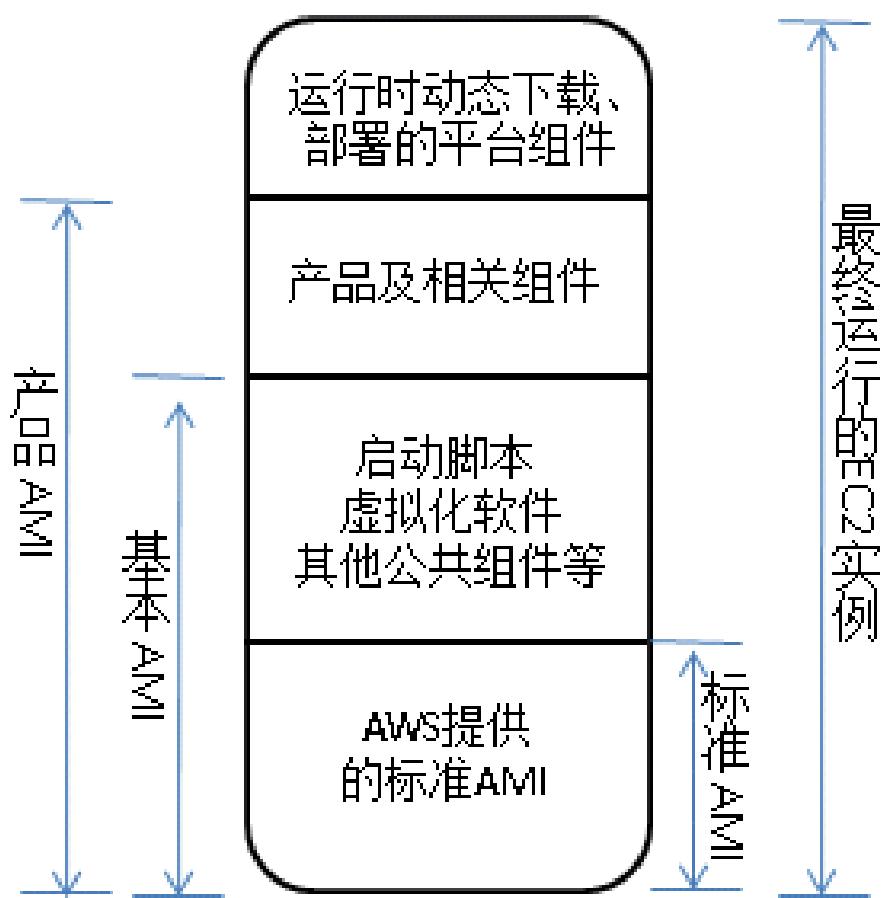


图4：自动化部署中的AMIs

为了保证图4中的基本AMI和产品AMI是平台版本无关，我们就需要保证烧入AMI的所有部分都是能够独立于平台版本的。但是，启动脚本需要做平台组件的运行时部署，显然这个运行时部署脚本也会随着平台更新不断变化，所以我们无法直接把整个运行时部署脚本放到启动脚本中。针对这个问题，我们的解决方案就是把整个运行时部署脚本分成两部分。一部分做平台组件的安装、检测工作，并随安装组件存到S3中，而另外一部分只做基本不会改变的事情（下载平台组件包并调用自动化部署脚本）并设置为操作系统的启动脚本。这也是图3中启动脚本和自动化部署分开两部分的原因。

正是因为采取了上面的AMI生成、管理策略，我们的产品部门基本上不用太频繁的更新它们产品的AMIs（除非产品自身有更新），真正做到平台版本和产品版本的去耦合，极大的降低了运维成本。

2.5. 运行时版本选择与回滚

由于产品AMI是平台版本独立的，以它为镜像运行起来的EC2实例无法知道应该

去下载哪一个版本的平台组件。幸运的是AWS的EC2服务提供了“User Data”机制。简单来说，就是在启动EC2实例的时候可以传入一段用户自定义数据给AWS。当这个实例运行起来后，实例内部程序可以通过调用AWS EC2的元数据服务取得先前传入的用户自定义数据。于是，我们可以把当前需要运行的平台版本号以“User Data”的方式传给EC2实例，然后让启动脚本读取相应版本号并下载相应版本组件来部署。

同样，基于“User Data”机制和平台版本无关的AMI，我们非常容易得实现版本的回滚。只需要修改传入给“User Data”中的版本号并保证要回滚到的平台组件版本仍然存在于S3中即可。即使是从零开始回滚到以前版本也可以在几分钟内完成（主要时间花费在启动EC2实例上）。在实际运营中，因为已经有了Prod/Staging的热切换，我们一般会在切换上新的版本后保持原来的版本运行一段时间（这段时间一般是问题的高发期）以便做到秒级的回滚。

2.6. 关于安全

如前面所说，我们需要格外关心在AWS上自动化部署的安全问题。在我们的实践中，时刻会遵循最小授权原则并利用AWS中的IAM服务实施，具体体现在下面的两个原则：

- 不要让管理员之外的任何人直接使用AWS根账户（包括它的Access Key和Access Security）。取而代之的是创建专门的IAM User并给予其必须的权限
- 不要在公司防火墙之外使用任何账号的Access Key和Access Security。取而代之的是使用IAM Role来获取EC2实例运行时需要的资源访问权限。

例如，我们的构建系统需要访问多项AWS服务资源（如S3、EC2、CloudFormation等），而构建系统又在防火墙内部，所以我们创建专用的IAM User来做自动化部署并给予构建系统需要的资源访问权限。另外，EC2实例需要访问S3并下载部署组件，所以EC2应该以专用的IAM Role来运行并在IAM Role中给予相应的S3桶只读属性。

2.7. 为什么不是AWS OpsWorks

熟悉AWS服务的人可能都知道Amazon已经在2013年发布了它的DevOps服务：OpsWorks。我们的自动化部署系统为什么没有选择这个服务呢？最直接的原因就是我们开始构建自动化部署系统时候，AWS还没有发布OpsWorks。在AWS发布OpsWorks后，我们对此做了仔细调研，并决定继续使用目前的系统。要了解其背后原因，就要完整理解AWS提供的一系列应用程序管理服务（Application Management Services）之间的关系。这里，让我们首先看看AWS文档中这张示意图：

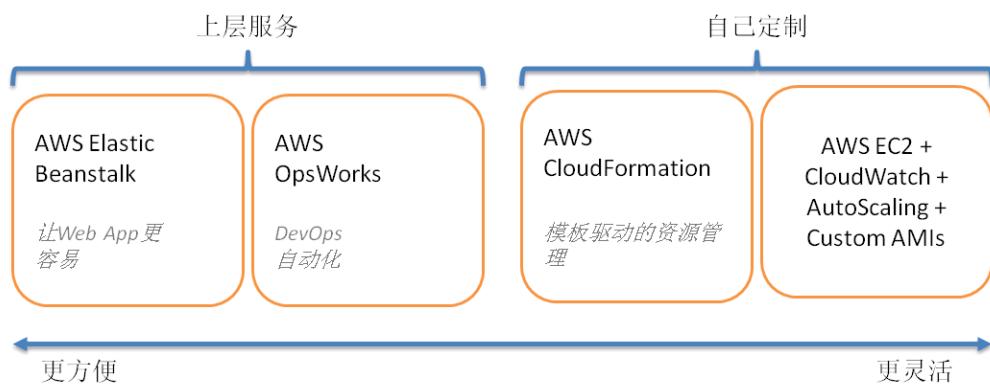


图5：AWS中的应用程序管理服务

就如上图所示，AWS为各种应用场景提供了不同的解决方案。由于针对的应用场景不同，这些服务的关注点也就不同。例如，AWS Elastic Beanstalk主要是为Web应用程序提供快速部署服务（有点类似国内SAE、BAE等服务），它帮助开发和运维人员隐藏了大量的底层细节，非常方便上手部署应用。但是，该服务在管理底层基础设施架构上就基本没有多少灵活性，无法适应项目的个性化需求。就我们的服务而言，它开始于最右边的完全自己定制方案。在AWS推出Cloud Formation后，为了提高系统的效率和自动化程度，我们迁移到以Cloud Formation为基础的新方案。但是我们没有继续向AWS OpsWorks迁移。究其缘由，让我们先多方面比较一下AWS OpsWorks和AWS Cloud Formation：

特性	AWS Cloud Formation	AWS OpsWorks
基础设施架构	以Stack的方式管理整个基础设施，你可以以任何你想要的架构组织你的基础设施。	遵循常见的架构实践，以Stack、Layer和App为核心观念来管理整个服务架构，并利用Chef来把App自动化部署到各个Layer上。尽管它也有很大的灵活性，但是你需要把自己的基础设施架构映射到上面的这些概念中以实施该服务。
AWS资源管理	支持几乎所有的AWS资源。你可以在Cloud Formation的JSON模板中方便地加入各种AWS资源。Cloud Formation会自动帮你管理各种资源之间的依赖关系。你也可以自定义一些资源之间的逻辑依赖关系。	支持主要的AWS服务资源（如EC2、EBS、ElasticIP, ELB等），并利用这些资源构建常见的Layers。当然你也可自己加入一些非内建的资源（如RDS服务）到OpsWorks中来，但是它无法达到Cloud Formation对于资源管理的广度。 另外，目前的OpsWorks仅仅支持Amazon Linux 和Ubuntu LTS操作系统，你可以使用这些系统的默认AMI或者在这些默认AMI基础上创建的新AMI。
定制化	支持利用参数改变由资源模板定义的Stack默认行为。	支持利用自定义JSON改变Stack的默认行为。
自动部署	支持各种自动化部署工具（	支持基于Chef的自动化部署，并且

	如Chef、Puppet等），但是你需要自己实现所有的自动化部署脚本。	提供了大量的内建自动化脚本。这些内建的自动化脚本会帮助开发和运维人员完成很多的常见工作（如自动部署Tomcat服务器等）并已经集成到整个服务中去。另外，它同样支持自定义的Chef脚本来实现项目相关的部署。
弹性支撑	支持完全自由控制的弹性策略。用户可以使用Auto-Scaling组加自定义的性能指标来确定Scale-up/down的条件。	提供基于负载（Load-Based）和基于时间（Time-Based）的弹性策略。其中基于负载的弹性策略主要考虑CPU、内存等几个主要因素。
系统监控	支持基于Cloud Watch的监控机制。用户可以监控大量的内建指标并添加自定义的监控指标到Cloud Watch中去。	提供以Stack为单元的整合监控界面，同样以Cloud Watch为基础提供监控服务。另外，它还提供了基于Ganglia的可选监控方案。
安全管理	支持IAM的完整功能。用户可以精细控制各个资源的访问权限。	支持IAM的完整功能。并能方便的把相应的安全策略批量实施。

表1：Cloud Formation与OpsWorks的比较

参考上面的比较和我们目前的自动化部署系统，OpsWorks对于我们仍然有一些限制：

- 无法支持Windows操作系统。我们的很多服务仍然是运行在Windows系统上。
- 无法提供自定义的弹性策略。我们的系统实现了自己的调度算法，目前把该调度算法映射到现在OpsWorks中还比较困难。

随着OpsWorks的发展，这些限制未来都可能会被解决。但是，作为一个已经运行的系统，我们仍然需要仔细评估OpsWorks带来的收益和成本以确定是否迁移。如果需要在AWS上面部署一个新的应用，推荐的实践就是如图5从左向右依次选择，并且在确认左面的方案有明确限制的情况下再考虑下一个选择。如果图5中左边服务已经能够解决问题，就不建议自己开发相应的系统。毕竟整个部署系统的开发和维护还是需要一个不小的成本，而AWS提供的这些应用程序管理服务都免费的。另外，在绝大多数情况下，Cloud Formation已经足够灵活以满足你在AWS上部署服务。但是，在某些情况下（例如，同时支持在多个云服务提供商上部署服务），你可能还得选择完全自己开发或采用第三方供应商的方案。

在过去的一年里，我们利用上面的自动化部署系统让整个部署过程的耗时从最初的几天快速下降到几分钟之内，大大降低了的开发、运维人员的负担。如此同时，自动化部署还把部署出错的可能性降到了一个极低的水平，提高了系统的整体可用性。

3. 总结

在上面的文章中，我仔细介绍了整个自动化部署系统，并讨论了怎样充分利用AWS服务的特点来提高自动化部署的效率。该系统运行高效、稳定，而且极大的降低了平台自身和运行于平台上产品的运维成本。接下来，我们仍然会持续改进整个系统，其中关注的重点有：

- 解决各个平台组件之间的兼容问题。我们希望让自动化部署系统能够根据设置的规则自动检测各个组件内的兼容问题并选择合适版本自动化部署，这样，各个组件可以按照自动的节奏（和版本号）独立发布（而不是像现在所有的组件必须以一个版本一块部署）。
- 开发整个自动化部署系统的控制台界面（Dashboard）。该界面可以让我们自己和产品部门看到系统的目前部署状态及部署历史，并且可以让一键部署在该控制台触发，从而让平台内部的构建系统彻底隐藏在背后。
- 尝试和AWS OpsWorks的结合。OpsWorks的一个优势就是集成了大量DevOps实践精髓并提供了丰富的内建部署脚本，可以降低自动化部署系统自身的开发和维护成本。尽管如前所述，目前还没有办法向该系统迁移，但我们仍然会密切关注OpsWorks自身的发展，并会在平衡收益与成本的前提下积极尝试和AWS OpsWorks的结合。

关于作者

徐桂林现为[Autodesk](#)中国研发中心的高级开发工程师。2007年加入Autodesk中国研发中心，先后在Autodesk Labs、CTO Office工作，现在Autodesk云平台事业部（Cloud Platform）工作。在2007年开始参与多个SaaS相关的开发，主要做前端开发。从2009年年底转向做后端，开始使用AWS服务并领导中国团队的开发工作。毕业于浙江大学计算机系CAD&CG国家重点实验室，关注云计算、前端开发等相关技术。

原文链接：<http://www.infoq.com/cn/articles/automated-deployment-practice-based-on-aws>

相关内容

- [NASA是如何使用AWS的](#)
- [媒体热议亚马逊AWS落地中国](#)
- [AWS将于2014年初在国内开启有限邀请服务](#)
- [聪明的客户端，木讷的服务器？AWS发布浏览器中使用的JavaScript SDK](#)
- [AWS增加Redis并对RDS进行若干改进](#)

本期专题：云计算的三个层面 | Topic

Windows Server基础架构云参考架构：硬件之上的设计

综述

作者 王枫

毫无疑问，移动互联网、社交网络、大数据和云计算已经成为IT发展的四个大的趋势。其中云计算又为前三个提供了一个理想的平台。今天不仅互联网公司，很多传统行业的大中型企业也在建设自己的私有云。本文旨在介绍一个基于Windows Server 2012和System Center 2012 SP1构建基础架构云其硬件部分的参考架构。

设计目标

- 从运维角度，整个架构应该易于扩展，从小到4个机柜至大到整个数据中心可以方便的进行扩展和容量规划。
- 从用户的角度，整个架构应该可以兼容不同的应用类型，比如对计算敏感型，大内存型，和IO密集型等不同类型。
- 从服务交付的角度，整个架构应该能够满足不同的服务等级需求，如需要高可用的和无需高可用的。
- 从经济型角度，整个架构应该不依赖特定的硬件厂商或产品。

Windows Server基础架构云的参考架构

Windows Server基础架构云的参考架构如下：

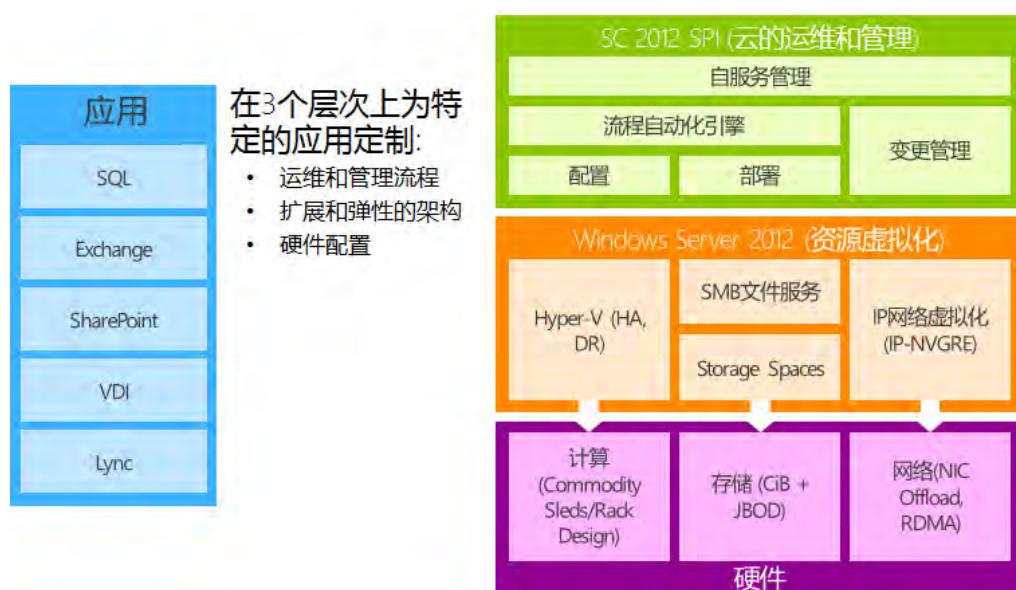


图1 Windows Server基础架构云的参考架构

本文中我们将集中讨论上面参考架构中的硬件设计部分。

为了实现前面拟定的设计目标，我们采用的模型首先要保证在计算和存储之间实现松耦合，只有这样才能保证我们更灵活地调度计算、存储、网络三种基础资源，组建足够体量的资源池按需承载不同类型的应用。基于相同的目的，我们采用了集中的存储而不是使用更低成本的直连存储。虽然直连存储的成本更低，但我们也所采用的存储方案具有更高的可靠性，弹性和灵活性，这些都是企业非常关注的。另外，我们没有采用传统的光纤存储而代以新的完全基于网络的文件共享存储，这样的好处是相对于光纤存储，硬件采购成本较低，而且降低系统复杂度以及维护成本，但这样设计要求网络的部分必须具备足够的可靠性和高性能来满足IO密集型的需求。

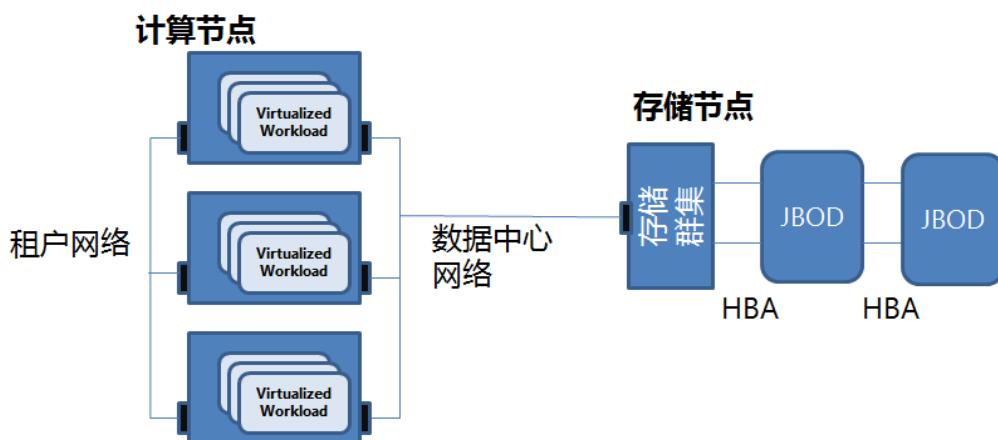


图2 Windows Server基础架构云的硬件架构模型

标准IaaS SKU

企业内部系统传统上主要通过冗余来实现高可用，例如对于一台服务器，所有的组件能冗余的都要冗余（比如内存、硬盘、网卡、电源），有的系统甚至实现了主板、CPU的冗余，但这样一来就意味着要投入昂贵的硬件。与此不同的是，市场上主流的公有云服务商为了给大部分客户提供低成本的服务，将高可用的责任很多时候交给用户来处理，比如要求用户的应用需要具备弹性能力（resilience），将相同角色的多个实例部署在不同的Fault Domain/Update Domain中来提高整个服务的高可用性。在本设计中我们借鉴了公有云的这一经验，不要求采用如此高度冗余的设备，相反我们的故障单元是机柜而不是里面某台服务器或存储，也就是说我们关注的是跨机柜的可靠性，包括了基础架构、平台、应用和数据。但同时考虑到对于私有云，更多的时候面临的是将企业现有的应用迁移到云上，这些应用很可能不能很好的处理stick session等问题，很难通过横向扩展多个无状态的实例来实现高可用，故而私有云IaaS在设计时还是应该考虑到为传统应用，甚至为传统上很难实现高可用的应用提供基础架构层级的高可用性服务。

。在这个设计中对于没有高可用性需求的用户，也可以在下面的设计中增加非群集的Hyper-V服务器和存储服务器来提供不同服务等级的服务。

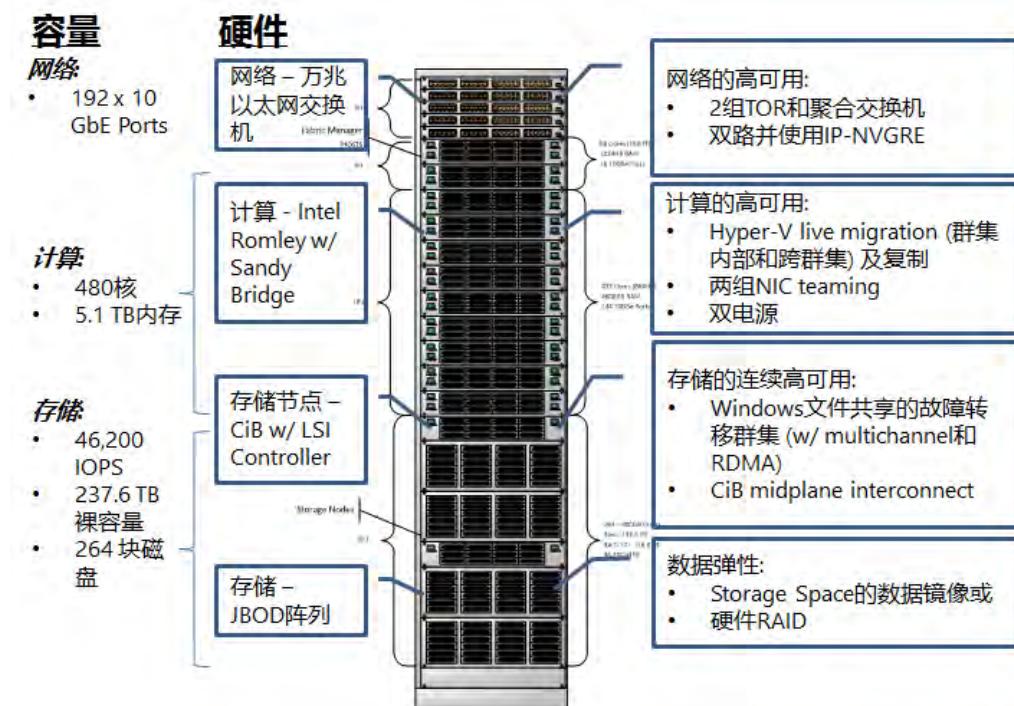


图3 标准IaaS SKU的构成

计算节点的设计

对于计算节点我们设计了两组网络，所有的虚拟机产生的网络访问流量都走租户网络上的虚拟网络，而虚拟机产生的存储访问流量都重定向到物理机操作系统通过数据中心网络使用具备SMB Direct的SMB3协议直接同文件服务器群集通讯。

。

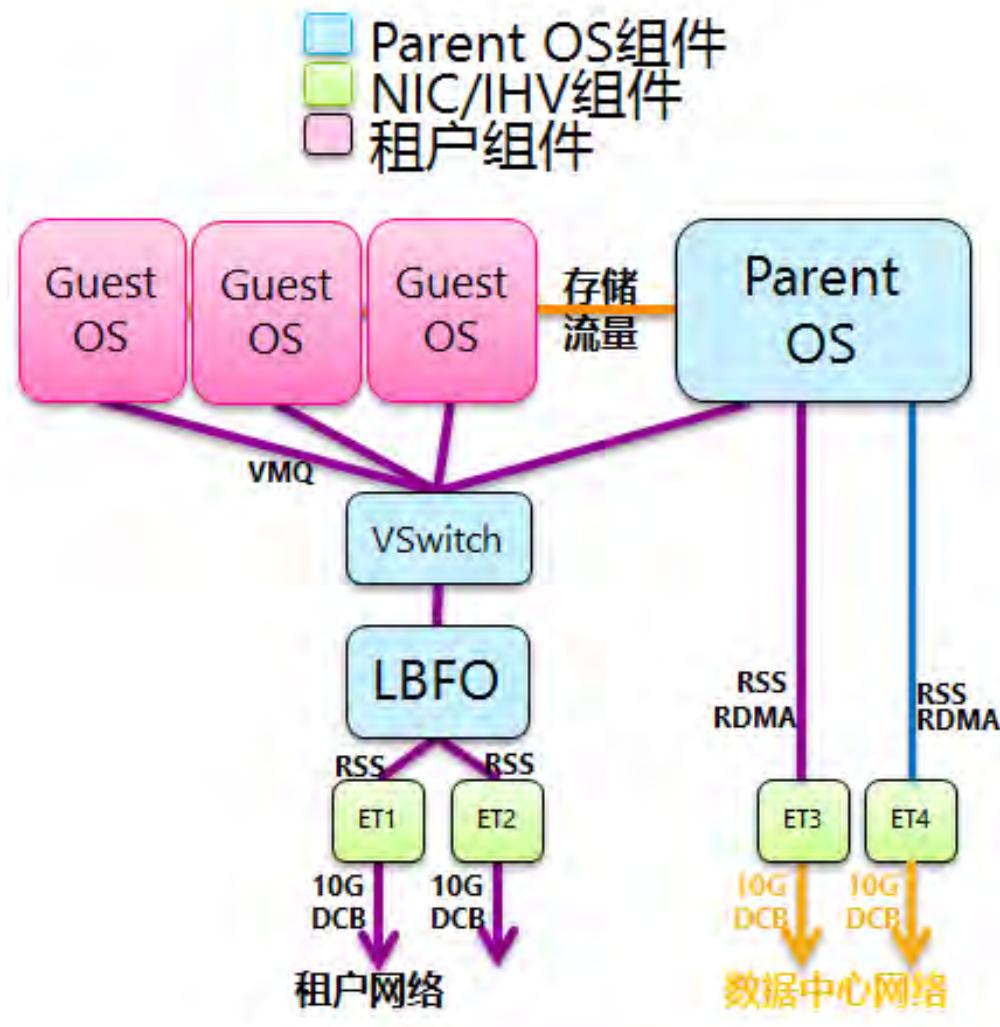


图4 计算节点的实现

租户网络和数据中心网络在计算节点的用途小结如下：

租户网络用于：

- 租户到租户的通讯
- 租户到外部的通讯

数据中心网络用于：

- 到存储节点的通讯（也就是到文件服务器，使用RDMA网卡）
- 虚拟机的实时迁移
- 虚拟机存储的实时迁移
- 群集心跳线

下面是一个标准的计算节点的硬件配置：

项目	标准配置
处理器	Intel Sandy Bridge CPU, 2 Socket x 6 cores (ES2640, Core frequency 2.5 GHz) = 12 cores

内存	16 DIMM x 8 GB = 128 GB
存储	内置 200 GB SSD (eMLC)
网络	2 x 10 GbE onboard (虚拟机所用的网络) 2 x 10 GbE mezzanine with RDMA (用于访问存储节点、管理和实时迁移)

表1 计算节点的标准配置

存储节点的设计

本设计中采用的存储架构是基于Windows Server 2012的故障转移群集。以两节点来实现连续高可用 (Continuous Availability)。为了提高性能，减少对于主机CPU的压力，采用了RDMA网卡，借助Windows Server 2012的SMB3文件服务器为计算节点提供Hyper-V虚拟机的存储。

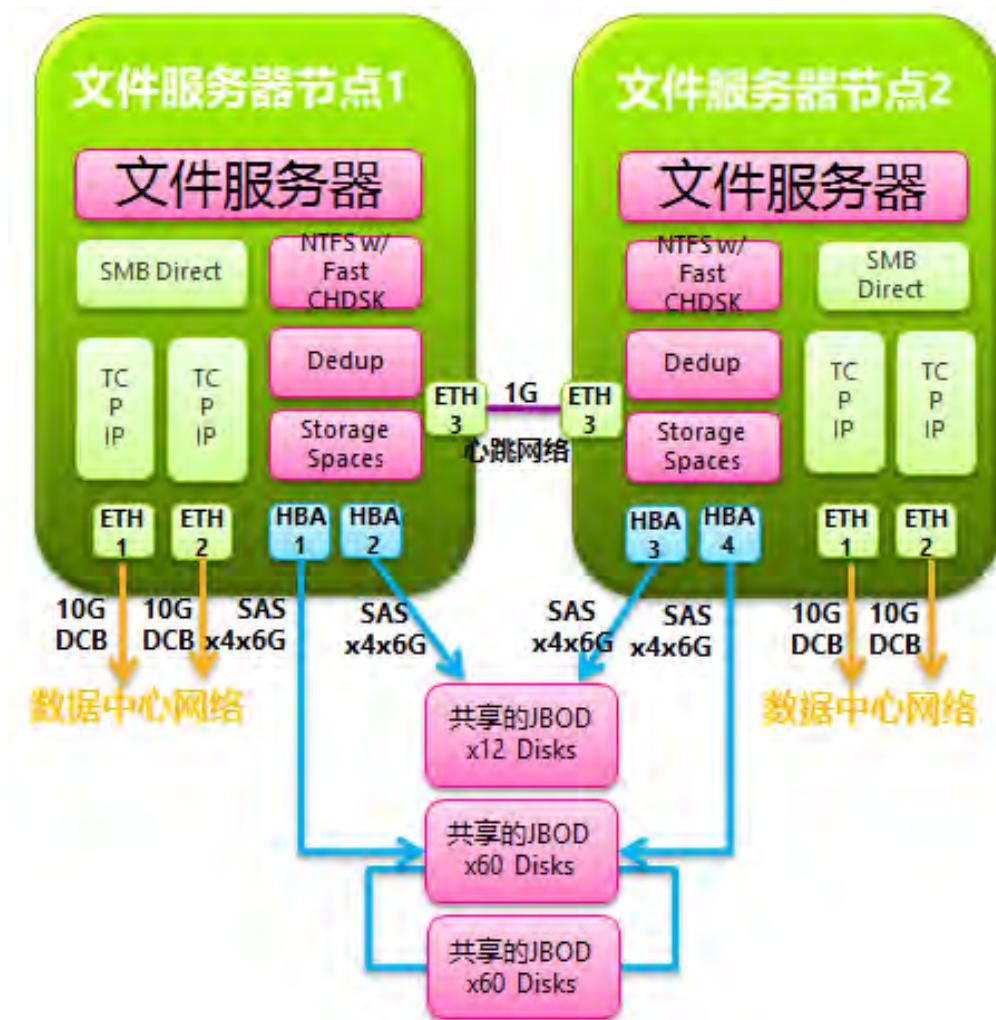


图5 存储节点的实现

存储节点通过SAS HBA卡连接了共享的SAS接口硬盘柜，每个节点到磁盘柜额带宽高达48Gb/s (2x4x6Gb/s)。

这里的数据中心网络用于：

- 到计算节点的通讯
- Storage Space重定向流量
- 备份和复制流量

下面是一个标准的计算节点的硬件配置：

项目	标准配置
处理器	Intel Sandy Bridge CPU, 2 Socket x 6 cores (Core frequency 2.5 GHz) = 12 cores
内存	16 x 8 GB = 128 GB
存储	2 x 10 GbE mezzanine with RDMA
网络	132 x 2.5" 10K SAS 900 GB Drives 2 x JB9

表2 存储节点的标准配置

下图展示了计算节点和存储节点的连接方式

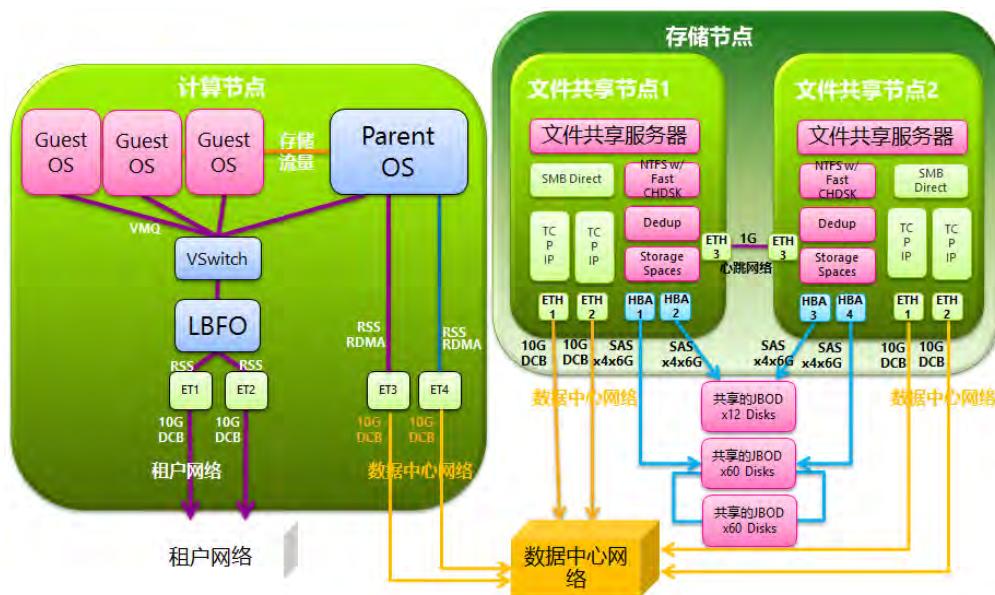


图6 计算节点和存储节点的连接

网络的冗余设计

本设计中使用的网络三层都有冗余，他们分别是：

- 每个数据中心2个汇聚层交换机
- 每个机柜2个租户网络交换机
- 每个机柜2个数据中心网络交换机

网络Trunk设置：

- 4 x 10 GbE 租户网络接口聚合
- 4 x 10 GbE 数据中心网络接口聚合 -> Aggregate
- 16 x 10 GbE 聚合到核心交换

下面是示意图：

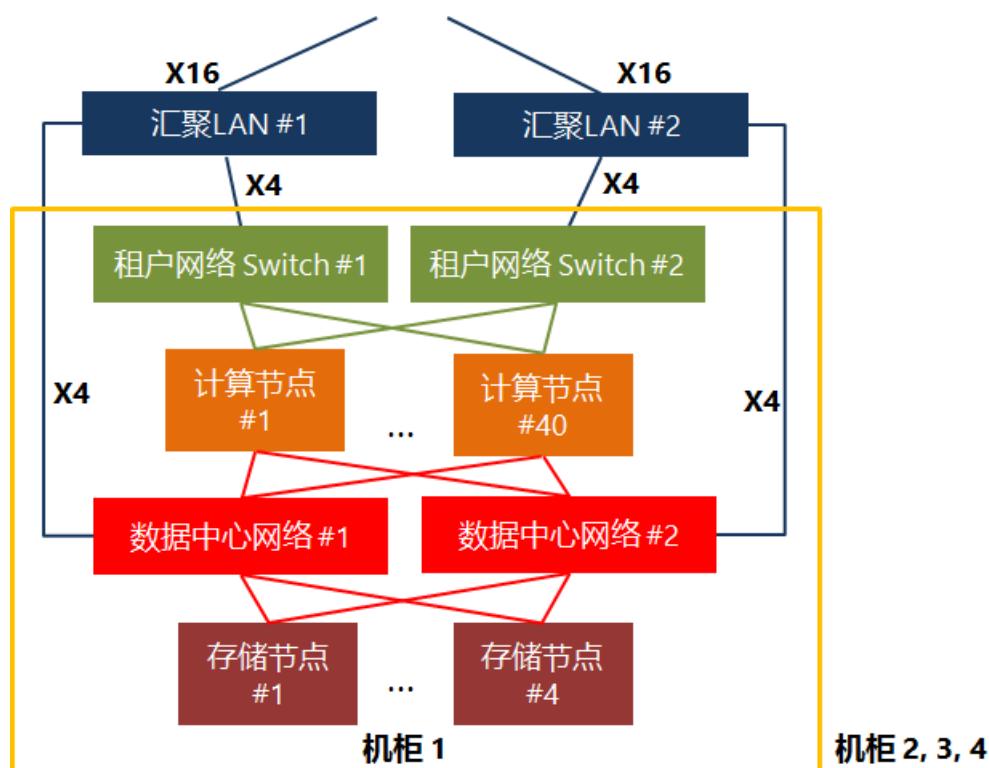


图7 网络的冗余设计

小结

企业在设计自己的基础架构云时可以考虑采用或者部分采用上面的设计，事实上微软自己的很多产品已经采用了上面的设计，比如微软SQL Server 2012的并行数据仓库（PDW）就采用了类似的架构。微软自己的模块化数据中心也是采用一样的架构。

由于篇幅所限，我们讨论硬件之上的设计，正如我们一开始所说的，我们的目标之一就是用应用的弹性取代对于硬件冗余的需求，当我们把一个机柜最为一个故障单元的时候，如何跨机柜、甚至跨数据中心来调度、部署应用就显得非常重要。这个部分我们将在以后进行介绍。

原文链接：<http://www.infoq.com/cn/articles/windows-server-infrastructure-cloud-reference-architecture>

相关内容

- [Microsoft通过Service Bus for Windows把云整合服务搬到本地](#)
- [便于运维的Windows服务](#)
- [微软发布Kinect for Windows SDK v1.7](#)
- [如何保持最新的Windows安全指南](#)
- [Amazon Web Services发布AWS Tools For Windows PowerShell](#)

QClub

我们影响有影响力的人

非盈利、非商业、纯技术交流的技术社区活动

We are QClub

QClub作为InfoQ线下技术沙龙品牌，定期在全国主要城市举办免费的技术沙龙，邀请国内外知名公司技术总监，项目经理，高级研发工程师等走入各地技术社区，分享他们的经验与对行业趋势的预测与讨论，为中国技术人员搭建交流、分享的平台，尽自己微薄之力架起中高端技术人员之间的桥梁，为中国技术社区的发展与价值的传播贡献自己的力量。

非盈利 非商业 纯技术交流

QClub的话题

QClub的话题可能包括任何当地技术人员感兴趣的话题，比如：编程语言、架构设计、企业级开发、运维、基础架构、过程与实践、云计算、大数据、互联网、移动开发、安全、敏捷、性能、业务流程、SOA、.....



参会人群：

开发人员、技术或
团队负责人、技术
爱好者、学生等

活动城市：

北京、长沙、成都、大连、福
州、广州、上海、太原、天津、
温州、西安、郑州、.....

举办周期：

每个城市
每1-2个月一次活动

活动形式多样：

- 讲师演讲 ◦ 线上交流
- Open Space ◦ 粉丝 QQ 群
- 户外活动 ◦ 等等.....

推荐文章 | Article

可视化Java垃圾回收

作者 [Ben Evans](#) , 译者 [马德奎](#)

垃圾回收，就像双陆棋一样，只需几分钟来学习，但要用一生来精通。

Ben Evans是一名资深培训师兼顾问，他在演讲[可视化垃圾回收](#)中从基础谈起讨论了垃圾回收。

以下是对其演讲的简短总结。

基础

当谈到释放不再使用的内存，垃圾回收已经在很大程度上取代了早期技术，比如手动内存管理和引用计数。

这是件好事，因为内存管理令人厌烦，学究式地簿记是计算机擅长的，而不是人擅长的。在这方面，语言的运行时环境比人强。

现代的垃圾回收非常高效，远远超过早期语言中典型的手工分配。通常，具有其它语言背景的人只盯着垃圾回收造成的中断，却没有完全理解自动内存管理发生作用的上下文环境。

标记&清除是Java（及其它运行时环境）用于垃圾回收的基本算法。

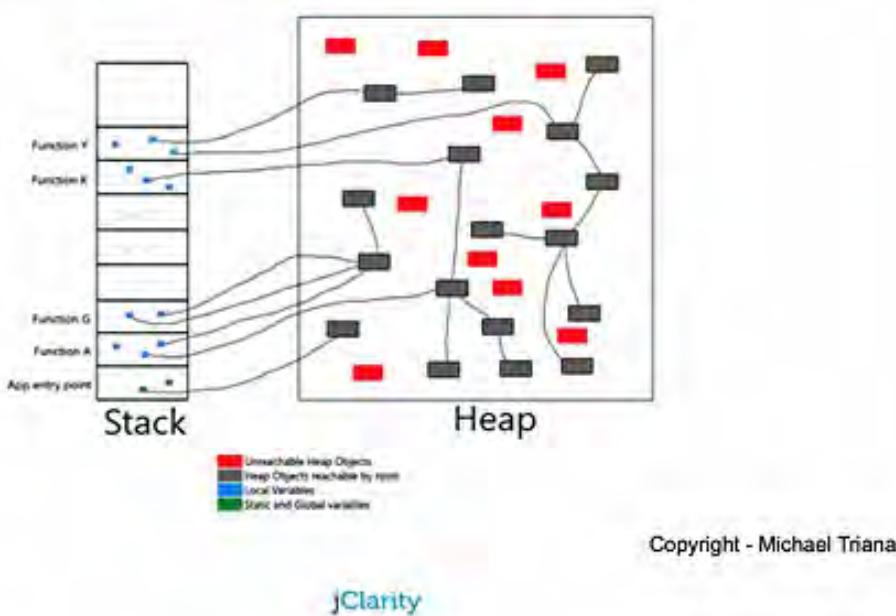
在标记&清除算法中，引用会从每个线程栈的桢指向程序的堆。所以，从栈开始，循着指针找到所有可能的引用，然后再循着这些引用递归下去。

当递归完成，就找到了所有的活对象，其它的都是垃圾。

请注意，人们经常漏掉的一点是，运行时环境本身也有一个“分配清单（allocation list）”，上面列出了指向每个对象的指针，该列表由垃圾回收器负责维护，并帮助垃圾回收器进行垃圾清理。因此，运行时环境总是可以找出由它创建但尚未回收的对象。

- **GC Roots**

– A pointer to data in the heap that you need to keep



图一

上面插图中所示的栈只是一个与单个应用程序线程相关的栈；每个应用程序线程都有一个类似的栈，每个栈本身都有一组指向堆的指针。

如果垃圾回收器试图在应用程序运行过程中获取活对象的快照，那么它就要追踪运动着的目标，那样很容易漏掉一些严重超时的对象分配，因而无法获得一个准确的快照。因此，“Stop the World”是有必要的；也就是，停止应用程序线程足够长的时间，以便捕获活对象的快照。

下面是垃圾回收器必须遵循的两条黄金法则：

1. 垃圾回收器必须回收所有的垃圾。
2. 垃圾回收器必须从不回收任何活对象。

但这两条规则并不是对等的；如果违反了第二条规则，结果会使数据遭到破坏。

另一方面，如果违反了第一条规则，则会是另一种情况，系统并不总是能够回收所有的垃圾，但最终会回收所有的垃圾，那么这是可以接受的，而实际上，这是垃圾回收器的基本原理。

HotSpot

现在，我们来说下HotSpot，它实际上是一个C、C++以及许多特定于平台的汇编程序组成的混合体。

当人们想到解释器，就会想到一个很大的while循环，其中包含一个很长的switch语句。但HotSpot解释器比那个要复杂的多（由于性能原因）。在开始阅读JDK源代码的时候，就会发现HotSpot中实在是有许多汇编程序代码。

对象创建

Java会预先分配大量的连续空间，就是我们所说的“堆”。之后，HotSpot完全在用户空间里管理这块内存。

如果一个Java进程占用了大量的系统（或内核）时间，那么毫无疑问，它不是在进行垃圾回收——因为所有的垃圾回收内存“簿记（bookkeeping）”都是在用户空间进行的。

内存池

- **Young Generation Pools**
 - Eden
 - Survivor 0
 - Survivor 1
- **Old Generation Pool (aka Tenured)**
 - Typically much larger than young gen pools combined
- **PermGen Pool**
 - Held separately to the rest of the Heap
 - Was intended to hold objects that last a JVM lifetime
 - Reloading and recycling of classes occurs here.
 - Going away in Java 8
 - May lead to new problems

图二

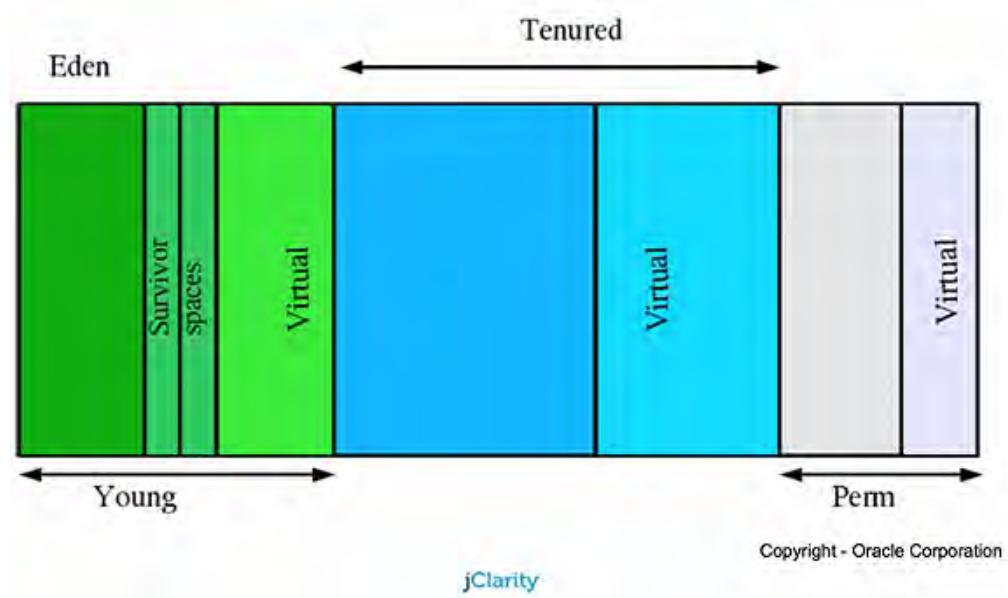
“永久代（PermGen）”是一个存储区域，用于保存那些需要在程序生存期内一直存活的东西，如类的元数据。不过，随着应用程序服务器的出现，它们有自己的类加载器，并且需要重新加载类的元数据，永久代作为一个优化决策开始显得糟糕，所幸，它在Java 8中消失了。

Java 8将会使用一个名为“元空间（Metaspace）”的新概念。元空间与永久代并不完全相同。它在堆的外面，由操作系统管理。这意味着，它不会在Java堆中，而是在本地内存里。目前，这还是一个非常好的消息，因为没有多少工具能够让用户轻松地查看本地内存。所以，永久代消失是件好事，但工具赶上这个变

化还需要一些时间。

Java堆布局

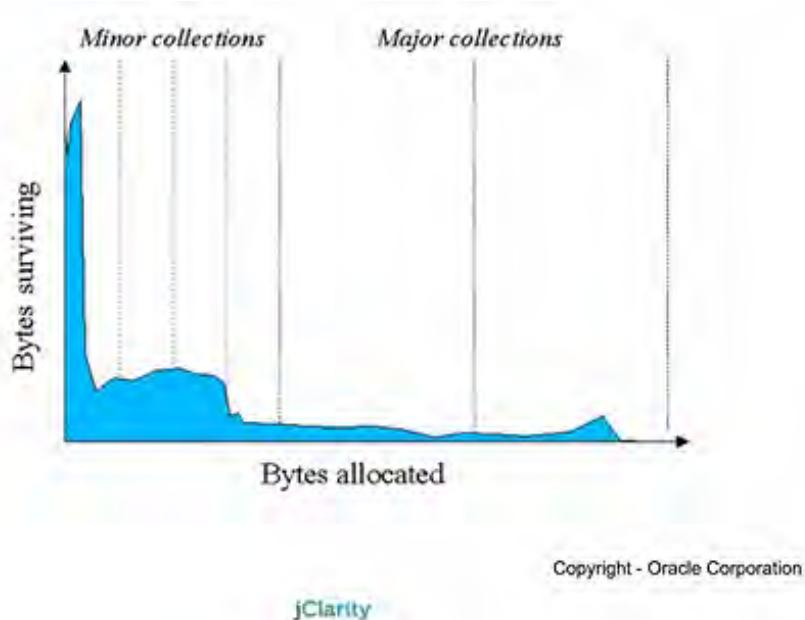
现在，我们来看下Java堆。注意堆空间之间的虚拟空间。它们提供了一点浮动量，以允许对内存池进行一定量的尺寸调整，又不用为任何对象移动付出代价。



图三

“弱代假设 (Weak Generational Hypothesis) ”

就现状而言，究竟为什么要将堆分成所有这些内存池？



图四

有的运行时事实无法通过静态分析推导出来。上面的插图说明有两组对象：一组存活时间短，一组存活时间长——所以，做额外的笔记以便利用这一事实是有意义的。在Java平台中，有许多类似的作用于优化写入平台的事实。

演示

Ben Evans进行了一系列的动画[演示](#)。第一个演示是个Flash，说明了对象在Eden区和一个新生代Survivor空间之间移动，并最终进入老年代的过程。

图五是用JavaFX再现了同样的过程。



图五

运行时开关

‘强制性’参数

- `-verbose: gc`——为用户输出一些GC信息
- `-Xloggc:<文件路径>`——指定日志输出路径，要确保磁盘有空间
- `-XX: +PrintGCDetails`——为辅助工具提供“最低限度信息 (Minimum information) ”

——用这个参数代替`-verbose: gc`

- `--XX: PrintTenuringDistribution`——“过早提升 (Premature promotion) ”信息

基本堆大小参数

- **-Xms<size>** —— 设置预留给堆的最小内存值
- **-Xmx<size>** —— 设置预留给堆的最大内存值
- **-XX:MaxPermSize=<size>** —— 设置永久代的最大内存值

——有利于Spring应用程序和应用服务器

以前，我们被教导要把-Xms和-Xmx的值设的一样大。不过这已经变了。因此，现在可以为-Xms设置一个合理范围内较小的值，或者根本就不设置，因为堆的适应能力现在已经非常好了。

其它参数

- **-XX:NewRatio=N**
- **-XX:NewSize=N**
- **-XX:MaxNewSize=N**
- **-XX:MaxHeapFreeRatio**
- **-XX:MinHeapFreeRatio**
- **-XX:SurvivorRatio=N**
- **-XX:MaxTenuringThreshold=N**



图六

为什么要有日志文件

日志文件的好处是能够用于取证分析，可以使用户免于为了再现问题而不得不再次

执行一次代码（如果是一个罕见的生产环境错误，那么重现并不容易）。

另外，它们包含的信息比针对内存的JMX MXBeans所能提供的信息更多，且不说轮询JMX本身会引入一系列GC问题。

工具

- **HP JMeter** (用Google查询一下)

——免费，非常可靠，但不再提供支持/功能增强

- [**GCViewer**](#)

——免费，开源，但界面有点丑

- [**GarbageCat**](#)

——名字最好听

- [**IBM GCMV**](#)

——支持J9

- [**jClarity Censum**](#)

——界面最美观，而且最有用——不过，这是我们的偏见！

小结

- 需要了解一些**GC基础理论**
- 要让新生代的大部分对象在年轻时死亡
- 打开**GC日志**！——原始日志文件难以阅读——使用工具
- 使用工具来帮助自己调优——测量，而不是猜测

查看完整演讲视频，请点击[这里](#)。

关于作者



Ben Evans是一家Java/JVM性能分析创业公司jClarity的CEO。在业余时间，他是伦敦Java社区的一名负责人，也是Java社区过程执行委员会成员之一。他先前的项目包括：对Google IPO、金融交易系统做性能测试，为若干90年代最大的电影开发获奖网站等等。

原文链接：<http://www.infoq.com/cn/articles/Visualizing-Java-Gar>

bage-Collection

相关内容

- [从Ruby向Java的迁移帮助Twitter挺过了美国大选](#)
- [Java EE企业系统性能问题的原因和解决建议](#)
- [全速前进：Oracle计划于3月份发布Java 8，即便有Bug亦如此](#)
- [Java 8将在三月份发布](#)
- [实现Java中的高性能解析器](#)

推荐文章 | Article

Backbone与Angular的比较

作者 [Victor Savkin](#)，译者 邵思华

将不同的思想和工具进行对比，是一种更好地理解它们的方式。在本文中，我首先将列举在创建web应用程序时需要重复进行的各项任务，随后为你展现Backbone和Angular将如何帮助你完成这些工作。

我们所尝试解决的问题

作为web开发者来说，我们的大部分工作都可以归结于以下的某个类别中：

- 实现业务逻辑
- 构建DOM
- 实现视图逻辑（声明式与命令式）
- 在模型与视图间进行同步
- 管理复杂的UI交互操作
- 管理状态和路由（routing）
- 创建与连接组件

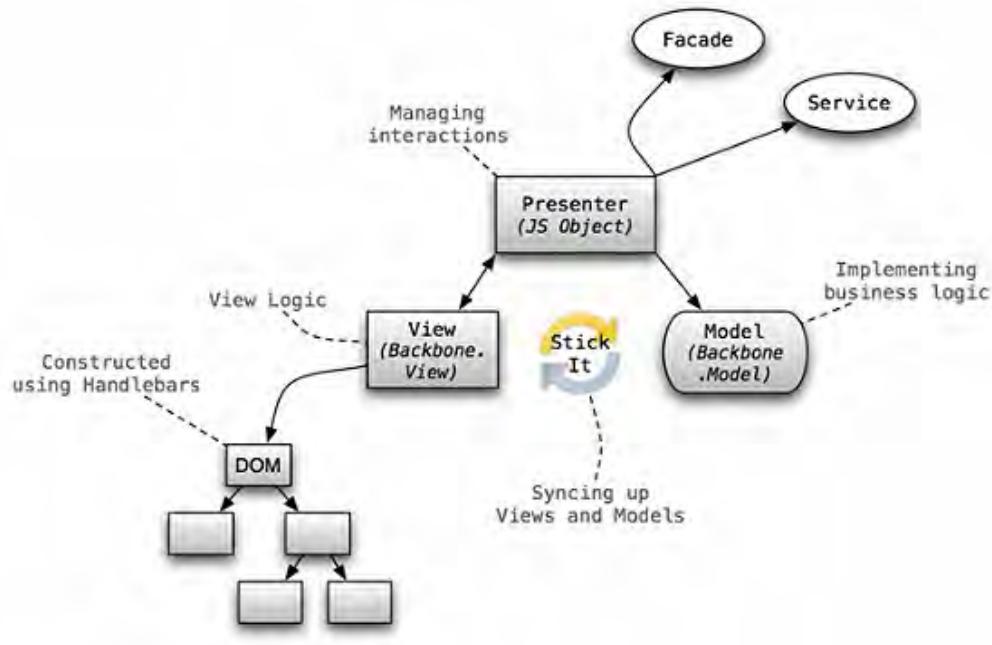
如你所料，大多数客户端框架都以某种方式帮助你完成这些工作。

Backbone

首先让我们看看Backbone为解决这些问题提供了哪些功能：

业务逻辑	Backbone模型（Model）和集合（Collection）
构建DOM	Handlebars
声明式视图逻辑	Backbone视图（View）
命令式视图逻辑	Backbone视图
视图与模型同步	StickIt
管理UI交互	JS对象或 Marionette Controllers
管理状态与路由	Backbone.Router
创建与连接组件	手工实现

以下图片以更直观的方式表现了这些功能：



我所指的Backbone.....

将最原始的Backbone与Angular直接进行对比有些不太公平，因此在本文中所指的Backbone实际上是Backbone + Marionette + 插件的这套组合。

业务逻辑

Backbone应用程序中的很大一部分业务逻辑由模型（model）和集合（collection）负责实现，这些对象往往对应着服务端后台的资源，它们将包含视图显示所必须的内容。

由于使用者必须扩展Backbone.Model和Backbone.Collection对象，因此造成了许多额外的复杂性。

首先，Backbone使用POJO（简单JavaScript对象）和Backbone model对象两种方式表现领域对象。在显示模板（template）、或是与服务端交互时需要使用POJO，而在需要使用可观察（observable）属性时（例如需要建立数据绑定的时候），则需要使用Backbone模型。

其次，Backbone推荐你使用不可变对象。由于Backbone不支持对函数进行观察，因此每次有某个属性发生改变时，与之相对应的推断（computed）属性必须被重置。这就为你的应用增加了许多额外的复杂性，也使得最终产生的代码难以理解和测试。除此之外，所有的依赖项必须以（"change:sourceProperty, this.calculateComputedProperty）的形式显式地进行指定。

构建DOM

Backbone使用模板引擎构建DOM。虽然从理论上说，你可以选择你所中意的引擎，但基本上在大型应用程序中都会在Mustache和Handlebars这两者之间进行选择。Backbone中的模板定义通常不包含逻辑，并且多数是基于字符串的，不过这两点也并非必需。

视图逻辑

将视图逻辑划分为命令式与声明式逻辑是一种由来已久的方式（可以追溯到原始的MVC模式）。事件处理配置和数据绑定属于声明式，而事件处理本身则是属于命令式。不过Backbone并没有为这两者划分出一条清晰的界限，它们都由Backbone.View对象处理。

在模型与视图间进行同步

由于Backbone在本质上追求最简化，因此它本身并没有为数据绑定提供支持。这一点对于小型项目来说或许不是一个问题，毕竟你可以选择让视图负责对模型和DOM进行同步。但当应用程序的功能开始不断增加时，这种方式就很容易渐渐失控。

好在如今已经有各种各样的插件（例如Backbone.StickIt）能够帮助Backbone解决这一问题了，因此你可以不用再理会琐碎的模型-视图同步操作，而是专注于复杂的交互工作。这些插件中的大多数都可以使用简单的JavaScript进行配置，因此使用者可以在它的基础之上创建一个抽象层，以满足你应用程序的需求。

在Backbone中使用数据绑定的一个缺点就是它们依赖于可观察属性，而另一方面，模板引擎又是使用POJO实现的。由于同时存在着两种与DOM交互的方式，经常会造成代码的重复。

管理复杂的UI交互操作

所有的UI交互操作都可以划分为简单交互操作（使用[观察者同步](#)（Observer Synchronization）方式进行管理）和复杂交互操作（必须使用[流同步](#)（Flow Synchronization）方式）两种类别。

如前文所述，简单交互操作是通过使用数据绑定和事件处理函数的Backbone.View进行处理的。由于Backbone本身没有硬性规定处理复杂UI交互的解决方案，你可以随意选择最适合你的应用的方式。有些人选择使用Backbone.View作为解决方案，但我建议你不要这么做，因为这种方式会造成Backbone.View的职责过多。我会倾向于使用[主动控制显示](#)（Supervising Presenter）模式管理复杂的交互操作。

管理状态和路由

Backbone包含了一个非常简单的路由器的实现，但它并不支持管理视图和应用状态的功能，必须要手动实现这些功能。因此在实际应用中经常会选择使用其它类库（例如router.js），而不是它自带的路由器。

创建与连接组件

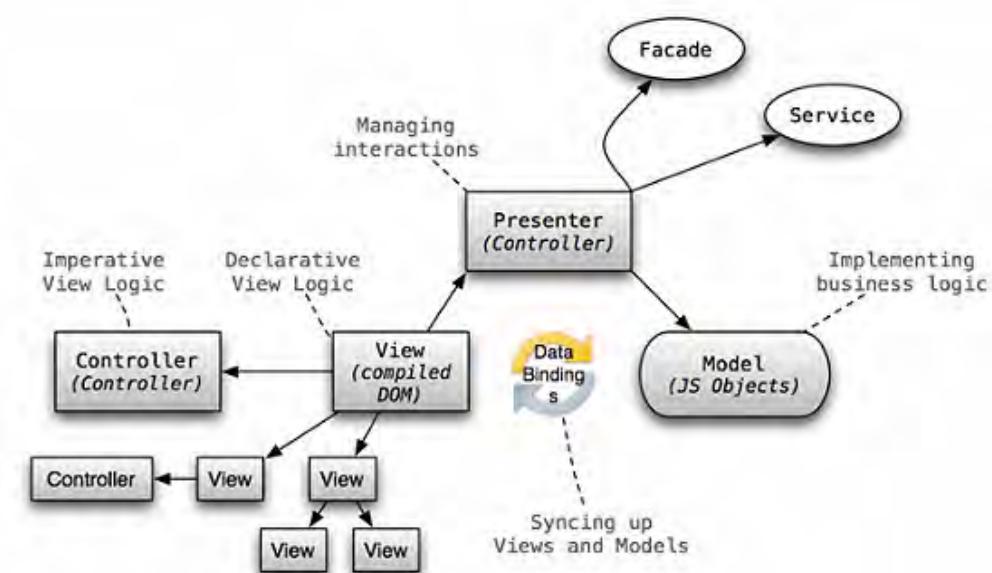
在Backbone中，你可以自由选择最适合你的应用的方式创建并连接组件。这种方式的缺陷在于你必须编写大量的样板代码，而且为了保持代码的合理组织，必须始终遵循良好的代码规范。

Angular

现在让我们来比较一下，看看Angular是如何解决这些问题的。

业务逻辑	JS对象
构建DOM	指令 (Directive)
声明式视图逻辑	指令
命令式视图逻辑	控制器 (Controller)
视图与模型同步	内置的机制
管理UI交互	控制器
管理状态与路由	AngularUI Router
创建与连接组件	依赖注入

以下图片以更直观的方式表现了这些功能：



业务逻辑

由于Angular没有使用可观察属性，因此在实现模型时没有这方面的限制。你不需要扩展某个类、或者遵循某个接口，而是可以自由地选择你喜欢的方式（包括使用现有的Backbone模型）。在实际开发中，多数开发者选择使用简单的JavaScript对象（POJO），这种方式有以下优点：

- 所有的领域对象都不依赖于任何特定的框架，因而更容易在不同的应用中重用。
- POJO对象和与服务端交互的对象非常近似，因而简化了客户端与服务器的通信。
- POJO对象将用于视图表示，因此无需实现toJSON方法。
- 推断属性可以用函数的形式表现

模板与视图

Angular中的模板被编译之前只是一些DOM片断，而在编译过程中，Angular会将这棵DOM子树进行转换，并为其添加一些JavaScript。编译的结果会生成另一棵DOM子树，也就是视图。换句话说，视图并非由你自己所创建，而是由Angular对模板编译后所生成的。

构建DOM

Backbone将DOM的构建与视图逻辑进行了清晰地分离，前者使用模板引擎实现，而后者则使用数据绑定与命令式的DOM更新操作实现。与之相反，Angular并未将这两者进行区分，它使用相同的机制和指令（directive）构建DOM，并定义声明式的视图行为。

视图逻辑

Angular对声明式与命令式的视图逻辑进行了清晰的划分，前者由视图处理，而后者由控制器负责。

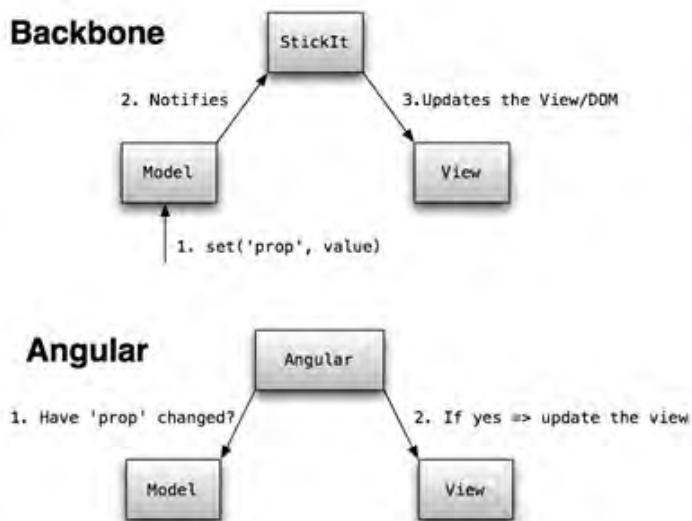
这种划分看起来似乎有些刻意，但它确实是非常重要的。

首先，这种方式清晰地指出了哪些部分需要进行单元测试。嵌入在模板中的声明式逻辑（例如ng-repeat）无需进行单元测试，反之，为控制器编写单元测试通常是个好主意。

其次，所有的依赖都是单向的，即视图依赖于控制器，因此控制器并不了解视图或DOM的任何逻辑。这种方式促进了代码重用，也简化了单元测试。与之相反，Backbone.View经常需要对DOM节点进行操作，随后使用模板引擎对页面中的很大一部分进行重新渲染。

在模型与视图间进行同步

Angular包含了原生的数据绑定功能，与大厦多数其它客户端框架所不同的是，它并不依赖于可观察属性，而是使用了脏检查（dirty checking）方式。



Angular的脏检查方式有着以下一些优点：

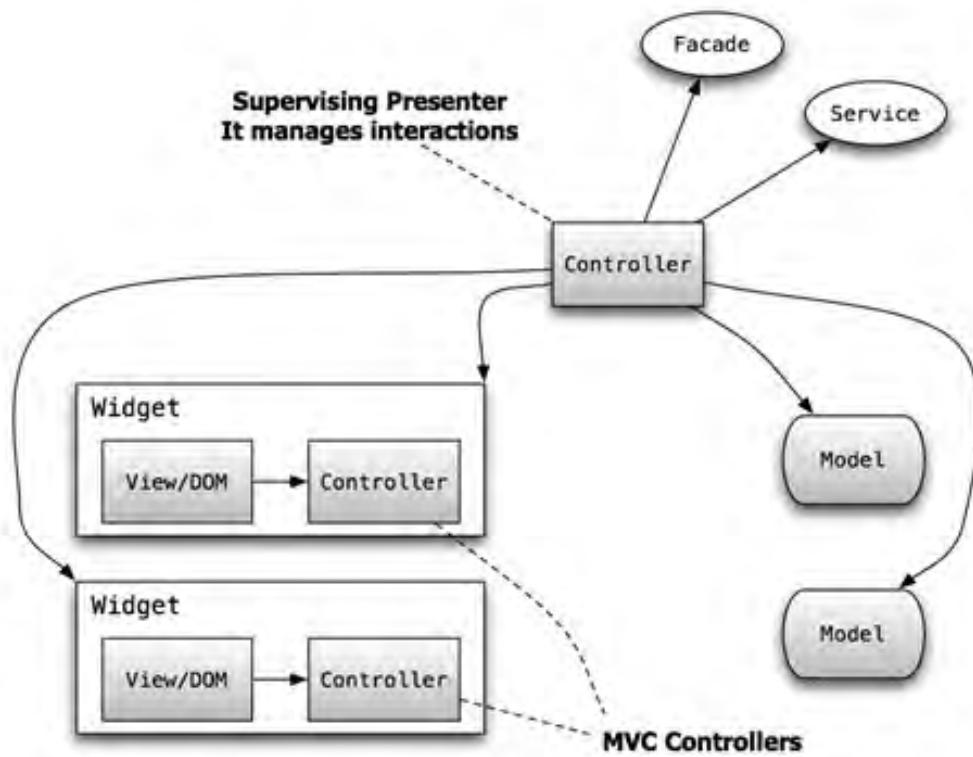
- 模型本身不会意识到它已经成为一个被观察的对象。
- 无需在可观察属性之间指定任何依赖。
- 函数同样表现可观察对象。

但这种方式也带来了以下一些负面影响：

- 在与第三方组件或类库集成的时候，你必须保证Angular能够响应它们对你的模型的任何改变。
- 在某些情况下会带来性能方面的影响。

管理复杂的UI交互操作

如前文所述，控制器将负责实现UI元素的命令式逻辑。除此之外，还可以将控制器实现为一种[主动控制显示模式](#)，以协调复杂的UI交互。



管理状态和路由

与Backbone类似，Angular自带的路由器功能非常基础，并不适合于创建实际的应用。令人欣慰的是，AngularUI Router项目填补了这一空白。它能够管理应用状态、视图，并且支持嵌套视图。换句话说，它能够满足你对路由器的全部功能需求。当然，和Backbone的情况一样，你并非只有这一种选择，你也可以选择其它的路由功能类库（例如router.js）。

创建与连接组件

Angular包含了一个IoC容器，它与通常意义上的依赖注入方法非常相似，这就要求你必须编写模块化的代码。这种方式能够改善代码的可重用性和可测试性，也使你免于编写大量的样板代码。它的负面影响在于一方面增加了使用的复杂度，一方面削弱了对组件创建过程的可控程度。

总结

本文简单地介绍了Backbone和Angular如何处理我们在创建web应用时所遇到的各种问题。这两个框架在某些问题的处理上使用了截然不同的方案，Backbone在显示模板、创建数据绑定和连接组件方面给使用者更多的选择。与之相反，Angular为这些问题提供了规定的方案，不过在创建模型与控制器方面的限制就比较少一些。

关于作者



Victor Savkin是一位就职于Nulogy的软件工程师。他所感兴趣的技术包括函数式编程、web平台和领域驱动设计。他拥有使用JavaScript编写大型应用程序的经验。作为一名编程语言的狂热爱好者，他投入了大量的时间学习Smalltalk、JS、Dart、Scala、Haskell、Closure和Ioke等语言。他的博客victorsavkin.com中有大量关于使用Ruby和JS创建大型应用程序的文章，你也可以关注他的Twitter [@victorsavkin](https://twitter.com/victorsavkin)。

查看英文原文：[Contrasting Backbone and Angular](#)

原文链接：<http://www.infoq.com/cn/articles/backbone-vs-angular>

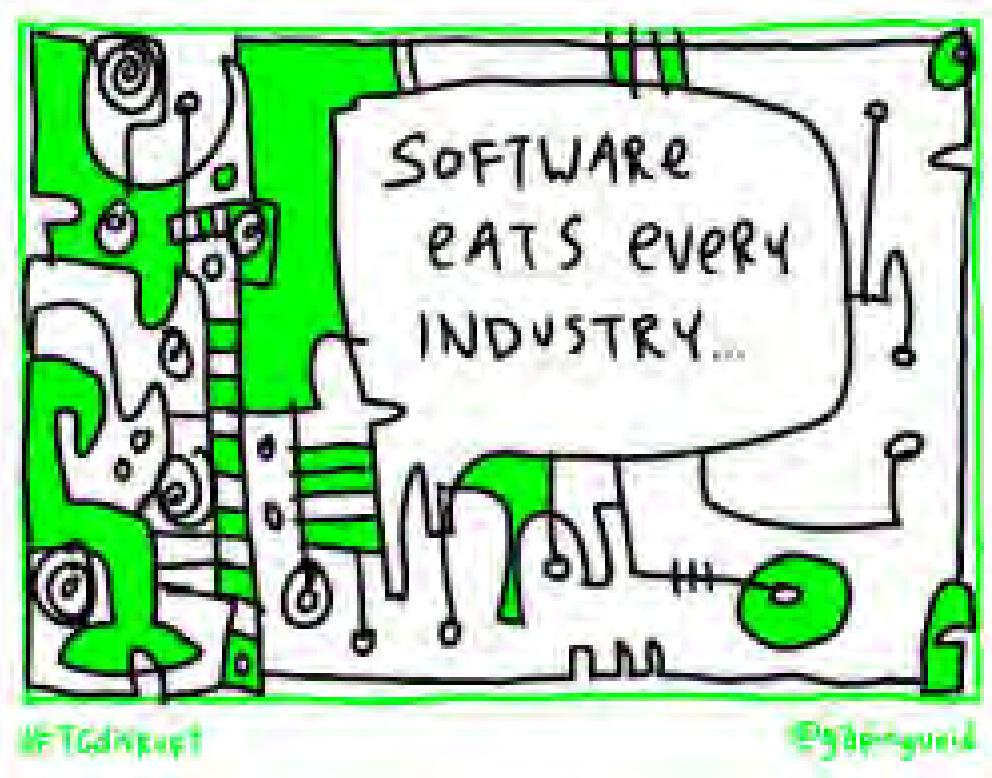
相关内容

- [Backbone与Angular的比较](#)
- [PayPal从Java切换到JavaScript](#)
- [Google Dart新进展：Polymer代替Web UI](#)
- [OSChina的技术选型：就是为了短平快](#)
- [Juergen Fesslmeier谈端到端的JavaScript开发](#)

推荐语

软件与敏捷

Marc Andreessen曾说过，“软件正在吞噬这个世界”，而在数据中心领域，我们也正在目睹这一潮流。软件正在接管那些过去一直由硬件实现的功能；而对于人们如何共事，这一变迁正在引发戏剧性的改变。



特别专栏 | Column

Scrum Master：职位还是角色？

作者 [Brian Lawrence](#)，译者 徐涵

几个月前，我在一次本地敏捷用户组聚会上，加入了关于Scrum Master的同行交流。我在会上的发言造成了一些小骚动，我问的第一个问题是“各位所在公司采用全职Scrum Master的有多少？”出席的公司代表们表示，他们都有全职Scrum Master。于是我接着问了第二个问题“他们做些什么工作？”也许我的提问方式已经暗示了雇佣全职Scrum Master比较奇怪，因为当时我在现场经历了几次鸦雀无声的片刻。除了我的同事，其他大部分人都奇怪地看着我，好像我有三只脑袋一样。最终，他们还是迁就了显然没有好好学习的我，用《Scrum指南》里关于Scrum Master的描述来回答我。当我说我们已经用不着全职Scrum Master时，他们都震惊了。其实我还没说，我们从来就没有用过全职Scrum Master。

《Scrum指南》对Scrum Master的角色做了一般性的描述：负责指导、训练、指引和消除障碍。对刚出道的Scrum团队，这些事情要耗费一些时间。还不熟悉Scrum的团队要通过书本来学习Scrum，需要有一个人来负责指导、训练、指引和消除障碍等工作。可是，交付团队成长起来之后呢？指导减少，训练减少，尽管仍然需要在持续改进方面加以训练。指引还是必需的，不过这只占用很少的时间。随着团队学会了自己处理障碍，障碍也会减少。如果某个团队成员的全职工作是负责指导、训练、指引和消除障碍，而所有这些工作的工作量只占用他们10%到25%的时间的话，那他们多余的时间干什么呢？

所以，我们是这样操作的。我们也从Scrum起步，但并没有严格照搬书本。我们的项目经理同时负责好几个项目，所以要与多个项目团队协作。我们首先对项目经理进行敏捷培训。他们中有些参加的是高阶课程，有几个甚至拿到了Scrum Master和PMI的Agile Project Managers认证。于是，这些项目经理成了我们第一批Scrum Master。因为资源有限（我们的项目经理还没有团队数量多），或许也是有点先见之明，我们没有立即让Scrum Master去负责单个团队。没错，我们从一开始就违反了Scrum原则。一般来说，每个项目经理负责两个团队，他们承担着Scrum Master的角色。所以他们很忙，但是还能应付。正如大家能想象到的，他们的职责就是对团队进行Scrum指导训练、指引Scrum仪式并消除障碍。在前九个月甚至一年内，我们一直都是这样运作的。

很幸运，在这种运作方式之下，我们的Scrum团队成长起来了。当我在与员工的一对一谈话中不断被问到项目经理（注意，他们没被称呼为Scrum Master）在

团队中的作用时，我意识到，我们将项目经理作为Scrum Master的做法开始出问题了。交付团队成员觉得他们被不同的人过度管理了。他们已经有自己的职能经理（functional manager）了，现在又多一个Scrum Master，而且后者不是交付团队中的成员。

因此，我们在架构上做了一些改动。我们允许各个交付团队在内部选择自己的Scrum Master。这个Scrum Master仍然作为全职的交付团队成员，不过他同时还要负责指引Scrum仪式，并在出现障碍时作为第一联络人。那项目经理怎么安排呢？首先，他们还是项目经理，负责一些项目管理工作，譬如进行跨团队协作等。不过他们还被委以敏捷教练的职责。这样一来，敏捷训练不再是我们强加给交付团队的任务，而是成为一种应要求提供的支援。这样效果还不错。那些在敏捷方法和思想上已经成熟的团队，不再有被过度管理的感觉；而那些仍处于成长中的团队，他们也有人可以请教。

在我们继续成长的道路上，还有一个小小的转变。我们把项目经理（Project Manager）提升为程序经理（Program Manager）。所以，他们现在要负责跨团队协作的行动，比如那些跨产品以及涉及多个交付团队的项目。我们还在管理汇报上做了调整，我们把“职能经理/上级”改为“交付团队经理/上级”。后者我们称之为交付经理（delivery manager）。另外，现在是由交付经理来负责敏捷训练与指导了。我们现在仍然这么做，尽管想法古怪，但团队一直交付正常，所以似乎效果还不错。

对我们来说，Scrum Masters刚开始是个职位，然后随着我们在敏捷上的成长，演变成为一个角色。我们将Scrum Master的职责分成两个部分，一是训练与指导，另一个是指引与消除障碍。让团队进行自我管理，允许他们可持续成长，这样运作很有效，因为训练不再是强加给他们的任务，而是应要求提供的支援。而且，很多团队采用轮流担任Scrum Master角色的办法，这有利于让更多团队成员体验这个角色。此外，我们发现，交付团队里的所有成员都可以担任Scrum Master。比如，我们的业务分析师、开发人员、QA测试都当过Scrum Master。我们还发现，Scrum Master还时常在交付团队内部兼任着敏捷训练与指导的角色，因为那些有兴趣担负Scrum Master角色的人，往往也对学习敏捷原则感兴趣，他们愿意在团队里分享学习所得。

最后，让管理层高兴的最重要一点是，用较少的人完成同样的工作量。我们有16个交付团队。不采用专职Scrum Master，让我们腾出了16个人及其相关预算。正是这样，我们才得以拥有16个交付团队，而不是13个，我们需要这些多余的人员去填充真正有意义的角色。

实际上，我这篇文章是想提醒正在走向敏捷的各位。如果你雇佣一大把全职Scrum Master（就像圣路易斯这里一样），你需要有一个应对方案，以便在交付团队成熟以后知道该怎么做。至少在这里，大部分受雇的Scrum Master具有项目

管理背景。这没什么错，但正如我一再提到的，要准备好回答“现在如何安置这些曾经作为Scrum Master的项目经理呢？”这个问题。

关于作者



Brian Lawrence目前是密苏里州圣路易斯TriZetto Provider Solutions公司IT总监，他负责若干敏捷和看板产品开发团队与应用架构。自爱德华·尤登 (Edward Yourdan) 时代起，他就醉心于流程。他在职业生涯中，曾对开发组织进行过若干方法论改革，包括统一软件开发过程 (RUP)、精炼统一过程 (EssUP) 和敏捷。他热爱过程改进，寻求更好、更快、代价更低的软件开发方式。他目前正热衷于敏捷环境中的管理方式。

原文链接：<http://www.infoq.com/cn/articles/scrum-master-position-role>

相关内容

- [你要招聘怎样的Scrum Master？](#)
- [项目经理和Scrum Master](#)
- [怎样做一名高效的ScrumMaster](#)
- [采访和书评：《Essential Scrum》](#)
- [实施敏捷时可以检查的一些事项](#)

特别专栏 | Column

敏捷流畅度：找到适合需求的敏捷方式

作者 [Diana Larsen](#)，译者 邵思华

“从本质上说，所有的模型都是错的，但有些模型还是会起到作用。”这段话来自George E.P.Box，他是来自美国威斯康星大学的一位统计学家，并且是该大学的“质量与生产力改善中心”的创始人。

坊间一直存在一种说法，即James Shore和Diana Larsen共同设计了一种敏捷成熟度模型（AMM），它推出了一套敏捷的标准与级别的提议，该提议同时还包括了参与者评估与认证方案的内容。

事实并非如此！原因在于：我们只是在经过实地的测试后，推出了一个专注利益和交换的模型，我们并没有为这个模型定义任何成熟或者不成熟的标签。

大约3年以前，我们（James Shore和我）对于我们之前教授和指导如何在团队工作中吸收敏捷原则与实践的方法与技术进行了一次认真的评审，经过一系列的尝试之后，我们得到了一个模型的草稿。

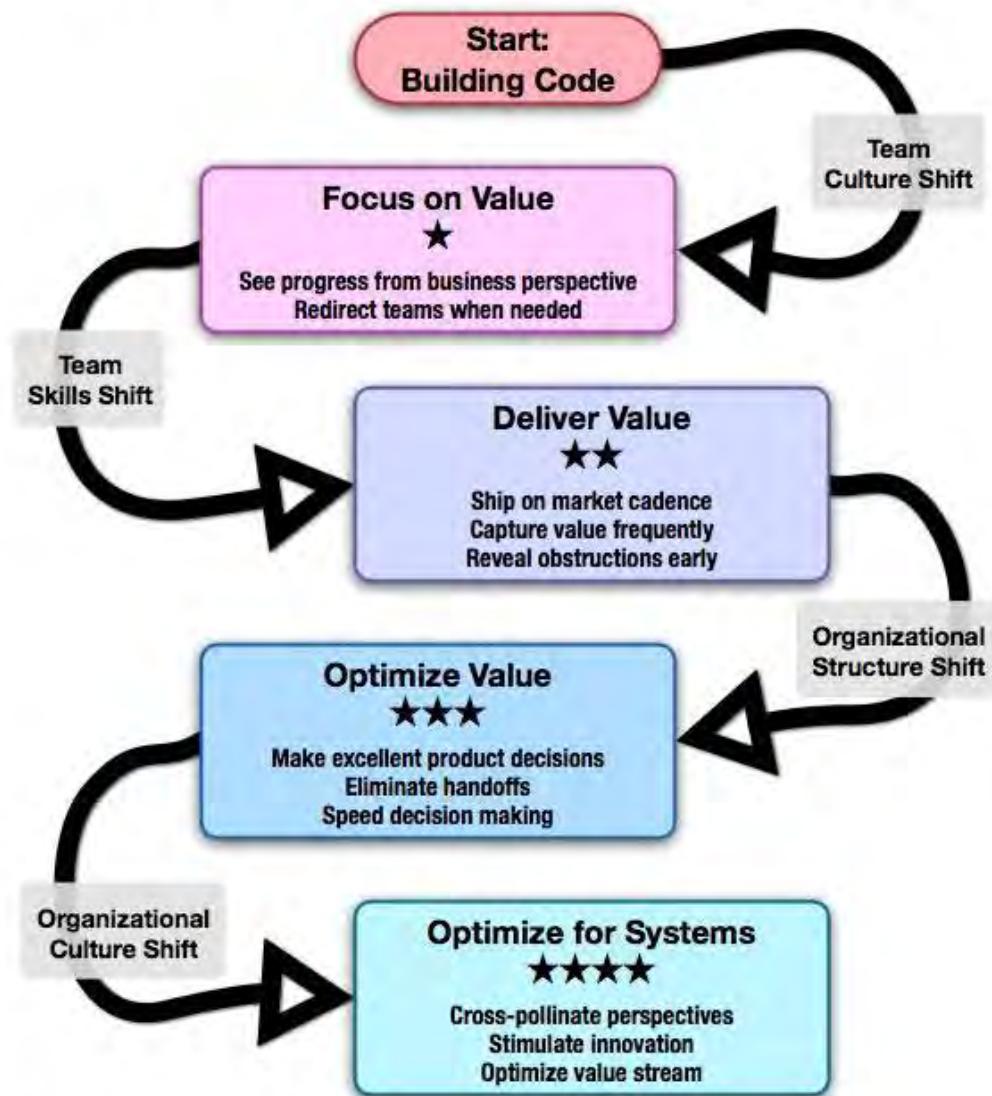
经过了多位评审者的审查以及持续不断地修正之后，我们认为这个模型已经足够实用，可以将它推广开来了。在2012年8月，Martin Fowler在他的个人网站发布了我们的文章“[Your Path through Agile Fluency](#)”，其中对敏捷流畅度模型进行了描述。

“我们观察到敏捷团队的成长会经历4个不同阶段的流畅度，流畅度是指某个团队在面临压力时开发软件的方式。如果有足够的时间待在课堂里，每个人都能做到遵循一系列良好实践，但真正的流畅度是一种有技巧的日常实践，即使当你为其它事分心的时候，这种流畅度也不会离你而去。

对于敏捷来说，我们更关注于团队的流畅度，而不是个人或整个组织的流畅度。敏捷开发从本质上来说是一个团队任务，敏捷能否在整个组织中取得成功，要取决于团队的流畅度。

而团队的流畅度则更多地取决于团队中每个成员的能力，也取决于管理结构、关系、组织的文化等方面。请别误会，这并不是说团队的流畅度低下要归咎于个人，也不是说一个高水平成员的存在就能够保证整个团队的流畅度。

Path Through Agile Fluency



© 2012 James Shore and Diana Larsen.
You may reproduce this diagram in any form so long as this copyright notice is preserved.

简单的来说，按照我们在多个组织中的观察来看，敏捷团队的敏捷流畅度会通过4个不同的阶段演进。我们这样命名这4个阶段：关注价值（一星级流畅度）、交付价值（二星级流畅度）、优化价值（三星级流畅度）和优化系统（四星级流畅度）。经验告诉我们，不同的组织在每个阶段似乎已找到了团队与商业需求之间的匹配度。我们也注意到，追求达到二星、三星或四星级流畅度的团队似乎在试图以一种可预测的模式获得流畅度，首先是获取在团队工作中所需的技能、专注于业务与客户价值，随后建立起足够的工程技术知识，根据市场节奏进行交付。

人们问：通住流畅敏捷之路是否是一种成熟度模型？

从那时起，一方面我们从同事中获得的正面反馈让我们非常高兴，另一方面，对于我们创建了一个敏捷成熟度模型的断言又我们非常困扰。对于一个新的成熟度

模型的观点各不相同，有批评，也有掌声！我们是否无意中创建了一个成熟度模型呢？你可以想象我们的沮丧，因为这一点从来都不是我们的初衷。

软件开发中最为人熟知的“成熟度模型”（MM）大概是能力成熟度整合模式（CMMI）了，它是CMM和其它成熟度模型的一种集成。有些人断言它是一种商业过程改进的框架，甚至是一种建立商业过程改进系统的模型，包括开发、采购和服务等过程。这些人在网站中问道：“如果某个公司对它们的内部商业过程缺乏洞察力和控制力，这个公司又怎能够了解它们做的是否足够好呢？也许等它们发现问题时一切都已经太迟了”。

嗯，或许原因在于：“最值得去做的项目，是那些足以将你的商业过程规模降低整整一个级别的项目。”（DeMarco & Lister，来自Peopleware）。此外，对于整理后的最佳实践的专注也遭到了抨击，它被认为试图建立一种可以适应所有情况的解决方案。毕竟，在业界不可能存在两个完全类似的开发项目或部门，可以用一种相同的一揽子方案来解决它们的需求。

某一个认证团队（也是各种认证的提倡者）说道：“成熟度是由每一个‘成熟度级别’的奖励所体现的”。我们很怀疑他们是否注意到这种说法有多么荒谬和自私？

综合我们能够找到的所有对成熟度模型的定义，我们认为成熟度模型有以下特点：

- 专注于流程改进系统
- 对于行为、实践和过程等指标的“最佳实践”的一种结构化分级
- 对它的使用包括了对各种比较结果的评估，并且有助于理解这些比较的结果

从定义来看，我们的模型并不是一种成熟度模型。首先，我们并非结构化地按照级别顺序描述好的、更好的、最好的团队，而是对整个流畅之路的每一阶段都同等进行描述。这个模型并没有对流程改进的评估，而是反映了团队的注意力和组织的投入的调整情况。其次，虽然这个敏捷流畅度模型会对可观察的行为，包括实践和流程进行描述，但我们并不打算将这些观察结果作为事后比较评估的依据。这并非我们的本意，天哪！

我们绝对不想把这个模型作为一个工具，以作为打击某些团队“缺乏成熟度”的理由，而是希望把它作为一种技术，让它能够设立和实现专属于你的团队的前进动力，无论你的团队当前处于什么状况，它们都是最适合于你的目标。

找到你的目标

在维也纳举办的XP2013大会上，Diana说道：“不能说因为威尼斯更远，就认为它比慕尼黑要好，尤其是如果你要参加慕尼黑啤酒节的话！另一方面，如果你想参加一个盛大嘉年华的话，那威尼斯就是你的选择，即使去威尼斯要消耗你更多

的汽车燃烧，这一选择也是明智的。”在俄勒冈，她又说道：“从波特兰沿着州际高速公路开1至5公里就能到达阿什兰，那里有着历史悠久的盛大的莎士比亚戏剧节。但如果你的目的是去国会大厦做生意，那么塞伦才是你应该去的地方，虽然它没有阿什兰那么优美。”

与之类似，一个二星级团队并不代表它比一星级或三星级团队要做得更差或者更好。这取决于许多环境因素，尤其是取决于它对于实现目标的合适度。或者如果我们把流畅敏捷之路的每一点用公园中的某个位置来命名能够更清晰地表达我们的意图（不过在图标库中更容易找到星星就是了）。我们或者可以选择这样的名称：停车场团队、操场团队、运动场团队，以及登山道团队。选择公园中的哪个位置完全取决于你想做些什么活动。在去操场的路上，你可能需要在停车场放下自行车，或者从穿梭巴士上下车。或者你必需先到操场去，才能找到去运动场的路。而如果你很想留在操场上荡秋千或是玩跷跷板，那就没必要往登山道前进了。

为了有效地使用我们的敏捷流畅度模型，我们建议你首先抛弃团队成熟度这种思想，也别让交付商业价值之外的任何思想来评判你的团队。你要做的是花一点时间去理解以下这两点：

- 你需要从团队中获得什么价值，并希望你的团队创造什么价值？
- 你愿意为此投入多少精力和交换代价？

是星级还是公园里的位置？

一颗星（或者说停车场）——你真的需要某个专注于创造商业价值和客户价值的团队吗？你是否需要能够与其他人协作进行计划、优先级安排、并且为了实现目标能够踏实工作的人？你是否希望能够很快指出团队是否在向交付目标的方向前进？如果能够满足你的目标，那么一个一星级的熟练团队对你来说就是最合适。你需要投入一定的精力来建立一个强大的、互信的、有协作意愿的团队，以及一个专注于将工作内容按照交付价值进行优先级排序的工作流程设计方法（例如Scrum、Kanban或Scrumban）。

如果这正是你需要的，那么就为建立一个团队文化做些计划和投入。你已经找到的正确的目标，因此请停在那里。但如果这并非最适合于你的情况，那么请考虑一些不同的投入。

两颗星（或者说操场）——你是否需要一星团队所拥有的专注于价值的特点，而且你的客户要求交付的产品的缺陷率为0、或者非常低，并且期望你能够按照一个常规的预定义节奏进行交付？如果是的话，那么你需要投入精力去招聘人才，而更可能的是你需要团队成员的成长，他们需要掌握各种工程技术能力，例如持续集成、TDD、结对编程（或结对工作）、持续部署以及集体所有制（这些概念

来自于极限编程、软件工艺、DevOps等等）。虽然一开始时会以团队的生产力下降作为代价，但当团队成熟后就能够完全弥补这些投入，毕竟对于业务来说高收益是非常关键的。

我所合作过的大多数组织都会从二星团队中获得充分的价值。不过，许多敏捷教练、思想领袖和一些商务人士（包括我们）都希望看到一种更有抱负的实现，这对团队和组织来说可能会更好，也可能不会。

如果这正是你需要的，那么就为建立一个团队文化、并且开发团队成员的技术能力做些计划和投入。你已经找到的正确的目标，因此请停在那里。但如果这并非最适合于你的情况，那么请考虑一些不同的投入。

三颗星（或者叫运动场）——除了二星团队的价值之外，你是否期望了解你的开发团队是否已经抓住了产品的所有价值？你是否期望你的团队（或多个团队）对你的业务和客户需求有着良好的理解，因而能够贡献出创造性的产品开发思想呢？你是否愿意让团队能够持续地了解业务知识，并且在软件开发者、产品开发者和业务策划专家之间建立信任关系呢？你是否为了发现不断发展的客户与产品而给每个团队都配备了专门的产品经理/负责人专家（可以是个人或小组的方式）呢（精益创业的思想对此有所帮助）？三星流畅度的价值通常可以称为“对敏捷的承诺”，无论是不断变化的组织结构、重新定义的管理角色、还是不断变化的行政管理与运营过程，面对这些令人生畏的挑战都能从容应对。这些都是实现三星敏捷流畅度所不可或缺的（提示：千万别相信那些组织发展顾问或者敏捷教练所说的，他们会告诉你实现这些很简单快速，而且只要一个Scrum Master就能办到了）。

如果这正是你需要的，那么就为建立一个团队文化、开发团队成员的技术能力、并且调整你的组织设计做些计划和投入。你已经找到的正确的目标，因此请停在那里。但如果这并非最适合于你的情况，那么请考虑一些不同的投入。

四颗星（或者叫登山道）——你是否希望某个开发团队成为整个组织商务动作的重要一环，为新产品交付有创造性的思想，并且积极参与设定前进方向的对话？你是否希望团队能够为了对整个公司来说更重要的价值而暂时停下他们手头上优先级最高的工作？你是否迫切希望在决策会议上为团队成员保留一个座位，只因为你相信他们很清楚每个决定是如何影响到公司中的每个关系人的？如果你的公司并不是一家小型初创企业，你是否愿意投入精力进行一次全盘的企业文化改变？有些人将此称为“超越敏捷”，我们则建议敏捷最终往这个目标前进，因此整个社区在持续地推进它对敏捷的各种可能性的理解。

这些公园里位置的名称能够更好的表达我们的意图，但我觉得星更容易记住，只要你能明白星数越多并不一定代表对你、你的组织和你的处境越好就可以了。最适合你的目标、你的能力、以及你对改变现状所愿意投入的时间、精力、资产和

社会资本的，就是最好的流畅度。

计划你的团队投入

成熟度模型将我们指到了错误的方向，它是对人们的活动进行评估，而不是对结果进行评估。敏捷本身强调的是为客户和商业提供价值，它让我们将目光投入在有价值的东西上：即创造并维护高质量的、有用的、市场驱动的软件。我们看到许多公司能够指出适合他们需求的各种敏捷流畅度，并且为改变文化、员工培训和结构调整方面做出了投入，而这些公司都获得了巨大的回报。我们希望敏捷流畅度模型将证明它是一个贡献出其作用的模型之一，尽量它未必完全正确。敏捷流畅度模型是一种思考与计划敏捷投资的方式，它能够帮助你创建最适合于公司的研发投入、商业需求和客户价值的敏捷环境。

关于作者



Diana Larsen是FutureWorks Consulting公司的合伙创始人之一，她的工作专注于敏捷软件开发、团队集团和敏捷转型。她是以下几本书的共同作者之一：《Agile Retrospectives: Making Good Teams Great》、《Liftoff: Launching Agile Teams and Projects》和《QuickStart Guide to Five Rules for Accelerated Learning, and the article》，此外还撰写了一篇文章：“[Your Path through Agile Fluency: A Brief Guide to Success with Agile](#)”，这些书籍与文章深入地讲述了工作团队如何成长、适应与进步。

原文链接：<http://www.infoq.com/cn/articles/agile-fluency-fit-purpose>

相关内容

- [增进敏捷团队中测试和开发人员间的协作](#)
- [Justin James谈高效开发者的特质](#)
- [架构师（4月刊）](#)
- [物理墙和虚拟墙之争](#)
- [阅读者（二十三）——《精益和敏捷开发大型应用指南》](#)

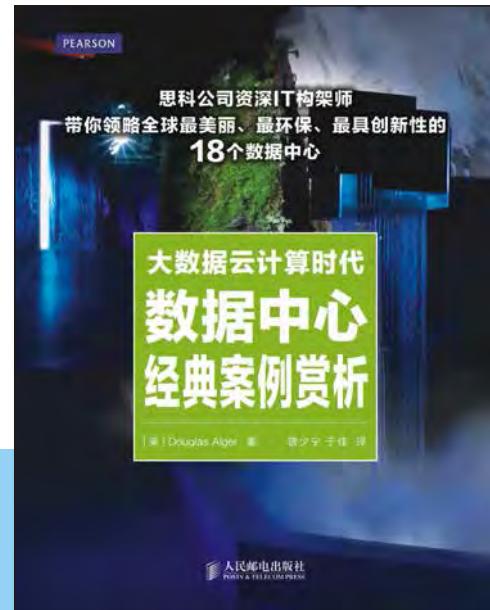
- * 思科公司资深IT构架师最新力作
- * 全面介绍世界上18个极具特色的数据中心
- * 采访问答方式，直接与项目单位就技术关键点展开探讨
- * 丰富精美的图片，直观感受数据中心之美

思科资深技术专家通过与数据中心设计师的访谈，揭示了数据中心决策要点，并说明了如何建设数据中心以及如何面对其他挑战，其中包含了思科，易趣，脸谱，雅虎等著名企业在不同环境下的的数据中心实践案例。思科资深技术专家通过与数据中心设计师的访谈，揭示了数据中心决策要点，并说明了如何建设数据中心以及如何面对其他挑战，其中包含了思

大数据云计算时代 数据中心经典案例赏析

作者：[美] Douglas Alger

译者：曾少宁 于佳



- * 最畅销的Oracle RAC原创图书之一
- * 立足于云端时代的全新视界
- * 安装、平台、私有云、高可用性全是干货

全书分4个部分，第一部分“安装”，从安装入手，分析安装过程出现中的新元素，第二部分“平台”，着重介绍Grid，包括Grid的内部组成、ASM、ADVM、ACFS、SCAN和SIHA等。

大话Oracle Grid：云时代的RAC

作者：张晓明





图灵社区 iTuring.cn



图灵教育微信



图灵访谈微信

图灵教育推荐

在线出版

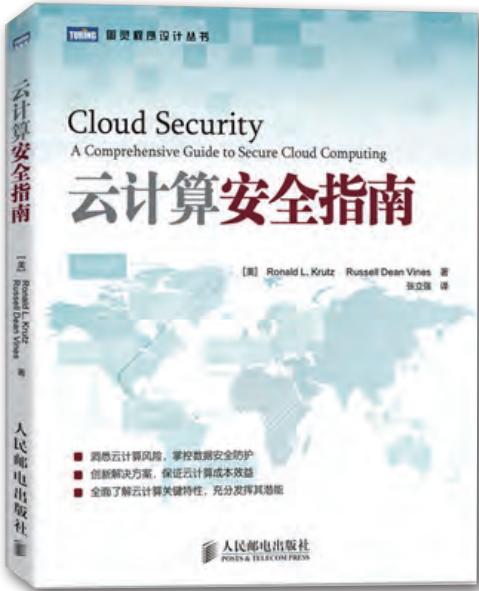
电子书

《码农》杂志

图灵访谈

线下活动

技术交流



- 洞悉云计算风险，掌控数据安全防护
- 创新解决方案，保证云计算成本效益
- 全面了解云计算关键特性，充分发挥其潜能

《云计算安全指南》全面探讨云的基础知识、架构、风险、安全原则，让我们不仅了解其灵活性、高效性、经济实用性，而且充分认识其脆弱性并学会补救方法。两位知名的安全专家 Ronald L. Krutz 和 Russell Dean Vines 将带你攻克数据所有权、隐私保护、数据机动性、服务质量与服务级别、带宽成本、数据防护、支持等难题，帮助你保证信息安全并通过云计算获得最大的投资回报。

书名：云计算安全指南

作者：[美] Ronald L.Krutz,
Russell Dean Vines

译者：张立强

书号：978-7-115-32150-3

详情请点击：<http://www.ituring.com.cn/book/991>

- 国内第一本云计算网络书
- 云计算与大数据时代，网络技术人员必看
- “弯曲评论”网站“拨云见日”系列热文加量
- 10倍的强烈之书首次完整呈现

通过阅读本书，读者将清楚地了解到如何在云计算与大数据时代构建安全、可靠、高速与灵活的网络。本书主要内容包括：云计算对基础架构的驱动、云计算网络的组成、如何构建安全可靠灵活的网络通道、虚拟化数据中心的扩张、外部和内部网络的实现、大数据网络设计要点，以及厂商解决方案等等。

本书语言通俗易懂，内容深入浅出，可作为云计算网络技术入门和提高阶段的自学、参考书籍。适合国内云计算网络、新一代网络建设、网络管理、系统集成行业的开发人员、技术工程师、售前与售后技术支持人员学习。

详情请点击：<http://www.ituring.com.cn/book/966>

书名：腾云：云计算和大数据时代网络

技术揭秘

作者：徐立冰

书号：978-7-115-31150-4

我们在微博：@图灵教育 @图灵新知 @图灵社区

我们在微信：图灵教育: turingbooks 图灵访谈: ituring_interview

读者俱乐部：218139230 (QQ群)

避开那些坑 | Void

从MVC在前端开发中的局限性谈起

作者 李光毅

有名无实的Router

ASP.NET MVC

如果你还没有接触过后台的MVC框架的话，不妨先看看下面这段ASP.NET MVC代码并且了解一下后台MVC的工作原理。它摘自ASP.NET MVC教程中非常著名的项目[MVC Music Store](#)一段Controller组件代码：

```
public class StoreManagerController : Controller
{
    private MusicStoreEntities db = new MusicStoreEntities();
    // GET: /StoreManager/
    public ViewResult Index()
    {
        var albums = db.Albums.Include(a => a.Genre).Include(a => a.Artist);
        return View(albums.ToList());
    }

    // GET: /StoreManager/Details/5

    public ViewResult Details(int id)
    {
        Album album = db.Albums.Find(id);
        return View(album);
    }
}
```

我们知道Controller的职责之一是负责响应用户在视图上的行为，而具体每个行为应如何进行响应，需要落实到Controller具体的方法上，这个方法我们可以称之为action。上面代码中的两个公开方法Index()与Details()就是两个action。它们都属于StoreManager这个Controller。如果你有使用过前端的Ember.js的话，应该对这两个概念非常熟悉。

但问题来了，如何将用户在视图上的行为，与响应行为的方法action关联起来？甚至与Controller关联起来？URL便是方法之一。上面代码每个action上的注释便代表这个action对应的URL。也就是说，当用户点击该URL时，框架中的

Router服务便能通过URL解析出应该调用哪个Controller及该Controller下的哪一个action进行响应，以上面的例子为例，可以知道URL的规则为{controller}/{action}/{id}。

那么响应的结果应该是什么呢？从上面的代码ViewResult和return View()两处可以看出，两个action返回的都是新的视图。

举一个最熟悉的现有MVC站点的例子便是[github](#)。你会发现你在github网站上的每一处点击都有唯一的URL对应，每一次交互的结果都是服务器返回新的页面。它使用javascript非常少，比如当你选择编辑时，它也会跳转到一个新的页面，而非在当前页弹出一个编辑框。

为什么首先要聊这么多的服务器端MVC框架的特性。因为接下来回过头来看前端的MVC框架时，你会发现有非常多的差异之处。

Javascript MVC

从上面可以看出，服务器端的MVC框架服务的是整个站点，它依靠不断的返回页面来响应用户请求，因此Router服务至关重要。而使用MVC框架的前端页面，大多数是Single Page Application，甚至还不如单页面，只是页面上的某一个组件，比如一个Slide。因此将用户的行为转化为URL是不现实。你或许会说的确无法生产新的页面，那么降低页面粒度如何呢？也就是说在服务器端一个URL映射的是一个页面，那么我们将URL映射为页面的某个区域或者功能呢？

比如以下面这段Backbone.js的TodoList应用Router为例：

```
var TodoRouter = Backbone.Router.extend({
  routes: {
    'todo/add': 'add', // 新增项
    'todo/edit/:id': 'edit', // 编辑项
    'todo/remove/:id': 'remove', // 删除项

    'filter/completed': 'filterCompleted', // 过滤出已完成
    'filter/uncompleted': 'filterUncompleted' // 过滤出未完成
  }
  // Todo
});
```

如果依照这样Route规划，我们希望当用户输入http://example.com#todo/add时，我们弹出的是一个新增输入框；而当用户输入http://example.com#todo/edit/123456页面出现编辑id为123456的这条记事的编辑框。这样我们便将URL映射的页面粒度降低为输入框粒度。

但是这样会引起另一个问题，注意上面route的差别：todo/域名下操作的是单条的记录，而filter/域名下操作的是对列表进行筛选。所以还不得不考虑一种情况，如果用户想在筛选的情况下是否对每一项进行操作？如果允许的话，参考排列组合，route是否需要新增为 $2 \times 3 = 6$ 项？如新增<http://example.com/filter/completed/todo/add>这样的路由。

这样的设定明显是不合理的。之所以会产生这样的问题是由于对后端而言URL与页面是一一对应的关系。而如果降低页面粒度的话，无法将页面功能与URL对应起来，或者说如果想让URI覆盖单一页面上的所有功能的成本太高了。

前端的解决之道

当然Route在前端MVC框架中并非武功尽废，我们仍然可以保留这样的机制，但是仅用于高粒度的操作，比如上面例子中的筛选功能。

其实Route仅仅是桥梁，将用户的行为与响应用户行为的方法联系起来。Route机制已经在前端行不通了，问题便是寻求另一种联系的机制。

Observer Pattern + MV

Google程序员[Addy Osmani](#)(同时也是[Learning JavaScript Design Patterns](#)的作者)有一个非常著名的项目——[Todomvc](#)。他用所有的前端MVC框架将To do List这个app重写了一遍。我们看看他的解决这样的问题的。

Backbone

以Backbone.js为例，为了方便说明，在他的基础上我再次进行了简化，简化到只有50行代码，只保留了新增和删除方法。实际演示效果在[这里](#)：

```
var Todo = Backbone.Model.extend();

var Todos = Backbone.Collection.extend({
  model: Todo
});

var todos = new Todos();

var ItemView = Backbone.View.extend({
  tagName: "li",
  template: _.template($("#item-template").html()),
  render: function () {
    this.$el.html(this.template(this.model.toJSON()));
    return this;
  }
},
```

```

initialize: function () {
    this.listenTo(this.model, 'remove', this.remove);
},
events: {
    "click .delete": "clear"
},
clear: function () {
    todos.remove(this.model);
}
});

var AppView = Backbone.View.extend({
el: $("body"),
initialize: function () {
    this.listenTo(todos, 'add', this.addOne);
},
addOne: function(todo) {
    var view = new ItemView({
        model: todo
    });
    this.$("#list").append(view.render().el);
},
events: {
    "click #create": "create"
},
create: function () {
    var model = new Todo({
        title: this.$("#input").val()
    });
    todos.add(model);
}
})
}

var app = new AppView();

```

这个应用有两个视图，全局视图`AppView`和单个项目视图`ItemView`，还有一个相当于Model层的Collection`todos`。它没有显式的将Controller层独立出来。你可能会说Backbone不像Angular和Ember，它本就没有Controller的设定，我不这么认为，Backbone里Router就是Controller与Router的结合体。比如它可以加入一个Router这么干：

```

var TodoRouter = Backbone.Router.extend({
routes: {
    'todo/add': 'add'
},
add: function () {
    // do something
}
})

```

```
});
```

翻译过来是：#todo/add对应的action是add——如果放在服务端的话这不就是把用户的行为(URL)与action联系起来了吗。

回到这个应用，它没有将Controller显式的独立出来并不代表Controller不存在。我们需要的不是一个叫做**Controller**的东西，而是一些能够响应用户行为的方法。那么它把Controller放在哪了？它使用了的是最原始的解决办法，放在用户行为事件的回调函数中，比如这段代码：

```
events: {
  "click #create": "create"
},
create: function () {
  var model = new Todo({
    title: this.$("#input").val()
  });
  todos.add(model);
}
```

虽然这种方法比较原始，但不应该认为它是一种低级的解决方案。当然接下来我们会在Emberjs和Knockoutjs看到一些其他的解决方法，从职责上来说，他们的目的是一致的。

回想在文章开头介绍的ASP.NET MVC的一个action，它做了哪些事情：

```
public ViewResult Details(int id)
{
  // 执行了Model层的查找(query)操作
  Album album = db.Albums.Find(id);
  // 将查找返回的结果传递给视图(View)
  return View(album);
}
```

用户的操作会涉及数据和视图更改，一般来说视图会被动一些，Model的修改可以通知视图的更新，当然视图也可以向Model请求数据（注意从这里可以看出Model与View可以直接通信而非通过Controller）。上面的代码很好的说明了这一点，并且视图的更新和数据的操作都是在同一个action里完成。

但是前端MVC框架却不这么做。

因为在前后端代码与视图的关系是不同的，后端代码生产出页面交给浏览器之后，基本上就不关心也无法关心页面了；而在前端不同，虽然前端也有“生产”这么

一说（比如说使用模板引擎），但前端代码时刻都可以对HTML代码和DOM元素进行监听和操作。

上面的Todo里Backbone是这么做的：

```
var AppView = Backbone.View.extend({
  el: $("body"),
  initialize: function () {
    // 在初始化该视图时(执行当前initialize函数),
    // 它会对这个应用唯一的Model层，就是Collection`todos`进行监听;

    // 更准确来说是对它的新增操作进行监听
    // 一旦发现数据有新增操作，那么更新视图(调用addOne方法)
    this.listenTo(todos, 'add', this.addOne);
  },
  addOne: function(todo) {
    var view = new ItemView({
      model: todo
    });
    this.$("#list").append(view.render().el);
  }
})
```

这个时候你再看看这个AppView的代码会觉得有一些绕，因为我们明明可以在create方法中，在新增数据之后立即调用更新视图的方法：

```
create: function () {
  var model = new Todo({
    title: this.$("#input").val()
  });
  todos.add(model);
  // 更新视图
  // this.addOne(model)
}
```

但我想这么做的原因无非是为了解耦。考虑今后可能增加多个新增数据的入口(如create1, create2, create3)，在每一个新增方法中调用更新视图的方法会增加代码的维护量，那么就不如采用这种观察者模式(Observer pattern)(或者采用fire事件机制也行)，只在指定处更新视图。

ItemView采用的也是同样机制，就不在这里赘述了。

值得一提的是，todomvc Backbone的原版本中仍然保留了Router服务：

```
var TodoRouter = Backbone.Router.extend({})
```

```

    routes: {
      '*filter': 'setFilter'
    },

    setFilter: function (param) {
      // Set the current filter to be used
      app.TodoFilter = param || '';

      // Trigger a collection filter event, causing hiding/unhiding
      // of Todo view items
      app.todos.trigger('filter');
    }
  });
}

```

如果有兴趣浏览一下原版代码，你会发现这只是作为更新视图的触发器而已。

Ember.js

不仅仅Backbone采用这样的机制，我们也可以看看todomvc中简化过后的Emberjs代码，采用的也是同样机制，线上[DEMO](#)：

Html:

```

<script type="text/x-handlebars" data-template-name="index">
{{ input type="text" value=title }}
<button {{action "create"}}>Add</button>
<ul>
{{#each item in content itemController="todo"}}
  <li>
    <span>{{item.title}}</span>
    <button {{action "remove"}}>delete</button>
  </li>
{{/each}}
</ul>
</script>

```

Javascript:

```

App = Ember.Application.create({});

App.ApplicationAdapter = DS.LSAdapter.extend();

App.Todo = DS.Model.extend({
  title: DS.attr("string")
})

```

```

App.IndexRoute = Ember.Route.extend({
  setupController: function(controller) {
    var todos = this.store.find('todo');
    controller.set("content", todos);
  }
});

App.TodoController = Ember.ObjectController.extend({
  actions: {
    remove: function () {
      var todo = this.get('model');
      todo.deleteRecord();
      todo.save();
    }
  }
});

App.IndexController = Ember.ArrayController.extend({
  actions: {
    create: function () {
      var title = this.get("title");
      var todo = this.store.createRecord('todo', {
        title: title
      });
      todo.save();
    }
  }
});

```

这一段代码不做详解，与前一个Backbone的机制是一致的，需要注意的是：

1. 如果你觉得Backbone中将用户行为与方法联系起来的做法比较原始的话，不妨看看Emberjs的解决方法，它直接写进需要编译的模板中：

```
// 指定调用IndexController中的create的方法(action)响应按钮的点击事件 <button {{action "create"}}>Add .....
```

```
// 指定TodoController中的remove的方法(action)响应 <button {{action "remove"}}>delete
```

2. 在新增事项的create方法中，它做的也仅是对Model层进行数据操作，

```

create: function () {
  var title = this.get("title");
  var todo = this.store.createRecord('todo', {
    title: title
  });
  todo.save();
}

```

但你不用调用任何的视图方法，视图已经随数据更新了。这不也是一种Observer模式嘛。

3. 值得一提的是Ember.js中的Router有一些特别，除了一般路由拥有的定义URL规则之外，它还负责向对应的Controller提供model层数据。

MVP(Passive View)

Addy的解决方案是Observer Pattern。但我更提倡另一种解决方案，MVP(Model-View-Presenter)模式中的Passive View模式。

为什么会有MVC，MVP，甚至MVVM？三个模式其实主要围绕的是两个问题：

1. 一是Model与View之间的通信问题，完全隔离的？单向通信还是双向通信？
2. 二是M-V-XX中的"XX"需要完成哪些功能，简单流程调度？还是复杂规则处理？

MVP模式将用户的交互逻辑的处理流程定义在Presenter层中，但是具体的实现并不是完全在Presenter中。View和Presenter采用单向的沟通方式。View单纯地将用户的交互请求汇报给Presenter；Presenter接收到请求之后，整合相应的资源、执行相应的处理逻辑。对处理流程的某一个步骤，如果设置到业务逻辑和数据模型，则调用Model，如果涉及到对视图的更新，还会调用View。Presenter和View接口都应该只包含返回类型为void的方法即可

同时参考Kjell-Sverre Jerijærvi提出的MVP的设计原则[Design Rules for Model-View-Presenter](#)，可以找到MVP模式中一些非常重要的特征，例如以下几条：

- View不允许通过Presenter直接调用Model和Service，并且Presenter的方法应该是不具有返回值的；
- Presenter必须通过View接口的方式调用View
- 除了对View接口成员的实现外，View中的其他方法不应该是public的；
- View接口的成员应该仅限于方法，不应该包含属性；
- 所有的数据应用保持在Model中

既然Presenter对于View是相对透明的，View不能直接对Presenter进行操作（目的是实现Presenter和View之间的分离）。那么如何实现View与Presenter层之间的通信呢——通过注册事件，在Presenter上注册View的事件，并且这

样以来数据的传递就不是View向Presenter去“拉(pull)”，而是Presenter“推(push)”给View层。

所以我们的目标非常的简单，分别定义三个层：

1. Model层：提供操作数据的接口
2. View层：提供更新视图的接口，当用户有行为发生时触发事件
3. Presenter层：定义用户的交互逻辑与流程，但所有与数据操作和视图的更新都通过调用Model层与View层的接口实现，注册View层事件。

实现代码如下：

```

var Todo = Backbone.Model.extend({
});

var Todos = Backbone.Collection.extend({
  model: Todo
});

var ItemView = Backbone.View.extend({
  tagName: "li",
  template: _.template($("#item-template").html()),
  render: function (container) {
    this.$el.html(this.template(this.model.toJSON()));
    container.append(this.$el);
  },
  events: {
    "click .delete": "_clear"
  },
  _clear: function () {
    this.$el.trigger("delete", this);
  }
});

var AppView = Backbone.View.extend({
  el: $("body"),
  events: {
    "click #create": "_create"
  },
  _create: function () {
    var data = {
      title: this.$("#input").val()
    };

    this.$el.trigger("create", data);
  }
});

```

```

    })

var Presenter = Backbone.View.extend({
  el: $("body"),
  initialize: function () {

    this.appView = new AppView({
      container: $("#list")
    });
    this.todos = new Todos;
    var _this = this;

    this.$el.on("create", function (e, attrs) {
      var model = new Todo(attrs);
      _this.todos.add(model);
      var itemView = new ItemView({
        model: model
      })
      itemView.render($("#list"));
    });

    this.$el.on("delete", function (e, view) {
      _this.todos.remove(view.model);
      view.remove();
    });
  }
}

var p = new Presenter;

[DEMO] (http://jsfiddle.net/JPL94/8/)

```

让我们来看看它是否复合我们的要求：

- AppView不提供任何接口(所有的私有方法都以_开始)，只负责捕捉用户的行为。一旦用户点击新增，即向外广播“新增”事件.trigger("create", data)
- itemView提供一个插入新视图的接口render与移除视图接口remove，并且捕捉用户的删除行为，广播“删除”事件
- Presenter层捕获到事件后，执行一系列的流程，但皆以调用Model层与View层接口的方式执行，而非直接操纵实例属性。

上面的两个例子我们可以看到，介于前端的特殊性，在前端MVC框架中View层与Controller层中的action联系较为密切，例如把action注册到视图元素的事件回调中。虽然这样能够代替服务器端的路由方案，但代价是牺牲了不同层之间的低耦合性，并且不易测试。

MVP在MVC的基础上，进一步进行了解耦，把视图和数据抽象仅剩下接口，并且把业务逻辑全部归纳到Presenter层中。这样不仅能够把View单独拎出测试，还能提高这一套MVP组件的可复用性。如果我们想更换View层或者Model层的话，只要保证新层具有与原层相同的接口，而不用涉及到其他层的修改。

MVVM

最后一个解决方案是MVVM(Model-View-ViewModel)，以Knockout.js为例，完成相同功能只需要非常少的代码：

Html:

```
<input type="text" data-bind="value: title">
<button data-bind="click: create">Add</button>
<ul data-bind="foreach: todos">
    <li>
        <span data-bind="text: title"> </span>
        <button data-bind="click: $root.remove">delete</button>
    </li>
</ul>
```

Javascript:

```
function Todo (title) {
    this.title = title;
}

var ViewModel = function() {

    var _this = this;
    this.todos = ko.observableArray([new Todo("test1"), new Todo("test2")]);
    this.title = ko.observable();

    this.create = function () {
        var title = this.title().trim();
        if (title) {
            _this.todos.push(new Todo(title));
        }
    }

    this.remove = function (todo) {
        _this.todos.remove(todo);
    }
};
```

```
ko.applyBindings(new ViewModel());
```

[DEMO] (<http://jsfiddle.net/mfUVk/>)

在MVP与MVC模式中，用于渲染视图的数据往往是action处理Model层返回的数据之后的结果。而MVVM中的ViewModel则是视图所需要渲染数据的直接映射，无需再经过其他服务的处理。

我不确定MVVM在WPF中(MVVM起源于微软Windows Presentation Foundation框架)是如何处理ViewModel层方法与数据的关系。但是从上面Knockoutjs的代码可以看出，ViewModel的代码是比较混乱，方法和属性都书写在一起。其实从html的绑定方式也可以看出，无论是数据还是action，都一视同仁的使用data-bind属性。

虽然Knockout的代码写起来很舒服(我们近乎只用关心Model层的数据操作)，但这样的代码无疑对代码的复用和维护提出了挑战。仅作参考，不作推荐。

参考文献：

- [Digesting JavaScript MVC - Pattern Abuse Or Evolution?](#)
- [JavaScript MVC](#)
- [Single page apps in depth](#)
- [Journey Through The JavaScript MVC Jungle](#)
- [大话MVP](#)
- [谈谈关于MVP模式中V-P交互问题](#)
- [从三层架构到MVC,MVP](#)

原文链接：<http://www.infoq.com/cn/articles/startng-from-limits-of-mvc-in-front-end-development>

相关内容

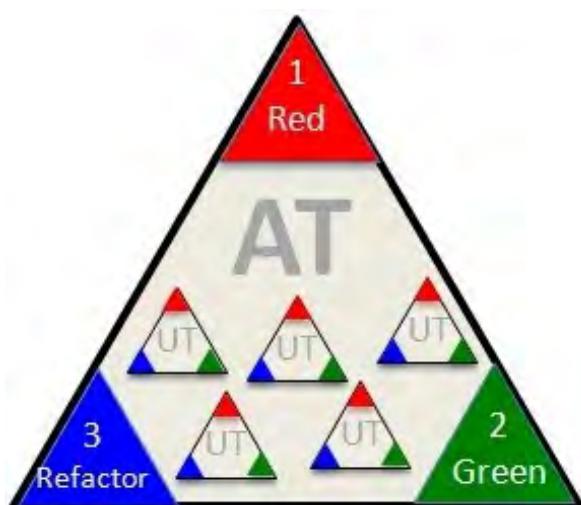
- [12种JavaScript MVC框架之比较](#)
- [Sails 0.8.9:深受Rails启发的实时Node MVC框架](#)
- [ASP.NET MVC 4 浮出水面](#)
- [大众点评资深前端工程师张颖：一站式的前端开发](#)
- [前端工程精粹（二）：静态资源管理与模板框架](#)

避开那些坑 | Void

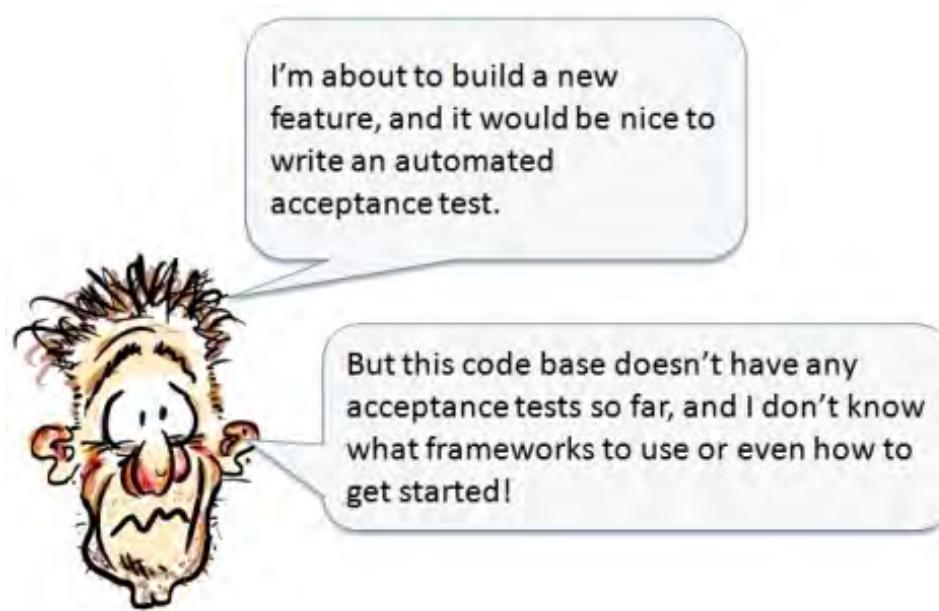
ATDD实战

验收测试驱动开发入门

作者 [Henrik Kniberg](#)，译者 张卫滨



你是否遇到过这样的场景：



那么本文就是为您而作——以一个具体的例子阐述了如何基于已有的代码库启用验收测试驱动开发 (acceptance-test driven development)。这是[应对技术债解决方案](#)的一部分。

这是带有一定缺陷的现实世界的样例，并不像教科书中的样例那样完美。所以完全是来自于实战。我只会使用Java与Junit，而没有使用其他的第三方测试框架（这些框架基本上已经被过度使用了）。

免责声明：我并不是说这就是正确地方式，关于ATDD有很多其他的“偏好（flavors）”。同时，本文中也没有太多新鲜的或革新性的内容，它只是一些确定的实践以及通过辛苦得来的经验。

我想做什么

几天前，我开始为webwhiteboard.com（我的小项目）添加密码保护的特性。很久以来，人们就希望有一种密码保护在线白板的方式，现在该实现这个功能了。

听起来这是一个很简单的特性，但是需要做很多的设计决策。到目前为止，webwhiteboard.com是基于匿名使用的，没有任何的账户、登录或密码这样的东西。什么人能够保护白板呢？谁能访问它呢？如果我忘记密码该怎么办？我们如何在足够安全的同时保证尽可能简单？

[webwhiteboard](http://webwhiteboard.com)的代码库有着很棒的单元测试和集成测试的覆盖率。但是它并没有验收测试，也就是站在用户的角度进行端对端流程的测试。

设计要素

[webwhiteboard](http://webwhiteboard.com)的主要设计目标是很简单的：尽可能简化登录、账户以及其他繁琐事情的需求。所以我为密码特性建立了两个设计限制：

- 对白板设置密码需要用户认证，但是访问密码保护的白板并不需要。也就是说，用户访问保护的白板需要输入密码，而不需要“登录”。
- 登录会使用第三方的OpenId/Oauth服务提供商，最初会使用Google。按照这种方式，用户就不需要再创建账号了。

实现方式

这里有很多尚未确定的事情：我并不确定它会如何工作，更不确定如何去实现它。因此，以下就是我的实现方式（基本的ATDD）：

- **步骤1：**在较高的层面编写预计的流程。
- **步骤2：**将其转化为可执行的验收测试。
- **步骤3：**执行验收测试，但是会失败。
- **步骤4：**使得验收测试成功执行。
- **步骤5：**清理代码。

这是一个迭代式的过程，所以每一步中我都可能会返回上一步进行调整（这是我经常做的事情）。

步骤1：编写预计的流程

假设这个特性已经完成了。在我睡觉的时候有个天使来了并实现了这个特性。这听起来美妙得难以令人置信！那我该如何对其进行检验呢？要进行手工测试的话，我首先要做什么呢？应该是：

1. 我创建一个新的白板。
2. 我对其设置一个密码。
3. Joe试图打开我的白板，被要求输入密码。
4. Joe输入错误的密码，被拒绝访问。
5. Joe再次尝试输入正确的密码，可以进行访问。（当然，“Joe”就是我自己，只不过使用另外一个浏览器……）。

当我编写完这个小的测试脚本后，我意识到要考虑很多可选的流程。但这就是主要的场景。如果我能够让它运行起来，就已经取得了很大的进展。

步骤2：将其转化为可执行的验收测试

这是一个需要技巧的步骤。我并没有其他端到端的验收测试，所以我该如何开始呢？这个特性会与第三方的认证系统（我最初的选择是使用Janrain）以及数据库交互，并且这里还有Web相关的内容，包括弹出对话框、令牌（token）以及重定向等等。

现在该退后一步。在解决“我该如何编写这个验收测试”之前，我首先要解决一个更为基本的问题，那就是“基于这个代码库，我到底该怎样编写验收测试呢？”

为了推进这个问题，我试图识别可以进行测试的“尽可能简单的特性”，这就是今天已经可用的一些特性。

步骤2.1 编写尽可能简单的可执行验收测试

以下就是我能够想到的测试步骤：

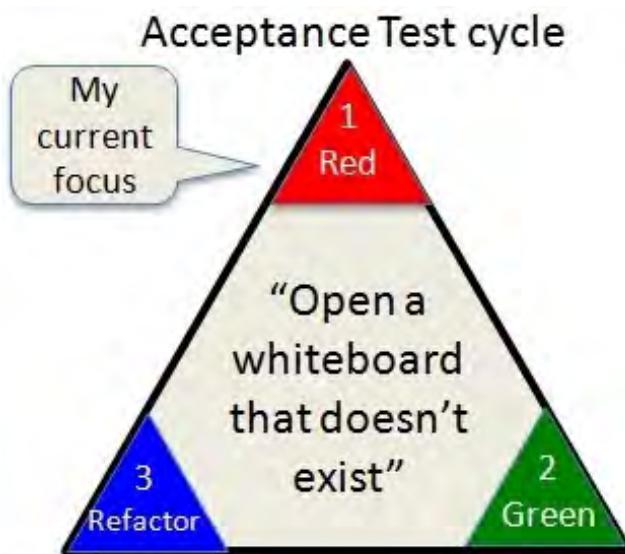
1. 试图打开一个不存在的白板
2. 检查确认无法得到白板

我该如何实现这个测试呢？使用什么框架？什么工具？它是否应该涉及到GUI，或者忽略它？是否涉及到客户端代码还是直接与服务器进行交流？

有很多的问题。技巧在于：不要回答这些问题！只要假设这些问题已经以某种方式很漂亮地解决了，并将测试编写为伪代码的形式。如下。

```
public class AcceptanceTest {
    @Test
    public void openWhiteboardThatDoesntExist() {
        //1. 试图打开一个不存在的白板
        //2. 检查确认无法得到白板
    }
}
```

我运行它，并且成功了！太棒了！但是稍等一下，这是错误的！在TDD三角（“红-绿-重构”）中的第一步是红色。所以，我需要让这个测试失败，以证明这是一个需要实现的特性。



我会编写一些真正的测试代码。不过，这些伪代码能够保证我的方向是正确的。

步骤2.2 将尽可能简单的验收测试变为红色

为了将其变成真正的测试，我构建了一个名为AcceptanceTestClient的类，我假装它已经魔法般地解决了所有的问题并且为我提供了漂亮的、高级的API来运行验收测试。它的使用很简单，如下：

```
client.openWhiteboard("xyz");
assertFalse(client.hasWhiteboard());
```

当我编写这些代码的时候，创建了一个API来适应这个测试用例的需求。它应该与伪代码的行数差不多。

接下来，我使用Eclipse的快捷键自动生成空白版本的AcceptanceTestClient以及我所需的方法：

```
public class AcceptanceTestClient {
    public void openWhiteboard(String string) {
```

```
// TODO Auto-generated method stub
}

public boolean hasWhiteboard() {
// TODO Auto-generated method stub
return false;
}
}
```

以下是完整的测试类：

```
public class AcceptanceTest {
AcceptanceTestClient client;

@Test
public void openWhiteboardThatDoesntExist() {
//1. 试图打开一个不存在的白板
client.openWhiteboard("xyz");

//2. 检查确认我无法得到白板
assertFalse(client.hasWhiteboard());
}
}
```

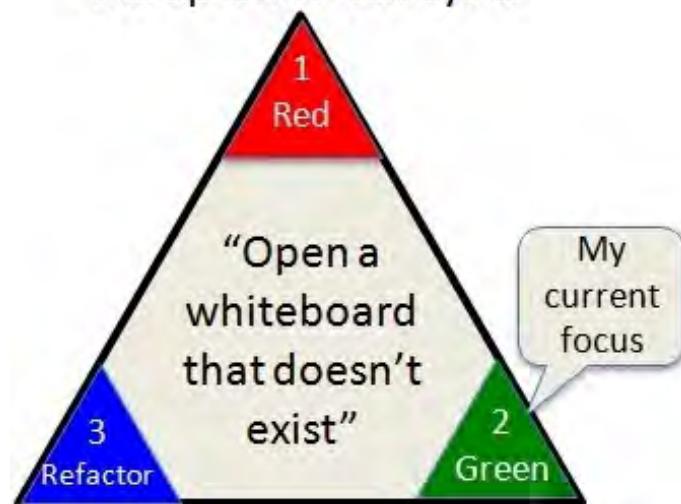
运行测试，但是它会失败（因为客户端为null）。很好！

我都做了些什么呢？其实没有太多。但这是一个起点。验收测试的帮助类AcceptanceTestClient已经有了雏形。

步骤2.3 将尽可能简单的验收测试变为绿色

下一步就是将这个验收测试变为绿色。

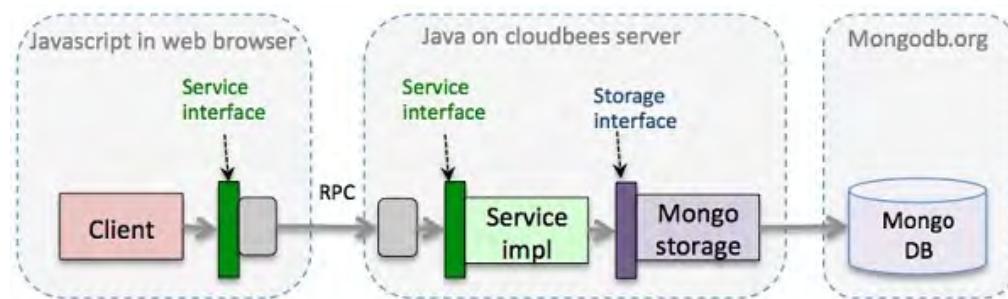
Acceptance Test cycle



我现在所要解决的是一个更为简单的问题。我不需要关心认证以及多用户等等的问题。稍后我可以为这些问题添加测试。

对于AcceptanceTestClient，实现很标准——模拟数据库（我已经有这样的代码了）并运行一个内存版本的完整的webwhiteboard系统。

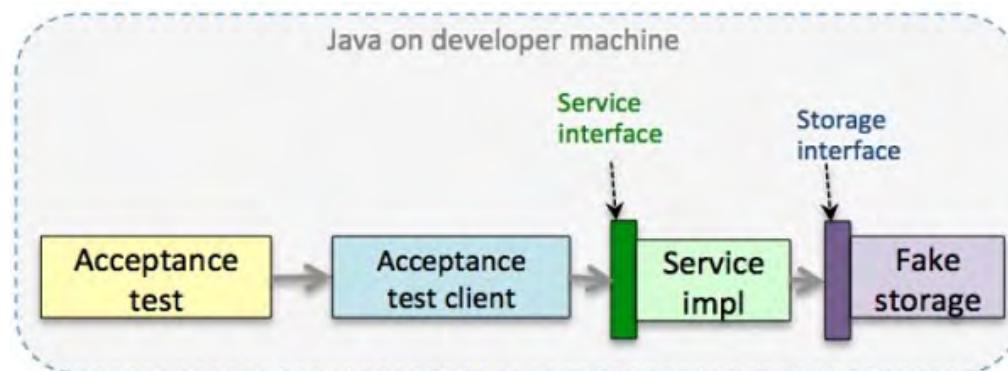
以下为生产环境的设置：



(点击图片放大)

技术细节：Web Whiteboard使用了GWT (Google Web Toolkit)。任何事情都是使用Java编写的，但是GWT会自动将客户端代码转换为JavaScript，并插入RPC (Remote Procedure Calls) 的逻辑，从而封装异步客户端-服务器通信的繁琐细节。

在验收测试的环境中，我对系统进行了“短路（short circuit）”，移除掉了所有的框架、第三方服务以及网络通信。



(点击图片放大)

我创建了AcceptanceTest客户端，它会与web whiteboard服务进行交互，就像真实的客户端一样。区别都是在幕后的：

- 真正的客户端与web whiteboard服务接口进行交互，它会运行在GWT环境之中，这个环境会将请求转换为RPC调用并转发到服务器端。
- 验收测试客户端也会与web whiteboard服务接口进行交互，但是它会直接连接到本地实现中，运行测试时，没有必要进行RPC调用，因此也没有必要使用GWT。

T。

同时，验收测试配置将mongo数据库（基于云的NoSQL数据库）替换为虚拟的内存数据库。

这种虚拟的原因在于简化环境、让测试运行得更快并且确保独立于框架和网络因素来测试业务逻辑。

看起来这似乎是一个很复杂的环境搭建过程，但实际上只是包含3行代码的初始化方法。

```
public class AcceptanceTest {
    AcceptanceTestClient client;

    @Before
    public void initClient() {
        WhiteboardStorage fakeStorage = new FakeWhiteboardStorage();
        WhiteboardService service = new WhiteboardServiceImpl(fakeStorage);
        client = new AcceptanceTestClient(service);
    }

    @Test
    public void openWhiteboardThatDoesntExist() {
        client.openWhiteboard("xyz");
        assertFalse(client.hasWhiteboard());
    }
}
```

WhiteboardServiceImpl是web whiteboard系统中已有的服务端实现。

注意AcceptanceTestClient的构造函数中接受一个WhiteboardService实例（这种模式称之为“依赖注入”）。这种方式给我们带来了一种便利：它不关心配置。相同的AcceptanceTestClient可以用于真实环境的测试，只需将真实配置的WhiteboardService实例传递给它即可。

```
public class AcceptanceTestClient {
    private final WhiteboardService service;
    private WhiteboardEnvelope envelope;

    public AcceptanceTestClient(WhiteboardService service) {
        this.service = service;
    }

    public void openWhiteboard(String whiteboardId) {
```

```

boolean createIfMissing = false;
this.envelope = service.getWhiteboard(whiteboardId, create
IfMissing);
}

public boolean hasWhiteboard() {
return envelope != null;
}
}

```

总而言之，AcceptanceTestClient模拟了真实的web whiteboard客户端所做的事情，同时又为验收测试提供了较高层次的API。

你可能想知道，“既然我们已经有了WhiteboardService可以进行直接交互，为什么还要AcceptanceTestClient呢？”。这里有两个原因：

1. WhiteboardService API是更为低层次的，而AcceptanceTestClient就是验收测试所需要的，并且能够使它尽可能地易读。
2. AcceptanceTestClient隐藏了测试代码不需要的内容，如WhiteboardEnvelope的概念、createIfMissing布尔值以及其他低层次的细节。在现实中，会涉及到更多的服务，如UserService和WhiteboardSyncService。

我不会向你过多地阐述AcceptanceTestClient的细节，因为本文不会探究web whiteboard的内部实现。简单来说，AcceptanceTestClient将与白板服务接口交互的低层次细节匹配到验收测试的需要上。这很容易实现，因为真正的客户端代码可以作为如何与服务进行交互的教程。

到此为止，我们尽可能简单的验收测试可以通过了！

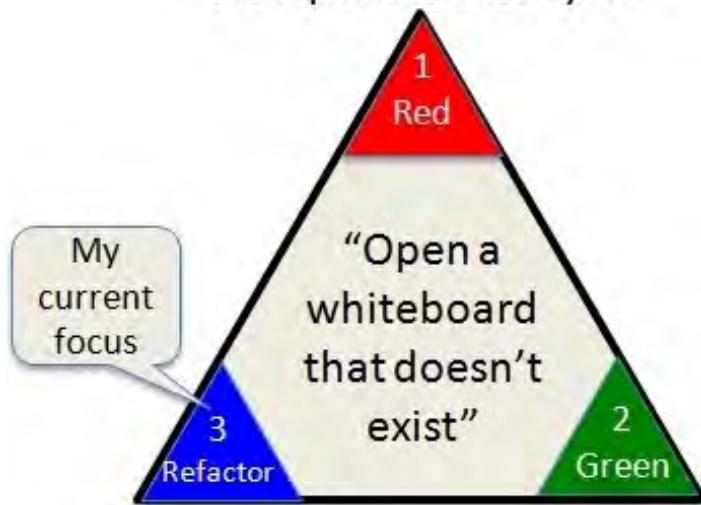
```

@Test
public void openWhiteboardThatDoesntExist() {
    myClient.openWhiteboard("xyz");
    assertFalse(myClient.hasWhiteboard());
}

```

下一步要进行一些清理。

Acceptance Test cycle



实际上，我并没有为此编写任何的生产环境代码（因为这些特性已经存在并且可用），这是测试框架的代码。我需要花几分钟的时间对其进行清理、移除重复内容、让方法名更为整洁等。

最后，我又添加了一个测试，只是为了完整性，而且它确实很简单。

```
@Test
public void createNewWhiteboard() {
    client.createNewWhiteboard();
    assertTrue(client.hasWhiteboard());
}
```

非常好，我们有了一个测试框架！我们甚至没有使用任何第三方的库，只是Java和Junit。

步骤2.4 为密码保护特性编写验收测试代码

现在，该对我的密码保护特性添加测试了。

首先，我将最初的“规范（spec）”复制为伪代码：

```
@Test
public void passwordProtect() {
    //1. 我创建一个新的白板。
    //2. 我对其设置一个密码。
    //3. Joe试图打开我的白板，被要求输入密码。
    //4. Joe输入错误的密码，被拒绝访问。
    //5. Joe再次尝试输入正确的密码，可以进行访问。
}
```

现在，我再次编写测试代码，假设AcceptanceTestClient已经具备了所有需要

的东西，并且完全按照我要求的方式，我发现这种技术是相当有用的。

```

@Test
public void passwordProtect() {
    //1. 我创建一个新的白板。
    myClient.createNewWhiteboard();
    String whiteboardId = myClient.getCurrentWhiteboardId();

    //2. 我对其设置一个密码。
    myClient.protectWhiteboard("bigsecret");

    //3. Joe试图打开我的白板，被要求输入密码。
    try {
        joesClient.openWhiteboard(whiteboardId);
        fail("Expected WhiteboardProtectedException");
    } catch (WhiteboardProtectedException err) {
        //Good
    }
    assertFalse(joesClient.hasWhiteboard());

    //4. Joe输入错误的密码，被拒绝访问。
    try {
        joesClient.openProtectedWhiteboard(whiteboardId, "wildgues
s");
        fail("Expected WhiteboardProtectedException");
    } catch (WhiteboardProtectedException err) {
        //Good
    }
    assertFalse(joesClient.hasWhiteboard());

    //5. Joe再次尝试输入正确的密码，可以进行访问。
    joesClient.openProtectedWhiteboard(whiteboardId, "bigsecre
t");
    assertTrue(joesClient.hasWhiteboard());
}

```

这个测试代码只需要几分钟就能编写完成，因为我可以在进一步编写代码的时候再将这些逻辑组织起来。这些方法在AcceptanceTestClient中几乎都（还）不存在。

当我编写这些代码的时候，我需要做出一些设计决策。不要费力去想，做第一时间进入你脑海的事情。完美是足够好的敌人，现在，我已经足够好了，也就是一个可运行的失败的测试用例。稍后，当运行测试变成绿色时，我再进行重构并进一步思考设计。

现在就进行重构是一件很有诱惑力的事情，尤其是重构这些丑陋的try/catch语句。但是TDD规约中有一点就是在进行重构之前要首先将其变成绿色，因为测试

会保护你的重构。所以我决定先暂时等待一下再进行清理。

步骤3 执行验收测试，但是会失败

按照测试三角，下一步要运行测试，但是会失败。

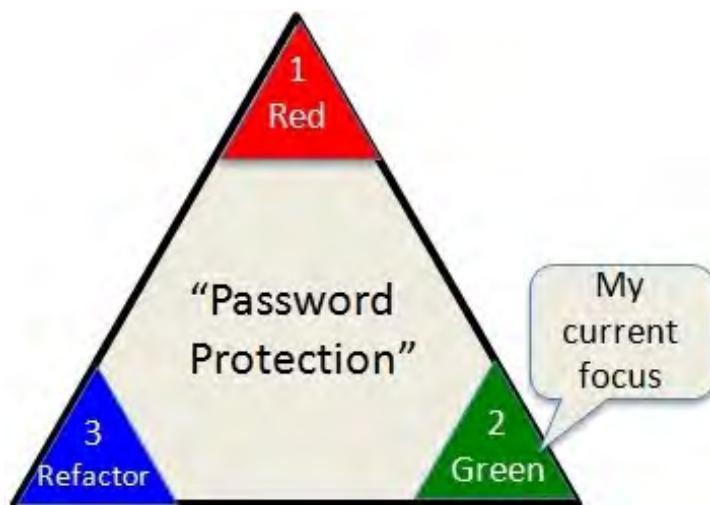


同样，我使用Eclipse快捷键来创建缺失方法的空白版本。很好！运行测试，看，出现了红色！

步骤4：将验收测试变为绿色

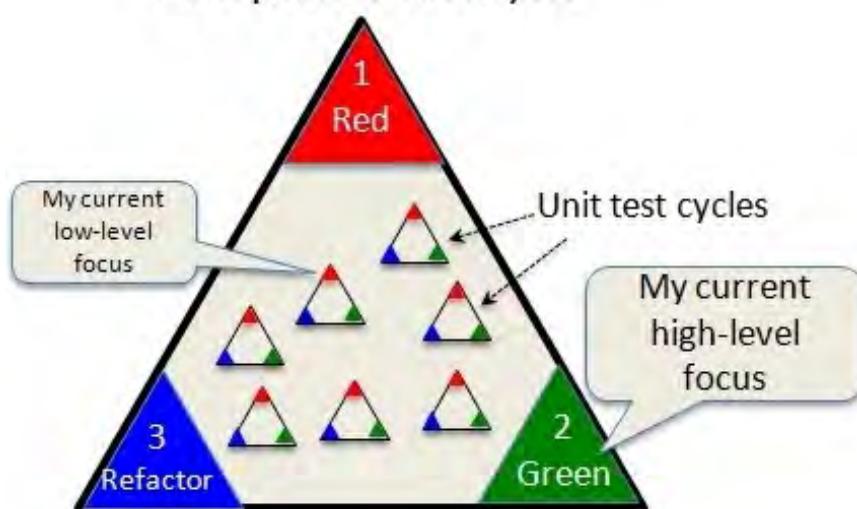
现在，我需要编写一些生产级别的代码。我为系统添加一些新的概念，有一些所添加的代码并不是试验性的，因此需要进行单元测试。我使用了TDD的方式，它与ATDD类似，但是范围更小一些。

以下展现了ATDD和TDD如何组合在一起。可以将ATDD视为外部的循环：



对于每个验收测试循环（在特性级别）的回路中，我们都会有很多单元测试的回路（在类和方法级别）。

Acceptance Test cycle

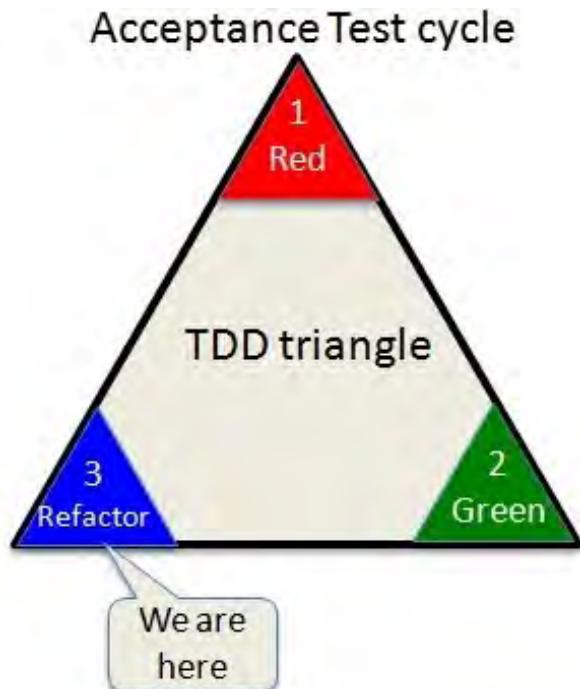


所以，尽管我在较高的层次上关注于将验收测试变为绿色（这可能会耗费几个小时的时间），但是在较低的层次上我可能会关注于将下一个单元测试变为红色（这可能只会耗费几分钟的时间）。

这并不是非常严格的TDD（Leather & Whip TDD）。这更像是“至少要保证单元测试与生产级别的代码是同时提交的”。这种提交每小时会发生多次，大致就可以将其称之为TDD了。

步骤5：清理代码

像通常那样，在验收测试变成绿色之后，就要进行清理工作了。不要试图越过这个步骤！就像在饭后清洗餐具一样——需要马上去做。



我不仅清理了生产环境中的代码，还清理了测试代码。例如，我将凌乱的try-catch部分抽取到一个帮助方法之中，从而最终实现了漂亮且整洁的测试方法：

```

@Test
public void passwordProtect() {
    myClient.createNewWhiteboard();
    String whiteboardId = myClient.getCurrentWhiteboardId();

    myClient.protectWhiteboard("bigsecret");

    assertCantOpenWhiteboard(joesClient, whiteboardId);

    assertCantOpenWhiteboard(joesClient, whiteboardId, "wildguess");

    joesClient.openProtectedWhiteboard(whiteboardId, "bigsecret");
    assertTrue(joesClient.hasWhiteboard());
}

```

我的目标是让验收测试尽可能简短、整洁并且易于使用，以至于注释都是多余的。最初的伪代码或注释会作为模板——“我希望代码就是如此得简洁！”。移除注释会给我一种成就感，它的一个积极作用就是让方法更加简短了。

下一步做什么？

重复地进行净化。在第一个测试用例通过之后，我就要开始思考缺失了什么。例如，密码保护应该还需要用户认证。所以，我为此添加一个测试、使其变红色、再变成绿色然后进行清理。诸如此类。

以下就是我（到目前为止）为该特性所添加的完整的测试：

- passwordProtectionRequiresAuthentication()
- protectWhiteboard
- passwordOwnerDoesntHaveToKnowThePassword
- changePassword
- removePassword
- whiteboardPasswordCanOnlyBeChangedByThePersonWhoSetIt

当发现缺陷或添加新特性时，我稍后肯定会展开新的测试。

我总共用了大约两天的时间进行高效地编码。在这个过程中，有很大一部分是回过头去重新编码和设计，并不像本文所展示那样线性进行。

那手工测试呢？

在自动化测试变成绿色后，我也会进行很多的手工测试。但鉴于自动化测试已经覆盖了基本的功能和很多边界场景，因此手工测试可以更多地关注主观性和探查性的内容。高水平的用户体验是什么样的？流程合理吗？它易于理解吗？我需要在什么地方添加帮助文本？按照美学，设计是否可接受？我不想去做争取什么设计大奖，但我也不想让它很丑陋。

强大的验收测试能够让我们不必再进行单调且重复性的手工测试（也被称为“搞怪测试monkey testing”），进而节省出时间来进行更有意思和更有价值的手工测试。

理想情况下，我应该在开始阶段就构建验收测试，所以一定程度上来讲这种方式是在偿还技术债。

关键点

就这样，我希望这个样例对你有用！它阐述了一种典型的场景——“我要实现新的特性，最好要编写验收测试，但是到目前为止还没有这样的测试，我不知道该使用什么框架，甚至不知道该如何开始”。

我非常喜欢这种模式，借助这种方式我多次走出了困境。总结如下：

1. 在便利的帮助类（在我的场景中也就是AcceptanceTestClient）背后假设封装了复杂的框架。
2. 为已经可以运行的特性编写非常简单的验收测试（如只是打开应用）。使用它来驱动你的AcceptanceTestClient实现以及相关的测试配置（如假的数据库连接和其他外部服务）。
3. 为新的特性编写验收测试。运行它，但是会失败。
4. 使其变成绿色。在编码的过程中，对所有非试验性的内容编写单元测试。
5. 重构。可能会额外编写更多的单元测试或移除多余的测试。保持代码的整洁！

完成这些后，你就已经越过了最困难的门槛，已经开始了ATDD！

关于作者



Henrik Kniberg是斯德哥尔摩[Crisp](#)的敏捷/精益教练，主要的工作内容是[Spotify](#)。他很乐意帮助公司在软件开发的技术和人力方面取得成功，就像他的图书“[Scrum and XP from the Trenches](#)”（本书中文版书名为《硝烟中的Scrum与XP》）、“[Kanban and Scrum, making the most of both](#)”以及“[Lean from the Trenches](#)”（本书中文版书名为《精益开发实战：用看板管理大型项目》）所描写的那样。

新品推荐 | Product

Intel 发布新版移动跨平台开发工具HTML 5 XDK

作者 [Sergio De Simone](#) , 译者 [廖煜嵘](#)

2013年12月，Intel宣布计划将其最新的XDK NEW变为正式的Intel XDK，以使其变为主流，并邀请所有的开发者在2014年2月月底之前从旧版本迁移到该版本。让我们看一下Intel XDK是什么以及在最新的版本中有什么新功能。

原文链接：<http://www.infoq.com/cn/news/2014/01/Intel-HTML5-XDK>

F#Tools 3.1.1增加对用于桌面和网络开发的Visual Studio 2013速成版的支持

作者 [Anand Narayanaswamy](#) , 译者 [李彬](#)

Visual F# Tools 3.1.1发布，将支持用于桌面和网络开发的Visual Studio 2013速成版，并能够直接从PowerShell命令行中安装。

原文链接：<http://www.infoq.com/cn/news/2014/01/visual-fsharp-tools-3-1-1>

12款免费与开源的NoSQL数据库介绍

作者 [张龙](#)

Naresh Kumar是位软件工程师与热情的博主，对于编程与新事物拥有极大的兴趣，非常乐于与其他开发者和程序员分享技术上的研究成果。近日，Naresh撰文谈到了12款知名的免费、开源NoSQL数据库，并对这些数据库的特点进行了分析。

原文链接：<http://www.infoq.com/cn/news/2014/01/12-free-and-open-source-nosql>

Math.js：多用途的JavaScript数学库

作者 [Roopesh Shenoy](#) , 译者 [马连浩](#)

Math.js是一款开源的JavaScript和Node.js数学库，用于处理数字、大数、复

数、单位和矩阵。它还有一个灵活的表达式解析器。为了解更多信息，InfoQ联系了该项目的创始人Jos De Jong。

原文链接：<http://www.infoq.com/cn/news/2014/01/mathjs>

Koa Web框架发布0.2.0版本

作者 [Burke Holland](#)，译者 夏雪

基于Koa的NodeJS web应用框架发布了0.2.0版本。Koa是广为流行的Express MVC平台的后续产品，但它在很大程度上依赖了ES6的新概念。这个版本的标号是重要的暗示，它重申了团队要从0.1.0开始发布的构想，本版本针对Koa的未来版本和产品用途充实了一些API。

原文链接：<http://www.infoq.com/cn/news/2014/01/koa-0.2.0-release>

使用Cordova 3.3.0在Android或iOS上部署Chrome应用

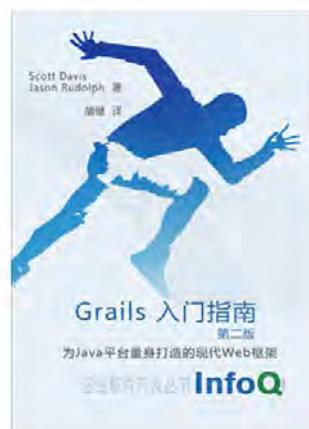
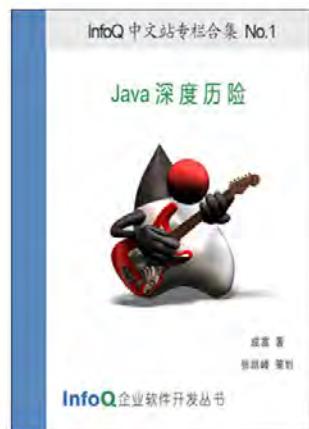
作者 [Abel Avram](#)，译者 李彬

开发者现在可以使用Apache Cordova 3.3.0在Android或iOS上部署Chrome应用。2.5.0和2.7.0将在近期被废止，而对黑莓、WebOS或塞班开发者来说，则建议改用2.9.0版本。

原文链接：<http://www.infoq.com/cn/news/2014/01/chrome-apps-android-ios-cordova>

InfoQ 软件开发丛书

欢迎免费下载



商务合作: sales@cn.infoq.com

读者反馈/内容提供: editors@cn.infoq.com

架构师

www.infoq.com/cn/architect

每月8号出版



封面植物

铃兰



铃兰（学名：Convallaria majalis），又名君影草、山谷百合，是铃兰属中唯一的种。原产北半球温带，欧、亚及北美洲和中国的东北、华北地区海拔850~2500处均有野生分布。也有以铃兰为名的日剧，一些动漫、游戏、轻小说中也有以铃兰为名的角色。铃兰落花在风中飞舞的样子就像下雪一样，因此铃兰的草原也被人们称为“银白色的天堂”。性喜凉爽湿润和半阴的环境，极耐寒，忌炎热干燥，生长在山坡阴面，林下，林缘的草地上。生长于山地阴湿地带之林下或林缘灌丛，有红色的果实和白色铃形的花朵。铃兰植株矮小，铃兰凉爽、湿润及半阴的环境。耐严寒、忌炎热干燥。喜肥沃排水良好的沙质壤土，夏季休眠。除了常见的白花外，变种有大花铃兰及红花铃兰。特别是大花铃兰，在四月间会从一对深绿色长椭圆形叶子上伸出弯曲优雅的花梗，绽开清香纯白的花朵；除了单瓣，还有重瓣铃兰的品种；有的园艺杂种呈现斑叶，称它为斑叶铃兰。

推荐编辑 | 吴海星



这篇文字本应该是做个自我介绍的。但回顾下来，从初入行时的编码捉虫到如今的遣词造句，虽然一直在业内，但所经所历均是随波逐流懵懵懂懂。实在平平淡淡乏善可陈，像是一个被工作牵引着的木偶，这种路人丙的故事讲出来既不足以励志，也构不成谈资，你应该不会感兴趣的。

不过既然承蒙InfoQ给这个机会，就卖一卖下我翻译的两本书吧。

一是《量化：大数据时代的企业管理》，副标题请直接忽视，扯上大数据完全是出版社出于营销上的考虑。量化这个词对应的是Metric，其实这个词一般译作指标，但从书中内容来看，measure指代的才是我们工作中常说的指标。而这个metric的内涵更像数据分析，又稍有差别，权衡再三，最终选定了量化一词。书中内容侧重于体系化地梳理企业业务，对构建数据仓库的业务域确实能起到一些启发作用。并一再强调做量化分析一定要有的放矢，明确根本问题，否则盲目地开展量化分析项目就像一场漫无目的的旅行，会死的很惨。书中还有很多生动有趣的小故事，各种寓言，看起来不会太枯燥。比如开篇第一章，就以经典的童话《三头小猪》为基础讲了一个庸医误人的故事，可怜的三头小猪虽然逃过了大灰狼的魔爪，却有两头小猪最终因为医生错用指标而双双毙命。这样的故事书中还有很多，非常适合为客户提供量化咨询服务。因此希望从事数据分析方面工作的各位同仁能拨冗一阅，不吝赐教。

二是即将付梓的《Node.js实战》，因为我自己曾长期从事Java web程序的开发工作，经受过各种web框架的折磨，所以在对node.js简单了解后就决定深入学习一下，也因此促成我接下了这本书的翻译工作。《Node.js实战》比较全面地涉及了node.js的整个技术体系：从Node.js的源起和编程范式入手，紧接着推出它的主战场web开发上的两员大将，Connect和Express，抽丝剥茧，娓娓道来，讲解地非常细致到位。最后又更进一步，开疆拓土。Node.js基于单进程，在充分利用硬件资源方面有天生的缺陷，因此先针对这个问题抛出了Node.js新推出的cluster API，表明Node.js靠自身实力挑战大型应用的态度和决心。接着又祭出Socket.IO这一让Node.js大受欢迎的法宝，顺势探讨了它在TCP/IP网络编程方面所做的努力，又介绍了它跟操作系统如何亲密接触，如何能够一路畅通无

阻。在前面这些内容中还穿插介绍了Node.js的编码、测试、部署和优化等开发流程。最后扫尾部分就是Node.js的外围体系了，甚至连GitHub都花了一节的篇幅来讲。所以可以说整本书对Node.js的讲解全面细致，适合用来系统地学习和了解Node.js。

不过就我目前浅薄的认识来看，node.js及相关框架要逊于用scala写成的play2，但一种技术体系能不能占据主导地位，起决定作用的因素很多，看IT业内的花开花落、云卷云舒也是个有趣的过程。

我的邮箱：wuhaiying@gmail.com，微博：@数据水墨。



高德地图
amap.com

LBS开放平台

高德地图街景API发布

- 22个城市街景数据免费开放，更多城市即将上线
- 丰富的组件接口，助开发者轻松构建360度街景应用

现在试用

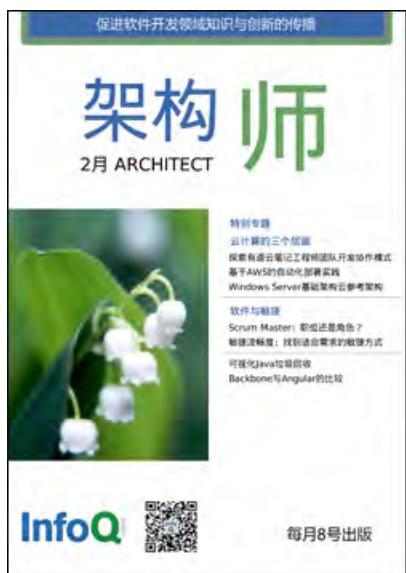


关注高德地图API

1kg 多背一公斤
.org

爱自然 | 更爱孩子





架构师 2 月刊

每月8日出刊

本期主编：杨赛

美术/流程编辑：水羽哲

总编辑：霍泰稳

发行人：霍泰稳

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

商务合作：sales@cn.infoq.com

InfoQ 中文站：[新浪微博](#)



本期主编：杨赛

杨赛 (@lazycai)，InfoQ高级策划编辑。写过一点Flash和前端，现在只是个伪码农。在51CTO创办了《Linux运维趋势》电子杂志，偶尔也自己折腾系统。曾混迹于英联邦国家，学过物理，做过一些游戏汉化，练过点长拳，玩过足球、篮球、羽毛球等各类运动和若干乐器。喜欢读《失控》。

InfoQ

《架构师》月刊由InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。