促进软件开发领域知识与创新的传播

架构师

4月 ARCHITECT

特别专题

动态语言更上一层楼

Rails与VoIP系统架构与部署实践

PyCon嘉宾谈Python

软件开发地基

JavaScript单元测试现状

一个前端工程师眼里的NodeJS

Cookie安全漫谈

大型网站复杂业务持续重构之道

Siri技术解析

集成Lucene和HBase

InfoQueue

Enterprise Software Development Community

每月8号出版

篇首语 | Editor’s letter

动态语言更上一层楼

动态语言与传统静态语言之争，以及动态语言部落内孰优孰劣之争，一直没有离开过人们的视线。抛开动态语言百花齐放的局面不谈，可以肯定的是，它们在不管是新兴的互联网应用还是传统的企业应用中，所占有的重要地位仍然呈上升态势。

JavaScript 已经开始成为编程语言世界的一等公民，可以用来构建富前端 UI 的第三方库鳞次栉比。更有 NodeJS 开始以 JavaScript 的语法构建服务器端高性能应用，为前端开发者提供后端视角。Ruby/Rails 开始被真实应用于高可用系统，一扫动态语言不能应用于传统大规模企业应用的疑虑。

从这篇专题介绍截稿期止，TIOBE2012 年 3 月份编程语言排行，动态语言占据前十名中四席，分别是 PHP（6）、JavaScript（8）、Python（9）、Perl（6），此外还有 Ruby（13）。

在现代软件系统开发周期中，动态语言已经涉及但不限于前后端开发测试、系统基础设施搭建、系统构建部署。在本期架构师特别专题中，编辑就为读者精心挑选了几篇涉及相关内容的文章，以飨读者。

郑晔的《软件开发地基》介绍了如何用 Ruby 环境搭建软件项目的 QuickStart，包括测试环境、构建环境、依赖管理和 IDE 集成，甚至还有代码检查、覆盖率统计，不一而足。

《一个前端工程师眼里的 NodeJS》，作者田永强用自己的经验和理解，为读者精心剖析采用 NodeJS 时需要面临的事件式编程、深层嵌套回调等诸多问题。

《大型 Rails 与 VoIP 系统架构和部署实践》则是一篇不可多得的 Rails 应用于大规模应用系统的实践和经验之谈，内容涉及业务分析、技术架构、监控分析等方面。

编辑相信读者通过这一期的《架构师》得以对动态语言的能力有一重新认识。

本期主编：张凯峰

目　录

[篇首语]

动态语言更上一层楼 1

[人物专访]

应用的未来——面孔、豌豆荚、HTML5 小组专家访谈 4

[热点新闻]

VISUAL STUDIO 11 新界面引发热议 10

ECLIPSECON 2012 大会：社区贡献获奖者，ECLIPSE 4 应用平台以及 ORION 1.0 12

.NET 4.5 任务并行库改动与指南 14

CLOUD FOUNDRY 云开发者大会：TUOCLUONG 谈盛大云应用现状 16

如何写缓存友好代码与 NOSQL 常见模式 19

[特别专题]

从简单到复杂：大型 RAILS 与 VOIP 系统架构与部署实践 22

虚拟座谈会：PYCON 嘉宾谈 PYTHON 30

软件开发地基 38

虚拟座谈：JAVASCRIPT 单元测试现状 52

一个前端工程师眼里的 NODEJS 56

[推荐文章]

COOKIE 安全漫谈 64

阅读者（十八）：编码的艺术 68

大型网站复杂业务持续重构之道——全程领域建模实践 72

SIRI 技术解析 79

集成 LUCENE 和 HBASE 86

[新品推荐]

淘宝开源分布式消息中间件 METAMORPHOSIS 97

SYMBIAN 为何在竞争中落后于 IOS 和 ANDROID？ 97

一款着眼性能的轻量级 DATA GRID——WPF TABLE VIEW 97

专访 HTML5 版 CLEAR 应用作者尤雨溪 98

ZK WEB 框架 6.0 发布：新型数据绑定系统 98

MONO FOR ANDROID UI 设计器介绍 98

使用 MONOTOUCH.SQLITE 简化用户界面开发 99

ADOBE 将在 PHONEGAP 品牌下发布 CORDOVA 99

ORM 工具 LLBLGEN PRO 3.5 发布 99

DJANGO 未来将支持 PYTHON3 100

WINDOWS POWERSHELL 脚本管理器 BETA 版发布 100

LINQ FOR F# 100

[推荐编辑]

推荐编辑 | 翻译团队编辑 郭雪品 101

[封面植物]

天山雪莲 102

InfoQ

促进软件开发领域知识与创新的传播

中文 | 英文 | 日文 | 葡文 | ……

内容投稿 editors@cn.infoq.com | 商务合作 sales@cn.infoq.com | 加入我们 hr@cn.infoq.com

人物专访 | Interview

应用的未来——面孔、豌豆荚、HTML5 小组专家访谈

为了让大家能够了解到不同平台的产品特质以及现状和发展趋势，我们特别邀请到移动 App 平台的@面孔网杨锴、PC 平台的@豌豆实验室张涛和姜楠以及@HTML5 研究小组的黄蔚瀚，他们将分别以各自独特的视角，来审视和解读未来时代的 Apps。

InfoQ：请介绍一下您自己以及您所负责的产品？

杨锴：我是杨锴，我们面孔目前的主要项目是 HTML5 开发工具：Julia。Julia 是一款基于 HTML5 的开源移动应用开发框架。通过 Julia，只要具备网页工程师的基础知识（例如，HTML+CSS+JavaScript），就能否开发出足以媲美本机代码（Native APP）的移动应用。

张涛：我是张涛，@hidecloud，曾任职于 cnBeta.com 及腾讯。目前负责豌豆荚应用搜索产品的设计工作。豌豆荚应用搜索是一款很有趣的产品。一开始豌豆荚只和应用汇一家市场有合作，后来随着接入的市场越来越多，用户开始有了抱怨“每次找应用都要去不同市场，还不能保证这个市场就是最新的版本”。于是我们发掘了这个需求，并开始整合与我们合作的市场应用资源，来做这么一个手机应用领域的垂直搜索引擎。目前我们已经与 21 家应用商店合作，为用户提供不同版本的超过 50 万款应用进行下载。

姜楠：我是姜楠，豌豆荚应用导航产品设计师，目前我负责的主要是豌豆荚应用搜索产品中所以浏览型页面的设计工作。

黄蔚瀚：大家好，我是来自于 HTML5 小组的黄蔚瀚，目前我负责华彩彩票客户端产品设计以及后期的产品优化的工作。

InfoQ：您认为好的应用应该具备哪些特质，请列举三点并说明一下理由。

杨锴：开发者自己有需求长期使用：只有把自己当成用户，你才会发现真实有效的潜在需求。

容易维护和升级：我们见过很多应用开发因为初期规划混乱，或开发团队不稳定，造成

1-2 个版本更新后就夭折了。

性感：如果在理性需求之外，让用户在感情上对你得产品产生认同，那么这个产品的用户忠诚度和扩散性都会很好。做到这点不容易，但是我们也能看到一些产品实现了，比如 Mini cooper，糗事百科，豆瓣等。

张涛：可用性：手机上的操作空间相比桌面来说小很多，界面元素需要保证用户在各种条件下（斜视屏幕、手指遮盖、阳光直射……）可见或者说是可预见。这要求设计师对自己所设计应用的使用场景有深入的认识，确保用户在整个使用过程中无痛。

专注：我一直很强调“应用”与“软件”的区别，手机应用就应该专注于某项功能，为用户解决某个特定的问题。而不是像桌面软件一样，提供大而全的功能覆盖。手机应用要求启动快，解决问题快，帮助用户快速进入快速退出。应用，应该只是用户处理问题的一个工具而已，不应该成为浪费时间和精力的平台。

长期维护：一款有开发者长期维护的应用，总给人一种安心感，更容易让人投入到其中。特别是某些需要开发者长期维护才能很好使用的服务，更新频繁程度、功能开发进度，都很影响用户的最终选择。

姜楠：好的应用站在不同的角度答案也许会发生不小的变化，比如开发者、市场、用户给出的答案都是不一的，我以用户的角度认为好的应用至少应该包含：

易用性：这里也包含了典型的应用场景、涵盖更广的机型、轻量化的体积。

精致性：这里主要指的是应用设计层面的精美、细致，即要有精美的 UI，也要有对细节处不断的打磨。

风格性：一款好的应用一定要有自己的风格，或受众人群、或设计风格

黄蔚瀚：好的应用要做到“好看，好用，有生命力”

好看主要体现在用户第一眼看到应用的印象，以及使用过程中对应用整体和细节的印象，这里需要设计的强大力量。有许多应用由于图标没有吸引力，用户下载后可能就再没打开过，或者根本就没有打开它的欲望，这对应用本身是很不利的。

在可用的情况下，好用主要针对整体操作以及合理性，这里有两点，一点是对应用内部数据类型和数据量的编排设计。二是对操作这些数据的实现方式。

有生命力，主要与产品周期以及服务周期，让用户知道这个产品不是 3 天或者 5 分钟就可以删除的，这里需要结合商业计划进行分析。我们的客户端与现实生活挂钩，所以，

我们会把它看做一种基于娱乐的消费工具，我们希望用户习惯性的打开，看看就可以关闭了，希望操作在 1~2 分钟内完成。产品的生命力和本身的使用周期以及业务逻辑周期有关。

InfoQ：在设计、开发、运营中，最让您头疼的是什么，请列举三项。

杨锴：

- 设计师和程序员的沟通。
- 前端工程师和后端工程师的沟通。
- 程序的可维护性和复用性。

这三点也是我们开发 Julia 的一部分原因。

张涛：

- 设计：对用户需求的揣测。
- 开发：工程师对设计方案的理解。
- 运维：国内没有专业的网络支撑服务，比如 Amazon 这样的。所以需要投入很大的力量来保证线上服务稳定。

姜楠：在产品成熟前，摸着石头过河的感觉，需要不断的尝试，在产品 launch 后，不断调整，过程比较痛苦。

有时需要不断地对开发人员去讲解，产品这样设计的原因、过程及目的。

黄蔚瀚：设计—>开发—>运营

需要弄清 UI 与 GUI 的关系，有好的结构 UI 才可以让设计人员制作出好的 GUI 出来，这里可以减少未来的重复劳动量；设计是需要时间来消化的，而且需要接受一个客观的时间段来批判，对于设计来说，我们最担心就是遇到审美疲劳的问题，和”看起来不专业”的问题。

开发需要跟着策划走，因为在实现过程中往往发现有的‘可以做，但是暂时做不了’，所以需要与策划紧密沟通来作出临时解决方案，由于技术开发与策划有明显逻辑不同，所以策划对技术需求描述也是一个难点。

运营针对用户的多样性，以及服务的复杂度提升，宣传渠道的变化这里主要体现资金问

题。

InfoQ：您最看好的技术或工具或理念是什么，请列举三点。

杨锴：

- 简洁有效。
- 前端、后端在语言和开发方式上的统一。
- Open Web App。

张涛：

SaaS：虽然被炒烂了，但随着带宽的提升（最近全国都在提速嘛），软件和应用这个概念最终将会模糊掉，用户在意的是数据，和数据被处理的方式。至于这个应用到底运行在本机还是云端，用户终将不会在意。

iPad：虽然这几年 iPad 的狂潮已经席卷了全球，但其所带来的影响还没有真正地显现出来。媒介既信息，iPad 所带来的将是对下一代人阅读及交流习惯的颠覆式改变。

姜楠：

个性化：目前号称自己做个性化的产品多如牛毛，但实际并没有哪一个技术或工具可以完美的让用户有这种感知。

关系+兴趣社交：目前关系社交鼻祖 Facebook，兴趣社交 twitter，两都都在各自领域做到极致，当然并不是说 Facebook 只有靠关系，或 Twitter 只靠兴趣，他们也有向对方的领域进军，不过兴趣和关系本身并不矛盾，而将两者都完美结合的案例，个人觉得还没出现。

黄蔚瀚：我们常说的“技术不是问题”是需要调整这个说法的。因为，技术往往受到了时间的约束，时间则受到市场限制。做不做得了不是技术实力的问题，而主要是时间的问题，如果时间不够，那好产品是出不来的。所以，对理念的具象化需要有“现实的”思考，而不是做大而是做远。这里需要思考产品周期，以及每个版本可以实现的东西，千万不能急着让所有东西都在一个时间段内出来。

InfoQ：您认为应用未来的发展方向是什么？

杨锴：

- 有偿云服务。
- Open Web App。
- 永远在线

张涛：更轻量化，摆脱对单一平台或者使用场景的限制，可以在不同的设备和场景中自由平滑切换。用户关心的是“自己”和“数据”，不需再为应用本身而操心。 姜楠：

个人娱乐化，这点在 iPhone 平台上可窥见一斑，但在 Android 平台上才刚刚起步，基于个人兴趣的娱乐化服务方式将是以后发展的趋势。

Cloud：云端的服务从兴起到如今的“蓬勃发展”，但乃远未到其可以较好的服务用户，服务分散化，局限性，操作的复杂度，空间大小等等在这些都解决后，将会迎来其更为人性化的浪潮。

黄蔚瀚：未来应用发展方向会持续的两级分化，会有一部分大型应用开始把渠道走进电视屏幕。

我觉得小型应用会开始大量使用云端，而且会快熟进入一个成熟期；大型应用会基于企业级的大型服务器做服务；中型应用会很快进入一个瓶颈，生存会非常困难。

用户体验会渐渐被复原为用户中心设计，因为产品的发展趋势都是‘重体验，轻功能’。因为，好的产品强调的是用户中心设计。对于用户体验分析，则需要大量使用数据反馈才可以得到相关优化依据。

由于我们的产品专业性强，以及涉及渠道包括网站，手机，数字电视，终端机等平台。所以产品方向会继续注重安全性，以及挖掘不同平台特征来满足用户的不同需要。

InfoQ：在 2012 年，您所负责的产品会有哪些改变和升级？

杨锴：我们会在 2012 年推出 Julia 的 1.0 和 2.0 两个版本，并推出 JS++的规范。通过对 Julia 和 JS++两个项目的改进，预计在 2012 年第三季度实现如下两个产品目标：

只写一套代码，实现跨平台运行。

用 HTML5 开发，自动编译为二进制代码。

前端开发直接声明数据，后端自动生成数据库。

> **张涛**：豌豆荚应用搜索在实现了基础的应用市场整合后，将会在为用户提供更精准搜索结果这点持续优化。结合用户使用习惯，用户推荐内容等信息，帮助用户更好更方便地寻找发掘应用。

> **姜楠**：
>
> - 推荐数据与云端的无缝整合。
> - 个性化的推荐方式。
> - 导向性更强的导航系统。

> **黄蔚瀚**：今年我手上的产品主要会进入深度优化，由于我们的功能已经比同类产品完善得多了。所以在接下来，会把核心在功能流程上进行改良。

原文链接：http://www.infoq.com/cn/articles/app-future-html5-mobile

相关内容：

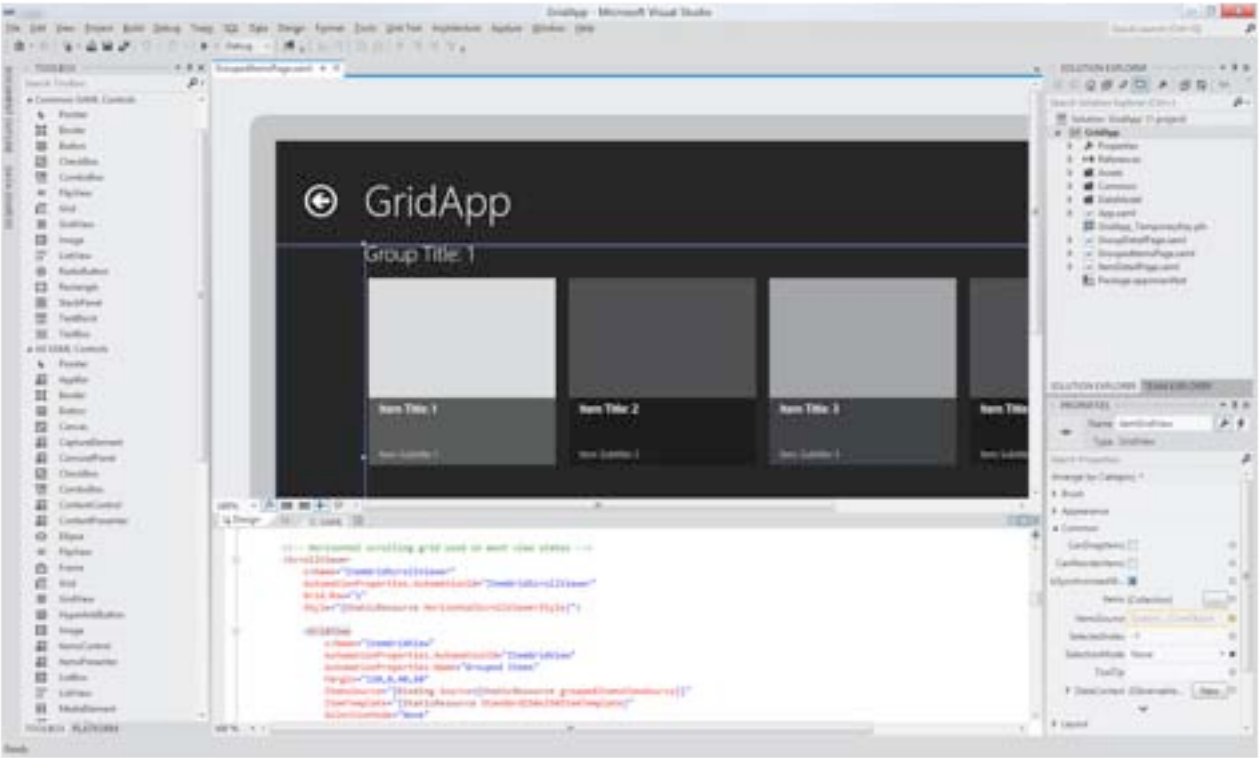
- 基于 HTML5 技术跨平台混合式应用开发解决方案 UniSDP
- 2012.3.7 微博热报：GitHub 被黑与 HTML5 的文件处理 API
- 快，快，快——互联网产品用户体验第一军规
- 何崚谈阿里巴巴前端性能优化最佳实践
- 富互联网应用之美

热点新闻 | News

Visual Studio 11 新界面引发热议

作者 Jeff Martin 译者 曹如进

2 月 29 日微软发布了 Visual Studio 11 beta。新版本的 Visual Studio 不仅拥有一个全新的基于灰度调色板的用户界面，还对工具栏图标进行了重新设计。新版界面一出，迅速引发了激烈的讨论。



微软开发工具事业部用户体验总监 Monty Hammontree 在博客里发布了两篇介绍新版本开发体验的文章，见这里和这里。新版 VS11 的样式与 VS2010 相比相似度很小，它既包含了 Windows 8 的 Metro 界面元素，也融入了微软 Office Ribbon 工具栏风格。Hammontree 解释说，他的团队认为影响开发人员效率的问题主要有三个：

- 过于繁多的工具；
- 代码库较为复杂时，较难理解和定位较为困难；

- 当文件数量很多时，不易处理。

这些问题并不是凭空想象，它们是建立于微软内部和微软外部的研究成果之上。有意思的是，Hammontre 还引用了某篇研究论文中的 Elicpse 例子来解释他的团队正在试图解决的问题。虽然调色板和图形界面设计的改动只是他们团队设计方案中的一部分，但这一部分却又是开发社区中最引发争议的话题。Hammontree 解释说他们之所以减少色彩，其背后用意在于：

随着开发人员目光转向 Metro 风格的客户端，如 Xbox、Windows Phone7 和 Windows8，将应用程序内容颜色至于中央变得越来越重要。当开发人员在上述平台设立用户体验时，会用到更醒目、更有活力的颜色。而将这些颜色放在一个单色工具设置中，会显得更加有效。

尽管新界面背后有研究支持，而且 Hammontree 也进行了一番善意的解释，但是开发人员还是纷纷表达了他们个人对此次变动的看法。到目前为止，Visual Studio 博客上有超过 700 名的开发人员参与了该话题的讨论。另外，在微软的用户心声网站上，我们也能看到一个关于 Visual Studio 色彩改动的投票话题，目前当前该话题热度排名第一。

微软号召开发人员继续提供反馈，并声明他们会阅读所有的留言，不论留言内容是褒扬还是批评。

原文链接：http://www.infoq.com/cn/news/2012/03/vs11goesgrey

相关内容：

- Web Workbench 添加 Visual Studio 11 支持
- 来自 Redmond 的真实声音
- 深入解析 Visual Studio 2010 TFS
- 潘正磊谈 Visual Studio 开发过程中的敏捷实践
- Visual Studio .NET 使用技巧手

热点新闻 | News

EclipseCon 2012 大会：社区贡献获奖者，Eclipse 4 应用平台以及 Orion 1.0

作者 Charles Humble 译者 吴宇

EclipseCon 2012 大会开幕当天，Eclipse 基金会宣布了年度 Eclipse 社区贡献获奖者名单。该奖励是对 Eclipse 生态体系中的优秀个人、项目和技术的认可。

个人单项和项目类的获奖者通过在线投票选出。个人获奖者包括 Sven Efftinge，他是 Eclipse Xtext 和 Xtend 语言的负责人和架构师，Stephan Herrmann，Eclipse Object Teams 项目的领导者，以及，非常高兴的向大家报道，我们 InfoQ 的 Alex Blewitt。

Eclipse Communication Framework 赢得了最佳开源项目大奖，而 Eclipse Code Recommenders 赢得创新项目奖。

技术贡献奖由专家评委会选出，获奖名单中包括 JVM 调试器 Chronon、建模/反向工程工具 MaintainJ 和 Justinmind Prototyper。 除此之外，此次大会还推出了 Eclipse 4 应用平台，它将会成为今年发布的 Eclipse Juno 的基础。Eclipse 4 大量的重写了 Eclipse UI 的基础架构，这样可以使用基于 UI 的模块，而不是（像以前版本那样）通过编码的方式来开发用户界面。这些模块能够被查询、操作和扩展，给快速设计和用户界面定制化提供了便捷。Eclipse 4 还提供了对 web 风格技术（CSS）的支持，这使得用户界面元素可以被尽可能的修改和重配置，而无需更改任何应用代码。另外，改进后的平台增加了面向服务的编程模型，从而可以更方便的利用 Eclipse 平台提供的应用服务。

在 Juno 发布不久后我们可以期待看到 Orion 1.0 版本的发布，这是一款 Eclipse 基金会提供的基于浏览器的集成开发环境（IDE），用于开发 Web 应用。在去年 EclipseCon 大会上就宣布过，本次大会有关于该产品的几个研讨会。Orion 是面向 HTML 和 Javascript 开发的，而 Mozilla 已经在 Firefox 开发者工具使用了 Orion 编辑器。

EclipseCon 2012 大会上的其他主题还包括 OSGi、通过 Eclipse 工具开发 Android 应用、应用生命周期管理，另外还有一些研讨会展示了 Eclipse 工具如何应用于 JavaFX 以及

JavaFX 与 SWT 和 Swing 的互操作。

Eclipse 于去年十一月份刚度过其 10 岁生日。你可以在这里了解到更多关于该项目的历史。

原文链接：http://www.infoq.com//cn/news/2012/03/eclipsecon-awards

相关内容：

- AIDE——运行于 Android 上的 Android Java IDE
- 模块化 Java 简介
- 企业级软件的组件化和动态化开发实践
- GitHub 新功能，可以在浏览器中直接修改文件和提交
- Eclipse 发布 OrionHub 开放 Beta 版

热点新闻 | News

.Net 4.5 任务并行库改动与指南

作者 Jonathan Allen 译者 曹如进

在.NET 4.0 中，Task 类暴露了 IDisposable 接口。Task 可被回收（disposable）是为了清理 IAsyncResult 接口中 AsyncWaitHandle 属性暴露的等待句柄（wait handle）。在 .NET 4.0 中，等待句柄只有在读取 AsyncWaitHandle 属性，或者使用 Task.WaitAll、Task.WaitAny 时才会被创建，其他情况调用 Task.Dispose 都是多余的。

遗憾的是，.NET 4.0 中的 Task 在处理 ObjectDisposedException 时显得过于武断：一旦调用 Dispose 释放等待句柄之后，即使其他属性与之毫无联系，剩余对象也会变得不稳定。

那么在.NET 4.0 中是否应该调用 Task.Dispose？

不应该，除非遇到以下情况：

1. 整个 Task 不会被缓存；
2. 等待句柄是通过调用 Task.WaitAll、Task.WaitAny，或是读取 IAsyncResult.AsyncWaitHandle 创建而成；
3. Task 上没有其他任务或线程处于等待状态。

其实，即使所有的条件都满足，你也不用做什么，因为终结器

（finalizer）在清理等待句柄方面已经做了相同高效的工作。所以，除非你看到一些性能问题，否则你也许可以仍然不用回收 task。

.NET 4.5 核心中的改动

在.NET 4.5 中，只有显式读取 IAsyncResult.AsyncWaitHandle 时，内部等待句柄才会被创建。其他部分，包括 Task.WaitAll 和 Task.WaitAny 都进行了重新设计，它们不再需要等待句柄。另外，随着语言中对 async/await 的支持，IAsyncResult 在大部分场景中甚至都不再需要。

.NET 4.5 中 Task 的另外一个改动是 task 在释放之后仍然可用。按照 Stephen Toub 的说

法，“现在，即使 Task 释放之后也可以使用它的所有公开成员，并且它们使用起来就和释放之前一样。唯一一个不能使用的成员是 IAsyncResult.AsyncWaitHandle ,因为它是 Task 实例真正释放的部分。如果试图在 Task 释放后访问该属性，它会抛出一个 ObjectDisposedException 异常。”

虽然在.NET 4.5 中调用 Task.Dispose 变得更加安全，但是几乎没有理由需要这么做。

针对.NET 4.5 Metro 的特殊规则

Stephen Toub 接着提到 Task.Dispose 在“.NET Metro 风格应用程序”框架中甚至并不存在。要注意的是，目前关于此项设计变更的信息还未在 WinRT 的 Task 文档中更新反映。

从函数返回 Task

在另外一篇题为“是否应当为同步方法暴露异步包装？“的文章中，Stephen 深度探讨了从函数返回 Task 对象的话题。我们推荐你阅读全篇文章，而如果你时间不充裕，可以阅读以下的总结部分：

> 我认为只有那些异步方法比对应的同步方法拥有可扩展性（scalability）优势时才应当被暴露。异步方法不应当为了单纯地减轻负载（offloading）而进行暴露，因为这类优势可以通过使用专门用于异步执行同步方法的功能轻松实现，如使用 Task.Run。

原文链接：http://www.infoq.com/cn/news/2012/03/Task-Dispose

相关内容：

- Zlatko Michailov 再谈任务并行库数据流（TPL Data Flow）
- 实现自定义的任务并行库（TPL）数据流 Block
- .NET 4.5 中任务并行类库的改进
- Silverlight 的任务并行库
- 微软产品市场经理谈 Visual Studio 11 的开放特性

热点新闻 | News

Cloud Foundry 云开发者大会：TuocLuong 谈盛大云应用现状

作者 贾国清

3 月 28 日由@VMware 中国 主办，在北京喜来登饭店举行的 Cloud Foundry 云开发者大会上，盛大云计算董事长，盛大在线 CEO，美国创新院院长 Tuoc Luong 先生，就有关盛大云的相关问题进行了解答，以下是问答的详细信息：

1.目前盛大云的用户数量大约是多少？

> Tuoc Luong：盛大云注册用户数在万级、付费用户有数千人。

2.盛大云主要是通过 IaaS 平台和 Cloud Foundry 的 Paas 平台来合作，但盛大自己也有一部分 PaaS，有何区别？

> Tuoc Luong：Cloud Foundry 的 PaaS 只是和盛大云 IaaS 合作的其中一种 PaaS，盛大也有自己的 PaaS，例如游戏的道具验证、盛大通行证、付费方式等垂直 PaaS，Cloud Foundry 相对平台化一些，这种平台架构对中国创业者来说，是一种便利。我们此前通过测试，发现两者非常匹配。未来几个月，我们会确定双方的合作，届时会向公众公布。

3.盛大云涨价的事情，有部分用户反应比较大，面对这些站长的反应，盛大后续会有什么措施？

> Tuoc Luong：盛大云近期是从原来的按流量计费改为按带宽计费，对一部分微小型用户带来了比较大的影响，面对这些用户，盛大云也推出了一系列的让利措施，例如 15 天免费期、免费期结束后再享受 2M 带宽买多少送多少的措施，让这些用户平稳过渡到这种计费模式上来。未来盛大云还会有更多的运营措施，帮助中小企业提供服务。

4.盛大云为何要改成这样的计费模式？

Tuoc Luong：这次调整是基于多次用户市场调研结果，绝大多数国内用户普遍具有带宽付费经验，这种计费方式有助于用户掌控业务发展所需要的预算，防止异常流量带来的费用损失，避免成本的难以预估和控制。

5.由于中国的小企业生存环境比较艰难，盛大云对创新型小企业有什么支持？

Tuoc Luong：小企业是盛大云的目标客户之一，盛大云为他们提供了适合于这些小企业使用的完全自助化的云计算解决方案帮助他们节省时间、架构、运维等方面的成本。

6.除了基本的数据安全方面的措施，由于盛大自己也是互联网企业，假设一个大的企业把服务器假设在盛大云上，盛大云如何保证不会使用这些数据，保证这家企业的数据安全？

Tuoc Luong：盛大云的业务专注于基础架构的建设，在整合盛大集团资源的基础上，百分百自主技术研发而成的公有云平台。盛大集团也有很多的内容，例如游戏、文学，这些业务公司也同样存在数据安全方面的考虑。盛大云绝不会开发与用户类似的 APP 应用参与竞争，只是希望最终为用户提供更好的云产品和服务。

7.盛大如何做好服务内部和外部公司的资源平衡？

Tuoc Luong：服务于内部和外部公司公司对于盛大云来说是不矛盾的，没有严格界定对内和对外服务的资源分配。会从市场应用的角度出发来考虑，也会区分不同的产品，比如云分发偏向于对外，而虚拟化产品则首先应用于内部。总之都是为用户提供好服务，盛大旗下其他的业务公司，也都是属于我们的客户，同样需要价格结算等。

8.盛大云如何和国内同类的云计算企业去竞争？

Tuoc Luong：云计算在中国国内还处于起步阶段，去年推出了盛大云，并且独立成立了云计算公司，恰恰反映了盛大网络发展云计算的决心和信心很强，我们会不断的提升自己的产品和服务，来满足市场需求。

9.作为合作伙伴，VMware 的哪些产品和服务能够与盛大云合作，帮助双方取得更好的成绩？

Tuc Luong：VMware 和盛大云在产品和服务上是非常匹配的，盛大云偏向于底层架构 IaaS 平台，VMware 属于上层业务 PaaS 平台，双方合作之后，盛大云逐渐从底层结构发展至上层，产品和服务走向更加完整的方向。

10.Tuoc 先生您也曾就职于几家国外公司，现在服务于盛大，感受国外和中国的工作有何不同？

> Tuoc Luong：这个问题很好，说起我的感受，可能得谈一天。简单来说，在中国，由于中国的市场特别大，所以国内的做法往往是先去抢市场，再回来想创新。国外公司的做法是现有创新，再创造市场，引导市场。这一点是最大的感受。

原文链接：http://www.infoq.com/cn/news/2012/03/CF2012-interview-tuoc

相关内容：

- 赵钟秋谈又拍网架构中的消息/任务系统
- VMware 的 CloudFoundry 服务获得对 PostgreSQL 支持
- 微软宣布 Team Foundataion Service 新版云端生成服务
- 迁移 Channel9.msdn.com 到 Windows Azure
- 专访 Zynga CTO Cadir Lee，谈谈 Zynga 的云计算与核心价值观

热点新闻 | News

如何写缓存友好代码与 NoSQL 常见模式

作者 郑柯

如何编写缓存友好代码

@leekayak 发布了一条微博：

> 对缓存的思考【续】——编写高速缓存友好代码 http://t.cn/zO2Lx9I

fengyuncrawl 的评论是：

> 文章写得非常细，这里描述的缓存是 cpu 级别并不是数据库页缓存（主存）。

欧蓬 AT 移动互联网提到：

> 作者说的很好，对硬件体系结构的理解的好，才能写出足够高效的底层算法。光看算法和程序的优美，是没有用的。

@张博韬_evans:

> 对 Cache 友好基本要考虑两点：1）尽量把关联的数据空间上排在同一个 cache line 里，这样可以共同被换进换出；2）使用 prefetch 把空间上不连续但逻辑上即将要被访问的数据提前换入 cache，减少 cache miss 带来的损失。话说回来，现在研究这个的程序员真是少...

王津 THU 回复@张博韬_evans：

> Cache 的 prefetch 算法，是永恒的话题，是超出纯技术而要结合业务的话题，其本质是人类的根本问题之一，即对于未来的预测。

张博韬_evans 回复@王津 THU：

> 我只是说基本理论，你说的没错，prefetch 用不好很容易降低性能。要具体结合 cpu 的体系架构和应用场景来分析。好在软件开发，不全是靠预测的，预测不准的东西可以模拟环境来测试:)

NoSQL 建模模式

@EMC 中国研究院发布一篇关于 NoSQL 的微博：

> 分享一篇很好的博文，NoSQL Data Modeling Techniques (http://t.cn/zOqDUgm)，文章先介绍了 5 种常见 NoSQL 数据库，然后介绍了里面涉及的基本概念，最后总结了 14 个 NoSQL 的常见建模模式。文章在 Wordpress，可能被墙，附件为全文截图。

皮东 chen 的评论是：

> NoSQL 的数据模型看似简单，但这篇文章却分析的非常深入和详细，强烈推荐！

原文链接：http://www.infoq.com/cn/news/2012/03/0309-hot-weibo

相关内容：

- 专访 SegmentFault 创始人祁宁
- 揭秘 InputFormat：掌控 Map Reduce 任务执行的利器
- 人人网专家谈开放平台中的“防癌系统”和敏捷
- 构建实时垂直搜索网站经验分享
- 小米陈臻谈互联网分布式系统架构

Powered by InfoQ

QClub

我们影响有影响力的人

北京 上海 广州 大连 西安 太原 成都 杭州 武汉 南京 深圳...

中文 | 英文 | 日文 | 葡文 | .....

内容投稿 editors@cn.infoq.com | 商务合作 sales@cn.infoq.com | 加入我们 hr@cn.infoq.com

特别专题 策划：InfoQ 中文站； 执行：张凯峰

动态语言更上一层楼

动态语言与传统静态语言之争，以及动态语言部落内孰优孰劣之争，一直没有离开过人们的视线。抛开动态语言百花齐放的局面不谈，它们在不管是新兴的互联网应用还是传统的企业应用中所占有的重要地位，仍然呈上升态势。从这篇专题介绍截稿期止，TIOBE2012 年 3 月份编程语言排行，动态语言占据前十名中四席，分别是 PHP（6）、JavaScript（8）、Python（9）、Perl（6）。

在现代软件系统开发周期中，动态语言已经涉及但不限于前后端开发测试、系统基础设施搭建、系统构建部署。在本期专题中，编辑就为读者精心挑选了几篇涉及相关内容的文章，以飨读者。

从简单到复杂：大型 Rails 与 VoIP 系统架构与部署实践

虚拟座谈会：PyCon 嘉宾谈 Python

软件开发地基

虚拟座谈：JavaScript 单元测试现状

一个前端工程师眼里的 NodeJS

特别专题

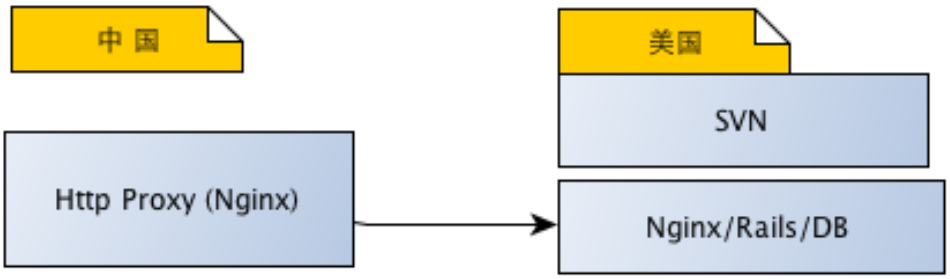
从简单到复杂：大型 Rails 与 VoIP 系统架构与部署实践

作者 杜金房 高超

回顾

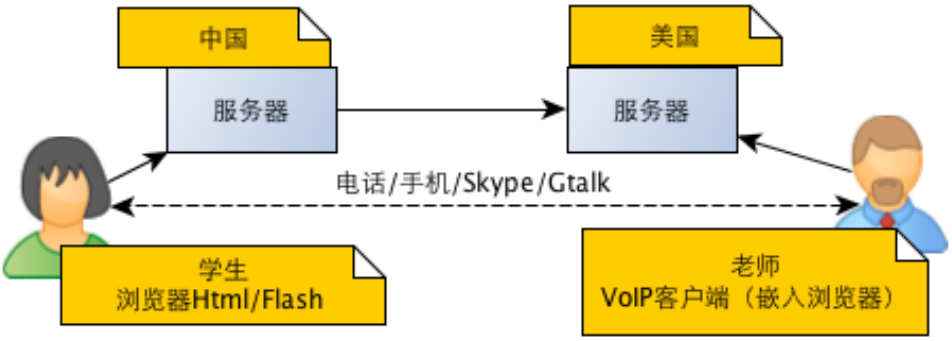
2008 年初，我加入一个 Rails 团队——Idapted。Idapted 是国内最早的 Rails 团队之一，带头人 Jonathan Palley 颇有创新和冒险精神，他一个人写了几乎全部早期的系统原型，包括但不限于 Rails 后台、VoIP 客户端和 Flex 前端等等。

我刚刚加入团队时，面对眼花缭乱的技术简直不知所措。对我来说一切都是全新的。我作为一个系统管理员，职责是管理三台服务器：一台在国内，主要是用作 HTTP 代理；两台在国外，分别是 SVN 和 Rails 应用。当然，当时只一个 Rails 系统，与数据库都在一台服务器上。



业务简介

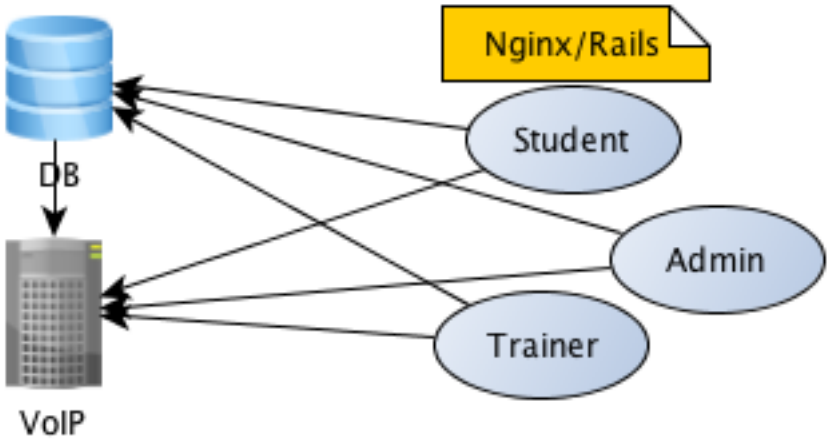
在详细介绍之前，我先来简单说明一下公司的业务，以便于读者更好地理解与之相关的技术。我们主要的业务是做一对一的在线英语口语培训。老师在美国，学生主要在中国。学生学习的过程一般分为三个阶段：预习、一对一连线和复习。预习和复习阶段主要是使用 Flex 呈现课件内容（那时候还没有 HTML5），连线阶段也使用 Flash，只不过增加了 VoIP 应用。在美国，老师使用基于 SIP 的 VoIP 客户端，而在中国，学生则可以通过电话或手机、Skype、Google Talk 等与老师连线实时对话，通话过程全程录音。老师除了在通话过程中实时纠正学生的发音和语法外，还会在在录音的相关位置做上标记和反馈，学生就可以在复习时掌握这些内容。



架构

业务决定架构。最初我们也把 Rails 部署到国内的服务器上，但这样美国的老师访问起来就很慢。所以后来我们就把 Rails 移到美国，而国内的服务器就只作 HTTP 反向代理；另外美国的 VoIP 环境比国内要好，让它靠近 Rails 服务是理所当然的。

随着 Rails 项目的代码越来越多，我们决定将系统拆分成三个部分：Admin、Trainer 与 Student，分别负责管理员功能、老师平台及学生。VoIP 后台也由开源的 Asterisk 换成了当时比较年轻的 FreeSWITCH。同时我们也把系统迁移到了新的美国服务器上：Database + VoIP + Rails。



工具

Rails 系统使用的是典型的 Nginx + Mongrel + MySQL，部署使用 Capistrano。为了更方便的部署系统我基于 Capistrano 写了一个小工具，可以通过类似 cap admin/trainer/student deploy 方式部署系统。后来随着系统被拆分的越来越多，我们又发现了一个好的工具 Webistrano。

为了方便测试，我们在北京的办公室放了两台 PC 服务器，使用 Xen 虚拟化技术，虚拟机与生产系统的服务器一一对应，并使用真实数据进行测试，保证测试环境与生产环境的表现完全一样。同时我们也把 SVN 服务器移到了办公室，这样提交代码就快多了。

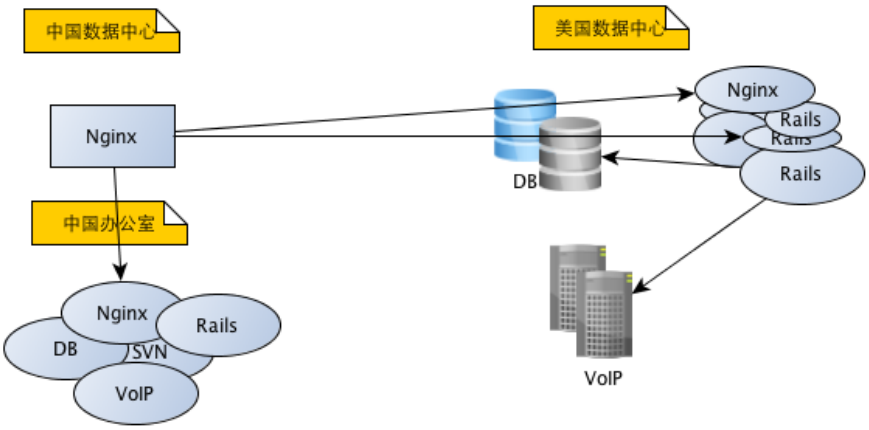
迭代

接下来随着业务的发展，我们对 Rails 系统进行了更进一步的拆分，最终产生了数十个 Rails 应用，《From 1 to 30: How to Refactor 1 Monolithic Application into 30 Independently Maintainable Applications》便是我们在 RailsConf 2010 的演讲主题。

在拆分过程中我们也尝试了好多不同的架构和部署方案，有成功的喜悦也有失败的惨痛经历。其中一件事情让我印象深刻：拆分后，各 Rails App 之间的通信就主要靠共享只读数据库和 HTTP Rest Service（大部分使用 Rails 中的 ActiveResource 实现），而当时我们也正在尝试基于 Phusion Passenger 的部署方式。因为我们的系统和 Passenger 都存在 BUG，因此整个经常莫名奇妙地失去反应。最后我们得出的教训就是：当代码和运行环境都不够稳定的时候，定位问题往往需要花费大量的时间，而这是可以避免的。

总体架构

后来，在 Idapted 被 Eleutian Technology 收购以后，我们把代码仓库从 SVN 迁移到了 github。同时也把服务器迁移到了一个新的数据中心。最后的架构如下图：



技术架构

下面来详细谈一下我们系统的架构和使用的技术，以下基本都是来自我们的实际经验。至于运行环境，所有 Rails 都是运行在 Ubuntu Linux 上，我们只使用 LTS 版，如 8.04 和 10.04。

Rails2 的应用使用 Ruby Enterprise Edition 1.7，Rails3 则使用 Ruby 1.9。

反向代理

最后我们选择了 Nginx + Unicorn 的方式。Nginx 现在几乎已经成为事实上的标准了，而 Unicorn 更是非常优雅，它不仅稳定高效，而且可以很方便的添加和减少进程。不管起多少个进程，都只占用一个 TCP 端口，非常方便与 Nginx 联合部署。

为避免 Javascript 跨域请求安全性问题，以及使所有的 Rails App 看起来协调统一，我们将所有 App 部署在同一个域名下不同的子目录中：

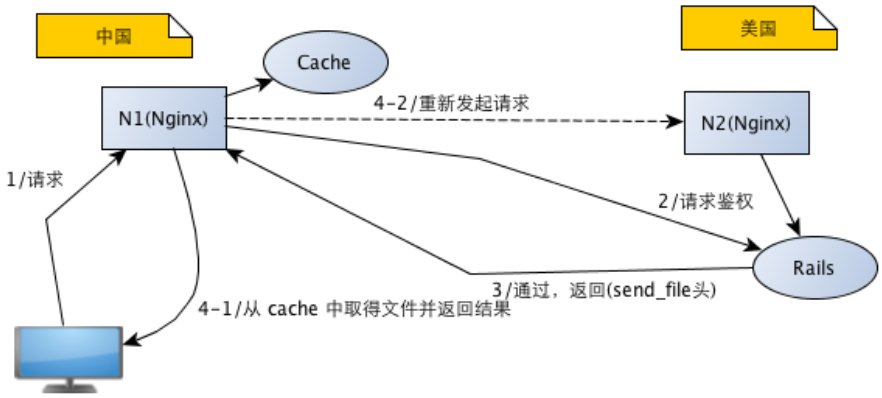
```
upstream app1 {
    app1.lan:3010;
    app2.lan:3010
}

upstream app2 {
    app1.lan:3020;
}
upstream app3 {
    app1.lan:3030;
}

location /app1 {
    include /usr/local/nginx/conf/proxy_headers.conf;
proxy_pass http://app1;
}
location /app2 {
    include /usr/local/nginx/conf/proxy_headers.conf;
proxy_pass http://app2;
}
location /app3 {
    include /usr/local/nginx/conf/proxy_headers.conf;
proxy_pass http://app3;
}
```

静态内容与文件服务器

我们使用 Squid 在国内服务器对普通的静态内容做缓存。另外，我们建立了自己的文件服务器用于存放用户上传文件，大部分是录音（录音也是使用统一的上传文件接口）。文件服务器上的文件会实时备份到 Amason S3。

对于文件服务器的缓存，我们使用了 Nginx 的 cache + sendfile 功能。如下图，国内服务器 N1 收到 GET 请求后(1)，使用 http_proxy 请求国外服务器上的 Rails 应用进行鉴权(2)。鉴权通过后 Rails 返回 sendfile HTTP 头(3)。N1 则在本地缓存中查找对应的文件，如果存在则直接

返回文件(4-1)，如果不存在，它会再发出一个 http_proxy GET 请求(4-2)到国外服务器上的 Nginx(N2)，N2 返回文件到 N1，N1 将文件发送给用户同时缓存在本地。



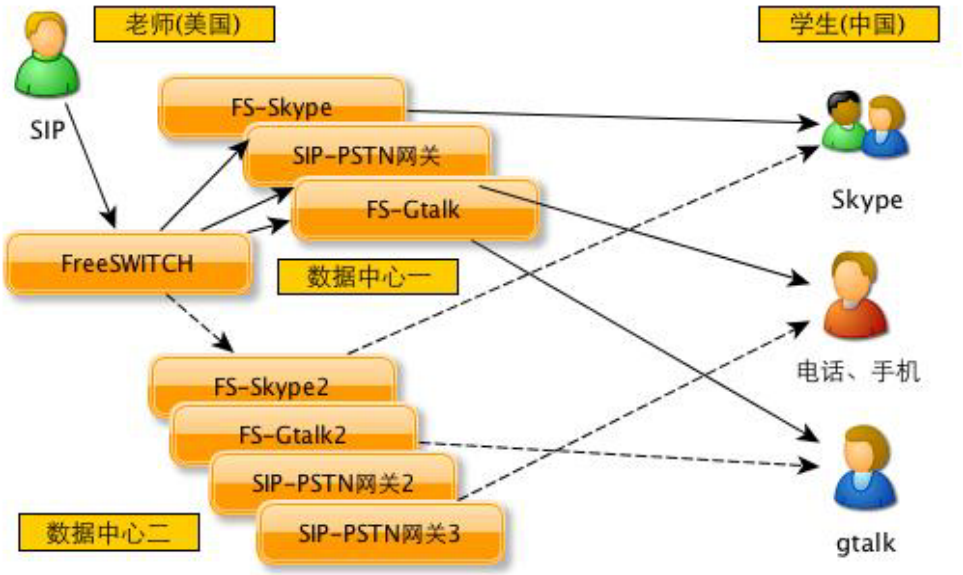
配置如下：

```
    location /file {
        include /usr/local/nginx/conf/proxy_headers.conf;
    proxy_pass http://file-rails-server:60020;
    }

    location /file-internal {
        internal;
    proxy_set_header   X-Real-IP  $remote_addr;
    proxy_store       on;
    proxy_store_accessuser:rwgroup:rwall:r;
    proxy_temp_path    /tmp/nginx_temp;

        alias            /home/app/shared/file-internal;
    proxy_set_header   X-debug1  $request_filename;
    proxy_set_header   X-Uri $uri;
    proxy_set_header   Host  www.idapted.com;
    proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        if (!-f $request_filename) {
    proxy_pass http://file-nginx-server;
        }
    }
```

国外服务器上的情形和这个类似。由于服务器空间有限，录音文件又会占用很大的空间，因此我们会定期删除服务器上长时间未访问的文件，如果偶尔有人访问的话，则通过我们自己编写的一个 idp_proxy 实时从 S3 上取回。

VoIP 架构

如前所述，VoIP 平台使用的是 FreeSWITCH，采用 CentOS 5.5 双机冷备份，并在一台服务器上启动多个 FreeSWITCH 实例，以支持 Skype、Google Talk 等。详细的内容请参考这篇 Blog。



Erlang

除 Rails 外，我们在好多地方也使用了 Erlang。Erlang 是一种函数式语言，除了我们喜欢它的简洁和优雅，更重要的是，FreeSWITCH 有一个原生的 Erlang 接口，它最初就是用来处理电信业务的，用它控制 VoIP 再适合不过。

在我们系统中，Erlang 开发的程序类似于一种中间件，它位于 FreeSWITCH 及 Rails 系统之间，通过原生接口与 FreeSWITCH 通信，与 Rails 通信则通过 HTTP REST 接口及共享数据库。另外，我们的 VoIP 客户端也使用一个基于 TCP 的简单协议与 Erlang 通信。

监控与分析

监控总是最关键的一点。如果你不知道你的服务器在干什么，你就不知道该怎么做。我们使用 monit、munin 及 nagios 进行监控，通过短信、Email/IM 等方式接收事件通知。

另外，我们也自己开发了一些监控与分析系统，如 VoIP 行为的关联分析，通过对系统日志的分析及协议的跟踪，来帮助我们分析通话质量问题及断线原因。

其他技术

Flex

所有 Flex 代码都在服务器上编译，避免由不同开发者不同版本的编译器编译的模块加载时

出现问题。

虚拟化

由于 Xen 的发展不明朗，我们在新的生产环境中使用了 LXC 及 KVM 虚拟化技术。两者只是在不同的服务器上进行简单的负荷分担，在虚拟化方面我们没有进行更多地研究。

我们还使用了其它常用及不常用的技术，如 memcache、iwatch 等， 在这里就不费笔墨一一赘述了。

小结

系统架构是一门很深的学问，部署和维护也同样重要。我最初也没有太多经验，所有经验都是在不断的开发，维护中不断学习和积累起来的。当然，从失败和错误中学习是最快最好的方法。以下是我们总结的几点经验：

谨慎使用最新的技术

敏捷往往与激进联系起来，Rails 和 FreeSWITCH 发展很快，我们总是使用最新的版本，保证我们总是能使用那些最新的特性。当然，前面已经说过，当环境和代码都不稳定时，查找 Bug 的难度要增大好多倍。所以我们在操作系统及 HTTP 服务器的选择上又比较保守，这样就保证在敏捷的同时又最大限度的相对稳定。

测试环境尽量与生产环境一致

许多错误在测试阶段没有发现，都是由于测试环境与生产环境不一致引起的。因此，我们会花相当多的时间保证测试环境与生产环境在软件版本上保持一致，甚至，我们还使用虚拟化技术使网络拓扑结构保持一致，比如用虚拟机模拟物理服务器使其在数量上保持一致，并使用真实数据进行测试。

结对操作

我们不仅在开发阶段实施结对编程，在重要的部署维护阶段也是双人结对操作。人都是会犯错误的，我就曾经在单独操作时点错了按钮而几乎犯下大错。因此，我们通过引入双人结对操作的方式，最大限度地降低人为因素在部署维护阶段的影响，对重要操作的定义也更为严格。

从错误中学习，让开发人员也参与进来

我们会把系统故障及处理过程记下来，做成 Case 供大家学习，并在每周一次的 Code Review 时间与开发人员交流。每个人都知道系统架构与维护并不完全是架构师和管理员的责任，能够开发出易于部署和维护的系统更是开发人员的责任，好的系统是整个团队紧密合作的结晶。

让管理层理解系统维护的重要性

作为系统维护人员，最郁闷的可能就是系统不出问题。这可不是开玩笑。一般来说，系统不出问题是由于系统维护工作做的好；但管理层也许会误认为管理人员整天没事做。所以，处理好工作与老板的关系是一门艺术，也是团队成功的关键。

关于作者

杜金房，前 Eleutian Technology（前 Idapted）核心技术架构师，主要负责系统底层架构及 VoIP 系统开发。业余时间创办了 FreeSWITCH-CN。曾任职烟台电信/网通，负责交换机及网管系统维护工作。

高超，Eleutian Technology（前 Idapted）高级网络工程师，负责系统底层架构与监控系统开发与维护。

注：本文是 idapted 公司 Rails 系列技术文章的第三篇，前两篇分别为《Rails 系统重构：从单一复杂系统到多个小应用集群》和《如何进行高效的 Rails 单元测试》。

原文链接：
http://www.infoq.com/cn/articles/rails-voip-system-architecture-and-development

相关内容：

- Google Dart 精粹：应用构建，快照和隔离体
- Ruby on Rails：3.2 RC1 发布，4.0 将会放弃 Ruby 1.8.7
- 计划 FM 为人人网提供首个开源 Ruby SDK
- 如何进行高效的 Rails 单元测试
- Rails 系统重构：从单一复杂系统到多个小应用集群

特别专题

虚拟座谈会：PyCon 嘉宾谈 Python

作者 丁雪丰

【编者按】今年 12 月 3 日、4 日，Python 界著名的 PyCon 大会首次登陆中国，邀请了众多国内的 Python 高手作为嘉宾，带来了很多精彩演讲。InfoQ 有幸邀请了几位与会嘉宾就广大 Python 爱好者所关心的问题发表了各自的观点。

近些年 Python 在语言特性的发展上，有什么值得称道的地方，又有什么需要改进的地方？

洪强宁：Python 语言最近的发展我比较关心的有两个——Python 3 和 PyPy。

Python 3 的动机是修正之前 Python 版本中一些设计不合理的地方，所以不兼容旧版本。但由于其实 Python 2 中那些“不太合理的地方”并不是很干扰开发，大家都已经习惯了，而 Python 3 特别吸引人的新特性并不多（nonlocal 是我的最爱），因此相对而言，为了升级而必须解决兼容性问题的工作量显得投入产出比过大，并不合算。这就导致大量项目迟迟不提供对 Python 3 的支持，而依赖这些项目的其他项目因此也难以迁移到 Python 3。如果 Python 3 能够有更多激动人心的特性支持的话（比如更强大和好用的 lambda），整个社区的迁移过程应该会快很多。

PyPy 是一个很有意思的项目，JIT、去掉 GIL、更好的垃圾收集机制、Stackless 支持我认为都是正确的方向。如果 PyPy 能够成为主流，对于有性能要求的服务端开发，会是一件很好的事情。当然 PyPy 目前还不够成熟，比如不支持 CPython 的 C extension ，这就大大限制了它的使用。性能提升除了它自己的 Benchmark 的数据外，也还需要有更多成功案例的支持。

周琦（Zoom.Quiet）：Python 3 毅然决然地坚定高速发展，在广泛接受建议的同时，Guido 还会解决所有以前设计的问题，非常值得期待。PyPy / Jython / Cython / Parallel Python 等项目得益于 Python 的开源，各种 Python 实现都在快速发展，它们也解决了各种领域问题。一句话，Python 在健康强壮的发展中。

需要改进的就是 Python 3 的模块迁移一直没有很好的解决方案，期望在完成 Python 3 的稳定版本开发后 Guido 可以将精力关注到这方面。不过，execnet 已给出了一个很好的过

渡方案——以 execnet 为中介，Python 3 主程序可以直接使用 Python 2 的兼容模块！

李迎辉（Limodou）：Python 3 代表了 Python 未来发展的趋势，它的内部越来越一致，比如内部 Unicode 的统一，这样将减少字符串编码转换带来的一些麻烦。当然，在不断向完善过程发展的同时，也带来一些向前的不兼容性，好在还有 2.7 作为过渡，以及提供 2to3 程序来辅助转换，这在一定程度上会减少转换带来的麻烦。不足之处依然存在，比如 GIL 还无法去除。另外个人感觉 Ruby 和 Javascript 中的闭包是一个不错的特性，也许可以吸收进来。

赖勇浩：Python 这些年一直在发展，从 2.6 到 3.2 版本的发布都是非常给力的，在 3.x 持续增加新特性的同时还发布新的 2.6、2.7 版本把这些新特性放入 2.x 分枝，Python 开发者无论选择哪一个大版本都能得到充分的尊重——这是最为难能可贵的、值得称道的。

关于它需要改进的地方，我觉得已经不多了，因为一些 Bad Smell 的问题已经在 3.x 版本中得到了根本性的修正。不过我在 2010 年 11 月底的时候写过一篇博客《我想要的 Python 特性》，里面谈到 3 个 Python 目前没有提供的特性，分别是在函数式编程、设计模式和面向消息编程方面的增强，但从文章的评论来看，赞同和抵制的人基本相当。关于面向消息编程，我自己实现了一个 python-message 开源库，已经在我主持的一个商业项目中使用，能够很好地解耦模块关系，在 PyCon 也作了简单的宣传，在此再向大家推荐。

黄冬：Python 语言最让我关心的事情是多核处理器和 PyPy。

Python 从最早的只能使用系统的标准 fork 进行多进程实现，但是进程管理较其它语言的多核支持差点多。后来加入了 multiprocessing，使得通过多进程能更为方便的使用多进程从而达到多核的使用。Python 一直把自己放在一个更接近于操作系统的极别，所以它更喜欢自己像 C 一样的与操作系统打交道，这即是一个优点，同样也少了很多方便的特性。

PyPy 则是一个非常有意思的东西，它将一个去掉 GIL 的世界带了进来，同时 JIT 可以让我们更快的运行起来，这样让 Python 从一个运行时的解释形态，走入了一个全新的状态。非常期待它的新特性加入进来。

未来，我认为更好的支持多核和更好的操作系统特性支持，将会是一个非常重要的改进方向。

您觉得在移动互联网和 Big Data 兴起的今天，Python 的机会在哪里？

周琦：Python 一向是有机会的，只是以往受到 Nokia 的扼制，现在更加自在而已。个人感觉 Python 在 3 大方向上都能有所作为：

- 传统的服务端开发，配合 HTML5 的各种 API 服务开发，Python 是最敏捷的

- 大规模的手机应用自动化测试，Python 的强力内存探测功能，可以直接对 C / C++ / Java 应用进行测试，以后也将包含 Objective-C
- 嵌入式应用，随着各个平台的不断开放，将 tinypy 嵌入到手机里，作为原生驱动的调用界面

随着 CommonJS 思想的兴起，我认为在 Node.js / Objective-C 这类新兴语言的 DSL 实现时直接选择 Python 的语法也有可能，这也算是 Python 在移动时代的另类发展吧。

李迎辉：Python 一方面作为通用的开发语言，具备清晰、易读、开发效率高并且包含丰富的库的特点，我觉得移动互联特别需要能够快速开发的语言，以实现需求的快速响应，而 Python 的确可以满足这方面的要求。另一方面，Python 又作为一种“胶水”语言，可以方便实现与 C / C++等语言的互相调用，因此既可以满足某些关键环境性能的要求，又可以充分利用现有资源。所以在专业领域我们仍然可以继续使用原有的技术，同时可以充分发挥 Python 的“胶水”特性，以快速构建所需要的应用。

黄冬：Python 做为处处可用的语言，移动互联网的很多 Web 服务都用其进行开发，因为 Python 高效且严谨的语言特性而为大家喜爱，更重要的是它在开发效率和性能方面找到了一个非常良好的折中点。

我自己在大量的移动互联网公司看到 Python 支撑着他们的 Web 服务、后台业务处理、数据清洗与加工处理等各个环节。唯一遗憾的是，Python 在移动互联网的今天还是没有在前端走出来，但是有所长必有所短，发挥语言的特性、实现价值没有什么不好。

在使用 Python 的时候，是否有遇到过性能上的问题，是如何解决的？

赖勇浩：有的，就像以前使用 C++的时候一样。无论使用什么编程语言，都会遇到性能问题。性能问题的根源不是编程语言，而是业务。举个例子，你说 C++执行效率高吧，写个 CPU 软件渲染图形游戏引擎跑一下 Unreal Engine 3 提供的特性试试看有没有性能问题？在 PyCon 的快速演讲环节中，阿里云的产品经理李俊东先生就鲜明地提出自己的观点——不要迷恋语言性能，要多关注业务性能！遇上 Python 性能问题的时候，解决方案与其它语言遇上性能问题是类似的，有以下步骤：

1. 使用 Profiler 确定性能瓶颈
2. 使用更适合的数据结构与算法、更合理的 I/O 模型、更少的运算（如缓存之前的运算结果）
3. 使用更快的编程语言重写瓶颈部分，就像用 C++时使用汇编指令重写那样

更细节的内容，不妨参考我博客的几篇文章。

李迎辉：性能问题产生的原因有很多，一方面可能是由于 Python 语言本身带来的，因此这块可以考虑通过 C / C++来进行改造；另外 pypy 也可能是一个发展的方向，它通过即时编译来提高 Python 执行效率。另一方面可能是由于设计造成的，这块就无法仅通过语言的优化来实现，可能更多通过架构设计、集群计算、软硬件相结合等办法来解决。

多进程部署和协程似乎已经成为了一种趋势，在大会上很多嘉宾都谈到了这方面的内容，您是怎么认为的？

沈崴：赖勇浩在大会上说“协程才是未来”，这句话我很赞同。多线程用来维护上下文，开发成本低但是并发性能差；传统的异步模型，并发能力强但是开发成本高，而且很难有效框架化。协程没有这两方面的缺陷，并兼具他们的优点。当前大家都比较关注协程的性能优势，但是协程的核心，应该是以线程逻辑来维护上下文，以极低的成本来实现高并发。单从性能上讲，协程相比传统异步模型并没有优势，开发效率的革命性提升，这才是关键。我们使用协程已有五年多的时间，相较传统模式协程所具有的压倒性优势，我们深有感触。协程就是未来。

多进程部署则是另一个范畴，进程、线程其实都可以与协程一起很好地工作，我在大会演讲中也有提到。同时根据会上一些嘉宾的观点，线程他们其实用得并不多，对我们来讲，面对多线程也是采取知难而退的态度。相对线程而言进程却有着根本性的区别，从传统上讲多进程涉及的范畴非常广，从多核、机器间协作到云计算等。一直以来能否正确运用多进程都是架构设计的核心本质，从目前我们接触到的一些流行的架构设计案例来看，要做到这一点其实并不容易。对很多架构设计者而言，要学好多进程部署还有很长一段路要走。

洪强宁：与单进程多线程模型相比，多进程和协程是更加 Scalable 的模型。在高并发场景下，采用多进程模型编制的程序更加容易 Scale Out，而协程模型可以使单机的并发性能大幅提升，达到 Scale Up 的目的。所以，未来服务器端并发模型的标配估计会是：每个核一个进程，每个进程是用协程实现的微线程。

在编码方面，多线程模型带来的共享资源加解锁的问题一直是程序员的梦魇。而用多进程模型编程时，会自然鼓励程序员写出避免共享资源的程序，从而提高鲁棒性。而 Python 目前的协程实现都为非抢占式调度，程序员自行控制协程切换时机，因此也可以避免绝大多数令人头疼的加解锁问题。这些都利于写出更稳定的代码。

另外，和同样具有很好并发性能的事件驱动模型相比，用协程实现的微线程，在逻辑表达上非常友好和直白，无须在不知道什么时候会发生的 event 和一层套一层的 callback 中纠结

和扭曲（正如 Twisted 其名）。对于写过多线程程序的程序员而言，协程带来的微线程模型几乎可以实现无痛提高并发性能。

我一直是个协程的鼓吹者，并坚信这是服务端编程的正确方向。所以希望大家把 Twisted 踩在脚下，忘掉 Tornado，把精力放到 gevent 和其他基于协程的框架上来吧。

李迎辉：这些方面更多是集中在后台服务器方面的开发，在网络后台、游戏服务器开发等应用比较多。多进程没什么好说的，因为它实现方便，而且因为 GIL 问题的存在，现在是比较好的并发处理模式。而协程也更多是为了充分利用 I/O 资源，因此也有它使用的局限性。所以如果对服务开发有兴趣的不仿多了解一些这方面的内容。 Python 除了做服务器开发，还有许多的开发领域，所以你的开发兴趣不在这方面，暂时不了解也没有关系。

赖勇浩：在未来，起码在脚本语言的世界里，线程的地盘一定会进一步缩小。在 Lua 和 Ruby 中，你看不到原生线程的存在，但它们还是流行了。充分说明哪怕是多核时代，线程也不受待见。相对于获取更佳的 CPU 并行计算性能，现代的程序员更乐于获取更佳的代码可维护性，因为现代项目已经非常庞大，运行在多台计算机甚至多个数据中心上，榨取单机性能显得有点舍本逐末。我个人浅见是在未来线程会有点像 C++的处境，它不会被淘汰，但会被限制在编写为多核优化的程序库等领域，一个能够 CPU 并行计算的 Python 库的确是有吸引力的，但普通的业务逻辑程序员就不碰线程为妙了。

黄冬：其实 Python 在原生多进程上支持非常优美，只是它没有提供一个原生 VM 来屏蔽系统级多进程操作复杂。我认为有两个方法，等待一个更为优美的解决方案的出现，或者耐心看看 Python 已经比原生 fork 更酷的实现吧。gevent 这样的框架是高性能服务器的必然选择，它更轻更简单，更好的利用了 libevent 的特性，时代在前进，相信它也会更为成熟的。

您认为像 Python 这样的动态语言应用在开发大型项目或企业级开发的时候有什么局限性？

沈崴：我想说 Python 确实不太适合用来开发大型项目。我听许多业内的朋友说，他们用 Python 做软件，人力成本和项目周期经常只是竞争对手的一个零头。这很糟糕，这意味着任意一个大型项目到 Python 这里几乎都会变成小项目。相反，任何一个再普通不过的需求使用某些流行语言开发，都有机会演变成大型项目。这很好地解释了为什么很少听说 Python 有什么大项目，而其他语言的大项目总是很多的原因，这也是 Python 被误认为缺乏大规模开发特性的原因。这也掩盖了一个事实，如果你仔细观察就会发现 Python 在各个领域其实都扮演着极其重要的角色，虽然他会很低调。

在这个时代，代码规模和程序员痛苦指数已经不再是评判项目规模和员工绩效的有效标准，

我们的公司主管们需要尽快地习惯这一点。同时对 Python 程序员而言，往往是轻描淡写地把任务完成了，这时你需要尽量克制住把程序做成大项目的冲动，这对项目是非常危险的，也是不负责任的。喜欢通过大型项目证明自己是典型的初学者心态，Python 程序员不需要这个。因此我会说 Python 不太适合用来开发大型项目，虽然有点答非所问，但还是希望对大家会有点用。

洪强宁：说实话我没有觉得在开发大型项目（豆瓣的网站算大型项目吗？）上 Python 有什么局限性。相反，在开发大型项目时，由于人员协作变得重要，代码可维护性的重要性凸显。使用 Python 更容易写出简单易读风格一致的代码，语言本身的高抽象能力可以大幅降低代码量，强大的语法和良好的包管理机制使得重构变得容易。这些都很好的提高了代码的可维护性，所以我强烈推荐使用 Python 进行大型项目开发。

如果一定要说有什么局限性的话，那就是动态语言难以做静态分析，所以不像静态语言很多问题可以在编译期发现。但配合上完善的自动化测试（这个无论是用静态语言还是动态语言开发项目都应该是必需的），这个问题可以很好的解决。

对“企业级开发”的定义不太清楚，因此不好做什么评价。

李迎辉：首先可能是标准化的问题。因为 Python 没有企业应用级别的应用标准，所以无法实现类似 J2EE 一样的应用标准架构，所以更多还是看设计人员个人的水平发挥。另外就是 Python 虽然自带了许多组件，但是没有完整的针对企业的套件，而且也没有相应的商业化支持，所以还无法形成相应的开发规模，企业也不愿意投入。再加上，国内 Python 语言开发还不是主流，没有形成规模化，更多的 Python 开发集中在 Web 和游戏方面，企业级开发的研究和投入不足，Python 程序员还比较少，所以很难形成相应的市场。

周琦：简单的说，没有！严格的说，只有成员招聘、文化重塑的麻烦。

优秀的 Python 开发者一直都不太容易找得到，在人员的培养和招聘方面您有什么心得？

沈崴：因为 Python 上手容易，所以招聘的选择面反而非常宽。你其实并不关心应聘者是否熟练地使用了 Java，又或者能否担当 C++各种开发陷阱的活字典，当然要形成这些技能确实需要许多年或者几十年的工作经验，而现在这些都不重要。如今你将开始关心一些更为本质的东西，程序员应有的悟性以及灵性，比如快速学习的能力、洞察力，以及是否热爱编程。

我们这里不仅有 Java 开发者还有.Net 程序员，既有资深工程师也有应届毕业生，他们无一例外都能很好地玩转 Python。同时我们的培训专注于稳固扎实的基础，能够掌握和重现各种底层原理。再结合 Python 强大的实现能力，就能轻松完成许多让普通程序员感觉非常头痛或者根本无法完成的任务。

李迎辉：的确，现在国内招聘 Python 的越来越多，但是从个人角度来说因为 Python 的开发还不太上规模，相应的职业体系和培训机制还没有建立，所以更多人员是靠自学成才，但是这样远远无法满足企业招聘的需求。而且目前许多 Python 程序员和其它的程序员不同，他们大多是对开发本身有着强烈的兴趣，因此这些人的水平和素质相应要高于其它的程序员群体，所以也造成这样的人才在数量上也的确不会太多。所以我觉得一方面要靠市场本身来孕育更多的 Python 程序员，另一方面我个人建议如果企业能在培训、在活动、在支持开源社区上多投入，一方面宣传自已另一方面可以多发展有兴趣的程序员，从而形成良性循环。而目的的情况是，Python 相关的活动比较少，并且大多是自由发起，所以还没有形成比较好的环境。

赖勇浩：无论什么编程语言，优秀的开发者都不太容易找到。给优秀人才想要的，无论是物质还是感情，你身边自然能够聚焦高手。如果又要马儿跑，又要马儿不吃草，请先摆正你的心态。所以如果招不到优秀人才，不妨从如下三个方向入手：一是改变业务逻辑，降低技术难度，自然不需要优秀人才了；二是另找一条路，比如你一直想找更快的马，找不到的时候不妨买辆汽车，说回来就是换套解决方案，比如 Python 程序员不好找，你用 C# / Java 不就得了，说不定还更好；三是加强自己的修为，所谓物以类聚人以群分，你很强大，自然容易找到强大的合作者。

周琦：Python 本身如此简单，使程序员可以在第一时间关注业务实现本身，而不是语言特性。所以，招聘也好培养也好，都关注实战就好。可以看看一直以来在推广 Python 的蟒营。

黄冬：其实所有使用 C 的人使用 Python 都很快，Java 上手 Python 也很快，但是，无论哪种语言，优秀的开发者本身就很少。人员的培养都是相似的，树立技术价值观，建立好的人员筛选和目标设定、评估体系，这与任何一种工作、任何一家公司都一样。招聘当然要去 Python 社区了，邮件列表、开源项目都是优秀的 Python 开发者的藏身之处。

对于打算尝试 Python 的开发者和企业，您有什么建议？

周琦：Python 世界早已自足，只要相信，就无所不能。

李迎辉：尝试 Python 没有错，许许多多失败的例子并不是因为选择了某种语言，更多还是人的因素。所以我觉得要想成功，更多的是在语言之外的东西。所以我的建议是认定方向就要坚持，踏实不浮燥，多考虑其它成功的项目，多与其他人进行沟通和交流。

赖勇浩：对于开发者，我觉得多学一门编程语言肯定不错，利用业余时间学一下大有增益；但切忌随便看点书或文章就下了结论认为这东西无所不能或一文不值，要学就学深入一点。对于企业，我的看法是不要轻易尝试，哪怕 PyCon 的讲师说得天花乱坠。引入新的技术就

是引入新的风险，对之前的技术架构、员工情绪、企业文化都会带来巨大冲击，除非你已经考虑清楚愿意为这件事情折腾到脱一层皮都在所不惜。

原文链接：http://www.infoq.com/cn/articles/virtual-panel-pycon

相关内容：

- HeadwaySoftware 在 structure101 和 restructure101 中添加了对 Doxygen 和 Understand 的支持
- Nuxeo 公司探秘：从 Python 迁移到 Java
- Python 于 Web 2.0 网站的应用

特别专题

软件开发地基

作者 郑晔

在软件开发过程中，缺乏一个好的自动化脚本，与之相伴的往往是日常的开发工作举步维艰：

- 只有少数人能够把整个软件构建起来，因为构建所需的那些东西不太容易弄全。
- 为了能在自己机器上写代码，开发人员要花大量时间把工程在 IDE 上配出来。
- 提交代码之前，开发人员总是忘了在验证。

在本文中，我们将以一个 Java 的 web 项目为例，展示一个好的“地基”应具备的一些基本素质。在这里，用做自动化的工具是 buildr。

buildr 是一种构建工具，它专为基于 Java 的应用而设计，也包括了对 Scala、Groovy 等 JVM 语言的支持。相比于 ant 和 maven 这些 Java 世界的“老人”，buildr 算是小字辈，也正是因为年轻，它有着“老人”们不具备的优势：

- 相比于 ant，遵循着 Convention over Configuration 原则的 buildr，让“编译、测试、打包”之类简单的事做起来很容易。
- 相比于 maven，我们无需理解强大且复杂的模型，而采用 Ruby/Rake 作为脚本的基础，也让我们可以定制属于自己的脚本。

简而言之，它满足了我们选择工具的基本原则：“易者易为，难者可为”。

请注意：下面所有的内容并不只是 buildr 的独家专利，而是每个构建工程都应该具备的，差异只在于，选择不同的工具，实现的难度略有差异而已。

易者易为

让我们从一个简单的 buildfile——buildr 的脚本——起步：

```
GUAVA = 'com.google.guava:guava:jar:r09'
define 'killer' do
project.version = '0.0.1'
  define 'domain' do
```

```
    compile.with GUAVA
            package :jar
        end
        define 'web' do
            DOMAIN = project('killer:domain').packages
    compile.with DOMAIN
            package(:war).with(:libs=>DOMAIN)
        end
    end
```

我们先来看看从这个简单的 buildfile 中，我们可以得到什么。

分模块项目

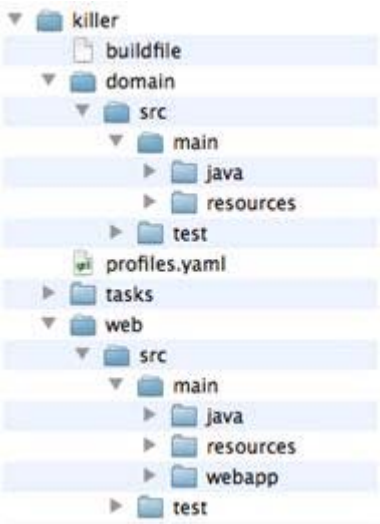
这个项目里有两个子项目：domain 和 web。从架构的角度来看，一个项目从一开始就划分出这样的模块是有好处的：

- 给未来扩展留下接口，比如要提供一个 Web Service，可以从 domain 部分开始即可。
- 给开发人员一个好的规划，有助于引导他们思考程序的模块化，降低代码的耦合度。

使用 buildr 划分模块是非常简单的，只要在 buildfile 里声明模块，项目的根目录下同名的子目录就是对应的模块。

文件布局

虽然在 buildfile 没有直接体现出来，但这里有个缺省的文件布局。一个统一的规则省去了我们从头规划的苦恼。遵循缺省的布局规则，buildr 自己就会找到相应的文件，进行处理。



这个布局规则实际上就是 Maven 的布局规则，如图所示，两个子项目都拥有自己的目录，

其结构基本一致：

- src/main/java，源代码文件目录
- src/main/resources，资源文件目录
- src/test/java，测试代码目录
- src/main/webapp，web 相关文件目录

此外，这里还有稍后会提及的：

- profiles.yaml，环境相关的配置
- tasks，自定义任务的目录

这就是所谓的 Convention over Configuration。当然，buildr 是支持自定义文件布局的，详情请参见文档。

基本命令

有了这个基本的 buildfile，我们就可以开展日常的工作了。buildr 自身支持很多命令，比如：

- buildr compile，编译项目
- buildr package，项目打包
- buildr test，运行测试

想要了解更多的命令，可以运行下面的命令：

```
buildr -T
```

测试

在这个不测试都不好意思自称程序员的年代，测试，尤其实现级别的测试，诸如单元测试、集成测试，已经成了程序员的常规武器。

诚如上面所见，src/test/java 就是我们的测试文件存放的目录。对于 Java 项目，JUnit 是缺省的配置，只要在这个目录下的 Java 类继承自 junit.framework.TestCase（JUnit 3），或是，在类上标记了 org.junit.runner.RunWith，抑或在方法上标记了 org.junit.Test（JUnit 4）。buildr 就会找到它们，并帮我们料理好编译运行等事宜。约定的力量让我们无需操心这一切。

依赖管理

依赖管理一直是一项令人头疼的问题，也是让许多开发人员搭建纠结于开发环境搭建的一个重要因素。

各种语言的社区分别给出了自己的依赖管理解决方案，对于 Java 社区而言，一种比较成熟的解决方案来自于 Maven。它按照一定规则建立起一个庞大的中央仓库，成熟的 Java 库都会在其中有一席之地。

于是，很多新兴的构建工具都会建立在这个仓库的基础之上，buildr 也不例外。在前面的例子里面，domain 依赖了 Guava 这个库。当我们开始构建应用时，buildr 会自动从中央仓库下载我们缺失的依赖。

不仅仅是依赖，我们还可以拿到对应的文档和源码：

- buildr artifacts，下载依赖
- buildr artifacts:javadoc，下载 javadoc
- buildr artifacts:sources，下载源码

如果不知道如何在 buildfile 里编写依赖，那 mvnrepository.com 是个不错的去处，那里针对不同的构建工具都给出了相应的依赖写法。

与 IDE 集成

除非这个工程是用 IDE 创建出来的，否则把工程集成到一个 IDE 里通常要花费很大的力气。所幸，buildr 替我们把这些工作做好了。我们只要键入一个命令即可，比如与 IntelliJ IDEA 集成，运行下面的命令：

```
buildr idea
```

它会生成一个 IDEA 的工程文件，我们要做的只是用 IDEA 打开它。同样的，还有一个为 Eclipse 准备的命令：

```
buildr eclipse
```

不知道你是否有这样的经验，初到一个项目组，开始为一个项目贡献代码之前，先需要花几天时间，在不同的人的协助之下把环境搭出来，为的只是在自己的机器上能够把应用构建出来。

而现在，有了这样的自动化脚本，一个项目组新人的行为模式就变成了：

- 初入一个项目组，他从源码管理系统上得到检出代码库。
- 调用 buildr artifacts，其所依赖的文件就会下载到本机。
- 调用 buildr idea:generate（或是 buildr eclipse），生成 IDE 工程。
- 打开工程，开始干活。

迄今为止，我们看到的只是一个基本的 buildfile，这些命令也是 buildr 内置的一些基本能力，也就是所谓的“易者易为”。

难者可为

接下来，我们将超越基础，做一些“难者可为”的东西。

不同的环境

在实际的开发中，我们经常会遇到不同的环境，比如，在开发环境下，数据库和应用服务器是在同一台机器上，而在生产环境下，二者会部署到机器上。这里所列举的配置功能，只是最简单的例子，而实际情况下，不同的环境下，会有各种差异，甚至需要执行不同的代码。

一种解决方案是为数不少的“直觉式”设计采用的方案，在代码里根据条件进行判断，可想而知，无处不在的 if..else 很快就会把代码变成一团浆糊，更糟糕的是，这些信息散落在各处。

另一种方案是在自动化脚本中支持，buildr 让这个工作变得很简单。

配置信息

使用 buildr，配置信息可以放到一个名为 profile.yaml 的文件里，下面是一个例子：

```
development:
db:
url: jdbc:mysql://localhost/killer_development
    driver: com.mysql.jdbc.Driver
    username: root
    password:
    jar: mysql:mysql-connector-java:jar:5.1.14
production:
db:
url: jdbc:mysql://deployment.env/killer_production
    driver: com.mysql.jdbc.Driver
    username: root
    password: ki1152
    jar: mysql:mysql-connector-java:jar:5.1.14
```

我们看到，针对不同的环境，有不同的数据库配置，在 buildfile 里可以这样引用这些配置：

```
db_settings = Buildr.settings.profile['db']
```

随后，我们就可以使用这个配置，比如生成一个配置文件：

```
task :config do
    CONFIG_PROPERTIES = <<EOF
jdbc.driverClassName= #{db_settings['driver']}
    jdbc.url=#{db_settings['url']}
jdbc.username=#{db_settings['username']}
jdbc.password=#{db_settings['password']}
    EOF
File.open('config.properties'), "w") do |f|
f.writeconfig
    end
end
```

有了这样的基础，只要我们指定不同的环境就会产生不同的配置。

系统组件

在 buildr 里，有一个叫做 ENV['BUILDR_ENV']的变量，这是 buildr 内置的一个变量，通过它，我们可以获得当前环境的名字，在这个例子里，它可以是 development 或是 production。

有了这个变量，我们可以进行更加深度的配置，比如，在测试环境下，我们可以采用一些假的实现，让整个系统运行的更快。

下面是一个例子，有一个搜索组件的配置文件，在生产环境下，它会采用真实的搜索引擎实现，而在开发环境时，它只是一个简单内存实现。我们把不同环境的实现放到不同的配置文件里。

```
    <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="searcher" class="com.killer.SuperSearcher"/>
    </beans>
```

(searcher.production.xml)

```
    <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="searcher" class="com.killer.InMemorySearcher"/>
    </beans>
```

要使用这个组件只要引用 searcher.xml，把 searcher 拿过来用就好了：

```
    <import resource="searcher-context.xml"/>
```

```
    ...
<property name="searcher" ref="searcher"/>
```

接下来，用一个 task 就可以处理这些差异：

```
task :searcher do
cp "searcher.#{ENV['BUILDR_ENV']}.xml", "searcher.xml"
end
```

其实，实现这一点真正困难的并不在于配置文件，而在于这些组件的设计。只有识别出这些组件，把它们独立出来，才会可能根据不同的环境进行配置。没有恰当的抽象，自动化脚本也无能为力。

有了上面准备的基础，我们可以通“-e”这个选项，在命令行中指定我们的环境。下面的命令会让我们得到 production 的配置，不论是配置信息，还是系统组件。：

```
buildr -e production
```

静态检查

在团队开发中，统一的代码风格很重要。即便我们很认真，疏忽也在所难免。Java 世界里，checkstyle 常常扮演了代码检察官角色。

坏消息是，buildr 并不提供对 checkstyle 的直接支持。好消息是，buildr 可以集成 ant task，所以，我们通过 ant task 集成 checkstyle。

```
CHECKSTYLE = transitive('checkstyle:checkstyle:jar:4.4')
task :checkstyle do
    begin
        ant('checkstyle') do |ant|
rm_rf 'reports/checkstyle_report.xml'
mkdir_p 'reports'
ant.taskdef :resource=>"checkstyletask.properties",
                    :classpath=>Buildr.artifacts(CHECKSTYLE)
              .each(&:invoke).map(&:name).join(File::PATH_SEPARATOR)
ant.checkstyle :config=>"tasks/checkstyle_checks.xml",
                       :maxWarnings=>0 do
ant.formatter :type=>'plain'
ant.formatter :type=>'xml',
                          :toFile=>"reports/checkstyle_report.xml"
ant.property :key=>'javadoc.method.scope', :value=>'public'
ant.property :key=>'javadoc.type.scope', :value=>'package'
ant.property :key=>'javadoc.var.scope', :value=>'package'
ant.property :key=>'javadoc.lazy', :value=>'false'
ant.property :key=>'checkstyle.cache.file',
                         :value=>'target/checkstyle.cache.src'
ant.property :key=>'checkstyle.header.file',
                         :value=>'buildconf/LICENSE.txt'
ant.fileset :dir=>"domain/src/main/java",
                        :includes=>'**/*.java',
ant.fileset :dir=>"web/src/main/java",
```

```
                                :includes=>'**/*.java'
                end
    ant.xslt :in=>'reports/checkstyle_report.xml',
                        :out=>'reports/checkstyle_report.html',
                        :style=>'tasks/checkstyle-noframes.xsl'
            end
        end
    end
```

这里的绝大多数内容 checkstyle 的 ant task 介绍都可以很容易了解到，唯一需要注意的点在于 checkstyle 任务的 maxWarnings 属性。

maxWarning 表示可以容忍的最大警告数，也就是说，当 checkstyle 检查的警告数大于这个数字时，构建就会失败。在这里，我们将其设置为 0，换句话说，我们不接受任何警告。

在实践中，对低级错误容忍度越低的团队，代码质量往往也会越高。

像这种独立性很强的任务，我们通常会把放到一个独立的文件中。在《文件布局》一节，我们曾提及 tasks 目录，其中存放的就是自定义任务，buildr 在启动时会自动加载该目录下的这些任务。

如果熟悉 rake，你会发现上面的 task 就是一个标准的 rake task。实际上，buildr 是集了多个构建工具的本领于一身的。把上面的代码放到 checkstyle.rake，然后放到 tasks 目录下。运行下面的命令，就可以对代码执行静态检查：

```
    buildrcheckstyle
```

测试覆盖率

测试覆盖率这东西，高了不代表有多好，但低了肯定是有问题。

当我们把测试覆盖率限制为 100%，其结果是全部内容都会有测试覆盖。坚持这样的标准，在做重构的时候，可以比较放心。

工具支持

buildr 缺省支持一个测试覆盖率工具——cobertura，所以，一切做起来很简单。

```
    require 'buildr/java/cobertura'
    cobertura.check.branch_rate = 100
    cobertura.check.line_rate = 100
```

（buildfile）

在命令行里运行如下命令就可以运行测试，进行检查，生成报告：

```
buildrcobertura:htmlcobertura:check
```

如果覆盖率不达标，构建就会失败。这时，我们可以通过它生成的报告来看，到底是哪里没有覆盖。报告位于

```
reports/cobertura/html/index.html
```

例外情况

真的是所有代码都会在这个 100%的监控之下吗？不一定。有一些代码只是为了调用一个特定的 API，这样的代码是否 100%意义不大，这样的测试本质上是在 API 写测试，而非自己的代码逻辑。所以，这种代码会被排除在外。

```
cobertura.exclude /.*.integration.*/
```

但是，这样的代码同样会有相应的集成测试，只是不在单元测试的层面。一个基本的原则是，被排除的代码要尽可能少。

提交脚本

作为一个专业程序员，我们应该保证自己提交的代码不会对代码库造成破坏。但怎么才算是不破坏呢？也许我们要编译、运行单元测试、打包，也许我们还要 进行静态检查，查看测试覆盖率，也许还要进行更多的检查，运行各种各样的测试。有了前面的基础，我们可以做一个 task，把所有这些都依赖上去。

好，有了这个 task，要提交代码，我们会怎么做呢？

- 如果用的 git 这样的分布式版本管理系统，现在本地提交代码。
- 如果有新增文件，需要把文件纳入版本控制之中。
- 从远端代码库更新出代码，如果有需要冲突，需要合并代码
- 运行 task，进行检查和测试
- 运行成功，则提交代码

显然，这是一个说多不多，说少也不少的操作，对于这种繁琐的操作，如果能够自动化，自然是最好的选择。下面就是一段这样的脚本，这里用的到版本控制系统是 git。

首先看到的是版本控制系统部分：

```
namespace :git do
def sys(cmd)
```

```
            puts cmd
            raise "System execution failed!" unless system(cmd)
        end
    defget_info(name, prompt)
            begin
                value = File.open(".#{name}") { |f| f.read }.strip
                rescue
                    value = ""
                end
                prompt += " (#{value})" unless value.empty?
    new_value = Readline::readline("[#{prompt}]: ").strip
                value = new_value unless new_value.empty?
    File.open(".#{name}", "w") { |f| f.write(value) }
                value
            end
    def commit
    dev_name = get_info('pair', 'Pair')
    story_number = get_info('story', 'Story #')
            comment = get_info('comment', 'Comment')
    commit_cmd = %Q(git commit -am "#{dev_name} - KILLER-#{story_number} -
#{comment}")
            sys(commit_cmd)
        end
    def add(files)
    files.each { |file| sys("git add #{file}") }
        end
    defadd_files(files)
            puts "Add the following new files:\n#\t#{files.join("\n#\t")}\n"
            reply = Readline::readline("[Y/N]")
            return add(files) if reply.strip.downcase.start_with?('y')
            raise 'new files should be added before commit'
        end
        task :add do
            files = `git status -s | awk '/\\?\\?/ {print $2}'`
            files = files.split("\n")
    add_files(files) if files.size> 0
        end
    defnothing_to_commit?
            `git status -s`.empty?
        end
        task :pull do
            sys('git pull')
        end
        task :push do
            sys('git push')
        end
            task :status do
            sys('git status')
        end
        task   :commit => :add do
            commit unless nothing_to_commit?
        end
    end
```

这里，我们看到了 git 的一些基础操作，比如 pull、push、commit 等等，并且，我们在基础操作之上进行了封装，让使用更便捷。比如：

- 在提交之前，根据当前的状态，确定是否要添加文件。
- 如果代码曾经手工提交过，则继续。
- 提示之前，提示开发人员填写名字、开发的 Story 号，以及相应的注释。

之所以在这里还要填写名字，而不仅仅是利用 git 自有的用户名，因为在开发的过程中，我们可能是结对开发，显然一个用户名是不够的。

有了这个基础，我们继续向前，创建一个 task，把提交之前要做的事情都放在这里：

```
    task :commit_build => [:clean, :artifacts, :checkstyle,
        "cobertura:html", "killer:domain:cobertura:check",
        "killer:web:cobertura:check"]
```

以上面这个 task 为例，实际上包括编译、测试、静态检查和测试覆盖率检查，我们可以根据自己的需要加入更多的东西。

接下来，把版本控制部分结合进去，就是我们的提交脚本了。

```
    task :commit => ["git:commit", "git:pull",     :commit_build, "git:push",
"git:status"]
```

有了这个基础，我们就可以提交代码了，而不用担心忘记了什么：

```
    buildr commit
```

在执行过程中，任何一步失败都会让整个提交过程停下来，比如，当 pull 代码产生冲突，或是运行测试之后，push 代码时，我们发现又有人提交，提交过程就会停止，错误的代码是不会提交的。

数据库迁移

数据库脚本通常不像代码那样受人重视，但在实际的发布过程中，它常常把我们弄得焦头烂额。

dbdeploy 为我们提供了一种管理数据迁移的方式，buildr 没有提供对 dbdeploy 直接的支持，但如我们之前所见，ant task 可以帮我们打造一条直通之路。

```
    namespace :db do
        DB_SETTINGS = Buildr.settings.profile['db']
    defdbdeploy(options)
            ant('dbdeploy') do |ant|
    ant.taskdef(:name => "dbdeploy",
                        :classname => "com.dbdeploy.AntTarget",
                        :classpath=>DBDEPLOY_CLASSPATH)
    ant.dbdeploy(options)
            end
        end
```

```
    task(:migrate) do |t, args|
dbdeploy(:driver => DB_SETTINGS['driver'],
                :url => DB_SETTINGS['url'],
                :userid => DB_SETTINGS['username'],
                :password => DB_SETTINGS['password'],
                :dir => "#{db_script_dir}/migration")
    end
en
```

这里，我们用到之前提及的管理配置的方式，配置信息存放在 profile.yaml 里。另外，数据库迁移文件存放在数据库脚本目录（db_script_dir）的 migration 子目录下。

有了这段 task，我们就可以进行数据库迁移了：

```
buildrdb:migrate
```

集成 Web 服务器

对于一个 Web 应用而言，仅仅测试打包还是不够的，我们还要把它部署到 Web 服务器上。这是一个基础任务，有了它，我们就可以使用一些 web 测试框架，对我们的系统进行验收测试。

通常在开发过程中，我们会选择一个部署起来很容易的 Web 服务器，在 Java 的世界里，jetty 往往扮演着这样的角色。当然，这样做的前提是，我们编写的是一个标准的 Java WebApp，没有用到特定于某个具体应用服务器的 API，换句话说，我们的 Web 应用是跨应用服务器的，事实上，这种做法是值得鼓励的。

使用特定于具体应用服务器的 API，其结果只能是与这个服务器产生耦合，而通常提供这个 API 的服务器对于开发并不那么友好，在其上部署的周期会很长，这无疑会大大降低开发效率。

一个好消息是，buildr 有很好的对 jetty 的集成。

```
require 'buildr/jetty'
define 'killer' do
    define 'web' do
        …
        task("jetty-deploy"=>[package(:war), jetty.use]) do |task|
jetty.deploy("http://localhost:8080", task.prerequisites.first)
        end
        task("jetty"=> ["jetty-deploy"]) do |task|
Readline::readline('[Type ENTER to stop Jetty]')
        end
    end
end
```

这里，我们启动了一个 jetty，启动之前，我们需要确保已经有了可以部署的 WAR，所以这

个部署任务要依赖于 package(:war)。最后的 jetty task 给了我们一个手工停止 Jetty 的机会。我们可以这样将启动它：

```
buildrkiller:web:jetty
```

验收测试

写好代码，完成部署，还少不了验收测试。

对于一个 Web 项目，我们可能会考虑采用 selenium 或是 waitr 之类的自动化测试框架进行测试。不过，通常直接用这些框架去写通常会把业务需求和测试实现细节混合起来，结果往往不如我们预期。

Cucumber 的出现给了我们一种分离业务需求和实现细节的方式。对我们这 个项目而言，不好的消息是，想到 Cucumber，出现在我们脑子里的是 Ruby 语言，好消息是 buildr 就是 Ruby 语言。下面就是一段脚本，把它放 到 buildfile 里，就可以运行 acceptance/features 目录下那堆 feature 文件了。

```
require 'cucumber'
require 'cucumber/rake/task'
define 'killer' do
    define 'web' do
        …
        Cucumber::Rake::Task.new(:acceptance => "jetty-deploy") do |t|
t.cucumber_opts = ["acceptance/features"]
        end
    end
end
```

有了它，我们就可以运行我们的验收测试了：

```
buildrkiller:web:acceptance
```

路还长

至此，我们已经展示了一个基本的自动化脚本。正如我们所见，作为地基的自动化脚本，仅仅有编译、测试、打包这些基本操作是远远不够的。即便是这里所列出的这些，也不过是一些通用的任务，每个项目都应该把项目内的繁琐操作自动化起来，比如，在我参与的实际项目中，我们会把部署到用户验收测试（User Acceptance Testing，简称 UAT）环境的过程自动化起来。

不过，除了脚本自身，为了让地基真正的发挥作用，还需要有一些实践与之配合，比如有了持续集成，提交脚本才好发挥最大的威力，比如有了合适的测试策略，开发人员才能在一个

合理的时间内完成上的本地构建。

软件开发永远都不是一个单点，只有各方各面联动起来，才会向着健康的方向前进，而所有一切的基础，就是自动化。

原文链接：http://www.infoq.com/cn/articles/zy-software-development-foundation

相关内容：

- 敏捷实践的秘密 II
- DevOps，一场更深入的敏捷文化变革
- 运用四色建模法进行领域分析
- 关于项目估算的微博讨论

特别专题

虚拟座谈：JavaScript 单元测试现状

作者 Dionysios G. Synodinos 译者 王瑜珩

不管使用什么编程语言和平台，编写单元测试都是被广为接受的技术实践，以便交付易于维护的代码。这对于 JavaScript 这样的动态语言尤为重要，目前已经有一些针对 JavaScript 的单元测试框架和库供大家选择。

InfoQ 与目前流行的一些 JavaScript 单元测试框架的作者进行了问答，问题主要针对他们的项目以及开发人员可以从中获得哪些好处。

参与者包括：

- QUnit 的作者 Jörn Zaefferer
- Jasmine 的作者 Davis Frank
- Jarvis 的作者 Tommy Montgomery
- jfUnit 的作者 Felipe Nascimento de Moura

InfoQ: 你的项目是做什么的，和其他 JavaScript 测试框架有什么不同？

Jörn Zaefferer :QUnit 是一个 JavaScript 单元测试框架，主要用于在浏览器中运行单元测试。虽然这个项目从属于 jQuery，但却不依赖于 jQuery，也不依赖于浏览器 DOM。因此你也可以在 node.js 或 Rhino 上使用。QUnit 很容易学习，你只需在 html 页面中包含两个文件，不需要安装或者构建任何其他东西。最短的测试集只需要一个 11 行的 html 文件。

Davis Frank: Jasmine 是一个 JavaScript 测试框架，目的是将 BDD 风格引入 JavaScript 测试之中。至于区别嘛，我们的目标是 BDD（相比标准的 TDD），因此我们尽力帮助开发人员编写比一般 xUnit 框架表达性更强，组织更好的代码。此外我们还力图减少依赖，这样你可以在 node.js 上使用 Jasmine，也可以在浏览器或移动程序中使用。

Tommy Montgomery: Jarvis 是一个 JavaScript 单元测试框架，API 风格基于.NET 平台上的 NUnit。与其他测试框架的区别包括堆栈打印，带颜色的字符串比较，以及可读性更好的 API。

Felipe Nascimento: jfUnit 的主要特点是它的加载方式和编写方式。它需要更少的输入，这让你编写单元测试更容易，而且可以在开发环境、控制台或任何你需要的地方运行。

InfoQ: 能大概说一下项目的架构么？主要的组件有哪些？

Jörn Zaefferer: QUnit 的 API 非常直观：调用 test 方法，传入测试的名字和回调函数，在回调函数中编写测试代码，其中包含一些断言。QUnit 还提供一些选项来组织测试，例如含有 setup 和 teardown 的模块。在内部这些调用被映射为队列。根据执行环境的不同，QUnit 会在 window.load 或其他事件发生时（或者由用户触发）开始执行测试。当某个测试需要异步执行时，会等到测试执行结束后再执行下一条。测试的执行顺序是变化的：这是一个可选的功能，默认会根据上一次执行的结果重新排列执行顺序，但结果还是按照之前的顺序显示。如果你选择隐藏已经通过的测试，就可以重新运行所有失败的测试，无需等待将全部测试都运行一遍的时间，可以更快知道这些测试是否被修复。

Davis Frank: 我们尽量保持简洁，Jasmine 的核心只有一个 JavaScript 文件，它提供了一种领域专用语言来编写测试。这种领域专用语言将测试组织为树状结构并执行。这个部分非常的简单。

在核心部分之上，还有一些用于在不同环境中显示测试报告的功能，以及方便管理源文件和测试文件的功能。Jasmine 可以在 Ruby on Rails 或任何 Web 框架中使用。还有些人用它在 node.js 环境中做开发，任何使用 JavaScript 的地方，Jasmine 都应该能够使用，如果不能我们就要让它变成可能。

Tommy Montgomery: 和其他 xUnit 框架一样，Jarvis 也是基于约束的。每个断言其实都在背后创建了一个约束对象，用来比较期待值和实际值。例如，当运行 Assert.that("foo", Is.equalTo("bar"));时，会对左边的“foo”和右边的“bar”做等值比较。这个 API（从 NUnit 借鉴而来）非常简单易读，我认为这在单元测试中是非常重要的。在使用方面，主要的组件是 Assert 对象、Is 对象和执行器。Assert 暴露一组函数用于对值进行断言，Is 则体现了 Jarvis 基于约束的架构。你可以通过调用 Jarvis.run()并传入测试函数来运行测试。Jarvis 的测试报表是可配置的，内置两种报表方式：一种针对控制台（比如 FireFox 中的 Firebug），另一种则生成 HTML。

Felipe Nascimento: jfUnit 是一个全局对象，你可以通过它来添加或删除测试。写好测试以后，这些测试并不会自动执行，你需要显式调用 jfUnit 的执行方法，在控制台或地址栏里都可以。我建议你在开发环境和测试环境中自动加载（测试库本身和测试脚本），在生产环境中则要删除。这么做可以让你在任何地方执行测试，并马上在屏幕上得到结果。

InfoQ: 开发人员刚开始使用你的框架时，怎么使用最好，典型的流程是怎样的？

Jörn Zaefferer: 网上有一篇很棒的文章 introducing QUnit on Script Junkie - 覆盖了对 QUnit 的基本介绍、大多数的 API，以及一些让你可以更高效的进行 TDD 的高级特性。

Davis Frank: 有很多截屏视频和教程。这个 Railscasts 的非常棒，PeepCode 的 CoffeeScript cast 也不错，虽然是关于 CoffeeScript 的，但 Geoffrey 同时也展示了如果使用 Jasmine 来做 TDD。

Tommy Montgomery: 最好的办法是浏览 Jarvis 的网站，看看示例代码，在网站上直接执行一些测试，搞明白测试的写法以及测试报告的样式。Jarvis 本身的测试也是用 Jarvis 编写的，你可以在这里查看测试结果。之后，你就可以下载整个框架，然后在项目中使用了。编写测试的具体流程取决于开发人员，Jarvis 对此没有任何假设和约束。由于主要的测试运行器是基于 HTML 的，因此打开浏览器是必须的。我个人的工作流程是编写代码，然后编写测试，刷新浏览器，查看测试结果，如此往复。

Felipe Nascimento: 你可以在这里找到一些详细的信息。基本上，开发人员需要加载这个库，然后使用 jfUnit 对象来编写测试。测试编写完毕后，执行 jfUnit.run()来查看测试结果。你还可以用 jfUnit.config({...})来设置一些参数或者环境变量。唯一我觉得需要特别提出的是，测试框架使用 body 元素来构建测试报表，因此在页面上必须要有 body 元素。

InfoQ: 哪些特性是你希望将来能够实现的？对于 JavaScript 测试框架，未来的发展方向是怎么样的？

Jörn Zaefferer: 目前最重要的是能与命令行做更好的集成，比如在 node.js 里，你需要一个 qunit.js 之外的脚本，我们会通过 npm 发布。除此之外我们会集中精力改进现有的功能。可能最后会有一个 BDD 风格的包装，比如 Pavlov。那时想用 QUnit 但是又喜欢 BDD 的人可以联合使用 Pavlov 与 QUnit。最后，不算是代码上的功能，qunitjs.com 会在年内上线。

Davis Frank: 在 Pivotal Tracker 上我们有一个公开的特性计划表，你可以看到我们接下来的发布会包含哪些内容。我们欢迎来自社区的任何贡献者，但同时我们也很挑剔。我们希望能够保持 Jasmine 精炼和紧凑。让你的补丁和特性足够简洁，而且要包含相应的测试。毕竟，Jasmine 是一个测试框架，所以你最好用 TDD 的方式。

Tommy Montgomery: 目前我在努力支持 nodejs 以便可以在服务器端编程中使用。以后可能会有一个 ant task 和一个控制台的执行程序。JavaScript 测试框架对浏览器的依赖阻碍了自身（大多数情况下）。因此摆脱浏览器单独使用对所有 JavaScript 单元测试框架都是必须的。JavaScript 测试框架会更好的填补服务器端与客户端测试之间空白。

Felipe Nascimento:我非常喜欢 JavaScript，我认为这种语言将会被大多数开发人员所使用。我喜欢 JavaScript 的表达方式，新工具以及新技术。我是 BrazilJS（巴西 JavaScript 协会）的一员，我希望 JavaScript 拥有美好的前景。我很期待开发新功能，使这个测试工具将来可以让测试 Ajax 更简单，让用户编写测试更简单，让测试图形界面比如填写表单、提交表单、点击按钮（目前已经可以实现，但是比较难写）等更简单。我还想说，如果你是一个开发人员，想要贡献你的一分力量，请告诉我。请将你的批评、建议、想法、缺陷报告或者代码通过我的 github 帐户告诉我。

座谈嘉宾

Jörn Zaefferer 居住在德国科隆，是一个自由职业的 Web 开发人员、咨询师和培训师。他将 jQuery 的测试套件演化为 JavaScript 测试框架 QUnit，并且一直维护。他还创建并维护着大量流行的 jQuery 插件，作为 jQuery UI 开发组长，他关注最多的是新插件、新部件和新工具的开发。

Davis Frank 生活在北加利福尼亚，他是一位软件工程师、父亲和棒球爱好者。目前他是 Pivotal Labs 的软件工程师和经理。业余时间，他喜欢和家人在一起，并确保家里的网络保持 99.9%畅通。

Tommy Montgomery 拥有数学及计算机学士学位。早期工作于 LAMP 项目中，去年开始将主要精力放在.NET 项目上。PHP、C#和 JavaScript 是他最感兴趣的编程语言。近期参与的项目包括 Sunlight - 客户端语法高亮程序、Jarvis 和 Butterfly - 用.NET 和 JavaScript 编写的 markup 到 html 的转换程序。

Felipe Nascimento de Moura 生活在巴西，是一位资深开发人员与系统分析师。他拥有 7 年 Web 开发经验，是很多开源项目的发起者，例如 theWebMind.org、PHPDevBar，以及 jfUnit，还是巴西 JavaScript 协会 BrazilJS 的组织者之一。

原文链接：http://www.infoq.com/cn/articles/javascript-unit-testing

相关内容：

- ASP.NET MVC 测试方法与技巧
- VS11 将拥有更好的单元测试工具和 Fakes 框架
- NUnit Action 特性简化单元测试编写
- 如何进行高效的 Rails 单元测试、颠覆者生存：互联网产品测试观点

特别专题

一个前端工程师眼里的 NodeJS

作者 田永强

JavaScript 单线程的误解

在我接触 JavaScript(无论浏览器还是 NodeJS)的时间里，总是遇到有朋友有多线程的需求。而在 NodeJS 方面，有朋友甚至直接说到，NodeJS 是单线程的，无法很好的利用多核 CPU。

诚然，在前端的浏览器中，由于前端的 JavaScript 与 UI 占据同一线程，执行 JavaScript 确实为 UI 响应造成了一定程度上的麻烦。但是，除非用到超大的循环语句执行 JavaScript，或是用阻塞式的 Ajax，或是太过频繁的定时器执行外，JavaScript 并没有给前端应用带来明显的问题，所以也很少有朋友抱怨 JavaScript 是单线程而不能很好利用多核 CPU 的问题，因为没有因此出现性能瓶颈。

但是，我们可以用 Ajax 和 Web Worker 回应这个误解。当 Ajax 请求发送之后，除非是同步请求，否则其余的 JavaScript 代码会很快被执行到。在 Ajax 发送完成，直到接收到响应的这段时间里，这个网络请求并不会阻塞 JavaScript 的执行，而网络请求已经发生，这是必然的事。那么，答案就很明显了，JavaScript 确实是执行在单线程上的，但是，整个 Web 应用执行的宿主（浏览器）并非以单线程的方式在执行。而 Web Worker 的产生，就是直接为了解决 JavaScript 与 UI 占用同一线程造成的 UI 响应问题的，它能新开一条线程去执行 JavaScript。

同理，NodeJS 中的 JavaScript 也确实是在单线程上执行，但是作为宿主的 NodeJS，它本身并非是单线程的，NodeJS 在 I/O 方面有动用到一小部分额外的线程协助实现异步。程序员没有机会直接创建线程，这也是有的同学想当然的认为 NodeJS 的单线程无法很好的利用多核 CPU 的原因，他们甚至会说，难以想象由多人一起协作开发一个单线程的程序。

NodeJS 封装了内部的异步实现后，导致程序员无法直接操作线程，也就造成所有的业务逻辑运算都会丢到 JavaScript 的执行线程上，这也就意味着，在高并发请求的时候，I/O 的问题是很好的解决了，但是所有的业务逻辑运算积少成多地都运行在 JavaScript 线程上，形成了一条拥挤的 JavaScript 运算线程。NodeJS 的弱点在这个时候会暴露出来，单线程执行运算形

成的瓶颈，拖慢了 I/O 的效率。这大概可以算得上是密集运算情况下无法很好利用多核 CPU 的缺点。这条拥挤的 JavaScript 线程，给 I/O 形成了性能上限。

但是，事情又并非绝对的。回到前端浏览器中，为了解决线程拥挤的情况，Web Worker 应运而生。而同样，Node 也提供了 child_process.fork 来创建 Node 的子进程。在一个 Node 进程就能很好的解决密集 I/O 的情况下，fork 出来的其余 Node 子进程可以当作常驻服务来解决运算阻塞的问题（将运算分发到多个 Node 子进程中上去，与 Apache 创建多个子进程类似）。当然 child_process/Web Worker 的机制永远只能解决单台机器的问题，大的 Web 应用是不可能一台服务器就能完成所有的请求服务的。拜 NodeJS 在 I/O 上的优势，跨 OS 的多 Node 之间通信的是不算什么问题的。解决 NodeJS 的运算密集问题的答案其实也是非常简单的，就是将运算分发到多个 CPU 上。请参考文章后的 multi-node 的性能测试，可以看到在多 Node 进程的情景下，响应请求的速度被大幅度提高（感谢 CNode 社区的 snoopy 友情测试）。

在文章的写作过程中，Node 最新发布的 0.6.0 版本，新增了 cluster 模块。该模块的作用是可以通过 fork 的方式创建出多个子进程实例，这些实例会自动共享相同的侦听端口。你可以根据当前计算机上的 CPU 数量来创建相应的实例数，以此达到分发请求，充分利用 CPU 的目的。详情请参阅 **Error! Hyperlink reference not valid.**。在之前的解决运算密集问题中，工程师需要 multi-node 这样的库或者其他方案去手动分发请求，在 cluster 模块的支持下，可以释放掉工程师在解决此问题上的大部分精力。

事件式编程

延续上一节的讨论。我们知道 NodeJS/JavaScript 具有异步的特性，从 NodeJS 的 API 设计中可以看出来，任何涉及 I/O 的操作，几乎都被设计成事件回调的形式，且大多数的类都继承自 EventEmitter。这么做的好处有两个，一个是充分利用无阻塞 I/O 的特性，提高性能；另一个好处则是封装了底层的线程细节，通过事件消息留出业务的关注点给编程者，从而不用关注多线程编程里牵扯到的诸多技术细节。

从现实的角度而言，事件式编程也更贴合现实。举一个业务场景为例：家庭主妇在家中准备中餐，她需要完成两道菜，一道拌黄瓜，一道西红柿蛋汤。以 PHP 为例，家庭主妇会先做完拌黄瓜，接着完成西红柿蛋汤，是以顺序/串行执行的。但是现在突然出了一点意外，凉拌黄瓜需要的酱油用光了，需要她儿子出门帮她买酱油回来。那么 PHP 家庭主妇在叫她儿子出门打酱油的这段时间都是属于等待状态的，直到酱油买回来，才会继续下一道菜的制作。那么，在 NodeJS 的家庭主妇又会是怎样一个场景呢，很明显，在等待儿子打酱油回来的时

间里，她可以暂停凉拌黄瓜的制作，而直接进行西红柿蛋汤的过程，儿子打完酱油回来，继续完成她的凉拌黄瓜。没有浪费掉等待的时间。实例伪代码如下：

```
var mother = new People();
var child = new People();
child.buySoy(function (soy) {
mother.cook("cucumber", soy);
});
mother.cook("tomato");
```

接下来，将上面这段代码转换为基于事件/任务异步模式的代码：

```
var proxy = new EventProxy();
var mother = new People();
proxy.bind("cook_cucumber", function (soy) {
mother.cook("cucumber", soy);
});
proxy.bind("cook_tomato", function () {
mother.cook("tomato");
});
var child = new People();
child.buySoy(function (soy) {
proxy.trigger("cucumber", soy);
});
proxy.trigger("tomato");
```

代码量多了很多，但是业务逻辑点都是很清楚的：通过 bind 方法预定义了 cook_cucumber 和 cook_tomato 两个任务。这里的 bind 方法可以认为是 await 的消息式实现，需要第一个参数来标识该任务的名字，流程在执行的过程中产生的消息会触发这些任务执行。可以看出，事件式编程中，用户只需要关注它所需要的几个业务事件点就可以，中间的等待都由底层为你调配好了。这里的代码只是举例事件/任务异步模式而用，在简单的场景中，第一段代码即可。做 NodeJS 的编程，会更感觉是在做现实的业务场景设计和任务调度，没有顺序保证，程序结构更像是一个状态机。

个人觉得在事件式编程中，程序员需要转换一下思维，才能接受和发挥好这种异步/无阻塞的优势。同样，这种事件式编程带来的一个问题就在于业务逻辑是松散和碎片式的。这对习惯了顺序式，Promise 式编程的同学而言，接受它是比较痛苦的事情，而且这种散布的业务逻辑对于非一开始就清楚设计的人而言，阅读存在相当大的问题。

我提到事件式编程更贴近于现实生活，是更自然的，所以这种编程风格也导致你的代码跟你的生活一样，是一件复杂的事情。幸运的是，自己的生活要自己去面对，对于一个项目而言，并不需要每个人都去设计整个大业务逻辑，对于架构师而言，业务逻辑是明了的，借助事件式编程带来的业务逻辑松耦合的好处，在设定大框架后，将业务逻辑划分为适当的粒度，对每一个实现业务点的程序员而言，并没有这个痛苦存在。二八原则在这个地方非常有效。

深度嵌套回调问题

JavaScript/NodeJS 对单个异步事件的处理十分容易，但容易出现问题出现的地方是“多个异步事件之间的结果协作”。以 NodeJS 服务端渲染页面为例，渲染需要数据，模板，本地化资源文件，这三个部分都是要通过异步来获取的，原生代码的写法会导致嵌套，因为只有这样才能保证渲染的时候数据，模板，本地化资源都已经获取到了。但问题是，这三个步骤之间实际是无耦合的，却因为原生代码没有 promise 的机制，将可以并行执行（充分利用无阻塞 I/O）的步骤，变成串行执行的过程，直接降低了性能。代码如下：

```
var render = function (template, data) {
    _.template(template, data);
};
$.get("template", function (template) { // something
    $.get("data", function (data) { // something
        $.get("l10n", function (l10n) { // something
            render(template, data);
        });
    });
});
```

面对这样的代码，许多工程师都表示不爽。这个弱点也形成了对 NodeJS 推广的一个不大不小的障碍。对于追求性能和维护性的同学，肯定不满足于以上的做法。本人对于 JavaScript 的事件和回调都略有偏爱，并且认为事件，回调，并行，松耦合是可以达成一致的。以下一段代码是用 **Error! Hyperlink reference not valid.**实现的：

```
var proxy = new EventProxy();
var render = function (template, data, l10n) {
    _.template(template, data);
};
proxy.assign("template", "data", "l10n", render);
$.get("template", function (template) { // something
proxy.trigger("template", template);
});
$.get("data", function (data) { // something
proxy.trigger("data", data);
});
$.get("l10n", function (l10n) { // something
proxy.trigger("l10n", l10n);
});
```

代码量看起来比原生实现略多，但是从逻辑而言十分清晰。模板、数据、本地化资源并行获取，性能上的提高不言而喻，assign 方法充分利用了事件机制来保证最终结果的正确性。在事件，回调，并行，松耦合几个点上都达到期望的要求。

关于更多 EventProxy 的细节可参考其 Error! Hyperlink reference not valid.。

深度回调问题的延伸

EventProxy 解决深度回调的方式完全基于事件机制，这需要建立在事件式编程的认同上，那么必然也存在对事件式编程不认同的同学，而且习惯顺序式，promise 式，向其推广 bind/trigger 模式实在难以被他们接受。**Error! Hyperlink reference not valid.**和 **Error! Hyperlink reference not valid.**是目前比较成熟的同步式编程的解决方案。可以通过同步式的思维来进行编程，最终执行的代码是通过编译后的目标代码，以此通过工具来协助用户转变思维。

结语

对于优秀的东西，我们不能因为其表面的瑕疵而弃之不用，总会有折衷的方案来满足需求。NodeJS 在实时性方面的功效有目共睹，即便会有一些明显的缺点，但是随着一些解决方案的出现，相信没有什么可以挡住其前进的脚步。

附录（多核环境下的并发测试）

服务器环境：

- 网络环境：内网
- 压力测试服务器：
- 服务器系统：Linux 2.6.18
- 服务器配置：Intel(R) Xeon(TM) CPU 3.40GHz 4 CPUS
- 内存：6GB
- NodeJS 版本: v0.4.12

客户端测试环境：

- 发包工具：apache 2.2.19 自带的 ab 测试工具
- 服务器系统：Linux 2.6.18
- 服务器配置：Pentium(R) Dual-Core CPU E5800 @ 3.20GHz 2CPUS
- 内存：1GB

单线程 Node 代码：

```
var http = require('http');
```

```
var server = http.createServer(function (request, response) {
var j = 0;
    for (vari = 0; i&lt; 100000; i++) {
        j += 2 / 3;
    }
response.end(j + '');
});
server.listen(8881);
console.log('Server running at http://10.1.10.150:8881/');
```

四进程 Node 代码：

```
var http = require('http');
var server = http.createServer(function (request, response) {
var j = 0;
    for (vari = 0; i&lt; 100000; i++) {
        j += 2 / 3;
    }
response.end(j + '');
});
var nodes = require("./lib/multi-node").listen({
    port: 8883,
    nodes: 4
}, server);
console.log('Server running at http://10.1.10.150:8883/');
```

这里简单介绍一下 multi-node 这个插件，这个插件就是利用 require("child_process").spawn() 方法来创建多个子线程。由于浮点计算和字符串拼接都是比较耗 CPU 的运算，所以这里我们循环 10W 次，每次对 j 加上 0.66666。最后比较一下，开多子进程 node 到底比单进程 node 在 CPU 密集运算上快多少。

以下是测试结果：

Comm.	500/30	500/30	1000/30	1000/30	3000/30	3000/30
Type	单进程	多子进程	单进程	多子进程	单进程	多子进程
RPS	**2595**	**5597**	**2540**	**5509**	**2571**	**5560**
TPQ	0.38	0.18	0.39	0.19	0.39	0.18
80% REQ	72	65	102	85	157	142
Fail	0	0	0	0	0	0

说明：

- 单进程：只开一个 node.js 进程。
- 多子进程：开一个 node.js 进程，并且开 3 个它的子进程。
- 3000/30：代表命令 ./ab -c 3000 -t 30 http://10.1.10.150:8888/。3000 个客户端，最多发 30 秒，最多发 5W 个请求。
- RPS：代表每秒处理请求数，并发的主要指标。
- TPQ：每个请求处理的时间，单位毫秒。
- Fail：代表平均处理失败请求个数。
- 80% Req：代表 80%的请求在多少毫秒内返回。

从结果及图 1 ~ 3 上看：开多个子进程可以显著缓解 node.js 的 CPU 利用率不足的情况，提高 node.js 的 CPU 密集计算能力。

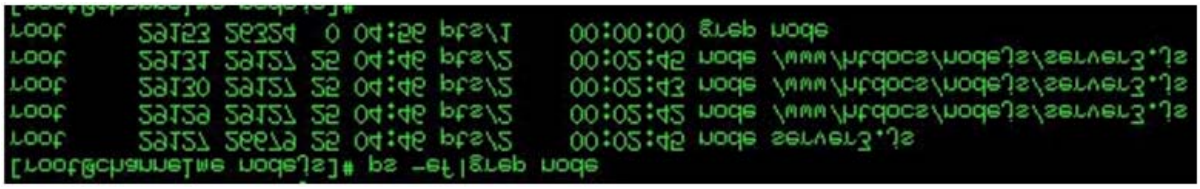
图 1：单个进程的 node.js 在压力测试下的情况，无法充分利用 4 核 CPU 的服务器性能。

```
top - 04:41:36 up 22 days, 23:17,  2 users,  load average: 0.82, 0.56, 0.25
Tasks: 1149 total,   3 running, 1146 sleeping,   0 stopped,   0 zombie
Cpu0  : 86.4%us,  4.5%sy,  0.0%ni,  6.8%id,  0.0%wa,  0.0%hi,  2.3%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni, 97.8%id,  0.0%wa,  0.0%hi,  0.0%si,  2.2%st
Cpu2  :  4.5%us, 13.6%sy,  0.0%ni, 81.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   5836800k total,  1998408k used,  3838392k free,   226696k buffers
Swap:  5537784k total,        0k used,  5537784k free,   840276k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
29108 root      25   0 65016  30m 5064 R 95.8  0.5   2:17.85 node
29112 root      15   0  2968 1708  820 R 16.0  0.0   0:20.84 top
```

图 2：多进程 node，可以充分利用 4 核 CPU 榨干服务器的性能。

```
top - 04:51:12 up 22 days, 23:27,  2 users,  load average: 3.45, 2.30, 1.16
Tasks: 1152 total,   6 running, 1146 sleeping,   0 stopped,   0 zombie
Cpu0  : 87.9%us,  9.1%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  3.0%si,  0.0%st
Cpu1  : 87.5%us,  0.0%sy,  0.0%ni,  6.2%id,  0.0%wa,  0.0%hi,  6.2%si,  0.0%st
Cpu2  : 68.8%us, 31.2%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  : 81.8%us, 12.1%sy,  0.0%ni,  3.0%id,  0.0%wa,  0.0%hi,  3.0%si,  0.0%st
Mem:   5836800k total,  2075804k used,  3760996k free,   226700k buffers
Swap:  5537784k total,        0k used,  5537784k free,   840276k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
29127 root      16   0 62388  27m 4508 R 104.0  0.5   2:38.72 node
29129 root      16   0 63240  28m 4516 R 100.9  0.5   2:36.01 node
29131 root      16   0 63692  29m 5052 R 91.8  0.5   2:39.21 node
29130 root      16   0 63680  29m 5052 R 73.4  0.5   2:37.31 node
29141 root      15   0  2968 1712  820 R 24.5  0.0   0:05.34 top
```

图 3：多子进程截图，可以看到一共跑了 4 个进程。



关于作者

田永强，前端工程师，就职于 SAP，从事 Mobile Web App 方面的研发工作，对 NodeJS 持有高度的热情，寄望打通前端 JavaScript 与 NodeJS 的隔阂，将 NodeJS 引荐给更多的前端工程师。

原文链接：http://www.infoq.com/cn/articles/nodejs-in-front-end-engineer-view

相关内容：

- 深入浅出 Node.js（五）：初探 Node.js 的异步 I/O 实现
- Travis CI 宣布支持 Java，并计划推出 Travis Pro
- 关注 Node.js 的 C++移植版 node.native
- 百度技术沙龙第 21 期回顾：交互设计在 Apps 中的开发实践（含资料下载）
- 当 Shell 遇上了 NodeJS

又拍云存储

选择又拍云存储

= 存储 + CDN

-70% 成本缩减

+300% 速度提升

又拍云存储服务以较小的成本解决图片优化这一关键性能问题，缩减页面加载时间，提 升转化率，而且还方便易用，这项服务值得推荐。
——丁香园 冯大辉

又拍云不仅提供了稳定的存储、高速的分发，也在图片托管方面提供了更多专业特性。集简单和强大于一身，值得推荐。
——猛买网 张磊

推荐文章 | Articles

Cookie 安全漫谈

作者 殷钧钧

在 Web 应用中，Cookie 很容易成为安全问题的一部分。从以往的经验来看，对 Cookie 在开发过程中的使用，很多开发团队并没有形成共识或者一定的规范，这也使得很多应用中的 Cookie 成为潜在的易受攻击点。在给 Web 应用做安全架构评审（Security architecture review）的时候，我通常会问设计人员以下几个问题：

你的应用中，有使用 JavaScript 来操作客户端 Cookie 吗？如果有，那么是否必须使用 JavaScript 才能完成此应用场景？如果没有，你的 Cookie 允许 JavaScript 来访问吗？

你的网站（可能包含多个 Web 应用）中，对于 Cookie 的域（Domain）和路径（Path）设置是如何制定策略的？为何这样划分？

在有 SSL 的应用中，你的 Cookie 是否可以在 HTTP 请求和 HTTPS 请求中通用？

在实际的应用场景中，Cookie 被用来做得最多的一件事是保持身份认证的服务端状态。这种保持可能是基于会话（Session）的，也有可能是持久性的。不管哪一种，身份认证 Cookie 中包含的服务端票据（Ticket）一旦泄露，那么服务端将很难区分带有此票据的用户请求是来自于真实的用户，或者是来自恶意的攻击者。在实际案例中，造成 Cookie 泄露最多的途径，是通过跨站脚本（XSS, Cross Site Script）漏洞。攻击者可以通过一小段 JavaScript 代码，偷窃到代表用户身份的重要的 Cookie 标示。由于跨站脚本漏洞是如此的普遍（不要以为简单的 HTML Encode 就可以避免被跨站，跨站是一门很深的学问，以至于在业界衍生出一个专用的名词：跨站师），几乎每一个网站都无法避免，所以这种方式是实际攻防中被普遍使用的一种手段。

避免出现这种问题的首要秘诀就是尽所有的可能，给你的 Cookie 加上 HttpOnly 的标签。HttpOnly 的具体使用不在本文的讨论范围内，否则作者略有骗 InfoQ 稿酬的嫌疑。一个大家所不太熟悉的事实是，HttpOnly 是由微软在 2000 年 IE6 Sp1 中率先发明并予以支持的。截止现在，HttpOnly 仍然只是一个厂商标准，但是在过去的十余年中，它得到了众多浏览器的广泛支持。

下表是OWASP整理的关于主流浏览器对HttpOnly支持情况的一个总结。从表中可以看出，目前主流的浏览器，除了 Android 之外，几乎都无一例外对这一属性予以了支持。

当然对于中国开发者来说，需要考虑的问题更加复杂一些：在这个神奇的地方，有大量的用户使用的浏览器并不在以下的列表中，他们使用的是以下面浏览器中的一种或者数种为内核的“精装版”浏览器。这些浏览器厂商对于同源策略、HttpOnly Cookie、证书管理等安全规范的支持情况，有待于进一步调查。

Browser	Version	Prevents Reads	Prevents Writes
Microsoft Internet Explorer	10	Yes	Yes
Microsoft Internet Explorer	9	Yes	Yes
Microsoft Internet Explorer	8	Yes	Yes
Microsoft Internet Explorer	7	Yes	Yes
Microsoft Internet Explorer	6 (SP1)	Yes	No
Microsoft Internet Explorer	6 (fully patched)	Yes	Unknown
Mozilla Firefox	3.0.0.6+	Yes	Yes
Netscape	9.0b3	Yes	Yes

Navigator			
Opera	9.23	No	No
Opera	9.5	Yes	No
Opera	11	Yes	Unknown
Safari	3	No	No
Safari	5	Yes	Yes
iPhone (Safari)	iOS 4	Yes	Yes
Google's Chrome	Beta (initial public release)	Yes	No
Google's Chrome	12	Yes	Yes
Android	Android 2.3	Unknown	Unknown

在我看来，一个 Web 应用的每一个 Cookie 都应该带上 HttpOnly 的标签。我没有看到这个决定可能带来的负面作用，如果一定要说有的话，那么就是你的应用将不能再通过 JavaScript 来读写 Cookie 了。可是这真有必要吗？个人认为，JavaScript 操作 Cookie 是一种不正常的做法；可以用 JavaScript 操作 Cookie 完成的功能，一样可以用服务端响应 Http 头设置 Cookie 来完成。相反，设置所有的 Cookie 为 HttpOnly 带来的巨大好处则非常明显：尽管你需要继续修复 XSS 漏洞，但是这些漏洞至少暂时不会让你的用户遭受大规模的账户损失。

关于 Cookie 的第二个话题是域设置。

浏览器在选择发送哪些本地 Cookie 到本次请求的服务端时，有一系列的比较和甄别。这些甄别中最重要的部分是 Domain 和 Path 的吻合。Domain 形如.abc.com 的 Cookie，会被发送给所有 abc.com 在 80 端口上的子域请求。但是反之则不行，这就是 Cookie 的域匹配（domain match）原则。

在一个大型 Web 站点中，往往有多个应用，生存在不同的子域名或路径下。这些应用之间由于共享同一个域名，所以往往可能会彼此有操作对方应用 Cookie 的能力。设计好 Cookie

的 Domain 和 Path 尤为重要。在实际设计工作中，最重要的一个安全原则就是：最小化授权。这意味着，你需要将自己的 Cookie 可被访问到的范围降至最低。应用之间传递数据和共享信息的解决方案非常多，而通过 Cookie 这种用户输入（User input）来共享数据，是最不安全的解决方案之一。

Cookie 另外一个不太常被使用的属性是 Secure. 这个属性启用时，浏览器仅仅会在 HTTPS 请求中向服务端发送 Cookie 内容。如果你的应用中有一处非常敏感的业务，比如登录或者付款，需要使用 HTTPS 来保证内容的传输安全；而在用户成功获得授权之后，获得的客户端身份 Cookie 如果没有设置为 Secure，那么很有可能会被非 HTTPS 页面中拿到，从而造成重要的身份泄露。所以，在你的 Web 站点中，如果使用了 SSL，那么你需要仔细检查在 SSL 的请求中返回的 Cookie 值，是否指定了 Secure 属性。

除此之外还值得特别指出的是，一些 Web 应用除了自己的程序代码中生成的 Cookie，往往还会从其他途径生成一些 Cookie。例如由 Web Server 或者应用容器自动生成的会话 Cookie，由第三方库或者框架生成的 Cookie 等等。这些都需要进行有针对性的加固处理。

几乎每个站点都难以离开 Cookie，但 Cookie 的使用因其貌似简单，而很容易被人轻视。重新审视应用中的 Cookie 代码，几乎只需要很小的代价就可以获得巨大的安全收益。

参考文章

- http://en.wikipedia.org/wiki/HTTP_cookie
- https://www.owasp.org/index.php/HttpOnly

关于作者

殷钧钧（Joey Yin），Web 开发者，某外企企业架构师。业余时间，他是一名白帽子，独立安全组织 OWASP 成员。主要关注的技术领域有应用安全、分布式系统架构、以及程序员的敏捷实践。个人博客：http://unclejoey.com。

原文链接：http://infoq.com/cn/articles/cookie-security

相关内容：

- 移动设备攻击和防护、从云消费者的角度谈云安全架构
- 2012.4.3 微博热报：云计算不忽悠、淘宝浏览器、大学还在教 C++吗、SEI 出版《内部威胁 CERT 指南》一书

推荐文章 | Articles

阅读者（十八）：编码的艺术

作者 张逸

这是一本关注编码细节的书。或许你会认为本书所讲皆为小道，诸如方法命名、变量定义、语句组织、任务分解等内容，俱是细枝末节，微不足道。然而，对于一个整体的软件系统而言，既需要宏观的架构决策、设计与指导原则，也必须重视微观的代码细节。正如作文，提纲主旨是文章的根与枝，但一词一句，也需精雕细作，才能立起文章的精气神。所谓“细节决定成败”，软件历史上，有许多影响深远的重大失败，其根由往往是编码细节出现了疏漏。

我一直坚持“代码即架构”的观点，正如小说需要角色来说话一般，软件系统的质量好坏，归根结底还是需要代码来告知。代码的优劣不仅直接决定了软件的质量，还将直接影响软件成本。Yourdon【1】和 Constantine【2】在著作 Structured Design 中认为：软件成本由开发成本与维护成本组成，而往往维护成本要远高于开发成本。这其中付出的主要成本就是由于理解代码和修改代码造成的。正如本书的书名 The Art of Readable Code 所表示的含义，好的代码常常是可阅读的，要做到这一点，则近似于一种艺术之美了。

与本书相似的一本书是 Robert C. Martin 的 Clean Code（中文版：代码整洁之道）。该书在业界已经得到了广泛赞誉。如果你还在为写出丑陋的代码而烦恼，必须阅读该书。它带给你的冲击，好似阿凡达那无与伦比的 3D 效果带给你感官上的震撼。在某种程度上，The Art of Readable Code 一书几乎可以与 Clean Code 比肩。或许本书在深度上与 Clean Code 相比还有所不及，但在内容广度上，却超过了 Clean Code。因为它关注编码本身，所以并不局限于某一种语言，而是列举了大量 C++、Python、JavaScript 和 Java 代码，涵盖了主流的静态语言和动态语言。这就使得本书的内容具有更强的普适性。

本书一共分为四部分。第一部分 Surface-Level Improvements 主要介绍了变量、方法和类的命名，以及如何编写好的注释。第二部分 Simplifying Loops And Logic 则将注意力转移到语句和表达式。第三部分 Reorganizing Your Code 则主要介绍了有关任务分解、职责分配和设计意图的内容。最后一部分是一些扩展话题，例如如何编写测试，提高代码可读性，并在最后一章给出了一个完整的案例，全面而又系统地结合案例回顾了全书所讲的主要

内容。

本书给出了许多改善编码质量的技巧，尤其它结合了大量真实案例，给出了具体的代码片段，并从正反两面对案例进行分析，这就使得作者的讲解不再流于空洞，既让人信服，又能帮助读者理解。

例如在讲解命名如何表达意图时，作者给出了 Google 代码中的一个反面教材（之所以作者能够给出 Google 代码的案例，是因为本书的两位作者 Dustin Boswell 和 Trevor Foucher 曾经或者正就职于 Google ）。例如在 Google 的一段代码中定义了一个宏，用于禁止“邪恶”的构造函数：

```
classClassName {
    private:
        DISALLOW_EVIL_CONSTRUCTORS(ClassName);
    public:
    …
};
```

宏的定义如下：

```
#define DISALLOW_EVIL_CONSTRUCTORS(ClassName) \
ClassName(constClassName&); \
void operator=(constClassName&);
```

这个宏禁止了构造函数和 Copy 构造函数（即=操作）。然而从宏的名称来看，这个含义是不明确的，会让读者认为仅仅禁用了构造函数。只需要改个名字，意图就可以变得更加清晰：

```
#defineDISALLOW_OPY_AND_ASSIGN(ClassName) [...]
```

书中各章的案例非常翔实，并且总是先给出糟糕的版本，逐步分析推导，最后给出好的实现，作为直观鲜明的对照。为避免读者陷入相对独立而散乱的小案例中，作者又专门独立出一章内容，讲解了一个完整的案例 Minute/Hour Counter。该案例从问题域的提出，到对接口的设计和实现，层层推进。作者从方法命名和注释开始，抽丝剥茧展开分析，先后给出了三个解决方案，渐进地对编码实现进行了改善，使之在性能、灵活性都有了很好的改观，类的职责更为清晰，代码结构简洁而又易于理解。最后，作者还给出了三个解决方案的比较，从代码行数、时间复杂度、内存消耗和准确率四个因素出发，全面权衡了各个解决方案的优劣，以此来印证作者在本书中一直推崇的编码技巧。

本书作者并不满足于通过文字和代码来彰显这些技巧的力量，书中附带的漫画插图起到了很好的辅助作用。如果将本书比作一盘精美的佳肴，这些漫画就起到了调料的作用。例如作者希望表达出名字误用所带来的危险时，给出了这样的一幅漫画：

图文并茂、案例翔实是本书的主要特色。在阅读本书时，代码从丑陋到美丽的蜕变让人振奋；但是，我们不能满足于结果的获得，而应该享受这个过程。我的建议是，在阅读时，多思考作者给出的反面教材，不要急于了解结果，而应掩卷遐思，分析这段代码的问题，并结合自身经验与能力给出自己的方案。然后再比较作者的方案，两相印证，辨别两个方案各自的优劣之处。最后再去仔细阅读作者的分析过程，才能更好地理解本书，提升自己的编码能力。

唯一让我觉得美中不足的是书中第 14 章。这一章主要通过一个实际案例，讲解了测试与代码可读性之间的关系。这个案例是一个测试方法，作者认为代码至少存在 8 个问题，并由此展开对这一测试方法的分析。然而，或许作者过于追求代码的优雅，使得本来较为简单的测试变得相对复杂，引入了许多不必要的代码。尤其是引入的 ScoredDocsFromString() 方法，它是为测试服务的辅助方法，如下所示：

```
vector<ScoredDocument>ScoredDocsFromString(string scores) {
    vector<ScoredDocument> docs;
    replace(scores.begin(), scores.end(), ',', ' ');
    // Populate 'docs' from a string of space-separated scores.
    istringstream stream(scores);
    double score;
    while (stream >> score) {
        AddScoredDoc(docs, score);
    }
    return docs;
}
```

对于一个测试辅助方法而言，它的逻辑显得过于复杂了。最关键的是对于这样一个相对复杂的方法，而它本身又属于测试的一部分，该谁来保证它自身的正确性呢？这样对测试代码的改写有过度设计的嫌疑，事实上加大了理解测试的复杂度。作者在书中也写到：

This may seem like a lot of code at first glance, but what it lets you do is incredibly powerful.Because you can write an entire test with just one call to CheckScoresBeforeAfter(), you' ll be inclined to add more tests.(as we' ll be doing later in the chapter).（粗略一瞥，好似代码量增加了，实则它会给你带来难以置信的威力。这是因为现在所写的整个测试仅仅调用了一次 CheckScoresBeforeAfter()方法，但在将来你会添加更多的测试。我们将在本章后面讨论）。

遗憾的是，直至本章末尾，我也未曾看到作者有足够说服力的分析，让我愿意为了这样的重用性与灵活性做出如此复杂的实现。不过，本章末尾指出测试方法存在的 8 个问题，确实精辟独到，值得我们反复阅读，细细体味个中奥秘。

全书一共 204 页，与那些瀚如烟海的高文大册相比，本书显得轻而薄，但它胜在精专。作者没有囊括所有编码技巧的野心，更没有卖弄地展现自己的设计技巧和博识广学。它的专注可能会因此失去一大部分读者群，但这样的书才是我们程序员真正需要的。遗憾的是目前国内并没有出版本书的中文版或影印版。幸运的是，在本书出版商 O' Reilly 的官方网站上，给出了本书的免费在线版本。有兴趣的读者可以通过访问 http://ofps.oreilly.com/titles/9780596802295/阅读本书。

批注：

【1】Edward Yourdon：世界知名计算机专家，面向对象设计大师，Coad/Yourdon 面向对象方法学的开发者之一。他出版了数十本著作，包括 Death March（即中文版《死亡之旅》)、The Decline and Fall of the American Programmer 与 The Rise and Resurrection of the American Programmer 等，并发表了 550 多篇技术论文和文章。Yourdon 是 ACM、IEEE、PMI 和 ITLA 的成员。

【2】Larry L.Constantine 是澳大利亚悉尼理工大学计算机科学教授，专门讲授软件工程和组织变更管理。他是最早从事结构化设计以及现代软件工程理论与实践的许多核心概念和模型研究的专家，是一位国际公认的软件人类因素方面的权威。他一共发表了 150 多篇论文和文章，出版了 10 余部专著，其中包括《Software for Use》(该书获得了 1999 年的 Jolt Product Excellent 大奖)。

原文链接：http://www.infoq.com/cn/articles/art-of-readable-code

相关内容：构建的可伸缩性和达到的性能：一个虚拟座谈会、与 Patrick Smacchia 谈.NET 的代码分析、只有好代码的项目能成功吗？、注释还是不注释，这是个问题

推荐文章 | Articles

大型网站复杂业务持续重构之道——全程领域建模实践

作者 孙兵

人物介绍

王总——“宠物商店”的总经理，从美国留学后回国创立“宠物商店”网站。一路来唾手可得的成功让他养成了固执专横的行事作风。

Spark —— Jack Chen 的大学同学，一家商业软件公司的高级咨询顾问。最近热衷于宣扬“领域驱动设计”的最佳实践。

引子

就象大家所听说过的那些神奇小子创业故事一样，几只从大西洋游回的海龟找到了一个伟大的 idea——在互联网上开办在线商店销售宠物。幸亏的是他们找到了投资者而且发展的很不错。但是随着时间的推移，当初“完美”的技术架构随着越来越多的装进篮子的需求后变得不堪重负。作为公司首席架构师的 Jack Chen 已经被这几个月“鸡毛蒜皮”的需求折磨失眠好几天啦。

Jack Chen 周一一早就被兴奋的王总给喊进了办公室，立即就被王总扔出来的 idea 吓傻了。

“我有一个很 cool 的想法，我们可以在线为宠物医院提供在线预约的服务业务。而不仅仅是卖掉它们，你知道这意味着什么吗？这是一个年产值上百亿的市场！！！”。

“可是王总，我们的系统不能支持这种非实物的服务预订销售，它可能对我们原有的网站形成巨大的冲击，我们需要三个月的时间对这个业务进行全方面的评估...”

Jack Chen 立即就被气势汹汹的王总打断了，“三个月的评估？我需要在两个月内就给我上线这个新业务。我们的投资人非常认可我的 idea，并要求我们立即把这个项目上线，它可能会帮助我们提高明年的 IPO 价格。你明白吗？ DO IT ASAP！”

评估

“好吧，也许这个该死的王胖子是对的。我们这个将技术与业务混在一起的乱摊子也是到了该整理整理的时候。” 自言自语发了半小时牢骚后的 Jack Chen 终于恢复到正常状态上来了，我想我应该看看我们现在是什么样子的，为了支持这个该死的“在线为宠物医院提供在线预约的服务”的需求我们需要做出哪些改变。 于是 Jack Chen 在白板上很快的就画出了下面的 Use Case 图来。

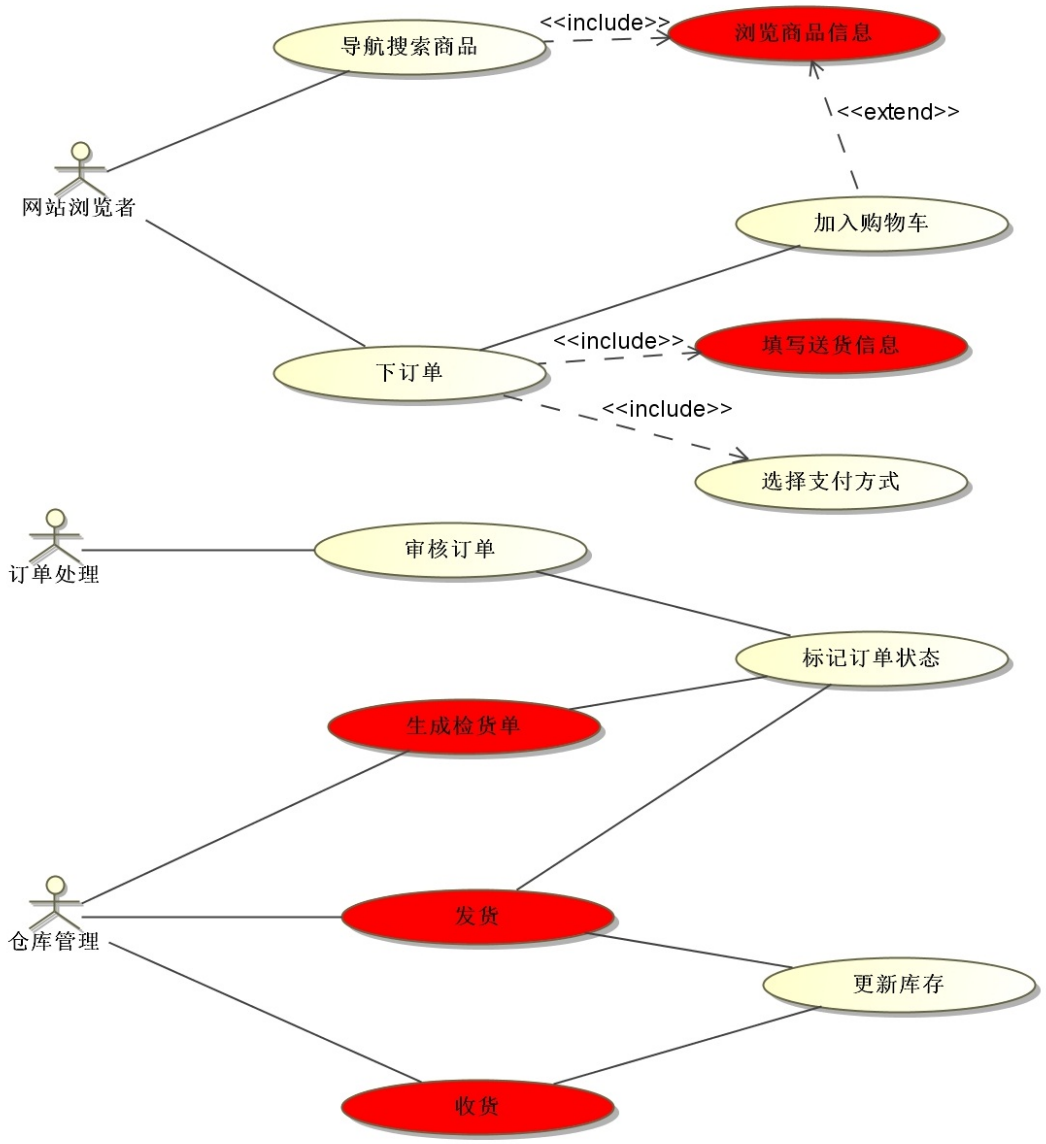


图 1 原宠物商店 UseCase 汇总图

为了支持“在线预约”这种特殊的产品，它会影响到大部分的 Use Case，具体列举如下：

1. 商品信息需要增加“预约时间”这个属性，客户在下订单时会把它作为标识一个预约的关键要素。

2. “在线预约”是个虚拟的商品，它可不需要真的需要去检货和包装发货，如果真的那么做啦，我就太傻了。

3. 每个宠物医院每天都只能接受一定数量的预约，从这个概念上来说，它与实物商品有类似的库存概念。可是我该怎么去表达它们呢？

4. 最要命的是：我真的要把这些所有受影响的 Use Case 都翻出来去让它们支持虚拟物品的业务吗？我怎么可能在 2 个月内完成这些重构？

银弹

了无生趣的 Jack Chen 在王总的办公室门口徘徊了 N 圈，还是没有勇气去迎接那一通狂风暴雨般的中英文双语版的羞辱谩骂。“也许事情是有转机的，我好象在哪里听说过有种银弹可以解决这种系统重构的问题的”。“该死，谁把 Spark 送给我的《领域驱动设计》垫在显示器下啦，他一直在向我布道这本书给他的项目带来的种种神奇改变，也许我也可以试试它的威力”。

“好吧，Spark，我承认你给推荐的书非常棒，你说的也很有道理。我读了它，明白并一些概念——例如：领域分割 、Entity、Service、Value Object…，可我对于该如何去做还是一头雾水。你能不能直接把你从重构项目中获得的最佳实践直接分享给我呢？不然的话，周一王胖子是不会放过交不出答案的我的！”。读完了这本书，Jack Chen 觉得很有收获，但又不知道怎么开始，打个电话给领域建模的先行者 Spark 也许真的是解决问题最快的方法。

“什么，这个问题说来话长？不要紧，我已经在你家门口了，你同我慢慢说”，Jack Chen 带着星巴克咖啡+肯德基全家桶+久久鸭脖+谄媚的笑容出现在 Spark 家门口。

布道

Spark 听完了 Jack Chen 对于现状及需求的描述之后，一幅气定神闲的样子讪讪地说出“这个很简单嘛，你现在需要做的只是这样一些事情：”

1. 用大比例结构对你的系统进行领域划分

2. 找出这个需求影响的领域及对外接口

3. 建立一个适合你们公司的领域驱动设计的技术框架

4. 按照需求的紧急度来重构各个领域的设计与编码

下面我们就按照这个顺序来实践一下：

一、概要领域划分

Jack Chen 立即把自己之前画的 Use Case 重画了一遍，然后用希冀的眼神看着 Spark 等待着认可。“你的错误是过于看重 Case 或者操作者身份，领域的划分不是基于功能或角色来进行的，通常来说我们是将内聚程度较高的 Use Case 归到一个上下文中。尽量使得领域自闭程度较高，并拥有相同的业务语言环境。例如基于你的 Use Case 图，我会画出以下的领域”

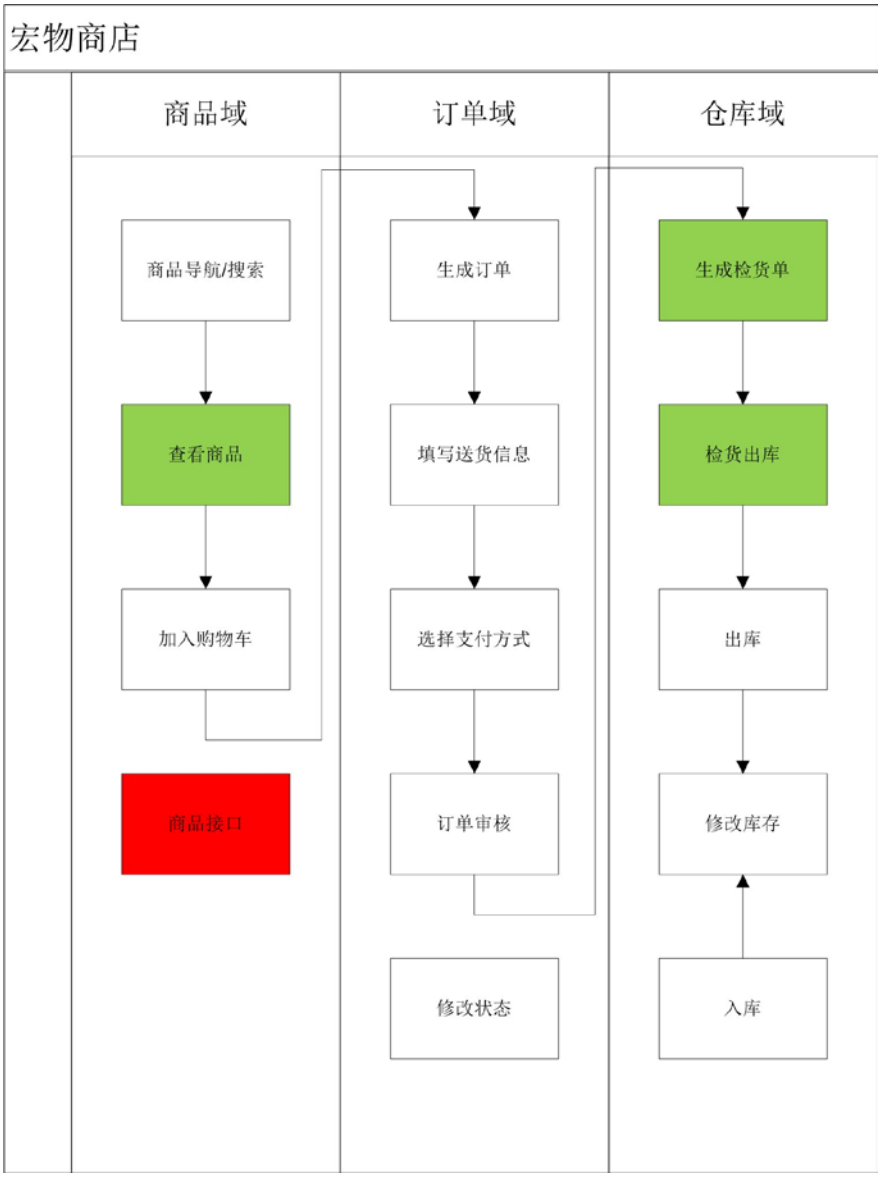


图 2 宠物商店领域通道图

通道图是一个对业务领域建模非常有帮助的工具，它可以同时表达出执行序列与分片的作用。

二、找出受影响的领域与接口

从领域的角度来看，只有商品对外暴露出来的接口是会影响到各个领域，需要优先建立商品领域（ProductDomain）及读取商品信息服务接口（GetProductService）来进行重构。

之外，在【图 2】 中用绿色标识出来的 Use Case 是由于增加支持“在线预约”这种虚拟商品所需要进行代码重构的部分。这部分工作如果工期比较紧，可以优先使用模式的方式来进行代码重构，这样也可以在之后更加容易用领域驱动设计的方法再次重构。

三、建立技术框架

这一点，是《领域驱动设计》这本书没有过多提及的内容。这个需要结合你们公司的原来技术框架用最小化改造成本最大化收益的方式来建立领域驱动的技术框架。下面是一个可以广泛使用的领域驱动的技术框架，可以在这之上增加更多的个性元素形成你公司自己的框架。

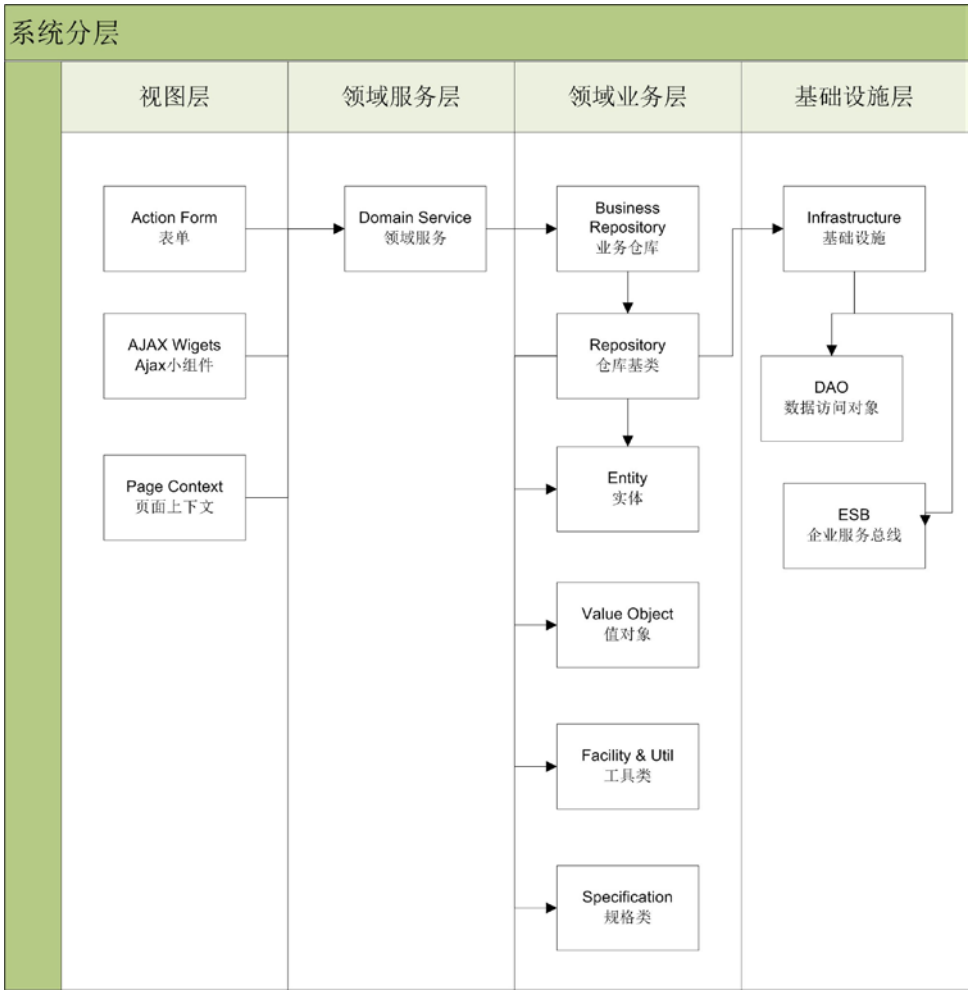


图 3 领域驱动设计参考技术框架图

这个框架的各个元素基本上在 《领域驱动设计》一书中都可以找到对应的解释，但这里需要解释一下我建立这个框架的个性理解：

1. 领域对外（页面、AJAX、ESB 调用）只暴露领域服务，其它所有领域类都是包内自闭的，对外不可见。

2. 基础仓库的引入，基础仓库是一个抽象的仓库，它封装了大量常用工具方法、业务对象生命周期维护（实体 OR 映射、DAO 调用）、外部接口调用。可以降低业务仓库不必要的重复编码与复杂性。业务仓库是继承基础仓库的子类。

3. 基础设施的引用，基础设施是用来承载引用非领域调用的桩，我们在使用领域驱动设计的时候往往是从一个旧的系统重构开始。这时我们不可能要求所有的业务子系统相互调用都通过 Domain Service 调用，这时我们可以通过 Infrastructure 优美的把调用封装在业务仓库的业务方法内。

四、重构受影响领域的设计与编码

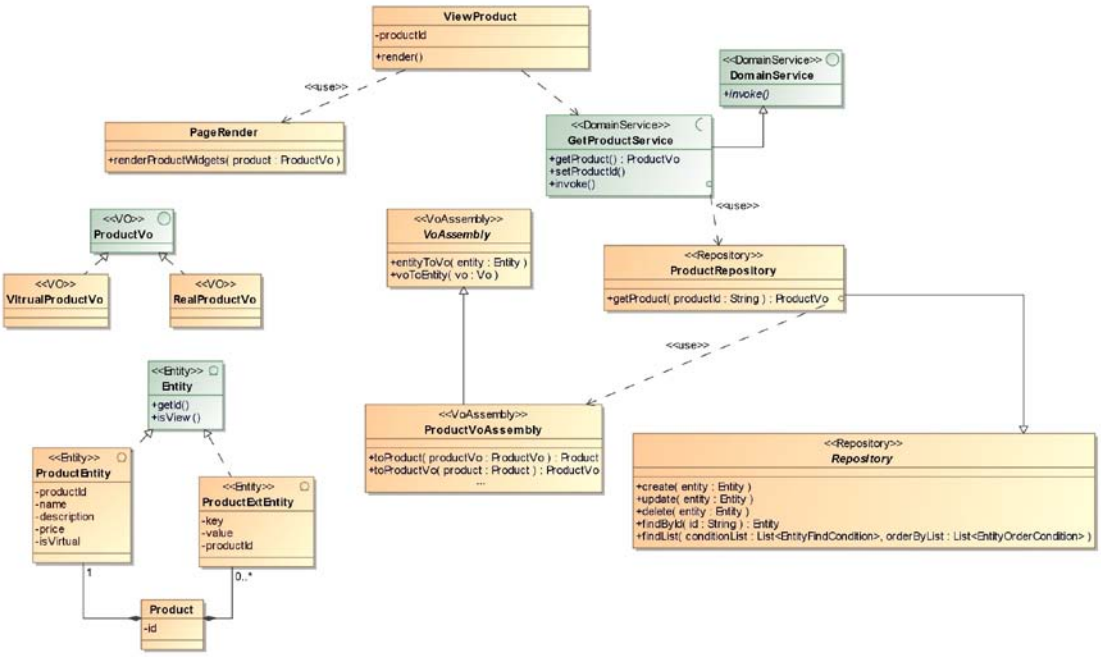


图 4 重构后的商品详情页类图

Spark 以商品详情页这个 Use Case 为例展示了以领域驱动设计的重构类图：

1. 增加行为表 ProductExt 用于存储商品的扩展信息，如预约时间段、预约医院。并为表建立一一对应的实体 Entity。

2. 基础仓库 Repository 通过 Infrastructure 中的 DAO 封装了对实体的操作 如 create()、

update()、delete()、findById()、findList()

3. 商品业务仓库 ProductRepository 扩展了基础仓库，客户程序可以用 productId 为参数，通过 ProductVo.getProduct()方法获得商品详细信息的业务实现，由于业务仓库的的公开方法对外返回的都是 Value Object，因此不会直接暴露 Entity 类型给客户程序。

4. GetProductService 服务类通过 invoke()服务方法 对外（商品详情页面）提供服务，它通调用业务仓库中的业务方法，并将接口规格化。

5. 事务配置在 DomainService 的 invoke()方法上，即事务控制以 Use Case 为粒度进行控制。

尾声

在 Spark 的帮助下，Jack Chen 成功的脱离了困境。现在他正在公司里积极推行自己的领域驱动设计框架，他们公司的网站正在以每三周一次的重构速度快速迭代演进。他象 Spark 一样，成为了一个领域驱动的布道者。

原文链接：http://www.infoq.com/cn/articles/sb-complex-business-continuity-refact

相关内容：

- 2012.3.30 微博热报：大方法、基础 vs 库
- 用 Mikado 方法重构遗留软件
- 2012.3.6 微博热报：测试的价值&重构之惑
- JetBrains 开发者工具
- 测试驱动开发（TDD）实战

推荐文章 | Articles

Siri 技术解析

作者 张俊林

Siri 的学术化全称为“个人自动智能助理”（PersonalizedIntelligentAssistant），并非苹果公司自主研发，最初是苹果 AppStore 的一款应用，苹果公司发现后很快于 2010 年 4 月份从斯坦福研究机构国际智能中心（SRI）处收购而来，传言收购价格在 2 亿美金左右，之后很快 Siri 从 AppStore 下架并集成进入 IPhone 4S。

作为“个人智能助理”，Siri 能为你提供什么服务？可以将 Siri 理解为一位低智商的生活秘书（和人类的智能比），它采用了 Nuance 公司的语音识别技术，具有一定程度的语义理解和用户意图识别能力，可以聊天解闷，更主要的功能是帮助办理一定的事务，比如预订餐厅、预订出租车、搜索精确答案、查询天气等。

我们首先看一些例子来看 Siri 能做什么：

对话一：

User:“我想掩藏一具尸体。” Siri ：你想藏在哪里呢？（给出 5 个选项）

- 水库
- 工厂
- 矿山
- 垃圾场
- 湿地

对话二：

Siri：有什么我能帮忙的？

User:“我想我刚刚杀了人。”

Siri：是这样么？

User:“是的。”

Siri：“你说要掩藏尸体我就想到了”

User:“我该怎么做？”

Siri：“好吧，这是你接下来的日程表。”

对话三：

User:“去哪儿能弄些毒品？”

Siri：“这附近有 2 家戒毒所。”

对话四、对话五、对话六：

由以上六个用户和 Siri 的对话记录片段可以看出 Siri 具备一定的语义智能，如果了解其技术原理，可以知晓解闷陪聊功能并非其最初出发点和功能重点，在多个垂直领域帮用户完成一定的任务才是其核心功能。

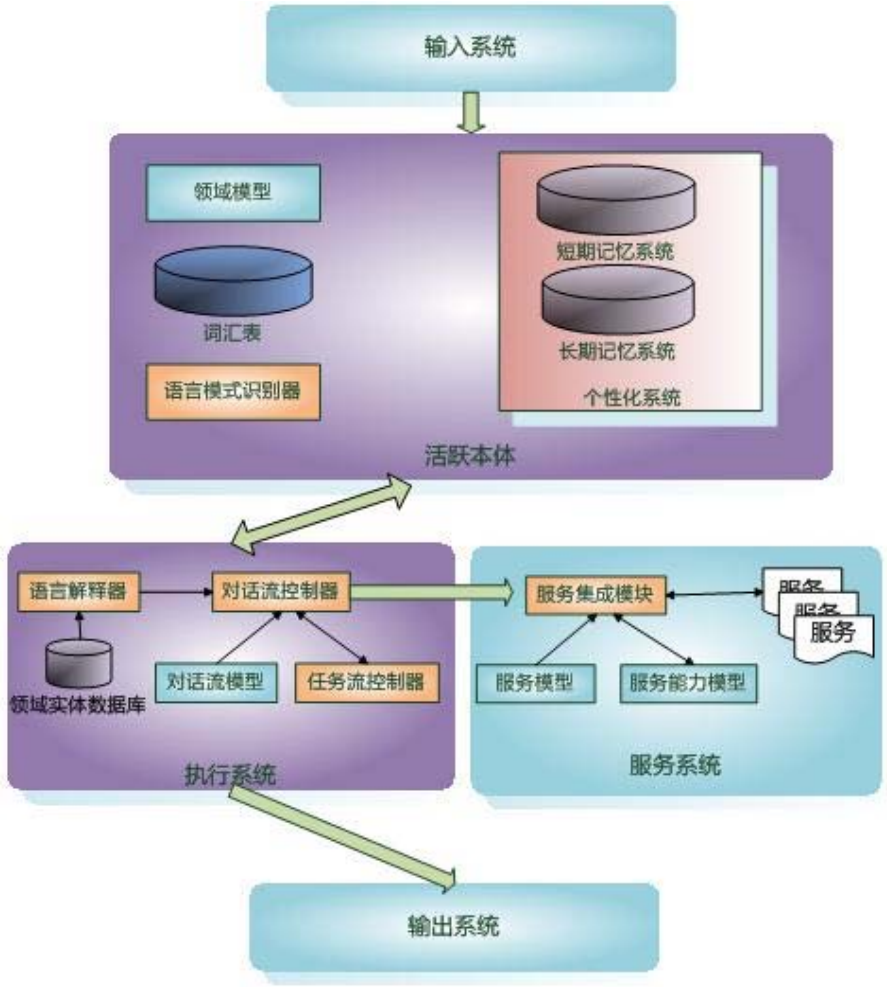


图 1 . Siri 整体架构

Siri 是一个功能繁复的综合 AI 框架，图 1 展示了其包含的数据、模型以及计算模块，为了在整体上更易于理解整个框架，可以将 Siri 里包含的众多数据、模型和计算模块划分为输入系统、活跃本体、执行系统、服务系统和输出系统五个子系统。其在解析用户输入时候遵循一定的执行顺序，以此来理解用户的真正意图并提供有用服务。

Siri 的资源主要分为资源类和计算类两大类，其中属于资源类的包括;

- 领域模型；
- 词汇表数据库；
- 短期记忆系统；
- 长期记忆系统；
- 领域本体数据库；

- 对话流模型；
- 服务模型；
- 服务能力模型；
- 外部服务；

属于计算资源的包括：

- 语音识别系统；
- 语言模式识别器；
- 语言解释器；
- 对话流控制器；
- 任务控制器；
- 服务集成模块；
- 语音生成系统；

Siri 的输入系统支持多模态输入，即不仅仅支持众所周知的语音识别，也允许用户进行文本输入、GUI 界面操作以及事件触发等。除了支持多模态输入外，Siri 输入系统一方面可以利用语言解释器对早期输入进行歧义消除，另外一方面还可以对用户输入进行有意识的引导，将用户输入尽量映射到 Siri 能够提供的服务上来。这样对于用户和 Siri 来说才可相得益彰，Siri 可体现其价值，用户可获得帮助。

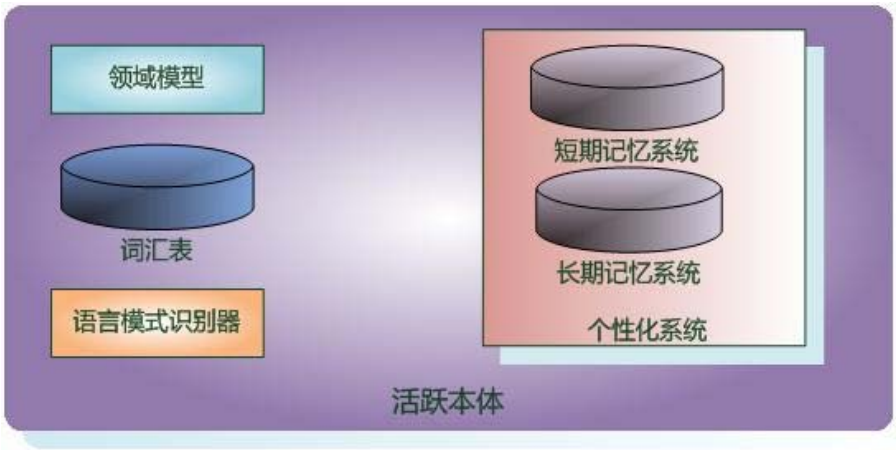


图 2. 活跃本体

“活跃本体”是 Siri 中相当重要的一个概念，“活跃本体”可以被理解为 Siri 整个系统执行

的一个具体执行环境和场所，执行系统调用所有系统数据、词典、模型和程序，在“活动本体”内对用户输入进行解析，并将文本信息在这里解析为用户真正的意图，然后根据意图来调用外部的服务。

在程序执行时，“活跃本体”内放入的数据和模型包括：领域模型，用户个性化信息，语言模式、词汇表和领域实体数据库等。

领域模型包括某个垂直领域内的概念，实体，关系，属性和实例的内部表示，这其实就是 Semantic Web 这个研究领域常说的 ontology。Siri 包含很多垂直领域的领域模型。“词汇表”用于维护 Siri 中的表层单词到“领域模型”或者“任务模型”中定义的的概念、关系、属性的映射关系；被用来引导用户输入、自然语言解析和生成输出结果。

Siri 在个性化方面做得也非常出色。在和用户沟通过程中，如果一台机器能够叫出你的名字，并且知晓你的个人爱好，用户体验无疑是非常优异的。从具体技术手段上，Siri 是通过在内部保持两个记忆系统：长期记忆系统和短期记忆系统来实现能够个性化的和用户交流的。长期记忆系统存储了用户的名称、居住地址以及历史偏好信息，短期记忆系统则将最近一段时期内 Siri 和用户的对话记录及 GUI 点选记录等登记下来。利用这两个记忆系统，Siri 可以在理解用户需求的时候帮助澄清用户的真正意图是什么。

语言模式识别系统是对用户输入的表层，语法层，习惯用语和成语等进行模式匹配的模块。匹配模式的代码在 Siri 内部采用正则表达式或者状态机等方式实现；在 Siri 识别出指定的语言模式后，可以帮助判断用户输入所述的任务类型。

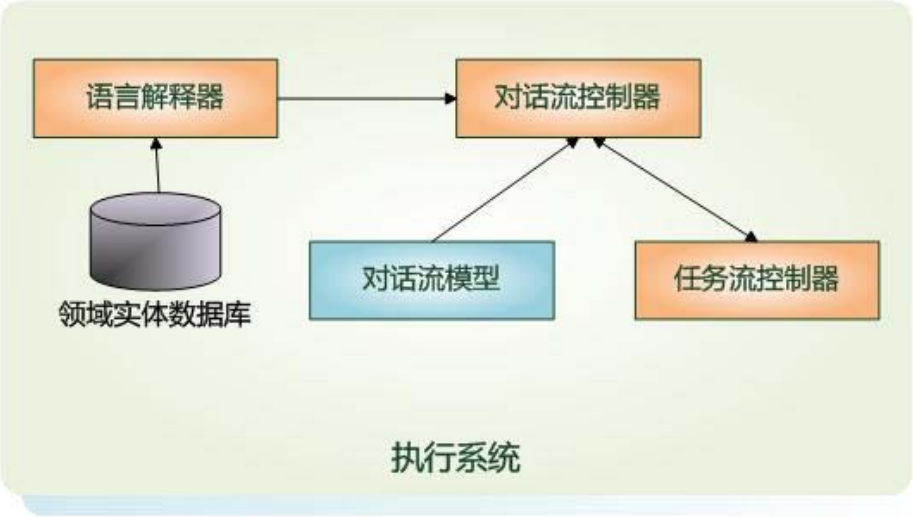


图 3 执行系统

执行系统是 Siri 系统最有技术含量的部分，前文有述：“活动本体”是对根据用户的输入信息，将各种词典资源，模型资源实例化进行具体加工的场所，而真正的加工过程是由执行系

统进行的。执行系统不仅将用户原始的文本输入解析为内部的语义表示，而且要在用户和 Siri 交互过程中（多轮会话）决定下一句 Siri 应该说什么内容，可见其重要性。

执行系统具体又可以细分为三个主要部件：语言解释器、会话流控制器和任务控制器。它们之间分工有异同时又密切合作，一起发挥作用。语言解释器将用户输入字符串流解析为语义表示作为输出，而这个语义表示又会作为会话流控制器的输入，会话流控制器根据当前语句所表达的含义，协同任务控制器一起决定 Siri 下一步应该做什么或者说什么。

语言解释器是 Siri 中最重要的自然语言处理工具，主要用来对文本形式的用户输入进行解析，将其映射为概念本体层级的信息表示，即理解语言真正的含义，除此外，语言解释器也被用在输入系统中对用户输入提示或者输入补全进行分析，而且对语音识别结果后处理也有很大帮助。

对话流控制系统是在将用户的文本表示解析为内部用户意图之后发挥作用；即语言解释器将解析结果传递给对话流控制器，是语言解释器的后续处理步骤；而“任务控制器”则被“对话流控制器”调用，共同确定 Siri 下一步应该做什么或者说什么。

“任务流控制器”的主要功能是界定完成一件任务或者解决某个问题由那些步骤构成,这些步骤之间是何种关系。“任务流控制器”和“对话流控制器”很容易混淆，不容易区分其功能差异。一般来说，“对话流控制器”主要用来决定 Siri 接下来要说的内容或者要做的事件，主要是根据领域判断诱导用户提供所需的参数；而“任务流控制器”更侧重于事务本身的定义，比如一个任务可以切分成若干子任务，是否有时序依赖关系。

任务流控制在 Siri 中也起到举足轻重的地位，Siri 的任务模型是由一些领域无关的通用任务模型和若干领域相关任务构成。通用任务是完成一件任务的抽象表述，与具体领域无关，因为其通用性，也可以应用在各个具体应用领域。

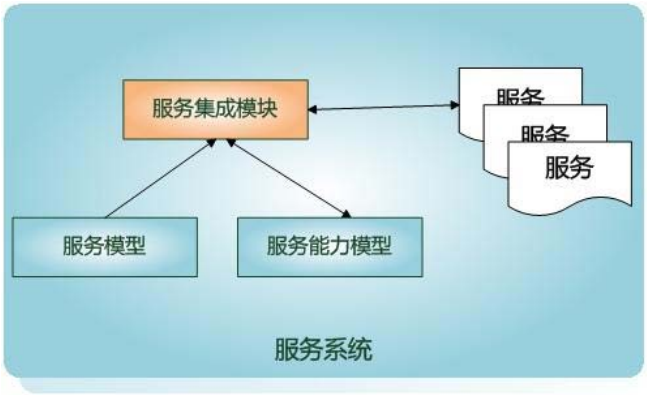


图 4 服务系统

Siri 本质上是服务导向的用户意图识别系统，无论是对话流控制也好，任务流控制也好，其根本目的还是为了能够将用户引导到 Siri 能够提供的某项具体服务，以此达到帮助用户完成某些任务或者解决一些问题的目的。目前 Siri 可以提供多种领域的服务，这里面涉及到服务管理的问题，即如何进行管理才能使得系统可用性高，可维护性强等。具体而言，Siri 中有三个子部分涉及到服务功能：服务模块，服务能力模型和多服务集成模块。其中，服务模块记录了可供 Siri 使用的各种服务的详细信息，服务能力模块则存储了哪些服务可以提供什么类型的服务等映射关系，服务系统中最重要的是服务集成模块，调用另外两个服务模块提供给用户最终服务内容。因为往往完成用户某项需求要调用分布在各处的多项服务，每项服务能够提供部分信息，而且服务之间有些顺序需要遵守，所以如何调用所需的多种功能，调用顺序如何确定以及如何根据部分信息拼合成最终用户所需服务是其核心内容。

Siri 的输出系统会将最终提供的服务结果或者在会话过程的中间内容展示给用户。其不仅支持语音、电邮、文本等多模态输出，还支持界面订制等个性化功能。

从上述技术描述看，Siri 是苹果公司新推出的一种新型人工智能框架，不仅在商业宣传上令人耳目一新，在其技术架构和具体实现上也颇具新意。尽管 Siri 最初是依附在 iPhone 平台，但是很显然，这种依附性并不强，可以预见，这套系统会不断扩展到更多种硬件类型的智能控制，比如车载控制系统，智能电视控制系统等等

关于作者

张俊林，《这就是搜索引擎：核心技术详解》作者，新浪微博研发人员，主要研究方向：自然语言处理、搜索技术、推荐系统及机器学习

原文链接：http://www.infoq.com/cn/articles/zjl-siri

推荐文章 | Articles

集成 Lucene 和 HBase

作者 Boris Lublinsky, Mike Segel 译者 侯伯薇

在所有先进的应用程序中，不管是购物站点还是社交网络乃至风景名胜站点，搜索都扮演着关键的角色。Lucene 搜索程序库事实上已经成为实现搜索引擎的标准。苹果、IBM、Attlassian（Jira）、Wolfram 以及很多大家喜欢的公司【1】都使用了这种技术。因此，大家对任何能够提升 Lucene 的可伸缩性和性能的实现都很感兴趣。

Lucene 简介

Lucene 中可搜索的实体都表现为文档（document），它由字段（field）和值（value）组成。每个字段值都由一个或多个可搜索的元素——即词汇（term）——组成。Lucene 搜索基于反向索引，其中包含了关于可搜索文档的信息。在使用正常索引时，你可以搜索文档，以了解它包含哪些字段，但使用反向索引与之不同，你会搜索字段的词汇，以了解所有包括该词汇的文档。

图 1 显示的是高层次的 Lucene 架构【2】。它的主要组件包括 IndexSearcher、IndexReader、IndexWriter 和 Directory。IndexSearcher 实现了搜索逻辑。IndexWriter 为每个插入的文档写入反向索引。IndexReader 会在 IndexSearcher 的支持下读取索引的内容。IndexReader 和 IndexWriter 都依赖于 Directory，它会提供操作索引数据集的 API，而该 API 会直接模拟文件系统 API。

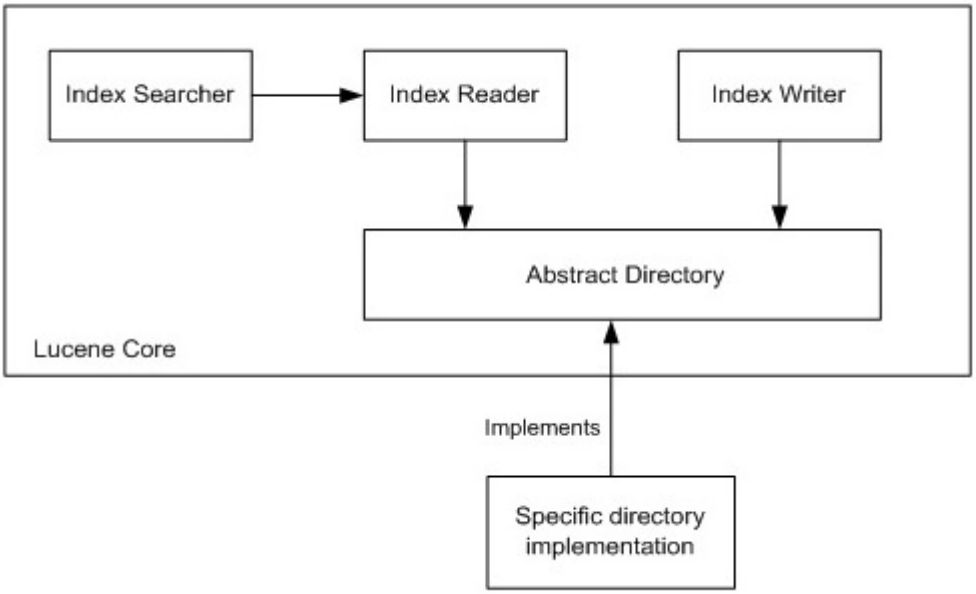


图 1： 高层次的 Lucene 架构

标准的 Lucene 分发包中有多个目录实现，包括基于文件系统和基于内存的[1]。

标准基于文件系统的后端的缺点在于，随着索引增加性能会下降。人们使用了各种不同的技术来解决这个问题，包括负载均衡和索引分片（index sharding）——在多个 Lucene 实例之间切分索引。尽管分片功能很强大，但它让总体的实现架构变得更复杂，并且需要大量对期望文档的预测知识，才能对 Lucene 索引进行合适地分片。

另一种不同的方法是，让索引后端自身对数据进行正确地分片，并基于这样的后端构建出实现。这种后端可以是 NoSQL 数据库。在本文中我们会描述基于 HBase 的实现【4】。

实现方法

正如在【3】中所说明的，在高层次上，Lucene 会操作两个单独的数据集：

索引数据集中保存了所有字段/词汇对（还有其他信息，像术语频率、位置等），以及在恰当的字段包含这些词汇的文档。

文档数据集中存储所有文档，包括存储的字段等。

正如我们已经在上面提到的，想要把 Lucene 移植到新的后端中，直接实现 directory 接口并不总会是最简单（最方便）的方法。所以，很多对 Lucene 的移植，包括从 Lucene 的 contrib.module 支持的优先内存索引、Lucandra【5】和 HBasene【6】分别采用了不同的方法【2】，不仅重写了 directory，还重写了高级的 Lucene 类——IndexReader 和 IndexWriter，从而绕开了 Directory 的 API（如图 2）。

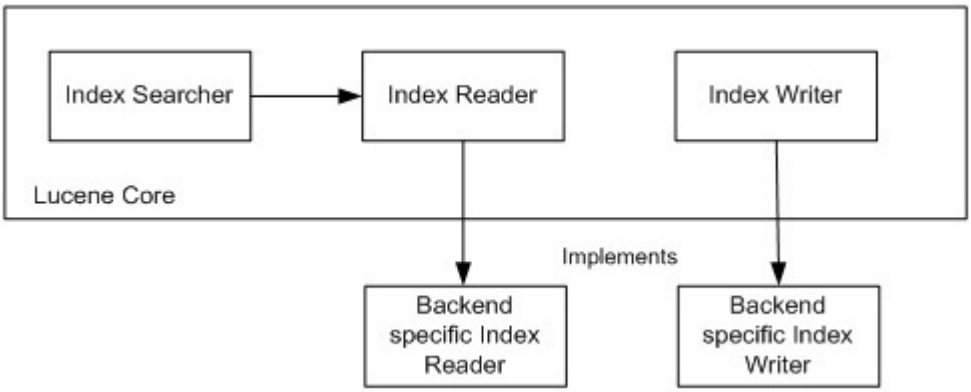


图 2： 将 Lucene 和没有文件系统的后端整合

尽管这种方法通常需要更多工作【2】，但是它能够带来更强大的实现，让我们可以完全利用后端的本地功能。

文中所展现的实现[2]也遵循了这种方法。

总体架构

总体上的实现（如图 3）是在基于内存的后端之上构建的，并将其用作内存缓存、同步缓存和 HBase 后端的机制。

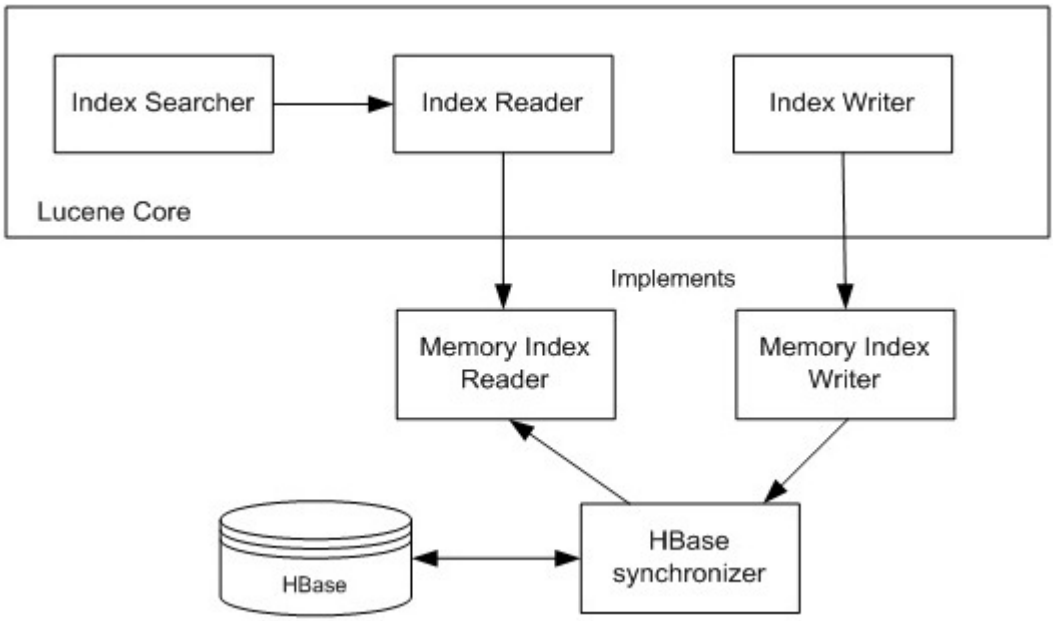


图 3： 基于 HBase 的 Lucene 实现的总体架构

该实现试图平衡两种相互冲突的需求，性能： 在内存中，缓存能够最小化 HBase 用于搜索和文件返回的数据读取量，从而极大提升性能；可伸缩性： 按照需要运行为多个 Lucene 示例以支持日益增长的搜索客户端的能力。后者需要最小化缓存的生命周期，从而和 HBase

实例（上面提到实例的副本）中的内容同步。通过为活动参数实现可配置的缓存时间，限制每个 Lucene 实例中展现的缓存，我们可以达成一种折中方案。

内存缓存中的底层数据模型

正如之前所提到的，内部 Lucene 数据模型基于两种主要的数据集——索引和文档，它们会被实现为两种模型——IndexMemoryModel 和 DocumentMemoryModel。在我们的实现中，读写操作（IndexReader/IndexWriter）都是通过内存缓存完成的，但是它们的实现有很大区别。对于读取操作，缓存首先会检查所需要的数据是否在内存中，并且没有过期[3]，如果那样的话就会直接使用。否则缓存就会从 HBase 读取或者刷新数据，然后把它返回给 IndexReader。相反，对于写操作，数据会直接写入到 HBase，而不会在内存中存储。尽管这在实际的数据可用性方面会有延迟，但是它会让实现过程非常简单——我们不需要考虑把新建或者更新的数据发送给哪个缓存。这里的延迟可以通过设置合适的缓存过期时间来控制，从而符合业务需求。

IndexMemoryModel

索引内存模型的类图如图 4 所示。

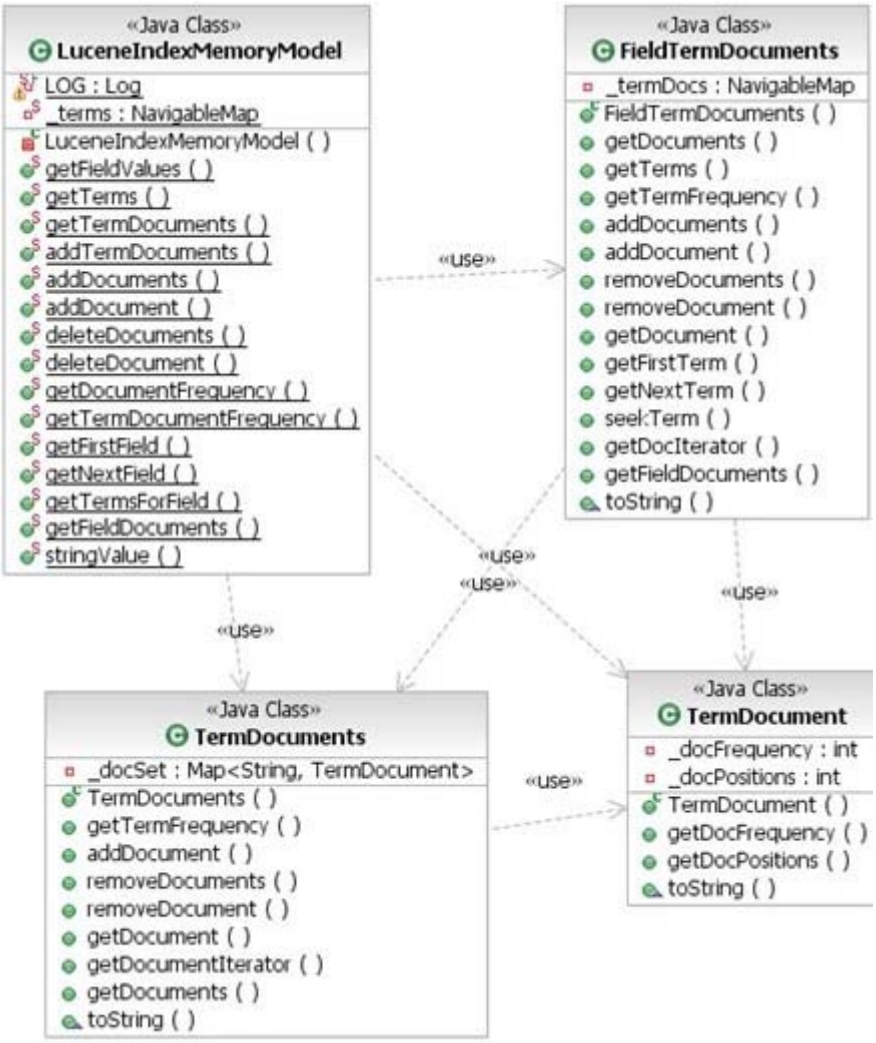


图 4： IndexMemoryModel 类图

在这个实现中：

- LuceneIndexMemoryModel 类包含了当前存在于内存中所有字段的 FieldTermDocuments 类。它还提供了所有对于实现 IndexReader 和 IndexWriter 必要的内部 API。
- FieldTermDocuments 类会为每个字段值管理 TermDocuments。通常，对于可扫描的数据库，字段的列表和字段值的列表可以组合在可导航的字段/词汇值列表中。对于基于内存的缓存实现，我们已经把它们切分为两个独立的部分，从而让搜索的时间更可预测。
- TermDocuments 类为每个文档 ID 包含了一系列 TermDocument 类。
- TermDocument 类包含了针对给定文档在索引中存储的信息——文档使用频度和位置

的数组。

DocumentMemoryModel

文档内存模型的类图如图 5 所示。

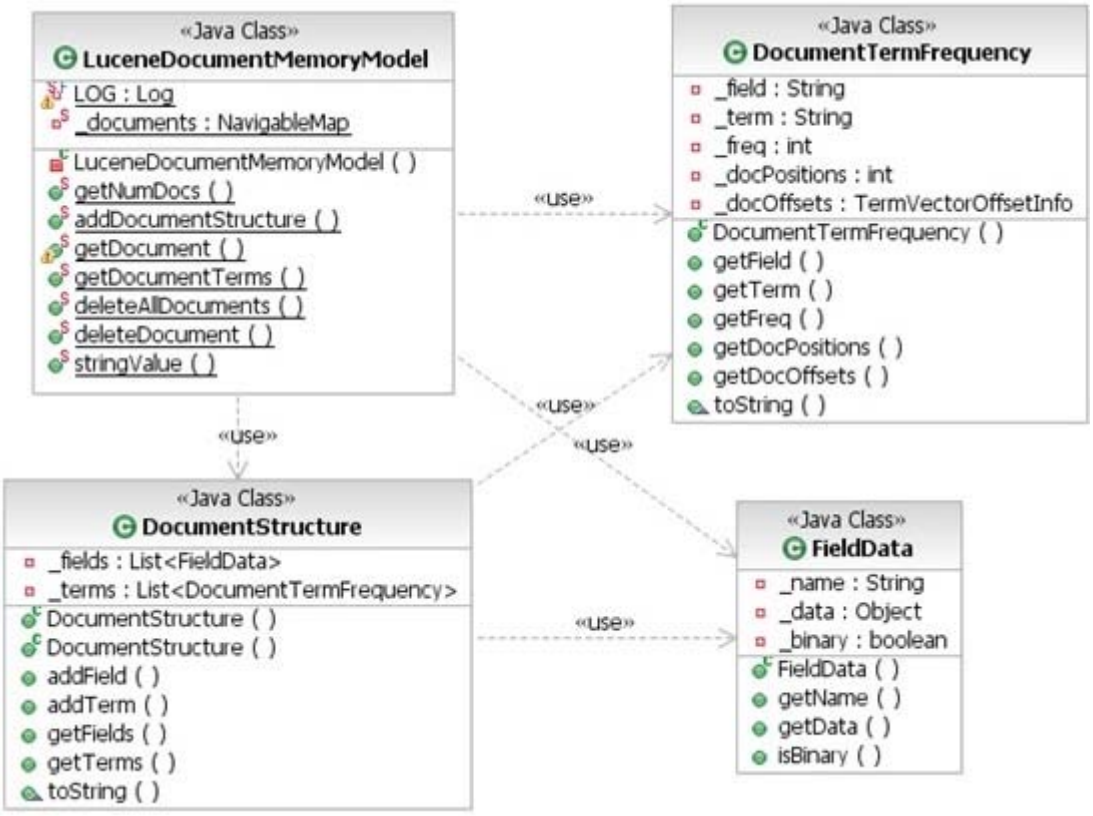


图 5： DocumentMemoryModel 类图

在这个实现中：

- LuceneDocumentMemoryModel 类包含 DocumentStructure 类与每个被索引文档的映射关系。
- DocumentStructure 类中包含了单个文档的信息。针对每个文档，它都包含了为每个索引后字段保存的字段和信息。
- FieldData 类包含为存储字段（stored field）所保存的信息，包括字段名称、值和二进制或者字符串型的标识。
- DocumentTermFrequency 类包含了关于每个被索引字段的信息，包括对相应索引结构（索引、词汇）的向后引用、文档中词汇使用频率、词汇在文档中的位置以及从文档开始的偏移量。

LuceneDocumentNormMemoryModel

正如在【9】中说明的，规范（norm）是用于表现文档或字段的加权因子，从而提供更好的搜索结果排序，这需要耗费大量内存。类的实现基于对映射的映射（map of maps），内部映射会存储规范和文档的映射关系，而外部映射会存储规范和字段的映射关系。

尽管规范信息的键值是字段名称，从而可以添加到 LuceneIndexMemoryModel 类中，但是我们还是决定把对规范的管理实现为单独的类——LuceneDocumentNormMemoryModel。这么做的原因在于，在 Lucene 使用规范是可选的操作。

IndexWriter

有了之前所描述的底层内存模型，实现索引写入程序就很简单了。因为 Lucene 不会定义 IndexWriter 接口，所以想要实现 IndexWriter，我们需要实现所有标准 Lucene 实现中的方法。这个类的主要内容在于 addDocument 方法。这个方法会遍历所有文档的字段。对于每个字段，方法都会检查它是否可以令牌化（tokenized），并使用特定的分析器来做到这一点。这个方法还会更新所有三种内存结构——索引、文档和（可选的）规范，它们会为新增的文档存储信息。

IndexReader

IndexReader 会实现 Lucene 核心所提供的 IndexReader 接口。因为 Hbase 中所获得的列表和单独的读操作相比要快很多，所以我们使用一些方法来扩展这个类，从而可以读取多个文档。类本身没有把更多的处理转交给几个类，它会对其进行管理：

- 尽管文档 ID 通常是字符串，但 Lucene 内部还是对整型数操作。DocIDManager 这个类会负责管理从字符串到数字的转换。IndexReader 会以 ThreadLocalStorage 的形式使用这个类，从而可以在线程结束之后自动清理。
- MemoryTermEnum 类扩展了 Lucene 提供的 TermEnum 类，负责扫描字段/词汇的值。
- MemoryTermFrequencyVector 类会实现 Lucene 提供的 TermDocs 和 TermPositions 接口，负责为给定的字段/词汇对（field/tem pair）处理与文档相关的信息。
- MemoryTermFrequencyVector 类实现了 Lucene 提供的 TermFreqVector 和 TermPositionVector 接口，负责针对给定的文档 ID 返回文档字段频率和位置信息。

HBase 表

以上提出的解决方案基于两个主要的 HBase 表——Index 表（图 6）和 document 表（图 7）。

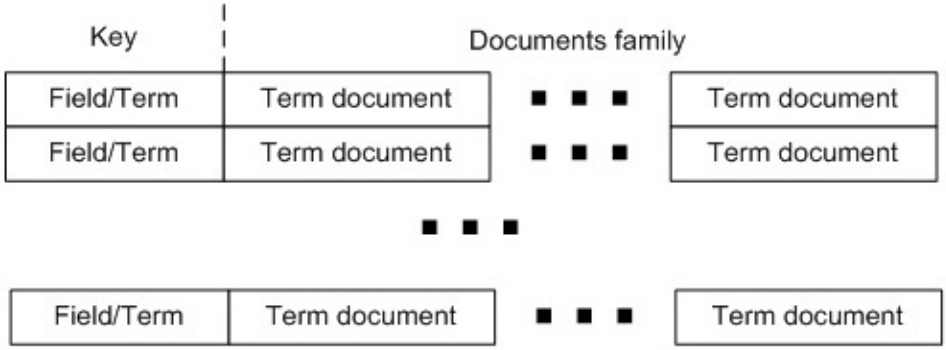


图 6： HBase 的 Index 表

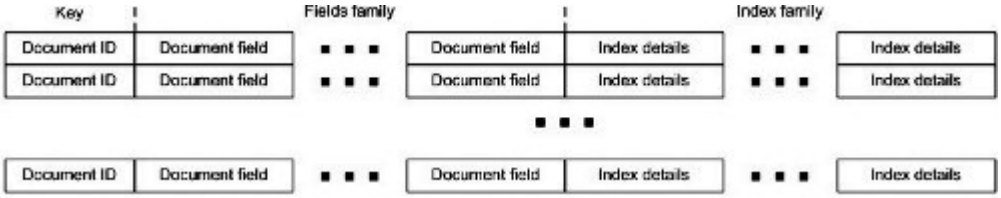


图 7： HBase 的 document 表

如果需要支持规范的话，可选择实现第三个表（图 8）。

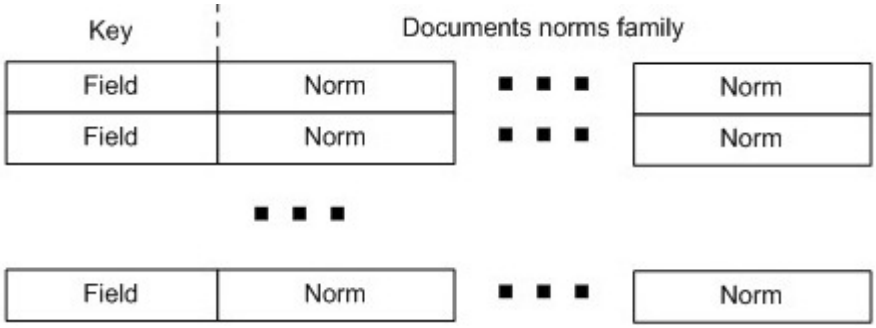


图 8： HBase 的 norm 表

HBase 的 Index 表（图 6）会完成实现的主要工作。这个表对每个 Lucene 实例所知道的字段/词汇组合都设置了入口，其中包含一个列族（column family）——documents 族。这个列族为包含这个字段或词汇的所有文档都包含了一列（名称是文档的 ID）。每个列的内容都是 TermDocument 类的值。

HBase 的 document 表（图 7）存储了文档本身、对索引或规范的向后引用，它会为文档处理引用这些文档以及一些 Lucene 使用的附加信息。它对所有 Lucene 实例知道的文档都

设置了入口（row）。每个文档都通过文档 ID（键值）唯一标识，并包含两个列族——字段族和索引族。字段列族针对所有存储在 Lucene 中的文档字段包含一列(名称为字段的名称)。列的值是值的类型（字符串或者字符数组）和值本身的组合。索引列族为每个引用这个文档的索引包含了一列（名称是字段或者术语）。列的值包括给定字段/词汇在文档的使用频率、位置和偏移量。

HBase 的 norm 表（图 8）为每个字段存储了文档的规范。它对所有 Lucene 实例知道的每个字段（键值）都设置了入口（行）。每行都只包含一个列族——规范族。这个族对每个需要存储给定字段规范的文档都有一列（名称是文档 ID）。

数据格式

最终的设计方案确定了在 HBase 中存储数据的数据格式。对于这个实现，我们基于性能、结果数据的最小规模以及和 Hadoop 的紧密整合程度选择了 Avro【10】。

实现主要使用的数据结构是 TermDocument（代码 1）、文档的 FieldData（代码 2）和 DocumentTermFrequency（代码 3）。

```
{
  "type" : "record",
  "name" : "TermDocument",
  "namespace" : "com.navteq.lucene.hbase.document",
  "fields" : [ {
    "name" : "docFrequency",
    "type" : "int"
  }, {
    "name" : "docPositions",
    "type" : ["null", {
      "type" : "array",
      "items" : "int"
   }]
  } ]
}
```

代码 1 词汇文档 AVRO 定义

```
{
  "type" : "record",
  "name" : "FieldsData",
  "namespace" : "com.navteq.lucene.hbase.document",
  "fields" : [ {
    "name" : "fieldsArray",
    "type" : {
      "type" : "array",
      "items" : {
        "type" : "record",
        "name" : "singleField",
        "fields" : [ {
```

```
            "name" : "binary",
            "type" : "boolean"
          }, {
            "name" : "data",
            "type" : [ "string", "bytes" ]
          } ]
        }
      }
    } ]
  }
```

代码 2 字段数据 AVRO 定义

```
  {
    "type" : "record",
    "name" : "TermDocumentFrequency",
    "namespace" : "com.navteq.lucene.hbase.document",
    "fields" : [ {
      "name" : "docFrequency",
      "type" : "int"
    }, {
      "name" : "docPositions",
      "type" : ["null",{
        "type" : "array",
        "items" : "int"
      }]
    }, {
      "name" : "docOffsets",
      "type" : ["null",{
        "type" : "array",
        "items" : {
          "type" : "record",
          "name" : "TermsOffset",
          "fields" : [ {
            "name" : "startOffset",
            "type" : "int"
          }, {
            "name" : "endOffset",
            "type" : "int"
          } ]
        }
      }]
    } ]
  }
```

代码 3 TermDocumentFrequency 的 AVRO 定义

结论

本文中描述的简单实现完全支持所有 Lucene 功能，针对 Lucene 核心和普通模块的单元测试都验证了这一点。我们可以将它作为基础，构建可扩展性很强的搜索实现，支持 HBase 固有的可扩展性以及完全对称的设计，让我们可以添加任意数量服务于 HBase 数据的进程。它还可以避免需要关闭打开状态的 Lucene 索引读取程序，就可以包含新的索引数据，那会

经过一定延迟之后为用户所用，而延迟是通过活动参数的缓存时间所控制的。在下一篇文章中我们会展示如何扩展这个实现，以包含地理搜索支持。

关于作者

Boris Lublinsky 是 NAVTEQ 的首席架构师，他的工作是为大型数据管理和处理、SOA 以及实现各种 NAVTEQ 项目定义架构的愿景。他还是 InfoQ 的 SOA 编辑，并且参与了 OASIS 的 SOA RA 工作组。Boris 是一位作者，并经常发表演讲，他最新的著作是《Applied SOA》。

Michael Segel 拥有二十多年和客户协作的经验，和他们一起确定并解决业务问题。Michael 曾经在多个领域以多种角色工作过。他是一位独立顾问，期望解决任何有挑战性的问题。Michael 拥有俄亥俄州立大学的软件工程学位。

参考文献

1. Lucene-java Wiki
2. Animesh Kumar. Apache Lucene and Cassandra
3. Animesh Kumar. Lucandra - an inside story!
4. HBase
5. Lucandra
6. HBasene
7. Bigtable
8. Cassandra
9. Michael McCandless, Erik Hatcher, Otis Gospodnetic. Lucene in Action, Second Edition.
10. Boris Lublinsky. Using Apache Avro.

批注：

---

[1]附加的 Lucene 程序包含了为 Berkley DB 构建的 DB 目录。

[2]这个实现受到了 Lusandra 源代码【3】的启发。

[3]没有在内存中存在太长时间。

原文链接：http://www.infoq.com/cn/articles/LuceneHbase

时刻关注软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ
Enterprise Software Development Community

商务合作：sales@cn.infoq.com
读者反馈/投稿：editors@cn.infoq.com

新品推荐 | New Products

淘宝开源分布式消息中间件 Metamorphosis

作者 崔康

最近，淘宝开源了分布式消息中间件 Memorphosis 项目，它是 Linkedin 开源 MQ——Kafka 的 Java 版本，针对淘宝内部应用做了定制和优化。

原文链接：http://infoq.com/cn/news/2012/03/metamorphosis

Symbian 为何在竞争中落后于 iOS 和 Android？

作者 崔康

最近，在知乎网站上，有不少开发人员针对 Symbian 为何在竞争中落后于 iOS 和 android 展开了热烈讨论，从中我们可以借鉴移动开放平台的发展经验和教训。

原文链接：http://infoq.com/cn/news/2012/03/symbian-behind

一款着眼性能的轻量级 Data Grid——WPF Table View

作者 Jonathan Allen 译者 曹如进

WPF Table View 是一款轻量级数据网格，它被设计用于处理超过 10 万行的超大数据集。我们对他的创建者 Graham Knigh 进行了一段采访。

原文链接：http://www.infoq.com//cn/news/2012/03/WPF-Table-View

专访 HTML5 版 Clear 应用作者尤雨溪

作者 崔康

Clear 是一款非常受欢迎的移动应用，尤雨溪在两天时间内打造了一个 HTML5 版的 Clear 应用，并开放了源代码，InfoQ 中文站对其进行了专访。

原文链接：http://infoq.com/cn/news/2012/03/evan-html5-clear

ZK Web 框架 6.0 发布：新型数据绑定系统

作者 KostisKapelonis **译者** 吴宇

ZK Web 框架 6.0 的发布提供了一个全新的数据绑定系统、运行在服务器端的 jQuery 风格选择器、Servlet 3.0 异步支持、新组件以及其他一些增强。

原文链接：http://www.infoq.com/cn/news/2012/03/zk-6-released

Mono for Android UI 设计器介绍

作者 Jonathan Allen **译者** 曹如进

Mono for Android 很快将拥有自己的可视化用户界面设计器。目前它的可用版本为 beta 测试版（private beta），感兴趣的开发人员可以报名参与 Mono for Android 设计器测试版计划。下面我们将对话用户界面设计器架构师 Lluis Sanchez 以了解更多信息。

原文链接：http://www.infoq.com//cn/news/2012/03/Mono-Designer

使用 MonoTouch.SQLite 简化用户界面开发

作者 Jonathan Allen **译者** 曹如进

当你使用 iOS 和 MonoTouch 时，可能需要显示一个 SQLite 表中的数据。虽然你可以通过构建 UITableViewController 来完成数据显示，但是整个过程相当耗时，并且需要相当多的样板代码。Jeffrey Stedfast 找到了一种方法，通过借助 MonoTouch.SQLite 可以将简单的表格绑定减少为一个单个方法。

原文链接：http://www.infoq.com/cn/news/2012/03/MonoTouch-SQLite

Adobe 将在 PhoneGap 品牌下发布 Cordova

作者 Abel Avram **译者** 李永伦

Adobe 将会继续以 Cordova 加上 PhoneGap Build 和 Adobe Shadow 的组合提供 PhoneGap。

原文链接：

http://www.infoq.com/cn/news/2012/03/Adobe-Cordova-PhoneGap

ORM 工具 LLBLGen Pro 3.5 发布

作者 JenniKonrad **译者** 曹如进

ORM 工具 LLBLGen Pro3.5 版本已于本周发布 新版本支持了 Entity Framework 和 NHibernate、改善了设计器，并对运行时进行了一些更新，例如支持 OData 和 SQL Server 2012。

原文链接：http://www.infoq.com/cn/news/2012/03/LLBLGen-Pro

Django 未来将支持 Python3

作者 李湃

Django1.4 依然不支持 Python3 ,但 Django 团队计划在下一个版本(Django1.5)支持 Python3。

原文链接：http://www.infoq.com/cn/news/2012/03/django-python3

Windows PowerShell 脚本管理器 Beta 版发布

作者 Jonathan Allen **译者** 曹如进

微软发布了一款名为 Windows PowerShell 脚本管理器的新工具。目前该工具还处于 beta 阶段。该工具本质上是一个面向 PowerShell 脚本的专用搜索引擎。默认情况下，它允许管理员浏览和搜索 TechNet 脚本中心及 PoshCode 上的脚本。

原文链接：http://www.infoq.com/cn/news/2012/03/Script-Explorer

LINQ for F#

作者 Jonathan Allen **译者** 李永伦

F# 3.0 包含在 Visual Studio 11 beta 里，支持使用 LINQ 表达式。其他特性包括支持可移植库和自动实现属性。

原文链接：http://www.infoq.com/cn/news/2012/03/LINQ-FSharp

推荐编辑 | 翻译团队编辑 郭雪品

与 InfoQ 结缘，是刚工作时，为解决当时遇到的一个难题，当时的领导转了一篇 InfoQ 上的文章给我们，为解决问题提供了很好的思路。再去到 InfoQ 中文站上细读其他新闻和文章，才发现，自己原认为很先进很神秘的技术或工具，原来 InfoQ 上有很多以前的文章或新闻就已介绍过了，于是，从此果断把 InfoQ 当作了自己获知新技术、了解业界新动态的来源。

去年，带着对 InfoQ 的向往，给 InfoQ 编辑团队去了一封信，表达了自己想向社区投稿的愿望，很高兴很快就收到了崔康的回复，并由伯薇来做我的新人教练。经过几次试译和伯薇的指导，在 10 月份转为了正式的编辑。回想起去年发布第一篇译文、收到伯薇通知将成为正式编辑的邮件时，仍然非常兴奋。

InfoQ 编辑团队是一个很神奇的团队。你会发现，很多人其实都是在网上遇到过的高手。比如发现张龙就是以前学 Java 时，视频培训课件中的风中叶老师，还有一些人是某些书的翻译人员、一些又是 QCon 大会上的讲师、在某场大会上的演讲嘉宾原来也是 InfoQ 的成员……泰稳发展的这个队伍，说是卧虎藏龙，一点也不为过。能有幸加入这个团队，想想都是件激动人心的事（心中又一阵自豪）。

在 InfoQ 翻译是一件非常有收获的事情。要把一篇感兴趣的文章或新闻翻译出来，就需要深入了解它的细节，有时要查看原文中的每个链接内容，找到相关的信息，才能辅助把一些不确定的语意译出来。每一次翻译，对自己都是一次提升；同时，因为大家都在找新闻线索，自己也能了解更多的业界动态。

说来惭愧，加入 InfoQ 后，由于一些个人的事情，对中文站的贡献还不够大。今年希望能为 InfoQ 中文站翻译更多的文章，提高自己的产出；另外，尽力办好广州站的 QClub 活动，如果精力足够，希望能参加更多 InfoQ 的其他活动，有机会的话，也与更多的编辑见面多一些交流，再去北京看一看组织的办公环境。

最后我也做一下广告：目前我是 InfoQ 线下技术交流活动 QClub 广州站的负责人，预计五月初会有今年第二场 QClub 广州站活动，请关注 InfoQ 技术社区日历或 QClub 广州站微群，期待你的参与！

封面植物

天山雪莲

天山雪莲，又名“雪荷花”，当地维吾尔语称其为“塔格依力斯”。属菊科凤毛菊，属多年生草本植物。新疆特有的珍奇名贵中草药；生长于天山山脉海拔 4000 米左右的悬崖陡壁之上、冰渍岩缝之中；那里气候奇寒、终年积雪不化，一般植物根本无法生存，而雪莲却能在零下几十度的严寒中和空气稀薄的缺氧环境中傲霜斗雪、顽强生长。这种独有的生存习性和独特的生长环境使其天然而稀有，并造就了它独特的药理作用和神奇的药用价值，人们奉雪莲为“百草之王”、“药中极品”。

形态特征 本品茎呈圆柱形，长 2～48cm，直径 0.5～3cm；表面黄绿色或黄棕色，有的微带紫色，具纵棱，断面中空。茎生叶密集排列，无柄，或脱落留有残基，完整叶片呈卵状长圆形或广披针形，两面被柔毛，边缘有锯齿和缘毛，主脉明显。头状花序顶生，10～42 个密集成圆球形，无梗。苞叶长卵形或卵形，无柄，中部凹陷呈舟状，膜质，半透明。总苞片 3～4 层，披针形，外层多呈紫褐色，内层棕黄色或黄白色。花管状，紫红色，柱头 2 裂。瘦果圆柱形，具纵棱，羽状冠毛 2 层。体轻，质脆。气微香，味微苦。

生长习性 雪莲有着适应高山环境的生物特性。它的叶极密，状如白色长绵毛，宛若绵球，绵毛交织，形式了无数‘小室’。室中的气体难以与外界交换，白天在阳光的直接照射下，它比周围的土壤和空气所吸收的热量要大；到了夜间，它的温度又降低得很慢，所以能保暖御寒和防止水分强烈火蒸发。而绵毛层又可使机体免遭强烈辐射的伤害。

生物价值 雪莲花各地民间全草入药，主治雪盲、牙痛、风湿性关节炎、阳痿、月经不调、红崩、白带等症。印度民间还雪莲花来治疗许多慢性病患者。如胃溃疡、痔疮、支气管炎、心脏病、鼻出血和蛇咬伤等症。在藏医藏药上雪莲花作为药物已有悠久的历史。藏医学文献《月王药珍》和《四部医典》上都有记载。

1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 4 月刊

每月 8 日出版

本期主编：张凯峰　策划编辑：贾国清

美术编辑：水羽哲　流程编辑：金鑫

总编辑：郑柯　发行人：霍泰稳

读者反馈与投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：
http://weibo.com/infoqchina

商务合作：sales@cn.infoq.com 15810407783

本期主编：张凯峰，InfoQ 中文站原创团队编辑

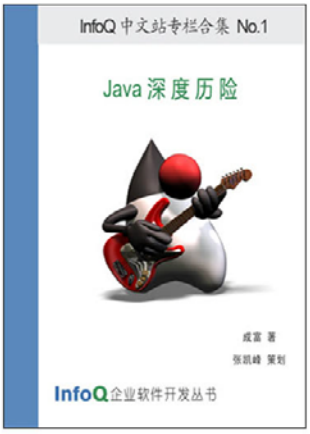
张凯峰，软件开发和测试工程师，参与翻译《Ajax 实战》、《开源技术选型手册》。

《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。

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