促进软件开发领域知识与创新的传播

架构师

9月 ARCHITECT

特别专题

可测试的架构

分布式体统测试在阿里云实践

"窥探"谷歌测试

如何控制单元测试的粒度

代码测试比率、测试&行文驱动开发

Somya Jain谈从C#到Object-C

Restful Objects简介

C++ 并发编程的一种思维

优秀的JavaScript模块是怎样炼成的

在MongoDB中实现聚合函数

InfoQueue

Enterprise Software Development Community

每月8号出版

卷首语 | Editor’s Letter

软件测试 路向何方

在 The Testing Planet 发行的期刊中，提到了测试未来的 3 个发展方向：在生产环境中测试（TiP，Testing in Production）、TestOps 以及基于云的测试。

TiP 通过暴露新代码给有限用户，减少缺陷可能带来的负面影响，通过在产品中暴露这些新代码，可以快速获得这些新代码的反馈，这些反馈来之于真实的用户，而不是少量的测试人员和有限的测试用例。另外，一旦发现新代码有严重的缺陷，那么 TiP 需要快速修复这些缺陷，通过发布新版本或则滚回到老版本。

TestOps 则在某种程度上改变了现在的测试人员与开发人员和运维人员的角色划分，开发人员将负责编写单元测试，并对功能测试的质量负责，如此，测试人员则将工作重点放在 TiP 上，通过基于面向测试的架构，测试人员更加专注 TiP 工具的开发，数据信号的监测以及数据的分析，这种行为为测试人员增加了开发特征，同时，这也使得测试人员肩负了一定的运营工作。随着更多的应用和服务步入云端，基于云的测试以及更深层次测试计划将成为可能。本次的专栏文章，既有经验分享的“窥探”谷歌测试，又有在云计算浪潮下的测试实践——阿里云的分布式体统测试，同时，对于关注技术的读者，还将了解到如何控制单元测试的粒度，专家畅谈关于代码测试比率、测试驱动开发及行为驱动开发等话题。后续，InfoQ 会联合几大社区，打造测试专题，从技术趋势到技术实践——展开，也会与国外社区合作，回馈最新的内容与读者，欢迎大家关注。

本期主编： 贾国清

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

目录

[卷首语]

软件测试 路向何方 2

[人物专访]

采访：视频 CDN 分发、调度与服务 7

[热点新闻]

《阅读者》第 20 期：架构模式的圣经 12

Facebook 元老王淮谈科技公司应有的工具文化 18

快速哈希算法破坏了加密的安全性 24

HTML5 VS. Flash&Flex? – 浅谈 Flash/Flex/HTML 5 技术 27

产品经理需要懂技术吗？ 35

[特别专题]

分布式系统测试在阿里云的实践 41

“窥探”谷歌测试 46

如何控制单元测试的粒度？ 50

虚拟座谈会：代码测试比率、测试驱动开发及行为驱动开发 54

采访和书评：Google 如何做测试 73

[推荐文章]

Somya Jain 谈从 C#到 Object-C 82

Restful Objects 简介 87

C++ 并发编程的一种思维.....104

优秀的 JavaScript 模块是怎样炼成的.....118

在 MongoDB 中实现聚合函数.....137

[新品推荐]

Blend for Visual Studio 2012.....159

500TB——Facebook 每天收集的数据量.....159

F# 3.0 新特性简介.....159

使用 Kentico CMS 7 轻松构建 ASP.NET 站点.....160

Napa 简介——基于 Web 用于 Outlook/SharePoint 的 IDE.....160

PhoneGap 2.0 发布.....160

Visual Studio 2012 和.NET 4.5 已经就绪！.....161

亲密接触 IIS 8 和 Web Deploy 3.0.....161

PowerDrill，Google 又一个大数据分析大杀器.....161

Terracotta BigMemory 3.7：TB 级支持、改进搜索、增强安全性.....162

[推荐编辑].....164

[封面植物].....165

翠柏.....165

勘误.....167

人物专访 | Interview

采访：视频 CDN 分发、调度与服务

作者 侯伯薇

吴岷是土豆网基础系统开发总监，在土豆长期负责视频上传、转码、CDN、点播、直播等系统的开发和维护。在全球架构师峰会期间，我们有幸采访了他，针对视频 CDN 分发、调度与服务方面的内容进行了交流。

InfoQ：与传统的网站相比，视频网站的架构需要应对哪些特定的非功能性需求，土豆网是如何应对的？

吴岷：用户观看视频需要一个高速且稳定的下载带宽，所以视频 CDN 节点要离用户更近。CDN 节点离用户近就意味着 CDN 节点会很多且相对比较小。另外从成本考虑，在二三级城市购买带宽单价往往会便宜些。小节点多就会引发很多问题，比如节点质量不稳定，机房不能提供高可靠性的服务，机房搬迁比较平凡，等等。所以视频 CDN 需要有足够的灵活性来应对这些变化。

土豆网的视频 CDN 的特点是视频文件纯分布式的，它不存在核心节点，并且所有视频文件在位置中心服务器都知道，这样做的好处是机房只要能服务好当地用户，不必考虑与核心节点间的链路质量；另外由于中心机房知道所有文件位置，所以硬件故障和机房搬迁都能非常容易的处理，只要在数据库中把相应的硬件或者机房置为不可用即可。

InfoQ：视频 CDN 需要关注哪些方面的问题？

吴岷：视频 CDN 需要关注用户服务质量，这和一般网站 CDN 是一样的。比较特殊的是视频 CDN 特别要关注成本。视频 CDN 在公司运营成本中占的比例相当大。为了节省成本，商务可能会随时把机房从贵的地方，搬迁到便宜的地方，视频 CDN 要有适应这种变化的能力。另外，从技术方面，降低单位服务成本永远是我们努力的方向。

InfoQ：您认为，在视频同步方面，架构的使命是什么，应该如何实现，存在哪些问题呢？

吴岷：整个视频 CDN 的使命是以较低的单位服务成本给用户较好的服务体验。当然架构在视频同步方面的使命也是一样。视频同步的目的是尽可能把视频放到离用户近的节点去，以提高用户的体验，但是视频同步也是要消耗带宽的，如何平衡这两者关系，找到一个最佳的平衡点，是视频同步一直要考虑的问题。

InfoQ：请您谈一下，视频同步的 Push 和 Pull 两种模式，各自的优缺点有哪些呢？

吴岷：pull 模式是 CDN 的经典模式比较成熟，直接用到视频上也比较容易。Push 模式需要实现很多东西。两者优缺点很明显，纯粹的 pull 模式实现比较简单，灵活差点；push 模式更灵活，但是非常复杂。也可以结合两者有点，采用混合的模型。

InfoQ：土豆在视频同步方面采用了哪些技术，效果如何？

吴岷：土豆的视频同步从策略上沿用了 cache 的思路，把最热的文件复制到边缘节点，淘汰最冷门的文件。在具体实现上我们做了几点优化，一个是采用类似 p2p 的技术， 可以同时从多个节点下载文件，当然这些节点也是离目标节点最近的。另外我们的 Web 服务器有下载文件的功能，这样做的好处是磁盘的读写都由一个进程调度，实现了磁盘吞吐最大化。效果不错，一般一个几十兆的新视频文件只需要几分钟就能同步到全网了。

InfoQ：您认为在访问调度方面，视频 CDN 有什么样的特点？

吴岷：土豆的视频 CDN 节点比较多，且每个比较小。小节点比较容易跑满，所以负载均衡要做得非常实时，否则用户观看视频就容易卡。另外我们视频 CDN 的服务器都没有做 raid，磁盘很容易损坏，服务器有时候也会当机，调度如果能够及时感知到服务器的健康状况的话就很容易屏蔽掉这些失效的硬件，用户就感知不到了。

InfoQ：土豆网采用了什么样的技术实现访问调度，有哪些优缺点，是否满足了需求？

吴岷：土豆的视频 CDN 的状态实时汇报到调度系统在中心服务器，加上中心服务器知道每个文件所在的位置，这相当于所有情报尽在掌握了，所以土豆的视频调度从策略上来讲相对不难做好。缺点时实现起来复杂，计算量大。因为复杂，所有总会碰到各种各样的问题，但是经过这几年的积累，现在已经可以满足需求了。

InfoQ：在 Web 服务方面，视频网站经常会出现的问题有哪些？

吴岷：多样性方面。互联网视频这几年发展的比较快，codec 从 h263 发展到 h264，容器有 flv, 3gp, mp4, ts 等，客户端从 flash player 到现在 ios，安卓等等。技术要随时跟上。在系统稳定和兼容中找到平衡。另外，和放图片的 CDN 一样，磁盘读写依然是 web 服务的瓶颈。

InfoQ：土豆网是如何解决 Web 服务方面的问题的，采用了哪些技术？

吴岷：土豆网视频 CDN 中使用的 Web 服务器是我们自己研发的。当时我们分析了用通用 http 服务器的实现原理，并针对视频的特点加以改善。

一般 http 服务器使用 sendfile api 把文件发到 socket 上。对于小文件，比如图片，一般一次读操作就能读取整个文件，这时由于不需要多余的内存复制，所以 sendfile 的效率很高。但是视频文件比较大，几乎不可能一次从磁盘上把文件读出来，要同时服务很多用户，这时哪个文件读第二次就是一个很大的问题。当时我们主要的改进是舍弃了调用 sendfile，

改成用 read 读取磁盘，然后磁盘调度由应用程序实现，而不是交给操作系统，所以我们能把磁盘调度做到最优化。最近我们把视频同步也集成到我们的 Web 服务器中，这样，不光硬盘的读，连硬盘的写都由一个程序来负责调度，大大提高了磁盘的读写吞吐。另外，因为整个 Web 服务器都是自己开发的，所以在实现比如支持 HLS 等新功能时就比较方便了。

InfoQ：请您从总体上说明一下土豆网的视频 CDN 架构的特点。

吴岷：总的来讲，土豆视频 CDN 架构最显著的特点就是采用 push 模式实现文件同步。中心控制节点知道所有信息，所有行为都由中心控制节点发出。这样做从策略上非常灵活，几乎没有策略上不能实现的需求。但是从实现上讲就比较复杂。对于中心节点服务器可扩展性、稳定性、计算能力等都有相当的考验。

原文链接：http://www.infoq.com/cn/news/2012/08/cdn-tudou

相关内容：

- 案例分析：基于消息的分布式架构
- 淘宝开源分布式消息中间件 Metamorphosis
- Twitter Storm：开源实时 Hadoop
- 使用绑定实现灵活通信

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

《阅读者》第 20 期：架构模式的圣经

作者 张逸

提及模式，开发人员的第一反应一定是 GOF 的《设计模式》。毫无疑问，这本软件领域的经典著作已经深入人心，差不多可以说是设计模式的圣经了。书中的 23 种模式已经成为开发者之间进行交流的术语，使用它们甚至像使用语言中的惯用法一般自然。然而，事实上，在模式领域里，还有一部伟大著作给予软件设计领域带来的影响同样的大，那就是以德国人 Frank Buschmann 为主要贡献者的《面向模式的软件架构》（Pattern-Oriented Software Architecture)系列。相对于《设计模式》而言，我以为这一系列可以称之为是架构模式的圣经。

这一套煌煌巨著一共分为五卷，其中第一卷《面向模式的软件架构：模式系统》（Pattern-Oriented Software Architecture Volume 1：A System of Patterns）的出版日期为 1996 年 8 月，最后一卷《面向模式的软件架构：模式语言》（Pattern-Oriented Software Architecture Volume 5：On Patterns and Pattern Language）则出版于 2007 年 5 月，也即是说整个系统从创作到出版跨度长达十余年的时间，单以这一点来说，本书的作者们，尤其是主要贡献者 Frank Buschmann 的毅力值得钦佩。

与《设计模式》不同，本系列主要的关注点放在了架构层面。根据 Richard Helm（GOF 之一）对设计的划分，这些内容应属于宏观架构（macro-architecture）的部分，可以作为处于微观架构（micro-architecture）中的《设计模式》的有力补充。正因为此，阅读

这一系列须得有足够的架构经验或基础，才能有所收获，而如此多卷的大部头著作，也是许多人望而却步的主要原因。

为了对架构以及架构模式有更深入的理解，我曾硬着头皮、下定决心、不辞万难地啃完了这个系统的三卷，分别为第 1、3、4 卷。在我未来的阅读计划中，本系列的第二卷是我必须要阅读的。关于第五卷《模式语言》，窃以为更多地是偏向理论介绍，大可以忽略不计。因此，以我的阅读经验来看，自认为有几分资格来谈谈我的阅读体验。

这个系列虽然都是讲解架构模式，但各有其特色，关注点也有不同。整体说来，第一卷《模式系统》介绍的内容更为宽泛而通用，书中讲解的各种架构模式也已得到了足够广泛的普及，例如大家耳熟能详的 Layer 、MVC 以及 Pipes And Filters 模式，都在本书进行了深入的讨论。当然，准确说来，本书涵盖的内容其实涉及到模式的各个方面。作者将模式分为三个层次，分别为架构模式、设计模式与惯用法（Idioms）。前面提及的模式就属于架构模式的类别，除了这三种模式之外，Broker 模式、Presentation-Abstraction-Control 模式以及 Microkenel 模式，对于我们进行软件架构活动而言，都有非常重要的参考价值。至于书中的设计模式部分，可以看做是《设计模式》的延续，有的模式表达的思想本质与 GOF 的《设计模式》完全一致，只是在命名上略有不同。例如 Whole-Part 模式与 GOF 的 Composite 模式，Publisher-Subscriber 模式与 GOF 的 Observer 模式都有着千丝万缕的关系。书中还提到了《设计模式》所未曾涵盖的内容，如 Forwarder-Receiver 模式与 Master-Slave 模式，以及 Client-Dispatcher-Server 模式，都非常有用，尤其可以运用在并发场景或分布式场景中，这正是 GOF《设计模式》较少提及的内容。

我虽然未曾阅读本系列的第二卷，但仍然粗略的翻看过，其中讲到的诸如 Reactor 模式、Active Object 模式与 Leader-Follower 模式等，都曾经有过深入了解并在项目中运用过。该书的主题是用于并发和网络化对象的模式，这对于真实的大型应用系统而言非常重要，

却又是我们在系统设计中最常面对的难题。本书的介绍与讲解方式完全延续了第一卷的风格，既有理论阐述，又有案例详解，同时又保持了一贯严谨的学术风格。

本系列的第三卷是唯一没有 Buschmann 参与的一本，也是最薄的一本，但书中讲解的概念却同样关键。该书主要介绍了与资源管理相关的模式，例如 Lookup 模式、Lazy Acquisition 模式与 Caching 模式，以及 Pooling 模式等。关于资源管理的相关问题，事实上在企业应用系统中相当普遍，无论是硬件资源、网络资源还是其他外部资源，从性能、可重用性与可扩展性角度来看，都是值得关注的。这其中很多模式也得到了广泛地运用。例如 Java 中的 JNDI 就采用了类似 Lookup 模式的设计，Lazy Acquisition 模式也在诸多框架例如 Hibernate 与 Spring 等得到运用。Leasing 模式在 WCF 或早期的.NET Remoting 中也得到了有效运用。

第四卷虽然名为《分布式计算的模式语言》，但事实上可以看作是一本模式目录，因为本书涉及到的模式一共包含了 114 种模式，这其中还没有计算参考到的另外 150 多种模式。形象地说，本书好像是模式大观园，其中的风景可以说是美不胜收。然而，正是因为美丽的景点实在太多，就很容易导致风景的欣赏者开始走马观花，最后充斥在脑海中的不过是肤浅的一堆表面印象而已。粗略看来，由于要介绍的模式太多，而书的容量有限，导致许多模式的介绍只能浅尝辄止。我不否认这确乎是事实，但另一方面，本书对模式的这些描述言简意赅，背后包含的思想与意义，却又值得我们反复思索。例如，本书将 Domain Object 模式描述为：“将自我完备的连贯功能和基础性责任封装成定义良好的实体，通过一个或多个 Explicit Interface 提供功能，并隐藏内部结构和实现。”在这个描述中，所谓的“自我完备”一词就值得我们反复玩味，若能充分地体会到封装的意义与面向对象的本质，当阅读到这个词语时，定能会心微笑，犹如遇见知音一般的快乐。

面对这样一套浩如烟海的著作，我们该怎样阅读，才能攻克架构模式这座堡垒呢？在阅读到第 4 卷时，通过回应前面几卷的内容，给了我一个很好的提示。我认为，最好的阅读方法就是《如何阅读一本书》中提到的主题阅读（Syntopical Reading）方式，即通过确定主题相关性，定位到相关章节，广泛查阅资料，汲取与主题相关的知识营养，进而深入地分析这个主题的方方面面。本系列的第四卷就起到了很好地提纲挈领的作用。如果大家希望阅读这一系列，并能在阅读之后真正有所收获，我建议的阅读顺序为：首先阅读本系列第一卷，其目的在于对模式（架构模式、设计模式与惯用法）与模式系统有一个初步的认识，可以算作是登堂入室的第一步。没有踏出这第一步，就很难谈得上你能够很好地理解架构模式。这一步骤可以是一个快速浏览的过程。正如陶渊明所言“好读书，不求甚解”，此次阅读完全可以遵循这七字法则，不必深入理解书中讲解的每个模式。阅读时不能理解的，可以暂时搁置在一边。

有了第一遍的基础，就可以翻开第四卷，开始真正的模式之旅了。本书有价值的内容全部都放在了《第二部分 模式故事》与《第三部分 模式语言》中。当然，如果你希望了解什么是分布式系统，也可以快速阅读第 2 章的内容。本书的第 5 章《基线架构》写得非常漂亮，极好地表现了作者非凡的写作功底与扎实的技术功底，这是我读过的除了《企业应用集成模式》第一章《运用模式解决集成问题》与《设计模式》第二章《实例研究：设计一个文档编辑器》之外，将理论与实际案例结合得最好的一章。事实上，这个案例应该是从第四章开始，简略地介绍了整个案例的需求，然后就从架构环境，层次分解，领域层，分布式通信，用户界面，功能分布，并发处理，数据持久化等诸多方面，层层递进地勾勒出整个仓库管理流程控制系统的架构过程，且非常完美地嵌入了本书所要讲解的主要模式。因此，用这个案例来开始你的模式之旅，就让自己现在走的一步站在了更高的起点上。

将如此众多的内容浓缩到这一章中，必然会产生阅读障碍。没有关系，你可以默念“不求甚解”的真言进行你的阅读，它可以给你心灵上极大的安慰与自信，使得你不至于在面对困难与挫折时，直接缴械投降。这确实需要足够的信心才能让自己不至于中途放弃，因为这样磕磕巴巴的阅读之旅，会一直持续到第 7 章《仓库拓扑》，你才可以停下来喘一口气（事实上，从第 4 章一直到第 7 章，才算完整地介绍完整个项目案例的架构设计与分析）。休息好了吗？收拾好心情，活动活动疲惫的双脚，做好继续迎面困难的准备吧。因为前方还有无穷多的高峰和险峰在等着你去征服呢。从本书的第三部分开始，你就进入了模式语言的世界中，希望你能保持足够的清醒，不要在模式语言中迷了路。在这个环节中，我认为最好的阅读方式应该是一种“超链接阅读”方式，简言之，就是以本书为纲，开展纵向和横向的阅读，就像浏览网页时不停地点击超链接了解相关知识一般。你需要准备足够多的资料，包括本系列的前面三卷以及其他相关书籍。

举例说来，当我们阅读到第 9 章介绍的 Domain Model 模式时，你绝对不能仅满足于本书的内容，而应该在阅读完本书对该模式的介绍后，再去翻阅 Martin Fowler 的著作《企业应用架构模式》，找找看 Martin 对 Domain Model 是怎样描述的。又例如在阅读到第 10 章的 Message Endpoint 模式时，可以翻阅《企业应用架构模式》第 3 章中对该模式的介绍。阅读到本书第 15 章的 Leader/Followers 模式时，应查阅《面向模式的软件架构》第二卷第 5 章的内容。如果你仍然没有很好地理解，你还可以通过本书提供的参考文献获取相关知识，又或者通过互联网搜索相关的文档。而这正是所谓“主题阅读”的阅读形式，严格地讲，它更像是一种学术研究，通过延伸与比较的阅读，彻底明白每一个知识主题。

当把这些模式（当然，你也可以只选取你感兴趣或认为重要的模式）全部吃透之后，或许光阴已经飞逝一年或者二载。不管怎样，通过这种精读的形式，并结合自己项目中的经验与收获，你已经在架构模式之旅中行进到足够远的距离了。此时的你可以再回过头来通读

本系列的第一卷，以及第四卷，特别包括第四卷的第二部分，你一定会产生豁然开朗的感觉。

会否觉得这是艰辛而又漫长的旅程呢？既然你已经选择了走上软件开发的道路，在希望提升自己的前提下，就一定做好了这样的准备。与其临渊羡鱼，不如退而结网，最重要的不是学习方法，而是执行力。即使是最美好的学习方法，如果没有执行，仍然是镜花水月。倘若是在犹疑不定中三心二意地阅读本系列，一定会半途而废；因而要想阅读本系列书籍，就必须下定决心，准备好把这一年的业余时间献给这无与伦比的不朽著作吧。顺带说一句，在阅读这一套艰深的书籍期间，可以配套阅读一些相对轻松的书籍，例如《黑客与画家》，《浪潮之巅》或者《程序员的思维修炼》之类的书籍，就当是给自己的大脑放放假吧。

热点新闻 | News

Facebook 元老王淮谈科技公司应有的工具文化

作者 郑柯

王淮是 Facebook 的早期员工，也是 Facebook 内部第二位中国籍工程师和第一位研发经理，曾经负责支付后台和安全系统，担任反欺诈部门的技术经理。现在，他是一位以兴趣为导向的顾问型天使投资人。

前不久，他发表了一篇文章——《以 Facebook 为案例剖析科技公司应有的工具文化》。其中，他开宗明义提出：

不断发展、改进公司的内部工具，可以极大提高每个员工的工作效率，可以减少运营人员的数目；这样既改善了整体协调，又减少了整体开支。

王淮提到：当时招聘他进 Facebook 的总监黄易山，是对内部工具的最有力倡导者：

他极度建议，公司要把最好的人才放到工具开发那一块，因为工具做好了，可以达到事半功倍的效果，所有人的效率都可以得到提高，而不仅仅是工程师。

王淮接下来介绍道：Facebook 有两个工具组——研发工具组、网站支持和工具组。

研发工具组的工具包括：

- 管理分配所有的开发专用服务器

在一个页面上你可以很清晰的看到所有的开发服务器,包括哪些人是现在的使用者，什么时候申请分配的，服务器的操作系统版本，配置信息等等；对于空余的服务器，你可以一键申请，并自动初始化该服务器。这让刚加入的菜鸟们可以迅速的获得自己的研发活动空间。

- Git 集成工具

在提交代码之前自动的检测所修改的代码是否符合公司代码规范，如果不符合，该工具会自动警告，但把决定权交给工程师。

- 现已开源的代码审查工具 Phabricator

工程师可以在页面上非常方便的针对每一段（单行或者多行）代码进行交互讨论；负责审查的工程师可以接受代码改变，可以提出疑问要求原作者继续修改，可以提出自己不适合以推出该代码审查，等等。只有代码被明确接受之后才能被工程师提交到服务器端的代码库，这一点集成到提交工具中强制执行。

- 灰度发布工具

在这个工具中，工程师和产品经理（也可以授权给其他非研发人员）可以设计新产品发布的目标人群特点（比如年龄，性别，地域，教育，等等方面做出限制）及发布的人群比例（在 0-100%之间自由调整），所有的改变不需要代码的改变，只需要在工具页面上点击鼠标即可，让灰度发布变得很轻松。

- 发布过程监控工具

发布的过程由一个利用点对点（BitTorrent）算法实现的工具进行多线程同时发布，对于更新几十万台机器只需要几十分钟。由于是不间断的发布，对公众的服务不可以停，所以 Facebook 会将一部分的机器从公众服务状态中拿下来，更新之后再放回，然后继续下一批，直到所有机器都被更新。这样就可以保证在任意状态都有足够多的机器来支持用户访问。……工具检测发布过程并且将其进度可视化，你可以很方便的看到哪些服务器更新了，现在正在更新哪些服务器，整个网站的进度是百分之几，等等。

- 数据检测工具

数据收集只要 1-2 行代码即可完成，数据的整理分类存储皆在后台的上万台服务器上自动完成，数据的可视化报表只需要通过一个页面工具点点鼠标设置即可即时生成，而不需要任何代码；数据波动的自动警报也可以设置，可以自动发邮件发短信。

王淮说到了这些工具背后的基本理念：

> 基本理念就是凡是被很多人不断重复的好的习惯，要将其自动化，绑定到工具之中。以“Don’t make me think”的方式来推广好的 practice。

当然，他们当时仍使用白板作为最重要的人为工具之一：

把最最重要的目标及相关的任务，目标日期，负责人等信息写到白板上挂到我们最近的墙。每天一抬头就可以看到，每次开会都会路过，时刻提醒我们最最重要的事情是什么。这种工具对我们组非常有效。

在网站支持和内部的通讯工具方面，王淮提到的工具包括：

- 用户问题自动分配工具

Facebook 内部的处理工具做得最重要的事情就是把相关问题自动分配到最相关的运营组（routing），比如和支付欺诈相关的问题应当自动分配到反欺诈运维组的那十几个人那边。然后工具会提供常见的通用解决方案，绝大多数的回信内容自动产生（用户姓名，原问题等个体信息都会自动嵌入）。

如果针对某一个功能的问题突然多起来，工具会自动发现，并提醒运维人员手动查看

所有用户问题的回复率，回复满意度，交互次数等等都会被统计或抽样统计，以保证客服服务的质量。

- 招聘工具

Facebook 有一套专门的做题系统（Puzzle System）尝试去筛选可靠的工程师。

所有的内部推荐都是通过专门的人才提交工具来上传简历，这个工具和整个招人系统结合。

当然还有必不可少的权限控制 – 只有参加面试的人员才能够看到关于应聘者整个流程的所有资料。

最后，该工具允许打印所有的相关资料以帮助决策委员会讨论该应聘者时拥有所有相关数据。

- 业绩评价工具

这个工具要允许员工自己对自己评价，员工互相评价，员工和老板之间互评，等等；还要考虑权限的管理。并不是一个很容易开发的工具。这个工具一开始是内部开发，但后来还是决定使用 Rypple 来提供专业的业绩评价工具。

接下来，王淮提供了一些对于工具的思考：

- 工具开发是手段，而不是目的。……如果某个需要的工具有其他更专业的人做得更好的话，Facebook 非常乐意付费买他们的服务，而把自己的精力集中在核心产品上。
- Facebook 希望通过工具的方式来解决所有可能想到的问题……能够想到的地方就尽可能用工具。与物理工具不同，计算机工具可以实现“杠杆效应”的反复累积，通过组合这些“杠杆效应”可以达到更高的层级。
- 工具团队不应该是一个由二线员工组成的“事后诸葛亮”的后勤部门，公司里最有才华的工程师应该用公司自己的工具来工作，并且企业文化里要优先反映这些。当公司过了最一开始开发原型证明概念的萌芽阶段之后，开发出优秀的工具并继续加以改善、更新，这比寻找下一个伟大的创意更重要。

王淮也谈到了工具文化面临的最大挑战：

> 现实的最大挑战是，工具团队要招聘新员工有一定的难度。Facebook 的用户已经达到数亿，而且还在不断增长，如果你开发的是直接面向用户的产品，每天有那么多人在使用，那带来的成就感非常棒。而你要说服新员工去开发内部工具，说这样可以带给工程师以及其他同事更高的效率、最终帮助公司做出更好的产品，相对是间接并缺乏吸引力的。

他的解决方法是：

- 用一些具体的工具提高效率的案例和数据来做理性说服；这需要在开发工具的同时要检测工具使用前后的效率变化。
- 为了吸引内部最好的人才愿意到工具团队，企业文化中也一定要着重反映出这一点：在不同的公开场合私下会面都不断的强调其重要性，让所有人都清楚，公司将内部工具视为持续的重要投资。
- 集中精力努力说服几位整个工程部门认同的顶尖工程师加入工具组，具有很好的示范效果和磁铁效应。如果真正做到如此重视，最优秀的工程师是愿意加入工具团队的，可以大大提升同事们的效率，从而更好地服务于用户，这也是一种外部用户所感受不到的成就感。

InfoQ 的读者，你们所在的公司有工具文化么？

原文链接：http://www.infoq.com/cn/news/2012/08/facebook-tools-culture

相关内容：

- 六大数字产业生态系统深入分析与对比
- Sean Lynch 谈 Facebook Claspin 监控工具的由来
- Facebook 工程师的发展之路

阿里云 aliyun.com

开发者大赛 首届

云 计 算 成 就 代 码 之 美

如果你是 开 发 者 ，你热爱 云计算；

如果你是 创 业 者 ，你热爱 云 计 算 ；

那么请与阿里云一起漫步云端，共创云端的精彩！

只要用你手中的“代 码”谱写出令人“心动”的 工 具

你就有机会 获 得 总额 百 万 大 奖

有可能 获 得 宝贵的 风 投 机 会

阿里云成就你的梦想！

活动时间

2012.7-2012.10

官方网站

2012.aliyun.com

官方微博

@阿里云开发者社区

热点新闻 | News

快速哈希算法破坏了加密的安全性

作者 Roopesh Shenoy 译者 侯伯薇

像 MD5、SHA 和 SHA1 之类的快速哈希算法的用意并不在安全性——保护重要的信息，特别是密码，哈希算法必须有意地减慢反向暴力破解速度。Troy Hunt 是一位微软的 MVP，它演示了 SqlMembershipProvider 所提供的密码哈希值如何易于被暴力破解。SqlMembershipProvider 是 VS 2010 中 ASP.NET web 应用程序模板中的默认成员（membership）提供程序。Troy 在他的文章《我们的密码哈希值没有遮挡（Our password hashing has no clothes）》中演示了经过 SqlMembershipProvider 中 SHA1 哈希算法处理的密码安全性是如何被破解的，其中使用的是 GPU、名为 hashkiller 的字典以及暴力破解算法。在规模为 40000 条的真实密码样本中（来自于之前的一次攻击），算法在 45 分钟内破解了 24710 个，占总体样本的 67%。 其中包括被认为是很强壮的密码：

像“volleyball6”，有 11 个字符，两种字符类型。甚至还有“zaq1@WSX”——由八个字符组成，其中有大写、小写、数字和符号，肯定足以通过大多数安全策略，但是即便被存储为“安全的”哈希值，也完全没用。

问题在于你拥有了可能的密码字典之后可以快速创建新的哈希值，并且硬件变得越来越快。那么如何来解决这个问题呢？密钥延伸（Key stretching），通过多次迭代哈希算法，能够提供一种方式来让哈希算法变得足够慢，使得暴力破解更加困难。Bcrypt 和

PBKDF2 都是这样的算法——它们叫做适应性算法，因为通过增加迭代的次数，会随着时间推移而变得更慢（因为硬件变得更快）。Bcrypt.NET 实现了前者，而 DefaultMembershipProvider 实现了后者。 DefaultMembershipProvider 对 SHA1 使用了 1000 次迭代，它是 VS2012 的 ASP.NET MVC 4 模板政所提供的默认选项。Troy 的文章《.NET 中更强的密码哈希算法（Stronger password hashing in .NET）》说明了如何使用这些方法，以及如何把应用程序迁移到使用更强壮的哈希算法，而不会破坏你的身份验证过程。

原文链接：http://www.infoq.com/cn/news/2012/08/hashing-dot-net

相关内容：

- Oracle 和苹果遭受 Java 安全问题困扰
- Windows Identity Foundation 已包含在.NET 4.5 中
- Java 惊现 0day 漏洞，Oracle 紧急升级

热点新闻 | News

HTML5 VS. Flash&Flex? – 浅谈 Flash/Flex/HTML 5 技术

作者 王磊

在 HTML5 发布以前，RIA 领域的技术解决方案一直相都是各展所长，并无争议。Adobe 体系中，Flash 无法胜任的事情，Flex 可以完成，反之亦然；.Net 系决策者在选用 RIA 解决方案时，Silverlight 是不二之选。

曾经我对 Flex 的迷恋到了欲罢不能的地步，与我有相同想法的人亦不在少数，Flex 也大有“一统江湖”的趋势。然而，随着 HTML 5 横空出世，Flex“易主”，Silverlight 被“雪藏”，RIA 领域的技术解决方案开始变得扑朔迷离。

HTML 5 无疑是“明日之星”，苹果公司前 CEO 乔布斯对它赞赏有加，绝大多数智能手机浏览器均支持 HTML 5，基于 HTML 5 的网站也如雨后春笋般出现。这些似乎预示着 HTML 5 时代来临，人们试图让决策者相信，Flash/Flex 时代已经过去了，HTML 5 才是 RIA 领域的最佳解决方案。然而，事实果真如此吗？ 我曾经见过一个项目，原计划使用 Flex 做为前端解决方案，由于当时 HTML 5“盛行”，最终决策者决定弃用 Flex 而转投 HTML 5。 接下来会发生什么呢？

- 由于 HTML 5 的浏览器兼容性问题，导致需求设计阶段的很多功能都需要推倒重做。

- 在实现过程中，不仅要写 HTML 5 标签，还要写 CSS 与 JavaScript，对于项目来说，增加了人员构成，项目的开发成本也随之增加。
- HTML 项目可以方便获取源码，因此需要增强保密性及安全性设计。
- 在插件的编写、框架的选择上，其难度也要远远大于 Flex。

从上述情况可见，HTML 5 也存在劣势，并不完美。同样，我也可以列举出诸多例子来体现 HTML 5 的优势。那么，“真相”到底是什么？

真相只有一个：

HTML 5 与 Flex 是两种截然不同的技术解决方案。HTML 5 的出现让 Flex 更加专注某些方向和领域。所以，它们是互补的，而非替代。因此，“替代”一说并不准确。

虽然，上述例子只是小概率事件。但概率小，不代表不发生，不代表不典型。所以，在这里我想跟大家谈一下 Flash、Flex、HTML 5 的技术选型。

首先明确一个观点：技术选型没有既定的规律可循，它是由诸多因素决定的，例如：开发人员的技术知识结构是否胜任、项目的开发成本、开发人员构成、项目的开发周期、项目的属性等等。

但是，我们仍可以从这些技术的特点出发，辨别你的应用程序适合采用哪种方案！

Flash 的特点：

- 优势：
  - 借助时间线（Time Line）和 Action Script 3.0 可以方便地制作出任意效果动画。
  - 完备的开发工具。（Flash Pro CS 系列开发工具）

- 完备的工作流。（Adobe CS 系列全线工具均可以导入到 Flash 并可二次编辑）

- 劣势：
  - Flash Player 不支持 iOS。
  - Flash Player 不支持 Android 4.1+。（Android 4.0 以下系统均可支持）

Flex 的特点：

- 优势：
  - 完备的、可以媲美 C/S 架构（桌面软件）的大量控件支持。
  - 与 Flash 及 Adobe CS 系列全线工具的完美结合。
  - 完整的企业化开发流程及工作流（代码的编写、编译、调试、发布等）
  - 多种框架可供选择，并支持高级特性，如：IoC、视图绑定、数据绑定实时更新等。

- 劣势：
  - 生成的 SWF 过大。（虽有完善的“瘦身”方案，但仍比 HTML 方案大很多）
  - 效率问题。（在某些情景下，比 HTML 5 的效率要差一些）

- 较差的图文混排支持。（无法媲美 HTML 5 的图文混排，这是 Flash 系的通病）
- 储备人员相对 HTML 来说还是太少。

HTML 5 的特点：

- 优势：
  - 完备的技术人员储备。（前端开发人员的数量完全可以跟 Java、.Net 程序员媲美）
  - 借助 HTML 5 的诸多新特性，在某些层面完全可以取代 Flash 技术。（Flash 属于 Plug-in 方式，而 HTML 则是浏览器原生支持）
  - 不逊色于 Flex 的大量控件。（Bootstrap、基于 jQuery 的控件比比皆是）
  - 真正意义上全平台支持。
  - 借助 Node.js 可以胜任后台（前/后台通吃）；借助 Coffee Script，可以媲美 Ruby/Python 的语法糖衣。
  - 比 Flex 拥有更大、更全面、更活跃的社区。
- 劣势：
  - 作为企业开发，不具有媲美 Flex 的工作流及开发流程。
  - 作为游戏开发，在支持 3D 及运行效率方面，不如 Flash Stage3D。

- 编写 HTML 5 的应用程序，很大程度上还要编写 CSS 与 JavaScript，对初学者来说，学习曲线较 Flex 高一些。
- HTML 5 依然存在浏览器兼容问题。（随着 W3C 与 WHATWG 的分裂，估计这种情况会被进一步加深）
- 在大型 HTML 5 的项目中，Flex 遇到的问题在 HTML 5 中依然存在（例如：效率问题），在此基础上还增加了浏览器兼容性、Ajax 跨域通讯等新问题。

下面的表格，描述了这三者在一些关键点的比较：

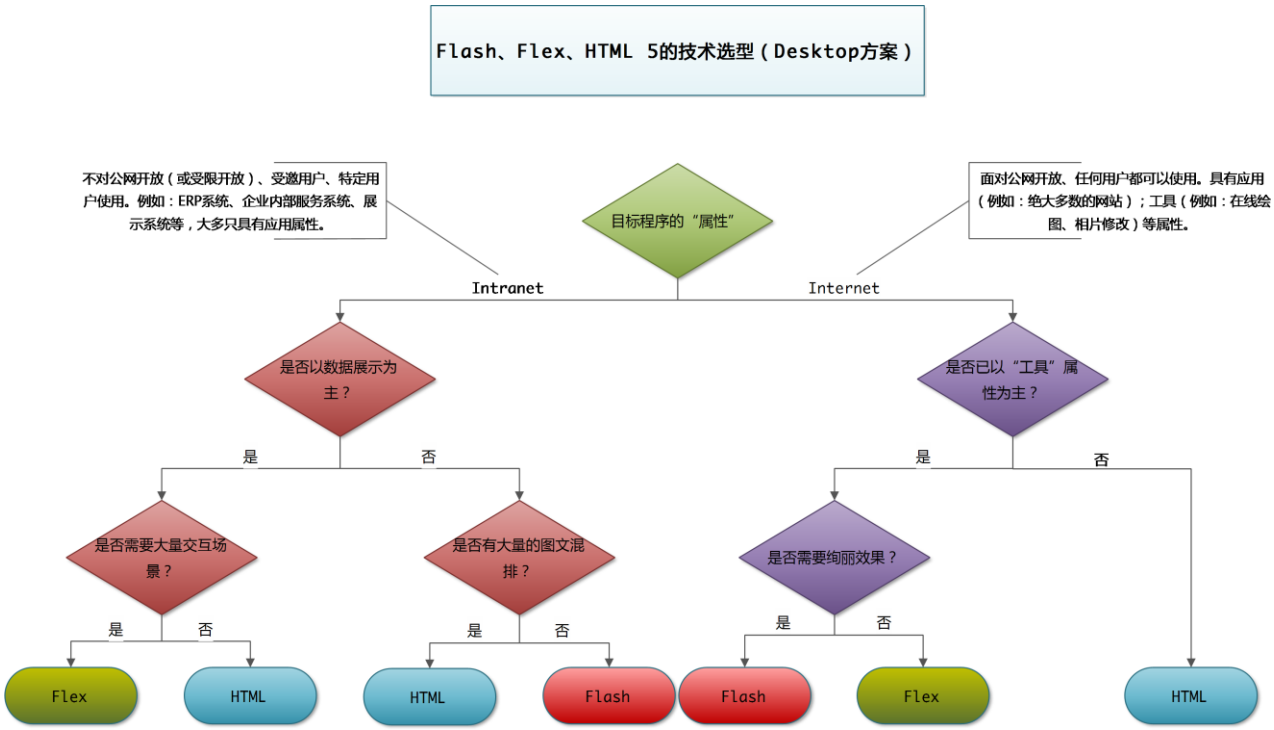
Flash、Flex、HTML 5的技术比较

	语言	动画（2D）	动画（3D）	浏览器（Desktop）	浏览器（Mobile）	后台	后台通讯	跨域通讯	开发人员储备
Flash	Action Script 3.0	•	•（Stage3D）	安装有Flash Player即可	iOS、Android 4.1+不支持	◦	HTTPService、WebService、Remote、Socket等多种通讯方式	较容易解决	适中
Flex	Action Script 3.0 + MXML（标记语言）	•	•（Stage3D）	安装有Flash Player即可	iOS、Android 4.1+不支持	◦	HTTPService、WebService、Remote、Socket等多种通讯方式	较容易解决	较少
HTML	HTML 5 + CSS 3 + JavaScript	•（<canvas>标签）	•（Web GL等）	部分浏览器不支持	•	•（借助Node.js）	GET、POST、Ajax、Socket（Web Socket）、Node.js（直接用JS写后台）	较难解决	较多

	iOS APP	Android APP	WP8 APP	社区支持	浏览器兼容性	框架	开发人员对它的态度	学习曲线	
Flash	可以媲美原生的APP，但缺少与iOS的交互（可以用ANE方案，但学习成本相对高）	通过安装AIR for Android，可以支持	◦	•	无兼容性问题	PureMVC、Robotlegs等（相对Flex来说，其支持的框架比较少）	是制作Flash Game的首选解决方案	中（美工并不是必备条件，但是需要掌握一些基础）	
Flex	可以媲美原生的APP，但缺少与iOS的交互（可以用ANE方案，但学习成本相对高）	通过安装AIR for Android，可以支持	◦	•	无兼容性问题	Cairngorm、Mate PureMVC、Swiz Robotlegs、Parseley等（支持IoC、数据绑定、视图绑定）	曾经是企业开发的首选方案，但，Adobe将其贡献给Apache基金会，导致很多人觉得它以无“前途”。相对于HTML 5来说，在Desktop端Flex还是有着非常明显的优势	较低	
HTML	Hybird APP方式，执行效率较差（PhoneGap打包）	Hybird APP方式（PhoneGap打包）	Hybird APP方式（PhoneGap打包）	•（比Flash、Flex更广泛的社区支持）	有兼容性问题	Backbone.js、Ember.js等（不具有IoC，在企业级开发中无法与Flex框架媲美）	明日之星，随着Jobs的大力追捧导致“过度生长”，被冠以“Flash Killer”存在。在某些方面的确能替代Flash。但，W3C与WHATWG的分裂加剧了HTML的不稳定性、不确定性。	较高（入门简单、提高难）	

版权 Kenshin
weibo.com/Z3784148

下图使用区分法，来辨别应用程序的技术选项方案：



版权 Kenshin
weibo.com/23784148

下面的表格从“项目属性”角度来比较这三者之间的优劣性：（图 3）

Flash、Flex、HTML 5的“项目属性”比较

	图文混排	显示效果	交互效果	应用程序“属性”	可移植性
Flash	差	优	优	展示/工具	良
Flex	差	良	优	工具	良
HTML	优	良	良	应用	良

版权 Kenshin
weibo.com/23784148

总结：

- Flash：

- 适合强交互、强效果、少数据展示、少图文混排、偏展示/工具属性的应用程序。例如：Flash 交互广告展示、页游（Flash Game）等。

- Flex：
  - 适合较强交互、适当效果、多数据展示、少图文混排、偏工具属性的应用程序。例如：图片在线修改、企业内部系统、ERP 系统、金融系统等。
- HTML 5：
  - 适合较强交互、适当效果、多数据展示、多图文混排、偏应用属性的应用程序。例如：Google 系网站、各种传统意义的网站、SNS 系网站等。

注 1：上文提到的 Flash 是指使用 Flash Pro CS 工具（IDE）生成的 SWF，而非 Flash Platform。

注 2：Flash Player 不支持 IOS 以及 Android 4.1+系统。在 2012 年 8 月 15 日，Adobe 将 Flash Player 从 Google Play 下架，并不在对其进行更新，但 Adobe 仍使用 AIR 的方式对智能系统进行 Flash 应用/游戏的支持。

注 3：虽然 Adobe 提供了 FTE（Flash Text Engine）与 TLF（Text Layout Framework）用于图文混排，但是由于其开发的难度比较大，所以并不完全适合企业开发。

关于作者

王磊 | Kenshin（kenshin.wangl@gmail.com），Adobe Flex 专家、国内第一本 Flex 图书《Flex 第一步》作者、05 年开始 Flex 至今，是国内第一批从事 Flex 的开发人员，曾

任某公司前端技术总监，目前从事 Flex 顾问、咨询、架构、高端培训等相关工作。 新浪微博：http://weibo.com/23784148， Flex 博客：http://www.k-zone.cn/zblog

原文链接：http://www.infoq.com/cn/news/2012/08/html5-vs-flash-flex

相关内容：

- JS Pattern 及 HTML5 Boilerplate 开发实践
- Embarcadero 更新 Delphi 和 C++ Builder，发布 HTML5 Builder

热点新闻 | News

产品经理需要懂技术吗？

作者 崔康

优秀的产品经理需要具备的素质是多元化的，从硬技能到软技巧，那么产品经理是否需要懂技术呢？对其职业发展会有什么影响呢？国内社区对此展开了讨论，各抒己见。

herock

> 忌不懂技术——更准确的说，是不能缺乏设计、开发一个互联网产品基本的技术常识，比如至少要清楚一个网站从不存在到能被用户访问，需要哪些必须的环节；也要明白一个 App 从你的脑海走到用户的手机里，需要经历怎样的过程。常识，当然不一定就能做出好产品，但没常识，就很象在村里呆了半辈子的人乍到城市，一举一动即使小心翼翼，也没法儿不透着突兀和不和谐。我接触过的业内人士里，有几个七十年代初的颇为典型，大概是互联网出现的时候，他们已经过了充满好奇和渴望未知的年龄，不愿意放低身段去学习新东西，喜欢只凭着想象和自己的生活经验就开喷，间或以若干近期热门关键词作为点缀，以示自己尚蹲在潮流尖端。这样的人也许能忽悠某些领导，但一定不招工程师待见，他们可能什么都不说，但心里已经开始等着看笑话，交给他们的开发需求，自然也是能拖则拖、能蒙则蒙.....
>
> 忌懂技术——我遇到不少工程师喜欢说：“只要产品需求明确，技术上一切都能实现。”这句话听起来相当豪迈，也让产品经理大为放心，觉得技术真是产品的坚强后盾。但其实传递了一个特别糟糕的信号。当工程师这么说的时候，潜台词是：“你弄

好你自己的事儿就行了，别来管我！” 而且这种说法隐含着一个乐观但显然并不现实的假设：技术是无所不能的，他（掌握技术的人）也象灯神一样，可以实现你的任何愿望，只要你能明确的描述它。我不知道阿拉丁说完愿望之后，假如胆敢继续追问灯神将具体采用何种技术方案来实现的话，会不会被塞到灯里，但我知道很多工程师在发现你关注技术层面过深的时候，都会有种领地被侵犯的感觉.....

Fenng

> 我的观点是：产品经理要比开发人员更懂技术。
>
> 懂技术，不意味着写代码比工程师更好，而是明白技术能用来做什么。这个非常重要。
>
> 懂技术，也会理解开发人员，一起做事情更会得心应手。
>
> 懂技术，在针对一些功能实现的可行性上，才会有所取舍。

Hans

> 见过几个小公司有真正的产品经理？ 见过多少百分比大公司的产品经理是从开发转过来的？
>
> 一切都是业务需要， 高级产品经理的核心价值，即做决策、统筹资源与迭代的频率，并踩准形势的 timing，不断被用户驱动，突破引爆点。
>
> 如果只是一个写规划、文档、项目协调的专业角色，技术背景和理解力不过是增加职业发展的砝码而已。 懂多懂少，又能影响什么呢？如果不懂，那怎么做决策呢？

范凯

> 我觉得产品经理不需要懂技术，至少不需要自己懂编程，会写代码。产品经理非逼着自己学习编程实在没有必要。但是，产品经理有一项素质非常非常重要，就是逻辑思

维能力和抽象能力必须非常强。

在现实环境中，产品经理被研发人员鄙视或者消极配合，很大程度上是因为产品经理设计的产品功能、流程存在逻辑上的硬伤。研发是负责具体执行的，一旦落实到细节，这些问题就会暴露。研发是很强调逻辑思维能力和抽象能力的，但是产品经理必须具有更好的逻辑思维能力和抽象能力，这样才能指导好研发做产品。

我招聘产品人员，培养产品人员，非常注重逻辑思维能力的训练。现实中逻辑性好的产品经理和研发配合都不错。

郭欣

> 这取决于做什么产品，以及做产品的哪部分，还有“懂技术”也有各种程度，总之这个问题不能一概而论。
>
> 如果是做监控宝，我们要求产品经理必须懂技术，因为这本来就是技术型产品。
>
> 如果是做一个非技术型的产品，懂技术的产品经理可以更好的在合理设计、实现难度、性能三者之间做好权衡，如果网站没啥流量，那性能就算了。
>
> 你可能说产品经理不用管实现和性能，理论上是这样，但是前提是你得有一个比你这个产品经理更懂产品的程序员来帮你预见未来可能会让你抓狂的问题。

林进鑫

> 在进行技术思考时，只需要知道可不可以实现，而不应过多去考虑它如何实现。
>
> 将表现机会留给团队成员。在与团队成员讨论功能的实现方案时，谨记你给出的只是建议，而不是命令，除非该成员对如何实现某个功能真的一愁莫展，除非该成员是个新手或者能力太烂，又或者这个功能至关重要。

因为是技术已经深入技术型产品经理的基因，所以理所当然对技术会有所关注，要注意的是，不能把太多时间投入到技术（假设你的公司不是技术驱动型），在运营、营销、用户体验也要给些精力。

姜沈励

> 我认为，如果你能达到下面三个要求，懂不懂技术都无关紧要。
>
> 1. 很多产品的诞生往往和技术创新有直接联系。如果你能对新技术的发展有敏锐的嗅觉，并且理解新技术可能对用户带来的价值，那么，你不需要懂技术。
> 2. 如果你可以和开发团队流畅沟通，你能明白他们的困难，他们能理解你的表达，那么，你不需要懂技术。
> 3. 如果你的需求从来不会以“不具备技术可行性”被开发打回，那么你不需要懂技术。

涂照锐

> 产品经理相当于一个打车的，知道要去哪儿，并且知道：哪条路最短；哪条路绕一点但是最顺畅；如果哪条路限行了，该走另外一条路.....那么，技术开发团队就相当于是司机。打车的可以不会开车，但是要到目的地，知道上面的东西，可以省钱，省时间~ 司机可以接不同的人，不同的路线，只要能走到目的地就行。

读者朋友对此有何看法？产品经理需要懂技术吗？

原文链接：http://www.infoq.com/cn/news/2012/08/pm-need-tech

特别专题 | Topic

一淘网测试架构师黄利在他的博客上发布了翻译的一个专题系列文章：《谷歌如何测试》，整个系列文章从全局到局部地介绍了谷歌有关测试的情况。

单元测试的粒度问题一直是软件开发社区面临的现实问题，最近，陈皓针对StackOverflow 上的老问题做了总结，并发表了自己的看法，读者在随后的评论中也进行了讨论。

过去几个月间，互联网上关于测试先行还是测试居后、代码测试比率或者行为驱动开发（BDD）是否真的是测试驱动开发（TDD）的讨论进行得如火如荼。InfoQ 就此访问了行为驱动开发（BDD）和测试驱动开发领域知名的专家们，请他们对测试驱动开发（TDD），行为驱动开发（BDD）以及测试比率的运用发表各自的观点。

分布式系统测试在阿里云的实践

作者 许吗兢 陈舟锋

分布式系统的测试是一个比较大的话题，在这里，我们仅用几个阿里云飞天分布式系统测试中比较有特点的实践方式来阐述一下我们对分布式系统测试的理解，希望对大家有所帮助。

1. 阿里云的分布式系统

飞天是阿里云独立开发的大规模分布式计算与存储系统，兼有分布式存储和分布式计算的多重功能。基于飞天大规模分布式系统，我们开发了弹性计算，海量邮箱服务，Key - Value 存储引擎，结构化数据存储引擎和海量数据处理服务等一系列的上层服务，并且基于这些上层服务，我们运营了了 Open Table Service 和 Open Storage Service 等多项开放服务。

2. 测试的层次结构

按照层次和功能，测试可以分为单元测试，功能测试，系统测试，集成测试，端到端测试等若干种测试。这里首先要提的是单元测试，单元测试更像是一种预防代码错误的手段，而不是检测代码错误的手段，我们用覆盖率（包括行覆盖和分支覆盖）来检验单元测试的

质量。当然我们也都知道这些指标不能作为单元测试好坏的标准，所以，在项目中，我们也非常注重代码审查，其中包括了单元测试代码的审查。

传统的一般来说，一个产品一般只有对外接口进行功能测试和系统测试，但是由于飞天的代码非常的复杂，模块非常多，每个模块都具有了一个产品的复杂度。所以，我们在模块层面也会进行功能测试和系统测试，为的是保证系统质量能够得到回归，在有代码修改的时候，影响上层业务使用的接口或者隐含逻辑一旦发生变化，能够第一实践知晓。功能测试非常注重对接口函数的覆盖，并且会对各种参数的边界进行复杂的探测，而系统测试比较注重稳定性，一致性，性能甚至容灾能力等各个方面。

在飞天系统内部，有专门的集成测试部门，集成测试团队有一个非常大的责任是通过回归测试用例来保证系统中的各个模块，在版本发布时在一起能够很好的工作，并对之前每个模块定下的质量验收标准进行验证。一旦发现重大风险，集成测试部门有权力不发布其中某些 Feature，如果这个 Feature 有某种重要的商业用途，则进行推迟整体版本的发布。这一点可能和传统软件不太一样。当分布式软件已经比较成熟的时候，底层很多模块的 Feature 在某个版本的删减，大多数时候不会影响到用户的正常使用，但是如果由于这个 Feature 的发布导致分布式系统出现不稳定或者性能大幅下降，则得不偿失。

端到端测试在内部有个简洁的名字叫做 E2E Testing。E2E Testing 的责任人需要负责向上层使用这个模块或者系统的用户方了解具体的需求，这个需求不仅是一个对接口功能的需求，还包含了数据量的需求，机器规模的需求，吞吐率的需求，延迟时间的需求以及业务量在一天或者一周内的曲线等等信息，然后设计 E2E 的测试程序，能够尽可能的模拟真实的数据情况。在本身上层业务就已经有数据的情况下，还可以通过导入真实业务的数据进行测试。通过长时间的 E2E 测试，验证各种指标。

一般来说，我们只有在通过了最后的 E2E 测试之后才能发布版本，上线让应用使用我们的分布式系统。

3. 注重探索试测试

分布式系统的测试非常的复杂，你既无法通过穷举来罗列所有的情况，也不能简单的对接口和性能进行回归来证明一个版本的合格。因为对于分布式系统，不同的数据量，不同的压力，不同的机器规模，有可能在代码里面走的路径完全不一样。而且，由于在大的压力下，磁盘，内存和网络有可能成为系统的瓶颈。更有甚者，在压力不均匀的情况下，单台机器可能会成为整个系统的瓶颈。所以，我们时没有办法设计这么多的测试用例来分别覆盖所有的这些分支情况的。

我们有详细的监控系统可以监控整个集群的各种参数，这些参数不光是硬件参数，更多的是软件本身通过调用我们监控系统的 API 来完成对自身某些指标的统计。这些统计不光在线上系统能够起到监控的作用，执行测试的人员可以通过不断改变测试的各项参数，结合这些指标的变化进行探索式的测试。通常，通过指标在某些压力变化下或者随着时间推移时的异常行为，测试人员会更容易找到一些深藏的 Bug。另外，在系统级测试时，通过对系统内部 Error Log 的监控也能达到很好的效果。

4. 灰盒测试是分布式测试的最有效方法

上面讲到了探索式的测试十分重要，但是光有探索，如果测试人员本身不了解系统的一些内部逻辑，会出现两种情况：第一种是，只验证设计好的场景，其他一些异常的情况，自己无法解释，但是本身又不是验证标准，导致很多隐藏的问题最终在线上爆发；第二种是，

想一个没头的苍蝇，漫无目的的进行探索，最终浪费了时间，却达不到好的效果。在飞天分布式系统测试中，我们要求测试人员必须从方案设计之初甚至是讨论需求的时候就和开发的同学在一起讨论，测试的同学需要比开放更加理解为什么这个系统设计成这个样子。有一个很典型的例子是，早期，我们内部开发一个基于表结构的存储引擎时，曾经出现过这样一件事情：测试程序在最终验证一致性时，一直都是通过的，但是业务方和我们一起做 E2E 测试的时候，会有很低的概率发现数据读出来是错误的。测试人员百思不得其解，最后发现，这份数据在写入的时候，会先在三个地方进行修改，但是由于一些时序和锁的问题，在改过了两个地方之后就返回成功了，第三个地方是在内存中，过一阵子就会被重新刷成正确的值。如果当时测试的同学知道系统里面这些设计，当时就会设计写入过程中，对数据一致性的检测，不会在用户场景中再发现问题，解决的效率也会因此而提高。

5. 带压力和随机故障模拟的长时间稳定性测试

分布式系统往往是在多台机器上运行多个进程，每个进程都是一个独立主体，运行着自己独立的逻辑，各个进程都有可能因为软硬件故障导致逻辑执行异常。分布式系统要保证的是任何局部的进程异常都不会影响系统的可用性，因此必须要做必要故障模拟和恢复测试。由于进程之间通讯协议的复杂，故障类型的多样，并不能够模拟所有可能出现故障和进程状态的组合，也就是说这样的组合是爆炸的。

在飞天实际的测试过程中，我们采用的是带背景压力和随机故障模拟的长时间稳定性测试。这也是飞天主版本发布的最后一道关。背景压力主要是各个主要模块的读写压力，还有一些诸如 CPU，Memory，Network 的资源消耗器，以模拟机器资源紧张场景。故障模拟操作有比如磁盘错误，包括坏盘，只读等，机器宕机，重启，断网，主要模块进程重启，假死等，这些操作都会按照预先设定比例进行随机组合；在这样的背景压力和故障操作下，

长时间（至少 7x24 小时）持续运行上层应用模拟程序/作业。同时通过在线监控工具来检查系统是否正常。飞天很多重要的 bug 都是通过这个测试被发现的。当然这类测试也有短处，就是问题调查需要较长的时间，要求测试人员对系统有较深的了解和诊断能力。

6. 如何保障测试的质量

单元测试虽然可以用 Coverage 工具来检验行覆盖和分支覆盖，但是功能测试和系统测试在运行时，进程运行在上百上千台机器上，很难通过既有的软件来搜集到 coverage 数据。我们内部开发了一个叫做 Log Coverage 的工具，能够通过程序在运行过程中输出的 Log 信息的多少来判断测试是否有足够的覆盖率。通过拿到生产线上的 Log 与测试中的 Log 进行比较，会找到我们之前没有测试到的地方。另外，通过对程序中从来没有打印出的 Error Log 的检查，我们也可以知道有多少异常逻辑我们没有测试过。

“窥探”谷歌测试

作者 黄玲艳

一淘网测试架构师黄利在他的博客上发布了翻译的一个专题系列文章：《谷歌如何测试》，整个系列文章从全局到局部地介绍了谷歌有关测试的情况。

译者黄利在《译者序》中阐述了在软件开发模式（尤其是互联网）中，近几年的快速迭代发布，以 Beta 版本线上运行，让大家对测试产生了一些误解：

- 这些应用没有经过很好地测试，好多功能使用上都有问题；
- 测试水平比较有限，没有能及时的发现潜在问题；
- 测试本身没有太多的技术，基本上是功能确认，点点鼠标、搭建环境验证下就可以；
- 只要认真仔细，有责任心就可以做好测试；

这些误解让很多人特别是应届生，都不会把测试作为职业规划来考虑，黄利表示:“想通过这个系列的讨论，让大家清楚测试工作如果做好，方法其实并不是想象中的那么简单和表面上的肤浅，是非常好有挑战的。”

关于《谷歌如何测试》，黄利已经翻译了六篇：

• 第一篇：介绍谷歌工程生产力部门构成，以及这种测试人员的项目分离和汇报组织结构的优点与缺点。

> “谷歌没有真正的测试部门，测试依托在各个产品部门里，即“工程生产力”，这个部门由以下几个团队组成：工具产品团队（负责内部和外部开源的促进生产力的工具开

发与维护）；服务团队（给产品部门提供一些专业的建议，包括一系列工具、文档、测试、发布管理、培训等方面，这些专家建议涵盖可靠性、安全、国际化等，甚至包括产品团队面对的功能问题）；嵌入式的工程师（在需要的时候被产品部门高效地“借”去使用）。

在测试人员的这种项目分离和汇报组织结构中，其优点是开发和测试将具有相同的地位，缺点则由于测试人员被看做外部资源，产品部门团队不能对测试人员有太多的依赖，他们自己必须要合理地控制产品质量。

- 第二篇：为了让开发人员效率提升，特别是在质量方面的提升，在传统的软件开发人员的之上，增加了几个角色，特别是需要工程技术方面的特殊角色。

> 将工程师的角色细分为：软件开发工程师【SWE，Software Engineer】，就是传统的开发人员；软件测试开发工程师【SET or Software Engineer in Test】，和软件开发工程师一样是开发工程师，主要负责软件的可测试性；软件测试工程师【Test Engineer】，和软件测试开发工程师【SET】恰恰相反，主要工作是做测试而不是开发。
>
> 从质量的角度来看，软件开发工程师对功能开发和质量负有全责，软件测试开发工程师是提供测试支持的开发人员，软件测试工程师的职责就是最终用户级别的测试。

- 第三篇：质量不等于测试，“质量不是被测出来的”。

> 最简单的办法是不要区分开发和测试，不要把他们当成对立的活动。测试和开发【注，两种行为，不是人】最好能手牵手的并行，写一点代码就立刻进行测试，写的越多，测的就要越多。最好是，在编码的同时，甚至在编码之前，就考虑清楚这些代码将如何被测试。测试不是一个单独的工作，测试就是开发的一部分。所以，质量并不等同于测试，当把开发和测试混在一起，搅拌直到分不清他们彼此的时候，就得到了质量。

对于质量来说，预防问题比发现问题本身更重要。质量是开发人员的问题，而不是测试人员的问题。通过把测试工作融入到开发过程中，我们能降低那些富产 Bug 的人的出错机会，不仅可以避免了大量最终用户的使用问题，而且还可以极大地降低测试人员报无效 Bug 的数量。在谷歌软件测试工程师的工作目标就是检查这种预防措施是否有效，软件测试工程师不停地寻找一些证据来证明作为 bug 的作者和预防者的“软件开发工程师-软件测试开发工程师”组合是否存在问题，一旦发现任何不正常，就会拉响警笛。

•第四篇：从爬到走、走到跑的模式，在一个产品的核心模块被开发后，如果有一定数量的受益人群就立刻发布，然后不断的得到用户反馈再迭代开发新功能。

> 这样爬、走、跑的模式对分析也有益处。例如，发现了一个 bug，测试人员可以根据这个 bug 创建一个测试用例，并针对所有的每一个版本都运行这个测试用例，从而可以验证这个 bug fix 是否在所有的版本中都真正得到了修复。

• 第五篇：测试范围的定义，以及自动化测试和手动测试。

> “哪些需要被测试及测试范围的确定，这是一个动态变化的过程，在不同的产品之间会有比较大的差异。谷歌更倾向于频繁发布，从产品的外面用户那里得到反馈之后再迭代开发。”
>
> “自动化测试和手动测试，对于所有的三种类型测试【小规模、中等规模、大规模测试】来说当然更喜欢前者。如果能够被自动化，而且不需要任何人智力和直觉判断，那就应该把它变成自动化的。只有在特别需要人为判断的时候，例如用户的界面是否漂亮、或暴漏一些涉及用户隐私的内容时，在这些情况下应该保留手动测试。
>
> 对于谷歌来说非常重要的是仍然使用了大量的手动测试，不管是使用文本记录的方式还是使用探索性测试，虽然有些已经进入了自动化测试的视线。业界使用的录制技术

将手动测试转变成自动化测试，可以在每个版本后自动地重复运行，这样保证了最少的回归工作，并把手动测试的重点放在新问题上。而且，谷歌已经将提交 BUG 的过程和一些手动测试的日常工作也自动化了。

人类智慧的最后一英寸”体现在测试设计上，谷歌的下一代测试工具也正在这个方向上努力尝试，将其自动化。

• 第六篇：软件测试开发工程师【SET】的生命

> “软件测试开发工程师【Software Engineers in Test】是软件工程师，专注在测试实现。首先，软件测试开发工程师是开发角色”
>
> “通常来说，软件测试开发工程师不会在早期设计阶段就介入。”
>
> “当我说“测试”时，并不是仅仅意味着单纯的检查验证代码路径。测试人员不是从开始就参与进来的，但“测试”却至始至终都有。实际上，一个开发尝试去 check in 代码的时，测试人员的影响力在这个时刻可能就已经显现出来了。”

这个系列的文章还没有完结，也欢迎大家继续关注黄利的博客分享，并通过 InfoQ 网站及 InfoQ 微博等方式参与讨论。

原文链接：http://www.infoq.com/cn/news/2012/07/how-google-do-test

相关内容：

- Android 中的单元测试
- Yeoman：构建漂亮 Web 应用的工具和框架
- 如何控制单元测试的粒度？

如何控制单元测试的粒度？

作者 崔康

单元测试的粒度问题一直是软件开发社区面临的现实问题，最近，陈皓针对StackOverflow 上的老问题做了总结，并发表了自己的看法，读者在随后的评论中也进行了讨论。

John Nolan 在《How deep are your unit tests?》中问道：

> TDD 需要花时间写测试，而我们一般多少会写一些代码，而第一个测试是测试我的构造函数有没有把这个类的变量都设置对了，这会不会太过分了？那么，我们写单元测试的这个单元的粒度到底是什么样的？并且，是不是我们的测试测试得多了点？

最佳答案是：

> 老板为我的代码付报酬，而不是测试，所以，我对此的价值观是——测试越少越好，少到你对你的代码质量达到了某种自信（我觉得这种的自信标准应该要高于业内的标准，当然，这种自信也可能是种自大）。如果我的编码生涯中不会犯这种典型的错误（如：在构造函数中设了个错误的值），那我就不会测试它。我倾向于去对那些有意义的错误做测试，所以，我对一些比较复杂的条件逻辑会异常地小心。当在一个团队中，我会非常小心的测试那些会让团队容易出错的代码。

最让人意外的是，这个答案是由 XP 和 TDD 的创造者 Kent Beck 给出的！ 难怪有人评论说：

> 只是要地球人都不会觉得 Kent Beck 会这么说啊！我们有大堆程序员在忠实的追求着 100%的代码测试覆盖率，因为这些程序员觉得 Kent Beck 也会这么干！我告诉过很多人，你在你的 XP 的书里说过，你并不总是支持“宗教信仰式的 Test First” ，但是今天 Kent 这么说，我还是很惊讶！

陈皓则是非常同意 Kent 的回答：“怎么合适怎么搞，爱怎么测试就怎么测试，只要自己和团队有信心就可以了。没有必要就一定要写测试，一定要测试先行。”

其他排名靠前的答案包括：

Dominic Rodger

> 为可能会出错的地方和边界情况编写单元测试。另外，单元测试应该跟着缺陷报告走，在修补缺陷之前编写好单元测试。开发人员就会对代码充满自信：一是 bug 已经修补，二是 bug 不会重现。

kitofr

> 关于 TDD 最大的误解之一就是第一个字母——测试（T）。这也是 BDD 出现的原因。因为大家没有真正理解第一个 D（驱动）的重要性。我们总是倾向于对测试关注更多，而对设计的驱动关注太少。我想这是一个模糊的回答，但是你应该考虑如何驱动你的代码，而不是测试什么，这是覆盖率工具该做的事情。设计是一个很大而且问题更多的问题。

最后，陈皓表达了自己的观点：

- 目前的国内教育模式让我们习惯于标准答案，习惯于教条，从而不会思考！敏捷开发中的若干东西似乎都成了软件开发中对某种标准答案的教条，实在是悲哀！
- 软件开发是一种脑力劳动，是一种知识密集型的工作，就像艺术作品一样，创作过程和成品是没有标准答案的。
- 软件的质量不是测试出来的，而是设计和维护出来的。就像工匠们在一点一点地雕琢他们的作品一样。

许多读者在原文的评论中表达了自己的看法：

- 对于要做多细，我的经验就是：当我不在的时候，如果同事打电话说有 Bug，询问是否会是我的问题，而我可以很自信的回答 No 的时候，就可以啦！ 所以最享受的就是大家加班，而我在放心玩乐的时刻。这让我想起了一件事情，有位同行朋友学车，一开始总想让师傅告诉他，需要“打点方向”的时候到底是多少度？ 需要“加点油”的时候到底是加到每小时多少公里？ 后来知道了，没人会告诉你，你得自己 Do!
- 其实我觉得在现今大型企业的团队中， 开发人员的职责就是开发和保证代码正确的实现 至于测试则基本是交给测试的人员来做，而测试人员又拿着测试的工资，他们不会考虑开发人员的水平，而是循规蹈矩的去完成整个测试流程。
- 关于 StackOverflow 上出现的大量排斥 TDD 的观点，我尝试这样来解释：占主流群体的底层程序员在没有监督的情况下缺乏责任心只求应付工作.....到了工作岗位上也是如此，领导在不在大不一样。总结下来就是，大多数人在没有监督的情况下容易慵懒应付。这应该是人的天性。后来工厂出现的流水线，应该就是对付这种情况的。把工作流程标准化、流水线化，让整个流水线上的人都没机会偷懒。我想这也是 TDD 的目标之一。如果没有足够的测试用例，你怎么确认某个模块是功能完整的无明显缺陷的？如果原作者离职了，后来的维护者修改了先前的代码，又如何保证不影响以前的功能？要知道，你修改一个新 bug，很可能会把以前的多个旧 bug 放出来，有经验的开发者应该明白我说的什么意思。于是软件越来越难于维护，维护者也越来越排斥修改代码，一潭死水无人敢碰，软件提前寿终正寝。我最终的结论就是，TDD 要由项目管理者推动实施，而不能征求下面程序员的意见，——他们虽然人数众多，但就像面对老师的学生一样，总是要求玩耍而不是学习，这对项目本身是无益的。
- 说白了还是看你怎么用，TDD 也只不过是个方法罢了，工具而已，又不是思想或者本质之类的。我个人开发就很不 TDD，即使需要写测试也是针对公开的接

口或者某些特定的调用去做测试程序，这样的好处是达到测试代码的目的，同时又有了一个测试程序。缺点是这个测试程序不带好集成到发布过程。但换言之，好的程序员写的代码自然质量较高，差的程序员有没有单元测试一样差，甚至于无法保证差的程序员写的测试单元没有 BUG，如果有 BUG 那不是更悲剧？

InfoQ 的读者朋友对此有何自己的见解？

原文链接：http://www.infoq.com/cn/news/2012/09/unit-test-topic

相关内容：

- Android 中的单元测试
- 编写综合的单元测试
- 大型搜索引擎的系统测试方法及案例分享

特别专题 | Topic

虚拟座谈会：代码测试比率、测试驱动开发及行为驱动开发

作者 Manuel Pais **译者** 江辉

过去几个月间，互联网上关于测试先行还是测试居后、代码测试比率或者行为驱动开发（BDD）是否真的是测试驱动开发（TDD）的讨论进行得如火如荼。InfoQ 就此访问了行为驱动开发（BDD）和测试驱动开发领域知名的专家们，请他们对测试驱动开发（TDD），行为驱动开发（BDD）以及测试比率的运用发表各自的观点。（译注：此后测试驱动开发和行为驱动开发均用简称 TDD 和 BDD 代替。）

座谈会成员：

> J. B. Rainsberg ——顾问及测试驱动专家，关注他的博客：The Code Whisperer
>
> Dan North ——DRW Trading Group 的精益技术专员, 行为驱动开发（BDD）的定义者
>
> Gojko Adzic —— 顾问，《Specification by Example》及《Bridging the Communication Gap》作者
>
> Ron Jeffries ——极限编程（XP）及敏捷模式的独立顾问，曾指导过极限编程（XP）的项目
>
> Steve Freeman —— 敏捷讲师及顾问，《Growing Object Oriented Software, Guided by Tests》作者

问题列表：

“

1. 对项目而言，你认为哪些标准会影响你做决定，是采用 TDD 还是 BDD 或是什么都不用呢？
2. 现在，人们似乎普遍认为 TDD 等同于单元测试，而 BDD 则等同于验收测试（无论使用的是哪种工具）。你们认为这种说法是否正确？其他类型的测试，如构件测试与系统集成测试，又与 TDD/BDD 有何关联呢？
3. 一直以来，TDD 被公认为是一种（代码）设计准则、测试准则或沟通工具。然而，在 TDD 方法中，作为这些准则和工具的目标是如何影响设计的？在测试方面，TDD 现在与未来的价值又是怎样的？
4. 关于单元测试、集成测试以及验收测试比率的自动化以及相应的维护成本，有哪些准则可以告诉大家？
5. 除了至关重要的系统，现在似乎比较统一的说法是，不能把 100%的测试覆盖作为目标，那也是不实际的。你是否认为代码/测试比率能提升测试的专注程度和效率吗？
6. TDD、BDD、验收测试驱动开发（ATDD），测试优先等等，对不同的人意义也不尽相同，并且容易让人搞混。我们是否需要一种通用的语言，能够描述我们的软件开发方法，并培养一种可达成共识的内容驱动的好实践呢？
7. 关于这些主题有什么最后寄语？

InfoQ：对项目而言，你认为哪些标准会影响你做决定，是采用 TDD 还是 BDD 或是什么都不用呢？

“JB：对这个话题做概述，我觉得有点不合适，莫不如让我来说明下，我在什么时候会用到 TDD 和 BDD，以及这么做的理由吧。

我第一次接触 TDD 是因为我试图寻找一种方法来减少代码中的错误（也叫代码缺陷或 Bug）。曾经，我在程序中引入的错误数量让我一度认为自己永远都无法完成编码——不管我怎么努力，都看不到任何希望。我猜测，如果那个时候，能自己测一下代码，就能找到许多简单、愚蠢的代码并自己修复掉。TDD 对我来说，并不只是为了不让别人觉得你写的代码有多糟糕，而是避免你错误地认为自己已经完成了编码，却留下一大堆 Bug。然而，TDD 帮我解决了这个问题，多年之后，我还意识到 TDD 不仅帮我避免了程序逻辑的错误，更帮我减少了程序设计中的问题。在我运用 BDD 之后，我发现 BDD 帮我减少了许多在选择特征与完成特征时的错误。日复一日，我逐渐明白代码的错误不仅让我费时费力，也让我无法按时完成编码工作。这个时候，我开始在项目中贯彻实施 TDD 与 BDD 的方法。

让我们再回到问题本身，我鼓励大家去思考为什么你们需要做 TDD，想想你的理由。不要限于典型的实施 TDD 与 BDD 的理由，例如：更好的设计、更少的程序缺陷、更有商业价值的产品特征以及更少的无用功。我鼓励人们多思考一下驱动你这样做的原因，这才是重中之重。我相信你会发现，一旦你采用了这两种方法，必然能按时完成分配的任务，因为这会让你更有条不紊，而不是到后来手忙脚乱。当然，我认为这些理由是因人而异的。

Dan：首先允许我给出一些定义。TDD 是一种编程技术，它引导程序员思考自己的代码是如何被其他代码所使用，从而避免由代码实现引起的“意外设计”（emergent design）。你首先要写一个测试来描述如何使用下一块代码，然后实现这块代码，并

保证它通过测试。这项技术是需要编程技巧的，并且需要考虑何时进行合适的运用（我稍后会展开）。这样做有很多优点，编写测试帮助你理解行业知识并帮助你更好地去命名。一个测试可以发现理解上的差异（“你认为在这个用例中应该怎么做？”），当然，一整套自动测试可以帮助我们发现回归测试中的缺陷。

我并不认为我们必须决定是否在这个项目中采用 TDD，你几乎可以在任何一个项目中使用 TDD。然而，我建议程序员在采用 TDD 前思考一下，这样做是否有价值。不可否认的是，除了 TDD，我们还有其他很多方法可以用来做设计、对某个行业探索和建模以及减少回归测试的风险。有些项目，最有效的方法是逐一采用这些方法，而有些项目不是。不夸张地说，我认为 TDD 是每个程序员都应该了解，并知道如何去做的、很有用的方法。对于重构，我也持有相同的意见——你必须去了解这个方法，但你需要时间的磨练，去区分何时何地采用这种方法，或转投他法。

BDD 是一种开发方法，形式类似于极限编程（XP），但不能把它单单看成是 TDD 的一种实现方式。BDD 的作用是把利益关系人、交付团队等不同方面的项目相关人员集中到一起，形成共同的理解，共同的价值观以及共同的期望值。如果你没有碰到过这样的难题，那就可能是哪里出问题了。当然，我也是最近才渐渐地运用起这种方法，之前我都是与较小的团队一起工作，通常也能有比较迅速和通畅的利益关系人反馈。因此，由理解不同而造成影响的情况比较少见。通常，利益关系人会说：“这里你能帮我改一下吗？”或者“这不是我想要的，我来举个例给你。”

BDD 会从业务目标着手，将这些目标与产品特征和故事衔接起来。BDD 提供了如何确定你的验收标准的建议，以及你如何将这些标准落实到决定代码行为的自动测试上。如果你的项目决定采用 BDD 方法，那就必须从整个项目的层面上来考虑问题（尽管你可以在子项目中运用 BDD），而且必须拉上整个团队一起做。

我赞成以下说法：只要能够提交合格的代码，那么每个程序员或配对编程的人员就都有权利以他们所偏好的方式交付软件。如果他们打算使用 TDD，我们就应该支持他们这样做。如果他们打算做些其他的，只要团队成员没有问题，我们也可以允许。

Gojko：这取决于你如何定义 TDD 和 BDD。看起来，在过去几年间，TDD 的定义被局限在仅仅作为设计的单元测试。而 BDD 则变成以实例和业务导向的测试来驱动功能的、涵盖所有开发阶段的方法。根据 Brian Marick 在《Agile Testing by Lisa Crispin and Janet Gregory》中的敏捷测试象限的定义，TDD 应该在第一象限，而 BDD 在第二象限。我并不完全同意这点，但既然你的问题是什么是影响你采用 TDD 还是 BDD 的标准，我假设你对 TDD 的定义是排除在 BDD 与其他方法之外的。基于这点，我们分析以下三个方面：

这是个一次性项目吗？是否为了将技术的不确定性降到最低，从技术层面，帮助团队了解他们最终想要什么？在这种情况下，维护很多自动测试用例集合都是一种浪费，而且极有可能团队真正需要的只是一小部分相关联的简单用户用例。因为我们不清楚技术壁垒，所以编写技术测试可能会是个问题。我可能会选择写一些指导性的测试，而非教条式地为每个设计写一段单元测试。这符合 Steve Freeman 和 Nat Pryce 所著的有关《成长性面向对象软件》的大型系统测试。

项目的复杂度是否由技术所决定？我们是否认同我们所要完成的是技术上的，而非业务逻辑上的风险和不确定性？是否所有项目相关人员都是技术出身，能够读懂代码？例如：搭建 Web 框架的项目、数据库平台的项目、查询系统的项目、或是云部署平台的项目。众多开源项目都可以归为这几类。如果是这样，那么偏技术的 TDD 就比较适

用了——我会用单元测试工具来驱动业务场景和技术设计。我可能还会在白板上写些例子以求得到普遍认同（BDD 的核心概念），但我不会浪费时间把这些例子录入到可执行用例或非技术自动工具（Cucumber）中了。

项目是否也有复杂的业务逻辑需要讨论，或者存在不清晰的业务需求，需要在不懂编程语言的成员之间沟通的情况？如果是，我将各个击破。先使用实例来探讨业务需求，确保大家对业务有共同认识。然后为这些实例建立规格说明书，并将这些实例录入到诸如 Cucumber、FitNesse 或类似的自动工具中去。最后，用单元级别的测试来驱动代码的设计。

Ron：我多希望在过去的半个世纪的开发生涯中，我能够在每个项目中都运用这两种方法。我并没有发明这两种方法，我只是最早接触的那几个人之一。TDD 和 BDD 让我确信，没有更好的方法了，这两种方法也让我的代码缺陷数量大大降低。并且，当我更好地了解到系统或产品该设计成什么样子时，我的设计就更得心应手了。

TDD 和 BDD 太难被超越了。

Steve：你需要为你的系统写测试吗？如果需要，为何不在代码实现前就写完呢？这样一来，你就能知道怎么把测试变得更简单。你也并不可能把所有的测试用例一下子都写完，何不边写代码边完成测试用例呢？

InfoQ：现在似乎有种普遍的认识，认为 TDD 等同于单元测试，而 BDD 则等同于验收测试（无论使用的是哪种工具）。你们认为这种说法是否正确？其他类型的测试例如构件测试与系统集成测试 又是如何与 TDD/BDD 关联的呢？

“JB：我想从两个方面谈这个问题：TDD 对我的设计提供了反馈，而 BDD 则是对我们开发的产品理解提供了反馈。起初，我以 TDD 作为一种测试技术开始实施，慢慢地我把 TDD 作为设计手段，再后来把 TDD 作为一种深入了解产品设计原理的方式。而对

于 BDD，我则一开始把它作为提醒我从真正终端客户和利益受益者的角度看待产品的一种方法，最终变成改进业务与技术人员之间沟通的一种技巧。你可以看到，虽然测试在其中扮演着配角，却举足轻重。

我并不把 TDD 和 BDD 当作测试技术，我认为测试策略与 TDD/BDD 是无关的。无论项目使用 TDD，还是 BDD，还是什么都不用，我都期望从微测试（microtesting）、系统测试和可用性测试的角度来考虑这个问题。

Dan：这真的是不幸的历史产物。事实上，TDD 和 BDD 都涵盖这两种测试，甚至更多。Kent Beck 在极限编程（此后在他的 TDD 书中）中说到——TDD 在不同等级粒度中均能适用，这也契合 Nat Pryce 和 Steve Freeman 在《成长性面向对象软件》所描述的。你撰写用户级别的功能测试与低级别的单元测试的目的是相同的，都是为了阐述你希望代码如何表现。TDD 的激励作用多过于实际意义：如果你只为了得到更多的自动测试覆盖率而撰写自动测试用例，那不是 TDD。类似地，你可以测试诸如并发、延迟、failover 或吞吐量等非功能性需求。

BDD 之中用户级别测试与代码级别测试的区别更加明显。用户级别的测试是基于自我澄清和自动化的场景。并且这些构建的步骤可以在其他场景中重用。代码级别的测试也叫做实例（或者规格 Spec，虽然我不倾向于这么叫），与 TDD 相比更接近人的思考方式。时过境迁，有不少不同类别的工具出现。比较有名的有：用户级别的跨语言工具 Cucumber，以及代码级别的工具 RSpec、NSpec 等。我的经验是，我倾向于使用团队喜欢的工具。比如，大部分的 BDD 源码，我用过时的 JUnit 与 JMock 的 Hamcrest matcher library 编写。最近，我用 Python 的 py.test 和 nodeunit 来写 node.js，这与 JUnit 风格类似，“BDD 风格”的框架在两者中均有体现。因此我的建议是，这只是代码，如何实现它由你决定。

Gojko：同样，这取决于你对 TDD，BDD 以及其他概念的定义。我对 Kent Beck 著作的理解是用户测试与单元测试属于 TDD 范畴。而我对 Nat Pryce 和 Steve Freeman 的《成长性面向对象软件》中的理解是 TDD 包括系统测试，构件测试以及单元测试。我对规格说明书的解释是好的文档会将业务概念拆分，并自动化，以求风险得到控制——如果大部分的风险来自于一个实例，我们则需要验证实例的 Java 方法；如果风险来自系统，我们则需要对系统进行 30 次的 Web 服务运行和 100 次数据库运行。

Ron：BDD 起源于 TDD 的另一种描述。而现在，在 Chris Matts 和 Liz Keogh 手中，BDD 演化成实现产品特征描述与验收测试的方法，这也是我所理解的 BDD 描述。

而其他形式的测试，当然也很重要。我特别指出，我们需要把用户体验测试加入到你的测试列表中。关于构件测试和系统集成测试，敏捷项目的最佳方式是使用持续集成（Continuous Integration）。这样，所有分散的构件就能关联起来，集成的系统所能承受的测试也随之增加。通常，这些分类很模糊。我们针对不同事件，在不同的间隔运行不同的测试集，比如：新的类库或构件，或新构建的版本。这样的测试集包含了用以描述单元测试、验收测试等所有的测试用例。 测试的精髓在于，分清什么需要测试，尽量在创建或有变化的时候就进行测试。这就是我们防止代码缺陷和及时发现代码缺陷的秘诀。

Steve：我可不会做这样的类比，很明显，这是基于对 TDD 的错误理解。TDD 中最基本的问题是“我怎么知道这样是可行的？”——所有的业务和组织都可以这样去考虑问题。

我并不认为将测试细分成筒仓（silo）会有多大帮助，相反顺畅的测试给团队带来自信，相信系统能够正常运行。

InfoQ：一直以来，TDD 被公认为是一种（代码）设计准则、测试准则或沟通工具。然而，在 TDD 方法中，作为这些准则和工具的目标会如何影响设计？在测试方面，TDD 现在与未来的价值又是怎样的？

JB：我认为这很大程度上取决于方法的实践者。当我实践 TDD 的时候，发现最有价值的部分是测试规范这块，这可能是因为我期望在这个方面有所提高。而只有当我的错误数大幅度地减少时，我才注意到 TDD 是如何帮助和指导自己改进设计。在你实践 TDD 的时候，所有这些都指向对测试的当前价值的个人感受：你可能期望从其他测试中获益。

我觉得测试的当前价值远比未来价值要高得多。虽然从未这样试过，但我曾经打算在几个月后把测试都扔掉，仅当我需要改进某些东西的时候才去重写测试。

我从未在不是我编写的测试上获益，我也不认为这会对我所工作的项目有什么样的帮助，或是对 TDD 实践者的基本规范有什么样的贡献。对此我有所顾虑，就好象合同工走进要装修的房间，然后对之前的装修出言不逊。

我曾经宣称， TDD 风格的测试会起到变更探测器的作用（引用 Cem Kaner 的术语），用来减少代码变更的代价。我也听到过有人像我这么宣称。尽管没有仔细地度量过，但的确见识过 TDD 所带来的益处。我甚至听说有人宣称，这些测试可以为从未了解系统和 API 的人，讲述清楚其中的内容。对我而言，这些益处仍旧是理想化和理论上的。

Dan：TDD 是一种设计规范，每样东西都有两面性。在 “测试驱动”这个词组中，“测试”这个词很不幸。 你所写的用来描述行为的实例并不是测试，你所写的代码也只是简单的实例。这些实例只有当与类似持续集成等实践联系在一起时，才成为一套的回归测试。但这些无法代替测试的需求，特别是代替类似 Brian Marick 和 James Marcus Bach 所倡导的有技巧的、直接的探索性测试。TDD 测试的另一个特性就是它

的决定性。在回归测试中，这是一个优势，但在发现阴暗角落方面（译注：指不易发现或重现的 Bug/Defect）做得却不怎样。随机化的测试技巧能够帮复杂的系统找出许多细微之处，然后你就可以利用 TDD 逐个解决。Haskell 和 Scala 的 QuickCheck 工具就是个很好的例子。

关于沟通，这是 TDD 的主要目的之一。特别是在你需要向其他程序员讲述你的代码意向的时候。在文章《Introducing BDD》中，我描述了有含义的命名方式对测试起到了多么大的帮助。否则，你就无法得知你的测试用例失败时所揭示的真相。你必须能够像读故事一样地去理解 TDD 测试用例，而出色的测试命名则决定了功能文档的易用程度。

Gojko：我认为答案总是位于这些因素的平衡点上。为了将 TDD 作为一种规范，我们必须找到一种方法来完成所有事情。好的单元测试，能够指引设计。但也必须能够帮助我们缓解关键技术难题带来的风险，并且告诉人们设计的代码应该怎样表现。

Ron：TDD 会用到测试，但不仅仅是测试而已。它是我们开发系统的方式，是所有测试和程序的骨架。TDD 以及其他相关实践让我们逐步地、一个特征接一个特征地开发系统。从始至终，它保证了代码的活性，以及可塑性。这让我们更清楚地了解我们究竟完成了什么，也让产品负责人或管理者清楚下一步要做什么，不论是通过揭露代码满是缺陷，或者设计是错误的，或者我们不能太快地改进。这也极大地减少了在项目最后阶段才了解到坏消息的可能性。

我不认为这些目标是“独立的”。好的软件开发，需要整合很多想法，也需要我们平衡很多目标。我们并不想放弃这些，相反，我们希望能够找到一种方法服务于所有的目标。而这一切，让开发产品变得更快捷。

Steve：当发觉沟通决定着其他方面的结果时，我必须强调所有级别测试中的沟通因素。例如：如果我致力于把一个测试用例写得可读性很高，这真的也能帮我发现对象引用的不恰当。 我见过很多团队陷入过维护性很差的测试用例的泥塘，而从拖累了整个进程。特别是在新的理解或概念出现的时候，你必须像对待生产代码一样（甚至超过）谨慎地对待测试代码。

InfoQ：关于单元测试、集成测试以及验收测试比率的自动化以及相应的维护成本，有哪些准则可以告诉大家？

JB：在团队实践 TDD 一到两年的时候，我不停地接到团队的消息，向我倾诉测试成本与收益的不平衡。每次这种情况发生，往往是因为团队尝试用较大的测试集（集成测试，系统测试以及端到端测试）去检验较小的事情（独立对象的具体行为）。这往往会导致更大的测试套件，更频繁地运行测试套件（一个失败意味着 23 个测试失败），也会降低程序员维护测试用例的信心和兴趣。这样的测试，反而会给项目带来负面影响。

在这种情况下，我建议为微行为编写微测试，然后合并微测试。契约式测试是通过连接相邻层（adjoining layer）的接口，来检查这个层，不会再深入。这就意味着从集成测试及系统测试转移到检查我所指的“基本对错”上——例如：在无限的时间与空间的条件下，这个对象是否能得到正确的计算结果？我所说的“集成测试是骗局”就是这个意思。

尽管对于不同的项目，不同的团队，要具体事件具体分析，但我还是强烈地建议程序员们从集成测试和系统测试转向微测试。

Dan：我不认为对于比率的建议会行之有效，对我而言，这甚至会有风险。如果某个错误出现的可能性较高，或某个错误的影响较大，我会花更多的时间去解决。举例说

明，我曾希望以测试驱动的方式编写转换数据的代码，因为我知道把转换数据搞糟是多么容易，而发现错误又是多么的困难。类似地，如果我在为系统的外围交互模块写代码，我会非常非常小心需要传送和接受的数据。一旦发现 Bug，我就会编写一段测试用例来隔开这个 Bug，并且用测试驱动的方式去修复它。其他时候，我会用 REPL（一种命令行接口的语言）来实践，并找出 Bug。

Gojko：我想这个问题过于宽泛，没有一个具体的项目，我没法给你答案。

Ron：比起“把这些事情做起来”，我并没有更好的答案。运用 TDD 及相关的实践方法进行编码，花费的成本会更少，结果也会更好。有些人或团队认为，如果他们运用 TDD，搭上的时间会更多，这也是解释得通。也许会存在一些真实的开发情况，但我并没有找到。通常来说，这些人比较简单，也并不喜欢 TDD。这样做的后果是，他们认为自己会很快速，但得到的只是不停增长的缺陷数。这些缺陷必须消除，却使设计变成恶梦，反过来又增加了缺陷，使缺陷不易被发现和修复，从而拖慢整个进程，让进程变得异常困难，造成恶性循环。往往在项目最后的几周，他们最后只能收到坏消息。

这就是“死亡行军”（译注：越做错越多，越无法收拾）。当然，有些人或产品也能侥幸“活”下来。但遗憾的是，花如此大的时间和精力“活”下来，会让团队以为，所有项目都必须这样完成。（译注：感触颇深，同时有过 3 个项目，我带 2 个，另一个 PM 带一个，我的项目成员几乎不用在 UAT 前加班，氛围也非常好；另个项目天天加班，士气低落，民不聊生）。这才是大错特错！在有更好、更简单的选择的时候，他们几乎将自己逼进绝路。

Steve：我没法给出答案，除非你已经为相同的团队搭建了可辨识的系统。敏捷方法的基本要领就是对应已发生的情况。同样重要的一点是这些比例会随着项目进程而改变。

InfoQ：除了至关重要的系统，现在似乎比较统一的说法是 100%的测试覆盖不能作为一个目标，也是不实际的。你是否认为代码/测试比率能提升测试的注意力和效率吗？

“JB：正相反，如果组织重点关注在这些目标，那么就无形中创造出灾难的、会受到炮轰的“成熟模式”。你懂的：级别 1 表示“我们写测试”；级别 2 表示“我们为所有新代码写测试”；级别 3 表示“我们对系统做 50%的覆盖”；我假设级别 5 表示“我测试故我在”。我认为这是没有意义的事情，我可以做这些，但结果还是交付了垃圾的产品。我觉得这是对我所教和我所崇尚的实践的一种嘲讽。

当我有我自己的“网络瞬间（Network moment）”时，我开始关心起这些事情了——你知道的，“我像个疯子，我再不会接受这些！”我尝试让人们学会有自己的“网络瞬间”，然后给他们建议如何去解决问题。我相信比起目标式的测试覆盖率，这更有效。

Dan：我认为教条式的代码测试比率恰巧有着相反的作用。这意味着所有的代码都是同等重要，具有相同的风险，这样的想法是错误的。相反，我提倡对不同的代码，采用不同关注程度及审查力度的测试。任何企图达到代码统一的测试覆盖率的机会成本都是疯狂的，特别在用户接口测试方面。把时间和精力放到改进需要重点关注的代码的质量上，会更加行之有效，立竿见影。

Gojko：只关注代码覆盖率很可笑。关注在 10%的风险最高的代码比关注 99%可忽略风险的代码，收益要多得多。我认为风险覆盖比起测试覆盖要重要得多。我偏好使用属性构架能力矩阵（Attribute Component Capability Matrix），然后决定什么需要覆盖及怎样去覆盖。（详见 James Whittaker 的《How Google Tests Software》一书——译注：好书一本！）

Ron：测试覆盖率永远都不该成为我们的目标。如果我们的测试很棒，那么我们势必能找到缺陷，这是显而易见的。那缺陷还会在哪里呢？在我们没有测的地方。因此覆盖率并不需要做得美好，只要“够用”就足够好了。我们需要做两件事情：

首先，我们需要不停地提升我们的测试技巧，那样我们测不到的地方就会越来越少。如果我们仅仅做 TDD 的教条是没有意义的——“为得到失败而写测试（译注：为了找到错误而拼命地写更多的测试）”。我们会自然而然地得到完整的代码覆盖，以及很好的路径覆盖。

第二，我强烈建议团队分析测试覆盖和这类信息，这样才能更好地决定什么需要改进。人无完人，但我们必须警觉，如果发现了缺陷，那么我们就需要回顾所发生的情况，补充漏掉的测试用例，保证将来不再发生。

Steve：我还是认为，在数据和划分不清楚之前，这是评价会有失精准。代码覆盖率有用，但作为外部的、过分强调的目标，也会影响团队应有的关注程度。

InfoQ：TDD、BDD、验收测试驱动开发（ATDD），测试优先等等这些，对不同的人都意味着不同的东西，也容易让人搞浑。我们是否需要一种通用的语言，用以描述我们的软件开发方法并培养一种可达成共识的内容驱动的好实践呢？

JB：不，我不这样认为。我想对每个实践，我们都有足够多的术语来描述。我发现，当我停止担心如何去定义它们、去分享我是如何理解、并鼓励人们分享他们的理解的时候，反而会有意想不到的收获。我还记得，我最有影响力的一次关于 TDD 和 BDD 的定义、意义和目的的讨论，发生在临晨 4 点半的一家旅馆的房间里。那次讨论，我言辞激励地与人争论 Dan North 和 Chris Matts。如果我们只是推动大家在社区网站

上发布某些定义集，而抵制形式上更生动的、有点疯狂的、激烈的辩论，这将是非常遗憾的事情。

Dan：这只是我们对这块领域发展的理解的一个征兆。我起初建议把 BDD 用作辅助传授 TDD 的目的。我喜欢 Gojko Adzic 的名言“将规格说明书实例化”，只因它很清晰，没有歧义。很长一段时间中，我在“测试”、“实例”和“规格说明书”这些专用词汇中挣扎，我无法做出我的选择。有个短语“通用语言”本身就是有误导。“通用”指的是在边界之内的世界。换句话说，我们依据内容对同样的事物做出不同的描述。某人的测试是另一个实例，也是另一个规格说明书。这关乎你是否能将你的意图很好地传达到你的内容中。

Gojko：我尝试用实例来定义规格说明书，将其变得更清晰、有内容边界并把范围缩小，就是为了避免 TDD、BDD、ATDD 或其他概念的混乱。我认为这个命名对实践的探索很有意义；对我们从业务角度确定要开发的系统并运用实例和搭建在线文档作为支持，也很有意义。某些建议和实践对此是有用的，有别于当我们用技术测试来驱动设计时所用的那些实践与建议，他们有自身的价值。

我不喜欢 ATDD 或验收 TDD 这个命名，因为这给人们一种错误的印象，让大家觉得是因为关注在错误的事情上所以才造成失败。我希望人们不要这样思考，但不幸的是近期出版的书籍已经使用了这个名称。

我理解的 BDD 是，包括了很多不仅仅是规格的实例，并以单元测试驱动技术设计。例如，我考虑特征注入（Feature Injection）、拉动需求、外围设计、定义商业价值的模式以及诸如此类的 BDD 事件。这些才是 Liz Keogh 所做的激动人心的事情。当然，除了这些实例化的规格说明书或单元测试，还是有很多其他事情可以做。例如，效果映射（Effect Mapping）是又一项令人激动的、全新的计划与路演的技术。这与整个

BDD 的系统价值完美地符合，将测试驱动拔高到业务对象的层面，并对任何形式的自动化都没有关联，也无需去做。

Ron：好吧，我认为这是人类沟通的方式。这个世界上并不存在一种每个人听到都能理解的、清晰的语言。在我们边学边成长的商业世界，差异是不可避免的。在我眼中，最重要的差异是人们花很少或不花力气去理解这是些什么东西。相反，他们要么没有理解就开始诋毁这些建议；要么就是没有真正地去运用这些建议或没有理解地去运用这些建议，只是宣称他们在做这些。

这样做有两个严重的后果：第一，许多项目和个人不去做他们可以做的。这将导致人员对工作的不满、失败的项目以及糟糕的结果。第二，误解通常会减缓大家对这些好建议的运用。

Steve：当我们拥有完美的准则时，就能把它归纳成术语 我认为现在讨论什么技术该运用在什么地方还为时过早。我还认为不同的“领域“该有更宽泛的范围，用来获知他们的不同，但不至于弄个底朝天。

InfoQ：关于这些主题有什么最后寄语？

“JB：没什么特别的。多多地实践那些技术吧，因为你希望它们能指导你改进工作。去做吧，因为这能给你“个人成功”。去做吧，只因它能帮助你更好地享受你的工作。总之，无论什么原因，去做吧！

Dan：我在这里讲的每个主题，今后我将分享更多我的感受。

Ron：尽管我会在项目中运用其他方法，但在半个世纪里，我所用的所有方法中，这些是我见过最好的方法，虽然我不会强迫每个人都去使用。

然而，我将做的是，我会建议关心这方面专业的每个人都能学习如何去运用这些技术……到一个“很好的程度”，然后拥有能够决定何时、何处使用它们的能力。在能够对某种技术进行客观的评测前，你不该去回避这种技术。

因此我将要做的是，展示给大家我所要做的软件开发，让大家放心、大胆地去尝试这些技术。并且能够让人们对此类技术有较好的口碑，愿意花足够长的时间去做出一个好的决定。

对我而言，好的决定是指能够在正确的时间正确地使用这些实践。我希望大家能发现这些技术的价值，并从中获益。

Steve：较常见的是，我所看到的关于 TDD 的主要问题，都不是测试难题，而是基本设计技巧的缺憾；人们之所以对测试比较纠结，是因为代码有着错误的结构。类似地，我看到过代码不能表现清楚。我越来越觉得，程序员的面试应该包括一个测试预选者编写一段可读性强的段落的测验。

关于座谈会成员

J. B. Rainsberger 帮助许多软件公司更好地服务他们的客户，提升服务的质量（jbrains.ca）。长久以来，他学习如何写有价值的软件，克服众多自身的不足，最终找到了自己想要的生活。他的足迹遍布全球，边走边分享他的所学，希望借此帮助别人也能得到他们想要的生活与工作。尽管最近两年他经常在欧洲出差，他和他的夫人 Sarah 及三只猫，住在加拿大的亚特兰大。J.B 的博客是 The Code Whisperer。

Dan North 自己编写软件并以敏捷及精益的方法影响着自己的组织。他崇尚以人为本，编写简单、实用的软件。他认为团队面临的主要问题就是沟通，这就是为何他强调“用对文字”，以及他为何如此奉行 BDD，沟通以及人们如何学习。自从 1991 年开始，他就在 IT 行业工作，偶尔他会在 dannorth.net 写自己的博客。

Gojko Adzic 是有战略性眼光的软件交付顾问，为致力于改进软件产品与流程的质量的，有野心的团队服务。他在敏捷与精益质量改进领域有着专长，特别是敏捷测试，以及实例化的规格和行为驱动开发。Gojko 是领先的软件开发和测试协商会的积极分子，他还运营着 UK agile testing user group。在过去的 11 年中，他曾做过程序员、架构师、技术总监及顾问。他做的项目涉及的领域有金融与能源交易平台、移动平台、电子商务应用，在线游戏及复杂的配置管理系统。他是《Specification by Example》、《Bridging the Communication Gap》、《Test Driven .NET Development with FitNesse》和《The Secret Ninja Cucumber Scrolls》的作者。

Ron Jeffries 是极限编程和敏捷方法（XProgramming.com）的独立顾问。他有着非常丰富的软件开发经验，甚至比某些人的岁数都大。Ron 曾经是最初极限编程项目的现场教练，也是《Extreme Programming Adventures in C#》和《Extreme Programming Installed》的作者以及《 Object Mentor's popular XP Immersion course》的合著者。

Steve Freeman《成长性面向对象软件，测试引导》（Addison-Wesley 出版社）一书的作者，英国敏捷软件开发的先驱。他曾为不同的机构开发过软件，从小型外包商到跨国投资银行。Steve 为全世界的开发团队做培训和顾问。此前，他曾服务于研究所和软件院，并获得剑桥博士学位。他为 IBM 写过 shrink-wrap 应用，并在伦敦大学学院任教。Steve 是国际工业协会的演讲者和组织者，第一届伦敦 XpDay 的主席。

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

相关内容：

- Wikispeed——用敏捷带来惊喜
- Dan North 谈误导的艺术
- IT 领域对技术的重视超过了思考

特别专题 | Topic

采访和书评：Google 如何做测试

作者 Craig Smith **译者** 公直 元逍

《Google 如何做测试》一书由 James Whittaker, Jason Arbon 和 Jeff Carollo 三位作者合著而成，正如其封面上描述的那样，看起来充满了知识性和趣味性，在其背后则揭秘了大型技术公司 Google，是如何应对和处理软件测试的复杂性的。

此书概述了 Google 测试软件的方法，花费几个章节说明了测试的两个角色，即测试开发工程师（Software Engineer in Test）和测试工程师（Test Engineer），后面还有一个章节专门讲述了测试经理的职责。整书也包含许多 Google 工程师的访谈，并在最后一章着重介绍了关于对 Google 未来测试方向的一些想法。Alberto Savoia（Google 测试总监）在本书的前言部分给了很好的介绍，

互联网的出现急剧地改变了许多软件设计、开发和发布的模式。许多曾经红极一时的测试书籍里提及的最佳测试实践，在当前的环境下，效率会大大地下降，甚至毫无效率，在一些极端的情况下甚至会适得其反。《Google 如何做测试》这本书会向你揭示出，这个世界上最成功增长速度最快的互联网公司之一是如何应对 21 世纪软件测试的独特挑战的。

此书的介绍部分还讲述了一些有趣的背景，包括那时 Google 为何要去改变其测试现状以及他们是怎样做的。所有在测试领域工作的人或者做测试管理的人，都需要去读一下这本书，正如 Patrick Copeland （Google 高级工程总监）在其前言部分说的那样，

如果想要去改变 Google 的测试，就要改变身为测试人员的意义所在。一个团队可以写出高质量软件的唯一方法是整个团队都对质量负责。我认为，最好的办法是让测试人员有能力把测试作为产品的真实的功能写到代码库里，测试作为产品的一个功能应该和真实用户可以看到的其他功能一模一样。而完成这个功能的技能要求和对开发人员的技能要求也几乎是一样的。开发工程师们似乎也被他们必须在测试上投入更大的精力这件事吓着了，反抗说“这是测试人员的事情”。在测试人员中，大家的态度也很消极，因为许多人不愿意走出舒适区，维持现状的惯性很大，改变很难。

Google 解决这个问题的关键思路是，不要在团队中配备数量众多的测试人员，特别是当你要考虑到 Google 全线产品线时，包括网络应用、搜索引擎、操作系统、移动、社交和企业解决方案。对于这些产品，产品的高质量是它们成功的关键。

不要再把开发和测试分成不同的学科，测试和开发需要齐头并进。写完一些代码后，立刻对这些代码做一些测试，然后再写更多的代码，再做更多的测试。测试不是独立的实践活动，它是开发的一部分，是和开发并行的过程。测试也不等同于质量，质量是当你把开发和测试同时扔到搅拌器中搅拌直到他们分不清彼此的时候才能达成。

作者在“测试开发工程师（SET）”的一章中，阐述了这个角色创建和维护的框架与环境。在这一章里，揭露了大量技术和测试的细节，包括 Google 一些有趣的背景，测试执行过程、持续集成和测试规模的定义，还包含一些关于 Google 测试认证计划的缘由，这个测试认证计划有助于引入开发测试文化，它的另外一个特别的好处是，

这样可以吸引很多优秀测试人员的注意力，并使得他们签署成为测试认证计划中的导师。在一个测试资源稀少的公司文化里，签约这个计划将会使产品团队得到比平时更多的测试资源。

测试工程师（TE）这章是本书的主要部分，讲述了许多测试工程师的实践，包括他们是如何开展测试计划、ACC（Attribute Component Capability）分析、Google 测试分析（Google Test Analytics ）和 James Whittaker 的十分钟测试计划。这章也讲述了一些关于风险，众包（译者注：关于 crowd-sourcing, 众包，参见 http://zh.wikipedia.org/wiki/%E4%BC%97%E5%8C%85），测试用例，缺陷分析和一些他们用来解决测试问题的试验和工具。

很少有学校系统地讲授软件测试，这样对于公司来说，招聘好的测试人员就变得非常有挑战，很好地掌握编码和测试技能的人真的太少了。测试工程师不是独立的个体，他们虽是技术工种，但需要关心用户，从系统和端到端的最终用户的角度去理解产品。对于 Google 和其他在意测试的公司来说，能够雇佣到这样的人将会是个意外的惊喜。

在花费了一章的篇幅描述测试管理的角色及其在流程中的重要性之后，作者在最后一章尝试去讲 Google 未来测试的样子，着重介绍了产品、dogfooding（译者注：见如下链接，http://gz.focus.cn/msgview/20520/182243247.html）和众包。

只要关心的焦点不在产品上，产品就将出现问题，毕竟软件开发的最终目的是构建一个产品，不是编码，不是测试，更不是文档。工程团队中的每个角色都服务于整个产品，角色本身倒是次要的。健康组织的特征是，团队成员会说：“我是为 Chrome 工作的。”，而不是说：“我是一名测试人员。”

尽管此书非常易读且整体易懂，但是此书还是有一个不足的地方，那就是不同的章节由不同的作者撰写而成，而且章节之间的风格明显不同。另外一个令人遗憾的是，三位作者在完成此书之后，先后相继离开了 Google。

整体上来说，这是一本值得所有涉及软件开发的人都应该阅读的一本书，特别是文中讨论的一些测试问题，对于其他公司也来说，无论是做网站、云或者移动终端之上的应用，这

些都是一些通用的问题。尽管文中涉及了比较多的软件工程师（SWE）的事宜，但本书还是一本从软件测试开发工程师和测试工程师角度出发的，以测试为重点的书籍。里面充满了大量测试相关的经验和教训，特别是对那些正在寻求测试方法的测试人员与测试主管来说，将非常有意义。

最近此书的作者们关于此书与 InfoQ 做了一次对话访谈。

InfoQ: 写著本书或与大家分享 Google 测试方法的主要动机是什么？

> 一群在 Google 工程生产力团队（测试人员属于这个团队）的人，一直在讨论如何出一本书。我们已经召开过一次会议并有一个博客，所以写书的需求非常清晰。但讨论如何写一本书总是比真正要去写一本要容易的多，因此我们一再推迟。最终我们三个意识到问题的严重性，并开始去写。有趣的是，当我们快要完成的时候，更多的 Google 工程师也开始对如何参与进来产生了浓厚的兴趣，本书的“Google 工程师访谈”部分就是他们的贡献，许多工程师还成为出版社的官方审核人员。Google 一直在云软件的测试方面处于领先，这本书的出版使得 Google 正式处于领先地位。

InfoQ: 本书重点详细地介绍了工程生产率和软件开发测试工程师这个角色， 这些从事这个角色的工程师工作在各自不同的项目中，主要从事可测试性和测试工具集的工作。你是否认为这是 Google 改善其测试实践的一项重要改变？

> 其实有两个重要的变化。第一，正如你指出的那样，测试角色在其产品项目自身的集中管理之下。这样可以保持测试不被沦为二等公民，在工程生产力形成之前。第二，是测试技术角色的集中管理。对于那些有这良好代码能力的人来说，Google 使得测试成为一项开发任务，它得到开发者的尊重并让他们也参与进来。无论如何，在你离这种模式走太远之前，读一下最后一章，一旦 Google 形成了面向质量的开发文化之后，工程生产力方面的需求就会改变，去创建这种文化也就不再重要。

InfoQ: 整书花费比较大的篇幅介绍不同的测试技术与测试工具（既有公司内部的，也有开源的）。对于一个团队来说，创建自己工具背后的动力是什么？ 对于测试工程师或软件工程师来说，怎样时刻保持对最新测试方法和状态的了解？

“背后的动力非常简单，市场上根本不存在可以满足我们在自动化测试方面需求的工具。Google 唯一全部从外界采纳的“工具”就是众包。我认为获取工具的好途径是通过开源（Google 也一直在支持开源）。商业的测试工具总是滞后半拍，因为其背后没有社区的强大支持。成为开源社区的一员，特别是要了解 Selenium、Web Driver 和 uTest 他们在测试工具方面正在做什么，这也是跟随最新前沿测试状态的最好方法。

InfoQ: 整书很少提及敏捷的概念，尽管许多 Google 的工程方法都基于敏捷原则与实践。Google 是如何看待其工程方法与广大敏捷社区的对比？

“Google 不想成为敏捷社区的一员，也不没有使用 Scrums，Scrum masters 这些类似的术语。我们已经制定了快速变化的内部流程，这也是非常敏捷的流程，不能陷入别人关于什么是敏捷的想法中。当你必须停下来去定义什么是敏捷和你的敏捷是属于哪一种的时候，你就已经变得不再敏捷了。

InfoQ：书中讲到的测试认证项目（Test Certified Program）貌似混合了游戏化和测试成熟度模型。对于这类项目如何激发人们的兴趣而获得成功、如何跟踪最新的测试技术和流程， 你有什么建议吗？

“给开发提建议的风险几乎不亚于评价他们的工作。测试认证两者兼顾，必须要谨慎行事。做成这件事的关键是，首先要有一个正确的模型，例如我们在这本书里所提供的，经过实战、测试和优化的模型，可以作为你的起点。其次，确保用你们最好的测试人员去推广和执行这个项目，因为如果在落地过程中（由于执行不力等原因而）失去开发团队的尊敬，那是我们所不能承受的。

InfoQ：测试工程师可能是最接近传统的、在很多组织中使用的测试分析师的角色，尽管有些人会提出反对意见，说这个角色仍然很技术化（所以和传统的测试分析师还是不同的）。会不会存在提高现有测试人员的技能去适应这个新角色的挑战，尤其是在技术技能方面？

“问题在于规模。两类技能兼备的人才一定有，但当然不是要多少有多少。实际上，雇到愿意学习编码的测试人员，比雇到愿意学习测试的开发人员要容易。

InfoQ：书中提到 Google 正走向“免费测试”的观点：测试的成本接近于零。这对 Google 的测试工程师意味着什么？这个想法离现实有多远？

“这在很多团队是现实，但在达成目标的过程中，会遇到很多阻力。有多少测试人员愿意把自己搞到失业？你的工作可以被自动化/众包，并不意味着 you have the stones to go through with it（你能坚持到底？）。本书的最后一章对这个问题做了比较详细的回答，所以我就不赘述了。我只想说，任何对云/Web 应用以及移动平台做大规模功能测试的人，都是在浪费时间，只会拖慢整个团队。

InfoQ：贯穿本书始终，你建议“不要雇佣太多测试人员”，未来测试工程师的角色在减少。对那些坚称你需要更多的测试人员、明确划分开发和质量保证的界线的组织，你想说什么？

“为什么需要这样一条界线呢？Google 已经证明，当写代码和优化代码之间的界线模糊了之后，结果是开发的速度会更快、潜在的缺陷会更少。雇佣太多的测试人员等于给开发更多的（质量上的）依赖，这对产品非常糟糕。我对于人们过分局限于他们的角色深感烦恼。“我是一名测试人员”是一种不健康的态度，“我是一名开发人员”也一样。只要人们不再过分受限于他们的角色，而是开始以（最终）产品为导向的时

候，奇迹就会发生，每个人都应该聚焦于尽其所能做任何有利于开发出最好的产品的事情。

InfoQ：一些读者可能会认为书中描述的很多方法难以推广，因为他们觉得你们能做到这个的唯一原因是“因为你们是 Google”。对这类说法，你怎么看？

“Google 是怎样成为今天这个样子的？通过更快的、更大规模的编写软件。并不是因为 Google 成就了好的测试，而是因为好的测试成就了 Google。这仍然是 Google 的强项，它可以更快的开发 Internet 规模的产品。

InfoQ：对那些希望从事测试工作的测试分析师或者新的毕业生，如何应对测试角色的不断变化的技能要求，你有什么好的建议？

“把测试当开发。拿到 CS 学位，拿到好成绩。认证和培训只能教给你简单的东西。（自己）学习更难的东西并擅长之。那些投机取巧的测试人员注定只会抱怨被当成二等公民，毫无长进。不想被那样对待？那就去掌握一等的技能。

InfoQ：对那些看了这本书以后“希望象 Google 一样测试”的组织，你有什么建议？

“当规模还小的时候，就建立一个集中的测试组织。招聘拥有一等技术技能的人。复制 Google 所做的全部，使之成为你的软件工程 DNA 的一部分。关注产品，而不是你的角色。不断的思考如何自动化日常重复性的劳动。众包一切可能的工作。

InfoQ：这本书的主要读者是谁？你希望他们在读完本书之后学到哪些重点内容？

> 软件开发相关的任何人、每个人。希望人们理解（通过实践本书所讲的技术）不会使你做到完美，但是可以做的更好。（OR：这本书难以做到完美，但本来可以写的更好）出版商提供了样章供 InfoQ 读者免费获取。

本 Q&A 基于《How Google Tests Software》一书编写。该书作者是 James Whittaker, Jason Arbon 和 Jeff Carollo，由 Pearson/Addison-Wesley Professional 于 2012 年 3 月发行，ISBN 0321803027，copyright 2012 Pearson Education, Inc. 详情参见出版商网站。

关于作者

James Whittaker 是一名技术高管，他的职业跨越学术界、创业公司和顶尖的技术公司。James 是一个富于创新和激情的技术领袖，以其在测试、安全和开发工具等方面的贡献闻名于世。他发表了几十篇论文、五本书，在多个国际会议上获得最佳演说者奖。在供职于 Google 期间，他领导的团队包括 Chrome, Google Maps 和 Google+。他目前在 Microsoft 从事重新发明 Web 的工作。James 的其它著作包括：《How to Break Software》，《How to Break Software Security》(与 Hugh Thompson 合著)，以及《How to Break Web Software》(与 Mike Andrews 合著)。在 Microsoft 期间，James 的很多测试想法都被开发成工具和技术，供开发和测试使用，并且写作了《Exploratory Software Testing》。

Jason Arbon 对于软件质量和分析有很大的热情。Jason 目前是 uTest.com 的工程总监。这是一个众包测试服务公司，他在其中推动软件工具、分析和自动化等方面的创新。之前，Jason 在 Google 做敏捷产品的工作，在 Chrome Browser、Chrome OS、Google Desktop 和 Google+等项目中管理 Web 搜索个性化团队以及工程生产力团队。再之前，Jason 曾有在多个创业团队的经历，也曾供职于 Microsoft，参与了 Exchange Server、BizTalk Server、Windows FileSystem、MSN 和 WindowsCE/IE4 等项目。

Jeff Carollo 目前是 uTest.com 的一名软件工程师，这是一家众包测试服务公司。在此之前，他是 Google 的一名高级软件测试工程师，参与了 Chrome、Chrome OS 和多个服务器端项目，包括 Google VoiceVoIP 平台。Jeff 毕业于 Texas A&M University，获得 CS 学位，他来自 New Orleans，是新奥尔良圣徒队的超级粉。

原文链接：http://www.infoq.com/cn/articles/how-google-tests-software

推荐文章 | Article

Somya Jain 谈从 C#到 Object-C

作者 Somya Jain 译者 陈菲

几年前掌上电脑很大程度上就是一新奇事物和玩具。但如今，商业界正发生着一场重大变化。许多传统上在 XAML，Flex 或 HTML 上编写应用的商业公司突然发现，iPad 应用有着数目巨大的需求。这些需求不止来自游戏，同时也存在很多标价为几十万甚至上百万的项目。尽管.Net 和 Java 在未来几年依然会活跃于后端平台。但是对于那些愿意尝试的团队，iOS 市场还是有很多机会。

我们与 Somya Jain，一个致力于此的开发人员，谈论了如何从 C#过度到 Objective-C。

Somya Jain：关于移动编程我学到的第一点是（通过不断实践总结出来的），它与服务器或桌面开发有着很大的不同。移动设备 RAM 空间小，网速慢，屏幕小；尤其 CPU，相对慢。尽管如此，人们却期待移动设备提供更高的用户体验。人们可以很容易地在应用商店将你的应用与其它应用进行对比。如果你的应用还是停留在作坊级，那就很难具有竞争力。

我学习 Objective-C 就是为了编写 iPhone 及 iPad 应用。因为拥有 C++/Java/C#背景，其学习并非那么困难。大部分概念都能简单互通。一开始，我尝试通过遵循书籍和阅读文档来学习，但是，我发现更快的学习方法是真正投入到项目中。

以下几点是理解 Objective-C 的关键。尽管它与 C#有很多相同之处，但依然有细微的差别。

内存管理

首先需要提的就是内存管理。对于 iOS 开发，理解 alloc，release 和 autorelease 是至关重要的。但是随着 ARC 的引进，它们可能会变得不那么重要。尽管 ARC 会在某些地方误导你，但它将帮助你理解其内含的原理。

内存管理问题，比如过度释放内存，将造成令人厌烦的 BAD_ACCESS 崩溃；还有不释放内存，造成的难以解决的内存泄漏。此外，他们对应用的性能和可用性也有很大影响。学会如何快速定位和解决这些问题非常关键，我建议学会如何使用 NSZombies 对象及性能调试工具 Instruments 来调试这些问题。

另外，Objective-C 支持垃圾回收，但它仅支持 mac osx，而不支持 iOS。

InfoQ：添加和释放对象听起来非常像 COM 引用计数，但“自动释放”又是怎么回事呢？

Somya:从根本上，你会使用自动释放返回一个不是由创建对象管理的对象。通过调用自动释放，将该对象添加到自动释放池中，并在清空池时将对象释放。如果接受对象想要使用它，需要保留其值。以下是更多来自 apple 文档的信息：

每个循环的每次轮询开始，应用工具箱都会在主线程中创建一个自动释放池，在并最后将其释放，以释放执行事件时产生的自动释放对象。

InfoQ：什么是 NSZombies？

Somya：NSZombies 可以用来检测之前提到的内存过度释放问题。在自动释放对象上不平衡释放是常见现象。事实上，最终释放时，一切都工作正常，对象也被重新分配了；但是当清空自动释放池时（有的时候随后才发生），应用会因为尝试去自动释放一个已经被释放的对象而崩溃。正因对象本身不存在，所以很难定位出真正问题所在。正是对应上述问题，提出了 NSZombie 对象。通过设置 NSZombieEnabled 标志将任何已释放对象转化为 NSZombie，以取代原始释放操作。这样我们就可以在任意调用 NSZombie 对象释放或自动释放处设置断点，以定位到问题代码路径。

InfoQ：您能解释下什么是 ARC，为什么您觉得它可使内存管理更简单？

Somya：原则上，通过允许使用 ARC，编译器会自动为你保留或释放请求。这意味了我们不必再去担心内存管理了，但是经验告诉我：事实正与之相反。我还没使用过 ARC，所以无法做过多评论。

方法调用 VS 消息

两个语言间另外一个细微差别是：C#中方法调用是静态的，绑定发生在编译时(暂时忽略动态编译）。Objective-C 消息则在运行时动态绑定对象，而非编译时。这意味着你可以发送任意定义好的消息给对象，编译器都不会报错。错误会在运行时抛出。

有意思的是，其中有一机制：就算消息没有被类执行，也会被截取。这通常发生在多继承，日志和往动态载入代码分配消息等高级方案中。

进程池 VS 中央调度

从高层来说，它们的定义是一样的：一个用于管理进程池的系统。对于该点，我认为 iOS 确实做得很出色。首先，它支持不同优先等级，甚至进程池也赋予我们更优的授权管理。比如我们可以在下载图片时操纵更高等级的进程，以使后台下载不影响我们的 UI。

另一优秀特征是分配组允许同时提交所有任务，但只在全部完成时才通知。

以下概念在 C#和 Objetive-C 中几乎一一对应：

- 扩展方式（C#）<==>类别（iOS）
- 接口（C#）<==>协议（iOS）
- 委托（C#）<==>块（iOS）

C#中我怀念的：

1. 泛型
2. LINQ

3. 不用创建头文件，将代码置于同一文件中。

给 C#开发者的其它建议：

- 从 Resharper（C#）开发者那了解 AppCode（Objetive-C）
- Xamarin 的 MonoTouch 团队将 C#引入到 iOS 设备中。

InfoQ：对 iOS 开发，您偏向哪种 IDE 呢？

Somya：当然是 AppCode 了，我非常熟悉它所带来的代码自动完成和重构支持，就如同 C# 中使用过的一样。我还喜欢它能够突出显示那些潜在的内存管理问题，对于 iOS 的初学者非常有帮助。一天下来，那大大提高了我的生产力，简直不费吹灰之力。

InfoQ：您说您想念泛型。那 Objective-C 提供类似工具吗，还是必须象.Net1.0 那样必须使用非类型化集合？

Somya：是的，我们使用类似 1.0 中非类型化集合。但是由于动态消息绑定，我们无需停止对象去发送消息。

InfoQ：Objective-C 是否有哪些特征您希望 C#能采纳呢？

Somya：在语言层上应该没什么。我还不曾发现有哪些显著特性让我觉得 C#也应该拥有。但是在 API 层上，我觉得 apple 确实在 iOS 上下了很好的功夫。尽管其 API 看上去有点严格，但是确实设计地很好。比如在 NavigationController 和 UITabBarController 里，通过简单标志就可在视图切换时实现动画效果。而在 android 和 WP7 中，这要消耗很大力气。

InfoQ：如何比较在 iOS 中，和在 WinForms 或 Silverlight 中创建用户界面呢？

Apple 有两个主要 UI 库：UIKit 和 GLKit。GLKit 主要关注基于 OpenGL 的应用和游戏。UIKit 更接近于 WinForms/WPF，它有类似 text boxes 文本框，labels 标签，drop downs 下拉列表等标准控件。这些 UIViews（.Net 中的控件）易拓展成或组合成试图，

类似于.Net 中创建自定义控件和用户控件。但是，iOS 只支持固定的基于坐标的布局，所以我们必须明确具体大小及每个视图在其父试图中的位置。其他 UI 定义就比较相似，比如：

- 依然有视图结构
- 当往后台进程导入数据时，依然要小心；并将其整合到主进程中去。
- 需了解视图生命周期等。

其实我最怀念的是对数据绑定的支持。

关于受访者

Somya Jain 有着 11 年的企业级软件开发员、技术领导及架构师的经验，他现在住在纽约，是一位移动自由开发者（包括 iOS， Android 以及 Windows Phone）。他主要关注于跨平台移动应用。

原文链接：http://www.infoq.com/cn/articles/CSharp-To-Objective-C

相关内容：

- 前 Sun 开发人员为 Android，iOS 等其他移动平台提供 JAVA 的 WORA 支持
- 前行中的移动互联网
- PhoneGap 2.0 发布

推荐文章 | Article

Restful Objects 简介

作者 Dan Haywood, Richard Pawson 译者 蔡坚安

Restful Objects 是关于领域对象模型的超媒体 API 的公共规范。该规范的 1.0.0 版本刚刚发布并提供下载，并且目前已经出现了两个实现了该规范的开源框架——一个基于 Java 平台，另一个基于.NET 平台。

Restful Objects 和其他 RESTful 标准有什么区别呢——比如说和 Java 平台的 JAX-RS (JSR311)相比较或者和.NET 平台的微软 Web API 相比较？答案是其他 RESTful 框架主要用来抽象纯粹的网络问题，它们并不保证所定义的和各种领域对象类型交互的资源结构的统一性。并且它们没有或者很少支持 RESTful 体系中被认为最重要的原则：“将超文本作为应用程序状态的引擎”，或者说 HATEOAS。平白地说这意味着访问系统的所有功能是可行的，只要跟随着来自资源主站的超媒体控制（链接）就行了。和 Restful Objects 意图最相近的框架或许就是 OData。但两者间还是存在很大差别的，Odata 主要专注于 CRUD（Create, Read, Update, Delete）功能，而 Restful Objects 提供了访问对象所有行为（方法）的能力。

实现了 Restful Objects 规范的新框架不仅使得在领域对象模型上编写符合 HATEOAS（超媒体即应用状态引擎）标准的 API 更加容易——事实上它们消除了这些工作量！你可以用简单的词法编写领域对象模型——按传统的 Java 或 C#对象编写——然后在几分钟之内在其上创建完整的 RESTful API。

另外，因为这些框架提供的资源和表述（representations）符合公共规范，这意味着所编写的客户端如果能和运行其中一种框架的服务器交互，那么就可以和运行另一种框架的服务器交互。我们希望在接下来的几个月里能看到更多符合 Restful Objects 规范的服务端实现（我们已经知道至少有一个第三方已计划这么做）。同时，我们也已看到使用 Restful Objects API 的标准客户端框架逐渐涌现。这些我们后面再谈。

可应用性和优势

Restful Objects 规范最有可能的受众是采用领域驱动的开发者，已经实现或者正在实现领域对象模型的开发者，以及那些想在该模型上提供 RESTful API 之类特性的人员。我们自己的领域驱动开发（DDD）经验包括了非常大范围的领域对象模型设计 [1, 4, 5]，包括了被封装为领域实体上的方法的大多数行为。我们希望能使用 Restful Objects 来更深入地挖掘领域模型资产。

除了能巨大地减少工作量，使用实现了 Restful Objects 规范的框架还有另外几项优势：

- **测试**。所有业务逻辑都只在领域类型中表示，因此可以在内存中快速而经济地将它们进行单元测试。相对的，如果 RESTful API 是根据特定用例自定义编写的，则只有通过跨服务器的（缓慢的）集成测试才能有效地将其进行验证。
- **文档**。Restful Objects 是由领域对象模型所驱动的，因此可以使用已有的成熟技术（例如 UML 或者简单的 Javadoc/Sandcastle）将其文档化。这比创建一些新的标记以显示地文档化手工编写的 RESTful API 更加可取。

在继续深入之前，我们想说 Restful Objects 是和领域对象模型的语法不相关的。所以，不管你是认为领域对象应该表示具有业务逻辑的实体，还是认为应该有单独的“资源模型”以表示用例实体，在这两种情况下 Restful Objects 规范都同样适用。为了简单明了，在本文中我们会优先使用和前者（领域对象作为实体）相关的例子，但在本文的后面我们还会继续讨论这个主题。

资源

在 Restful Objects 规范中每个领域对象就是一项资源；例如下面的 URI 指向 Id 为 31 的客户：

~/objects/customers/31

其中 ~ 是服务器地址（比如 http://myserver.com），customers 定义了对象的类型，而 31 则是该类型的“实体标识”。该规范并没有定义标识的格式，任意的唯一字符串都是可以的。为什么需要/objects 呢？这是为了和其他顶级资源区分开了，包括 services（例如 repositories 和 factories 之类具有方法但没有属性的单件对象）以及 domain-types（类型元数据）；这些顶级资源都具有各自的子资源结构。

每个对象资源都具有一些子资源以表示该对象的成员。所以：

~/objects/customers/31/properties/dateOfBirth

指向一个特定客户的生日，而：

~/objects/customers/31/collections/orders

指向和该客户相关联的订单的集合。下面的表格显示了和各种 HTTP 方法相对应的对象、属性以及集合资源。如果不能赋予某个 HTTP 方法有效的意义，则指定为 405 错误。

	对象 资源	对象 属性资源	对象 集合资源
GET	对象概述	属性详细及值	集合详细及内容
PUT	更新或清除多个属性值	更新或清除值	添加对象（如果集合具有 Set 语义）
DELETE	删除对象	清除值	从集合中移除对象
POST	n/a - 405	n/a - 405	添加对象（如果集合具有 List 语义）

以上的分析并非什么创新的东西——很多预定的 RESTful API 都具有类似的特性。但 Restful Objects 规范也算开辟了一片新天地，在对象的“action”上应用了同样的方式——“action”是指可以通过 RESTful 接口访问的对象上的方法。扩展些说，在其他设计（例如[2]）中这些问题已有解决方案，通常是将对 action 的调用映射到 HTTP POST 上。我们的目标使我们将提供对象的 action 信息（这需要参数集合）和实际调用 action 区分开来。因此，URL：

~/objects/customers/31/actions/lastOrder

描述了获取 Id 为 31 的客户的最后一条订单的 action，而：

~/objects/customers/31/actions/lastOrder/invoke

描述了调用该 action 的资源。

同时，我们意识到，要求所有 action 都通过 POST 进行调用并不是使用 HTTP 的正确方式：比如幂等 action 或者仅仅用于获取其他对象的查询 action（有时被描述为“没有副作用的（side-effect free）”）还要 POST 吗？下面的表格描述了我们如何应对这些问题：

	对象 Action 资源	对象 Action 调用资源

	（信息）	
GET	action 的详细信息（比如调用的参数和 HTTP 方法）。	调用——如果 action 被认为是只读的（没有副作用）
PUT	n/a - 405	调用——如果 action 被认为是幂等的，但不是只读的。
DELETE	n/a - 405	n/a - 405
POST	n/a - 405	调用被认为是非幂等且非只读的 action

在一些领域模型中，大多数行为（业务逻辑）是通过服务的形式实现的，其中领域实体仅仅是作为数据结构在服务的 action 上双向传递。Restful Objects 规范也适用于这种模式；唯一的小区别是 URL 被用来标识服务实体而不是领域对象实体。与此同时，该规范也适用于在“行为复杂（behaviourally-rich）”的领域对象上以方法实现大多数行为的模式；这种模式下，服务仅仅是作为提供对象访问（查找已存在的对象或者是创建新对象）的次要角色。

表述（Representation）

每个资源返回一个表述；Restful Objects 规范将这些表述定义为 JSON（JavaScript Object Notation）格式。对于使用 PUT 或者 POST 方法访问的资源，在请求体中也必须提供表述，以描述要怎样修改资源。

规范定义了若干个主要表述：

- object（代表任意领域对象或服务）
- list（访问其他对象的链接）

- property
- collection
- action
- action result（一般包含一个 object 或者 list，或者只是反馈信息）

同时，规范还描述了几个次要表述，比如 home 和 user 等。每个表述都具有一个正式媒体类型，其包含在 HTTPContent-Type 头部中。对于领域对象而言，其具有以下格式：

application/json;profile="urn:org.restfulobjects:repr-types/object"; x-ro-domain-type="http://~/domain-types/customer"

正式地，这是具有两个可选参数 profile 和 x-ro-domain-type 的 application/json 类型。这两个参数为客户端提供了额外语义，可以想象成将其中一个层叠在另一个之上。在最底层，application/json 仅仅告诉客户端负载为 JSON 格式。对于某些客户端而言，比如浏览器的开发者插件，这就是所需的所有信息了。在上面一层，profile 参数指明该表述是领域对象形式的。最后，x-ro-domain-type 参数指明对象的类型：在本例中为 Customer。客户端可以使用这个参数来自定义用户界面，或者校验返回的表述是否为所期望的。另外要注意的是，该规范使用.x-ro-前缀以便在没有现行标准或者草案标准可以利用时将命名空间冲突最小化。这种格式必然产生了一些自定义参数和查询字符串；然而 Restful Objects 没有专门自定义 HTTP 头部。

链接

与 HATEOAS 原则一致，每个表述包含了到其他资源的链接，而每个链接具有 rel 参数以定义关系的性质。该规范重用了几个 IANA 定义的 rel 值（例如：self、up、describedby），另外还加上了几个自定义的 rel 值。请看下面的例子：

urn:org.RESTfulobjects:rels/details;action="placeOrder"

定义了一个在对象表述中指向该对象的 action 资源的链接。前缀：

urn:org.restfulobjects:rels/details

对一般客户端而言用其已够用了，而额外的参数 action——在本例中其指明了该链接指向 placeOrder action——也许对定制的客户端有用。

综上所述，下面的表格描述了通过资源集合完成用户目标的典型流程。

描述	方法	URL	主体	返回的表述
跳到主页	GET	http://~/	-	主页
链接到提供的服务列表	GET	http://~/services	-	列表（服务的链接集合）
链接到产品库存服务	GET	http://~/services/ProductRepository/	-	服务
链接到 'Find By Name' action	GET	http://~/services/ProductRepository/actions/FindByName	-	action（在界面上以对话框呈现）
调用（只读）action 并传递参数"cycle"	GET	http://~/services/ProductRepository/actions/FindByName/invoke/?Name=cycle	-	action 结果，包含匹配 Product 对象的链接列表

链接到集合中的一个 Product 对象	GET	http://~/objects/product/8071	-	对象（代表一个产品）
调用对象上的（不带参数） ‘AddToBasket’ action	POST	http://~/objects/product/1234/ actions/AddToBasket/invoke	-	-
调用 BasketService 上的 ‘ViewBasket...’ action	GET	http://~/services/BasketService/ actions/ViewBasketForCurrentUser/ invoke	-	action 结果：包含指向 Item 对象的链接的列表
修改刚新增的 Item 上的 Quantity 属性	PUT	http://~/objects/orderitem/1234/ properties/Quantity	属性名及其值 3	-
从购物篮中删除（之前添加的）一个 Item	DELETE	http://~/objects/orderitem/517023	-	-

服务端实现

前面我们提到两个实现了 Restful Objects 规范的独立开源框架。其中，Restful Objects for .NET 完整实现了规范，但其目前还处于 beta 版本，因为它使用了 Microsoft Web API 框架（ASP.NET MVC4 的一部分——在写本文时 MVC4 正处于‘RC’阶段）。第二个框架 Restful Objects for Isis 运行于 Java 平台之上；该框架已经可以使用，但其目前仅实现了 Restful Objects 规范的前期草案，在正式发布前还需要继续开发和测试。我们一直在积极地参与这两个框架的开发。

使用这两个框架，你能够根据领域对象模型分别编写 POCOs 和 POJOs 代码，然后创建完整的符合 Restful Objects 规范的 RESTful API，而不用编写任何其他深入的代码。这个录制的在线视频（使用.NET 框架）演示了上述工作如何在仅仅几分钟内就可以完成。

这是可能的，因为这两个框架都是建立在实现了 naked objects 模式——根据领域对象模型利用反射自动创建面向对象的用户界面，并（默认）提供用户活动的公共方法——的现行框架之上的。新的 Restful Objects 框架以相似的方式反射领域对象模型，但以 RESTful API 的形式呈现对象的功能，而不是以用户界面的形式。两个新的框架都将反射、对象持久以及其他横向关注点（cross-cutting concerns）的职责委托给了已有框架（分别为 Naked Objects for .NET 以及 Apache Isis）。

上述的新框架能够识别一些简单的领域对象代码规范以及标示法（在.NET 中为‘attributes’）。例如：对象上的任何公有方法都默认会在 Restful Objects API 中以 action 提供出来，但允许通过将方法标示为 Hidden 以重写。如果某个对象定义了公有方法 foo([params]) 和另一个公有方法 validateFoo([params])，则后者会被认为是用来在前者执行前为传递给前者的参数提供验证逻辑的。

这两个框架还提供了细粒度的基于用户身份和/或角色的授权机制。对于给定的领域类型，如果用户没有被授权查看某个给定的属性、集合或者活动（action），则在相应的表述中指向该对象成员的链接就永远不会呈现给该客户。当用户试图通过直接构造指向该资源的 URL 以进行访问时，他们将接收到 404 错误；而如果用户拥有查看该对象成员的权限，但没有编辑的权限，则当试图进行编辑时就会接收到 403 错误。

Restful Objects for .NET 框架的源代码可以从 Codeplex 网站下载或者以 NuGet 的形式安装；Restful Objects for Isis 框架的源代码可以以源代码的形式下载或者使用 Maven 原型安装。

对于已经使用过 Naked Objects for .NET 或者 Naked Objects for Apache Isis 的开发者，使用新的框架意味着他们能够在几分钟之内（形象地说）在他们已有的领域模型上创建 RESTful API。但我们的意图是将新框架推荐给没有 naked objects 模式知识同时也对其不感兴趣的开发者，事实上我们已经看到这些开发者逐渐感兴趣了。

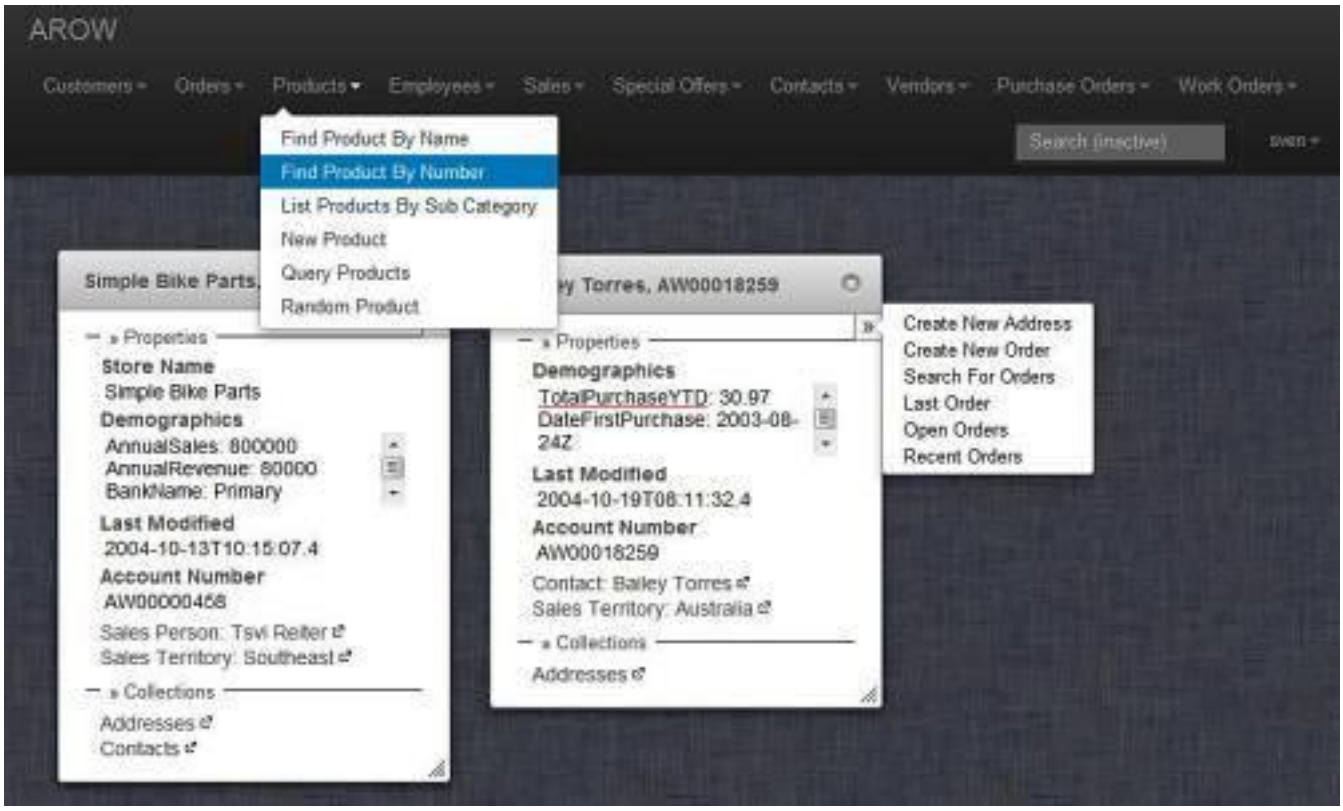
客户端

目前我们大部分开发工作都聚焦在服务器端——根据领域对象模型产生 Restful Objects API 的相关代码。不过与此同时，我们也在使用这些 API 的客户端应用程序上做了一些工作。

在其中一个例子中，客户端是一个规范的 web 应用程序（使用 ASP.NET MVC 编写），该应用中的控制器方法调用了在另一台服务器上实现的 Restful Objects API。我们希望在将来开发一个小的客户端应用程序库，用来调用 Restful Objects 服务端以及将返回的 JSON 表述转换为能够由 C#或者 Java 操纵的对象。

在另外一个例子中，我们编写的客户端是一个‘单页面应用程序’，其只由一个页面组成，仅包含了若干行静态 HTML 代码，主要由使用了 JQuery 的众多 JavaScript 函数支撑。JavaScript 负责调用服务器上的 Restful Objects API，然后将返回的结果以 HTML 在浏览器中呈现。再次，我们希望在将来能编写一个小的 JavaScript 应用程序库，专门用来消费(consuming)Restful Objects API，计划该库主要使用 JQuery 编写——也或许由其他者接棒。

Restful Objects 规范的一般性产生了另一个可能性：存在能在不必修改的情况下通过 RESTful API 和任意领域模型交互的通用客户端。我们已获知有三个不同的开源通用客户端正处于开发当中，它们在 Restful Objects 网站上被罗列出来了。它们都是单页面应用程序，使用已有的程序库和 JavaScript 编写。这些通用客户端中的 JavaScript 代码也能被作为创建一个或多个自定义客户端的基础。这三个通用客户端在用户交互风格上差别很大。下面展示了其中之一的截图——由 Adam Howard 开发的 Restful Objects 工作空间（AROW）：



以牙还牙

通过 Restful API 暴露领域实体的想法受到了很多争议。有些评论者认为这是一个糟糕的想法，认为它产生了安全漏洞；其他一些人甚至认为不可能通过该方式创建真正的 Restful API。现在让我们来详细审视下这些争论。

Rickard Öberg 声称将领域实体作为资源暴露出来“不可能是 HATEOAS 的，因为不存在在资源之间创建链接以暴露应用程序状态的合理方式”[3]。其实显然不是这样的：Restful Objects 将实体作为资源暴露出来是完全 HATEOAS 的，同时对链接来说也是相当合理的。Jim Webber 声称即使这是可能实现的，也还是一个糟糕的创意，因为这导致了客户端和服务端的紧耦合，然而在 Restful 体系中，客户端和服务端是应该能够独立改进的[6]。他以及其他人争论说应该只暴露视图模型和/或表示用例的对象，这两者都应该是版本化的，它们能使客户端不受领域模型变化的影响。

我们认为这些论点即使不是完全错误的，也是不分清红皂白的。我们认为真实的情况和他们所说的有很多微妙的差别：在一些场景下这些论点是有效的，但在大多数场景下则并不是。我们需要考虑到以下两个因素：

第一个因素是：客户端和服务端是否都在同一方的控制之下。对于可公共访问的 Restful API——比如 Twitter 或者 Amazon 的 Restful API——也就意味着服务端和客户端分别属于不同的独立组织，在这种情况下我们同意暴露领域实体不是好的方式。但是，在这种情况下，Restful Objects 也能够完美地跟视图模型和/或用例对象协同，这些模式在规范中有详细的描述。

反过来，事实上在很多潜在的 Restful API 用例中，客户端和服务端的变化发展是由同一方控制的——比如主要在内部使用的企业应用程序。在这些情况下，暴露领域实体不仅安全，而且还是个好创意。这些属于‘主权’系统（此概念由 Alan Cooper 提出）一类的应用程序，相对于面向公众的（或者‘短暂交互的’）应用来说，特别需要赋予用户访问更大范围数据和功能的权利。

第二个因素是：客户端是‘定制的’（专门用于和特定的领域模型协同工作）还是‘通用的’（能在不必修改的前提下和任意领域模型协同工作）。目前，不管是在内部网络中还是在开放的因特网中，几乎所有消费 Restful API 的客户端都是定制的——所以我们需要将它们同领域模型的变化隔离开来。但如果存在能够自动响应领域模型变化的通用客户端，则不需要这样做了。目前，很少有人关注通用客户端的可能性，因为没有综合的标准来指导怎样创建这样的应用程序。我们认为，事实上通用客户端的概念相比定制客户端的概念更加符合 REST 的精神（以及 REST 的语义）。要知道，网页浏览器就是在一定层级上访问 restful 接口的通用客户端，其工作于文档层级之上。而 Restful Objects 使得工作于更高一层抽象（即领域对象模型）的通用客户端的概念成为可能。

把这两个因素描绘成 2x2 的分析表，如下面所示，你可以看到如果你需要应对内部网络应用程序和/或通用客户端，则暴露通过 RESTful API 领域实体是安全和有效的。只有当你需要应对公开的因特网应用程序*且是*定制的客户端时才需要关心以下建议：领域实体应该用视图模型和/或用例对象掩藏起来，以将客户端和服务端的变化隔离开来。

部署环境： 客户端形式：	内部网	因特网
通用	可以暴露领域实体	可以暴露领域实体
定制	可以暴露领域实体	只能暴露版本化的视图模型和/或用例对象

再强调一遍：Restful Objects 在该表格中的任意位置都同样适用。我们认为，目前右下角受到最多关注的事实更多地反映了构建定制的 Restful API 的复杂性，而不是反映了固有的局限性。另一个常见的谬论——认为 Restful API 只能在小的严格定义的状态转换表之上创建——进一步证明了很多人的思维狭隘地集中在右下角的单元格中。

Öberg 进一步争论，因为授权问题，以 RESTful API 暴露领域实体的方式不适用于公共应用程序，并举例：访问重置密码的 action 可以由角色管控，但修改密码的 action 必须限定在某个具体的用户。以上面讨论的两个 Restful Objects 框架实现来说，这个特例对代码的影响事实上是微不足道的。我们认为能阐明他的论点的更好的例子是：访问某个特定的实体对象，而不是某个方法。这是面向公众的系统的常见需求：比如用户必须能够访问他们自己的购物篮，但不能通过猜测 URL 访问任意其他人的购物篮。

虽然目前上述的两个 Restful Objects 实现*本身*都没有支持基于实体的授权，但该问题存在完全可行的解决方案。在前面我们已经谈到，Restful Objects 规范中没有指定 URL 中实体标识符的形式。另外，URL 中实体标识符还可以被服务端加密，比如使用会话产生的

私钥。调用服务的 action 资源以返回我的购物篮，可以使用一个对象表述，其“自身”加密的链接是：

~/objects/baskets/xJDgTljGjyAAOmvBzIci9g

接着，该购物篮上的 Total 属性对应的 URL 是：

~/objects/baskets/xJDgTljGjyAAOmvBzIci9g/properties/Total

所以，在这种情况下仍然可以直接了当地解析资源标识符，但要直接获取另一个购物篮就很可能不行了。这显然不够美观，但值得一提的是 REST *不是*用来创建‘美观’的链接的。就像 Tim Berners-Lee 所强调的，除了服务器地址之外，应该以不透明的形式对待 URL；"rel"的值才是关键。这是 HATEOAS 的其中之一点。

在本文所论述的两个 Restful Objects 框架中，我们计划让用 session-key 加密的实体标识符成为一个可配置项。

结论

目前，在大型企业系统上构建 RESTful API 是一项非常昂贵的活动，从术语角度说，这需要在设计、开发、测试和文档上花费大量资源。而使用实现了 Restful Objects 规范的框架意味着所有工作会变得微不足道，这让开发者能够将精力集中在最需要的地方——领域对象模型。

另外，Restful Objects 规范所激励的资源和表述的标准化为客户端应用程序库开启了巨大机会，这样的程序库可以和任意的 Restful Objects 服务协同工作，若是完全通用的客户端，则可以和任意符合标准的应用交互。

我们希望读者能深究 Restful Objects 规范以及相关框架实现，同时，如果有人有兴致于编写新的符合 Restful Objects 规范的开源框架或程序库，我们将乐于倾听他们的声音。

参考

[1] Haywood, D, "Domain-driven design using Naked Objects", 2009, Pragmatic Bookshelf

[2] Masse, M. "REST API Design Rulebook", 2011, O' Reilly

[3] Öberg, R, "The Domain Model as REST Anti-pattern"

[4] Pawson, R. "Case Study: Large-scale pure OO at the Irish Government", QCon London 2011 presentation

[5] Pawson, R, "Naked Objects", 2004, PhD thesis, Trinity College, Dublin.

[6] Webber, J, "Rest and DDD".

关于作者

Richard Pawson 拥有 35 年的 IT 经验——据说是欧洲最先使用微软的编程语言编写和执行应用程序的先驱之一。除了从事过先进的机器人技术、玩具设计和技术报刊三方面的工作，他在计算机科学技术企业当任了 14 年的研究主管。他 2004 年的博士论文是关于 'Naked Objects' 架构模式的完全指南，也是从那时起他担任了 Naked Objects 项目组的主管。Richard 住在英国的 Henley-on-Thames，他使用 Christopher

Alexander 的模式语言（A Pattern Language）设计了他的房子。

Dan Haywood 是一名自由顾问、开发者、作家和培训老师，专注于领域驱动设计、敏捷开发以及 Java 和.NET 企业架构。他是一位出名的 Naked Objects 倡导者，也是 Apache Isis 项目的主要参与者。另外，Dan 还是 Restful Objects 规范的作者，并且创作了几本有关 DDD 和 OO 的书籍。他住在英国牛津大学附近。

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

相关内容：

- 微软随.NET 4.5 发布新 REST API 框架
- IG GROUP 开源 RESTdoclet 项目
- IT 领域对技术的重视超过了思考

推荐文章 | Article

C++ 并发编程的一种思维

作者 梁国栋

愈发紧迫的挑战

现今，单台机器拥有多个独立的计算单元已经太常见了，这点在服务器的处理器上表现尤为明显，据 AMD 的一张 2012-2013 服务器路线图显示，服务器处理器的核心数将在 2013 年达到 20 颗之多。合理的利用 CPU 资源已是一个不得不考虑的问题。不少 C++ 程序员依然使用着多线程模型，但是对多线程的掌控并不是一件容易的事情，开发中容易出错、难以调试。有些开发者为了避免多线程带来的复杂度而弃用多线程，有些开发者则另投其他语言阵营，例如 Erlang。其实我们还有其他的选择，Theron 就是其中之一。

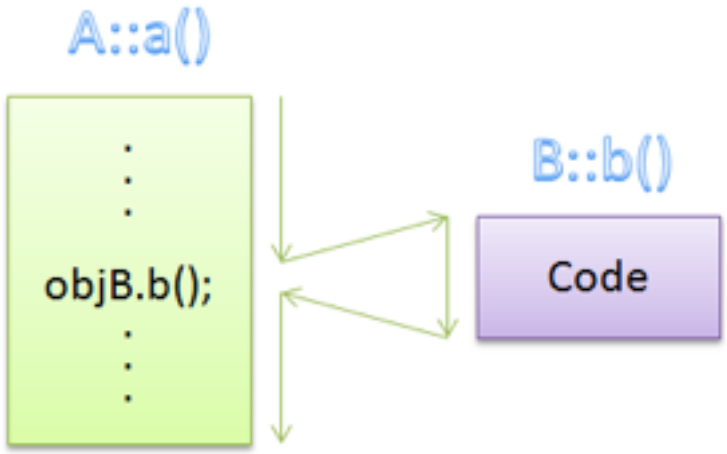
什么是 Theron？

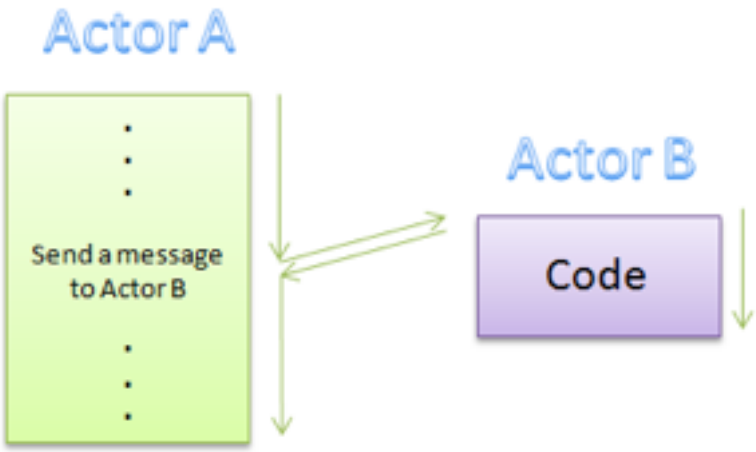
Theron 是一个用于并发编程的 C++ 库（http://www.theron-library.com/），通过 Theron 我们可以避免多线程开发中各种痛处，例如：共享内存、线程同步。Theron 通过 Actor 模型向我们展示了另一种思维。

什么是 Actor 模型？

Erlang 因为其优秀的并发特性而被大家所关注，而其并发特性的关键之一就是在于其采用了 Actor 模型（http://c2.com/cgi/wiki?ErlangLanguage）。与 Actor 模型相对应

的模型则是我们在面向对象编程中使用的 Object 模型，Object 模型中宣扬，一切皆为 Object（对象），而 Actor 模型则认为一切皆为 Actor。Actor 模型中，Actor 之间通过消息相互通信，这是其和 Object 模型的一个显著的区别，换而言之 Actor 模型使用消息传递机制来代替了 Object 模型中的成员方法调用。这样做意义重大，因为相对于成员方法的调用来说，消息的发送是非阻塞的，它无需等待被调用方法执行完成就可以返回，下图显示了此种区别：





A::a() 调用了 objB.b()，此时 A::a() 必须等待 B::b() 的返回才能继续执行。在 Actor 模型中，对应的做法是 Actor A 向 Actor B 发送消息并立即返回，这时候 Actor A 可以继续执行下去，与此同时 Actor B 收到消息被唤醒并和 Actor A 并行执行下去。

Theron 中的每个 Actor 都会绑定一个唯一的地址，通过 Actor 的地址就可以向其发送消息了，每个 Actor 都有一个消息队列。从编码者的角度看来，每实例化一个 Actor 都创建了一个和 Actor 相关的“线程”（非系统级的线程）。每个 Actor 总是被单线程的执行。总结来说 Theron 的并发特性的关键就在于：每个 Actor 在属于自己的单个“线程”中执行，而多个 Actor 并发执行。

Hello Theron

在谈及更多内容之前，我们先来看看 Theron 的一个简单的范例，借以获得一个最直观的印象。在 http://www.theron-library.com/可以下载到 Theron 的最新版，Theron 提供了 makefile 便于 gcc 用户编译，同时其也为 Windows 用户提供了 Visual Studio solution 文件 Theron.sln 用于构建 Theron。编译 Theron 很容易，不会有太多的障碍，需要注意的是构建 Theron 需要指定依赖的线程库，Theron 支持三种线程库：std::thread（C++11 标准线程库）、Boost.Thread 和 Windows threads。使用 makefile 构建时，通过 threads 参数指定使用的线程库（更为详细的信息参考：http://www.theron-library.com/index.php?t=page&p=gcc），使用 Visual Studio 构建时，通过选择适当的 Solution configuration 来指定使用的线程库（更为详细的信息参考：http://www.theron-library.com/index.php?t=page&p=visual studio)。下面我们来看一个最简单的范例：

```
#include <stdio.h>


#include <Theron/Framework.h>
```

```
#include <Theron/Actor.h>


// 定义一个消息类型
// 在 Theron 中，任何类型都可以作为一个消息类型
// 唯一的一个约束是消息类型的变量能够被拷贝的
// 消息按值发送（而非发送它们的地址）
struct StringMessage
{
    char m_string[64];
};


// 用户定义的 Actor 总需要继承于 Theron::Actor
// 每个 Actor 和应用程序的其他部分通信的唯一途径就是通过消息
class Actor : public Theron::Actor
{
public:
    inline Actor()
    {
        // 注册消息的处理函数
        RegisterHandler(this, &Actor::Handler);
    }
```

```
private:
    // 消息处理函数的第一个参数指定了处理的消息的类型
    inline void Handler(const StringMessage& message, const Theron::Address
from)
    {
        printf("%s\n", message.m_string);
        if (!Send(message, from))
            printf("Failed to send message to address %d\n", from.AsInteger());
    }
};



int main()
{
    // Framework 对象用于管理 Actors
    Theron::Framework framework;

    // 通过 Framework 构建一个 Actor 实例并持有其引用
    // Actor 的引用类似于 Java、C# 等语言中的引用的概念
    // Theron::ActorRef 采用引用计数的方式实现，类似于 boost::shared_ptr
    Theron::ActorRef simpleActor(framework.CreateActor<Actor>());
```

```
    // 创建一个 Receiver 用于接收 Actor 发送的消息
    // 用于在非 Actor 代码中（例如 main 函数中）与 Actor 通信
    Theron::Receiver receiver;

    // 构建消息
    StringMessage message;
    strcpy(message.m_string, "Hello Theron!");

    // 通过 Actor 的地址，我们就可以向 Actor 发送消息了
    if (!framework.Send(message, receiver.GetAddress(), simpleActor.GetAddress()))
        printf("Failed to send message!\n");

    // 等到 Actor 发送消息，避免被关闭主线程
    receiver.Wait();

    return 0;
}
```

这个范例比较简单，通过 Actor 输出了 Hello Theron。需要额外说明的一点是消息在 Actor 之间发送时会被拷贝，接收到消息的 Actor 只是引用到被发送消息的一份拷贝，这么做的目的在于避免引入共享内存、同步等问题。

Theron 的消息处理

前面谈到过，每个 Actor 都工作在一个属于自己的“线程”上，我们通过一个例子来认识这一点，我们修改上面例子中的 Actor:: Handler 成员方法：

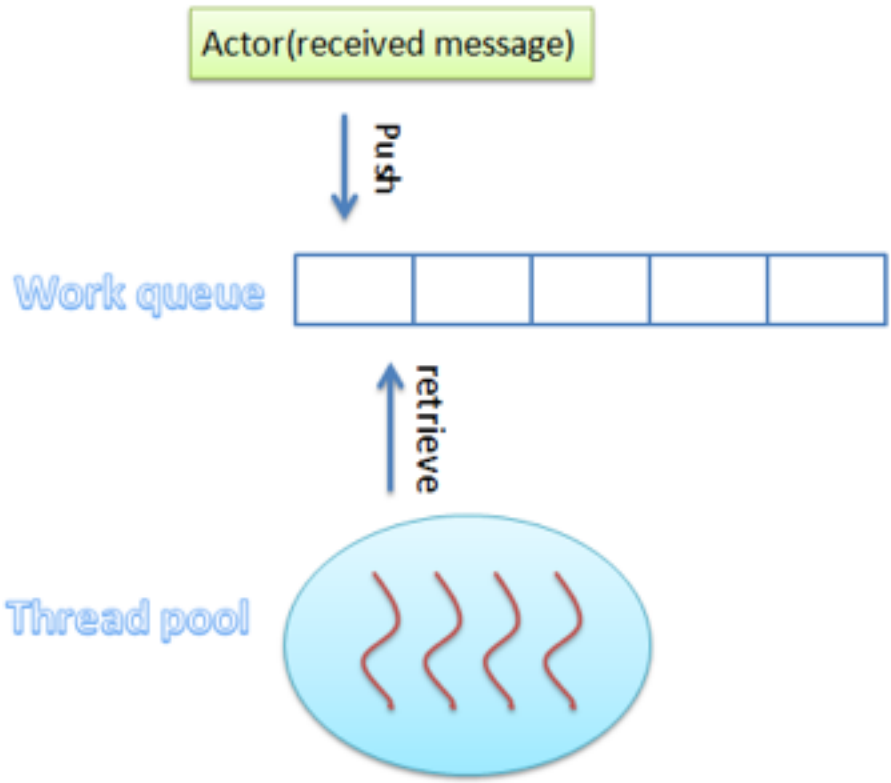
```
inline void Handler(const StringMessage& message, const Theron::Address from)
{
    while (true)
    {
        printf("%s --- %d\n", message.m_string, GetAddress().AsInteger());
#ifdef _MSC_VER
        Sleep(1000);
#else
        sleep(1);
#endif
    }
}
```

此 Handler 会不断的打印 message 并且带上当前 Actor 的地址信息。在 main 函数中，我们构建两个 Actor 实例并通过消息唤醒它们，再观察输出结果：

```
Hello Theron! --- 1
Hello Theron! --- 2
Hello Theron! --- 2
```

```
Hello Theron! --- 1

Hello Theron! --- 2

Hello Theron! --- 1

Hello Theron! --- 2

Hello Theron! --- 1

.....
```

这和我们预期的一样，两个 Actor 实例在不同的线程下工作。实际上，Framework 创建的时候会创建系统级的线程，默认情况下会创建两个（可以通过 Theron::Framework 构造函数的参数决定创建线程的数量），它们构成一个线程池，我们可以根据实际的 CPU 核心数来决定创建线程的数量，以确保 CPU 被充分利用。线程池的线程是以何种方式进行调度的？如下图：



接收到消息的 Actor 会被放置于一个线程安全的 Work 队列中，此队列中的 Actor 会被唤醒的工作线程取出，并进行消息的处理。这个过程中有两个需要注意的地方：

1. 对于某个 Actor 我们可以为某个消息类型注册多个消息处理函数，那么此消息类型对应的多个消息处理函数会按照注册的顺序被串行执行下去
2. 线程按顺序处理 Actor 收到的消息，一个消息未处理完成不会处理消息队列中的下一个消息 我们可以想象，如果存在三个 Actor，其中两个 Actor 的消息处理函数中存在死循环（例如上例中的 while(true)），那么它们一旦执行就会霸占两条线程，若线程池中没有多余线程，那么另一个 Actor 将被“饿死”（永远得不到执行）。我们可以在设计上避免这种 Actor 的出现，当然也可以适当的调整线程池的大小来解决此问题。Theron 中，线程池中线程的数量是可以动态控制的，线程利用率也可以测量。但是务必注意的是，过多的线程必然导致过大的线程上下文切换开销。

一个详细的例子

我们再来看一个详细的例子，借此了解 Theron 带来的便利。生产者消费者的问题是一个经典的线程同步问题，我们来看看 Theron 如何解决这个问题：

```
#include <stdio.h>


#include <Theron/Framework.h>

#include <Theron/Actor.h>


const int PRODUCE_NUM = 5;


class Producer : public Theron::Actor

{

public:
```

```
    inline Producer(): m_item(0)
    {
        RegisterHandler(this, &Producer::Produce);
    }

private:
    // 生产者生产物品
    inline void Produce(const int& /* message */, const Theron::Address from)
    {
        int count(PRODUCE_NUM);

        while (count--)
        {
            // 模拟一个生产的时间
#ifdef _MSC_VER
            Sleep(1000);
#else
            sleep(1);
#endif

            printf("Produce item %d\n", m_item);
            if (!Send(m_item, from))
                printf("Failed to send message!\n");
            ++m_item;
```

```
        }
    }

    // 当前生产的物品编号
    int m_item;
};

class Consumer : public Theron::Actor
{
public:
    inline Consumer(): m_consumeNum(PRODUCE_NUM)
    {
        RegisterHandler(this, &Consumer::Consume);
    }

private:
    inline void Consume(const int& item, const Theron::Address from)
    {
        // 模拟一个消费的时间
#ifdef _MSC_VER
        Sleep(2000);
#else
        sleep(2);
#endif
```

```
        printf("Consume item %d\n", item);
        --m_consumeNum;

        // 没有物品可以消费请求生产者进行生产
        if (m_consumeNum == 0)
        {
            if (!Send(0, from))
                printf("Failed to send message!\n");
            m_consumeNum = PRODUCE_NUM;
        }
    }

    int m_consumeNum;
};

int main()
{
    Theron::Framework framework;

    Theron::ActorRef producer(framework.CreateActor<Producer>());
    Theron::ActorRef consumer(framework.CreateActor<Consumer>());

    if (!framework.Send(0, consumer.GetAddress(), producer.GetAddress()))
```

```
        printf("Failed to send message!\n");


    // 这里使用 Sleep 来避免主线程结束
    // 这样做只是为了简单而并不特别合理
    // 在实际的编写中，我们应该使用 Receiver
#ifdef _MSC_VER
    Sleep(100000);
#else
    sleep(100);
#endif


    return 0;
}
```

生产者生产物品，消费者消费物品，它们并行进行，我们没有编写创建线程的代码，没有构建共享内存，也没有处理线程的同步。这一切都很轻松的完成了。

代价和设计

和传统的多线程程序相比 Theron 有不少优势，通过使用 Actor，程序能够自动的并行执行，而无需开发者费心。Actor 总是利用消息进行通信，消息必须拷贝，这也意味着我们必须注意到，在利用 Actor 进行并行运算的同时需避免大量消息拷贝带来的额外开销。Actor 模型强调了一切皆为 Actor，这自然可以作为我们使用 Theron 的一个准则。但过多的 Actor 存在必然导致 Actor 间频繁的通信。适当的使用 Actor 并且结合 Object 模型也许会是一个不错的选择，例如，我们可以对系统进行适当划分，得到一些功能相对独立

的模块，每个模块为一个 Actor，模块内部依然使用 Object 模型，模块间通过 Actor 的消息机制进行通信。

Theron 的未来

Theron 是个有趣的东西，也许你没有使用过它，你也不了解 Actor 模型，但是 Actor 的思想却不新鲜，甚至你可能正在使用。目前来说，我还没有找到 Theron 在哪个实际的商业项目中使用，因此对 Theron 的使用还存在一些未知的因素。还有一些特性，诸如跨主机的分布式的并行执行是 Theron 不支持的，这些都限制了 Theron 的使用，不过作者正在积极的改变一些东西（例如，作者表示会在今后添加 Remote Actors）。无论 Theron 未来如何，Theron 以及 Actor 模型带来的思想会让我们更加从容面对多核的挑战。

作者简介

梁国栋，热爱编程，热衷于撰写技术类文章，精益思想倡导者，UNIX 哲学实践者，专注于高性能服务器程序的研发多年，目前负责网游服务器的研发工作。

原文链接：

http://www.infoq.com/cn/articles/cpp_concurrent_programming_introduce_theron

推荐文章 | Article

优秀的 JavaScript 模块是怎样炼成的

作者 田永强

引言：如今的 JavaScript 已经是 Web 上最流行的语言，没有之一。从 Github 上的语言排行榜 https://github.com/languages 上即可看出，也是如今最为活跃的开源社区。随着 Node 的加入，JavaScript 开枝散叶进入服务器领域，为这个语言榜的占比，也贡献了几分热度。尽管经历了 Web2.0 的洗礼 ，但在国内谈及开源，开源人士似乎都当这门语言并不存在，这也意味着国内的开发中坚阶层，并没有改变 JavaScript 以及前端过去二流形象的认识，也没意识到 JavaScript 如今真正的价值。历史原因造就如今的局面，但是你并不能就此否认，一个出身不好，小时候还挺调皮的孩子，他长大后就是没有出息。本文将介绍一个优秀的 JavaScript 模块应该是怎样炼成的，以期望未来国内的开源社区能够涌现出更多的优秀模块。

引起我写这篇文章冲动的是近期接触到 moment 模块时的震惊。用 JavaScript 写程序，通常要经历两个阶段：第一个阶段是采用一些模块来擦除 JavaScript 语言（跨平台）自身的问题；第二个阶段是自己写一些方法来擦除引入的模块的问题。第三个阶段才会真正进入业务开发中，前两个阶段俗称“擦屁股”。时常在第三个阶段时，我会陷回到第二个阶段，和各种模块碰撞，往往会引起一些心情上的不舒爽。而在接触到 moment 时，这些烦恼瞬间消散，我在心底知道为何会如此释怀，如碰到心仪的女神。这也让我回头反思过去在使用和写模块过程中遇到的挫折和收获，这落差让我产生了如此的冲动，此文算是一个总结，期望能在汲取优秀模块的经验中，国内开源社区能够成长起来。

为什么是模块，而不是库或者框架

在过去几年做前端的同学多半谈论的是库，或是框架，提及库则是 Prototypejs 和 jQuery，框架则是 YUI3，Dojo、Extjs 等。关于库或者框架的，对应的比喻则是大教堂和集市。在最早的时间，大教堂和集市的建设都在同步行进，具有代表意义的是 YUI3 和 jQuery，前者 Yahoo!投入许多精力，历时两代完成，后者则是集市建造的典范，由 John Resig 创建，社区参与。最后看看后续的反馈：

大教堂未必优质

大教堂模式产出的作品，让享用者可以有一步到位的服务，无需其他。但是由于大教堂的建设过程中承担着解决所有问题的使命，这导致它逐渐变得庞大，尽管它有着宏大而美妙的结构，像一首长长的赞美诗，任何一段都可以被唱诗班演唱得动听。但是，弱点依旧暴露出来：

1. 局部的优良程度未必是最好的；
2. 庞大的结构导致灵活度下降，升级困难。
3. 任何一条龙服务，都不能保证每个环节都是优异的。

小集市钻石闪烁，沙砾良多

小集市的特点是开放式建设、周期短、成本低。大多数创建出来的集市是功能简陋、品质平庸的。jQuery 是一个值得玩味的现象，品质极高，但是它带来的插件市场，却体现了小集市的另一面，大多数的 jQuery 插件的质量却十分低下。

可以说小集市模式创建出来的作品，在单点上，是能够超越教堂模式作品中对应的那部分。前者缺乏一个宏伟的结构，但是在微观上，它是完善的，可以随意挪移的。后者虽然具备优良的结构，但在单点上并非优秀，这也容易让人怀疑是否其他点上也并不优秀，而且教堂式作品的一个特点则是，局部是不可移植的，非独立性的，这在 YUI3 中表现十分明显。上述对比很容易让人联想到，如果一个架构的特性具备单点可移植，可拆卸，自身极其轻量，汲取了大教堂和小集市的优点。那必将是一个全新的时代，那是一个可以 DIY 的时代。原有的大教堂模式由于缺失了灵活性，在迭代迅速的开发中，必将淘汰。而小集市尽管良莠不齐，好在预留了选择权利给用户，并且他的开放性也预示着它的可成长性，所以还有未来可期。

没错，如今这个时代已经到来。库通常是一个比框架小一个粒度的单元，模块则是比库更小一个粒度的单元。一个库可能由几个模块组成，框架则可能是在几个库的基础上构建。不再谈论库和框架的原因是将库和框架的架构部分还给开发者，以便开发者可以根据实际业务去构建合适的解决方案。任何框架或库都只具备解决某一方面的能力，不具备普适性。当粒度降低到模块级别的时候，构建任何上层业务，可以实现按需使用。由于模块的粒度更小，所以相对库而言，更加专注，变优秀的成本更低。类比孩子在充满沙砾的海边奔跑，但很容易装满一口袋的贝壳。对于稍具慧眼的开发者而言，挑选一堆适合的模块来解决业务的问题，比使用教堂式作品或者更高效。

这种方案因为 CommonJS 模块规范的影响，已经在既定事实中成型。在 Node 中，NPM 平台（https://npmjs.org/）上 13000+的模块数量可以说明这个问题。前端中由于受到历史原因的影响，进展较慢，国内玉伯的 SeaJS 在推动此事，Arale 是在 SeaJS 基础上创建了一些模块，这类模块可以非常容易迁移到其他符合 CommonJS 模块规范的环境中。

除此之外腾讯的 JX 也具备 SeaJS 的特性，可以轻松将别的库应用到 JX 中，但是还不够规范。

如果非得给这种模块提供方式一个名称，我觉得该叫做小教堂模式，具备小集市的小，意味着这个教堂可能只提供祈祷服务，如果注册结婚，则需要换另一家小教堂。但是这类小教堂的服务是最优质的。创建这类小教堂只比创建一个优异的小集市略费成本，换言之，优异的小集市就是小教堂，它的创建方式是沿袭小集市的开放、透明的、有既定目标的成长性的。

另一个考虑是，在大部分的情况下，我们是不需要大教堂的，现实中为了整体环境，我们宁愿在大教堂中祈祷，但是程序设计中，我们不会因为喜欢一棵树木，而买下整块山头，我们几乎不喜欢任何看起来多余的部分。同时我们也不需要小集市，我们需要的是一个品质优良的商场。开发者是采购者，采购模块的过程既是架构的设计的过程。

前文我也提到这种小教堂的模式依然存在不理想的功能重叠、功能多余、功能缺陷、功能冲撞等问题。可谓说完美的小教堂是可遇不可求的，但一旦它完美，将再难被替换。目前阶段的 JavaScript 模块开发还存在着这些问题。是故，如果开发者了解如何去打造一个优秀的 JavaScript 模块，并乐于贡献到开源社区，这将大幅提升社区 JavaScript 水平，后续的开发者在做业务架构时，将具备更优质的材料来 DIY 更优异的产品。

炼成优秀模块的最佳实践

其实这并不算是精确的最佳实践，只是从别的模块哪里学习到的和自己过去的一些经验，仅做一定的总结。欢迎补充和讨论。

模块自身的素质要求

要写出一个优秀的模块，模块自身的素质十分重要，如果自身条件太差，成为优秀模块的概率是极小的。这些自身素质包括美好的愿景、专注的定位、名字、API 设计、文档、测试等。

合理的愿景：设定既定目标

如果你打算写作一个模块，并贡献到开源社区中，并期望它是优异的，并被许多人使用的，那么为它设定一个既定目标是首要的。如果这个目标是没有意义，没有趣味的，那多半没有人使用，甚至自己开发到一半都没有兴趣继续写下去。没有理想的屌丝，注定不能成为高富帅。

对于具备众多坑爹问题的 JavaScript 语言而言，找到一个目标并非难事。典型的例子如：jQuery 专注解决 DOM 操作和 Ajax、Underscore 专注对象和集合的操作、QUnit 和 Jasmine 专注 BDD 和 TDD 的单元测试、moment 模块专注解决从 Java 那里学过来的 Date 的问题；拿近一些的例子，玉伯的 SeaJS 专注模块加载，老赵的 Wind.js 专注异步编程同步化来解决流程控制问题；拿一个有趣的例子，PNGDrivehttps://github.com/MadeInHaus/PNGDrive 这个项目虽然没有什么实际用处，但是将文件编码为图片显示出来的方式足够有趣。

另外这个目标必须是既定的。也就意味着饼不用画为无限的，这个目标一定是可以完成的。如果目标太大，也就意味着模块自身会变复杂。jQuery 兼容各种浏览器的 DOM 操作这个目标，在移动平台上变得没有意义，所以存在着 Zepto.js 这样的项目。更小的目标意味着模块自身简洁，且能够更专注，目标更容易抵达。一旦抵达目标，该模块就是稳定的，未来被替换的机会极小。

为模块或项目起一个贴切的名字

模块需要一个贴切而好记的名字。这个名字何以帮助用户最直观地感受模块。SeaJS 在起名上算是一个表率，让人很容易有海纳百川的联想，这也正是 SeaJS 的行为。一个好的名字可以使得模块的后期推广事半功倍，而且一旦开始推广，尽量不要换名字。

不做逾越的事

并非每个使用者都喜欢买一送一的感觉，因为后面这个一，对于使用者而言，并非是期望的，所以它的优良无法直观的判定好坏。无关的方法一定不要提供。评判一个模块是否完美，不是可以添加 API，而是无法再减少 API 了。

不污染公共环境

每个人都不喜欢公共环境被人污染。破窗效应揭示，如果一辆汽车的门窗稍有损坏，不立即修复，那么很快整辆车就会被破坏甚至偷走。 在 JavaScript，公共环境包括全局变量，原型链等。jQuery 和 Underscore 为了代码的写作方便，占用了$和_两个符号，尽管他们都提供了 noConflict 方法来避免冲突，但是抱怨者还是大有人在。所幸这两个库太过于知名，几乎没有再有的库来使用这两个变量名。另一个例子是 Prototype.js 库对对象进行扩展时，直接在 Array、Object 等原生对象的原型链上添加方法，尽管它看起来不影响，但是总有冲突的一天，并且使用者并不一定知晓原型链被改动，在他的默认上下文中，难保他也不去修改原型链。相比 Prototype.js 的做法，Underscore 提供的方法则优雅许多，另行提供 API 来处理操作，而不是修改共有的原型链。

在 Node 和浏览器中，global 和 window 是全局对象，如果随意放置变量到全局对象上，也容易遭到他人修改或者覆盖你变量的事情。

CommonJS 提供的 require、exports 则十分优雅解决这个问题。谁使用，谁引入。而不是通过全局变量。在前端没有 AMD 或者 CommonJS 环境下，则是采用命名空间和闭包来解决这个问题。

不污染公共环境是模块与模块之间互相不影响的基本保证。如果引入你的模块，导致其他模块失效的事情，多半是不招人待见的。

抵制墨菲效应

有可能变糟糕的事情，它变糟糕的可能性就会变大。模块在升级，迭代的过程中，如何避免这种糟糕的事情发生呢？答案是单元测试。当单元测试覆盖了你认为会出问题的地方，可以避免相同的错误再次发生。这是模块稳定迭代的基本保证。当我发现一个仅仅为了解决日期操作的 moment 模块，它为数不多的 API 竟然具有多达 7000+的断言时，十分惊讶。

过去 JavaScript 只做简单的事情，地位低下，所以对于 JavaScript 的质量保证也极少。这个思维需要改变，一个用户在评估采用你的模块时，如果单元测试都无法看到，心里该是有多不踏实。

除此之外，还有性能测试，性能测试结果可以横向比较，也可以纵向比较，有利于感知模块的具体性能表现。

数据通常容易打动理性的人。

保持简洁

对于任意看起来复杂的事物，用户均会觉得它很复杂。老赵的 Wind.js（前身是 Jscex）是一个想法独特的模块，但在提供的 API 上由于 eval，以及需要引入的模块较多，让它看起来比较复杂，让用户感觉潜意识里复杂，这种复杂的信号很容易变成它有问题的信号，让人心生疏远。

将能不暴露给用户看到的东西，尽量隐藏，过多的步骤只会让用户觉得麻烦和不靠谱。

职责单一

这里的反面例子来自于 Require.js。如果你用过 RequireJS，可以看到 require 方法是一个变态的设置，参数为’a’、[‘a’]、[‘a.js’]、{}他们的行为都是不一致的。这种参数类型可以随意变化是 JavaScript 的灵活的地方，但是函数的行为如果也发生变化，这会让人产生迷惑。适当的重载并不意味这行为也要完全不同。

功能过多，带来的问题的可能性也会变大，使用者在调用过程中也会增加无形的压力。分离功能可以保持方法职责单一会是你 API 优秀的一部分。

编码风格

编码风格一定需要统一，而且编码风格一定不要显得外行。如果你的用户发现你的编码风格是 PHP 或者 C#的，他们可能会产生你不是专业的这个感觉。推荐贴近 JavaScript 社区，采用 JSLint 或 JSHint 来矫正编码风格。这有利于源代码的阅读，在不同的人之间传递不会有不换了一门语言的感觉。

API 接口漂亮

API 接口漂亮包含几个方面：

命名

对外 API 的命名需要谨慎对待，方法名太长、方法名不直观、方法名大小写不对、方法名单词太复杂都会影响到使用者的直观感受。由于国内的英文水平高低不一，使用者遇见不认识的单词，都会造成障碍。

jQuery 在方法命名上十分优秀。简短，直观，优雅。

调用

实参的传入也是考验 API 设计者的地方。如果需要调用者传入的参数较多，则该反思该 API 是否适合暴露给调用者。如果调用参数确实较多，并且支持可选项，则传入一个对象作为参数较为合适。由于对象带有明确的 key，获取参数也无需一个一个检测。

$.ajax()是一个典型的例子，它支持的参数非常多，并且大多可选。所以暴露的 API 为 $.ajax(url, obj)或$.ajax(obj)。

在 JavaScript 中，链式调用也是让用户较为喜爱的一点。Underscore 除了提供普通的 API 外，还支持包装对象之后进行链式调用。这让熟悉函数式编程的人，顿生亲切。

习惯

Zepto.js 是一个经典的案例，它提供了与 jQuery 几乎完全兼容的 API，为的是照顾用户对于 jQuery 的熟悉。这让它可以零成本被应用到移动浏览器上。

jQuery 的另一件反面例子则是它的 each 方法，与原生数组的 forEach 方法的回调传入值次序不同，这与习惯不同的接口，会造成一定反感心理。 在设计 API 的过程中，尽量寻找贴合用户习惯的已有形式，这会让用户易于接受。

可扩展性

模块在开发的过程中，可以包括有限的部分和无限的部分。有限的部分将会通过项目迭代，臻于完美。如果模块存在无限的部分，并且在有限的部分留出扩展来衍生无限，这对于模块的成长，这是一个大大的加分项。

jQuery 留出$.fn 来供用户扩展它，形成的影响是大量的 jQuery 插件涌现了出来。尽管大多数情况没有被正确使用，但不能掩盖它是一个漂亮的设计。

moment 模块在它的 lang 部分，也提供了优雅的扩展它的部分。使得不同地区的用户可以自定义语言的显示。

扩展性的存在，使得开源社区能够参与，能够起到抛砖引玉的效果，反过来会增进模块的功能。

使用合适的设计模式

合适的设计模式可以让模块自身无瑕。不合适的设计模式则会适得其反。

这里的正反例子都与 jQuery 相关。正例子是 Deferred 的应用。过去 ajax 操作 success 和 fail 回调都必须写到$.ajax(obj)的参数对象中，但是 Deferred 对象使得调用更加自然：

```
$.get("test.php").done(function() {    alert("$.get succeeded");  });
```

LABjs 的设计模式也十分优秀，script 和 wait 两个用于加载和阻塞的 API，通过链式调用，其乐融融：

```
<script>
$LAB    .script("framework.js").wait()    .script("plugin.framework.js")    .script("myplugin.framework.js").wait()    .script("init.js").wait(); </script>
```

反例子则是来自 jQuery 插件。jQuery.fn 不失为一个好的扩展点，但是有大量的 jQuery 插件操作的并非 DOM，却生搬硬套将方法挂靠在 jQuery.fn 上。另一个例子则来自 jQuery 社区得意的 UI 组件。

$(foo).dialog('open')

这类通过传入参数，又不能得到期望的返回值的场景，并不适合操作这个组件对象，API 的参数传递更是显得不伦不类。如果是直接操作组件对象，则更友好一些。相比 jQuery UI，YUI3 的组件则优秀太多，API 漂亮，组件的层次结构分明，易于扩展和自定义，jQuery UI 通过插件方法的调用方式，自定义组件的代价极大。

合理的目录结构

开发方式多半会影响到后续的使用方式。尽管前端脚本多半都是提供一个文件给用户，但是在开发过程中，合理地组织自己项目的目录结构是值得注意的。jQuery 的源代码目录中，各个功能点都分别写在各自的文件中，使得开发过程中编写代码方便。

另一方面，CommonJS 的包规范还定义了以下目录和文件：

bin doc test lib package.json

分别用于存放二进制文件、项目文档、单元测试用例、源代码。package.json 文件则用于描述该包的一些包括包名、版本号、依赖等的信息，详情见 http://wiki.commonjs.org/wiki/Packages/1.0。遵循规范的目录结构通常更好一点，因为大家都有同一个准则来参考，彼此更熟悉。

巨细的注解

通常用户真正需要去阅读你的代码的时候，是出现问题的时候。在开源社区，如何让发现你问题的人刚你改进代码，注释的作用功不可没。

另外，当一个用户真正是来欣赏你的代码时，如果看到 Underscore 这样密度的注解时（http://documentcloud.github.com/underscore/docs/underscore.html），还有拒绝它的勇气吗？

清晰明了 API 的文档

优秀的模块不仅仅体现在代码写得好上，更多的体现在如何让用户使用时更舒适。API 文档必不可少。我心目中的 API 文档应该详细描述方法作用、参数、返回值的。甚至还应该有 demo 代码伴随。

API 文档可以通过 jsdoc 之类的注解文档来生成。也可以另起文档来写。

API 文档的一个特点是应该能方便查找，能在一个页面中展现完成的，尽量不要页面跳转或翻页。

API 文档最大的作用让用户精确理解 API，使得不造成误解，和清楚输入输出。

Underscore 的 API 文档（http://documentcloud.github.com/underscore/）借用模版生成了漂亮的页面，使得查阅方便。

一见钟情的 demo

相比 Node，前端 JavaScript 模块更擅长做这件事情，尤其是 UI 库。男女之间首次见面的第一印象，很大程度可以决定某两个人是否会谈恋爱的概率。demo 提供的形式可以影响到用户的直观体验，一般而言，用户觉得越炫，但是旁边提示的示例代码越简单，越会引发用户的好奇心。如果只有代码，或者只有 demo，都会在表现性上打折扣。

对于 Node 的模块而言，由于没有 live demo 的感觉，尽量展示 sample 代码，让用户了解到他的目标是否与模块的目标一致。不要让用户错过你的模块，也不要让你的模块错过了它的用户。

README.md

README 文件承担的作用仅次于 demo，无需让用户产生心动的感觉，但是一定要引导用户更深度的了解你的模块，详细阅读你 README 的人，多半是有兴趣使用你的模块的人在做实地考察了。一个 geting start 入门是必不可少的，到 API 文档的链接也应该有。如果空落落的 README，会立马有生疏感的。后期用户的心得体会等相关文章，也记得更新到 README 中。

模块的社区打造

话说酒香也怕巷子深。在满足成为优秀模块的基本素质要求后的首要事情是如何将模块推向开源社区，积极到社区中宣传。

放到 Github 开源：不仅仅是代码

每一个程序员都应该有属于自己的 Github 帐号。如果你没有，不是 Github 的遗憾，而是你的遗憾。关于 Github 有什么，可以参看这篇文章：如何高效利用 GitHub（http://www.yangzhiping.com/tech/github.html）。 排除掉文化上带来的好处。Github 可以帮我们托管代码，帮我们解决版本控制的问题。它可以提供一个 wiki，帮我们存放文档。它提供 Issues 页面，使得别人可以为模块提出意见和反馈。提供 Pull Requests 页面，使得别人可以帮我们修改代码后，供我们合并修改。

交流平台的创建

交流平台主要用于讨论、答疑，使得模块作者和用户之间可以产生思维的碰撞。交流过程可以不断完善模块自身的不足，形成文档。主要的形式有如下：

邮件列表

国外的社区大多采用 Google Groups 的邮件列表来交流。邮件列表可以将讨论发送到每一个关注它的人手中。

实时交流

国外多采用 IRC 频道来进行此事。国内的情况，可以选择合适的 QQ 群来进行交流。

留言版

留言版的功能 Github 的 issue 具备了该功能。但是如果具备单独的介绍页面或者站点，集成 Disqus 来收集用户的反馈会是个不错的选择，因为这更符合普通用户的习惯。

版本控制

这里的版本控制不是 git 的版本控制。而是模块的整体版本。标注好模块的版本是分场重要的，如果用户需要升级模块，它能够得到明确的指导。前端模块中通常是在文件名上，或者文件内部写明版本。对于 Node 而言，写在 package.json 文件中。

最好能够在大版本的发布时，通过正式的新闻页面，或是邮件列表发出新闻通知，以显得版本发布的正式。

悠扬的历史：Change Log

不要小看 Change log 的作用，长长的，细碎的 Change log 可以给用户该模块历史悠扬的感觉，也能让用户体会到写作该模块的过程细节。每次迭代发布的过程中，展现给用户看当前的 change log 也能让用户知道改动的影响范围。如果碰巧发布了一个用户需要的方法，他会有收到礼物的感觉。

选择合适的 License

国内的环境中，也许大家不太关心 License 的问题。事实上，License 的选择，会影响到用户的评估。目前只有 BSD 和 MIT 两种 License 可以让人放心使用。对于商业公司而言，如果不是这两种 License，他们需要投入更多的精力和时间周期来评估这个模块是否可用。但是对于 JavaScript 社区而言，通常采用的是 MIT，所以也基本没有问题。选好 License，并在明显且不重要的位置表明该模块在什么 License 下发行。

发布到 NPM 中

在 Node 中，搞定代码后，发布到 NPM 中是最应该做的事。

易于获得的感觉

npm install module 或是一个显眼又不失素雅的 Download 按钮会在潜意识中让用户觉得易于获得，容易集成到现有系统中。不要做了各种分享介绍之后，让用户找不到获取模块的地方。

持续集成

一个随时都能拿出钥匙，开出跑车的青年，必将被认为是高富帅。支支吾吾不能随时给出结果的人，基本是屌丝无疑。

推荐注册你的项目到 travis-ci，并运行它。绿色的 Passing 时刻昭示该项目的可靠指数、稳定指数较高。这个小图标也能帮助你检测迭代是否出现问题。

漂亮的站点

注册一个模块名字的 org 域名，一套简洁明了的 UI，清晰的导航，简单的 demo 等等。细节在前后的实践中都有提及。

标致的 Logo

如果你没有任何视觉设计的天赋，厚着脸皮找你的视觉设计师同事要一枚吧，尽管它是一个锦上添花的事情，但是 Logo 在品牌认知上的功劳是不用质疑的。

线下分享交流

开发完模块后，一定记得分享给团队的同事，他们应该是模块的首批用户，他们给予的反馈也是最宝贵的。

另外，还可以在线下社区分享，将你的模块的推广范围从身边延续到这个城市。尽管在线下社区分享模块的机会不多，但是一旦有，不要忘记介绍自己得意的模块给他人。

分享带来的收益是明显的，有利于自己梳理对模块的认识，也能收到用户的直观反馈。

及时响应反馈

也许你的模块已经很久没有更新，但是用户还是会发来反馈或提出问题，甚至提交 pull request，请及时响应需求。如果是提出问题，说明你的文档还不够完善。如果是提出需求，则说明你最初设定的目标还没有完成。

善意营销：赏金猎人

如果有用户指出你的低级 bug，或是帮你写作了模块的体验文章，这些事情都是值得模块作者有所表示的时候，因为这些用户是你的忠实用户，他们在帮你的模块成长。有所表示并不意味着要给多少钱，一些有意义的奖品或纪念品更适合拉近这些用户与你之间的距离。

与兄弟社区的互动

一个社区如果过小，需要到隔壁的社区中发布一些信息让大家知晓。如果可以，无论国内还是国外的的兄弟社区的协助推广，将会对社区运作有很大的帮助。友情链接是一个典型的方式。

模块开发者的自我修炼

模块自身的优秀加上社区的打造可以保证模块拥有不错质量和口碑。但是决定模块能否走得远，模块开发者的自身素质也十分重要，这其实模块的另一个软素质。经历了开发阶段和社区运作阶段后，越发展到后期，开发者的自身修炼的影响越会凸显。

忍得住寂寞&持续坚持

典型的例子是老赵对于 Wind.js 的坚持。在对 Wind.js 的开发和推广上，可谓是寂寞的。略呈偏门的异步同步化、eval、编译等工序，被误解和排斥的多次。这是需要忍受的，如果没有持续坚持和忍住寂寞，模块就没有明天。作者的热情一旦消散，用户的热情也必然消散。

简单专注

前文的愿景部分提及到了简单专注。简单专注，不仅仅是在项目初期保持，在长久发展中，也应当如此，只有简单专注方能保证小教堂的服务质量是最顶尖的。

奉献精神

模块开发和推广事实上是没有直接的经济收入的，而且还会耗费大量的时间和精力。但是开源事业的收益，实际上是无法用金钱进行衡量的。并且这个过程也是自愿自发的，没有人会主动来推动你。

如果确实对经济造成影响，可以在页面上加上 donate 的链接。优秀的模块不会让这个 donate 链接的功能白费的。

幸运的是开源社区中不会缺乏具有奉献精神的人，是他们在推动社区和技术发展。

演讲能力

前文提到线下分享，这对于开发者的演讲能力有更多的要求。演讲能力的高低，是在另一个层面上提升模块的表现力，尽管这种能力不需要运行在 CPU 中。

文笔

模块开发者的文笔在文档的影响上是能够直观反映的。除此之外在社区推广中，文案的品质也有一定的影响。如果开发者文笔能够好到用优美的文章去传递模块的思想，这是令人愉悦和容易接受的。

人格魅力

模块的开发者应当具备良好的人格魅力，这包括人际关系、交流能力等。一个具备人格魅力和一个并不为人所知需要进一步了解的开发者，他们展现在大家面前的收效是并不相同的。后者更容易吸引他人，为模块带来更多的用户，这些人是你的用户，也是你的合作者。他们的涌入，会帮助改进模块。 关于人格魅力这个软素质，这里不再多说，相信都了解他的重要性的。

总结

通过对最佳实践的罗列，我自己相当于重温了一遍 JavaScript 开源的一系列文化。而这些最佳实践部分其实都是必要条件，通过这些最佳实践，未必保证会有优秀的 JavaScript 模块产出。但是观察如今的那些优秀模块，他们基本上都具备这些特征。最后，JavaScript 社区虽然活跃，佼佼者不在少数，但是整体水平的提升，还需时日，希望此文能够帮助到一些开发者并欢迎讨论。

关于作者

田永强，新浪微博@朴灵，前端工程师，曾就职于 SAP，现就职于淘宝，花名朴灵，致力于 NodeJS 和 Mobile Web App 方面的研发工作。双修前后端 JavaScript，寄望将 NodeJS 引荐给更多的工程师。兴趣：读万卷书，行万里路。个人 Github 地址：http://github.com/JacksonTian。

原文链接：http://www.infoq.com/cn/articles/how-to-create-great-js-module

相关内容：

- 微软发布 C++ AMP 规范，旨在降低数据并行门槛

Powered by InfoQ

2012 QCon

全球软件开发大会 [杭州站]

International Software Development Conference

- **Amr Elssamadisy**
  软件开发教练
- **Blake Matheny**
  Tumblr平台工程总监
- **Gerard Meszaros**
  Independant Consultant敏捷教练
- **Dave Thomas**
  敏捷宣言发起人
- **Jonathan Siegel**
  RightScale创始人

× **更有国内讲师，来自——**
百度、Boilerplate、RedHat、Scala社区、阿里巴巴、爱迪德等

专题及主持人：

1. 知名网站案例分析　冯大辉
2. 高效运维　高山渊
3. PaaS引擎已经启动　胡喜
4. 脚本代码之美　鄢学鹏（秦歌）
5. Java 20年　江南白衣
6. 大数据思与行　卢东明
7. 构建高效能团队　滕振宇
8. 让移动用户High起来　郝培强
9. 优秀测试实践分析　吴穹

10月报名开始啦！
5人以上团购，更享超值优惠！
另有Cloud Foundry启动营、技术狂欢夜、社区之夜等精彩免费晚场活动！

10月25-27日 | 杭州之江饭店

主办方：InfoQ　战略合作伙伴：Alibaba Group 阿里巴巴集团

赞助单位：天翼空间·应用工厂 www.189works.com　QINIU 七牛云存储　CLOUD FOUNDRY　ebay　PayPal　SanDisk　vmware

www.infoq.com/cn　www.qconhangzhou.com

推荐文章 | Article

在 MongoDB 中实现聚合函数

作者 Arun Viswanathan, Shruthi Kumar 译者 肖中南

随着组织产生的数据爆炸性增长，从 GB 到 TB，从 TB 到 PB，传统的数据库已经无法通过垂直扩展来管理如此之大数据。传统方法存储和处理数据的成本将会随着数据量增长而显著增加。这使得很多组织都在寻找一种经济的解决方案，比如 NoSQL 数据库，它提供了所需的数据存储和处理能力、扩展性和成本效率。NoSQL 数据库不使用 SQL 作为查询语言。这种数据库有多种不同的类型，比如文档结构存储、键值结构存储、图结构、对象数据库等等。

由于 NoSQL 数据库具有高速的读写能力，因此它的的典型应用包括归档历史日志、事件日志、电子商务日志、游戏数据、社交数据等，存储下来的数据是为了进行后续处理，以得到关于用户以及他们使用情况的有用信息。

我们在本文中使用的 NoSQL 是 MongoDB，它是一种开源的文档数据库系统，开发语言为 C++。它提供了一种高效的面向文档的存储结构，同时支持通过 MapReduce 程序来处理所存储的文档；它的扩展性很好，而且支持自动分区。Mapreduce 可以用来实现数据聚合。它的数据以 BSON（二进制 JSON）格式存储，在存储结构上支持动态 schema，并且允许动态查询。和 RDBMS 的 SQL 查询不同，Mongo 查询语言以 JSON 表示。

MongoDB 提供了一个聚合框架，其中包括常用功能，比如 count、distinct 和 group。然而更多的高级聚合函数，比如 sum、average、max、min、variance（方差）和 standard deviation（标准差）等需要通过 MapReduce 来实现。

这篇文章描述了在 MongoDB 存储的文档上使用 MapReduce 来实现通用的聚合函数，如 sum、average、max、min、variance 和 standard deviation；聚合的典型应用包括销售数据的业务报表，比如将各地区的数据分组后计算销售总和、财务报表等。

我们从本文示例应用所需软件的安装开始。

软件安装

首先在本地机器上安装并设置 MongoDB 服务。

从 Mongo 网站上下载 MongoDB，解压到本地目录，比如 C:>Mongo

在上一个文件夹内创建数据目录。比如：C:\Mongo\Data

如果数据文件存放在其他地方，那么在用 mongod.exe 命令启动 MongoDB 时，需要在命令行加参数—-dbpath

启动服务

MongoDB 提供了两种方式：mongod.exe 以后台进程启动；mongo.exe 启动命令行界面，可做管理操作。这两个可执行文件都位于 Mongo\bin 目录下；

进入 Mongo 安装目录的 bin 目录下，比如：C:> cd Mongo\bin

有两种启动方式，如下：

```
mongod.exe –dbpath C:\Mongo\data
```

或者

```
mongod.exe –config mongodb.config
```

mongodb.config 是 Mongo\bin 目录下的配置文件，需要在此配置文件中指定数据目录（比如，dbpath= C:\Mongo\Data）的位置。

连接到 MongoDB，到这一步，mongo 后台服务已经启动，可以通过 http://localhost:27017 查看。 MongoDB 启动运行后，我们接下来看它的聚合函数。

实现聚合函数

在关系数据库中，我们可以在数值型字段上执行包含预定义聚合函数的 SQL 语句，比如，SUM()、COUNT()、MAX()和 MIN()。但是在 MongoDB 中，需要通过 MapReduce 功能来实现聚合以及批处理，它跟 SQL 里用来实现聚合的 GROUP BY 从句比较类似。下一节将描述关系数据库中 SQL 方式实现的聚合和相应的通过 MongoDB 提供的 MapReduce 实现的聚合。

为了讨论这个主题，我们考虑如下所示的 Sales 表，它以 MongoDB 中的反范式形式呈现。

Sales 表

#	列名	数据类型
1	OrderId	INTEGER
2	OrderDate	STRING
3	Quantity	INTEGER
4	SalesAmt	DOUBLE
5	Profit	DOUBLE
6	CustomerName	STRING
7	City	STRING
8	State	STRING

9	ZipCode	STRING
10	Region	STRING
11	ProductId	INTEGER
12	ProductCategory	STRING
13	ProductSubCategory	STRING
14	ProductName	STRING
15	ShipDate	STRING

基于 SQL 和 MapReduce 的实现

我们提供了一个查询的样例集，这些查询使用聚合函数、过滤条件和分组从句，及其等效的 MapReduce 实现，即 MongoDB 实现 SQL 中 GROUP BY 的等效方式。在 MongoDB 存储的文档上执行聚合操作非常有用，这种方式的一个限制是聚合函数（比如，SUM、AVG、MIN、MAX）需要通过 mapper 和 reducer 函数来定制化实现。MongoDB 没有原生态的用户自定义函数（UDFs）支持。但是它允许使用 db.system.js.save 命令来创建并保存 JavaScript 函数，JavaScript 函数可以在 MapReduce 中复用。下表是一些常用的聚合函数的实现。稍后，我们会讨论这些函数在 MapReduce 任务中的使用。

聚合	Javascript 函数

SUM	db.system.js.save({ _id : "Sum" ,
	value : function(key,values)
	{
	var total = 0;

	for(var i = 0; i < values.length; i++)
	total += values[i];
	return total;
	}});

AVERAGE	db.system.js.save({ _id : "Avg" ,
	value : function(key,values)
	{
	var total = Sum(key,values);
	var mean = total/values.length;
	return mean;
	}});

MAX	db.system.js.save({ _id : "Max" ,
	value : function(key,values)
	{
	var maxValue=values[0];
	for(var i=1;i

MIN	db.system.js.save({ _id : "Min" ,
	value : function(key,values)
	{
	var minValue=values[0];
	for(var i=1;i

VARIANCE	db.system.js.save({ _id : "Variance" , value : function(key,values)

	{
	var squared_Diff = 0;
	var mean = Avg(key,values);
	for(var i = 0; i < values.length; i++)
	{
	var deviation = values[i] - mean;
	squared_Diff += deviation * deviation;
	}
	var variance = squared_Diff/(values.length);
	return variance;
	}});</br>

STD DEVIATION	db.system.js.save({ _id : "Standard_Deviation"
	, value : function(key,values)
	{
	var variance = Variance(key,values);
	return Math.sqrt(variance);
	}});</br>

SQL 和 MapReduce 脚本在四种不同的用例场景中实现聚合函数的代码片段如下表所示。

1.各地区的平均订单量

下面的查询是用来获取不同地区的平均订单量。

SQL Query	MapReduce Functions
SELECT	db.sales.runCommand({

	mapreduce : "sales" ,

City,
State,
Region,
	map:function()
	{ // emit function handles the group by
	emit({
	// Key
	city:this.City,
	state:this.State,
	region:this.Region},
	// Values
	this.Quantity);
	},

AVG(Quantity)	reduce:function(key,values)
	{
	var result = Avg(key, values);
	return result;
	}

FROM sales	
GROUP BY City, State, Region	// Group By is handled by the emit(keys, values) line in the map() function above
	out : { inline : 1 } });

2.产品的分类销售总额

下面的查询是用来获取产品的分类销售额，根据产品类别的层级分组。在下面例子中，不同的产品类别作为个体维度，它们也可以被称为更复杂的基于层次的维度。

SQL 查询	MapReduce 函数
SELECT	db.sales.runCommand({ mapreduce : "sales" ,

ProductCategory,
ProductSubCategory,
ProductName,
	map:function()
	{
	emit(
	// Key
	{key0:this.ProductCategory,
	key1:this.ProductSubCategory,
	key2:this.ProductName},
	// Values
	this.SalesAmt);
	},

SUM(SalesAmt)	reduce:function(key,values)
	{
	var result = Sum(key, values);
	return result;
	}

FROM sales	
GROUP BY ProductCategory, ProductSubCategory, ProductName	// Group By is handled by the emit(keys, values) line in the map() function above

	out : { inline : 1 } });

3. 一种产品的最大利润

下面的查询是用来获取一个给定产品基于过滤条件的最大利润。

SQL 查询	MapReduce 函数
SELECT	db.sales.runCommand({ mapreduce : "sales" ,

ProductId, ProductName,	map:function()
	{
	if(this.ProductId==1)
	emit({
	key0:this.ProductId,
	key1:this.ProductName},
	this.Profit);
	},

MAX(SalesAmt)	reduce:function(key,values)
	{
	var maxValue=Max(key,values);
	return maxValue;
	}

FROM sales	
WHERE ProductId=’ 1’	// WHERE condition implementation is provided in map() function
GROUP BY ProductId, ProductName	// Group By is handled by the emit(keys, values) line in the map() function above

	out : { inline : 1 } });

4. 总量、总销售额、平均利润

这个场景的需求是计算订单的总数、总销售额和平均利润，订单 ID 在 1 到 10 之间，发货时间在 2011 年的 1 月 1 日到 12 月 31 日之间。下面的查询是用来执行多个聚合，比如，在指定年份以及指定的不同区域和产品类别范围里订单的总数、总销售额和平均利润。

SELECT	db.sales.runCommand({ mapreduce : "sales" ,

Region,
ProductCategory,
ProductId,
	map:function()
	{
	emit({
	// Keys
	region:this.Region,
	productCategory:this.ProductCategory,
	productid:this.ProductId},
	
	// Values
	{quantSum:this.Quantity,
	salesSum:this.SalesAmt,
	avgProfit:this.Profit});
	}

	reduce:function(key,values) { var result=

Sum(Quantity),
 	
Sum(Sales),
 	
Avg(Profit)
	{quantSum:0,salesSum:0,avgProfit:0};
	
	var count = 0;
	
	values.forEach(function(value)
	
	{
	
	// Calculation of Sum(Quantity)
	
	result.quantSum += values[i].quantSum;
	
	// Calculation of Sum(Sales)
	
	result.salesSum += values[i].salesSum;
	
	result.avgProfit += values[i].avgProfit;
	
	count++;
	
	}
	
	// Calculation of Avg(Profit)
	
	result.avgProfit = result.avgProfit / count;
	
	return result;
	
	},

FROM Sales	
WHERE	
Orderid between 1 and 10 AND Shipdate BETWEEN '01/01/2011' and '12/31/2011'	**qquery : {** **"OrderId" : { "$gt" : 1 },** **"OrderId" : { "$lt" : 10 },** **"ShipDate" : { "$gt" : "01/01/2011" },** **"ShipDate" : { "$lt" : "31/12/2011" },** **},**

GROUP BY Region, ProductCategory, ProductId	// Group By is handled by the emit(keys, values) line in the map() function above
LIMIT 3;	**limit** : 3,
	out : { inline : 1 } });

既然我们已经看了在不同业务场景下的聚合函数的代码示例，接下来我们准备来测试这些函数。

测试聚合函数

MongoDB 的 MapReduce 功能通过数据库命令来调用。Map 和 Reduce 函数在前面章节里已经使用 JavaScript 实现。下面是执行 MapReduce 函数的语法。

```
db.runCommand(

  { mapreduce : <collection>,

    map : <mapfunction>,

    reduce : <reducefunction>

    [, query : <query filter object>]
```

```
        [, sort : <sorts the input objects using this key. Useful for
 optimization, like sorting by the emit key for fewer reduces>]

        [, limit : <number of objects to return from collection>]

        [, out : <see output options below>]

        [, keeptemp: <true|false>]

        [, finalize : <finalizefunction>]

        [, scope : <object where fields go into javascript global scope >]

        [, jsMode : true]

        [, verbose : true]

    }

)
```

Where the Output Options include:

```
{ replace : "collectionName" }

{ merge : "collectionName"

{ reduce : "collectionName" }

{ inline : 1}
```

下面是用来保存聚合函数并在 MapReduce 中使用的命令。

启动 Mongo 命令行并设置表

- 确保 Mongo 后台进程在运行，然后执行 mongo.exe 启动 Mongo 命令行。
- 使用命令切换数据库：use mydb
- 使用命令查看 Sales 表的内容：db.sales.find()

find 命令的输出如下：

```
{ "_id" : ObjectId("4f7be0d3e37b457077c4b13e"), "_class" : "com.infosys.mongo.Sales", "orderId" :
1, "orderDate" : "26/03/2011", "quantity" : 20, "salesAmt" : 200, "profit" : 150, "customerName" :
"CUST1", "productCategory" : "IT", "productSubCategory" : "software",  "productName" : "Grad",
"productId" : 1 } { "_id" : ObjectId("4f7be0d3e37b457077c4b13f"), "_class" :
"com.infosys.mongo.Sales", "orderId" : 2, "orderDate" : "23/05/2011",  "quantity" : 30, "salesAmt" :
200, "profit" : 40, "customerName" : "CUST2", "productCategory" : "IT", "productSubCategory" :
```

```
"hardware",  "productName" : "HIM", "productId" : 1 } { "_id" :
ObjectId("4f7be0d3e37b457077c4b140"), "_class" : "com.infosys.mongo.Sales", "orderId" : 3,
"orderDate" : "22/09/2011",  "quantity" : 40, "salesAmt" : 200, "profit" : 80, "customerName" :
"CUST1", "productCategory" : "BT", "productSubCategory" : "services",  "productName" : "VOCI",
"productId" : 2 } { "_id" : ObjectId("4f7be0d3e37b457077c4b141"), "_class" :
"com.infosys.mongo.Sales", "orderId" : 4, "orderDate" : "21/10/2011",  "quantity" : 30, "salesAmt" :
200, "profit" : 20, "customerName" : "CUST3", "productCategory" : "BT", "productSubCategory" :
"hardware",  "productName" : "CRUD", "productId" : 2 } { "_id" :
ObjectId("4f7be0d3e37b457077c4b142"), "_class" : "com.infosys.mongo.Sales", "orderId" : 5,
"orderDate" : "21/06/2011",  "quantity" : 50, "salesAmt" : 200, "profit" : 20, "customerName" :
"CUST3", "productCategory" : "BT", "productSubCategory" : "hardware",  "productName" : "CRUD",
"productId" : 1 }
```

创建并保存聚合函数

- 通过 MongoDB 命令行窗口执行如下命令：

```
> db.system.js.save( { _id : "Sum" ,
value : function(key,values)
{
    var total = 0;
    for(var i = 0; i < values.length; i++)
        total += values[i];
    return total;
```

```
}});
```

- 在示例表 Sales 表上执行 MapReduce 程序

```
> db.sales.runCommand(
{
mapreduce : "sales" ,
map:function()
{
emit(
{key0:this.ProductCategory,
key1:this.ProductSubCategory,
key2:this.ProductName},
this.SalesAmt);
},
```

```
        reduce:function(key,values)

        {

            var result = Sum(key, values);

            return result;

        }

        out : { inline : 1 } });
```

输出如下：

```
"results" : [

    {

        "_id" : {

            "key0" : "BT",

            "key1" : "hardware",

            "key2" : "CRUD"

        },

        "value" : 400

    },

    {
```

```
        "_id" : {
            "key0" : "BT",
            "key1" : "services",
            "key2" : "VOCI"
        },
        "value" : 200
    },
    {
        "_id" : {
            "key0" : "IT",
            "key1" : "hardware",
            "key2" : "HIM"
        },
        "value" : 200
    },
    {
        "_id" : {
            "key0" : "IT",
            "key1" : "software",
            "key2" : "Grad"
```

```
            },
            "value" : 200
        }
],
"timeMillis" : 1,
"timing" : {
        "mapTime" : NumberLong(1),
        "emitLoop" : 1,
        "total" : 1
},
"counts" : {
        "input" : 5,
        "emit" : 5,
        "output" : 4
},
"ok" : 1
```

总结

MongoDB 提供了面向文档的存储结构，可以很容易扩展支持 TB 级数据。同时也提供了 Map Reduce 功能，可以通过批处理方式使用类 SQL 函数来实现数据聚合。在这篇文章中，我们描述了安装 MongoDB 并使用 MapReduce 特性执行聚合函数的过程，也提供

了简单 SQL 聚合的 MapReduce 示例实现。在 MongoDB 中，更复杂的聚合函数也可以通过使用 MapReduce 功能实现。

关于作者

Arun Viswanathan Infosys 公司 Cloud Center of Excellence (CoE) 的技术架构师，该公司在 IT 和商业咨询服务上位于全球领先的地位。Arun 在 Java、JavaEE、云计算以及大数据应用架构的定义和实现方面有 9 年半的工作经验。他现在从事大数据解决方案的设计、开发和咨询。Email:Arun_Viswanathan01@infosys.com.

Shruthi Kumar Infosys 公司 Cloud Center of Excellence (CoE)的技术分析师，该公司在 IT 和商业咨询服务上位于全球领先的地位。Shruthi 在 Java、网格计算、云计算以及大数据应用架构上有 5 年的工作经验。她现在从事大数据解决方案的开发和咨询。

Email: Shruthi_Kumar01@infosys.com.

原文链接：http://www.infoq.com/cn/articles/implementing-aggregation-functions-in-mongodb

相关内容：

- ThoughtWorks CTO Rebecca Parsons 专访
- 开发者 Hasen 谈为什么选择 CouchDB
- Poang，基于 Node.js 的自动化测试范例

新品推荐 | News

Blend for Visual Studio 2012

作者 侯伯薇

微软最近发布了新的 Blend for Visual Studio 2012，该工具可以用于在 Windows Store 8 中为 HTML 和 XAML 设计用户界面。Visual Studio 2012 包括快捷版在内的所有版本都会包含 Blend。

原文链接：http://www.infoq.com/cn/news/2012/08/Blend-Visual-Studio-2012

500TB——Facebook 每天收集的数据量

作者 郑柯

相对于 2 年前，Facebook 现在的用户数增加了 127%，达到 9.5 亿，每天收集的数据量也超过了 500TB。本文对 2 年前后的 Facebook 的数据和相关处理基础设施进行了对比，并介绍了他们如何使用 A/B 测试完善网站功能。

原文链接：http://www.infoq.com/cn/news/2012/08/FB-collect-500TB-everyday

F# 3.0 新特性简介

作者 侯伯薇

即将发布的.NET Framework 4.5 会给 F# 3.0 语言带来多项新特性，除了类型提供器（type providers）和 LINQ 查询之类的核心特性之外，还有新的类型属性（type attribute）、三引号字符串常量、自动属性以及对未使用变量的警告等。

原文链接：http://www.infoq.com/cn/news/2012/08/FSharp-3

使用 Kentico CMS 7 轻松构建 ASP.NET 站点

作者 侯伯薇

Kentico CMS 7 引入了一种先进的工作流方法，支持开发者使用新的可视化设计器创建工作流。开发者还可以为每种通知邮件类型指定邮件模板，并在工作流安全性中定义排除的角色。

原文链接：http://www.infoq.com/cn/news/2012/08/Kentico

Napa 简介——基于 Web 用于 Outlook/SharePoint 的 IDE

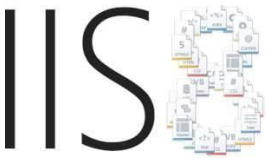
作者 侯伯薇

Office 365 引入了名为 Napa 的基于 web 的 IDE。 这种 IDE 让程序员可以使用 HTML 和 JavaScript 创建 Office 和基于 SharePoint 的插件。 它支持 jQuery 之类的程序库，但那并不是必需的。

原文链接：http://www.infoq.com/cn/news/2012/08/Napa

PhoneGap 2.0 发布

作者 贾国清



Adobe 于近日发布了 PhoneGap 2.0 版本，作为一款开源框架，开发人员可以通过熟悉的 HTML，CSS 和 JavaScript 来开发跨平台的移动应用。PhoneGap 提供的应用开发能力包含了多个移动平台，如 iOS、Android、Blackberry、Windows Phone、Palm WebOS 以及三星 Bada 和 Symbian 等。PhoneGap 2.0 提供了对 Cordova WebView 的支持，Cordova WebView 支持整合 PhoneGap，作为原生应用的一部分。

原文链接：http://www.infoq.com/cn/news/2012/08/PhoneGap-2

Visual Studio 2012 和.NET 4.5 已经就绪！

作者 侯伯薇

在 Windows 8 和 Windows Server 2012 发布生产之后，微软已经完成了他们的工具套件——Visual Studio 2012 和.NET 4.5。

原文链接：http://www.infoq.com/cn/news/2012/08/VS2012-Live

亲密接触 IIS 8 和 Web Deploy 3.0

作者 侯伯薇

IIS 8 是和 Windows Server 2012 一起发布的。它带来多项有趣的特性，像对 NUMA 的支持、WebSockets、安全性改进和更好的 web 部署工具等。

原文链接：http://www.infoq.com/cn/news/2012/08/iis8-webdeploy

PowerDrill，Google 又一个大数据分析大杀器

作者 郑柯

将近十年前，Google 放出的两篇论文催生了 Hadoop。最近，Google 又有两篇论文放出，描述了他们用来处理大数据的利器。其中一篇提到的是 Dremel。前不久，他们在正在举行的 VLDB 2012 大会的论文中提到了 Google 内部使用的一个工具——PowerDrill。论文中说：相比提供类似信息分析功能的传统数据库，该工具要快 10 倍到 100 倍。

原文链接：http://www.infoq.com/cn/news/2012/08/powerdrill-in-memory-column-stor

Terracotta BigMemory 3.7：TB 级支持、改进搜索、增强安全性

作者 丁雪丰

Terracotta Inc.最近发布了 BigMemory 3.7，用于 Enterprise Ehcache 的堆外存储管理单元。BigMemory 通过将数据保存在内存里实现应用程序的提速，这么做避免了大 JVM 堆常见的垃圾回收时间较长的情况。该版本新增了对 TB 级服务器的支持，还减少了查询索引的开销并增强了安全性。

原文链接：http://www.infoq.com/cn/news/2012/08/terracotta-bigmemory-3.7

推荐编辑|原创团队编辑 张逸

求得心安

做文字功夫，大约三分靠灵感，五分靠学识，剩下的两分，就得留给才气来炫耀，方才能作得一篇好文。若论技术文章的写作，似乎又不然，没有技术功底，文字再好也白搭。可若是没有足够的语文学识，文字又干瘪瘪的，不耐看。若硬要留下那两分才气，余下的八分就留给学识与技术来分配好了。其实，技术文章没有别的要求，就是要把道理讲通透，讲明白，让读者能够看得懂。技术编辑不需要玩花活儿，不一定要把文章写得花团锦簇，关键还是看你对这门技术掌握得有多透彻，再转而以平实的语句描述出来，足矣。作文与设计一样，真正的大美，还在于一个“简”字。

董桥在《无灯无月何妨》中借一位老先生的话提点作文之道，曰“人贵饱学，文贵简练，古今中外大方之家都靠这两笔本钱讨生活。”道理平实，做起来其实不易。我在 InfoQ 讨生活算起来已有三年春秋，似乎无功无过，但求一个心安。小编的理想不是要改变 IT 世界，也不敢奢望像鲁迅那样希望通过文字唤醒国人的灵魂，无非就是一个知识的传递者，攒出文字让人吸收一点儿养分。我之所谓心安，其实就是一个道德底线。我们这种知识的传递者，就好比餐厅的厨师，把菜炒熟，是你的本分，把菜炒好，是你的本事，不在菜里放地沟油，那就是求得心安了。可惜，很多人连这点心安也不能做到。

不过，一个人的学识总是有限。从前看很多技术著作，作者或译者总会在后面写上“囿于学识，本书或有疏漏，敬请读者见谅”云云。我以为是作译者谦虚，等到自己成了作者和译者，才能体谅那种战战兢兢。于是，自己也学到这个套路，算是一种免责声明。然则，免责声明不是免死金牌，不要因为有了这几个大字做挡箭牌，就可以为所欲为了。那么，该如何衡量这个度？无它，还是一个心安。搜索这几年自己在 InfoQ 发表的新闻文章，有翻译有原创，有洋洋洒洒上万字的大文章，也有豆腐干一般的简短小文，也有采访——采访别人，又被别人采访。若要算字数，可能也有十余万字了吧。这些文字，这些观点，一定会有疏漏，一定会有错误，但却一定是在那个时刻，以我的学识我的功底我的精力我的能力所能做到最好的。

做个心安的编辑，我觉得已经是我最大的成就了。

封面植物

翠柏

常绿直立灌木，分枝硬直而开，小枝茂密短直。叶披针状刺形，长 6～10mm，3 枚轮生，两面均显著被白粉，呈崔蓝色。果实卵圆形，长 0.6cm，初红褐色逐变为紫黑色；内具种子 1 粒。常绿乔木，高 15-30 米，胸径达 1 米；树皮灰褐色，呈不规则纵裂；小枝互生，幼时绿色，扁平，排成一平面，直展，叶鳞形，二型，交互对生，4 片成一节，长 3-4 毫米，中央一对紧贴，先端急尖，侧面的一对折贴着中央之叶的侧边和下部，先端微急尖(幼树之叶呈尾状渐尖)；小枝上面的叶深绿色，下面的叶具气孔点，被白粉或淡绿色。雌雄同株，球花单生枝顶，着生雌球花的小枝圆或四棱形，长 3-17 毫米，弯曲或直。球果当年成熟，长圆形或椭圆状圆柱形，长 1-2 厘米，直径约 5 毫米，成熟时红褐色，具 3-4 对交互对生的种鳞，种鳞木质，扁平，先端有凸尖，下面 1 对小，微反曲，上面 1 对结合而生，仅中部的种鳞各生 2(1 稀粒种子)种子 1 个短翅和 1 个与种鳞近等大的翅，种翅膜质。

种群现状：渐危种。翠柏主要分布于云南中部及西南部，间断分布于贵州、广西及海南的个别地区。生于产通方便及村镇附近山坡、山麓的翠柏，常被砍伐作材用或薪柴，森林面积已逐渐缩减。

保护价值：翠柏属仅两个古老残遗的种，间断分布于北美与中国，我国台湾还有其变种——台湾翠柏 var.formosana(Florin)Cheng et L.K.Fu，对研究植物区系有重要价值。林质优良，生长快，枝叶茂密而浓绿，可作为分布区内荒山造林树种和城镇绿化与庭园观赏树种。本种在国内及贵州的分布区均十分狭窄，种群数量稀少；翠柏高大挺拔，叶色苍翠欲滴，树形美观，木材清香，是极好的庭园观赏树种，还可作建筑、板料、桥梁、家具等用材。

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

勘误

感谢热心读者 Fan Deng 对 《架构师 8 月刊》第 82 页 jQuery 介绍内容的纠正，

正确应为：

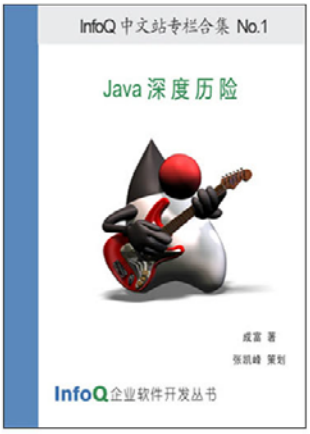
> 近日，jQuery Core 团队发布了 jQuery 1.8 Beta 1。其代码位于 jQuery CDN 上，地址是 http://code.jquery.com/jquery-1.8b1.js。GA 版预计于这个月发布。jQuery Core 团队还谈到了他们对于 jQuery 下一个版本 1.9 与 2.0 的规划，并且提到将要移除对 IE6/7/8 的支持。

1kg.org 多背一公斤

爱自然 | 更爱孩子

InfoQ软件开发丛书

欢迎免费下载



商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

架构师 9 月刊

每月 8 日出版

本期主编：贾国清

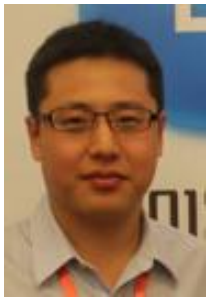
美术、流程编辑：水羽哲

总编辑：贾国清　　发行人：霍泰稳

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：@InfoQ



本期主编：贾国清，InfoQ 中文站主编

毕业于北京工业大学，土木工程本科，硕士转行投入软件工程与系统设计方向。热爱生活，喜欢旅游和体育运动，热衷于关注苹果（Apple）产品的动向和使用，现主要负责 InfoQ 中文站的内容和活动策划以及商务项目执行等工作，内容上尤其专注 HTML5、移动开发等方面。

《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。