

架构师 ARCHITECT



解读2014 | Review

解读2014之前端篇

推荐文章 | Article

阿里云RDS团队专访

从商业角度探讨API设计

专题 | Topic

GitHub中国开发者年度分析报告

2014开源软件发展调查报告发布

GitHub开源大牛谈中国开源



QCon

全球软件开发大会

2015.4.23-25 北京国际会议中心



9折
优惠

现在报名

(截至3月29日)

- 针对软件开发领域的热点与趋势，议定18个最具实践价值的专题，覆盖更广的开发者人群，为参会者提供更多选择

可扩展、高可用架构设计

移动开发最佳实践

新兴大数据处理技术与工具

安全、隐私与风控

自动化运维

互联网金融背后的技术架构

云计算平台构建与应用

微服务和响应式框架

团队建设

Java和新锐语言

软件质量

超越JavaScript

敏捷之后，是什么

思考开源

知名移动案例分析

基于大数据的机器学习和数据挖掘

挑战全栈开发

技术创业

抢票热线：010-64738142

会务咨询：qcon@cn.infoq.com

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

卷首语

在业内，“建设生态圈”的说法从去年说到现在，但是要如何建设生态圈，想必是见仁见智，我认为“建设生态圈”我们或许可以借鉴小米的布局和经验。为何选择小米，因为小米在去年年底的最新一轮融资之后，小米俨然已经成为了世界上最值钱的未上市科技公司之一，在国内已是继 BAT 之后的又一计算机科技巨头。如此看来，新霸主的故事还是很值得人们学习和借鉴的。

生态圈的布局意识应该越早越好，据说雷军在小米成立之前就已有了生态的概念，这四年来自不断地演进和落地。据媒体分析，目前小米的生态圈有四层，最里面最核心的圈是自己的主力产品，包括手机、电视、路由器，第二个圈是小米和顺为基金投资的上百家上市公司，第三个圈是电商平台，第四个圈则是 MIUI 系统和支持的硬件产品。其中智能硬件生态链是贯穿其中的重要布局。雷军认为智能硬件是下一个风口，“连接一切”则是小米的愿景。

如何布局智能硬件生态链？小米的主要做法是投资。据媒体报道，在决定投资之前，小米曾经想自己来做，但结果并不理想。小米从 2011 年开始自己做移动电源，发现性价比极低，所以紧急叫停。投资的目的是让尽可能多的硬件产品和小米连接在一起，所以准备花 3 年时间投资 100 家公司。由此，包括小米手环在内，不断有科技创业公司加入小米生态链。为何从小公司开始合作呢，这里有一点农村包围城市的味道，当通过投资方式，小米初步形成自己的智能硬件生态链时，才有资格和更大的话语权与美的这样的国内电器巨头形成战略合作关系。

在建设生态链的时候，小米一直保持专注的态度，自己只做三件产品，手机（移动互联网入口）、电视（盒子，家庭娱乐入口）、路由器（智能家居入口），从而给生态圈更多的发展空间，与其合作伙伴的界限也非常明确。而且，小米和其投资的公司保持着一定的安全距离，据媒体报道，小米可以提供资金、供应链、技术、互联网思维，但这些创业公司却不能依附于小米，必须尽快产生价值。

小米的生态圈故事很长，我们在这里只是简单地回顾一下，对于相关的读者会有帮助。InfoQ 在这方面也在贡献自己的力量，为了推动云计算事业的发展，我们推出了《云生态专刊》，致力于“打造中国最优质的云生态媒体”，其创刊号已经[发布](#)，欢迎大家阅读。

——InfoQ 中国总编辑 崔康

目 录

解读 2014 | Review

解读 2014 之前端篇：任性的颠覆式改变

观点 | Opinion

唱衰 Docker——给大红大火的 Docker 泼点冷水

为什么不能用 memcached 存储 Session

Bowery 为什么从 Node.js 转向 Go

专题 | Topic

GitHub 中国开发者年度分析报告

2014 开源软件发展调查报告发布

虚拟座谈会：GitHub 开源大牛谈中国开源

推荐文章 | Article

关于有效的性能调优的一些建议

React Native：Facebook 出品，可用 JavaScript 开发移动原生应用

专访阿里云 RDS 团队：WebScaleSQL 是一个怎么样的数据库

从商业角度探讨 API 设计

封面植物

解读 2014 之前端篇——

任性的颠覆式改变

作者 黄丹

编者按

2014 年，整个 IT 领域发生了许多深刻而又复杂的变化，InfoQ 策划了“解读 2014”年终技术盘点系列文章，希望能够给读者清晰地梳理出技术领域在这一年的发展变化，回顾过去，继续前行。

本篇是解读系列的前端篇，小编邀请到天猫前端团队的三七、铁军、不四、鬼道这四位专家来解读 2014 年前端领域最引人注目的几大热点。

HTML5 正式定稿

2014 年 10 月 28 日，W3C 宣布 HTML5 正式定稿为[标准](#)，这不仅仅标志着历经 8 年的标准纷争告一段落，也代表着 HTML5 这个名词会逐步洗去铅华，其技术真正融入到 Web 开发的每个角落，就像当年 Ajax 一样，当大家不再大张旗鼓吹概念和商业炒作时，正是其成熟时代到来了。HTML5 规范和以前最大区别是让 Web 最基础架构从 Web Page 升级到 Web Application，正符合主流互联网从桌面端迁移到移动端的趋势，是移动互联网终端碎片化的一剂良方。在移动智能终端性能和网络速度到达消费者需求时，将会是 Mobile Web 大展跨平台神威时，这从网络基础设施领先的韩国应用从纯 Native 到大量采用 Hybrid 形式就可以看出一些端倪。

HTML5 在尚未定稿前已有了大量实践，以至于其定稿之日也是成熟之时，同样下一代的 JavaScript 标准 ECMAScript 6 虽尚未定稿，但浏览器端 Chrome 和 Firefox 等新版本已实现 ES 6 的部分语法（Promise、Generators 等），同样在服务器端，Node v0.11 最新版本也已支持部分核心 ES 6 语法。这些 ES 6 新特性，大大地提升了开发者的效率。在 Node.js 中，已经有了非常优秀的基于 ES 6 特性的新框架，并已开始广泛地运用在生产环境中。随着 Regenerator、6to5 等转换工具的出现，在前端使用 ES 6 的新特性也完全不是问题，这将大大提高整个 JavaScript 开发群体的效率，让大规模应用 Nodejs 的时代更快到来。

HTML 和 ECMAScript 是前端开发的基石，其快速升级和革新意味着这个领域面临的挑战和旺盛的生命力。

YUI 停止开发，小而美的 mv*库的流行

2014 年 8 月 29 日，Yahoo 宣布停止开发 YUI，如 Julien 在该博文所说，这几年前端行业越来越活跃，新技术和工具层出不穷，对于大而厚的基础库越来越不适应业务的发展需求。与此同时，单页应用技术符合 Web 应用化的趋势，一方面随着业务越来越复杂，前端 API 能力越来越强，数据和展现结合也越来越紧密，另一方面，Mobile 的发展对 Web 人机交互体验有更高要求，效果上要交互体验极致到 Native 的程度，性能上要前端库的高效且粒度及灵活性精细化，这也是类似 reactjs 等新型 mv*库开始流行的一个原因。

类似问题在 jQuery 身上也挑战很大，过去小而快的优点在移动时代已没有优势，需要面对移动端新的极致人机交互体验挑战。阿里开源框架 KISSY 正在使其核心模块粒度更加细小灵活，对低级浏览器的兼容拆分，在 Mobile 等高级浏览器下加载更少的代码，这是应对这一趋势所必须做的改变。前端框架和类库是为了提升前端开发的效率和品质，当人机交互环境发生重大变革时，这些基础设施都必须敢于大胆提早顺势而变，否则只能被淘汰。

基于 Web Components 的跨终端组件快速发展

2014 年通过指令化/声明式调用前端组件的形式发展迅速：比如 Angularjs、Reactjs 及新晋的 vuejs 等各种热议和实践，其中一个特点是 Directive 的引入。[Web Components 规范](#)将组件定义使用标准化，这种标准化正式跨平台跨终端业务急需的，为前端开发方案带来巨大机会。2014 年 Google IO 上《[Polymer and the Web Components revolution](#)》介绍，2014 年北京 QCON 豆瓣的《[DOMO UI](#)》、百度的《[跨终端组件实战](#)》，都是基于 Web Component 的实践落地（DOMO UI 类似 Web Component）。究其背后原因：一方面前端开发越来越富交互化，组件共享复用也越来越频繁，如何高效一致地使用是每个组件库需要解决的问题，而 Web Components 的到来让我们看到了机会；另一方面，Mobile 的高速发展，让前端开发不仅仅只面对桌面一个终端，更要面对 Phone、Pad 乃至 TV 终端，Web 和不同的 Native 开始混用，如何让 Native 代码也能像 Web 组件一样方便调用，就需要引入类似 HTML 之类得声明描述组件，而 Web Components 正式符合这一特性的原生标准，为组件的跨终端带来无限想象。面对消费者终端的碎片化，Web Components 会成为跨端 UI 模块化协作的基础。

目前天猫正在构建跨终端高品质 UI 体系 MUI，从设计到客户端和 Web 前端一起打造一套 UI 设计规范和模块化组件体系覆盖所有端的天猫业务，从 iPhone 到 Android Phone，从 iPad 到 Android Pad，从 Mobile Web 到 Desktop Web，还有 TV 等，实现任何标准的 UI 设计都能够快速覆盖全站，其背后技术思想之一就是 Web Components。

基于 Node.js 的前后端分离方案流行，同时社区和企业边协作边竞争使 Node.js 在稳定服务和创新发展中平衡

1989 年 3 月 12 日，Tim Berners-Lee 创立了 WWW (Word Wide Web)，Web 的迅猛发展成为 Internet 上最重要的内容承载方式，以至于很多人会认为 Web 就是 Internet。亿万互联网用户催生无数的 Web 开发者和巨无霸网站，Web 的规模化促使了前后端的分工，于是 2001 年雅虎有了全世界第一个前端工程师职位，此时前端专注于 HTML、CSS 和 JavaScript，后端专注于业务和数据，而数据 (Data) 和展现 (View) 结合部分由于成本较低和技术难度不高而分工模糊，大部分情况下这部分工作依旧是后端工程师在负责。2007 年 iPhone 诞生，互联网全面向移动快速进化，各种系统和硬件配置的 Phone 和 Pad 兴起使得用户访问互联网的终端碎片化，导致互联网产品都需要一套数据 (Data) 多个展现 (View)，所以 Data 和 View 结合的技术难度和成本剧增使得这部分工作必须从后端向前端转移，前端负责客户端和服务端所有的 View 及 View 相关的 Control，后端负责业务逻辑和数据并以 API 服务的方式向前输出，这样前后端彻底分离，对于产品开发而言前端只需要控制 View 和标准化的 Data 服务，不存在后端了。

前后端分离技术的难点是在服务端的前端，这个领域一直被后端开发语言和思想所统治，对于本来就很稀缺的前端工程师在技能和工作量上提出太高的要求，导致进展不顺利，直到 Node.js 横空出世。Node.js 出现，不仅让前端工程师终于有能力自己为自己打造提高工作效率的工具，让前端工程师发挥程序上的想象力，也让前后端分离有了更好的选择，所以整个业界非常多公司在这方面尝试，有些甚至尝试使用 Node.js 完全取代后端语言，比如 Java。目前还处于风起云涌的初期，所以即使在同一个公司如阿里巴巴内部都很多类似尝试，比如淘宝的 Midway、支付宝的 iChair 和天猫的 Wormhole 等，主因是难点并不在于 Node.js 技术本身，而在于和原有业务服务体系对接和运维能力上，所以切入点很多且难以标准化，先多点尝试相互竞争，后续在基于实际方案的基础上进行合并统一是我们目前的思路。天猫的首页已经构建在 Node.js 上，不仅经受了 2014 双 11 的

考验且性能表现优异，目前正在把这个方案应用到天猫所有活动和频道页面，到 2015 双 11 会有相当多的流量运行在 Node.js 上，那将是激动人心的时刻。

Node.js 开始大规模使用和其逐渐成熟完善且社区非常有活力关系密切，但从七月初开始，Node.js 核心开发者与 Node 社区核心参与者认为 Joyent 管理下的 Node 开发进展太慢，且对于社区的需求响应不及时，开始与 Joyent 公司进行谈判，希望将 Node 源码从 Joyent 公司拿出来，放到 Linux 基金会下基于社区来进行维护。最终事件以 Node.js 核心开发者 fork 了 Node 源码，重命名为 io.js 结束。2015 年 1 月 13 日，[io.js](#) 发布 1.0 版本，同时，node 也将发布 0.12 版本。这事件对 Node 社区影响非常之大。首先，造成 Node 源码的开发工作停滞了三个月左右，其次，io.js 开发活跃程度已经大于 Node.js，且 io.js 和 Node.js 的开发理念不同必然导致之后两者会渐行渐远，但从长远角度来看，竞争虽然带来阵痛但有利于更好的产品出现。

2014 年初，Node.js 当时项目掌门人从 Joyent 离职，基于 npm 创立了 [npm, Inc.](#)，开始致力于 Node.js 的包管理平台开发和维护。之前 npm 属于社区维护性质，服务不够稳定，随着 Node 社区的发展壮大，npm 服务的稳定性越来越重要，因此 npm, Inc 的成立保障了维系 Node 社区最重要的基础服务设施的稳定性。2014 年底，npm, Inc 发布新官网，同时重新定义 npm, Inc 为 JavaScript 的包管理工具和平台。此时 npm 已经拥有了接近 12 万个模块，超越了 maven 成为了最大的包管理中心。随着模块数量的急剧增加，企业使用 npm 的需求也越来越高，npm, Inc 开始将目标瞄准了企业版 npm 市场，现在处于邀请公司试用期。而早在 2014 年中，阿里巴巴内部的私有 npm 服务已经非常完善，现在已经有每月超过 300 万次的下载，服务于全公司的 JavaScript 程序员。所以，社区驱动了创新和快速发展，企业会推动服务稳定和健壮，两者相互协作和竞争会让整个生态更有旺盛的生命力。

Native 定义为前端的一部分，并开始深入融合

2007 年 1 月 9 日，iPhone 诞生，带来了整个人机交互领域的颠覆式创新，对于前端技术也有了颠覆式改变，初期甚至到了讨论 Web is dead 的地步，加速 Web 世界的危机感和积极向移动端转型，同时随着 Phone 和 Pad 的严重碎片化和整个互联网都从桌面转向了移动，直接导致移动应用内容的规模化和多元化及连接和整合整个世界，越来越发现仅仅靠 Native 本身是不够的，需要 Web 和 Native 结合起来才能够满足极致人机交互和规模化联通世界的要求，比如微信其实就是这方面的表率。前端的工作就是为人机交互的 UI 提供工程化方案，当整个互联网向移动转移时，原来的 Web 体系和工程方案已不适用了，这就是为什么 YUI 会倒下，而 HTML 和 JavaScript 要快速地推出革新版本，同样 Web Components 必

须满足移动终端碎片化的模块化方案才能高速发展，而 Node.js 的流行恰好迎合上前后端分离前端工程师需要掌控服务端前端的趋势。这一些也仅仅是刚刚开始，TV 和 Watch 等越来越多碎片化的终端进入到日常生活，前端的挑战也刚刚开始且前所未有，这是最好的时代。Web 是桌面时代人机交互技术方案的王者，但在移动智能终端时代目前无法及时满足新兴的人机交互能力。这非常类似 Ajax 到来时，HTML + CSS 已经无法很好地满足人机交互 UI 开发的需要，前端要快速掌握 JavaScript 一样，移动互联网时代，Web 前端工程师需要快速掌握 Native 开发能力成为跨终端的前端开发工程师，这不是抛弃 Web 转向 Native，而是把 Web 和 Native 结合起来，就像当年 HTML + CSS + JavaScript 结合起来产生巨大的威力一样。这不是 $1+1=2$ ，而是 $1+1>2$ 的问题，不是简单的技术领域扩充，而是真正的人机交互技术深入探索，前端技术方案从来都不是由稳定的单一技术所能解决的。现在 Native 开发规模越来越大已开始在探索类似 Web 的 View 发布机制和模块化依赖关系管理等等，而 Web 也正在探索弱网络或不确定网络性能、内存管理及硬件调用的技术方案，两者结合（Hybrid）对于我们的大规模平台化业务来说是事半功倍的正确方向。

ArchSummit2014 深圳大会，手淘 Android 负责人无锋分享的手淘 android 架构《[手机淘宝的客户端架构探索之路](#)》中提到“像 Web 一样轻盈的 Native App！”可见从 Native 同学的角度已开始考虑向 Web 融合，而在天猫我们定义：前端 = Web + Native，目前天猫已有 10% 的 Web 前端同学拥有 Native 开发的能力，预计不到 2015 年中会扩大到 50%。虽然目前已有大量的 Hybrid 应用和使用类 PhoneGAP 的混合开发，但 Native 和 Web 的深入融合远远不够，尤其是发布能力和大规模协作的能力上，以及对于组件、性能等方面的相互协作。之前一直讨论 Native 和 Web 孰优孰劣，谁取代谁，但经过 2014 相信更多人已意识到这是个伪命题，真正评判一项技术的价值是在业务场景中，选择合适且面向未来的技术最重要，需要思考如何用技术为用户和业务带来价值，天猫也正在前进的路上，随着越来越多人同时掌握 Web 和 Native，两者的协作会更加深入和相互发展，并作为整个前端的范畴带来更多的技术突破、效率提升和极致体验，而原来的 Web 前端工程师也会进行技能升级，勇敢地打破自我的壁垒拥抱移动端，尤其是 Native 技术，前端=Web + Native。真正成为跨终端的前端工程师。

对于天猫前端而言，在新的一年里，Mobile 会变革为主场，主要有三个很明确的方向：跨终端组件、大规模 Node、Native 和 Web 融合。

跨终端组件 MUI： MUI 是天猫统一的跨终端 UI 组件库，这是设计师、Native 开发和 Web 前端一起协作的全站性质项目。之前已经历了两个版本完成了基础视觉规范和 JS 组件规范及管理升级机制，新一年的重要方向是：跨终端。目前正在运行的 MUI3.0 核心是天猫内部称作 FEModule 的组件体系，就是一个完整

的组件规范（包括样式、脚本、模板和数据定义），实现前端、后端一致的组件体系，即一个模块完全独立，加上数据即可渲染，模块既可以前端渲染又可以后端渲染。MUI3.0 会基于 Web components 和 Native 组件融合规范，实现跨终端的组件体系。

大规模 Node: 首页在天猫双 11 中在稳定性及性能上已经被证实表现出色，同时对于前后端分离核心的数据 API 定义也有了系统化的规范和工具。新一代渲染引擎 Wormhole CDN 3.1 全网发布，支持 feLoader / feModule / 全局头尾，至此天猫应用、CDN、频道页环境的模版渲染环境基本都已经完善，Node.js 在天猫承担更多前端业务的时代已经到来。

Native 和 Web 融合: 2014 年我们在技术和组织结构上做了很多突破，尤其是组织上把 Web 和 Native 前端调整为以业务维度的一个团队，前端 = Native + Web，持续推动团队转型深入到 Native。我们要把 Native 的高性能和系统能力同 Web 的发布能力和规模协作结合起来，这其中也有 Native 和 Web 互调的 Hybrid API，利用 Native 的缓存和系统能力把 Web 的基础打开速度做到 Native 一样的通用方案等等。

智能移动终端带来人机交互变革不仅仅导致了前端开发这个职位需要自我突破革新、重新审视和定义，更导致 UI 设计师的设计场景发生翻天覆地的变化，从单一的鼠标键盘大屏幕变成了多终端的触屏声音陀螺仪传感器等，设计需要更透析这些新的人机交互形式和技术才能够面向未来。新的一年里，三七将开始负责天猫的 UED 团队，把设计和技术结合起来，就像 [D2 前端技术论坛](#) 理念那样“好的设计驱动技术创新，好的技术给设计无限想象”，MUI3.0 就是设计、客户端开发和 Web 前端结合的产物，但这只是开始，三七如是说，未来还将继续颠覆、成长、蜕变。

作者简介

本文作者[三七](#)是 QCon 上海 2014“没有后端”专题出品人，[不四](#)是该专题讲师。[鬼道](#)是 QCon 北京 2015 大会移动开发实践专题的讲师。更多精彩内容尽在 [QCon 北京 2015](#)，现在购票可享 8 折优惠。

唱衰 Docker—— 给大红大火的 Docker 泼点冷水

作者 Cal Leeming

从我上一次对 Docker 进行[评价](#)到现在已然经过了一年有余，想当初我在文章里对这套容器技术方案的架构设计缺陷与糟糕的用户体验做出了严厉的批判。不过在这段时间当中，Docker 项目也开始逐步走向成熟，迎来自己的[1.0 版本](#)并在 Amazon 的推动下[声名大噪](#)，但同时用户挫败感、[过度宣传](#)引发的指责甚至因[漏洞遭利用](#)而引发主机感染越来越多。当然，[Docker Hub](#) 中私有库的引入让用户不必再为了托管部署而运行自有 Registry 系统，再配合 webhook 以及同 GitHub 的紧密[集成](#)，Docker 看起来有一个良好的前途。

有鉴于此，我决定再给 Docker 一次机会，并以六个月为周期将其引入生产环境。结果非常糟糕，Docker 性能极差，而旁门左道的解决方案加上以用户体验为代表的种种短板简直令人抓狂。实际上，Docker 的性能表现实在太差，禁用缓存功能竟然能够加快 build 速度。

（感兴趣的朋友可以查看[reddit](#) 与 [ycombinator](#) 网站上与该主题相关的讨论内容）。

Dockerfile

Dockerfile 存在着一系列问题，它令人讨厌、充满局限、不伦不类而且包含根本性缺陷。假如要构建一个库的多个镜像，例如第二个镜像包含内容调试工具，但两个镜像拥有同样的基础运行要求。Docker 不支持这种做法（详见[9198 号问题](#)），我们无法扩展 Dockerfile（详见[735 号问题](#)），使用子目录会破坏构建上下文并导致用户无法使用 ADD/COPY（详见[2224 号问题](#)）或者“管道（piping）”（详见[2112 号问题](#)），我们也不能在构建过程中通过环境变量实现有条件的指令变更（详见[2637 号问题](#)）。

我们给出的[解决方案](#)是创建一个基础镜像，两个特定环境的镜像以及其它一些包括重命名以及 sed 替换功能的 Makefile 自动化。除此之外，Docker 中还有一些

意想不到的“[功能](#)”有可能导致环境变量\$HOME 消失，进而产生无用的错误信息。太令人厌恶了。

Docker 缓存/层

Docker 有能力利用 COW（即写入时复制）文件系统实现[缓存 Dockerfile 指令](#)，这一点与 LVM [快照机制](#)相似，而且直到最近都只支持 AuFS，而后者还存在大量问题。之后，[0.7 版本](#)引入了多种不同的 COW 实现方式以改进稳定性与性能，感兴趣的朋友可以[点击此处](#)了解更多细节信息。

然而，这套缓存系统不智能，它不能阻止单一指缓存（详见[1996 号问题](#)），产生了一些意料之外的[副作用](#)。它的运行速度也极为缓慢，如果禁用缓存并避免使用层，其构建速度甚至能够得到提升。而 Docker Hub 缓慢的上传/下载速度则让情况进一步恶化，这个问题我们将在下文作进一步评述。

这些问题均源自 Docker 整体所采用的糟糕的架构设计，这直接导致它即使是在完全不适用的情况下，依然会强制执行线性指令（详见[2439 号问题](#)）。作为构建缓慢的解决方案，可以使用支持异步执行的第三方工具，例如[Salt Stack](#)、[Puppet](#)甚至[bash](#)，它们完全能够达成层的目的而使层变得没用。

Docker Hub

Docker 鼓励用户通过 Docker Hub 进行社会化合作。用户可以在上面发布 Dockerfile——包括公开与私有文件。其他用户可以通过[FROM](#) 指令而不是复制/粘贴来继承并使用这些 Dockerfile。该生态系统类似于 AWS [市场](#) 以及 Vagrant [Boxes](#) 中的 AMI，从理论上讲还是非常有用的。

然而由于一些原因，Docker Hub 的实现存在缺陷。Dockerfile 不支持多 FROM 指令（详见[3378 号](#)、[5714 号](#) 以及 [5726 号问题](#)），这意味着只能继承单个镜像。此外，它没有版本强制。举例来说，dockerfile/ubuntu:14.04 的作者可以替换该标签的内容，这相当于允许用户使用软件包管理器但又不没有版本强制机制。而且正如下文所提到，Docker Hub 在这方面存在着令人沮丧的速度缓慢的限制。

Docker Hub 还拥有一套自动化构建系统，能够检测到库中新提交内容并触发容器构建。因为许多原因，这项功能也是完全没用。由于几乎不能定制，构建配置受到了极大限制，甚至无法支持最基本的脚本执行前/后的钩子。Docker Hub 采

用一套特殊的项目结构，一个项目下只能有一个 Dockerfile，这破坏了我们先前提到的构建解决方案，而且构建速度极为缓慢。

我们的解决方案是使用 [CircleCI](#)，它是一个优秀的托管 CI 平台，能够从 Makefile 触发 Docker 构建并推送到 Docker Hub。虽然这种方式无法解决速度慢的问题，但唯一的可选方案是使用我们自己的 Docker Registry，其[复杂程度](#)都到了荒唐的地步。

安全性

Docker 最初使用 LXC 作为默认执行环境，但现在，[0.9 版本](#)默认使用 libcontainer。这使得用户可以[调整](#)命名空间功能、权限，并且可以在使用合适的 exec-driver 时使用自定义的 LXC [配置文件](#)。

这需要一直在主机上运行一个 root 守护进程，而且 Docker 一直存在着[若干](#)安全漏洞，例如 [CVE-2014-6407](#) 以及 [CVE-2014-6408](#)。坦率地讲，这些问题起初就不应该存在。甚至 Gartner 公司也在其[追踪报告](#)中给出了糟糕评价，并表达了对 Docker 的不成熟和安全性问题的[担忧](#)。

按照设计，Docker 对于命名空间[功能](#)给予充分信任，这就导致其攻击面要比其它典型的虚拟机管理程序更宽。Xen 拥有 [129](#) 项 CVE，相比之下 Linux 则拥有 [1279](#) 项。在某些情况下上述问题并非不可接受，例如在 Travis CI 当中进行公开构建，但这对于私有、多用户环境来说无疑是危险的。

容器与虚拟机并不是一回事

命名空间与 cgroups 功能[极为强大](#)，允许一个进程及其子进程拥有一个共享内核资源——例如网络堆栈以及进程表——的私有视图。这种细粒度控制与隔离机制配合上 chroot jailing 与 [grsec](#)，能够提供非常出色的保护层。一些应用程序，如 [uWSGI](#)，可以在没有 Docker 的情况下直接利用这些特性的优点，而不支持命名空间的应用程序则可以利用 [firejail](#) 实现沙箱化处理。如果您有冒险精神，可以将这种支持直接添加到自己的容器化项目的[代码](#)中，例如 LXC 以及 Dokcer，从而在单一内核空间中利用这些特性的有点高效地运行多套发行版。[相较于](#)虚拟机管理程序，这种作法有时候会有降低内存使用率和减少启动时间的好处，但其代价就是降低安全性、稳定性以及兼容性。举个与 [Linux Kernel Interface](#) 相关的最糟糕的[极端案例](#)，在内核及用户空间中运行不兼容或者未经测试的 glibc 组合版本很可能引发意料之外的行为。

早在 2008 年 LXC 尚处于构思阶段时，硬件辅助虚拟化也仅仅诞生了几年时间，许多虚拟机管理程序都存在着性能以及稳定性问题。因此，虚拟化技术并没有得到广泛应用，而面对成本以及物理基础设施占用减少等优势，上述问题是完全可以接受的。不过如今我们的虚拟机管理程序在性能表现方面几乎与裸机设备不相上下，而且有趣的是，在[某些情况下](#)速度更快。另外，托管的、按需分配的虚拟机速度越来越快，成本越来越低，[DigitalOcean](#) 在性能与成本方面都要远远胜过 EC2，这也使应用程序与虚拟机之间进行一对一映射从经济角度讲成为可能。

[编辑意见]正如 [Bryan Cantrill](#) 提出的[观点](#)，虚拟化技术的性能将受到工作负载类型的显著影响。例如，IO 任务繁重的应用程序会导致性能[降低](#)。

在某些特定的应用场景中，容器化确实是恰当的解决方案，不过除非能够明确说明为什么在你的应用场景中选择此类处理方式，否则你可能应用使用虚拟机管理程序代替。而且即使使用虚拟化技术方案，你仍然应该利用命名空间的优点，而在应用程序没有对这些特性提供原生支持的情况下，像 [firejail](#) 这样的工具可以提供帮助。

Docker 并不是必须的

Docker 增加了一个复杂的侵入层，使开发、故障排查以及调试工作的难度大幅上升，它带来的问题往往多于它能够解决的问题。它在部署上也没有任何优势，因为你仍然需要利用快照实现响应式自动扩展。更糟糕的是，如果大家并没有使用快照机制，那么生产环境的扩展就会依赖于 Docker Hub 的稳定性。

目前有不少项目都在滥用容器化技术，例如 [baseimage-docker](#)，该镜像旨在通过运行作为入口的 init.d 简化检查、调试和兼容，甚至还提供一个可选的 SSH 服务器，实际上是将容器当作虚拟机对待，虽然作者本人以[无甚说服力的言词](#)反对这种观点。

总结

如果开发流程合乎情理，那么你已经明白 Docker 不是必须的。Docker 中号称能够带来助益的全部功能要么完全无用，要么实现很差，而其最大的优势直接使用命名空间就很容易实现。如果放在 8 年前，Docker 是一个有趣的概念，但以现在的眼光来看，它几乎是没用的。

修正/改进

从表面上看，Docker 还有很多事情要做。它的生态系统鼓励开发人员倾向于“[不可变部署（immutable deployment）](#)”这样一种理念，新项目能够更快更[轻松](#)地完成，在这一点上其实用性确实得到了许多数人的肯定。然而需要注意的是，这篇文章的主旨在于探讨 Docker 的日常使用和长远使用，包括在本地及生产环境中。

尽管前面提到的大部分问题都清晰易懂，但文章并没有提及 Docker 如何才能做得更好。有许多可选的方案可以替代 Docker，每种方案中都有自己的优点和不足，而我将在接下来的文章中进行详细阐述。

感兴趣的读者可以查看 [a-ko](#) 和 [markbnj](#) 的进一步讨论，前者讨论了容器化的长远影响，后者从技术上进行了反驳，你可能会觉得它们都非常有用。

我要对在百忙中抽出时间给出个人反馈意见的每位朋友表示由衷的感谢。看到文章受到大家的关注确实令人倍感振奋，而且在读过多位工程技术大牛——其中包括许多年来都一直给我启发的那些人——的回应后，我也颇有种诚惶诚恐之感。

查看英文原文：[Lets review.. Docker \(again\)](#)

AWSome Day
—让您大有作为



作为 AWS 基础的技术培训课程，AWSome Day 在去年广受关注，场场爆满。2015 年 AWSome Day 如期而至，3 月 24 日、27 日将分别在成都和广州举行，赶紧[报名抢位](#)吧。

为什么不能用 memcached 存储 Session

作者 谢丽

Memcached 创建者 Dormando 很早就写过两篇文章[\[1\]](#)[\[2\]](#)，告诫开发人员不要用 memcached 存储 Session。他在第一篇文章中给出的理由大致是说，如果用 memcached 存储 Session，那么当 memcached 集群发生故障（比如内存溢出）或者维护（比如升级、增加或减少服务器）时，用户会无法登录，或者被踢掉线。而在第二篇文章中，他则指出，memcached 的回收机制可能会导致用户无缘无故地掉线。

[Titas Norkūnas](#) 是 DevOps 咨询服务提供商 [Bear Mountain](#) 的联合创始人。由于看到 Ruby/Rails 社区忽略了 Dormando 那两篇文章所指出的问题，所以他近日[撰文](#)对此进行了进一步的阐述。他认为问题的根本在于，memcached 是一个设计用于缓存数据而不是存储数据的系统，因此不应该用于存储 Session。

对于 Dormando 的那两篇文章，他认为第一篇文章给出的原因很容易理解，而人们经常会对第二篇文章给出的原因认识不足。因此他对这个原因进行了详细地阐述：

Memcached 使用“最近最少使用（LRU）”算法回收缓存。但 [memcached 的 LRU 算法针对每个 slab 类执行，而不是针对整体。](#)

这意味着，如果所有 Session 的大小大致相同，那么它们会分成两三个 slab 类。所有其它大小大致相同的数据也会放入同一些 slab，与 Session 争用存储空间。一旦 slab 满了，即使更大的 slab 中还有空间，数据也会被回收，而不是放入更大的 slab 中……在特定的 slab 中，Session 最老的用户将会掉线。用户将会开始随机掉线，而最糟糕的是，你很可能甚至都不会注意到它，直至用户开始抱怨……

另外，Norkūnas 提到，如果 Session 中增加了新数据，那么 Session 变大也可能导致掉线问题出现。

有人提出将 Session 和其它数据分别使用单独的 memcached 缓存。不过，由于 memcached 的 LRU 算法是局部的，那种方式不仅导致内存使用率不高，而且也无法消除用户因为 Session 回收而出现随机掉线的风险。

如果读者非常希望借助 memcached 提高 Session 读取速度，那么可以借鉴 Norkūnas 提出的 memcached+RDBMS（在有些情况下，NoSQL 也可以）的模式：

- 当用户登录时，将 Session “set”到 memcached，并写入数据库；
- 在 Session 中增加一个字段，标识 Session 最后写入数据库的时间；
- 每个页面加载的时候，优先从 memcached 读取 Session，其次从数据库读取；
- 每加载 N 页或者 Y 分钟后，再次将 Session 写入数据库；
- 从数据库中获取过期 Session，优先从 memcached 中获取最新数据。

关于 memcached 的更多信息，可以查看[这里](#)。

Bowery 为什么从 Node.js 转向 Go

作者 李小兵

随着业务的发展、性能的挑战、需求的变更以及技术的更新，一个应用从某一个技术栈转向另一个技术栈是很正常和合理的，如淘宝从 PHP 转向 Java、[Twitter 从 Ruby 转向 Java/Scala](#)、[Linkin 从 Ruby 转向 Node.js](#)、[Groupon 从 Ruby 转向 Node.js](#)、[Dropbox 从 Python 转向 Go](#) 等。Go 语言从一面世就受到了很多开发者的关注，它能够提高开发人员的编程效率，它的并行机制使得开发者能够非常容易地编写多核和网络应用，开发人员利用它的类型系统能够很容易地构建出富有灵活性和扩展性的模块化程序。现在，越来越多的项目基于 Go 语言实现，如著名的开源容器 [Docker](#)、PaaS 平台 [Deis](#)、Google 的云计算平台 [Kubernetes](#) 以及国内的[七牛云存储产品](#)等。[Bowery](#) 是一个基于云技术的开发平台，其在 2014 年进行了一次从 Node.js 到 Go 的转换，且这次技术栈的变更加快了开发和部署的速度。近日，Bowery 对这次技术栈转换进行了[总结](#)，并归纳出了他们认为 Go 优于 Node.js 的一些原因。现对这些原因进行一个全面整理以供读者参考和学习，具体内容如下。

1. 强大的跨平台编程能力

Go 具有很强的跨平台性，基于 Go 的程序能够在不同系统(Linux、Windows、OSX 等)中编译，所以在开发过程中，借助 Go 能够轻松实现跨平台编译，而开发者只需设置不同的环境变量，这样就大大提高了开发效率。

2. 快速部署

Go 属于编译语言且具有跨平台性，从而使得 Go 开发的分布式应用能够运行在不同的平台上。在 Go 平台上，从测试环境到正式环境的切换无需额外的系统依赖，从而能够实现快速的部署。

3. 并发原语的支持

Node.js 没有提供较多的并发原语，仅有 I/O 程序或计时器运行在并发模式，所以 Node.js 在并发处理方面处于劣势，且很难构建出快速响应的跨程序通讯系统。而 Go 提供了语言级别的并发特性 goroutine、基于通道 (channel) 的通信机制和更底层的并发处理基元，如 [mutexes](#)、[wait groups](#) 等，Go 显得更加适合于构建高并发的应用。

4. 标准化的集成测试框架

用于 Node.js 的测试框架有很多，有些适用于前端测试，有些适用于后端测试，但多是第三方的测试框架，如 [Jasmine](#)、[Mocha](#)、[JSUnit](#) 和 [PhantomJS](#) 等。而 Go 提供了内建的完整测试包，如果开发者想编写一个新的测试套件，只需把 `_test.go` 文件添加到相同的包里即可。有关 Go 测试的更多相关信息，请读者点击[这里](#)查看，此外，Go 还提供了[使用 http test 包进行测试的文档](#)。

5. 标准库

如果使用 Node.js 开发一个应用，开发者不得不引入额外的依赖扩展库，从而增加了应用的部署时间和不稳定性。然而 Go 提供了标准库，标准库的好处是无需包含其他扩展库即可实现一个应用的开发，从而节省应用的开发、部署的时间，并且还能够增强应用的健壮性。

6. 强大的开发者工作流工具

除了使用 NPM 和脚本控制外，Node.js 没有提供真正、标准的工作流工具。而 Go 所提供的工作区布局能够帮助开发者建立标准化的工作流、规范应用的开发。尽管使用标准的工作区布局会损失开发的灵活性，却获得了一个结构化、有条理的[工作区](#)，该工作区包括三个根目录：`src` 用于放置源码包，`pkg` 用于放置编译包，`bin` 用于放置的是执行文件。把源码和依赖文件集中放到一个单一的工作空间是一个最佳的实践，这样使的团队成员都有一个标准的文档结构。此外，`gofmt` 也是一个非常使用的工具，它能以相同格式对代码进行格式化。

以上这些原因是 Bowery 根据自己的实际经历而得出，另外，还有其他公司/团队觉得 Go 值得喜欢的一些原因，如 MongoDB 的项目管理团队喜欢 Go 的智能、统一的开发体验，音乐分享服务 [Soundcloud](#) 团队喜欢 Go 严格的代码格式规则以及单一方式实现功能的理念。总之这些特征能够节省针对代码规范和格式审查所花费的时间，从而使得开发者能够集中精力来解决关键的问题以提高工作效率。

最后作者为 Go 语言开发者提出了几点建议，如经常访问[官方博客](#)和[官方学习文档](#)、访问 Bill Kennedy 的[Go 编程博客](#)等。

此外，关于 Go 和 Node.js 的选择，Node.js 社区最活跃和高产的成员之一 [TJ Fontaine](#) 在个人博客中公布了自己放弃 Node.js 而转向 Go 的[原因](#)，主要是因为 Go 更适合高并发和分布式应用开发；最高产的 Node.js 开发者之一 [Duncan Smith](#) 在个人博客中列举出了自己[为什么不从 Node.js 转向 Go](#)的原因，有兴趣的读者可以前去阅读。

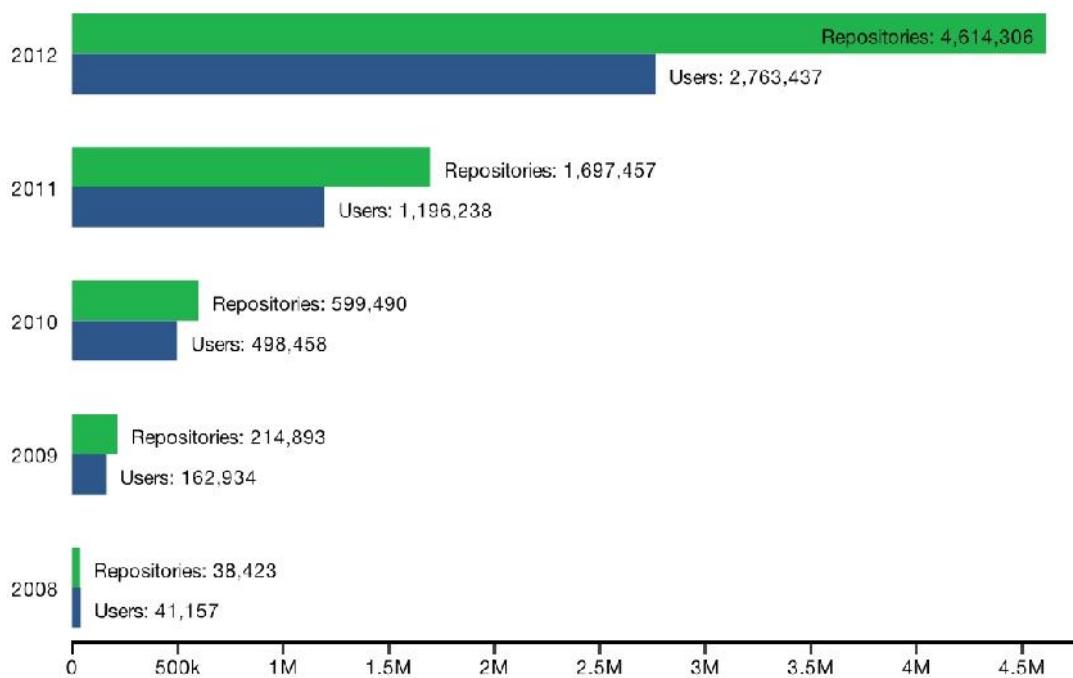
GitHub 中国开发者年度分析报告

作者 郭蕾

近日，GitHub.info 以 [GitHub API](#) 为基础，结合 GitHub 用户的地理位置信息，整理发布了 [《2014 年 GitHub 中国开发者年度报告》](#)。报告中分析了 GitHub 中国用户的比例、活跃用户数、活跃时间段、仓库数量、PR 数量、项目 Star 数量、组织等信息，并就相关指标与美国开发者做了对比。同样，GitHub 官方也会发布 [年度的用户报告](#)，InfoQ 编辑结合官方与社区发布的数据，对 GitHub 上的中国开发者情况做了一个简单的回望。

1. GitHub 用户数

GitHub 成立于 2008 年，是一个社区型的代码协作平台。根据 [2012 年的报告](#) 显示，GitHub 在 2008 年就有 4 万多的用户，在 2009 年迅速增长为 16 万，根据官方发布的增长数据可以推断出 2013 年已经有近 600 万（具体数字是 5843193）用户，新增用户有 300 万之多。GitHub.info 社区通过 API 计算得出，截止到 2015 年 1 月 20 日，GitHub 用户数已经超过千万（10475867）。如果这个数据准确，那 2014 年 GitHub 的用户增长就有 400 万。

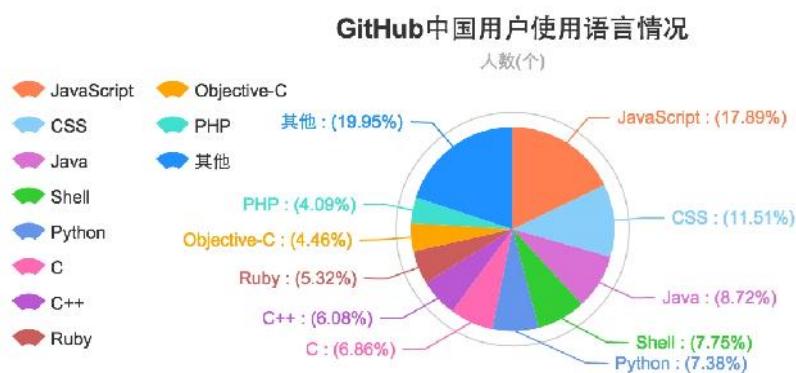


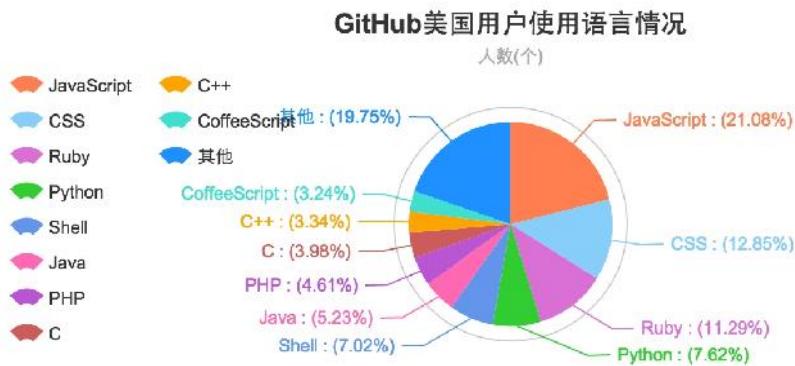
从 2012 年的报告中得知, GitHub 中用户数最多(官方给出的数据是访问量最多,这里姑且认为访问量和用户数成正比)的五个国家分别是美国、德国、英国、中国和日本, 2013 年是美国、德国、中国、英国、印度, 而据[微博](#)上一位 GitHub 员工透漏, 2014 年中国已经成为 GitHub 用户数最多的第二大国家。2013 年官方公布的数据显示, 所有的注册用户中中国用户占 5.8%, 如果以此比例来计算, 那现在 GitHub 上的中国开发者大概有 60 万。

2. 开发语言

2014 年 5 月就曾有分析机构[利用 GitHub 来统计分析编程语言的发展趋势](#), 分析结果显示 GitHub 上主流的五种开发语言分别是 JavaScript、Ruby、Java、PHP 和 Python, 其中 CSS 占的比例也比较大, C 和 C++ 处于中等水平, Go 之类的新型语言体量还是比较小。GitHuber.info 对中美开发者的语言情况做了对比, 其中 JavaScript 和 CSS 占绝对优势, 两者加起来接近 1/3, 这也不难理解, JavaScript 和 CSS 是前端开发的必备语言, 更何况还有 Node.js 之类的服务器端 JavaScript 语言。而排名第三的语言中美国和中国分别是 Ruby 和 Java, 紧接着是 Shell 和 Python。值得注意的是, PHP 并没有进入前五, 这也可能和该语言的使用场景有关(见过最多的 PHP 类开源项目就是各类 CMS)。两个国家的开发者中使用.NET 的用户都比较少, 这和微软一直不鼓励开源的社区生态有关, 随着新一年微软在开源方面的投入, 相信.NET 相关的开源项目会逐渐增多。

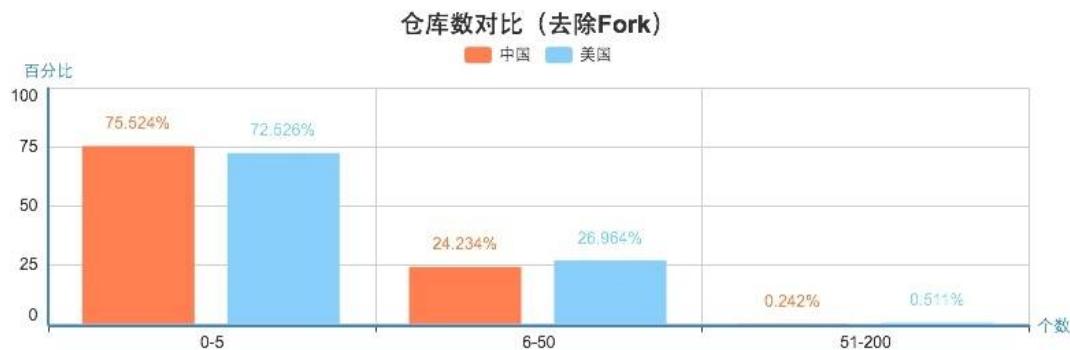
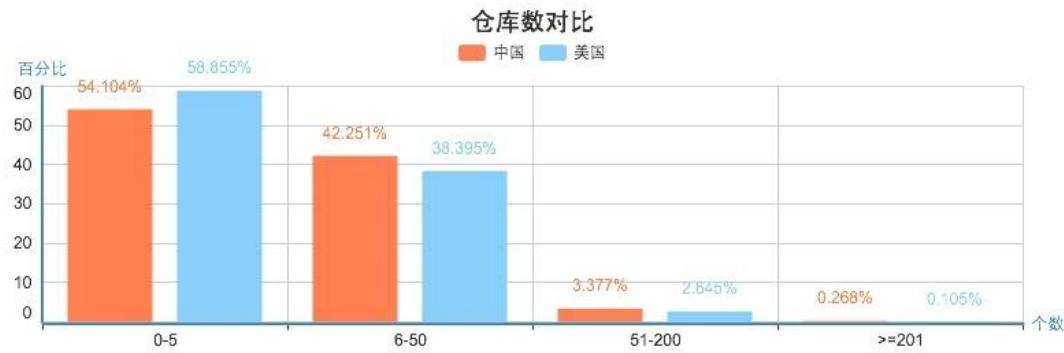
需要注意的是, 由于 GitHuber.info 统计国家时只能根据用户的 Location 信息识别, 而很多用户都没有填写 Location 信息, 所以文中涉及到 GitHub 方的数据时均指带有 Location 信息的用户。





3. 项目情况

从仓库数来看，大多数开发者（75%）都拥有 0-5 个仓库，中美开发者差别不大。由于统计的是去除 Fork 的仓库，所以 0-5 个也属于正常情况，一个用户能玩转属于自己的几个开源项目，已经非常不错了。



而考量项目的质量可以从项目的 Star 数量来看，同样，绝大多数的项目都只有 0-5 个 Star，其中中国开发者的比例为 67%，美国开发者为 80%。如果以 Star 大于 1 万的标准来衡量优质项目，那中国没有，Star 数量最高的是 [awesome-python](#)，有 9393 个 Star。

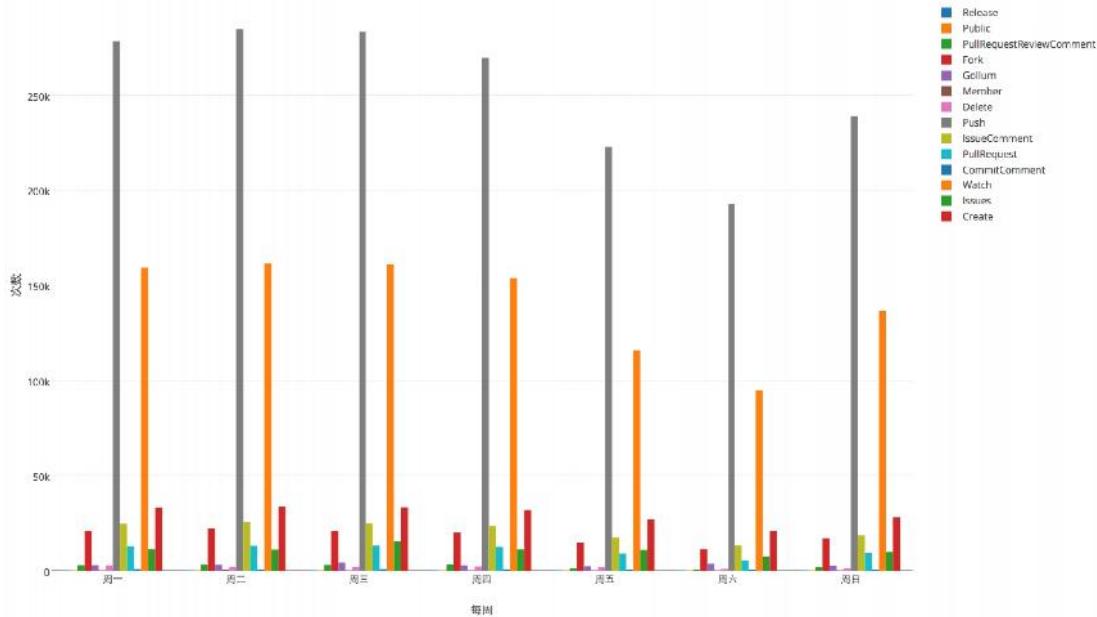


从项目的代码量来看，中美开发者悬殊比较大，GitHub.info 统计到的数据中美国用户的代码数量(字节)大多在 1 万到 10 万之间，而中国用户大多在 0 到 1000 之间（也就是基本为 0），美国开发者的代码量远大于中国。从分支数来看，美国和中国的对比相差不多，大多项目的分支数量都在 3 个以下，这也反映出大部分项目还是纯个人业余开发，并没有达到需要规范分支的程度。但同样由于美国开发者基数比较大，所以总体来看，美国开发者好于中国开发者。另外，90% 的项目的贡献者不超过 5 个人，90% 的项目几乎没有 PR 和 Issue，所以在人力投入比较少，且没有反馈的情况下，大多数的项目都没有长远的发展。

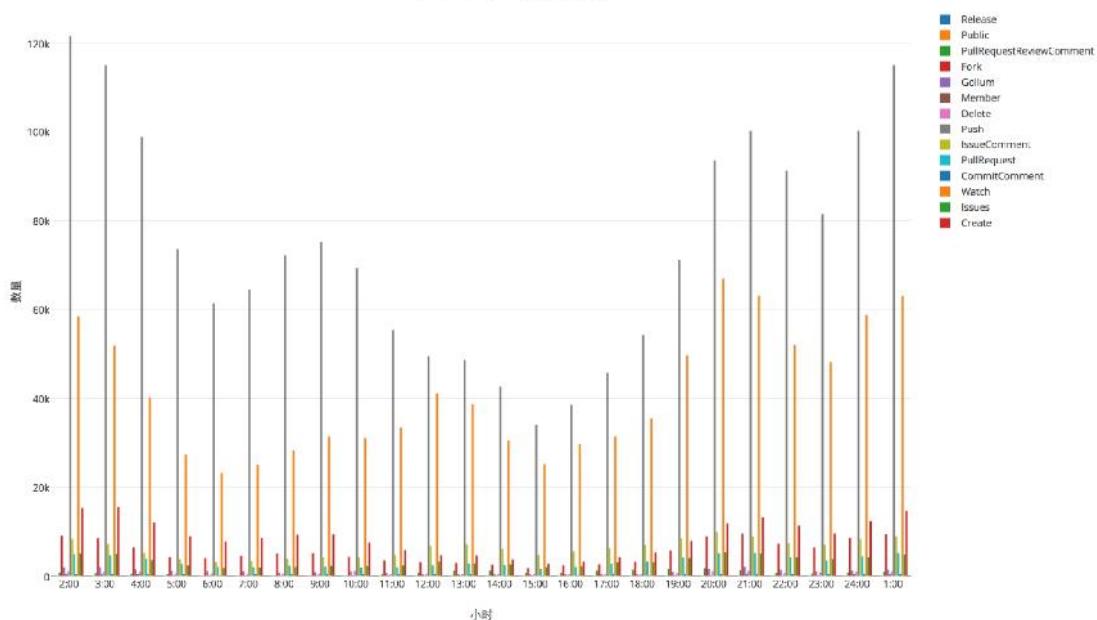
4. 活跃时间

GitHub.info 官方没有列出美国开发者的活跃时间，这里将官方 2013 年的数据作为对比，与美国开发者一样，一周中中国开发者最活跃的是周二和周三，最不活跃的是周六。比较有意思的是，周末两天美国开发者的活跃度都比较低，而中国开发者周日非常活跃，甚至比周五的还要高。细化到每天，更是伤人心。中国开发者每天最活跃的时间是凌晨 0 点到 2 点之间，而美国开发者是上午 9 点左右和下午 1 点左右。总体来看，代码推送的活跃度上午明显高于下午，美国开发者是白天明显高于晚上，而中国开发者是晚上明显高于白天。

每周平均动态



日平均动态

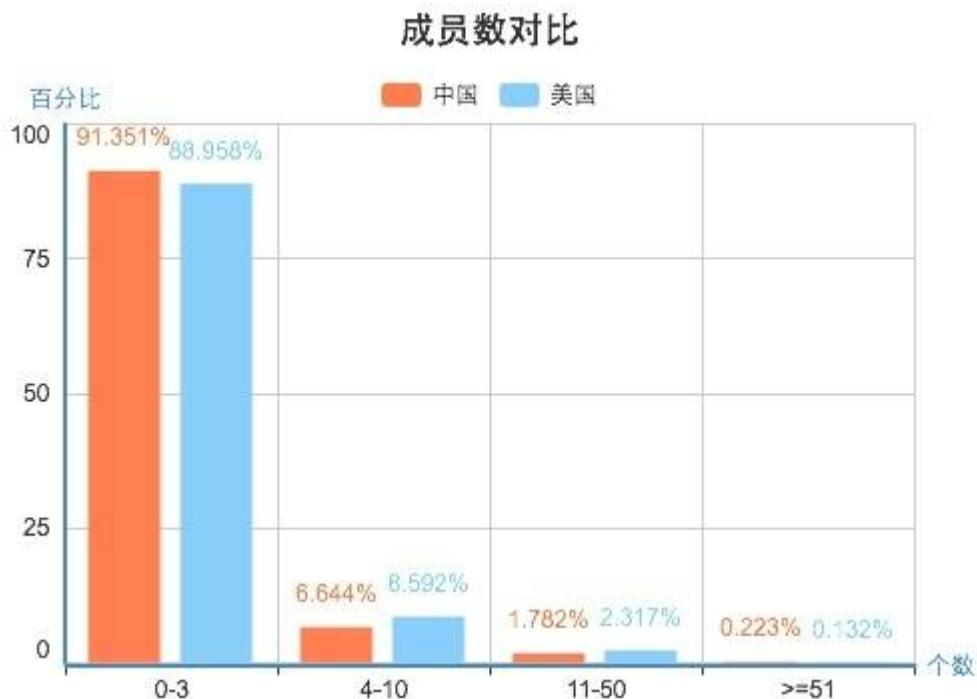


5. 组织

组织是 GitHub 新的账号管理模式，旨在满足大型开发团队的需求。组织是非登录账号，需要以 GitHub 用户身份登录，然后再创建自己的组织。截止 2015 年 1 月 20 日，GitHub.info 根据 API 统计到一共有 2500 个中国区组织和约 25000

个美国区组织，GitHub 的用户数美国是中国的 4 倍，但是组织数美国却是中国的 10 倍，看来中国使用者并不喜欢团队协作。

组织部分总体的分布趋势和项目很类似，不过各项统计按比例来算要比项目信息稍差一些。按理说组织中的项目应该更容易吸引成员参与，从而在 PR、Issue 上有更好的表现。反过来说，目前中国的组织并没有充分发挥组织的作用，对开源项目的发展推动不大。目前国内的优秀开源项目主要还是依靠线下团队的合作开发，距离美国开源项目的众包模式还有些差距。



刚刚从 GitHub 官方邮件中得知，GitHub 今年不会再发布 [Octoverse 报告](#)，不过感兴趣的读者可以邮件 press@github.com 获取一些官方可以公布的数据，InfoQ 正在与官方联系，希望获取一些中国开发者的详细报告，我们也会及时公开各项数据。从 GitHub 用户增长数量也可以看到开源的趋势，随着国内开源环境的成熟，优质的国产开源项目也越来越多，InfoQ 也会全程跟踪报道国内的开源状态，敬请关注。

感谢 GitHuber.info 的梁杰对本文的贡献。

2014 开源软件发展调查报告发布

作者 曹知渊

每年，Black Duck 软件公司和 North Bridge 风险投资公司都要进行开源软件发展（Future of Open Source）调查，今年已经是第九个年头了。

Black Duck 软件公司在 [slideshare](#) 上分享了完整的报告。这份长达 87 页的报告为读者提供了关于 2014 年开源世界变化趋势的深刻见解。

2014 年参与调查的人（包括公司）数持续增长，从 2013 年的 822 人，一下子增加到 1240 人。这些参与者中，42% 的人（或公司）在开发开源软件，而其余的则不是开发者。软件开发者、公司 CEO 或创始人、IT 管理这三类人占了很大比例。

开源软件在不断改变世界。从 2013 年的调查开始，质量成为人们选择开源软件的第一大因素，而 2014 年，质量因素继续扩大领先优势，10 个人中有 8 个人因为质量而选择了开源软件。Black Duck 的总裁兼 CEO 说道：

开源软件已经证明了自己的质量和安全性，现在应该推广，使其大众化。

还有几个重要的因素也是大家选择开源软件的原因。可以得到源代码，这个因素从第 8 位上升到第 4 位。80% 的参与者选择了出色的功能，75% 的参与者认为高曝光率的开源软件更加安全，另外易于部署这个因素，也从第 6 位上升到了第 3 位。

报告认为开源软件的未来将是一个良性循环。

首要环节就是人。小型的开源软件项目正在呈现爆炸式增长，其中的首要因素就是很多开发人员首次加入到开源软件的大军中来了。CHEF 公司主管市场的副总裁 Jay Wampold 评论说：

开发者是业务增长和革新的发动机。这份报告显示开源软件和技术正在吸引一个独一无二的开发者社区。

开源软件也带来了职业机会。GitHub 为开源软件带来了一场革命，Twitter 收购了开源培训公司来建立“Twitter 大学”，一代开源软件程序员正在脱颖而出……超过 50% 的企业或使用开源软件，或给开源软件带来贡献。它们从中获得的最大收益就是降低了成本。它们的员工也乐于参与其中。

当人和企业都积极参与时，开源软件带来的应用和服务会大量增长。根据报告的统计，云计算/虚拟化、内容管理、移动、安全、协作、网络、社交媒体这些领域，开源软件技术已经占据了领导地位，比例从 63% 到 46% 不等。在 3D 打印和智能商务领域，开源软件的势头也非常迅猛，分别达到了 27% 和 26%，而无人机、游戏和 ERP 领域，则是开源软件的未来阵地。

开源软件也影响着 IT 以外的其他各个行业。在这些行业中，教育业高居榜首（76%），其次是政府（67%）和医疗（45%）。教育系统中的账户和密码管理系统、教室和实验室、教学工具等方面都运用了大量的开源软件。Stony Brook 大学的 Web 及 IT 策略总监 Kerrin Perniciaro 说，很多大学都已经在使用开源方案，并且会坚持下去，因为开源软件成本低，部署快。同时，开源软件的繁荣也给个人生活带来了深远的影响。移动设备、可穿戴设备、物业管理、汽车等日常生活的各个方面，都有开源软件在运行。

未来的世界是物联网的世界。互联网起初是不同地点间的互联，发展到人与人之间互联，最后是万物相连。报告援引 2013 年爱立信的一份数据指出，到 2020 年，世界上可能会有 500 亿的物品互联到一起。报告认为基于这样的趋势，物联网领域是开源软件未来大显身手的地方。Intel 副总裁、Intel 开源技术中心总经理 Imad Sousou 认为，无论你在网上做什么，免不了要和开源软件发生关系，开源软件无疑将主导计算、移动互联和即将到来的物联网。

个人和厂商积极参与，内容和服务空前丰富，企业和个人生活从中获益。以参与促进繁荣，以繁荣促进参与。这样的良性循环给开源软件带来美好的未来。读者可以从[这里](#)查看这份报告的完整内容，并且可以到[这里](#)参与 2015 年的调查。

虚拟座谈会

GitHub 开源大牛谈中国开源

作者 崔康

最近，githuber 发布了一份“[GitHub 中国开发者 2014 年度报告](#)”，比较详尽地分析了中国开源的发展现状，其中对 star 排名前几位的开源项目负责人进行了访谈，InfoQ 整理了其中比较通用的部分，尝试总结出这些大牛们对中国开源的发展看法和建议。

githuber 采访的开源大牛分别是：

- ECharts 林峰
- ThinkJS 李成银
- Cocos2d-x 林顺
- Pen 小鱼
- Vue 尤雨溪

对国内开源现状的看法

李成银：目前国内的开源项目基本上都是团队内部在开发，即使是非常成功的项目 PR 也非常非常少，所以目前来说国内的开源环境仍然不够活跃不够开放。一个项目出来会被很多人骂，不过关键就是别人骂了我们我们还不知道，也就无法改进。我们觉得骂本身不是坏事，说明用户还是需要你的项目的，只是项目不够好而已。但是关键是骂也要去 Issue 里骂，这样我们才能看到。总体来说国内的开源环境已经比之前要好了，Issue 多起来了，PR 也有一些，不过目前来说还不够成熟，不能像国外的项目一样能够通过 PR 完成很多功能。

其实也不能怪大家，国内和国外的工作情况就不一样。国外大家把编程当兴趣来做，工作也没有国内这么忙，所以有更多的时间和兴趣投入开源项目中。国内经

常加班，压力很大，大家对于开源的热情就不高，更多的是把开源项目当做一个宝库，遇到问题的时候去找现成的解决办法，而不是参与其中。

此外，大家更喜欢用国外开源项目还有一个原因，就是国外的项目更加稳定，不太容易出现项目无人维护的情况。国内的开源项目有时候开发者会放弃并停止更新，这样依赖这些项目构建的项目就会很难处理，而国外的开源项目即使维护者停止更新，他也会找到其他人继续维护，比如前段时间的 Express，这就让用户很有安全感。

个人、团队、公司在开源项目的不同角色和选择

可以总结一下 ECharts 的发展模式吗？比如先从兴趣出发最后得到公司支持，或者从诞生就由公司支持和运作等等。如果其他开发者也想遵循这个模式的话，有什么话想对他们说吗？

林峰：我很幸运，这事本身就是自己的兴趣，也是公司的需要，遇到各种好领导放任自如的让我随意发挥，并且当事情越做越大的时候能够不断的给予支持和帮助。

做好一个项目，最最重要的，团队的力量，要找到那些志同道合，才华横溢的小伙伴与你并肩，ECharts 团队是个跨部门的虚拟组织，我们面向百度全体 FE 招募，组建时我立了个规矩，“如果你忙或者没时间做这件事情，请暂时离开，我们随时欢迎你归来”，刚开始几个月我两周清一次场，各种进进出出，但半年后团队就基本稳定至今了，用叮叮的话我们也成了一个小小的家庭。

你觉得公司在开源项目中扮演的是什么角色呢？在当前国内的大环境下，如何在公司中做一个成功的开源项目呢？

林峰：公司是不会无缘无故启动一个跟自身业务无关或者自己都用不到的项目。好的项目能获得更多人的关注、反馈、代码贡献，开源后如果能让这个项目发展更好不仅对公司自身项目需求有意义，甚至可以让公司在某个领域确立自己的技术领导地位（想想看 Android、Linux、jq、Bootstrap），这无疑是对公司极好的事情。

公司是否可以运作一个开源项目跟公司状况和基因有关，这个真不好说。只能说一点，在公司内做开源项目，这个项目本身是否对公司带来价值是关键，短期的长期的都最好要有。

什么事都是有利有弊，你觉得公司主导的开源项目相比个人或者社区主导的开源项目利在哪里，弊又在哪里呢？

林顺：有公司或者资本提供支持的开源项目，相对于个人或者没有资本支持的开源项目的优势：有更多的资源投入，对开源项目的后期发展至关重要，允许有更多专职的研发人员，产品的迭代周期和质量也能得到很好的控制，提供更加持续长久的维护，可以让开源产品走的更高、更远。至于弊端，那就得看对开源项目的态度，如果本着服务行业，推动行业升级，用开放的心态来做开源项目，并不会有存在着什么弊端，全世界范围内也并不乏有各个公司支持的开源项目。当时我们的操作系统公司做的不好了，引擎项目发展的却是很好，愿意投资我们的有好几家，但是最后还是觉得陈昊芝思路很开放，能坚持不把一个开源的项目做成闭源商业项目，最终和他一起做，一路走来，也发现我们当初的选择是最正确的。

如果其他公司也想走开源路线，有什么话想对他们说吗？

林顺：非常欢迎一起加入开源路线，开源项目不论是对个人和对公司，能学习到很多宝贵的知识，社区里汇集的智慧是巨大的宝贵的，国外资深程序员教你两招，你就能发现原来代码还能这么写，框架还能这么设计优雅。另外，和社区做好互动，有效采集用户需求和反馈，是推动开源项目往正确方向发展的关键，也是产品化和易用化的捷径。

你觉得作为个人开发者来开发和维护一个开源项目难度大吗？中间有没有想要放弃的时候呢？

小鱼：难度还是要看项目吧。不过我相信人多不一定能解决问题，因为技术问题普遍都有天花板，对于核心思想和技术，大多情况下应该是由更少的人产出的。思想定了，核心技术定了，添加功能可能并没有想象中那么难。而对于有没有放弃，我通常是这样想的，最差的可能就是放弃。坚持会发生很多美好的事情，比如写博客。如果有很多紧要的事，时间不多，有时候也只能放弃，只做觉得紧要的。

尤雨溪：这要看项目的规模了。一般来说适合个人维护的项目，最好是专注于解决一个较小的专门问题的库，否则可能会占用过多精力。项目的规模大到一定程度以后，最好是由社区或者团队来共同维护。说实话 Vue 现在的 issue 增长速度已经挺累人的了，好在现在也有很多社区开发者会积极地帮忙回答问题，让我省了很多精力。

你觉得纯个人开发开源项目和有公司背景的开源项目比起来有什么本质的区别呢，作为一个开发者应该如何选择？

小鱼：本质上都是开源。个人并没有统计过个人开源的东西更成功，还是公司开源的东西更成功。不过像 ElasticSearch 大多代码都是一个人写的，非常成功；Docker 是一个公司维护的，非常成功；Bootstrap 是 2 个人开发的，公司维护的，非常成功。本质上我觉得是开源的产品真正有用，就会有人用；如果有公司给时间和金钱支持，那相当好；而最好的是有一个社区，大家一起维护。比如你可以在 Google 上找到关于 jQuery 的几乎所有答案，这就是社区的力量。所以如果你有一个好的项目，那么尝试培养一个社区，比一个人写，或者只有公司支持没有人开发的僵尸项目好。

尤雨溪：有公司背景的开源，其背后肯定是有商业利益的推动，所以只要公司的商业利益和项目的发展状况是正相关的，这个项目就会有比较稳定的财力和人力支持。但这类项目通常更受公司决策的影响，对社区的意见不如个人开发的项目来得敏感。个人觉得选择一个项目的时候是个人还是公司开发并不是关键，关键是看背后的公司/个人是否靠谱。

开源对程序员的意义

林峰：太有意义了，学习啊！看大牛们的代码是一种幸福，从模仿到领悟到融入自己的程序里，这就是成长。身边很多大牛们都把 GitHub 视为游乐场或者玩具店，不是说儿戏了，是要有玩家的态度和享受玩的快乐，要在上面学会折腾，GitHub 上有无数好的项目，多动手，多折腾，尝试融入到这些开源社区去做些贡献，一开始哪怕就是跟 Issue 凑热闹，给些使用反馈，文档错别字纠正都是有意义的，然后就是贡献自己的想法，帮助别人解决问题，当你开始贡献代码，或许你就能体会到开源对你的意义。

参与开源项目对于程序员来讲是一种高效、快速学习成长的方法，不仅如此，如果你是一个技术爱好者，参与开源项目你有可能找到自己的兴趣，擅长结合点。当然，如果能找到和商业的结合点，进而从事自己喜欢的工作，那就更爽了，这点是很难得的。

一般有秩序的开源社区都提供很好的知识和经验交流平台，深入参与到开源项目中，对个人的技术成长和视野会有很大的帮助。

GitHub 在全球的火爆程度无需多表，提供非常高效的项目开发协作机制，是了解开源项目运作机制的好入口。在 GitHub 上，开发人员可以随时与全世界的人共享代码，也允许接受来自全球不同地方的人贡献各种 idea，代码片段，也是社区交流的基础，越来越多的开源项目迁移到 GitHub 上。

小鱼：开源是一种共享的精神。意义可能有很多种。让别人受益，自己得到改进反馈，让更多人从代码认识你，诸如此类，于每个人不同。开源并没有直接改变过我的生活，不过我喜欢写写代码，还能帮到人，于我已经是很大的乐趣，而有乐趣的生活就是我的意义。

对于 GitHub，他只是工作/协作平台，这样的平台还有更多选择。不过我一直用它，是因为其他产品都做的太丑，无论是细节还是体验，而我更愿意选择好用的工具，即使付费。

尤雨溪：我觉得开源的意义对于普通开发者来说，可以看别人的源码学习自然是最重要的了。在 GitHub 上利用高级搜索去搜自己语言排在前列的项目和开发者，可以学到很多东西。另外每周看看 trending 的新项目也可以发现很多好东西。另一方面，尽可能多地开源自己的代码也有好处，因为这可以迫使你对自己的代码保持一个高水准的要求，而不是得过且过。

感兴趣的读者可以访问 [githuber](#) 查看 GitHub 中国开发者 2014 年度报告和完整的采访内容。

关于有效的性能调优的一些建议

作者:李小兵

只有采用有效的性能调优手段，才能使得性能调优达到事倍功半的效果。近日，个人博客 Liganglei 中发布了一篇关于有效性能调优建议的[文章](#)，该篇文章是作者阅读《性能调优：综合指南》的读书笔记。作者从影响系统性能的算法、算法运行环境与所需资源以及算法和环境资源的交互等因素讲述了性能调优的一些建议。新手能够直接根据这些建议进行系统调优，老手也可以拿来当作调优的参考。现对这些建议进行一个全面的梳理，以供读者参考和学习，具体内容如下。

1. 算法本身的优化

算法优化是性能局部优化的首选，并常采用各种性能监控软件来度量 CPU 时间、内存占用率、函数调用次数以问题定位，然后实施各种调优方法，如优化循环、利用空间换时间、采用合适的数据结构等。但是算法本身的优化只能帮助大家消除一些明显的编程细节引起的瓶颈，尤其单单通过算法优化的手段还不能完全解决性能问题，且具有非常大的难度。

2. 优化运行环境与资源

运行环境与资源包括各种软硬件平台，硬件环境包括 CPU、内存、磁盘以及网络等。最简单且最省事的调优方法是优化硬件资源，使用快速计算资源代替慢速计算资源，提升资源的计算能力。

优化硬件资源的方式包括：

- 更快的 CPU；
- 更快的本地 IO 设备，比如内存代替硬盘，SSD 代替机械硬盘；
- 加内存减少分页；
- 快的网络 IO 设备，比如使用光纤及专线增加网络带宽，使用万兆千兆网卡代替千兆百兆网卡。

快速计算资源代替慢速计算资源，比如快速存储代替慢速存储（属于同类型资源）；本地计算换网络传输的优化最好采用压缩传输内容的优化手段（属于不同类型的资源），该方式尽管增加了 CPU 的压缩/解压时间，但减少了大量网络传输时间。

软件环境包括操作系统、数据库、中间件等。软件环境调优的成本要相对较高，并且工作量很大，如从 Windows 平台迁移到 Linux 平台、从数据库 A 切换到数据库 B、从 EJB 切换到 Spring 等。这类调优见效快，但受制于预算和硬件本身的限制。由于资源始终是有限的，随着资源的消耗，仍然存在性能瓶颈。

3. 优化算法和资源间的交互

当前各种调优实践最集中的领域是优化算法和资源间的交互，如减少单台服务器（或单位计算资源）的处理量、充分利用系统资源、减少不必要的计算、减少不必要的 IO 等。

具体内容如下。

□ 减少单台服务器(或单位计算资源)的处理量

当在单台机器处理能力已达上限的情况下，就需要把压力分散到多台机器上，从而使每台机器都能获得可接受的延迟或吞吐量。总的优化原则是分而治之，具体维度包括业务、组件边界、访问频率或对系统资源的消耗程度、瓶颈资源等。

具体内容如下。

1. **业务：** 把大应用按业务分成独立的互相合作的系统，如高层的采用 SOA 方式，低层的采用数据库分库方式。
2. **组件边界：** Web 服务器、应用服务器、数据库服务器、文件服务器。
3. **对系统资源的消耗程度：** 采用读写分离的方式。
4. **瓶颈资源：** 对数据库进行分表、分片。

一旦按上述维度处理好了，大家还可以在所有维度上应用负载均衡，把访问量分散到不同服务器。

□ 充分利用系统资源

采用多进程、多线程、异步操作以及负载均衡等手段，其中负载均衡主要做到了防止某台服务器过满和防止某台服务器过闲。

□ 减少不必要的计算次数

缓存计算结果，尤其是服务端缓存，以减少不必要的计算。

□ 减少不必要的 IO 次数

■ 网络 IO 次数：客户端缓存、CDN 缓存、合并资源以减少请求次数。

■ 磁盘 IO 次数：缓存常用数据，如利用 Redis、Memcached 进行缓存。

最后，作者总结指出缓存是减少不必要的计算和 IO 的重要手段，缓存的设计主要是根据资源变化频率对资源进行分类，比如动静分离等；其前提是恰当的状态管理、分离无状态的逻辑和有状态的逻辑，但会付出对一致性的一定妥协和运维的复杂为代价。缓存的适用场景包括热点不均衡、有效时间不太短、一致性牺牲程度可接受。作者还指出以上所有优化手段可以组合使用，有冲突时再做权衡。

作者还推荐了一些参阅文章：有关压力测试、负载测试的《[重述：性能、容量、负载以及压力测试](#)》和《[性能调优技术的几个角度](#)》。此外，有兴趣的读者还可以参阅酷壳陈皓发表的一篇题为《[性能调优攻略](#)》的文章。

React Native：Facebook 出品，可用 JavaScript 开发移动原生应用

作者 李小兵

近日，在 [React.js 2015 大会](#)上，Facebook 公布了即将开源的 React Native，它基于开源框架 [React.js](#)，并可用来开发 iOS 和 Android 原生应用。目前，Facebook 已经将 React Native 投入到了实际生产环境中，并开发出了基于 iOS 平台的聊天工具 Groups。

从 ProgVille 发布的一篇题为《[React Native—使用 React.js 开发原生应用](#)》的文章中得知以下 React Native 的相关信息。

1. React Native 已实现了对 iOS 和 Android 两大平台的支持。
2. 使用 React Native 开发原生应用的原理是：在 JavaScript 中用 React.js 抽象操作系统的原生 UI 组件，继而代替 DOM 元素来渲染，比如使用<View>取代<div>，使用<Image>替代等。在后台，React Native 运行在主线程之外，而在另一个专门的后台线程里运行 JavaScript 引擎，两个线程之间通过异步消息协议来通信（有个专门的插件）。
3. 在 UI 方面，React Native 提供了一个跨平台、类似 [Flexbox](#) 的布局系统，并且还支持 CSS 子集。
4. 可以用 [JSX](#)、JavaScript、[CoffeeScript](#) 和 [TypeScript](#) 来开发。

React/React Native 团队成员 [Jordan](#) 在 [Hack News](#) 上分享了 React Native 的[一些基于个人观点的信息](#)，他说到 React Native 为提高开发效率提供了大量的益处，但是在性能方面，React Native 还存在一些问题。同时还指出 React Native 同其他原生开发应用方式的不同之处，如 React Native 完全不用 DOM、React Native 既保证对应用程序性能的要求，同时兼顾 Web 开发优点；能够使用 JavaScript 来写高质量的应用等。

[Reddit](#) 上也有了[相关评论信息](#)，用户 [lunchmeat317](#) 认为学习 React.js 的时机到了。用户 [BishopAndWarlord](#) 表示对 React Native 很好奇并期待获得更多相关信息。

用户 [jrm2k6](#) 评论到：

自己已了解和喜欢他们的理念：一次学习，即可以做自己想着的任何事情。但是现在需要展示一些代码实例了，也许这是一个愚蠢的问题，但是它和 [Ionic](#) 有什么不同，同使用 AngularJS 开发 iOS/Android 应用有什么不同？

用户 [arx707](#) 接着回答到：

React Native 使用 React.js 作为原生组件的抽象层，而 [AngularJS](#) 和 Ionic 使用 WebViews 模拟本地组件，React Native 的性能应该和 [Appcelerator](#) 的[跨平台工具 Titanium](#)一样。

React Native 基于 React.js 实现，而 React.js 是 Facebook 推出并开源的一个用来构建用户界面的 JavaScript 库，其已经应用于构建 Instagram 网站及 Facebook 部分网站。React.js 同 AngularJS、MeteorJS 和 Polymer 类似，它们都属于 Model-Driven Views 结构的框架，但是 React.js 又与他们有不同之处，即 React.js 使用 JavaScript 而非 HTML 来构建用户界面。

专访阿里云 RDS 团队

WebScaleSQL 是一个怎么样的数据库

作者 郭蕾

2015 年 1 月 20 日, [Facebook 宣布](#)阿里巴巴旗下的阿里云 RDS 团队正式加入 WebScaleSQL。WebScaleSQL 是 Facebook、Google、Twitter 和 Linkedin 四家公司的 MySQL 团队发起的 MySQL 开源组织, 旨在改进 MySQL 在规模和性能等方面的问题。阿里云 RDS 团队有专门的源码小组负责 MySQL 源码级别的改进, 他们也经常活跃在 MySQL 社区中, 此次受邀加入 WebScaleSQL 组织也是对他们工作的肯定和认可。近日, InfoQ 编辑采访了 RDS 团队的负责人褚霸, 听他分享了整个邀请的背景以及接下来的工作重心。

InfoQ: 阿里巴巴受邀加入 WebScaleSQL, 与 Facebook、Google、Twitter 和 Linkedin 这样的世界顶级团队共同研发 WebScaleSQL, 这可以说是中国公司在国际开源项目上的一次亮剑。能介绍下整个受邀的背景和过程吗?

褚霸: 阿里云 RDS 团队有 MySQL 源码小组专门负责维护阿里云的 MySQL 分支。团队的主要工作是源码级别上的改进, 包括 Bug 修复、性能优化和定制化需求等。在我们改进过程中, 如果是 Bug 修复类型的补丁, 或者是足够通用的 feature 类型的补丁我们都会同时提交到上游, 包括 Oracle 官方、MariaDB 和 Percona 分支。阿里云 RDS 现在有大量的 MySQL 用户, 在系统维护和服务用户的过程中会碰到各式各样的问题, 我们在解决过程中也经常发起讨论, 在社区是一个活跃的团队。

WebScaleSQL 其他四家公司的参与者也是社区内的专家, 讨论技术问题时经常互动, 我们的工作大家也是比较了解的。因此在希望扩大 WebScaleSQL 的参与人员的时候, 会邀请我们是情理之中的。最初是 Facebook 的一位华裔员工先联系的我们, 然后我们也很快做了确认答复, 整个过程是比较顺畅的。

InfoQ: 能介绍下 WebScaleSQL 吗? 它有哪些吸引人的特性? 接下来阿里云 RDS 是否准备使用 WebScaleSQL?

褚霸：WebScaleSQL 是基于 MySQL 5.6 社区版本改编的 MySQL 通用分支，基于 GPL 开源协议发布。WebScaleSQL 目前已经做了很多性能改进工作，包括：客户端异步协调、逻辑预读、查询限流、服务端线程池优化、InnoDB 大页支持等等。由于我们的分支上本身有一些定制化的需求，因此不会直接使用 WebScaleSQL 分支提供线上服务，但是这些改进对于我们都是很感兴趣的，好的特性会被吸收进来。因为我们有各种各样应用场景的用户，对 MySQL 本身的要求也比较高。比如大并发连接的用户，就需要线程池；存大量历史数据的用户，就要求高的压缩比，等等。

WebScaleSQL 上的功能都是很“Web Scale”和接地气的。

比如线程池优化，大家都知道线程池是 Mariadb。WebScaleSQL 基于 Mariadb 的线程池实现进行重写并优化，对读写队列进行分离，重新设计队列优先级策略，避免了饿死现象。要知道线程饿死在有些场景下是很严重的。尤其是在并发连接数往往很大的互联网应用里面。

语句自动超时是一个很“WebScale”的特性。随着数据量的增大，同样的 SQL 的语句执行时间会越来越长。而语句执行期间占用的资源也可能越大。若不加以限制可能几个语句就可能拖垮一个服务，在互联网应用中这种场景更常见。WebScaleSQL 引入了来自 Twitter 工程师的代码，可以设定单语句的执行时间，超时则自动放弃。对数据库起到保护的作用。

GTID 是 Oracle 官方版本 5.6 引入的新概念，在解决主备切换，尤其是级联主备架构的切换方面提供了很大的便利。阿里云 RDS 的只读实例就直接利用了这个特性，其实现上还有可以优化的空间。系统需要维护一个全局的结构，在高并发更新场景下性能非常差。WebScaleSQL 优化了事务提交过程，减少了不必要的 GTID_OWNED SET 的维护。主库高并发更新的性能可以提升 20%。

InfoQ：阿里巴巴加入 WebScaleSQL 团队后，接下来具体会有哪些方面的工作？你们团队会有专职的人去开发 WebScaleSQL 吗？

褚霸：具体的工作其实已经展开，主要包括提交补丁和代码 review。我们团队都是专职的 MySQL 研发，但是不会专职开发 WebScaleSQL，实际上参与 WebScaleSQL 的维护工作都是在晚上和周末时间完成的。在开发和 review 阶段，使用的是 Facebook 内部的一个 review 平台，最终代码合并是在 GitHub 上。

这个是松散的组织，大家都可以主导一个功能。在邮件组里面沟通，之后在 review 平台交互。代码被 review 过后，由发起人提交到主干。团队内部主要的沟通方式还是邮件，也没有固定的 IM 沟通时间，成员之间需要点对点沟通的时候自己约。每个公司有 3~4 个人在里面，人力上不确定，其实是用的都是大家的私人时间。

InfoQ：自从 Oracle 收购 Sun 公司后，MySQL 社区开始走向分裂，出现了很多 MySQL 的分支，你认为像 WebScaleSQL 这样的分支会不会取代 MySQL？它们会影响 MySQL 的发展吗？你建议用户直接在生产环境使用这些分支吗？

褚霸：WebScaleSQL 的目标并不是取代官方的 MySQL，实际上它一直在跟随官方的 GA 版本在同步的升级。WebScaleSQL 本身也在促进 MySQL 的发展。由于这个分支的影响力，我们相信一些补丁的在官方的合并速度会加快。

WebScaleSQL 主干版本都是在官方 GA 版基础上，做了 Bug 修复、性能优化、新增功能等。当然一些细节需要注意，比如官方版本默认是编译了 PERFSHEMA，而 WebScaleSQL 从默认性能考虑，需要加上 -DWITH_PERFSHEMA_STORAGE_ENGINE 选项才会包含这个特性。要知道即使是官方版本，每个版本都有 Bug，因此不论使用哪个版本，都必须进行业务测试。新增的代码在合并入主干之前，大多已经长时间运行于各自的生产环境。总体上我认为新增的补丁是高质量的代码，可以用于生产环境。

InfoQ：之前和你也聊过开源方面的话题，新的一年我们看到微软、华为这样的公司都非常重视开源，开源是大势所趋。阿里巴巴此次加入 WebScaleSQL 团队，也是中国公司在开源方面得到认可的一个很好的例子，你认为一个公司如何才能做好开源？

褚霸：其实大家都是开源的受益者，首先就是要保持开源的初心：回馈社区。我们评估后觉得足够通用的补丁都会提交到 WebScaleSQL 或 Oracle 官方。还有坚持参与社区讨论、提交满足主干代码质量的补丁这样的工作其实很耗心力的，持续投入不容易，需要真正的兴趣驱动。

我们公司对团队在开源方面是很支持的。公司有淘蝌蚪这个开源平台，可以说阿里不止是开源的参与者，也是倡导者。我们团队内部对团队同学在社区的贡献也是给予很高的认可。建议和代码被社区接受对工程师个人来说也是很有意义的。仅 2014 年我们团队在社区提交的 Bug 和补丁被确认的就超过 40 个。活跃的个

人和团队更容易得到社区的认可，此次 WebScaleSQL 邀请我们参加维护也是如此。社区的认可也更促进个人的活跃，我们觉得这是良性的互动。

参与开源代码的维护，我们能够最快了解业内最新的动态，最大的收获还是在于我们可以学习全球顶级专家的经验，每个代码的方案设计过程、讨论过程、代码 review 过程都和结果一样，可以让我们从中学到很多东西，团队成长也会很快。

从商业角度探讨 API 设计

作者 Matt McLarty 译者 邵思华

为 Web 设计、实现和维护 API 不仅仅是一项挑战；对很多公司来说，这是一项势在必行的任务。[本系列](#) 将带领读者走过一段旅程，从为 API 确定业务用例到设计方法论，解决实现难题，并从长远的角度看待在 Web 上维护公共 API。沿途将会有对有影响力的人物的访谈，甚至还有 API 及相关主题的推荐阅读清单。

这篇 InfoQ 文章是 Web API [从开始到结束](#) 系列文章中的一篇。你可以在这里进行[订阅](#)，以便能在有新文章发布时收到通知。

如今，API 已经成为了每个重要信息技术趋势的核心内容。移动设计、云计算、物联网、大数据及社交网络等应用都依赖于一个基于 web 的界面与它们的分布式组件进行连接，为全球范围内的各个商业领域提供具有创新性和颠覆性的解决方法。智能电网（Smart grid）技术改变了能源行业的形态，联网汽车（Connected Car）解决方案则被视为汽车行业中的关键因素，亚马逊使得每个所接触的行业都产生了巨大变化。在所有这些例子中，API 的使用既是催化剂，也是促成这一成果的主要力量。

由于 API 对于商业的巨大影响，因此有关“API 的商机”的各种文章也是层出不穷。在开放性的互联网上，使用 API 作为一种外部频道进行创新及盈利已经成为一种独特的商业模式。在由 [Kin Lane](#) 所创建的 [API 传道](#)（Evangelist）网站中可以找到关于这一话题非常全面的信息，[Mehdi Medjaoui](#) 则在[最近的一篇帖子](#)中用精练的语言对此进行了总结。然后，在跨科技领域的 API 应用范围内，开放式 API 模型仅仅表现出其实用性的冰山一角。实际上，Web API 的主要能力还没有从各种使用 API 实现的解决方案中被发掘出来。从这种意义上说，API 的商机本身就是一种商业模式。

本文将从商业角度对 API 进行全面的讲解分析，无论它是否是开放式并且公开发布的。我会谈到尝试用 API 为你带来商业价值的重要性、分析在其中应该使用的数据类型、并学习 Amazon 及 Twilio 的成功经验。希望这些内容能够有助于你打造有用的、并且可用的 API。

评估 API 的商业价值

API 的通用商业价值是可以进行评估的。一切从数据出发，许多公司及组织将他们的数据视为一种负担，毕竟服务器和存储方案的价格不菲。但在如今这个越来越趋向于电子化的世界中，很显然，数据也是一种宝贵的资产。数据提供了各种宝贵的客户资料，它能够产生可辨别的商机与新的收益方式。“大数据”狂潮正是追求通过海量数据的分析处理电子中的混乱信息。即将到来的物联网（IoT）爆炸将使数据的规模呈现指数级的增长，因此对各个公司来说，对于数据进行正确的分析就变得至关重要。

对于一家公司来说，数据到底是一种资产，还是一种负担，是取决于以下三个方面的：即数据的可访问性、准确性和可应用性。每个 Web API 都在某种程度上提供了某些数据的可用性，而有价值的 API 则为公司的核心商业数据提供准确的数据。这使得公司能够达到一种我称之为“Data-Enabled Disruption”的迭代发展模式，在下文中我会为这种模式做出解释。此外，在决定应该由 API 暴露哪些数据及服务时，以及如何实现这些 API 时，这三个方面的数据属性也提供了一种有效的方法论。

数据可应用性	<input type="checkbox"/> 这些数据是否有助于我的商业目标决策 <input type="checkbox"/> 这些数据是否能够为我的业务带来独特的价值 <input type="checkbox"/> 如果我将这些数据公开化，是否能产生某些商机
数据准确性	<input type="checkbox"/> 当前提供的数据时效性如何 <input type="checkbox"/> 数据的来源是否可靠 <input type="checkbox"/> 数据是否由期望中的用户所使用？是否用于正确的目的
数据可访问性	<input type="checkbox"/> 哪些数据是可以由编程方式获取的 <input type="checkbox"/> 有哪些不同的方法可以获取这些数据 <input type="checkbox"/> 开发者创建使用这些数据的应用难度有多大 <input type="checkbox"/> 数据访问的规模能否满足客户的需求

如果从 API 的角度对这套方法论进行验证，那么可以将数据的这三种属性合并为 API 的两种属性：

1. “实用的 API”提供准确与合适的数据；
2. “可用的 API”提供可访问的数据。

显然，最有价值的 API 应当满足实用与可用两个条件。不过，为了更进一步定义这些 API 属性，让我们分别来进行一下分析。

实用的 API

在开发 API 时，人们最常见的一种错误就是认为所有的数据都是有用的。有一种流传甚广的奇谈是这么说的：一旦你拿出这些数据，神奇的开发者们就会出现在你面前，他们会撒下一些具有魔力的粉末，让你的收益得到增长、涌现各种创新的想法、并打通各种商业渠道。但仅仅使用 API 和开放数据是不足实现这几点的。正是这种“媒体即讯息”的想法造成了过去十多年间在企业整合这一领域中出现了大量失败的 SOA 尝试。某家超大型企业曾经花费了 5 千万美元以上的资金企图打造一个 SOA 及私有云的项目。而当我问及他们打算为哪些客户提供什么样的服务时，他们立刻就哑口无言了，因为他们只关心如何打造基础设施。毫无疑问，这个项目最终失败了。

从好的方面来说，如果能够使用正确的 API 公开正确的数据，那么就能够实现收益的增长与创新的进步。Google Maps 利用 Google 压倒性的占有率为提供的基于 API 的服务，正好填补了市场在这方面的空白。由于这一服务相当便利，因此 Google 可以将它作为商业产品收取大量费用。由于 API 的存在，Google Maps 也成为了早期 iOS 平台上的常驻应用，而苹果自己推出的替代品在一开始的反响就相当差，这反而突显了 Google Maps 的价值。由 Facebook 与 Twitter 领衔的社交网络平台的发展与成功也离不开 API 的应用，正是后者促进了他们的 web 链接数目与移动平台上的应用实现。实用的 API 甚至[对联邦竞选活动产生了深远的影响](#)。

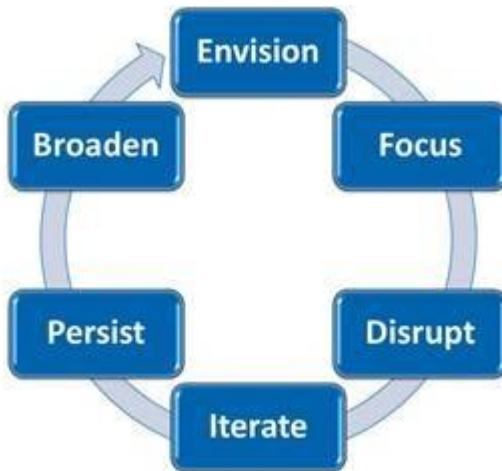
Amazon 的 API 故事

正如我在文章开头所说的一样，企业从 API 中获得商业成功的潜力远大于开放 API 在表面上的能力。Amazon 就充分利用了这种潜力，其实它们的 API 最初只是在公司内部进行使用。API 为 Amazon 所带来的这种长远的成功在任何一个行业中都找不出相似的案例。Amazon 为使用其 API 的客户提供了宝贵的经验，帮助这些企业使用它们的 API 获得商业上的成功。

Amazon 为我们所上的第一堂课，也是最为明显的一堂课，就是它是怎样将 API 设计为产品与解决方案的构造块的。Brad Stone 在由他编写的[书籍中专门用了一章](#)的篇幅来描述 Amazon 是怎样将基础 API 发展为它的技术航母的。Kin Lane 也对 Jeff Bezos (Amazon 的 CEO) 是怎样从固执的陈旧观念中慢慢转变，让 Amazon 最终提供编程式访问能力的过程进行了[精彩的总结](#)。按照[这些报告及一些其它资料](#)所说，产品经理必须指出他们的提案中的商业价值的最小公分母。随

后技术团队就会利用这些原始数据创建 API，将这些商业价值提供给其他开发者，让他们继续创建整个方案中剩余的部分。背后的逻辑在于这些商业价值的增长能够直接使用，并且能够非常容易地进行结合，以进行未来的产品开发。这就是为什么 Amazon Web Services 能够如此迅速地发布及改善：Amazon 对它们的基础设施服务都进行了 API 化，从而优化了扩展基础设施能力的过程。AWS 的产生过程就是将这些内部功能转化为外部产品。正如你所见，Amazon 不仅仅保证了他们所提供的 API 提供了有用的数据，而是走得更远，他们将这一原则进行了倒转，保证了整个解决方案中的每一份数据都能够由 API 方式进行提供。

Amazon 在 API 方面的另一个例子是它如何使用基于 API 的方式进行有价值数据的收集、分析、改善以及分布的。当早期的 Amazon 还以在线书籍的销售为主营业务时，Jeff Bezos 对于 Amazon 的核心价值观观念就有了清晰的预期：“[我们不是通过销售盈利，而是通过帮助客户进行采购决策而盈利。](#)”他始终保持公司的运营与这一核心价值观相一致，从而产生了策略方向的各种改变，例如提供个性化及扩展频道等举措。实际上，从以上核心价值观中就可以看到，正是这些适用的、准确的、并且可访问的数据为客户的决策作出了引导，让 Amazon 一步步走向成功，而不是靠着在线图书或其它商品的销售获得的成功。在 Amazon，正是 API 将数据进行不断积累与改进，这一周期性的过程刺激了 Amazon 的成长。我将这个 360 度的周期称之为数据革新“Data-Enabled Disruption（DED）”，下图是我对它的总结。



DED 的成功运用，使得 Amazon 得以击败无数竞争对手。由于在整个数据生命周期的每一步都应用了 API，因此 Amazon 能够持续地改进数据的准确性、适用性及可访问性。

我们最后将学习 Amazon 是怎样在公司的战术产出和策略定位之间进行平衡的。自从 Jeff Bezos 认识到万维网的潜力，并制定出“提供所有商品的商店”这一愿景之后，他就非常清楚地认识到这种平衡的重要性。Bezos 认识到他先要从一个较小的目标开始，经过对市场进行一番分析之后，他认准了在线图片销售这一领域，它的发展时机已经成熟，并且供应链也十分理想。在随后的发展中一方面保证快速的执行，一方面始终不忘对未来的愿景设计，这种齐头并进的发展理念已经成为了 Amazon 文化的一条根深蒂固的宗旨。每个解决方案既要为公司产生价值，也要为未来的发展铺好基石。基于 API 的交付方式就是实现这种原则的一种理想的方式，因为 API 不仅能够服务于新的应用与服务，也能够为未来的各种用例打好基础。这种迭代式发展的方法论促进了亚马逊业务的不断发展（见下图的说明）。



[图片中的每个服务都对应着一套外部的 API](#)，同时，每套 API 都是建立于已经完成的 API 基础之上的。任何一家打算纵向或横向进行业务扩展的公司，都可以认真参考一下 Amazon 是如何打造一套有用的 API 的方法：持续地收集和获取可用的数据、使用 API 作为这些数据的通用访问点、只交付短期内有用的数据，同时兼顾长期的发展计划，以此作为公司的竞争力进行不断扩展。

可用的 API 及 API 设计的重要性

尽管 Amazon 在这方面取得了令人瞩目的成就，并且 API 在公司的成功中扮演着重要的角色，但 Amazon 的 API 仍然没有被公认为设计最优秀、使用最简单的 API。随着 [API 数量的爆炸性增长](#)，并且对于 API 的[必要性的认可度也在不断增加](#)，API 的可用性正是让那些在行业中处于支配地位的公司，甚至是那些仅仅打算用 API 建立创新性服务的创业公司能够获得成功的关键因素。

移动设备的出现及 IT 的消费化趋势，是对于传统的企业级应用开发的一次全面转变。在过去，普遍存在着各种功能单一的终端主机、客户-服务器系统、以及最近出现的 web 的分布式布局。我过去也曾谈及这种我称之为“层的脱落”的现象，即将业务从 n 层的 web 模型转向以 API 为中心、以移动和云为优先的设计。这种转变也包括了代码开发从 Java 企业版转到 JavaScript 及其衍生语言的情况。所有以上这些都表明，正有一波新的开发者，他们将逐渐成为开发新企业级解决方案的主力。这些开发者们习惯于主动寻找有什么 API 可以满足他们所需的功能。当前的公司应当预计到这种转变的发生，并迎合新一代开发者的需求，方式就是拥抱 API 的可用性。

让我们看一下电信业的情况。多年以来，各大电信巨鳄们都在处心积虑地想要打倒竞争者，同时也在积极地推出各种跨网络的增值服务。这个行业在过去的 15 年间产生了巨大的革新，包括 VOIP 的出现、业务及运维服务的整合，以及移动设备服务的革命。在种种革新的进程中，API 都扮演了重要的角色。即使在传统电信服务方面仍占据领先地位，但这些电信巨鳄们也难以从这波革新浪潮中受益。而当他们试图与 Parlay X 及 OneAPI 等创业公司进行合作时，他们遇到的困难比这些小公司更多，Alan Quayle 在他的一篇文章中就总结了这一现象。如果这些大佬们都难以抓住这次机遇，又有谁能做得到呢？

创建于 2007 年的 Twilio，其发展目标就是为客户提供易于使用的语音及文字消息服务，并且完全在云端进行托管。他们一开始就计划打造这样一个平台，并且意识到 API 会成为他们第一位的业务方向。SMS 和 VOIP 服务固然很有用，但为了与电信巨鳄们展开竞争，他们所需的不仅仅是一些便利的电话服务而已。

Twilio 最关键的洞察力在于：他们已认识到所提供的服务的第一批客户并非那些调用 API 的应用的终端用户，而是那些负责开发这些应用的开发者本人。他们也知道，移动端应用的增长速度必然是最高的。因此，他们定制了一套指标，用以衡量这批客户对 API 的满意程度。除了传统的终端用户统计数据，例如端到端的 API 调用响应时间之外，他们还额外对新开发者注册这套 API 所需的时间进行衡量，并且设定了很高的目标。这套指标的设定改善了 API 的可用性，从而为 Twilio 建立了对于各大电信巨鳄的领先优势。当应用开发者们在为应用的开发选择 SMS 或 VOIP 提供者时，Twilio 的这套响应迅速的轻量级服务就明显比起竞争们胜出一筹。有了这套实用的 API 之后，Twilio 就可以理直气壮地对服务收取费用，通过客户的每次 API 调用的付费实现盈利。也正是因为这套实用的 API，Twilio 提高了公司的名气，同时也增长了公司的利润。

电信之外的各个行业也对数据革新的方向准备就绪了。就拿 Ingenie 来说，这家保险业的创业公司在基于精算的定价方法方面推陈出新，对于 16-25 岁这一阶段的年青人会进行适当的惩罚。他们通过在每台汽车里安装的一种专利智能设备对每个驾驶员的数据进行收集，随后依据这些数据为这些驾驶员们提供相应的保险折扣。实用的、可用的 API 使得 Ingenie 能够实现数据带来的革新，让他们像 [Twilio](#) 一样征服了整个保险行业。

实用的、可用的 API 指南

在此进行一下总结，要确保 API 的成功，可以通过以下几个步骤来实现。

- 确保你的 API 与公司的策略相一致
- 在 API 中包含可访问的、准确的并且适用的数据
- 确保你的 API 是实用的，并且可用的
- 学习 Amazon 的方式，建立一种规范的文化，迭代式地实现数据带来的革新
- 学习 Twilio 的方式，创建一种优秀 API 开发者体验，从而使你的业务更胜于其它各大竞争者。

只要按照这套指南的方法，你的 API 终将为你的业务带来巨大成功，并成为这方面的典范。只有你能够最好地判断怎样为客户提供实用的 API。此外，也请各位拜读一下本系列中的其它各篇文章，它们会为你实现实用的 API 提供许多宝贵的建议。

关于作者

Matt McLarty(@mattmclartybc) 是 CA Technologies 公司的 API Academy 部门的副总裁。该部门为各公司提供 API 策略、架构及设计方面的专家指导，以帮助他们在电子经济时代茁壮成长。

为 Web 设计、实现和维护 API 不仅仅是一项挑战；对很多公司来说，这是一项势在必行的任务。[本系列](#) 将带领读者走过一段旅程，从为 API 确定业务用例到设计方法论，解决实现难题，并从长远的角度看待在 Web 上维护公共 API。沿途将会有对有影响力的人物的访谈，甚至还有 API 及相关主题的推荐阅读清单。

查看英文原文：[Article: A Business Perspective on APIs](#)



《云生态专刊》创刊号

创刊序

2015年是中国云计算产业的生态之年，“单打独斗”的模式已经没有出路，越来越多的云服务厂商认识到构建生态系统的重要性，从提供创业孵化环境到通过共享、共赢的理念打造一个自上而下的健康生态链，参与者都有“肉”吃，客户也更加满意，这样的场景令人向往。

这也是国内IT产业成熟的一个标志。中国的云计算市场开始进入加速发展的阶段，在未来5年，市场增长率预计都在30%以上，云厂商及相关合作伙伴将超过千家，云领域相关的IT从业人员将达数万人。公有云目前处于低总量、高增长的时期，IaaS规模在迅速扩大，竞争也最激烈；PaaS依然在培育期，发展前景看好；SaaS最成熟，盈利状况最好；私有云、混合云则处于闷声发财的阶段。InfoQ有幸参与并见证了中国IT产业的高速发展。技术变革是推动产业发展的主要力量，我们尝试站在技术浪潮的前沿，让国内的架构师、开发者了解和借鉴整个技术社区的成果。《云生态专刊》是InfoQ为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”，包含的内容包括：

1. 引入国外云计算领域的先进思想和技术；
2. 报道国内外云厂商和生态圈的发展动态；
3. 分享深度和前沿的热点技术 关注产业发展的难点、痛点；
4. 分享解决方案 挖掘产业发展的亮点，分享宝贵经验。

我们期望通过这种形式推动国内云计算产业的发展，让生态圈良性循环，也希望大家对《云生态专刊》的发展多提建议和批评，让我们走的更踏实一些。感谢时代给我们机遇，我们只能奋勇向前，与大家携手进步。

目录

创刊序

嘉宾寄语

热点回顾

对话大咖

探究 AWS 开发者生态最佳实践

技术热点

从一个 OpenStack 的失败案例看 Ironic 和 Neutron 组件的现状

成功部署 OpenStack 的十个小技巧

Docker 专栏

etcd：从应用场景到实现原理的全方位解读

观点&趋势

BaaS 服务的定义、发展以及未来

谷歌关于容器的观点分享

生态圈新闻

[InfoQ 网站免费下载](#)

——InfoQ 中国总编辑 崔康

QClub

Docker专场——聚焦国内Docker创业以及企业实践

Docker显然已经成为2014年最火的开源项目之一，它受到越来越多的开发和运维人员的亲睐。本期QClub聚焦国内Docker创业以及最佳实践案例，以及Docker创业公司将如何利用这一支点来撬动新的市场。



docker

时间：2015年3月29日（周日）9:00 ~ 17:30

地址：北京市海淀区知春路25号 丽亭华苑酒店

限量免费点击报名

春天花会开——迎春花



迎春花, 拉丁文名: *Jasminum nudiflorum* Lindl。木犀科、素馨属落叶灌木, 直立或匍匐, 高 0.3-5 米, 枝条下垂。枝稍扭曲, 光滑无毛, 小枝四棱形, 棱上多少具狭翼。叶对生, 三出复叶, 小枝基部常具单叶; 长圆形或椭圆形, 先端锐尖或圆钝。花期 6 月。迎春花与梅花、水仙和山茶花统称为“雪中四友”, 是中国常见的花卉之一。迎春花不仅花色端庄秀丽, 气质非凡, 具有不畏寒威, 不择风土, 适应性强的特点, 历来为人们所喜爱。迎春花栽培历史 1000 余年, 唐代白居易诗《代迎春花召刘郎中》以及宋代韩琦《中书东厅迎春》和明代周文华撰《汝南圃史》均有记载, 迎春花现在为河南省鹤壁市的市花。

在早春时节, 大地万物刚刚复苏, 到处仍然是一片荒草枯木的时候, 若在你眼前突然出现一簇迎春花, 定会使你欢欣惊喜, 你会情不自禁地到它跟前去仔细观赏一番。

促进软件开发领域知识与创新的传播

架构师

ARCHITECT

解读2014 | Review
解读2014之前端篇

推荐文章 | Article
阿里云RDS团队专访
从商业角度探讨API设计

专题 | Topic
GitHub中签开发者年度分析报告
2014开源软件发展调查报告发布
GitHub开源大牛谈中国开源

InfoQ

2015年03月

架构师 3月刊

每月 8 号出版

本期主编：崔康

流程编辑：丁晓昀

发行人：霍泰稳

读者反馈/投稿：editors@cn.infoq.com

微博、微信：[@infoqchina](#)

商务合作：sales@cn.infoq.com

本期主编



崔康 InfoQ 中国总编辑，致力于中国 IT 领域知识与创新的传播，目前负责 InfoQ 整体内容的品牌和质量，同时担任 QCon、ArchSummit 大会的总策划。技术人出身，毕业于天津大学计算机学院，在加入 InfoQ 之前，在某大型外企长期担任协作软件平台的技术负责人和架构师，在相关技术领域积累了一定的经验。从 2008 年开始参与国内社区的技术传播，在多家科技媒体先后发表过数十篇文章，并出版多本译著，总产量超过 50 万字。可以通过 tyler.cui@infoq.com 或者微博（“崔康 Tyler”）与他联系。



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 |