

# 架构师

1月 ARCHITECT



特别专题

电商很忙

- “中国规模” 负载背后的技术支撑
- 淘宝开放低耗服务器设计规范
- 淘宝双十一架构优化及应急预案

- 聊聊并发（五）：原子操作的实现原理
- 视图模型（View-Model）到底是什么？
- 使用并行计算大幅提升递归算法效率
- 用HTML5/Java开发客户端/服务器应用

## 卷首语

### 电商很忙

在 2011 年 11 月刊《架构师》上，InfoQ 组织了题为“光棍节狂欢的背后——电商系统初探”的专题内容，主要从电商系统演进、购物袋实现和 API 设计等内容展开。时隔一年，电商系统究竟在技术支撑乃至运维上又有哪些提升和突破呢？

以淘宝为例，双十一当天，天猫交易系统的峰值发生在第 1 个小时，当时系统每秒成功处理了 1.3 万的下单请求。系统峰值 QPS（每秒查询数）为 4 万/秒，系统的平均响应时间在 200 毫秒。天猫的商品详情页面当天的系统访问数高达 16 亿次，峰值吞吐率达到了 6.9 万次访问/秒，并且在峰值时还保持了 12 毫秒的高响应能力。天猫搜索双 11 当天 PV 高达 5.9 亿，峰值吞吐率到了 1.4 万次访问/秒。淘宝和天猫面对这样的业务压力，又是如何应对的？

本期专栏中，《双 11 后续报道：“中国规模”负载背后的技术支撑》将向您展示天猫如何通过业务降级来应对高负载的情况，以及水平扩容和应急决策上的方案。此外，还谈到了淘宝和天猫的开源策略，大量代码都已在 <http://code.taobao.org> 开源。包括远程通信框架 HSF、消息中间件 Notify 和数据访问中间件 TDDL 在内；在《淘宝双十一架构优化及应急预案》中，可以了解到导购系统静态化改造、优惠系统架构改造以及支付核心路径异步化上的一些考量。

本期的任务专访栏目，《Go 语言编程》作者 Mark Summerfield 将谈到他对 Go 语言的见解以及新书的一些内容。热点新闻涉及一直在社区被热烈讨论的话题《开发者应该开始学习 C++ 吗？》以及关于 HTML5 的话题《HTML5 性能之争——单线程：缺点还是特点？》和《关于 HTML5 的 5 个误解》。本期深度文章推荐《聊聊并发(五)——原子操作的实现原理》、《视图模型( View-Model )到底是什么》及《James Ward 谈使用 HTML5 和 Java 开发客户端/服务器应用》。

**本期主编 贾国清**



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 | .....

# 目录

## [卷首语]

电商很忙 .....	2
------------	---

## [人物专访]

Go 语言编程 .....	5
---------------	---

## [热点新闻]

开发者应该开始学习 C++ 吗 ? .....	9
-------------------------	---

HTML5 性能之争 —— 单线程 : 缺点还是特点 ? .....	12
------------------------------------	----

修改一行代码需要 6 天时间 ? .....	16
------------------------	----

关于 HTML5 的 5 个误解 .....	18
------------------------	----

HTTP 2.0 首个草案发布 .....	21
-----------------------	----

## [特别专题]

双 11 后续报道：“中国规模”负载背后的技术支撑 .....	24
---------------------------------	----

淘宝开放低耗服务器设计规范 .....	27
---------------------	----

淘宝双十一架构优化及应急预案 .....	33
----------------------	----

## [推荐文章]

聊聊并发 (五) —— 原子操作的实现原理 .....	36
-----------------------------	----

视图模型 ( View-Model ) 到底是什么 ? .....	43
-----------------------------------	----

使用并行计算大幅提升递归算法效率 .....	55
------------------------	----

James Ward 谈使用 HTML5 和 Java 开发客户端/服务器应用 .....	62
---	----

## [新品推荐]

---

使用 Apache Ambari 管理 Hadoop .....	67
Eclipse Orion : 基于浏览器的 Web 应用程序编辑器 .....	67
Lucene.Net:一个顶级 Apache 项目和它的未来.....	67
移动站点生成便捷之路 : 百度 SiteApp .....	67
JetBrains 发布 IntelliJ IDEA 12 .....	68
ASP.NET 实现了更好的加密算法 .....	68
Atmosphere 1.0 : 支持 Java/JavaScript 的异步通信框架 .....	68
Orubase : 为 Windows Phone、Android 和 iOS 平台开发混合本地手机应用程序 .....	69
Oracle 表彰伦敦 Java 社区 , Gosling 现身 JavaOne .....	69
Java 8 for Raspberry Pi 开发者预览版 .....	69
RavenDB 性能提升、增加了工具与异步 API 支持.....	70
GE Energy 利用 InvokeDynamic 指令将 Magik 移向 JVM .....	70

## [封面植物]

南湖柳叶菜.....	71
------------	----

# 人物专访 | Interview

## Go 语言编程

作者 Jeff Martin 译者 侯伯薇

Go 语言是由 Google 在 2009 年 11 月份公布的，它的目标是要应对软件开发所面临的最新挑战。Go 语言特别被设计为快速（包括在编译时）、支持多核的语言，并且兼顾了动态语言的简单性和静态类型语言的安全性。Mark Summerfield 最近出版了《Go 语言编程》一书，目的是要帮助当前的程序员学习 Go 语言。InfoQ 最近有幸和 Mark 一起讨论了 Go 语言和他的著作。

**InfoQ：你喜欢 Go 语言的哪些特点？**

“Mark Summerfield：Go 语言有很多特点我都非常喜欢，其中最主要的是：像闪电一样快的编译。这使得编辑/编译/运行的周期和 Python 的编辑/运行周期一样快。

非常高级的并发。你可以很轻松地使用 Go 语言编写并发程序，而不会有任何显式的锁。另外，goroutine 通过操作系统线程多路传输的方式，这意味着，如果你的算法最好以成千上万个并发线程来表示，那么你就可以创建那么多 goroutine——而对于线程，通常最好不要创建过多。

无初始化和垃圾回收。这让我们避免了整整两类错误的发生，让编码更简单。

语言本身非常小，让一般的程序员就可以掌握。当考虑模板语言的时候，C++98/03 已经不是一般程序员所能接受的，而 C++11 更大，也更复杂。与它们相比，Go 语言：

使用了新奇的方式来实现面向对象。我发现这种方法很有趣。

对 Unicode 的支持。我非常喜欢 Go 语言让你可以使用原生 UTF-8 或者使用根据你想要的来使用 Unicode 字符的方式。

**InfoQ：你不喜欢 Go 语言的哪些特点？**

“**Mark:** 最初我不喜欢它的错误处理方法（返回错误值作为唯一或者最后一个返回值），因为我不习惯使用这样的异常处理方式。然而，现在我非常乐于使用它了。

我还忘了说操作符重载。IMO 这个大程序包( 针对 big.Int 和 big.Rat 类型 ) 很难使用，因为你无法对操作符重载。另外，尽管 Go 缺少泛型，但那只是针对于类库编写者的问题。由于 Go 拥有其它语言特性，所以不太需要泛型，比方说它对 “<” 的操作符重载对于定义自定义的数据类型就非常好。

**InfoQ :从你作为作者的经历看，我发现你已经撰写了好几本关于 Python 的书。这两种语言相比，你的感觉如何？ Go 的那些方面让你想念 Python ？ 相对而言，Go 的哪些方面让你可以忘记 Python ？**

“**Mark:** 我要说，总体上看来，Python 3 是我最喜爱的语言。然而，Go 是我最喜欢的编译型语言。

我知道很多 Python 程序员都已经试过 Go，但是从个人来说，如果他们想要的是更好的性能，那么最好使用 numpy，如果想要快速的算法，可以使用 Cython。

Go 想要成为 21 世纪的 C 语言，当然我认为时间会验证，Go 会成为一种非常有影响力的语言。我确信它会导致其他语言至少增加 CSP 类库来支持 Go 样式的并发。

在语言和标准库方面，Go 已经比 C 更令人惊奇了。然而，C 拥有很多很多第三方类库（尽管使用 cgo 包我们也可以在 Go 中使用它们）。

我认为 Go 拥有足够的高级特性，可以吸引那些想要更高性能、更好的并发模型以及更自由许可的 Java 程序员。

我还认为 Go 可以替换 C++ 11，为 C++ 程序员提供很多内容。Go 的标准类库比 C++ 更小——但是实际上包含了更多实用的现实功能。当然，C++ 程序员会发现 Go 语言没有它们所习惯的一些东西——但是在几秒钟或者几分钟内完成编译，而不是花费几十分钟甚至几个小时会让我们更有生产力，并且 Go 要比 C++ 更易于维护，因为它是一种更小、更易于理解的语言。当然，Go 并不是 C++，在三点上还是有区别的( C 预处理器，它没有 scope 的概念；C++ 运行时语言；C++ 编译时语言，即模板 )。

此外，Go 程序会编译成单一的可执行文件，而没有任何外部依赖。这使得部署——比方说，在一个组织中所有计算机上——更容易，不需要担心所有计算机是否都拥有相同的类库。

Go 语言即让大家感到熟悉（在 C 家族中），并且还在关键问题上（面向对象、并发、错误处理、轻量级语法）采取了非常先进的标准，这使得 Go 非常值得学习，至少我认为是主流编程中可选的一种方法。

**InfoQ：非常感谢你对 Go 做出的与其他语言的比较，特别是描述了与 C++11 相比，它是如何降低了复杂度。你是否发现 C++11 也做出了足够重大的变化，会导致现有的程序员在拥有足够自由度的时候考虑转换到 Go 语言？**

**“** **Mark:** 尽管我认为从 C++98/03 到 C++11 的发展历程表示有机会可以重新评估并转换到 Go 语言，但是我认为大多数人实际上不会那么做。C++需要在学习上投入很多。我猜想很多程序员会继续使用 C++11 的子集，而不是更换语言，特别是对于比 C++11 小的语言（那样看起来特性也比较少）。当然也会有些人转换到 Go 语言，特别是那些拥有能够从 Go 的高级并发模型中受益的应用程序的人。

使用 C++98/03 我们已经有一种形势，很多程序员使用语言的特定子集，让他们感到很舒服。当两位或更多使用不同子集的 C++ 程序员必须一起工作，或者维护不是他们编写的 C++ 代码的时候，就会导致问题。我认为 C++11 会显著地放大这个问题。

Java 的设计要比 C++ 更简单，并在很多方面都很成功。Go 是要设计成更好的 C，所以我期望它会赢得更多 C 程序员（对他们来说全都是收获）而不是 C++ 程序员（对他们来说既有收获，也会失去一些特性，像泛型）。

**InfoQ：你对 Go 语言的简洁性大加赞赏，但是否有一些你认为应该增加的特性呢？（可能会期望增加操作符重载和泛型。）**

**“** **Mark:** 我认为你可能需要操作符重载，而不需要泛型。但是我认为唯一真正缺少的是排序的映射表（ordered map）（尽管我在书中提供了对左倾红黑树的 Go 实现作为例子，它也填补了特定的空白）。

**InfoQ：你在书的目录中还特别说明了专利流氓的危险。是什么让你包含这项内容呢？你是否认为有必要提醒大家意识到这个问题，或者是特定的什么事情？**

“Mark: 我一点儿也不喜欢专利：我认为那是反竞争的，并且大公司会比小公司更喜欢它。对于软件，版权就已经足够了。我之所以包含了附录，是因为现在我已经开始听说有些程序员开始从美国的市场（例如编写 iOS 程序的人们）撤回软件，因为不那么做的话，就需要给专利流氓支付“保护费”。软件专利使得个人程序员或者小团队更难竞争，我认为这不仅仅是可耻，而且在道德上是错误的。

在过去的几年间，我对商业化软件有一些看法：利基（niche）产品会很有用，而不是大规模的销售团队。然而，我并不准备付出时间，因为我主要的市场还是美国，对于我来说冒着花费时间和精力的风险，而只是让专利流氓能够“偷到”更多东西，是非常不经济的。

## 关于书的作者



**Mark Summerfield** 以一等毕业生从威尔史旺西大学的计算机科学专业毕业。在工作之前，他继续做了一年的研究生。他从事软件工程师职业多年，在加入 Trolltech 之前在多家公司中工作过。他还作为 Trolltech 的文档管理员工作了将近三年，在那期间他编辑了 Trolltech 的技术杂志《Qt 季刊》。Mark 是《C++ GUI Programming with Qt 3》和《C++ GUI Programming with Qt 4》的共同作者，并且是《Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming》的作者。Mark 是 [Qtrac Ltd.](#) 公司的老板，在其中他作为独立作者、编辑、培训师和顾问工作，专注于 C++、Qt、Python 和 PyQt。

原文链接：[http://www.infoq.com/cn/articles/programming\\_in\\_go](http://www.infoq.com/cn/articles/programming_in_go)

### 相关内容：

- [Go，互联网时代的 C 语言](#)
- [2012.4.18 微博热报：QCon 北京大会开幕](#)
- [Google Go 语言推出第一个正式版本：Go 1](#)
- [Nodejs，脱离了浏览器的 Javascript](#)
- [Google App Engine 涨价，开发者深受打击](#)

## 热点新闻 | News

### 开发者应该开始学习 C++ 吗？

作者 [Jonathan Allen](#) 译者 [臧秀涛](#)

随着 [C++ 11](#) 和 [C++ CX](#) 的引入，很多人重新燃起了对这门语言的兴趣。不少开发者，尤其是 Windows 开发者，都想知道是否应该放弃 C# 和 Java，转而支持 C++。John Sonmez 认为这并不需要。

在 “为什么 C++ 并没有 ‘王者归来’（[Why C++ Is Not ‘Back’](#)）” 一文中，John Sonmez 认为只有如下三个原因才会使用 C++：

- 需要榨干软件每一寸可能的性能，并且想用支持面向对象抽象的语言来实现。
- 编写直接面对硬件的代码。（例如，编写底层驱动。）
- 内存控制与定时极为重要，因而系统的行为必须是完全确定的，还必须能够手动管理内存。（想一下控制机器移动部件的嵌入式实时操作系统。）

[Herb Sutter](#) 高度称赞了这篇文章，认为文中的“观点有些深度，没有夸张”。关于 C++ 的应用场景，他又做了一些补充：

- 服务，依赖于运行时会更为困难。
- 测试，对比一下全部或者大部分采用静态链接的应用程序与在最终用户机器上往往是首次执行时才编译或即时编译（JIT）的应用程序，后者无法完整地测试。

John Sonmez 反对学习 C++，过于复杂是原因之一。即使 C++ 11 让开发容易了一些，但是程序员仍然不得不学习各种老式的 C++ 编码方法。“你会碰到 20 年前的 C++ 代码，看起来就像是完全不同的语言。”为了加强其观点，他向准备应聘 C++ 职位的开发者提出了 36 个问题。下面列出几条：

1. 在 C++ 中，基本数据类型有多少种初始化方式？你能都说出来吗？
12. 什么是复制构造函数，何时会用到？尤其是与赋值操作符相比，你能区分吗？

16.在 C++ 中，何时适合通过引用来返回值，何时不适合？

33.为什么绝对不应该在析构函数中抛出异常？

反对 C++ 的另一个理由是“编程语言真正需要的是简化并提高抽象层次，而不是反其道而行之”。他继续道，

“编写底层代码的需求总是存在的，但我们今天编写的大部分都是较高层次的代码。

很多年前，当我终于无法再坚持认为我用 C++ 开发应用的速度比 C# 快时，我跳下了 C++ 这条船。

我坚持良久，试图让自己相信我在 C++ 上的所有投入并没有白费，但是事实证明，C# 带来的简化是如此之大，以至于与此相比，C++ 所提供的额外的力量并不值得这些额外的付出。

在文章结尾，John Sonmez 说到，学习 C++ 对于理解计算机的一般工作原理仍然是有用的，“但是我认为 C++ 不会东山再起，这是好事”。

关于这一点，Alo 补充到：

“我是从 C++ 开始的，而且我职业生涯的前四年都花在了 C++ 上。这种经验对我非常有价值，正如您的文章中所指出的那样，因为一旦把 C++ 学到了足够的水平，就可以很快地捡起其他任何语言；此外，还能从一个更低的层次上更深刻地理解软件工作原理——如果从其他层次更高的语言开始学习编程，获得这种知识的难度就大多了。正因如此，我一直不赞成让程序员从 Java 开始学起。

Richard Dunks 反驳到：

“我认为，在第一学期的程序设计导论课程和数据结构的教学中，C++ 是没什么帮助的，因为光实现就要耗费很多时间，反而让同学们忽略了他们要复现的结构。我很高兴自己能够精通 C++，但我认为这并不值得，而且 C++ 绝对不是一门万能的教学语言。

Stephen Cleary 有一条评论谈到了可重用性：

“我原来是 C++ 开发者，几年之前，市场的压力让我成了一名 C# 开发者。C# 的确更有生产率，但是完全不可能实现 C++ 模板那种级别的代码复用。

经典的例子就是容器、迭代器和算法这三驾马车。在 C++ 中，能够创建一个用于任何容器的算法，而且可以在编译时对算法加以调整以便必要的情况下利用随机访问能力。你可以用 C# 试试。这还是尚未谈到“新 C++”的情况；1998 年的 C++ 对代码复用的支持就比现在的 C# 好了。

关于性能，Herb Sutter 给出了如下建议：

“在任何语言中，如果非常关注性能，都会大量使用数组（未必“总是”使用，只是“大量”用到）。不过这在有些语言中很容易，可以很好地控制一般内存布局，特别是控制数组；而在其他语言或环境中就困难一些（有可能让你使用，但更为困难），如果这些语言或运行时特别偏爱通过指针构造的数据结构，你就不得不“放弃”或者“尽量避开”。

除了在 Herb Sutter 和 John Sonmez 的相关博客上的大量高质量评论，Reddit 的 [Programming](#) 和 [Coding](#) 子群组也有很多可以学习的东西。

**原文链接：**<http://www.infoq.com/cn/news/2012/12/Learning-Cpp>

**相关内容：**

- [将现有 C++ 代码移植到 Windows 8/Windows Phone 8](#)
- [微软的新编译器增加了对 C++11 特性的支持](#)
- [使用 C++/CX 开发 Windows Store 应用程序的注意事项](#)
- [Embarcadero 更新 Delphi 和 C++ Builder，发布 HTML5 Builder](#)

## 热点新闻 | News

# HTML5 性能之争 —— 单线程：缺点还是特点？

作者 [彭超](#)

2012 年对 HTML5 是多事的一年。年初各界人士对 HTML5 进行了乐观的畅想。而 Facebook 在进行了长时间的尝试之后，8 月份发表声明，在 iOS 平台转向 native 开发。昨天（12 月 14 日），facebook 发布了 Android 平台的原生应用。Facebook 在官方博客中表示，新版 Android 应用在显示照片和加载时间线条目时速度是此前版本的 2 倍。

Facebook 的早期员工，中国籍第二位工程师和第一位研发经理王淮 Harry ([微博](#))在他的博客文章《[HTML5 的明天，局部有小雨](#)》中谈到了该不该给 HTML5 投怀送抱这个问题。他表示，这个问题需要分两步：

“ 你在 WEB 端还是移动端？

对于 Web 端而言，HTML5 将是一个完整的操作系统。它在不同的底层系统之上，借助于浏览器的实现，封装了统一标准的 API 允许开发的程序跨设备(PC or Mac or Smart Phone)，跨平台 (Windows，MacOS，iOS，Android，whatever)的运行。最大的好处，就是一处开发，多处使用。审核新版本的发布也不用看苹果爷爷的脸色。直接在服务器端推送新代码就好了。对于开发人员而言，这对效率的提高，有着致命的诱惑。像“你们是先开发 Web，还是移动”之类的问题，将愉快的失去意义。对于 Web 端的开发而言，你可以尽情的享受 HTML5 这种统一封装带来的好处，唯一要等待的就是浏览器对其支持的完善。但这种完善的到来，无疑是确定的。

如果是移动端，取决于你的产品形态。

因为你的产品需要的功能可能永远也无法在移动端的浏览器的 HTML5 实现中被很好的实现。

Harry 表示：

“

App Store 上超过 50%的应用已经是用 HTML5 来开发，将来可能 90% 的应用会是 HTML5，而那 10%，可能永远也不适合 HTML5。

在分析情况之后，他介绍了一个动态检测浏览器对 HTML 支持程度的测试工具 [ringmark.io](#) 来帮助大家判断自己的位置，并道出了为什么 HTML5 在移动端不适用的原因：

照片分享，浏览相关的功能极度依赖 CSS Overflow Scrolling，这一点，iOS 上的浏览器支持极度不给力。而换成 Object C 的 Native Implementation 之后，速度快上了 2 倍之多。

最后，Harry 从浏览器的编程模型阐述了问题的成因，并给出了技术选型的策略。

“ 浏览器的编程模型还是 90 年代流行的单进程单线程 (single process single thread)，但原生实现(比如用 Object C)的 APP 可以用多线程。这一点带来的作用是致命的。

移动端编写 APP，可以使用多个线程，第一个线程，被称作主线程(main thread)，编程的第一原则是 don't do heavy work on main thread。通常只让它处理 UI 事件等，其他重度的工作让其他背景线程来做。

但浏览器只有一个线程，所有的事情都是它干。移动浏览器编程一上来就破了第一原则。

在台式机上，浏览器编程还没有太多问题，因为够快(此外，桌面浏览器对于 Web Worker 的支持也比较完善)；但在移动端，这个弊端很明显。

我来举个例子，比如你在用浏览器看朋友的照片，你发的评论被发到服务器端，此时你接着用手指往下拉屏；此时，服务器端返回信息，评论发布成功，浏览器中唯一的线程可能停止处理屏幕滚动(scrolling)而来处理服务器的返回信息，由于移动设备的处理器(尤其单进程浏览器只能用上单核，即使是多核手机！)和内存(处于省电原因使用低耗电的 DDR1，这一点和现在 PC 使用的 DDR3 相差甚远)的不给力，完全可能造成滚动处理的不连续。通常手机的刷新率是 60MHz，即每一帧不超过 15ms；如果处理的延时大大超过 15ms，那么就会出现跳帧，肉眼就能看出来。

这是交互操作(比如拉动，滚动等)很多的 APP，如果是由 HTML5 实现，出现拉动的时候停在那里一个很重要的原因。

所以，如果你的 APP 是相对静态的，不需要很多对于照片，多点触摸，多向拉动的处理，那完全可以用 HTML5 来实现；如果不是，比如信息流的展示，游戏等等，还是乖乖的用原生的去实现。

故事到这里还没有结束。HTML5 开发专家大城小胖（[微博](#)）专门撰文《[HTML5 与“性能”障碍](#)》回应了 Harry。在文中他赞同了 Harry 的观点：

- 在移动端是否采用 HTML5 技术，取决于你的产品形态。
- 将来可能 90% 的应用会是 HTML5，而那 10%，可能永远也不适合 HTML5。
- HTML5 性能的提升很大程度上将取决于低耗电高性能 CPU/内存的出现，或者电池技术的极大改善。

但他认为，HTML 不支持多线程不是缺点，而是特点。

“要提高 HTML5 的性能，不应该靠“引入多线程”来实现，应该靠“提升单线程内处理每一项时的性能”（以及如何将大量的第一项里的工作分解）来实现。而 WebWorker 的引入，其实就是为了提高第 1 项的性能。

WebWorker 本身并不是传统的 Thread 模型，虽然底层是多线程实现的，但是它并没有引入同步锁、线程调度一类高级特性，而是用简单的消息机制尽可能的保持了和单线程之间的匹配度。换言之，WebWorker 并不是给单线程的 HTML 带来的多线程特性，而是给单线程的 HTML 带来了后台计算的能力。

最后，大城小胖指出了 HTML5 现在最重要的三个问题：性能、工具和能力：

“性能低下，这个事情基本上大家能够达成共识，至少和各种强大的 Native 展现层技术相比，确实有差距。但是这个问题不是致命的根本性问题。当年 Doom3、孤岛危机 1 这些游戏出来时，也都是当时的硬件杀手，但是后来随着硬件的提升，性能问题也渐渐不再是核心问题了（这两个游戏在 FPS 游戏里，是真不好玩啊）。换言之，当市场上对性能要求高的好的产品和应用越来越多时，那些底层（浏览器、os、硬件）厂商不会坐视不理的，因为这对于他们来说也是机会。所以作为 HTML 前端工程师，我们所要做的就是尽可能的优化自己的代码，但不要被性能束缚了产品的手脚，同时在保证

自己代码质量和算法没问题的情况下，行动+呼吁+等待就 OK 了。当然，不要强迫 HTML5 去做不应该它来做的事情。

工具匮乏，从开发调试到测试维护整个过程中，确实缺少强有力的工具。这个问题在可预见的未来，应该还是比较难解决，不过对成熟的 HTML 开发团队而言，似乎也不是大问题，因为大家已经比较习惯和适应现在的开发环境和方式了。但是对于围绕上层应用所需要的辅助工具确实欠缺。拿 HTML5 游戏来说：地图编辑器、精灵编辑器、粒子效果、游戏脚本编辑器、音效管理工具、性能监控。。。等等，虽然理论上开发这些并不难，很多公司也都在尝试开发自己的基础架构，但是和 unity3d flash 这些比起来，还是太弱了。期待 cocos2d-x 能够为我们带来不一样的局面(此处为植入广告，请林顺 王哲自行考虑所需费用)。总之，HTML5 为 web 应用带来了更多新的形式，不过围绕这些新形式的相关辅助工具确实还很欠缺。但是，未来可期。

能力不足，这个主要是指 HTML5 本身的定位和它的原则导致很多事情是它根本做不了的。举个极端点的例子，你希望在网页里借助 HTML5 技术来格式化你的 U 盘、刻录一张 CD 几乎是不可能的（谁知道 HTML6789 时会不会提供一组 Disk API 呢？）。因为很多事情根本就不在 HTML 的发展蓝图里。而且浏览器根本的目的是为了保证用户可以高效便捷安全的网上冲浪，这个根本目的导致了浏览器本身会存在一些制约，例如安全性上的。所以，指望 HTML5 能完全取代 Native 是不可能的，至少我觉得在我退休之前是不可能的。

各位读者，你们怎么看待 HTML5 的性能问题，适用场景和它的未来呢？

**原文链接：**<http://www.infoq.com/cn/news/2012/12/html5-performance>

## 相关内容：

- [HTML5 技术在移动领域的发展未达预期](#)
- [按需搭建灵活多变的 Web 应用 —— 以豆瓣阅读计划为例](#)
- [移动 WebApp 的新平台 —— UC 技术总裁梁捷的采访](#)
- [PhoneGap 2.0 发布](#)
- [朱永光谈 WP7 开发特性与应用场景 \(二\)](#)

## 热点新闻 | News

### 修改一行代码需要 6 天时间 ?

作者 黄玲艳

修改一行代码需要 6 天时间 ,你信吗 ?[这篇文章](#)的作者给我们讲了一个真实的故事。

首先我们来看一下有哪些人物 :

Philip : President , 会长

Lee : Operations Manager , 执行经理

David : IT Director , IT 总监

Judy : IT Admin , IT 管理员

Ed : programmer , 程序员

Shirley : Code Review , 代码复查人员

Julie : IT Testing , IT 测试人员

Joe : IT Security , IT 安全人员

Tony : IT Testing , IT 测试人员

故事是这样的 :

Philip 认为工厂的 10% 未得到充分使用 , 要么选择有更多积压 , 要么解雇员工 , 因此询问 Lee 的建议。Lee 建议将积压的时间从 3 个月以上改为 4 个月以上 , 而这个修改 , 也许只需要修改传统软件中的一项配置就可以。接下来 , 将这件事交给了 David , David 同意了 , 并将这件事安排给 Judy , Judy 建了一个单据号 #129281。

两天后 , David 询问 Judy 事情的进展 , Judy 回复需要等开发人员修改玩 14 个 bug 后才能处理 , David 指示将这项任务提前。

1 个小时后 , Ed 修改完代码 , 将某个硬编码属性的取值从 “3” 改成 “4” , Ed 将代码提交给 Shirley 进行审查。Shirley 要求 Ed 对硬编码的参数文件进行登录。

2个小时后，Ed再次提交代码审查。Julie反馈，代码未通过用户接受度测试，并让联系Joe确认。

2个小时后，Joe因为访问授权问题，以及代码命名问题，将Ed的提交驳回修改。

1天后，Ed修改好命名问题后，再次提交代码测试。Tony认为没有给时间写测试用例，无法进行测试，因此无法让代码通过测试进行发布。

2天后，Philip指示David，让Tony通过Ed的代码并发布产品。

至此，总共消耗的时间为6天，重要代码修改了1行，重要代码修改了1个字节。

也许这种故事在我们身边时有发生，常常产品经理会对程序员说，“我这个需求很简单，就把那个放大一点，显示时间久一点，我觉得几分钟就能搞定了”，类似的需求会有很多。结果真的是这样吗？我很想问一下，“元芳，你怎么看？”欢迎大家讨论。

另外，推荐一下作者的博客，有很多有趣的博文分享：

<http://edweissman.com/>，以及作者分享的一本电子书：

[http://v25media.s3.amazonaws.com/edw519\\_mod.pdf](http://v25media.s3.amazonaws.com/edw519_mod.pdf)

**原文链接：**<http://www.infoq.com/cn/news/2012/12/one-line-6-days>

## 热点新闻 | News

### 关于 HTML5 的 5 个误解

作者 程劭非

12月17日W3C的CEO Jeff Jaffe 宣布HTML5 定义完成 ,即进入了Candidate Recommendation 阶段 ,这距离它成为 W3C 的正式推荐标准

( Recommendation 阶段 ) 已经很近。然而技术社区对于 HTML5 仍存在很多误解以至于错误理解这一事件的意义。本文试谈几点常见的误解帮助大家了解 HTML5。

#### HTML5 标准是超炫的技术

HTML5 标准本身是标记语言和语义的规范 ,所以它不会包括诸如 API 和样式这样的内容 ,标记语言和语义这种东西甚至是不可见的 ,当然更不可能 “炫” 了。通常社区中提到 HTML5 所指的是与 HTML5 差不多同时开始制定的一组新的标准 他们包括一些 CSS3, Canvas 2d API, WebGL 等 API 和新特性标准 ,WebGL 甚至并非 W3C 标准。 这一次宣布完成的 HTML5 是 HTML5 标准本身 ,这仅仅意味着 HTML5 的新语法、新标签和语义已经有了稳定的定义 ,不会有大变更。而真正与开发密切相关的一些 API 标准并不在此列。

#### HTML5 是一项新技术

HTML5 所用的技术差不多在 20 年前就已经成熟 ,而 HTML5 本身也并非技术 ,而是标准。即使作为标准 HTML5 也并非新标准 ,而是一个工作了 10 年的标准 HTML4.01 的新版本 ,它是基本向下兼容的。 作为一份标准 ,HTML5 的发布意味着这项技术已经完全成熟并且各大浏览器厂商和其它 W3C 会员达成了一致意见。尽管各方完全实现标准尚需时日 ,但是鉴于 W3C 几乎已经聚集了所有这份标准的相关方 ,所以几乎不会再出现另外的声音了。

#### HTML5 现在还不可用

对于 HTML5，既不应该说“可用”又不能说“不可用”。谈论 HTML5 是否可用是不恰当的，因为如上文所说通常意义所指的 HTML5 包含若干互相独立的技术标准，它们的可用性是相互独立的。诸如 WebGL，WebSocket 这样的标准现在不论在移动还是桌面 Web 环境都几乎完全不可用。而语义化标签和 HTML5 标记语言语法（即 HTML 标准本身所规定的内容）现在则是可用到不能再可用的状态。在这种状态下，更务实地去讨论具体的特性是否可用才是合理和正确的做法，在这方面 <http://caniuse.com/> 做了非常多深入而细致的研究。

## HTML5 现在已经制定完成了

因为一些历史原因，现有的 HTML5 的制订大部分是在 WHATWG 完成。而 HTML5 回到 W3C 之后，WHATWG 将会继续维护一份 HTML 规范文档，这份文档将不会设定完成日期。这意味着仍然可能有大块的特性加入其中，且它永远不会像 W3C 规范一样变得稳定。按照现有的模式，W3C 将会逐步发布 WHATWG 版本的 snapshot 作为 HTML 新规范，在 W3C 的 CEO 宣布 HTML5 “完成”之时，HTML5.1 版本已经开始制订了。

## HTML5 的性能很差

性能问题可能来自多方面，硬件、操作系统、应用代码都可能导致性能问题，然而性能问题唯独与标准关系不大。HTML5 是一个标准，它本身不涉及任何性能。有这样的断言应当是因为现在主流的浏览器实现都在移动端表现不佳，这与浏览器环境本身的复杂性的确密切相关，更涉及到不少 W3C 和 WHATWG 之外的标准（如 JS、WebGL）。应当明确的是，“HTML5 程序性能表现不佳”仅仅是其相对于各个平台的原生应用来讲的，并不是 HTML4.01 性能更好了。比起 HTML 统治了桌面互联网大半江山的前几个版本，HTML5 不论性能还是功能都是全面增强的。HTML 更早版本的产品更新到 HTML5 将会是无需置疑和理所当然的。HTML 在移动领域面临的新的应用场景（Web Application）和面临的问题（性能和功能）则正是 HTML5 出现的原因，也是 HTML5 标准和它的实现者需要解决的问题。

## 作者简介

程劭非/winter，一淘网移动前端技术专家，技术 Blog 是  
<http://winter-cn.cnblogs.com>，微博：[@寒冬 winter](#)。

**原文链接：**

<http://www.infoq.com/cn/news/2012/12/html5-misunderstanding>

## 热点新闻 | News

### HTTP 2.0 首个草案发布

作者 Dio Synodinos 译者 雷慈祥

HTTP 规范的编辑们已经发布了 [2.0 版本的首个草案](#)，它直接复制于 SPDY，并将作为后续改进的基础。很多修改还有待完成，例如添加新特性、移除现有特性以及修改线上文档等。供测试实现使用的草案[预计明年初发布](#)。

HTTP 2.0 由 IETF 的 Hypertext Transfer Protocol Bis( httpbis )工作组开发，这将成为 1999 年 1.1 版本 ([RFC 2616](#)) 发布以来的首个新版本。

HTTP 2.0 的目标包括：

- 异步连接多路复用
- 头部压缩
- 请求/响应管线化

保持与 HTTP 1.1 语义的向后兼容性也是该版本的一个关键目标。

SPDY 是一种 HTTP 兼容协议，由 Google 发起，目前 Chrome、Opera、Firefox 以及 Amazon Silk 等浏览器均已提供支持。

HTTP 实现的瓶颈之一是其并发要依赖于多重连接。HTTP 管线化技术可以缓解这个问题，但也只能做到部分多路复用。此外，已经证实，由于存在中间干扰，现有的浏览器无法采用管线化技术。

SPDY 在单个连接之上增加了一个帧层，用以多路复用多个并发流。帧层针对 HTTP 类的请求响应流进行了优化，因此现在运行在 HTTP 之上的应用，对应用开发者而言只要很小的修改甚至无需修改就可以运行在 SPDY 之上。

SPDY 对当前的 HTTP 协议有 4 个改进：

- 多路复用请求
- 对请求划分优先级
- 压缩 HTTP 头
- 服务器推送流（即 Server Push 技术）

SPDY 试图保留 HTTP 的现有语义，所以 cookies、ETags 等特性都是可用的。

SPDY 中的很多架构方法（如多路复用），[W3C HTTP-NG 工作组](#)曾经进行过早期的探索，但该工作组已于 1998 年暂停。改进 HTTP 这一问题已经讨论了多年，不久前还成了 [InfoQ 的愚人节玩笑](#)。

**原文链接：**<http://www.infoq.com/cn/news/2012/12/http20-first-draft>

**相关内容：**

- [Google 与微软想要改进 HTTP](#)
- [Apache 发布了 HTTP Server v2.4](#)
- [HTTP API 可演进性最佳实践](#)
- [“Apache 杀手” 一种利用 Range HTTP 头域的 DDoS](#)

## 特别专题 | Topic

本期，《双11后续报道：“中国规模”负载背后的技术支撑》将向您展示天猫如何通过业务降级来应对高负载的情况，以及水平扩容和应急决策上的方案。此外，还谈到了淘宝和天猫的开源策略，大量代码都已在 <http://code.taobao.org> 开源。包括远程通信框架 HSF、消息中间件 Notify 和数据访问中间件 TDDL 在内；在《淘宝双十一架构优化及应急预案》中，可以了解到导购系统静态化改造、优惠系统架构改造以及支付核心路径异步化上的一些考量。对于淘宝来说，今年是双十一准备较为充分的一年，前期的功能测试、容量评估、应急预案都比以往做的更加仔细，一共 400 多个应急预案，从 9 月开始进行系统演练。

## 特别专题 | Topic

# 双 11 后续报道：“中国规模”负载背后的技术支撑

作者 [Roopesh Shenoy](#) 译者 [臧秀涛](#)

24 小时之内实现 [30 亿美元的销售额](#)，中国的电商巨头阿里巴巴最近做到了这一壮举。天猫和淘宝在处理这种规模的负载时遇到了哪些挑战，又是如何应对这些挑战的呢？InfoQ 有机会就此向天猫和淘宝的架构师[庄卓然](#)和优昙请教了一些问题。

天猫是中国领先的 B2C 电子商务网站，而淘宝是中国最大的 C2C 在线购物平台，二者都是阿里巴巴集团的子公司，总共有超过 5 亿的注册用户。双 11 大促活动今年已经是第 4 年，UV 数总计达 1.47 亿，实现总商品价值量（Gross Merchandise Volume）191 亿元人民币（大约 30 亿美元）。

面对“中国规模”电子商务的挑战：

“ 在 2012 年 11 月 11 日双 11 大促当天，天猫和淘宝迎接了 1.47 亿的用户访问，3000 万人的购买，产生了近 1 亿笔支付订单。在零点的一瞬间，并发在线的用户数超过了 1000 万。除了满足双 11 的各种功能需求之外，如何在前期准备过程中对系统有一个完整准确的评估，如何有效推进各种优化和容灾预案，如何在活动当天各种紧急情况下正确决策，以及如何保证在顶级流量的冲击下整个网站的稳定、性能和用户体验，这是对技术团队的极大考验。

双 11 当天，天猫交易系统的峰值发生在第 1 个小时，当时系统每秒成功处理了 1.3 万的下单请求。系统峰值 QPS（每秒查询数）为 4 万/秒，系统的平均响应时间在 200 毫秒。天猫的商品详情页面当天的系统访问数高达 16 亿次，峰值吞吐率达到了 6.9 万次访问/秒，并且在峰值时还保持了 12 毫秒的高响应能力。天猫搜索双 11 当天 PV 高达 5.9 亿，峰值吞吐率达到了 1.4 万次访问/秒。

庄卓然解释到，从应用层面上讲，天猫和淘宝的应用都构建于自主研发的服务化架构以及 MVC 框架和 Spring 之上。这是由分布式文件系统、分布式缓存、消

息中间件和 CDN 网络带宽支持的。核心数据库可以通过自主研发的数据访问中间件访问，底层数据库的水平拆分和数据搬运对应用是完全透明的。

基于这种水平扩容架构，面对促销活动引起的流量压力，天猫和淘宝的系统可以灵活地添加机器来应对。

“ 前期我们花了很多时间进行容量计算，对于网站所有应用之间的依赖关系、流量分配比例和应用内部的调用链路做了深入的分析，通过在线的压力测试对各个应用单机的 QPS 进行准确评估，从而达到对网站目前集群处理能力的客观判断。这个过程操作起来实际上是非常有挑战的，因为天猫和淘宝本质上不是一个耦合性很弱的系统，通过单一系统的压测不能很好地反映系统的瓶颈。同时，我们也不可能完全照搬线上的环境和配置一套完整的压力测试环境。所以会更多地依赖线上的压力测试，真实地反映系统的短板。

最后，则是根据网站的自然增长趋势和双 11 的历史数据，评估当天有可能达到的业务指标，再从这些业务指标对各个系统扩容目标进行准确地计算。

当然，仅仅依靠水平扩容方式，对于大促高峰过后的机器利用率是存在弊端的，同时也会大量依赖运维人员的灵活调配能力。因此，今年我们在以聚石塔 (<http://cloud.tmall.com>) 为代表的一些应用中也尝试了大量的弹性计算框架，在塔中很多商家的不同应用共用一个集群的系统资源。双 11 当天弹性升级带宽、VM 和存储资源。同时，我们的很多内部应用也采用了这样的机制。这也是今年在双 11 准备过程中我们在技术上的一个突破。

在双 11 大促的准备过程中，淘宝和天猫的团队对系统进行了针对性的优化，包括 SQL 和缓存命中率的优化、数据库连接和应用服务器参数的调整、JVM 参数的配置、代码的复审和梳理等等。此外，大量固态硬盘 (SSD) 的使用也提高了数据库存储的整体性能。

为了在负载超过预期时关闭非核心操作，团队也准备了业务降级和限流预案。0

“ 所谓业务降级，就是牺牲非核心的业务功能，保证核心功能的稳定运行。简单来说，要实现优雅的业务降级，需要将功能实现拆分到相对独立的不同代码单元，分优先级进行隔离。在后台通过开关控制，降级部分非主流程的业务功能，减轻系统依赖和性能损耗，从而提升集群的整体吞吐率。

当出现了降级还无法缓解的大流量时，就要通过限流的方式来应付。首先从最前端的 Web 应用进行排队，控制流入应用的流量，也就是通过 Web 服

务器的定制模块来实现 QPS 限流功能。根据被保护的 Web 服务器所能承受的最大压力做强制的 QPS 流控，超出 QPS 上限的用户进入排队等待页面。另外，为了避免前端某个 Web 应用出现大规模流量激增时造成后端服务无法承载的雪崩效应，后端的服务会针对低优先级的业务进行限流，以确保不同来源的业务压力不会压垮后端服务，保障核心业务的访问。

针对 2012 年的双 11，天猫和淘宝总共准备了 400 多个系统降级预案。为了保证所有降级和限流预案的准确执行，我们在前期做过了大量的预案演习，所有的应急预案虽然原则上我们一个都不希望使用，但是必须确保每个预案执行的准确性和便捷性。

### 应急决策过程：

“ 双 11 当天，我们有近 400 多位工程师集中办公，确保整个活动的顺利进行。整体流程上我们的目标是希望实现决策的最短路径，并且保证信息传达的简单有效。那么怎么实现呢？首先我们会有一个战地情报分拣中心，负责从客服、运营、安全、产品和商家等不同的信息来源收集和汇总各种用户反馈，剔除重复和无效的反馈，确保不会出现技术团队的信息爆炸。

其次，虽然我们会有一个战地指挥部，但是应急决策的决定权大多数还是交由一线开发工程师的。所有集中办公的工程师分工明确，每个应用都有 1-2 个核心负责人，他们会根据监控中心的各种系统指标变化情况，快速做出应急决策，确保响应的及时性。只有当涉及到对业务影响较大或是对用户体验伤害较大的时候，才会升级到指挥部来进行判断。

淘宝和天猫也有一个有效的开源策略，大量代码都在 <http://code.taobao.org> 开源了。包括远程通信框架 HSF、消息中间件 Notify 和数据访问中间件 TDDL 在内的一些框架已经开源。

### 原文链接：

<http://www.infoq.com/cn/news/2013/01/interview-taobao-tmall>

### 相关内容：

- [从关系数据库向 NoSQL 迁移：采访 Couchbase 的产品经理主管 Dipti Borkar](#)
- [Hazelcast 2.0 发布，推出堆外存储和分布式备份](#)

## 特别专题 | Topic

### 淘宝开放低耗服务器设计规范

作者 崔康

淘宝的开源项目一直保持着较快的发展，InfoQ 中文站先后介绍过其[开源平台](#)、[数据存储系统 Tair](#) 和[文件系统 TFS](#) 等，最近淘宝又推出了“[开源绿色计算项目](#)”，将 CDN 所用的低耗服务器的设计规范（设备主板、机箱和电源、测试报告）对外发布，在技术社区中引起了很好的反响。

众所周知，淘宝的海量数据处理技术一直受到社区关注和追踪，淘宝的岑文初在刚刚结束的 QCon 杭州 2011 大会上就做了名为《[Web 请求异步处理和海量数据即时分析在淘宝开放平台的实践](#)》的演讲，参会者互动积极。其实，淘宝的技术领域已经不限于软件方面，其在硬件方面做了积极的尝试。最近推出的“[开源绿色计算](#)”项目就是很好的例证。从其项目网站上，可以看出数千台服务器的能源消耗推动淘宝技术团队寻找在硬件方面的优化措施：

“ 随着淘宝用户数和访问量的增加，网络上部署的服务器数量也在不断的增长。为了保障用户体验，淘宝在全网部署了几十个用于加速的 WebCache 节点，服务器的数量达到数千，其每天消耗的电量非常惊人。在整个淘宝网运营成本中，电消耗成本已占了相当比重并逐年增加。因此，节约服务器用电量，已经成为不得不考虑的问题。最直接的方法，就是在满足性能要求的前提下，采用比传统服务器省电的低功耗服务器。定制低功耗服务器，就成为降低能耗的主要方案。

目前市场上的低功耗服务器，与传统服务器的主要区别，在于采用了低功耗 CPU。低功耗 CPU 在带来低功耗的同时，也损失了处理速度。因此，消耗 CPU 资源少的应用，是低功耗服务器首要应用场景。从目前淘宝整个服务器体系看，满足这一要求的是 CDN web cache 服务器。因此，淘宝网的低功耗服务器的定制实践，始于 Cache 服务器的应用需求。

传统的低功耗处理器，其节省功耗的原理，是在原有高性能处理器的基础上，通过控制处理器在闲时的主频和耗电，同时简化乱序处理的逻辑模块，来减少不必要的消耗。但受限于原有的高性能架构，整体功耗降低效果并不明显，特别是在 IO 密集型的业务上。而用于静态内容加速的 CDN web cache 服

务器，主要功能正是对静态的网页和图片进行读写操作，属于 IO 密集型业务。因此，针对 IO 密集型业务，我们需要寻找区别于以往传统的、更低功耗的处理器及其服务器方案。

据淘宝基础核心软件研发负责人章文嵩表示，“淘宝的 CDN 服务绝大部分是图片流量（访问对象平均大小为 18K 字节），有一点静态数据和视频流量，还有一点点动态网页加速的流量。小图片的离散访问在 CDN 中最有挑战的”。在需求的驱动下，淘宝团队通过英特尔和美超微等厂商针对 CDN 缓存定制了一款绿色低功耗的服务器，并以此发起了“开源绿色计算”项目。该项目的目标除了更低功耗方面的要求外，还包括高 IO、方便运维等方面的要求；当然最重要的还是该项目面向整个行业开源，便于大家共同参与这个项目。

目前，该项目已经发布了定制服务器的设计规范，包括三个部分：

“**服务器设备主板设计规范**——主板的设计目标是高功效、低成本和易维护，许多在传统服务器主板上消耗额外功率的特点都将根据应用服务的要求被裁减掉。

**服务器机箱和电源设计规范**——基于主板低功耗的设计目标机箱可以进行高密度方面的考虑；而在电源设计方面则需要考虑高密度情况下的用电；同时两种结合起来后需要考虑整体服务器的散热问题，以保证服务器的稳定性。

**服务器测试报告**——在完成了整个服务器的主板设计和机箱设计后，原型机的测试是非常有必要的；而这种测试将基于某种应用进行，这样就涉及到具体服务器软件和硬件方面的优化配置，以使得服务器在这种配置下得到最高的性能。

下面我们来看看服务器设计规范中提到的一些核心技术。

主板采用嵌入式处理器，Intel® Atom™ D525 (Pineview-D) 双核, 1.8GHz (13W) 处理器，其特点包括：热设计功耗 (TDP) 不超过 13 瓦；在套片上，首先是 32KB 的结构性缓存和 24KB 的回写数据 (L1) 缓存；Intel® 超线程技术，每个核可配置成 2 个超线程；在套片上还有 2 个 512KB8 通道的二级缓存等。系统管理总线是基于 I2C 的操作原则设计的，它提供一条系统控制和电源相关任务管理的总线。一个系统可以用 SMBus 来传递和发送来源于外围设备的消息，而不用单独设计控制总线，从而减少了管脚的数量。能够接受消息也保障了未来的可扩展性。为了节省功耗，主板采用低功耗的网络控制芯片，2 块 Intel 的

82574L 以太网控制芯片。采用 12V 的电压，接口设计成可热插拔的方式；主板通过一个转接板连接电源和硬盘。在 Intel 最新的处理器上已经采用了最先进的热处理技术，因此主板采用的 CPU 可以报告绝对的温度值（华氏/摄氏）。其基本原理是在 CPU 中植入一个主板能够读到的独立温度信息。在出厂的时候“温度门限”或者“温度容忍范围”都由厂家设置到 CPU 中，它将作为主板在不同 CPU 温度条件下采取相应操作（例如：增加 CPU 风扇的速度、触发过热报警等）的基线。由于 CPU 有不同的温度容忍范围，因此目前的 CPU 能够将热容忍范围发送给主板可以给 CPU 带来更好的热管理效果。主板设计应该利用上述 CPU 的热特性，给处理器分配特定热条件下的温度状态（低、中、高）。这样可以让用户很容易理解 CPU 的温度状态，而不是简单的只显示温度的数值（如 25°C）。在 BIOS 中设置风扇全速运行可以让板上的风扇运行在全速（以 100% 宽度脉冲宽度调制占空比）以提供最好的散热。全速设置推荐在特殊系统配置或者调试时使用。选择面向性能的选项，板上的风扇将以 70% 的初始脉冲宽度调制周期来运行，这样也可以带来比较好的散热效果。面向性能的选项建议在高功耗和高密度的系统中使用。如果选择平衡的选项，风扇将以 50% 的初始脉冲宽度调制周期来运行，目的是考虑在散热和功耗间的平衡。平衡模式选项建议在一般配置的系统中使用。还有一种模式就是节能模式，该模式将以 30% 的初始脉冲宽度调制周期来运行，这样可以带来最好的功效和最大的静音效果。

机箱采用热浸镀锌板材料，使得其拥有一个独特的光亮表面。机箱配件设备的安装只需采用飞利浦的螺丝刀即可简单完成。作为一个 2U 八节点的配置，该款机箱是一个非常独特的系统。8 个系统的板卡放入到一个单独的机箱内，每块板卡作为一个独立的服务器节点系统。每两个服务器节点系统放置在一个可热插拔的服务器托盘上，整个机箱共支持 4 个独立的服务器托盘，每个托盘可独立开关机而不影响其他托盘中的系统。另外，承载两个服务器节点系统的托盘可以进行热插拔操作，能够从服务器的后面板抽出；这样一个两节点的托盘通过一块主板上的附加卡与机械的背板连接。机箱支持 24 块 2.5 英寸硬盘的使用。一块单独的背板通过综合各个子系统的信息对机箱系统进行电源控制和风扇速度的控制，同时将三个一组的硬盘与玫瑰节点相连接。因此 RAID 的策略只能应用在同一个节点的三块硬盘上，而不能进行全部 24 块硬盘的 RAID 配置。硬盘通过安装在硬盘托盘上来简化硬盘插入和拔出机箱的操作；同时硬盘托架也可以辅助提高系统的空气流通，使得整个机箱系统更好的散热。机箱可以插入两个电源模块以便可以为系统提供冗余的电源。如果两个电源中其中任何一个电源失效，另外一个电

源会接手所有的用电负荷并且允许系统无中断继续工作。电源模块支持热插拔；如果只有一个电源失效可以在不关闭系统的情况下进行更换。机箱设计了一个独特的散热系统，这个散热系统有四个 8 厘米高性能风扇组成。风扇的速度控制可以在 BIOS 里面进行设置，其速度值由系统的温度决定。四个风扇都直接连接到背板上，但是需要从它们逻辑连接的服务器主板上获取供电。在 2U8 的机箱中，每个节点（服务器板卡）控制在机箱中与其处在同一侧的风扇。这意味着四个节点将共享两个风扇。如果对于这四个节点在 BIOS 中的风扇速度设置不同，那么 BIOS 设置速度最高的将作为应用的设置。

CDN 实现网站加速的基本原理在于将缓存节点部署到靠近终端用户的网络边缘，然后将网站的内容缓存在这些边缘缓存服务器中，然后通过调度将用户的访问调度到离用户最近的网络边缘节点，从而使用户能够从最近的缓存服务器得到网站的内容，从而实现加速的效果。虽然不同的 CDN 缓存节点的架构有所不同，但是都要考虑整个缓存节点的稳定性、可用性、IO 性能和可扩展能力。开源软件 LVS、Haproxy 和 Squid 分别承担四层负载均衡、七层负载均衡和缓存服务的角色。来自于终端用户的请求首先由 LVS 进行四层的处理，基于后端的连接数将用户请求调度到后端的 Haproxy；当 Haproxy 收到 LVS 调度过来的请求后，它将对用户请求的 URL 进行哈希运算，根据运算的哈希值将用户的请求调度给后端的 squid；squid 根据请求从本地存储或者后端的源站获取用户请求的内容然后将响应返回给 Haproxy，Haproxy 将该响应的内容直接返回给用户而不再经过 LVS。考虑到性能 LVS 将单独部署在一台服务器设备上，而考虑到稳定性和高可用性整个节点机器将采用两台 LVS 服务器设备，互为热备；而考虑到扩展性 Haproxy 和 Squid 会部署在一台服务器设备上（对于更低功耗服务器来说 squid 和 Haproxy 在服务器上基本上是一对一部署的）。更大的存储空间意味着缓冲服务器能够缓存更多的内容，这样就可以减少回源站取内容的概率从而提高了用户访问的命中率，进而保证了用户请求的响应时间。IO 的响应时间则表示对于存储介质的访问速度，这个对于用户请求的响应速度影响也非常大。IO 的响应时间可以通过存储介质的 IOPS（单位时间完成的 IO 数）来进行衡量。服务器针对于 CDN 缓存应用的存储机制配置的原则是：内存用于存放最频繁（最热）的内容对象，这部分对象的访问次数一般占到整个被访问次数的 30% 左右，但是存储大概只占到 1% 左右；接下来次热的内容被存放在 SSD 上，这部分内容的访问次数大概占到整个被访问对象的次数的 60% 左右，而存储大概只占到 20% 左右；冷点的内容存放在机械硬盘上，这部分内容的访问次数大概占到被访

问次数的 10% 左右，但是存储却占到 80%。每片服务器有 3 个 Sata 接口的硬盘盘位 配置一块 SSD 和两块 Sata 机械硬盘来满足 CDN 缓存分层存储的需要。由于 SSD 本身的响应时间已经完全超越了目前这款更低功耗服务器的 CPU 所能够处理的速度 ,因此对于 SSD 硬盘的选项主要关注在存储大小的选择。由于 SSD 的大小直接和其成本相关 因此需要根据缓存内容的大小来选择合适大小的 SSD 硬盘。根据刚才的分析 ,SSD 大概需要存储占总缓存内容大小 20% 左右的对象内容 ,大概估算一下整个服务器需要缓存内容的大小约为 300GB 左右 ,因此 SSD 的大小选择为 60GB 左右的硬盘比较合适 ;目前市场上和上升大小规格相近的硬盘型号有 64GB 和 80GB 两种 ,考虑到 SSD 本身需要一定的冗余来保证性能 ,因此选择了 80GB 的型号。对于机械硬盘的选择则需要考虑的因素包括 :存储大小、功耗、稳定性、成本、硬盘转速、性能 ( 主要用随机访问模式下的 IOPS 来衡量 ) ,最终选择了型号为 ST95005620AS 的硬盘。由于内存的选型依赖于 CPU 的类型 ,目前 Atom D525 的处理器只能支持不大于 4GB 的 DDR3 的内存 ,内存的主频不超过 800MHz ,而且没有 ECC 的支持 ;另外考虑到低功耗的要求所以选择的是两条 2GB DDR3 的笔记本内存。

淘宝对低耗服务器做了详细的测试 ,性能测试方面包括实验室环境和线上环境 ,还有功耗测试和稳定性测试。测试指标包括 : 响应时间 ( 从 squid 接收到请求到将响应发送出去的时间 ) 、每秒处理的请求数 ( 每秒 Squid 能够成功处理的请求数 ,该指标可以通过 squidclient 工具得到 ) 、流量 ( 单位时间内 squid 针对用户请求所响应内容的比特流量 ) 、功效 ( 单位功耗下能够服务的 QPS 数 ,或者每瓦服务的流量 ) 。

实验室的性能测试表明 ,大片更低功耗服务器的 CDN 缓存服务最大能力大概为 1700QPS( 或 229.5Mbps ) ,而最高功效为 70 QPS/ 瓦 ( 或者 9.46 Mbps/ 瓦 ) 。线上环境测试结果 :整体的缓冲命中率在 95% 以上 ;95% 的响应时间都在 20ms 以下 ,最高不超过 100ms ; 测试到的最高性能为 1100QPS ,而此时的 CPU 利用率只有 70% 左右、磁盘的利用率也在 40% 以下。就稳定性而言 ,线上一个月的运行并没有出现应用服务的明显中断 ,而且整体性能曲线平稳 ,基本上可以判断稳定性能够满足 CDN 服务的要求。

对于服务器的下一步优化 ,文档给出了提示。功耗优化 : 在 BIOS 中设置可关闭 USB 接口 ; 设置关闭 CPU 中的图像处理逻辑 GPU 。运营维护优化 : 提供 1 对多的 OOB 或 IPMI 管理以太网接口 ; 进一步提高服务器的密度 , 提高单台服务器的服务能力 ,降低单位节点的服务器设备数 ; 同时也可提高电源的转换效率 ;

---

集成网络交换模块到服务器机框中 ,降低服务器对外的网络接口数量。性能优化 : 扩展 Sata 硬盘的盘位 , 进一步提高机械硬盘的服务能力 , 进而提高整体服务器的服务能力 ; 更换效能更高的 CPU : SandyBridge ( 15 瓦 ) 。

据章文嵩在微博中透漏 , 目前该定制服务器已经实现量产 , 建设了十几个 CDN 节点 , 约 800 台 ATOM 服务器。跟基于 Intel L3406 或 E5602 处理器的服务器节点比起来 , QPS/ 瓦是最高的 , 成本最低 , 响应时间平均最短 , 得益与服务器数目多硬盘多。可运维性还有待加强。

如果读者想详细的了解服务器的设计规范和参数 , 可以访问项目的网站来 [下载](#) 相关的文档。

**原文链接 :**

<http://www.infoq.com/cn/news/2011/11/taobao-greencompute>

**相关内容 :**

- [为速度而生——看百姓网如何优化网速](#)
- [Node.js 与 Rails 如何选择 ?](#)
- [VMware 发布 Cloud Foundry 的免费版本](#)

## 特别专题 | Topic

# 淘宝双十一架构优化及应急预案

作者 贾国清

每年的双十一，在整体架构上最依仗的是过去几年持续建设的稳定的服务化技术体系和去 IOE 带来整体的扩展性和成本的控制能力。

今年在架构上做的比较大的优化有三点：

“一是导购系统的静态化改造。今年淘宝和天猫都分别根据自身的业务特性进行了 Detail 页面的静态化改造，但核心的思路都是通过 CDN+nginx+JBoss+缓存的架构，将页面信息进行静态化缓存。这次的优化突破了现有系统在 Web 服务器和 Java 处理信息的 CPU 瓶颈，将访问最密集的 Detail 集群的性能提升了 10 多倍，极大程度的缓解了突发流量场景下的性能和成本压力。在今年的阿里技术嘉年华上，淘宝 Detail 团队的君山分享了淘宝静态化改造的一些经验。后续我们也可以安排天猫 Detail 团队的同学分享下在不同的业务背景下技术选型的思考。

第二是天猫优惠系统的架构改造，整体上的思路是规则中心化计算本地化。简单来说，将优惠规则的计算推送到所有需要展现价格的上层业务系统进行，而不再集中在后端进行处理。通过这样的方式减少了分布式的调用和计算单点的瓶颈问题。

第三是支付核心路径的异步化。支付环节的系统瓶颈一直在于后端的银行系统吞吐率无法跟上支付宝的处理能力，大量的线程阻塞等待银行系统的返回。今年支付宝的同学们通过一个高可用的队列，异步处理对关键支付路径的调用，这样做的好处一方面让支付系统的处理资源得到了释放，最大化系统处理能力，另一方面也支撑了更加灵活的限流控制。

有了以上的基础，接下来就是对突发情况的应急预案准备了，我们的做法通常有两种：

“一是业务降级，即舍弃一些非核心的业务，确保核心业务的系统资源和稳定运行。举个例子，天猫的美妆类目交易流程有个特性，购买部分商家的商品会附赠小样。这部分逻辑是会给核心交易系统增加额外的系统依赖的，如果

大促出现紧急情况，我们就会考虑砍掉这部分特殊的逻辑，确保核心交易的稳定。

## 二是系统限流，关键服务进行限流控制，保护核心集群的系统稳定。

这两点看似简单，却对系统和代码段之间的耦合性要求极高，我们前期也做了大量的工作尽可能将系统之间的强依赖变为弱依赖，避免因为某一系统的性能下降导致反向的雪崩效应。

另一方面，对于限流的阈值评估同样是一个难点，我们首先要从根据历年来的业务变化数据和运营计划估算一个可能达到的交易额以及 UV 数量。然后计算我们可能达到的交易笔数和访问峰值。

其次，分析出每笔交易需要跟多少个系统进行交互，计算出每笔交易会对后端的商品中心、用户中心、库存中心、优惠、物流等要调用多少次，得出一个支撑一次访问或是一笔交易的合理系统消耗公式。再次，通过线上压测的手段对线上各个集群的极限能力有个准确的认知。

最后再根据业务峰值、系统消耗公式、压测数据，去准备扩容的机器数量和限流的具体阀值。

今年是双十一准备的最为充分的一年，前期的功能测试、容量评估、应急预案都比以往做的更加仔细，一共 400 多个应急预案，从 9 月开始就不断的进行系统演练，确保每个应急预案的准确和便捷执行。所以总的来说，当天各个系统表现还是比较淡定的。

突发事件虽然也有发生，还是做到了冷静处理，所有情况都在掌控之中。当然，无论是限流还是降级，都是以牺牲用户体验作为代价的，我们内部也在反思和讨论，以后可以把流控做的更加有趣些，让大家买的开心，等的不无聊。

### 原文链接：

<http://www.infoq.com/cn/news/2013/01/taobao-double-11-sys-optimization>

### 相关内容：

- [曾宪杰谈 Java 在淘宝的应用](#)
- [岑文初谈淘宝开放平台技术发展历程](#)

# 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...

QClub

邀请  
业内知名专家

自由开放的  
讨论氛围

定期举办的线下活动

结识  
圈内技术好友

InfoQ



中文 | 英文 | 日文 | 葡文 | .....

## 推荐文章 | Articles

# 聊聊并发（五）——原子操作的实现原理

作者 方腾飞

## 1. 引言

原子 ( atom ) 本意是 “不能被进一步分割的最小粒子” , 而原子操作 ( atomic operation ) 意为 “不可被中断的一个或一系列操作” 。在多处理器上实现原子操作就变得有点复杂。本文让我们一起来聊一聊在 Intel 处理器和 Java 里是如何实现原子操作的。

## 2. 术语定义

术语	英文	解释
缓存行	Cache line	缓存的最小操作单位
比较并交换	Compare and Swap	CAS操作需要输入两个数值，一个旧值（期望操作前的值）和一个新值，在操作期间先比较下旧值有没有发生变化，如果没有发生变化，才交换成新值，发生了变化则不交换。
CPU流水线	CPU pipeline	CPU流水线的工作方式就象工业生产上的装配流水线，在CPU中由5~6个不同功能的电路单元组成一条指令处理流水线，然后将一条X86指令分成5~6步后再由这些电路单元分别执行，这样就能实现在一个CPU时钟周期完成一条指令，因此提高CPU的运算速度。
内存顺序冲突	Memory order violation	内存顺序冲突一般是由假共享引起，假共享是指多个CPU同时修改同一个缓存行的不同部分而引起其中一个CPU的操作无效，当出现这个内存顺序冲突时，CPU必须清空流水线。

## 3. 处理器如何实现原子操作

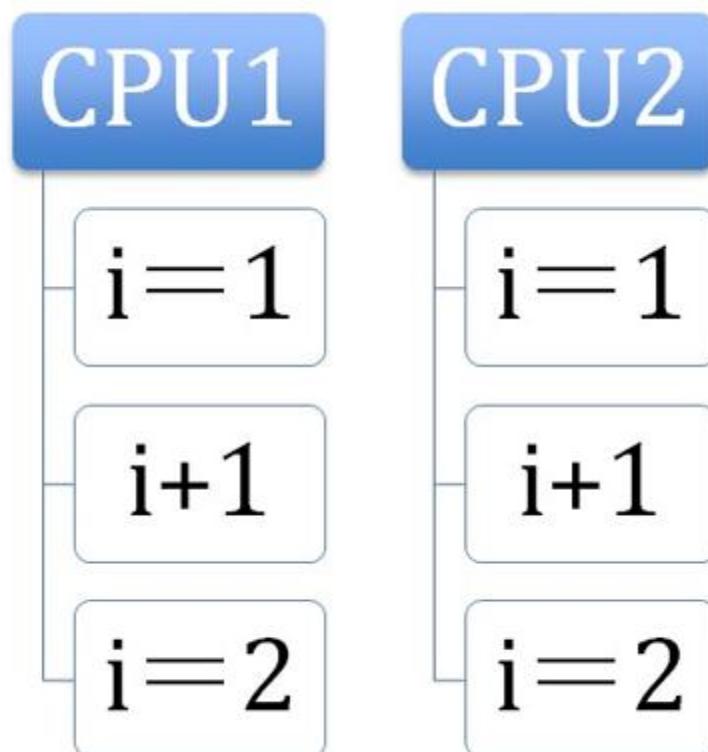
32 位 IA-32 处理器使用基于对缓存加锁或总线加锁的方式来实现多处理器之间的原子操作。

### 3.1 处理器自动保证基本内存操作的原子性

首先处理器会自动保证基本的内存操作的原子性。处理器保证从系统内存当中读取或者写入一个字节是原子的，意思是当一个处理器读取一个字节时，其他处理器不能访问这个字节的内存地址。奔腾 6 和最新的处理器能自动保证单处理器对同一个缓存行里进行 16/32/64 位的操作是原子的，但是复杂的内存操作处理器不能自动保证其原子性，比如跨总线宽度，跨多个缓存行，跨页表的访问。但是处理器提供总线锁定和缓存锁定两个机制来保证复杂内存操作的原子性。

### 3.2 使用总线锁保证原子性

第一个机制是通过总线锁保证原子性。如果多个处理器同时对共享变量进行读改写（ $i++$  就是经典的读改写操作）操作，那么共享变量就会被多个处理器同时进行操作，这样读改写操作就不是原子的，操作完之后共享变量的值会和期望的不一致，举个例子：如果  $i=1$ ，我们进行两次  $i++$  操作，我们期望的结果是 3，但是有可能结果是 2。如下图



原因是有可能多个处理器同时从各自的缓存中读取变量  $i$ ，分别进行加一操作，然后分别写入系统内存当中。那么想要保证读改写共享变量的操作是原子的，就必须保证 CPU1 读改写共享变量的时候，CPU2 不能操作缓存了该共享变量内存地址的缓存。

处理器使用总线锁就是来解决这个问题的。所谓总线锁就是使用处理器提供的一个 LOCK # 信号，当一个处理器在总线上输出此信号时，其他处理器的请求将被阻塞住，那么该处理器可以独占使用共享内存。

### 3.3 使用缓存锁保证原子性

第二个机制是通过缓存锁定保证原子性。在同一时刻我们只需保证对某个内存地址的操作是原子性即可，但总线锁定把 CPU 和内存之间通信锁住了，这使得锁定期间，其他处理器不能操作其他内存地址的数据，所以总线锁定的开销比较大，最近的处理器在某些场合下使用缓存锁定代替总线锁定来进行优化。

频繁使用的内存会缓存在处理器的 L1，L2 和 L3 高速缓存里，那么原子操作就可以直接在处理器内部缓存中进行，并不需要声明总线锁，在奔腾 6 和最近的处理器中可以使用“缓存锁定”的方式来实现复杂的原子性。所谓“缓存锁定”就是如果缓存在处理器缓存行中内存区域在 LOCK 操作期间被锁定，当它执行锁操作回写内存时，处理器不在总线上声言 LOCK # 信号，而是修改内部的内存地址，并允许它的缓存一致性机制来保证操作的原子性，因为缓存一致性机制会阻止同时修改被两个以上处理器缓存的内存区域数据，当其他处理器回写已被锁定的缓存行的数据时会起缓存行无效，在例 1 中，当 CPU1 修改缓存行中的  $i$  时使用缓存锁定，那么 CPU2 就不能同时缓存了  $i$  的缓存行。

但是有两种情况下处理器不会使用缓存锁定。第一种情况是：当操作的数据不能被缓存在处理器内部，或操作的数据跨多个缓存行（cache line），则处理器会调用总线锁定。第二种情况是：有些处理器不支持缓存锁定。对于 Inter486 和奔腾处理器，就算锁定的内存区域在处理器的缓存行中也会调用总线锁定。

以上两个机制我们可以通过 Inter 处理器提供了很多 LOCK 前缀的指令来实现。比如位测试和修改指令 BTS，BTR，BTC，交换指令 XADD，CMPXCHG 和其他一些操作数和逻辑指令，比如 ADD（加），OR（或）等，被这些指令操作的内存区域就会加锁，导致其他处理器不能同时访问它。

## 4. JAVA 如何实现原子操作

在 java 中可以通过锁和循环 CAS 的方式来实现原子操作。

### 4.1 使用循环 CAS 实现原子操作

JVM 中的 CAS 操作正是利用了上一节中提到的处理器提供的 CMPXCHG 指令实现的。自旋 CAS 实现的基本思路就是循环进行 CAS 操作直到成功为止，以下代码实现了一个基于 CAS 线程安全的计数器方法 safeCount 和一个非线程安全的计数器 count。

```

public class Counter {
    private AtomicInteger atomicI = new AtomicInteger(0);
    private int i = 0;
    public static void main(String[] args) {
        final Counter cas = new Counter();
        List<Thread> ts = new ArrayList<Thread>(600);
        long start = System.currentTimeMillis();
        for (int j = 0; j < 100; j++) {
            Thread t = new Thread(new Runnable() {
                @Override
                public void run() {
                    for (int i = 0; i < 10000; i++) {
                        cas.count();
                        cas.safeCount();
                    }
                }
            });
            ts.add(t);
        }

        for (Thread t : ts) {
            t.start();
        }
        // 等待所有线程执行完成
        for (Thread t : ts) {
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(cas.i);
        System.out.println(cas.atomicI.get());
        System.out.println(System.currentTimeMillis() - start);
    }

    /**
     * 使用CAS实现线程安全计数器
     */
    private void safeCount() {
        for (;;) {
            int i = atomicI.get();
            boolean suc = atomicI.compareAndSet(i, ++i);
            if (suc) {
                break;
            }
        }
    }
    /**
     * 非线程安全计数器
     */
    private void count() {
        i++;
    }
}

```

在 java 并发包中有一些并发框架也使用了自旋 CAS 的方式来实现原子操作 , 比如 LinkedTransferQueue 类的 Xfer 方法。 CAS 虽然很高效的解决原子操作 ,

但是 CAS 仍然存在三大问题。ABA 问题，循环时间长开销大和只能保证一个共享变量的原子操作。

1. ABA 问题。因为 CAS 需要在操作值的时候检查下值有没有发生变化，如果没有发生变化则更新，但是如果一个值原来是 A，变成了 B，又变成了 A，那么使用 CAS 进行检查时会发现它的值没有发生变化，但是实际上却变化了。ABA 问题的解决思路就是使用版本号。在变量前面追加上版本号，每次变量更新的时候把版本号加一，那么 A - B - A 就会变成 1A-2B - 3A。

从 Java1.5 开始 JDK 的 atomic 包里提供了一个类

AtomicStampedReference 来解决 ABA 问题。这个类的 compareAndSet 方法作用是首先检查当前引用是否等于预期引用，并且当前标志是否等于预期标志，如果全部相等，则以原子方式将该引用和该标志的值设置为给定的更新值。

```
public boolean compareAndSet
    (V      expectedReference, //预期引用
     V      newReference, //更新后的引用
     int    expectedStamp, //预期标志
     int    newStamp) //更新后的标志
```

2. 循环时间长开销大。自旋 CAS 如果长时间不成功，会给 CPU 带来非常大的执行开销。如果 JVM 能支持处理器提供的 pause 指令那么效率会有一定的提升，pause 指令有两个作用，第一它可以延迟流水线执行指令（de-pipeline），使 CPU 不会消耗过多的执行资源，延迟的时间取决于具体实现的版本，在一些处理器上延迟时间是零。第二它可以避免在退出循环的时候因内存顺序冲突（memory order violation）而引起 CPU 流水线被清空（CPU pipeline flush），从而提高 CPU 的执行效率。
3. 只能保证一个共享变量的原子操作。当对一个共享变量执行操作时，我们可以使用循环 CAS 的方式来保证原子操作，但是对多个共享变量操作时，循环 CAS 就无法保证操作的原子性，这个时候就可以用锁，或者有一个取巧的办法，就是把多个共享变量合并成一个共享变量来操作。比如有两个共享变量 i = 2, j = a，合并一下 ij = 2a，然后用 CAS 来操作 ij。从 Java1.5 开始 JDK

提供了 `AtomicReference` 类来保证引用对象之间的原子性，你可以把多个变量放在一个对象里来进行 CAS 操作。

## 4.2 使用锁机制实现原子操作

锁机制保证了只有获得锁的线程能够操作锁定的内存区域。JVM 内部实现了很多种锁机制，有偏向锁，轻量级锁和互斥锁，有意思的是除了偏向锁，JVM 实现锁的方式都用到的循环 CAS，当一个线程想进入同步块的时候使用循环 CAS 的方式来获取锁，当它退出同步块的时候使用循环 CAS 释放锁。详细说明可以参见文章 Java SE1.6 中的 `Synchronized`。

## 5. 参考资料

[Java SE1.6 中的 `Synchronized`](#)

[Intel 64 和 IA-32 架构软件开发人员手册](#)

[深入分析 `Volatile` 的实现原理](#)

## 作者介绍

**方腾飞**，花名清英，淘宝资深开发工程师，关注并发编程，目前在广告技术部从事无线广告联盟的开发和设计工作。个人博客：<http://ifeve.com> 微博：<http://weibo.com/kirals> 欢迎通过我的微博进行技术交流。

**原文链接：**<http://www.infoq.com/cn/articles/atomic-operation>

### 相关内容：

- [聊聊并发（四）——深入分析 `ConcurrentHashMap`](#)
- [聊聊并发（三）——JAVA 线程池的分析和使用](#)
- [聊聊并发（一）——深入分析 `Volatile` 的实现原理](#)
- [Rob Pike 谈 Google Go：并发，Type System，内存管理和 GC](#)
- [Intel 发布 JavaScript 扩展以支持并行运算](#)

## 推荐文章 | Articles

# 视图模型（View-Model）到底是什么？

作者 [Jonathan Allen](#) 译者 [姚琪琳](#)

在“视图模型（View-Model）”这个术语出现之后，很多开发者都有不少疑问。视图模型需要处理视图、模型和外部服务间的交汇的问题，这一点是清晰的，但准确的做法却往往被一笔带过。它应该包含哪些内容，不应该包含哪些内容，没有清晰的列表，它们往往最终会成为所有东西的大杂烩。本文无意给出明确的答案，而是要探索视图模型所承担的众多角色中的几个。

在你阅读本文中的不同角色和模式的时候，请记住以下三点：

1. 这些示例都来自真实项目。
2. 大多数视图模型都会承担多个角色。
3. 严格遵守某个模式的重要性低于可运行的程序。

## 模型端的角色

为视图提供数据是视图模型至关重要的角色。然而，即便仍然使用 XAML 数据绑定技术，提供数据的方式也是多种多样的。

### 用视图模型替代数据上下文

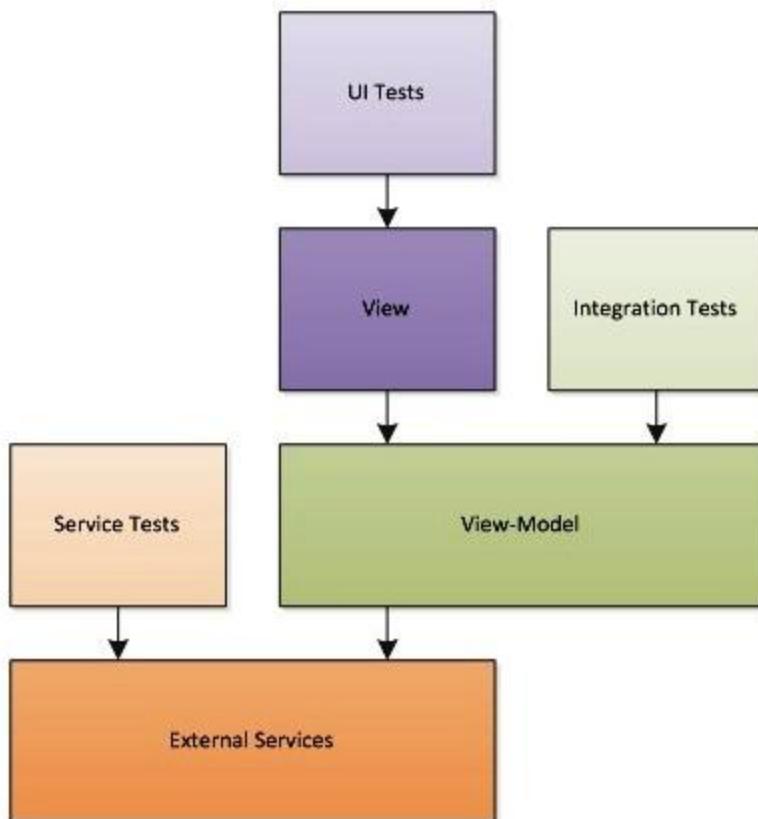
用视图模型替代数据上下文是最简单的模型端（model-side）模式，可能也是最常见的。真正的数据通过视图模型的一个或多个简单的属性暴露出来。在某种程度上，这并不是模式。它只是将视图模型和视图的真正数据上下文属性关联起来，并且注入其他功能，如包装导航或服务调用的 ICommand。本文后面还会讨论这个话题。

### 将视图模型作为活动记录

遗憾的是，“将模型作为活动记录（Active Record）”是一个常见的错误模式。在这种模式下，应用程序中没有真正的模型。相反，所有的字段都由视图模型本身直接提供。例如，CustomerViewModel 可能包含 FirstName、LastName、CustomerId 字段。由于这是一个视图模型，所以很可能还挂接了外部服务。可

以通过 LoadCustomerCommand 和 SaveCustomerCommand 等 ICommand 将他们暴露出来，这就将视图模型成功地转变成[活动记录](#)。

需要注意的是，活动记录模式本身在某些场景下是相当有效的。问题是，活动记录的用途一定程度上被夸大了，若再让它们担任视图模型的其他角色，就几乎成了“[万能对象 \( god object \)](#)”。



如图所示，单元测试没有安身之处。你可以通过使用附带模拟（mock）服务的集成测试来创建虚拟的单元测试，但这种方法往往非常耗时，并且容易出错。

**如果没有模型，就不是 MVVM。**

### 将视图模型作为适配器或装饰器

视图模型可以作为适配器（adapter）或装饰器（decorator），以临时包装一个模型，提供额外信息或新格式。然而，这是非常危险的实践，只要有可能就应该避免。

我们使用经典的 FullName 来作为示例，这样做会带来两种风险。

## 基于推送的包装器

在基于推送的包装器 ( wrapper ) 中 , 我们假设只有视图模型能向视图推送数据更新。

```
public class PersonViewModel : INotifyPropertyChanged
{
    private readonly Person m_Model;
    public PersonViewModel(Person model)
    {
        m_Model = model;
    }

    public string FirstName
    {
        get { return m_Model.FirstName; }
        set
        {
            m_Model.FirstName = value;
            OnPropertyChanged(new PropertyChangedEventArgs("FirstName"));
            OnPropertyChanged(new PropertyChangedEventArgs("FullName"));
        }
    }
}
```

这意味 , 如果不通过 PersonViewModel 包装器而直接更改模型 , 此操作就会失败。这些更改不会传播到视图 , 导致同步问题。

## 基于事件的包装器

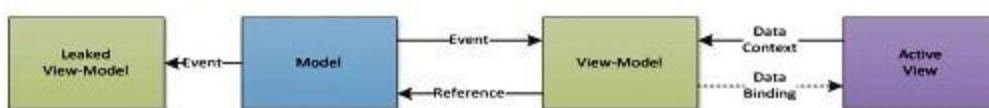
基于推送包装器的一个替代方案是 , 依赖由模型引发并由视图模型转发的事件。如下面的代码所示 :

```
public class PersonViewModel : INotifyPropertyChanged
{
    private readonly Person m_Model;
    public PersonViewModel(Person model)
    {
        m_Model = model;
        m_Model.PropertyChanged += Model_PropertyChanged;
    }

    public string FirstName
    {
        get { return m_Model.FirstName; }
        set { m_Model.FirstName = value; }
    }

    void Model_PropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        OnPropertyChanged(e);
        switch (e.PropertyName)
        {
            case "FirstName":
            case "LastName:":
                OnPropertyChanged(new PropertyChangedEventArgs("FullName"));
                break;
        }
    }
}
```

在此方案中 , 从模型向视图模型附加属性变化通知是存在风险的。当同一个模型被多个视图模型访问时 , 将可能出现内存泄露。



基于 XAML 的控件通过监听模型事件可摆脱该问题，因为数据绑定基础架构（infrastructure）可以防止内存泄露。而在 WinForm 中，可以使用组件的基础架构释放（dispose）事件处理器。但是，视图模型的生命周期并不依赖于控件，也不能依赖于控件，因为控件可以重新加载，因此，它没有适当的地方供我们释放这些事件处理器。

这未必就是问题。如果你 100% 确定没有其他东西持有对模型的引用，就可以使用视图模型包装器。但更安全的做法是增强模型本身（如将 FullName 属性放到模型内）或使用值转换器。这两种方法还能让你在不涉及视图模型的情况下编写单元测试。

**如果视图模型监听模型中的事件，就要检查是否会产生内存泄露。**

## 将视图作为聚合根

根据维基百科，聚合在[领域驱动设计](#)的词条中被定义为：

“与某个根实体对象绑定在一起的对象集合，这个根实体对象也叫聚合根。聚合根通过禁止外部对象持有对其成员的引用，来确保聚合内所发生的变化的一致性。”

聚合根与适配器或装饰器的主要区别是，其模型永远不会暴露。你根本无法访问任何模型、模型的属性或事件。只有聚合根持有对模型的引用，因此可以完全防止上一节看到的内存泄露。

**如果视图模型完全隐藏了复杂对象图的细节，那么它可能是一个聚合根。**

## 将视图模型作为 Web MVC 模型

这是我在 ASP.NET MVC 社区中见到的一个相对较新的现象。很多开发者把由控制器创建或加载，并传递给视图的类叫做“视图模型”。这是与“领域模型”和“输入模型”相对应的。Dino Esposito 的文章[“ASP.NET MVC 应用中的三种模型”](#)很好地解释了这一点。

这种类型的视图模型很容易识别，因为它除了包含数据和完全依赖于数据的业务规则外，不拥有其他任何角色和职责。因此，它具有其他纯模型所具备的所有优点，如易于单元测试。

这时就没有必要责怪这种命名方法了。就像传统 MVC 和 Web MVC 之间的区别一样，这是我们不得不接受的习惯之一。

**注意不要混淆“视图的模型”和“视图模型”，特别是 MVVM 应用程序可能包含这两种模型。**

## 视图端的角色

视图和视图模型之间存在不同程度的耦合。我们先来看看最紧密的耦合，然后讨论更理想化的视图模型。

### 将视图模型作为代码隐藏

对于 XAML 初学者来说，往往会犯一个反模式错误。人们常说在 “xaml.cs” 或 “xaml.vb” 当中不应该放入过多的代码。善良的初学者经常将其误解为不能放入任何代码。因此，他们将所有东西都扔进 “视图模型”，使其变成一种放错位置的代码隐藏文件。

这时，如果把所有东西都扔到代码隐藏文件中，所面临的问题是一模一样的。此外，把所有东西都扔进视图模型还会使得静态分析更加困难，并且带来内存泄露的可能。

**如果将所有的代码从 Xxx.xaml.cs 移到 XxxViewModel.cs，视图模型就变成了“代码隐藏”文件。**

### 将视图模型作为传统的 MVC 控制器

传统 MVC 和 Web MVC 一个关键的不同在于控制器和视图的关系。在 Web MVC 中，控制器可以创建并返回它想要的任何视图。除了为这些视图提供数据模型之外，与它们没有任何交互。

而传统的 MVC 视图和控制器则总是紧密耦合在一起。每个控制器都是特别构建的，为某个特殊视图的用户生成的事件提供服务。这种模式也出现在 MVVM 中，视图模型扮演控制器的角色，而 ICommand 则取代了事件。

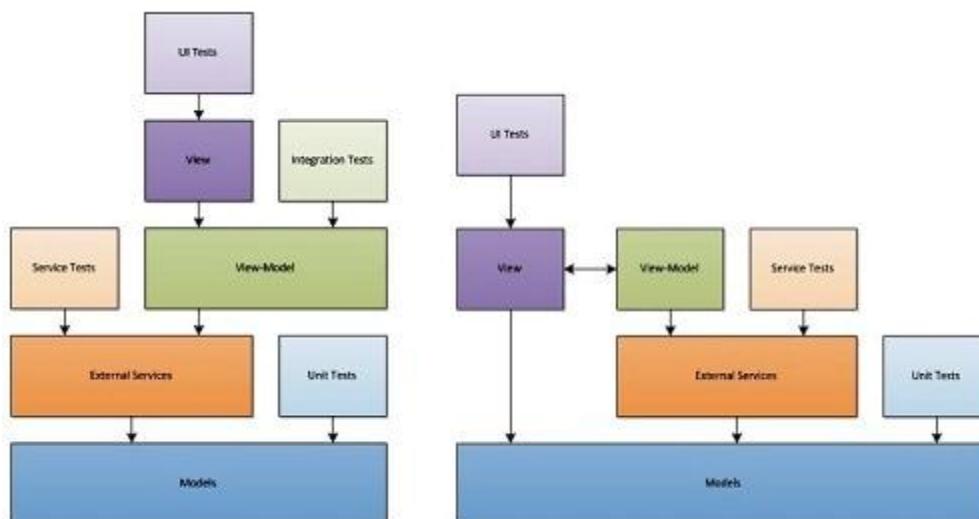
将视图模型作为控制器和将其作为代码隐藏文件之间的差别非常细微 ,因此有一些通用的准则 :

将视图模型作为控制器的标志

- 用 ICommand 处理外部资源
- 通过暴露附加属性的方式来触发可视状态 ( Visual State ) 的改变
- 公开了控件能够响应的属性和事件
- 只侦听视图上的一般事件 , 如 Loaded 和 Unloaded

将视图作为代码隐藏文件的标志

- 使用 EventToCommand 处理视图模型中的所有事件
- 通过直接调用可视状态管理器 ( Visual State Manager ) 的方式触发可视状态的改变
- 直接与控件交互
- 依赖于视图的具体布局
- 为了让视图模型正确工作 ,公开了控件必须响应的事件 下面的层关系图展示了控制器式和代码隐藏式视图模型的不同。



控制器和代码隐藏之间的这种区别对应用程序本身来说影响不大 ,但如果要对视图模型进行不依赖于视图的测试 ,这种差别就显现出来了。尽管我不赞成对视图模型这样的集成组件进行单元测试 ,但执行集成测试来确保其与外部资源 ( 数据

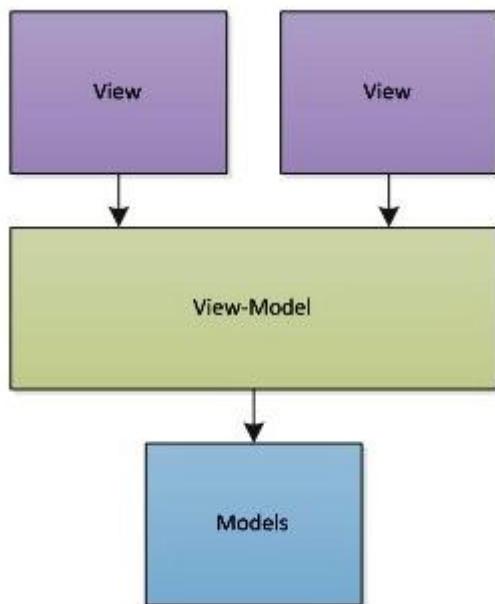
库、Web 服务 ) 正常工作将是十分有帮助的。视图与视图模型之间的这种双向耦合将使集成测试变得更加困难。

**如果视图模型与视图之间是一对一映射的，那么视图模型就成了一个控制器。**

## 将视图模型作为共享的控制器

包括我在内的很多人都宣扬，如果视图模型不能被多个视图共享，就不是真正的视图模型。尽管我现在对这个理论不那么教条了，但我仍然将其视为一个有用的设计模式。

如果你要在多个使用相同数据的视图之间进行同步，就可以使用共享控制器方法。例如，某个窗体中有一个数据网格（data grid），另一个窗体中有一个展示这些数据的图表。这时你可以直接共享模型，但共享视图模型却更不至于犯错。这样，如果你切换到完全不同的数据，两个视图将同时被告知。



注意，共享的视图模型写起来要比控制器式的视图模型难得多。而且它们还非常不灵活，你必须在代码隐藏文件中放入比平时更多的代码。其好处是，共享的视图模型更易于测试，因为它们必然排除了很多复杂性。

一个可行的替代方案是，在传统的 MVC 模式中对模型（而非控制器）进行共享。XAML 数据绑定能很好地支持这两种设计，因此就看哪种方案对整体设计的破坏性更小了。

**如果将视图模型设计为共享的，它们将更易于测试。**

## 将视图模型作为导航器

4个主要的基于XAML的UI框架都包含导航风格的应用程序，但Windows Phone或Windows 8 Metro应用则对它更加情有独钟，因为在这些应用中，导航框架都处于核心地位。

幸运的是，如果用其他东西（如顶层视图模型）来包装导航框架，也非常适于单元测试和集成测试。当触发页面转移时，视图模型不会将URI发送给导航框架，而会发送给一个可以检查的模拟类。可以参考Granite.Xaml的[NavigatorViewModel](#)和[SimpleNavigator](#)类。

**抽象的导航框架对于构建可测试的视图模型来说是至关重要的。**

## 将视图模型作为MVP的展示器

模型-视图-展示器模式与传统MVC模式最主要的区别在于模型和视图间的交互方式不同。在MVC模式中，模型直接触发视图监听的事件。而在MVP模式中，展示器监听事件，并更新视图本身。

由于XAML十分强大，在所谈论的技术中，我们几乎察觉不到这种模式的存在。但实际上它在多线程应用程序中扮演了重要的角色。

当模型在没有任何用户交互的情况下被后台线程更新时，常常会用到这种模式。在这种情况下，视图模型负责将通知封送（marshal）给UI线程。这可能会涉及将数据复制给数据绑定的模型，或直接更新控件本身。

值得注意的是，这并不是处理多线程的唯一方式。如果计算复杂度不高，最好简单地将所有异步消息封送到UI线程，并在那里进行处理。一些库（如响应式扩展）可以简化这些操作。

**考虑使用展示器模式来确保不在UI线程中处理异步消息。**

## 服务/资源角色

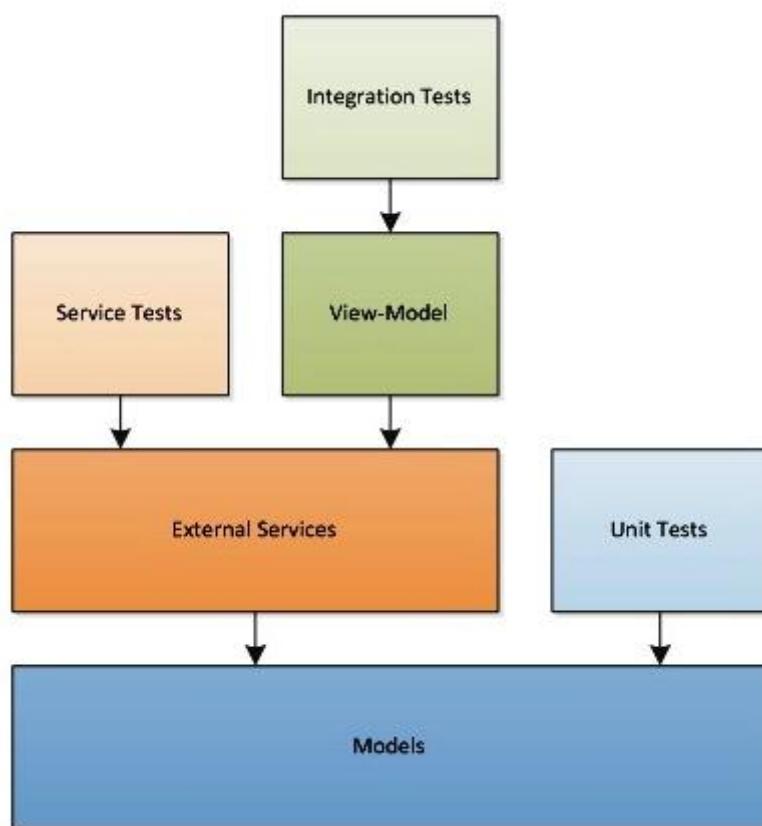
到目前为止，我们都将“外部服务”作为黑盒，它表示应用程序可能需要访问的文件系统、数据库、Web服务以及其他外部资源。现在，到了该考虑如何真正访问这些服务的时候了。有两种基本的方式可以实现这一点。

## 将视图模型作为数据访问层

对于小应用和定义良好的外部服务，通过视图模型直接调用是非常简单的。这种模型对于简单应用是非常理想的，并且 WCF 注重接口的设计使得实现依赖注入变得更加容易。

这样做的缺点是伸缩性不够好。随着应用程序越来越复杂，你会发现视图模型中的逻辑越来越多。在.NET 4.5 引入 `async/await` 关键字之前尤其如此。

如果你发现你的视图模型被回调逻辑搞得不堪重负，就可以考虑添加一个单独的客户端服务层。



注意，在使用这种模式时，很多开发者会选择放弃将外部服务作为独特的组件进行测试。其理论是，如果只有视图模型可以访问服务层，那么针对视图模型的测试就可以同时承担测试服务层的职责。

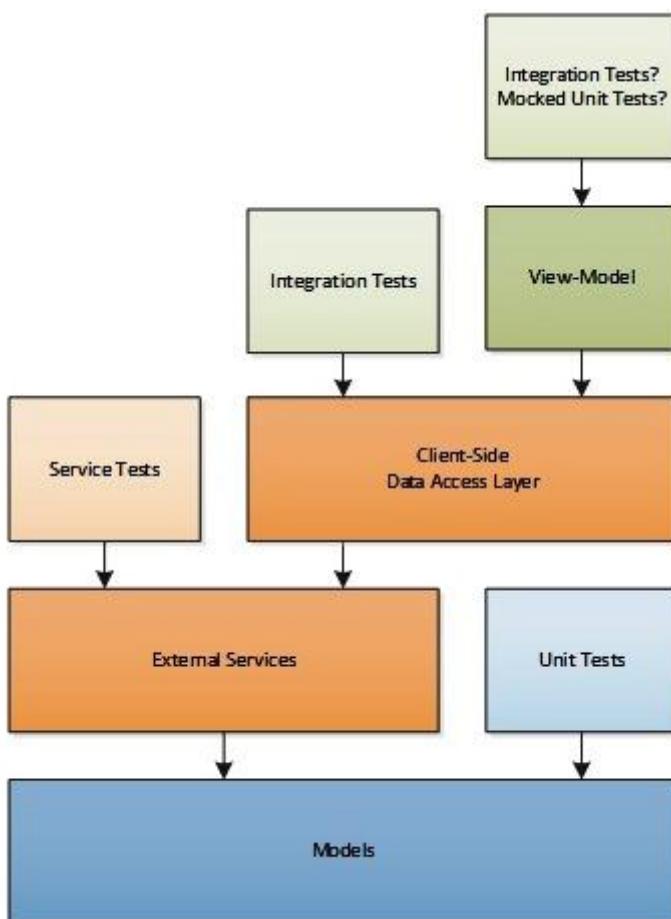
**WCF 接口是依赖注入和模拟的完美选择，特别是当应用程序不包含单独的数据访问层时。**

视图模型加数据访问层

有两种情况你可能需要单独的数据访问层：

- 视图模型变得过于庞大，很难维护。
- 在多个视图模型之间重复相同的外部服务逻辑。

这种模式很难写正确。如果在定义应用程序中 DAL 和 VM 的特定角色时不加注意，最后可能所有逻辑都会位于 DAL 或 VM 之中。这不仅违背了该模式的本意，而且当维护者看到几乎为空的组件作为没必要的样板代码时，也会感到十分迷惑。



在编写集成测试时，你有两种选择。可以像之前那样针对视图模型，同时测试这两个组件。或者，也可以针对数据访问层。

后者的优势在于为我们提供了引入基于接口的依赖注入的干净的做法。反过来，它使我们可以使用模拟的单元测试对视图模型进行测试。

## 结论：掌握视图模型

真正掌握视图模型并避免其成为大杂烩的唯一途径是，先判断该术语对你来说具体意味着什么。列出所有关注点，看看它属于哪个组件。

下面这个示例来自我的一个应用程序；你的列表也许非常相似，也许会截然不同。

关注点	组件
在页面间导航	基本的视图模型（Silverlight 导航框架）
格式化	视图（值转换器）
在视图中显示对话框	视图代码隐藏
加载依赖项	基本的视图模型
创建控制器	基本的视图模型
触发 WCF 调用来加载或保存模型	视图特定的控制器
验证用户输入	模型（INotifyDataErrorInfo）
错误报告	基本的视图模型

当然，该表格只包含客户端代码。对于真正的项目，还需要建立两个这样的表格，一个为服务端代码，另一个表示哪个数据表对应哪个数据库架构。随着项目的发展和设计的演化，需要更新这些表格以反应新的代码，或重构代码直到再次符合这些表格。如此一来，我们总是能在添加新功能之前就知道它们所属的组件。

如果能从本文学到点想法的话，我希望是：

**设计模式仅仅是建议；如何将其转化成实际的设计以满足你的需求，完全是由你来决定的。**

## 关于作者



**Jonathan Allen** 从 2006 年开始为 InfoQ 撰写新闻报道，现在是.NET 社区的主编。如果你对为 InfoQ 撰写新闻或教学文章感兴趣，可以通过 [jonathan@infoq.com](mailto:jonathan@infoq.com) 联系到他。

原文链接：<http://www.infoq.com/cn/articles/View-Model-Definition>

相关内容：

- [将 JavaScript 测试集成到开发工作流中](#)
- [.NET 工具和实践调查结果](#)
- [SpringOne China 2012](#)
- [通过.NET 和 Windows Phone 8 的 SDK 整合 SkyDrive](#)

## 推荐文章 | Articles

# 使用并行计算大幅提升递归算法效率

作者 [王峰](#)

## 前言

无论什么样的并行计算方式，其终极目的都是为了有效利用多机多核的计算能力，并能灵活满足各种需求。相对于传统基于单机编写的运行程序，如果使用该方式改写为多机并行程序，能够充分利用多机多核cpu的资源，使得运行效率得到大幅度提升，那么这是一个好的靠谱的并行计算方式，反之，又难使用又难直接看出并行计算优势，还要耗费大量学习成本，那就不是一个好的方式。

由于并行计算在互联网应用的业务场景都比较复杂，如海量数据商品搜索、广告点击算法、用户行为挖掘，关联推荐模型等等，如果以真实场景举例，初学者很容易被业务本身的复杂度绕晕了头。因此，我们需要一个通俗易懂的例子来直接看到并行计算的优势。

数字排列组合是个经典的算法问题，它很通俗易懂，适合不懂业务的人学习，我们通过它来发现和运用并行计算的优势，可以得到一个很直观的体会，并留下深刻的印象。问题如下：

请写一个程序，输入M，然后打印出M个数字的所有排列组合（每个数字为1，2，3，4中的一个）。比如：M=3，输出：

1 , 1 , 1

1 , 1 , 2

.....

4 , 4 , 4

共 64 个

注意：这里是使用计算机遍历出所有排列组合，而不是求总数，如果只求总数，可以直接利用数学公式进行计算了。

## 一、单机解决方案：

通常，我们在一台电脑上写这样的排列组合算法，一般用递归或者迭代来做，我们先分别看看这两种方案。

### 1) 单机递归

可以将  $n$  ( $1 \leq n \leq 4$ ) 看做深度，输入的  $m$  看做广度，得到以下递归函数（完整代码见[附件 CombTest.java](#)）

```
public void comb(String str){  
    for(int i=1;i<n+1;i++){  
        if(str.length()==m-1){  
            System.out.println(str+i);  
            total++;  
        } else {  
            comb(str+i);  
        }  
    }  
}
```

但是当  $m$  数字很大时，会超出单台机器的计算局限导致缓慢，太大数字的排列组合在一台计算机上几乎很难运行出，**不光是排列组合问题，其他类似遍历求解的递归或回溯等算法也都存在这个问题，如何突破单机计算性能的问题一直困扰着我们。**

### 2) 单机迭代

我们观察到，求的  $m$  个数字的排列组合，实际上都可以在  $m-1$  的结果基础上得到。

比如  $m=1$ ，得到排列为  $1,2,3,4$ ，记录该结果为  $r(1)$

$m=2$ ，可以由  $(1,2,3,4)^* r(1) = 11,12,13,14,21,22,\dots,43,44$  得到，记录该结果为  $r(2)$

由此， $r(m) = (1,2,3,4)^* r(m-1)$

如果我们从 1 开始计算，每轮结果保存到一个中间变量中，反复迭代这个中间变量，直到算出  $m$  的结果为止，这样看上去也行，仿佛还更简单。

但是如果我们估计一下这个中间变量的大小，估计会吓一跳，因为当  $m=14$  的时候，结果已经上亿了，一亿个数字，每个数字有 14 位长，并且为了得到  $m=15$  的结果，我们需要将  $m=14$  的结果存储在内存变量中用于迭代计算，无论以什么格式存，几乎都会遭遇到单台机器的内存局限，如果排列组合数字继续增大下去，结果便会内存溢出了。

## 二、分布式并行计算解决方案：

我们看看如何利用多台计算机来解决该问题，同样以递归和迭代的方式进行分析。

### 1) 多机递归

做分布式并行计算的核心是需要改变传统的编程设计观念，将算法重新设计按多机进行拆分和合并，有效利用多机并行计算优势去完成结果。

我们观察到，将一个  $n$  深度  $m$  广度的递归结果记录为  $r(n,m)$ ，那么它可以由  $(1,2,\dots,n)*r(n,m-1)$  得到：

$$r(n,m) = 1r(n,m-1) + 2r(n,m-1) + \dots + nr(n,m-1)$$

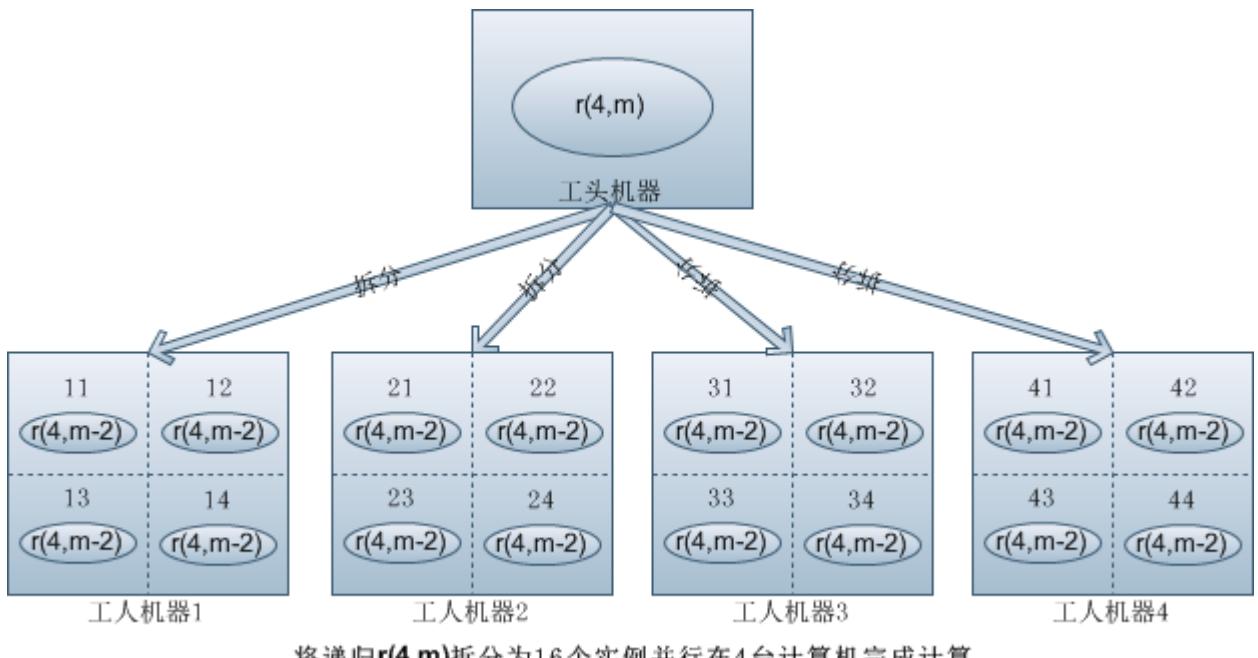
假设我们有  $n$  台计算机，每台计算机的编号依次为 1 到  $n$ ，那么每台计算机实际上只要计算  $r(n,m-1)$  的结果就够了，这里实际上将递归降了一级，并且让多机并行计算。

如果我们有更多的计算机，假设有  $n*n$  台计算机，那么：

$$r(n,m) = 11r(n,m-2) + 12r(n,m-2) + \dots + nnr(n,m-2)$$

拆分到  $n*n$  台计算机上就将递归降了两级了

**可以推断，只要我们的机器足够多，能够线性扩充下去，我们的递归复杂度会逐渐降级，并且并行计算的能力会逐渐增强。**



将递归  $r(4,m)$  拆分为 16 个实例并行在 4 台计算机完成计算

这里是进行拆分设计的分析是假设每台计算机只跑 1 个实例 ,实际上每台计算机可以跑多个实例 ( 如上图 ) ,我们下面的例子可以看到 ,这种并行计算的方式相对传统单机递归有大幅度的效率提升。

这里使用 fourinone 框架设计分布式并行计算 ,第一次使用可以参考[分布式计算上手 demo 指南](#) ,开发包下载地址 :  
<http://www.skycn.com/soft/68321.html>

ParkServerDemo : 负责工人注册和分布式协调

CombCtor : 是一个包工头实现 ,它负责接收用户输入的 m ,并将 m 保存到变量 comb ,和线上工人总数 wnum 一起传给各个工人 ,下达计算命令 ,并在计算完成后累加每个工人的结果数量得到一个结果总数。

CombWorker : 是一个工人实现 ,它接收到工头发的 comb 和 wnum 参数用于递归条件 ,并且通过获取自己在集群的位置 index ,做为递归初始条件用于降级 ,它找到一个排列组合会直接在本机输出 ,但是计数保存到 total ,然后将本机的 total 发给包工头统计总体数量。

运行步骤 :

为了方便演示,我们在一台计算机上运行:

1. 启动 ParkServerDemo：它的 IP 端口已经在配置文件的 PARK 部分的 SERVERS 指定。
2. 启动 4 个 CombWorker 实例：传入 2 个参数，依次是 ip 或者域名、端口（如果在同一台机器可以 ip 相同，但是端口不同），这里启动 4 个工人是由于  $1 \leq n \leq 4$ ，每个工人实例刚好可以通过集群位置 index 进行任务拆分。
3. 运行 CombCtor 查看计算时间和结果

下面是在一台普通 4cpu 双核 2.4Ghz 内存 4g 开发机上和单机递归 CombTest 的测试对比

	M=14	M=15	M=16	CPU 利用率
单机递归计算	10 秒	41 秒	169 秒	29%
单机并行计算	6 秒	26 秒	112 秒	99%
多机并行计算	按机器数量成倍提升效率			99%

通过测试结果我们可以看到：

1. 可以推断，由于单机的性能限制，无法完成 m 值很大的计算。
2. 同是单机环境下，并行计算相对于传统递归提升了将近 1.6 倍的效率，随着 m 的值越大，节省的时间越多。
3. 单机递归的 CPU 利用率不高，平均 20-30%，在多核时代没有充分利用机器资源，造成 cpu 闲置浪费，而并行计算则能打满 cpu，充分利用机器资源。
4. 如果是多机分布式并行计算，在 4 台机器上，采用  $4 \times 4$  的 16 个实例完成计算，效率还会成倍提升，而且机器数量越多，计算越快。
5. 单机递归实现和运行简单，使用 c 或者 java 写个 main 函数完成即可，而分布式并行程序，则需要利用并行框架，以包工头+多个工人的全新并行计算思想去完成。

## 2) 多机迭代

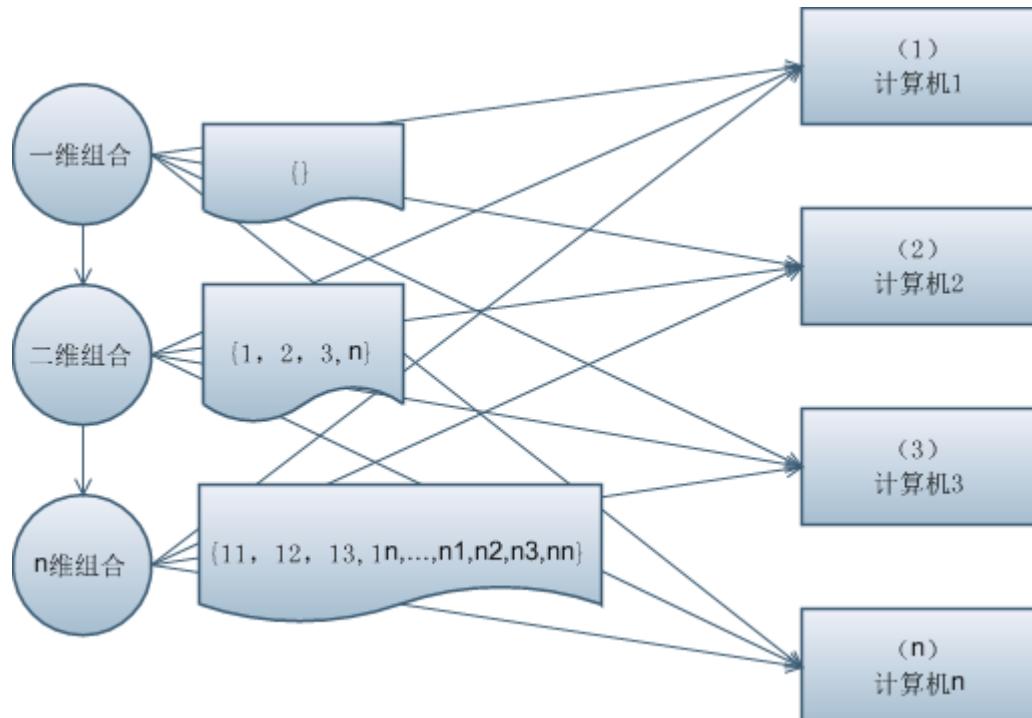
我们最后看看如何构思多机分布式迭代方式实现。

思路一：

根据单机迭代的特点，我们可以将 n 台计算机编号为 1 到 n

第一轮统计各工人发送编号给工头，工头合并得到第一轮结果 {1, 2, 3, ..., n}

第二轮，工头将第一轮结果发给各工人做为计算输入条件，各工人根据自己编号累加，返回结果给工头合并，得到第二轮结果：{11, 12, 13, 1n, ..., n1, n2, n3, nn}



这样迭代下去，直到 m 轮结束，如上图所示。

但很快就会发现，工头合并每轮结果是个很大的瓶颈，很容易内存不够导致计算崩溃。

思路二：

如果对思路一改进，各工人不发中间结果给工头合并，而采取工人之间互相合并方式，将中间结果按编号分类，通过 receive 方式（工人互相合并及 receive 使用可参见 [sayhello demo](#)），将属于其他工人编号的数据发给对方。这样一定程度避免了工头成为瓶颈，但是经过实践发现，随着迭代变大，中间结果数据越来越大，工人合并耗用网络也越来越大，如果中间结果保存在各工人内存中，随着 m 变的更大，仍然存在内存溢出危险。

思路三：

继续改进思路二，将中间结果变量不保存内存中，而每次写入文件（详见[Fourinone2.0 对分布式文件的简化操作](#)），这样能避免内存问题，但是增加了大量的文件 io 消耗。虽然能运行出结果，但是并不高效。

总结：

或许分布式迭代在这里并不是最好的做法，上面的多机递归更合适。由于迭代计算的特点，需要将中间结果进行保存，做为下一轮计算的条件，如果为了利用多机并行计算优势，又需要反复合并产生中间结果，所以导致对内存、带宽、文件 io 的耗用很大，处理不当容易造成性能低下。

我们早已经进入多 cpu 多核时代，但是我们的传统程序设计和算法还停留在过去单机应用，因此合理利用并行计算的优势来改进传统软件设计思想，能为我们带来更大效率的提升。

**下载完整代码点击[这里](#)**

**原文链接：**

<http://www.infoq.com/cn/articles/parallel-computing-improve-recursive-algorithm-efficiency>

**相关内容：**

- [2012.3.16 微博热报：生物学角度看并行计算与支持向量机](#)
- [基于 Spring Batch 的大数据量并行处理](#)
- [Node.js 获得企业开发者青睐](#)
- [Paul King 谈 Groovy 生态环境](#)
- [对象已死？](#)

## 推荐文章 | Articles

# James Ward 谈使用 HTML5 和 Java 开发客户端/服务器应用

作者 [Srinivas Penchikala](#) 译者 [臧秀涛](#)

谈到应用开发，不管是客户端/服务器类应用、传统的 Web 应用还是移动 Web 应用，最近的趋势是使用像流式网格布局（Fluid Grid Layout）和响应式网页设计（Responsive Web Design）这样的模式，以及像 HTML5、CSS3 和 JavaScript（客户端和服务器都支持）这样的技术。

此外，用于构建客户端现代 Web 应用（包括桌面应用和移动应用）的工具发展和变化都非常迅速。十年之前为了在服务器端处理 HTML UI 而构建的一切，现在都在客户端重新实现了，如 MVC 框架、模板库和组件库等。jQuery 是大多数现代应用的基础，因为从某种程度上说，它有点像 JavaScript 标准库。MVC 我们有很多选择，但是像 Backbone.js 这样的框架是越来越引人注目了。客户端模板库也有很多。其中有些技术既可以工作于客户端，又可以工作于服务器端；如果有的用户的浏览器中没有现代或快速的 JavaScript 引擎，这些技术是派得上用场的。

在今年的 [JavaOne 2012 会议](#) 上，来自 TypeSafe 的 James Ward 做了一个[讲座](#)，内容就是使用 HTML5 和 Java 技术开发客户端/服务器类应用。

InfoQ 采访到了 James，探讨了他在这次会议上的讲座以及这种新的应用开发趋势。

### InfoQ：请问开发者对基于 Java 的 HTML5 框架有哪些要求？



James：现代 Web 应用的新模型与老式的客户端/服务器架构类似，不过这时客户端变成了支持 JavaScript、CSS 和 HTML 的浏览器，而服务端则是 RESTful JSON 服务。在寻找现代 Java Web 框架时，你需要关注以下三个方面：

1. 这一 Web 框架是不是对构建客户端和服务器都有帮助？现代 Web 框架通常会提供“Asset Compilers”或“Asset Pipelines”，辅助开发者

编写 JavaScript、CoffeeScript 和 LESS。例如，Play Framework 所带的 Asset Compiler，将服务端代码（Java 或 Scala）和客户端代码（JavaScript、CoffeeScript 和 LESS）的编写体验统一起来。

2. 能否像管理服务端依赖一样管理客户端依赖？越来越多的代码正在移向客户端（JavaScript 和 CSS 等）。这意味着 jQuery、Bootstrap 和 Backbone 等客户端库的使用也会越来越多。因此，我们需要像管理服务端依赖一样管理客户端依赖（及其传递依赖）。我创建了“WebJars”项目（[www.webjars.org](http://www.webjars.org)），把常见的客户端库打包成 Jar 文件，这样就可以将其包含到 Maven（或其他构建工具）依赖列表中了。
3. 创建 RESTful JSON 服务是不是很容易？RESTful JSON 是目前向客户端暴露数据和服务的事实标准；不需要考虑客户端是移动应用还是现代 Web 应用（包括桌面或移动应用）。从 HTTP 动词和路径向后端服务的映射应该非常简单，而且是声明式的。大多数现代 Web 框架（如 JAX-RS、Play Framework、Spray 和 BlueEyes 等）都支持这一点。

**InfoQ：你能否谈一下在使用 Java 和 HTML5 开发客户端/服务器或移动 Web 应用时，在设计方面应该考虑哪些因素？**

 **James**：新的现代 Web 架构与传统的 Web 应用是非常不同的。要理解这些新架构所考虑的因素，最好是听一听那些已经成功地以这种方式构建过应用的人是怎么说的。下面是两个很好的资源，它们就是由成功构建过现代 Web 应用的人提供的：

视频：[Typesafe Console Architecture and Design](#)

博客：[The Trello Tech Stack](#)

**InfoQ：你的讲座中提到了无状态 Web 层架构。你能否更多地讨论一下，它对移动 Web 应用有何帮助？**

 **James**：现代 Web 应用的优势是，UI 状态从服务器移到了客户端。无状态的 Web 层使如下四个方面得以简化：

Web 层不包含状态时，水平伸缩很容易。这使 Web 层可以独立并即时地伸缩。

在无状态的 Web 层中，持续交付也更容易了，因为部署新版本时无需担心会话状态。

RESTful 服务应该是无状态的。每个请求都是原子的，不依赖于前面的任何请求。

不管是采用自动化测试（如单元测试或功能测试），还是开发者在其浏览器中通过点击刷新的方式测试新代码，无状态 Web 应用测试起来都要容易得多。在构建 Play Framework 应用时，我的典型开发周期很简单：编辑代码，保存，然后重新加载浏览器或重新运行自动化测试。这种开发过程非常有效，部分原因就是 Play Framework 应用是无状态的。

**InfoQ：请问当使用 Java 和 HTML5 创建移动 Web 应用时，开发者有哪些需要注意的地方？**

 **James**：在做出技术决策之前一定要理解用户。如果用户的浏览器不支持比较现代的功能，你可能需要退一步，使用特定的 API 和库。你的用户是哪些人，他们的设备和客户端能提供哪些功能，技术上你可能需要根据这些信息折中一下。

**InfoQ：你的讲座中也谈到，浏览器就是新的平台。你能否讨论一下这种新趋势？这种趋势对客户端/服务器或者移动 Web 应用的开发和部署方式有何影响呢？**

 **James**：Web 浏览器是通用的客户端平台。当然，它并不完美，但是它无处不在，而且一直处于发展之中。大约十年前，我们将 UI 移到了浏览器中，但只是将浏览器用做渲染标记（UI+数据）的非智能终端，而标记仍是在 Web 服务器上生成的。这种方式的优点是，用户什么都不需要安装，即可使用任何应用。缺点是每次用户交互 UI 都需要重新加载页面。Ajax 这种革命性的技术出现之后，这种局面有所改观，部分 UI 逻辑移向了客户端。现代 Web 和移动应用走得更远，将整个 UI 都移到了客户端。此举提供了最好的用户体验和性能。对于可能需要离线工作或者使用原生 UI 的移动应用而言，这也非常合适。

**InfoQ：对于响应式设计和流式网格布局等移动 Web 设计理念，请问你有什么看法？**

 **James**：过去，我们的 Web 应用主要是针对一种设备配置而设计的，也就是台式机/笔记本。这些设备大多具有相似的屏幕尺寸，所以 Web 应用开发

人员不需要考虑其他设备配置。移动和平板设备的增长改变了这一切。开发者需要确保其应用支持种类繁多的设备配置。响应式设计和流式网格布局就是根据设备配置修改应用布局和外观的策略。其中有些策略用到了 CSS Media Queries，让浏览器基于设备配置（如屏幕尺寸）选择不同的 CSS 文件。使用这些策略时，主要还是根据你构建的是什么以及用户需要什么。在不同的设备配置之间，如果只是 UI 和布局有轻微变化，这些方式可以工作地很好。否则，你可能需要查看其他更为重型的策略了。

**InfoQ：移动 Web 开发作为一种新趋势，也使其他技术——如 Rest Web 服务和用于服务器和客户端之间交换数据的 JSON——更为流行了。你能否谈一下这些技术？在整个移动 Web 开发中，这两种技术是如何互补的？**

**James** :传统的 Web 应用结合了 UI 和数据。在只有一种客户端，而且 UI 和数据的聚合发生在服务器上时，传统的 Web 应用表现良好。而在现代 Web 和移动应用中，可能有多个客户端使用同样的数据。RESTful 服务提供了一种简单方法，动词和路径可以通过 HTTP 协议映射到资源或数据。JSON 是一种数据序列化方法（将内存中的对象转变为可传输的数据）。XML 是另一种数据序列化方法。多数现代应用都选择了 JSON，因为它极为简单，而且大多数技术对它都有良好地支持。REST 和 JSON 组合起来非常简单，还能支持大量的服务器和客户端技术。

**InfoQ：请问 JavaScript 在哪方面比较适合这种新的客户端/服务器与移动开发范型？**

**James** :JavaScript 是现代 Web 应用的客户端编程语言。但是 JavaScript 并不完美，所以出现了很多可以编译为 JavaScript 的语言，如 CoffeeScript、TypeScript 和 Dart 等。不管怎样，在现代 Web 应用中，在客户端实际运行 UI 的代码都是 JavaScript。

他也谈到了 Web 与移动应用开发的未来。

**James** :使用 HTML5 和 Java 开发 Web 和移动应用，未来是非常光明的。客户端和服务器端都有大量的技术可供选择，而且这些技术都发展得非常快。所有这一切最终将帮助我们更容易且更快速地构建更好的应用。对 Web 开发者而言，这是激动人心的时刻。

## 关于受访人：



**James Ward** 是 Typesafe 的一名技术推广者。目前，他主要关注的是教给开发者如何将 Java、Play! 和 Scala 应用部署到云中。James 经常出席全世界的各种技术会议，如 JavaOne、Devoxx，以及很多其他的 Java 聚会。James 与 Bruce Eckel 合著了《First Steps in Flex》一书。他也发布了大量的教学视频、博客和技术文章。从上个世纪 80 年代使用 Pascal 和汇编语言编程开始，James 发现了他对编写代码的热情。之后在 90 年代初，他开始使用 HTML、Perl/CGI 和 Java 进行 Web 开发。在 2004 年为 Pillar Data Systems 构建了一个基于 Flex 和 Java 的客户服务门户系统之后，他成了 Adobe 的一名 Flex 技术布道者。你可以发现他在通过[@JamesWard](#)这个 ID 玩 Twitter，他也在[StackOverflow](#)回答问题，还在[这里](#)提交代码。

### 原文链接：

<http://www.infoq.com/cn/articles/client-server-application-development-with-html5-and-java>

### 相关内容：

- [Atmosphere 1.0：支持 Java/JavaScript 的异步通信框架](#)
- [w3ctech 2011 JavaScript 专题会议（广州站）综述](#)
- [OmniFaces：针对 Java Server Faces 的工具库](#)

# 架构师

[www.infoq.com/cn/architect](http://www.infoq.com/cn/architect)

每月8号出版

时刻关注软件开发领域的变化与创新

## 架构师

11月 ARCHITECT

特别专题  
光棍节狂欢的背后——  
电商系统深探  
1号店B2C电商系统深造之路  
百万点推荐引擎——从需求到架构  
麦当劳购物系统浅谈分享  
REST的远程API设计案例  
大型Rails与VoIP系统架构  
与部署实践  
什么是Node.js  
扩展Oozie  
浅谈dojox中的一些小工具

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

## 架构师

10月 ARCHITECT

特别专题  
大数据时代  
大数据  
大数据时代的数据管理  
阿里巴巴数据架构设计经验与挑战  
大数据时代的创新者们  
关系数据库还是NoSQL数据库  
向Java开发者介绍Scala  
HTML 5 or Silverlight?  
解析JDK 7的Garbage-First收集器  
了解云计算的基础

Steve Jobs  
1955-2011

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

## 架构师

9月 ARCHITECT

特别专题  
QCon全球企业大会精华点滴  
QCon在中国的三年回顾  
麦肯锡对阿里巴巴国际站架构演进  
畅销书《IPS》和《技术流年》  
新浪微博团队建设的虚与实  
沐泽宁谈主观决策架构  
跟着李航学Oozie  
如何查看我的订单—  
REST的远程API设计案例  
通用系统思考，走上改善之路  
Redis内存使用优化与存储

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

## 架构师

8月 ARCHITECT

特别专题  
云计算的安全风险  
圆桌会议：云计算的安全风险  
设计一种云级别的身份认证结构  
云应用和平台的现状：  
云采纳者如是说...

Java虚拟机家族考  
专家视角看IT转型构  
为什么使用 Redis及高产品定位  
架构演化之谜

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

## 架构师

7月 ARCHITECT

特别专题  
深入理解Node.js  
为什么要用后端工程语言Node.js  
虚拟机设计：Node.js生态系统之  
秘密，操作实践  
使用Java连接JavaScript并行编程  
Node.js的起源和实践应用—  
专访Node.js创始人Ryan Dahl

Java深度剖析十：Java对集群划分与热切换  
将数据打散之一：关于松散的数据设计  
分布式平台的组件化PaaS  
社区驱动的开源计划  
来自Padmanab的真言

InfoQ 每月8号出版

## 新品推荐 | Product

### 使用 Apache Ambari 管理 Hadoop

作者 Boris Lublinsky 译者 臧秀涛



Hortonworks 公司负责公司战略的副总裁 Shaun Connolly 在一篇新的博客文章中探讨了 Apache Ambari 孵化器项目的重要性，也谈到了该项目在 2012 年取得的主要里程碑：简化集群供应、预先配置关键运维指标、作业执行可视化、RESTful API 和直观的用户界面。

原文链接：<http://www.infoq.com/cn/news/2012/12/ambari>

### Eclipse Orion：基于浏览器的 Web 应用程序编辑器

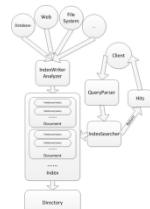
作者 Abel Avram 译者 廖煜嵘

Eclipse 基金会发布了 Orion 1.0，这是一款基于浏览器的编辑器，支持以 Javascript、HTML 和 CSS 编写 Web 应用程序。

原文链接：<http://www.infoq.com/cn/news/2012/12/Eclipse-Orion-1>

### Lucene.Net:一个顶级 Apache 项目和它的未来

作者 Abel Avram 译者 朱伟健



Lucene.Net 是 Lucene 全文检索开发库的 C#移植版本。其已经从孵化工场毕业并成为一个顶级的 Apache 项目。本文内容是采访 Prescott Nasser 时其谈到的有关该项目和 Solr.NET 的未来。

原文链接：<http://www.infoq.com/cn/news/2012/12/Lucene-net>

### 移动站点生成便捷之路：百度 SiteApp

作者 彭超



PC 终端和移动终端在屏幕尺寸、输入方式、网络状况和浏览器能力等方面存在较大差异，导致了传统 PC 互联网页面在移动终端上的使用体验较差。作为其他中小网站的风向标，各大门户网站及知名网站，在过去两年中均都投入了相当的资源去打造自己的移动站点。缺少资源构建移动版本的网站如何通过简单手段生成自己的移动站点？昨晚，在中关村 3W 咖啡馆的西二旗夜话中，百度为大家详细介绍了他们在 9 月 3 日发布的百度七武器之一：SiteApp 服务。

**原文链接：**<http://www.infoq.com/cn/news/2012/12/baidu-siteapp>

## JetBrains 发布 IntelliJ IDEA 12

**作者 Abel Avram 译者 徐浩然**

JetBrains 发布了 IntelliJ IDEA 12。该版本包含了许多新特性，诸如更好的编译器，对 Java 8 的支持，Android UI 设计器，一种新外观，更好的 Spring 和 Play 2.0 支持，以及对大量语言和框架支持的增强。

**原文链接：**

<http://www.infoq.com/cn/news/2012/12/JetBrains-IntelliJ-IDEA-12>

## ASP.NET 实现了更好的加密算法

**作者 Roopesh Shenoy 译者 陈菲**

.NET 4.5 引进了一系列对 ASP.NET 处理加密的改进，并带来了新的 Protect 和 Unprotect API，以及众多不易察觉的改变。Levi Broderick 通过一系列文章介绍了其动机、改变及兼容问题。

**原文链接：**

<http://www.infoq.com/cn/news/2012/12/aspnet-better-cryptography>

## Atmosphere 1.0 :支持 Java/JavaScript 的异步通信框架

**作者 Kostis Kapelonis 译者 臧秀涛**

Atmosphere 1.0 是一个新的 Java/Scala/Groovy 框架，它试图将 Web 浏览器与应用服务器之间的通信抽象出来。在 Web Socket、HTML5 服务器端事件和其他特定于应用服务器的解决方案可用时，该框架可以透明地支持，此外还可将长轮询作为一种备选方案。

**原文链接：**<http://www.infoq.com/cn/news/2012/12/atmosphere-1>

## Orubase :为 Windows Phone、Android 和 iOS 平台开发混合本地手机应用程序

**作者 Anand Narayanaswamy 译者 方盛**



Orubase 是一个开发框架，通过该框架开发者可以使用 ASP.NET MVC、HTML5、和 JavaScript 来为 Windows Phone、Android 和 iOS 平台开发混合本地手机应用程序。开发者还可以复用.NET 应用程序中的现有的业务层和数据层。

**原文链接：**<http://www.infoq.com/cn/news/2012/12/orubase>

## Oracle 表彰伦敦 Java 社区，Gosling 现身 JavaOne

**作者 Michael Floyd 译者 赵震一**

在前些天 Java One 会议的 Java 社区主题中，伦敦 Java 社区因在 Adopt a JSR program 中的创新而成为今年 Duke's Choice Awards ( 译者注：一种奖项 ) Oracle 荣誉的获得者。其他获奖者包括 Apache Hadoop 项目，AgroSense, Duchess, NATO 和 Parleys.com。而 Ram Kashyap 成为今年的学生赢家

**原文链接：**<http://www.infoq.com/cn/news/2012/12/Duke-Choice-Awards>

## Java 8 for Raspberry Pi 开发者预览版



作者 Fabian Lange 译者 侯伯薇

Oracle 上周发布了针对 ARM 处理器的 Java 8 开发者预览版。这个版本特别针对在 Raspberry Pi 设备上运行 JavaFX 做了剪裁。

原文链接：<http://www.infoq.com/cn/news/2012/12/java-raspberry-pi>

## RavenDB 性能提升、增加了工具与异步 API 支持

作者 Roopesh Shenoy 译者 张龙



近日，RavenDB 2.0 RC 发布了，该版本提供了更好的工具、新的 Changes() API、Eval Patching、更好的索引性能以及其他几处改进等。

原文链接：<http://www.infoq.com/cn/news/2012/12/ravendb-2-0>

## GE Energy 利用 InvokeDynamic 指令将 Magik 移向 JVM

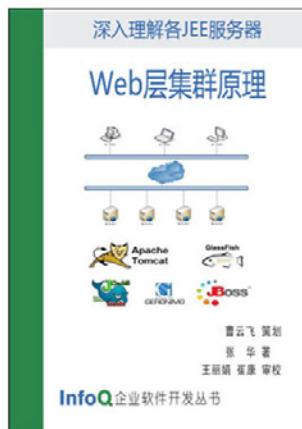
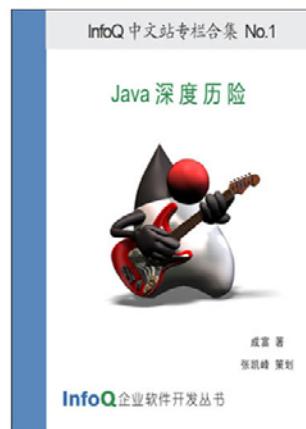
作者 Charles Humble 译者 臧秀涛

今年 7 月，GE 能源管理业务（ GE Energy Management ）的数字能源部门（ Digital Energy ）透露，他们正在将 Magik （一种受 Smalltalk 启发而设计的编程语言）从其专有的虚拟机 MagikSF 移植向 JVM 。移植工作早已开始， InfoQ 采访了该项目的团队主管、架构师 George Marrows ，以便获得更多信息。

原文链接：<http://www.infoq.com/cn/news/2012/12/magik-jvm-port>

# InfoQ 软件开发丛书

欢迎免费下载



商务合作: [sales@cn.infoq.com](mailto:sales@cn.infoq.com)

读者反馈/内容提供: [editors@cn.infoq.com](mailto:editors@cn.infoq.com)

# 封面植物

## 南湖柳叶菜



南湖柳叶菜为台湾特有柳叶菜科多年生草本植物。南湖柳叶菜植株低矮，花形硕大，呈粉红色或紫红色，白天时花朵开展，夜晚则呈闭合状态。为冰河时期孑遗的种类，因冰河退却而局限生长於少数的高山上。文化资产保存法所指定公告的台湾珍贵稀有动植物只有 5 种，南湖柳叶菜便是

其中之一。形态特征：多年生矮小草本，几成莲座状，茎极短，高 3 - 8 厘米。叶密生于茎上，近对生，顶端的叶互生，椭圆形或近圆形，全缘，先端圆，基部楔形或圆形，长 0.6 - 2.1 厘米，宽 0.3 — 1.4 厘米；叶柄不明显，长 1 - 2 毫米。花两性，生于茎上部叶腋内，花托延伸于子房之上呈萼管状，花萼裂片 4，花瓣 4，玫瑰紫色；雄蕊 8，不等长，4 长 4 短；子房下位，4 室，柱头 4 裂。蒴果长而狭，长 2.5 — 3.1 厘米；种子多数，有束毛，褐色，倒卵圆形，长 1.2 — 1.3 毫米，种翅长 5 — 6 毫米，白褐色，宿存。种群现状：稀有种。本种为台湾特有植物，目前仅在中央山脉北部南湖大山残存。由于南湖大山海拔达 3740 米，至今还保存着最完整的冰川遗迹，在其亚热带高山上，残遗着丰富的高山植物，其中南湖柳叶菜为珍稀植物之一，仅生长在高山上，数量较少，处于濒临绝灭的境地。保护措施：产地南湖大山非自然保护区，且系登山旅游之地。因此，应采取必要的保护措施，绝对禁止过量采折和破坏，开展繁殖试验，扩大分布范围。亦可少量移栽，在栽培条件下加以保护。

1kg.org 多背一公斤

爱自然 | 更爱孩子





# 架构师 1 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

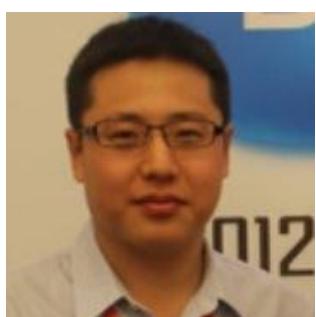
读者反馈：[editors@cn.infoq.com](mailto:editors@cn.infoq.com)

投稿：[editors@cn.infoq.com](mailto:editors@cn.infoq.com)

InfoQ 中文站新浪微博：

<http://weibo.com/infoqchina>

商务合作：[sales@cn.infoq.com](mailto:sales@cn.infoq.com) 15810407783



本期主编：贾国清，InfoQ 中文站总编

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。