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

NoSQL一览

书摘和采访：图数据库

MetaModel：统一的数据访问

结合MongoDB深入浅出Symfony2

## 主编有话说

云计算、SOA主编马国耀谈专栏建设

理解本真的REST架构风格

云计算时代的运维与安全

1688域名切换——看我空中换引擎

基于AWS技术实现发布/订阅服务

Java SE 8: Lambda表达式

InfoQ

每月8号出版

# 卷首语

## 回归原始

这两年设计领域给我给很大的撼动，当我们好不容易刚上拟物的浪潮，把能添加的装饰都添加上去以后， 科技大头们突然风头回转，扁平化的设计（Flat UI）开始兴起。不论是苹果的iOS7、微软的Windows UI亦或是谷歌的 Holo都在这条路上走出了自己的风格，且不谈新的设计缓解“围观”群众视觉疲劳的作用，其对信息原点凸显的含义却显而易见：

> 这个时代我们接触了太多的信息，但是反而丢失了对信息本身的关注，扁平的设计让内容回归，一切有回到了最简单的那个起点， 犹如原研哉的ExFormation理念。

我们极力克服荷尔蒙的影响，从理性的角度出发去解决问题，但是在解决的过程中最容易把当初的出发点给忘掉，这就是本末倒置吧。其实我们发现在技术构架的场景中也随处可以见到这种情况，我们学习了一种强大、新潮的语言、遇到了一个牛叉的类库， 然后我们号召在所有的设计中都使用这个东西，随着这些“装饰”效果很强的组件被添加到原本的问题解决方案中，一切又变得复杂了， 我们浪费更多的时间在讨论用MongoDB或是Redis、用ZeorMQ还是RabbitMQ……然后是各种新的语言的试错与吐槽， 原本的问题变得复杂了。

为何不停下来想想我们的问题或关注点是什么？我们擅长的架构是怎样的？即使在大家看起来是老掉牙的语言PHP也能写出Facebook、 新浪微博这种大型的系统应用，但是你会问道，这些系统还有很多牛叉的组件呀，所以我们才用的呀！是的，但是需要注意的是他们不是一开始 就在造轮子，而是遇到问题以后从痛点出发给出了一个原有的技术不能解决的方案，这个就是技术的进步，是循序渐进的过程。回归到原点不是 固步自封、不敢尝试新的东西，而是“逢山开路，遇水搭桥” ，但不要忘记行走。之前在ADC大会上采访了支付宝的前端大牛伯约（@oldfish），最后我问了他这样一个问题：你对新人的建议是什么？他脱口而出我意料 的答案：坚持与积累，认定了一个事情坚持下去，积累这个方面的经验。这和之前采访陈皓（@左耳朵耗子）时得到的答案是一样的，每个我们眼中的牛人 都有一个坚持的原点，虽然复杂世界精彩纷呈，但是在一条简单的路上，多少人能走到头？回归原始，勿忘初心；

# InfoQueue

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


## 人物 | People

从找需求到自动感知：阿里巴巴核心系统研发部的发展历程

## 观点 | View

如何让代码并发效率更高
《天空之城》助Twitter刷新纪录，新架构功不可没

## 本期专题：NoSQL一览 | Topic

书摘和采访：图数据库
MetaModel——跨多种数据存储提供统一的数据访问
深入浅出Symfony2 - 结合MongoDB开发LBS应用

## 推荐文章 | Article

1688域名切换——看我空中换引擎
基于AWS技术实现发布/订阅服务
Java SE 8: Lambda表达式

## 特别专栏 | Column

主编有话说：云计算、SOA专栏主编马国耀谈专栏建设
理解本真的REST架构风格
云计算时代的运维与安全

## 避开那些坑 | Void

从光大证券交易系统8.16事件说起
提升.NET应用程序启动速度

## 新品推荐 | Product

Google开放HTML5解析库Gumbo的源代码
Bootstrap 3拥有全新视觉方案和更多的组件
来自百度的移动应用框架Clouda：快速开发，一键部署BAE
WinJS 2.0：更快的速度、更多的功能
Vim 7.4发布，带来了更快的正则表达式引擎及1000多项修复和小改进
移动开发技术周报：使用GenyMotion加速Android开发，iOS 7 UI设计技巧
Weblogic+Coherence+Tuxedo=云应用程序基础（CAF）

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

• 想成为技术**牛人**？ • 想获得技术**干货**？ • 想结识技术**圈内朋友**？

# 百度技术沙龙

**与技术大咖一起，讨论时下技术热点**

牛人 + 大咖 + 圈内朋友 + 热点 + 干货

百度技术沙龙第42期

**Node.js框架的实现与开发实践**

**9月14日，车库咖啡**

salon.**baidu**-tech.com

畅想 • 交流 • 争鸣 • 聚会

百度技术沙龙是由百度主办，InfoQ负责策划、组织、实施的线下技术交流活动，每月一期，每期一个主题，由2场演讲以及Open Space开放讨论环节组成。旨在为中高端技术人员提供一个自由的技术交流和分享的平台。每期沙龙会邀请1名百度讲师分享百度在特定技术领域的成果及实践经验，同时还会邀请1名优秀的互联网公司或企业技术负责人对同一话题进行分享。活动主要面向开发者、技术负责人、项目经理、架构师等IT技术人员。

新浪微博：infoqchina

Baidu百度 InfoQ

# 人物 | People

# 从找需求到自动感知：阿里巴巴核心系统研发部的发展历程

作者 杨赛

在2013年7月的阿里技术嘉年华上，阿里巴巴核心系统研发部的王琤（长仁）分享了他们团队在计算密集型应用优化上的一些经验（观看视频，PPT下载）。会后，InfoQ中文站跟长仁进行了一次交流，了解他们在需求处理、团队组成与分工、与其他部门的合作方式等方面的一些情况。

嘉宾简介：王琤/长仁（微博：@王王争），阿里巴巴核心系统部专用计算组负责人。在阿里巴巴致力于专用计算的推广及实践，以追求最佳性能及性耗比为目的为特定应用针对特定硬件架构进行算法及实现优化。在图像搜索、机器学习等领域，针对特定 CPU 微架构，GPGPU等不同体系结构进行关键算法及实现进行优化并获得了指数级的性能提升。同时负责淘宝JVM的开发工作，基于OpenJDK VM 为淘宝定制、优化更加贴近应用需求的专用JVM。

**InfoQ：长仁你好，感谢接受我们的采访！您刚才的分享涉及到很多底层的优化，那现在，优化团队是怎么去和业务部门交互的？**

> 长仁：现在流程不太一样。早期的时候，是我们去到处找需求。自己去找是很难的。
>
> 后来有一个突破，是因为我们核心系统有一个图像团队，他们意识到出了性能问题，主动要求我们来帮忙做一做优化。那么我们一做，就有了十倍的性能提升。
>
> 现在，需求方面有很大改观。一方面是由于我们有很多优化成果的积累，有了一定的口碑。另一方面，我们接触到越来越多计算密集型的应用，我们的工作就有了需求。由于我们有各种各样的成功案例，开始有越来越多的、阿里的其他的团队主动来找我们。
>
> 现在的流程是，兄弟团队找我们，和我们讲，应用是什么样子，现在有什么样的性能问题，然后我们就会跟着应用团队一起做profiling，帮他们进行优化，

上线测试。

未来的流程会不一样：我们现在正在做一个系统，System-wide profiling，也就是一个系统层面的直接，持续profiling，我们会把这些agent点布到每一台机器上。Google有一篇有关google wide profiling的论文。我们的方案也类似：自动感知哪些程序跑的慢，哪里跑得慢。我们现在正在做这个，已经有原型出来了，马上就要布到我们的MPI集群上。部署完了以后，将来就是谁也不用找谁了，系统一看就知道，这个程序这跑的比较慢，我们来去帮你优化。

**InfoQ：这个系统方面的profiling是不是有点像应用层面的监控？**

长仁：不是应用层面，不用应用改任何代码。我们在底层自动进行profiling，应用跑在上面，性能数据我们就能大体知道了，比如说每一个时钟周期内平均执行多少条指令，跟理想值比跑得是不是慢了，程序里头具体哪个函数占的时间最长，我们都能知道。所以未来的流程就是，我们主动通过系统一看，哪里有问题，我们来帮你修哪里。

**InfoQ：你们做的这些事情，是怎么让别人知道的呢？**

长仁：做分享啊。为什么我每次分享的PPT的最后一页都是广告？分享对于我们来说就是去做广告。今天的分享也是做广告，招聘广告：没有广告的分享不是好广告。

**InfoQ：你现在在阿里这边做性能优化，跟你以前在甲骨文做的事情有什么不一样？**

长仁：甲骨文完全不一样。我在甲骨文是在Application Product Group，应用产品的部门。我们在做的是一个自己的产品，叫SDE，Solution Development Environment，它是一个Solution的开发平台。这个平台跟企业级应用靠的更近，跟互联网应用没关系，跟密集计算更没关系，所以跟现在完全不一样，但是有很多是相通的。相通的就是，你了解的越底层、越深，对于开阔你的思路，解决问题的方法是更有好处的。

**InfoQ：你们团队的组成是怎样的？**

长仁：我们团队很有特点：绝大部分都是应届生。真正有工作经验加入到团队

的，目前为止，算上我，只有两个人，剩下的六个人全是应届生培养出来的。为什么？因为对于我们来说，社招生跟应届生区别不大。你想，我们优化JVM，全国有多少人做JVM的？第一，很难找；第二，应届生培养也是培养，社招来了没准也需要培养，而我们的应届生同学都是非常优秀的，经过千挑万选来的，他们的学习能力很强，经过几年的培养，有的已经成为国内的首屈一指的。举个例子，你们都知道莫枢（注：莫枢在QCon上做过JVM定制改进的分享），JVM在国内是顶尖的，那是我们团队的；还有洪熙，现在在GC这方面可能国内没有人比他更熟悉。

**InfoQ：团队的分工大概是怎样的状态？**

长仁：我们组就两大部分，专用计算跟JVM。JVM部分四位同学，剩下的都做负责计算方面的优化工作。分工还是很明确的，因为这两个领域差别比较大。我们是一个小团队，人不多，关键是干练。

**InfoQ：所以，当一个需求过来的时候，你们跟应用开发团队是如何配合的？**

长仁：流程刚才介绍过。客户来了，可能是各种各样的应用，如果是Java Application，就是JVM组去处理；如果是计算密集型的应用需要优化，就由计算组去支持。

**InfoQ：万一各种各样的应用都来找你们，你们不是得搞懂所有的应用？**

长仁：是的，需要有很强的学习能力，同时，如果我们的客户多到那个程度就太好了。

**InfoQ：你们团队的工作效果是如何评估的？性能提升了多少倍这样？**

长仁：这个问题特别好。为什么？一般的人会觉得，你们的评估标准是不是提升了十倍，几十倍这些，但恰恰不是这样。对于我来说，我们提升几倍十几倍几十倍都是家常便饭，但是这些从来都不是我们的KPI。

真正的成绩是什么？成绩就是用户真正的用上，在线上使上，产生了好的结果。为什么要这样？因为在一个互联网企业，做一个工作必须要有结果；而且互联网企业节奏是非常快的，没有时间让你做纯研究性的东西，你做的东西必须有人用上，带来好的效果，这是最重要的。

所以我们的要求是很严格的：第一，技术有难度；你二，还必须要有好的结果。其实回顾一下我们的时间都用在哪儿，对于我来说就是，1/3的时间做优化，1/3时间推广上线，还有一个1/3是培养新人。对于我们组的员工来说就是基本上一半一半，一半的时间在写代码做调优，一半的时间在找应用来推广，或者出去做广告。所以，很多外面的同学会觉得，你们做得东西真好，有那么好的效果；但是你真正来了以后就知道，优化本身不是最难的东西，最难的是让用户真正的使上，产生效果。有时候由于各种原因，你做的东西可能最后用不上。

**InfoQ：万一有的同学，他就是不擅长推广，但是他特别擅长做这个东西，这样也不行么？**

长仁：有这样的同学。但我们标准就是这样，看结果。学会推广是必要的技能，必须要做，所以我们的同学其实确实挺苦。有的同学性能虽然提升很多，因为推不上去，没有用上，没有结果，最终年底Review就会很低。

**InfoQ：底层这块是相对枯燥的领域，你们每年要如何吸引新鲜血液的加入？**

长仁：这个领域确实是比较小众的，每年的应届生里边就几个对口专业的有一些人才，这里面还不是所有的人都想来。

这个领域不像其他领域，比如出了一种新语言，或者Web编程一种新的设计模式，或者新的架构，大家都在聊。这些东西确实太底层，真正对它有兴趣的人很少，有机会来做这个的人也很少，两者叠加确实人很难找。

所以我们现在就是应届生培养，来了以后先“洗脑”，告诉他这个东西很重要——越没有先前观念的，他越能接受；另外一部分人，确实奇葩中的奇葩，天生就爱这个，那就最好。我们组三年，现在算上我才有八个人，每年转换的应届生就一到两个人。

吸引点有几个，阿里巴巴公司品牌是一个，同时，我们做的工作对于对这个领域有兴趣的同学会很有吸引力，因为在企业里专门从事这个领域的机会的确不多。

**InfoQ：您自己是属于对这方面有特别爱的？**

长仁：我目前看搞这块，就三类同学：第一，自发的有兴趣，就是喜欢。这个就属于天生爱，你让我改行，我也不适应。

第二类同学是被逼的。这类同学，实验室导师也好，或者工作领导，就是让他干这个，为了生计他也就干了。他可能不喜欢，但是他干了以后也有成长。

第三类是被启发的。他可能以前没意识到，然后当自己技术发展到一个瓶颈的时候，比如说一个程序员发展到架构师了，以后不知道哪些方面是发展的方向，比较迷茫，技术上也得不到突破。很多人会遇到这样一个技术的瓶颈期。然后他可能偶然的，比如偶然接触到一些硬件的知识，软件究竟在硬件上是怎么跑的，我们怎么能提升那么多倍性能，可能他就觉得这像打开了一扇大门，进入了一个新的领域，他觉得很好奇，开始在这个领域里探索摸索，这是被启发的。

像今天我的分享，我相信里面很多人可能听不懂，但是会给他一个启发，会让他觉得其实这个领域应该关注关注。

**原文链接：**[http://www.infoq.com/cn/news/2013/08/moving-to-system-wide-profiling](http://www.infoq.com/cn/news/2013/08/moving-to-system-wide-profiling)

# 观点 | View

# 如何让代码并发效率更高

作者 崔康

随着计算硬件的快速发展，多核多处理器已经广泛应用于企业和个人环境中，开发人员利用多线程技术努力提高软件的计算速度，资深系统架构师Gurudutt Kumar总结了如何让代码并发效率更高的实践经验。

Gurudutt首先列举了几种影响软件可伸缩性的问题：

- 效率低下的并行化：单片应用程序或软件无法有效使用可用的计算资源。您需要将应用程序组织成并行任务。在传统的不支持多线程的应用程序或软件中，我们会经常看到这个问题。这些应用程序在多核、多处理器、芯片多线程硬件上无法伸缩，并且无法实现更好的吞吐量。线程太多可能会和线程太少一样，都不会产生好的结果。
- 串行瓶颈：在多个线程或进程之间共享数据结构的应用程序可能会有串行瓶颈。为了保持数据完整性，可能必须使用锁定和串行化技术（例如，读取锁、读写锁、写入锁、自旋锁、互斥等）将这些共享数据结构的访问串行化。设计得效率低下的锁可能会由于多个线程或进程之间的高度锁争用而导致串行瓶颈，从而尝试获取锁。这可能会潜在地降低应用程序或软件的性能。应用程序的性能可能会随着核心或处理器数量的增加而降低。
- 对操作系统 (OS) 或运行时环境的过度依赖：您不能依赖操作系统、运行时环境或编译器来完成伸缩应用程序或软件所需的一切操作。但是，编译器和运行时环境可以帮助提供一定的优化，您不能依赖它们解决所有可伸缩性问题。例如，不能依赖 Java™ 虚拟机 (JVM) 通过自动并行来发现 Java 应用程序的最佳可伸缩的机会。
- 工作负载的不平衡可能是一个瓶颈：工作负载的不均匀分布可能导致无法有效地利用计算资源。您可能必须将较大的任务划分成可以并行运行的较小的任务，还可能必须将串行算法更改为并行算法，以便提高性能和可伸缩性。
- I/O 瓶颈：由于阻止磁盘输入/输出 (I/O) 或高网络延迟而导致的瓶颈可能会严重抑制应用程序的可伸缩性。

- 无效的内存管理：在多核平台上，因为有很多处理单元，因此纯计算可能非常廉价，并且主要内存可能也不是问题，因为它正在变得越来越大。但是，内存带宽一直是一个瓶颈，因为所有处理器核心都贡献了一个通用的总线。无效的内存管理可能导致一些难以检测到的性能问题，比如伪共享。

以“避免内存争用”为例，Gurudut做了解释：

> 在内存和缓存中，各种不同的核共享一个通用的数据区域，这需要在它们之间进行同步。当不同的核同时访问同一个数据区域时，会发生*内存争用*。在不同的核之间同步数据会因总线通信、锁定成本以及缓存缺失而有很大的性能损失。如果应用程序有多个线程，并且所有线程都更新或修改同一个内存地址，那么正如前面部分所讨论的那样，为了保持缓存一致性，可能会产生一次重大的乒乓效应。这会导致性能降低。

如何避免内存争用呢？Gurudut认为“不要在核之间共享可写入的状态”：

- 为了最大程度地减少内存总线通信，可以通过最小化共享位置/数据尽可能地减少核心交互，即使共享数据没有锁保护，而有一些硬件级别原子指令（如 Microsoft® Windows® 32 位平台上的 InterlockedExchangeAdd64）保护也是如此。
- 减少线程之间的内存争用的一个方法是从多个线程中消除对共享内存区域的更新。例如，即便是在多个线程需要更新全局计数器或累计总数（如统计数据）时，各个线程也可以保持线程本地总数，并让全局总数仅在需要时通过一个通用的线程进行更新。因此，在共享内存区域上的争用会大大减少。
- 趋向于减少锁争用的模式会减少内存通信，因为它是一个共享的可写入状态，该状态需要使用锁并产生争用。

对于锁争用，Gurudut认为要从“避免”和“减少”两个方面来解决这个问题：

**避免**

- 避免在数据结构中发生锁争用的方法之一是采用并发数据结构设计和无锁算法，这会消除锁以及传统的同步技巧（比如互斥）。有多种并发数据结构的设计并不需要利用同步机制，比如互斥。
- 无锁算法的一些示例如下：
  - 使用 相对论编程 的可伸缩并发哈希表：该技巧的最简单示例是

Read Copy Update (RCU)，它专用于 Linux 内核，大大提高了 Linux 内核的性能，并简化了 Linux 内核的代码。
  - 无锁可扩展有序分割的哈希列表：这个无锁递归可扩展哈希算法使用了无锁的链接列表，这些列表使用原子指令来修改链接的列表。
- 在 Linux 内核中，广泛使用了每处理器变量，系统上的每个处理器都获得了自己的一个给定变量的副本。访问每处理器变量不需要使用锁，此外，因为在不同的处理器上，这些变量未在线程之间共享，因此没有伪共享或内存争用。这种技巧非常适合收集统计信息。

**减少**

- 当使用传统锁或同步技巧（如自旋锁）时，必须注意的是，不要使用单片锁或全局锁，而是将这些锁分成更细小的部分。因此，锁会保护数据结构中的某个特定区域以及较小的区域。这样多个线程就能够通过获取保护这些成员的相应锁，在同一数据结构的不同成员上并发进行操作。这种方法可以实现更多并发。
- 甚至当软件设计中的同步机制能够实现更好的并发和减少锁争用时，也可能会由于伪共享而导致发生性能问题。例如，考虑一个哈希数据结构。如果存在一个自旋锁数组，用于保护哈希中的每个哈希桶，那么在自旋锁数组中可能会出现伪共享。两个线程在两个不同的处理器上运行，每个线程都锁定哈希中的不同哈希桶，那么当它们所需的自旋锁位于同一个缓存行上时，可能会发生伪共享。因此，在设计此类算法时需要考虑采用避免发生伪共享的通用技巧。

**原文链接：http://www.infoq.com/cn/news/2013/08/code-parallel**

**相关内容**

- Vector Fabrics推出多核软件优化工具Pareon
- SaaS平台上的多线程控制与故障处理
- 为不规则应用设计新一代超大型多线程架构
- 深入理解Java内存模型
- 深入理解Java内存模型（七）——总结

# 观点 | View

## 《天空之城》助Twitter刷新纪录，新架构功不可没

作者 丁雪丰

本月，宫崎骏大师的《天空之城》在NTV迎来其第14次电视重播，剧情发展到高潮之时，“Blase祭”也将Twitter的TPS（Twitters per second）推上了新的高度——143,199 TPS，Twitter一般每天会发出5亿多条微博，平均5700 TPS，新纪录是平均值的25倍。

Twitter的“大鲸鱼”曾为人津津乐道，每次Twitter出现故障都会挂出大鲸鱼，但细心的朋友一定已经注意到“大鲸鱼”的出镜率越来越低了。自2010年世界杯后，Twitter便开始了大刀阔斧的架构重构，如今的Twitter早已不再是那个全球最大的基于Ruby on Rails的Web站点了。Raffi Krikorian在Twitter的官方博客中介绍了新架构是如何应对14.3万TPS这一高峰的。

在开始重构前，他们为自己定下了三个目标：

- 新架构能在性能、效率、可靠性上有所突破，即改善用户体验到的延时，减少10倍的服务器数量，在故障面前，基础设施能做到故障隔离。
- 实现一个松耦合的面向服务的模型，鼓励系统级的封装和模块化。
- 能更快地让新功能发布到线上。

Twitter曾经是Rails的“金字招牌”，发展初期大量使用了Rails，但后来Twitter从Ruby转向了Scala，一度让人们认为Ruby的性能问题阻碍了Twitter的发展，而Linkedin和Iron.io从Ruby转向其他语言，也加重了大家对Ruby性能的顾虑。**其实，Ruby并非幕后真凶，Rails才是！**为了提供一站式的Web建站设施，Rails默认提供了太多的功能，正如范凯在其博客中说的那样：

> Rails适合开发Website，但不太适合Web Service，而移动时代的发展趋势就是：未来服务器端会更多的使用Web Service而不是Website，这也意味着Rails将越来越不适合时代的发展。

虽然Ruby有点冤，但是Twitter却在从Ruby到JVM的转型中实实在在地得到了好处。Twitter在JVM上实现了它的搜索引擎、流API和社交图谱，这让其服务器的吞吐量从每秒处理200到300个请求提升到了10000到20000个请求，带来了10倍以上的性能提升。

SOA化也是Twitter重构中的重点，从一个庞大的Ruby应用拆分为多个相对独立、边界清晰的小系统，专注于各自的服务，服务之间通过Finagle实现RPC调用。很多知名的互联网站点都有过类似的经历，比如大家所熟知的支付宝和阿里巴巴，支付宝的首席架构师程立就曾在InfoQ上分享过自己在大规模SOA系统上的经验；在RPC框架方面，阿里巴巴也有自己的Dubbo框架。

存储上，Twitter对原先的MySQL主从结构做了调整，对数据库进行了拆分，在此期间，他们还开发了一系列的框架，比如Gizzard和Snowflake。淘宝也同样对数据库进行了拆分，大量地分库分表，并且开源了自己的TDDL框架。**由此可见，国内外的各大网站当发展到一定规模之后，必然会采取一些类似的措施来保证自己能够进一步发展下去，正所谓“英雄所见略同”。**

博文中还提到了监控与开关相关的内容，例如与Finagle整合在一起的Viz和Zipkin，能方便地对每个请求进行监控。而他们的Decider系统集成在Twitter的所有服务里，这个系统类似一个开关，能够快速地对功能进行切换，甚至能让某个功能仅为指定百分比的用户提供服务，如此一来，可控的灰度发布就能成为可能。说到功能开关，乔梁曾在介绍百度的持续交付经验时也提到过类似的概念，不过他说的仅是控制特定功能的打开与关闭，显然Twitter的Decider更胜一筹。

除了这次的博文，Raffi Krikorian去年在QConSF上还专门就Twitter的时间线优化做过分享，而著名的HighScalability.com上对Twitter的新架构也有介绍，如果您对Twitter的架构感兴趣，不妨再做进一步的了解，然后将您的体会分享给大家。

**原文链接：http://www.infoq.com/cn/news/2013/08/twitter-new-tps**

## 相关内容

- 扩展Twitter以支撑新负载峰值
- Martin Odersky谈Scala.Net及Scala语言
- 大家谈18岁的Java——李刚：Java需要引入更多的新的探索
- 大家谈18岁的Java——朱鸿：开过跑车后再去开大巴车总是有点不爽的
- Azul Systems发布面向WebSphere应用服务器的Zing平台版

# 专题推荐语

## NoSQL一览

NoSQL再也不像几年前那样成了一个炒作的话题，我们发现大家也越来越理性的对待数据库的方案，按照场景来做 区分与选型。理性的回归一方面代表了我们对于技术的普遍接受，二来是我们遇到了不少的坑让我们跌倒了以后能够在去审视走过的路。 在这一期的架构师中，我们选择了NoSQL的专题从一些基本的数据库的应用场景出发，把这些大家踩过的坑填平，也许对你而言可以再去审视 自己曾经走过的路吧！

# 本期专题：NoSQL一览 | Topic

## 书摘和采访：图数据库

**作者 Srini Penchikala ，译者 侯伯薇**

《图数据库》的作者是Ian Robinson、Jim Webber和Emil Eifrém，讲述了基于图的NoSQL数据库技术，以及在现实世界的应用程序中可以用来存储“相关联的数据”的不同方法。作者们讨论了很多主题，像使用基于图的领域模型来进行数据建模，以及使用图理论的技术来进行预测性分析等。

InfoQ采访了书的合作者Ian Robinson和Jim Webber，谈到了这本书、图数据库在NoSQL数据库领域的角色，以及图数据库的将来等问题。

**InfoQ：恭喜你们写出了新书。这本书当前进展如何？**

> **Ian and Jim：**多谢Srini！ 这本书是在去年撰写的，从粗剪版到最后花费了大概八个月时间。我们现在和O'Reilly正在进行最后的编辑阶段，所有工作会在六月十日完成。我们会向今年GraphConnect会议以及很多Neo4j社区活动的参与者发放。

**InfoQ：和关系型数据库以及其他NoSQL数据库相比，图数据库表现如何？**

> **Ian and Jim：**Alistair Jones创造了一个术语“关系型十字路口（relational crossroads）”来描述它们之间的区别。据Alistair所说，我们正处在分岔路口。一条路通往大多数NoSQL数据库所采用的方法，其中数据被严重非范式化，然后我们依赖于应用程序来把它们组合在一起，以得到进一步的见解，那通常需要很高的延迟。另一条路通往图数据库所采用的方法，其中我们使用图的强大表现能力来为手头的问题构建灵活的、相互连接的模型，然后可以用低延迟的方式查询以获得更深理解。
>
> 关系型数据库处于二者之间。和图数据库更相似的是，关系型数据库拥有以查询为中心的模型。但这个模型并没有图数据库那样强大。特别是，它缺乏在运行时创建任意广泛、语义上丰富并且连接可变的结构的能力。为了在关系型数据库中创建任何一种可扩展的结构，我们都需要预先对我们的连接做出计划。为了使其可变，我们最终会创建大量可空的列。结果是：稀疏的表，昂贵的连

接，以及对象-关系阻抗（object-relational impedance）问题，即便是在非常简单的应用中使用也是一样。

**InfoQ：图数据库最适用于什么样的用例？**

> **Ian and Jim：**和关系型数据库一样，图数据库主要针对于OLTP数据库，并可以应用在很广泛的解决方案中。也就是说，当问题的解决方案依赖于我们首先理解事物是如何连接的时候，它们会非常有效。这要比你所想象的更加常见。在很多情况下，不仅关系本身很重要，同时关系的名称、紧密程度以及关系权重也非常重要。
>
> 简言之，连接是关键所在。对于为连接建模和对其进行查询来说，图是最佳的抽象形式；因此，图数据库让应用程序开发者和数据专家可以把这种抽象应用到他们自己的特定问题中。直到最后，我们看到它们用于社交网络、推荐引擎、授权和访问控制、寻径和后勤、产品分类、数据中心管理、职业管理、诈骗检测、制定政策、地理空间甚至是移民的应用程序。把所有这些解决方案绑定在一起的关键点在于，它们依赖于能够对相关联的数据建模和查询。仅仅拥有键和值是不够的；拥有通过在语义上改善后的连接关联在一起的数据也不够。我们既需要连接，也需要丰富的上下文，才能够让这些解决方案生效。

**InfoQ：你可否告诉大家，当使用图数据库的时候，开发者需要记在头脑中的关于设计的考虑有哪些？**

> **Ian and Jim**：对于任意的特定应用来说，最重要的设计决定都会关注数据模型和查询。正如我们在书中所描述的，此处的关键是让数据提出的、你需要回答的问题来驱动你的模型。你可以发现应用程序目的的核心或者最终用户的需求，从而确定你感兴趣的内容，以及它们是如何连接的。接下来是一个简短的步骤，我们要把这些问题转换成可表达的图模型，然后转换到你要对模型执行的查询之中。
>
> 得到的模型和相关联的查询都是你想要对数据要提出的问题的投影。使用Neo4j的Cypher查询语言，这些投影的互补特性会变得非常明显：你用来创建图结构的路径和你用来查询的路径是相同的。
>
> 作为对你设计的很好的第一次测试，你应该可以读懂自己编写的内容。最重要的是，你应该能够读懂你写入的问题，而不需要做出任何额外的假设和推断。像“(Emil)-[:WROTE]->(Graph Databases)<-[:PUBLISHED]-(O’Reilly)”这样的结构很易读，并清晰地帮助我们回答了以下问题：“Emil写了那些书？”，“谁出版了《图数据库》？”以及“Emil为哪家出版社写了书？”。

**InfoQ：图数据库支持哪些架构和设计模式？**

> **Ian and Jim：**这个问题和图数据库并没有太大关系，它们只是数据库，只是对于连接的数据，它的执行速度更快一些。所以任意你想要采用的应用程序模式都会有效。你喜欢MVC？ 当然，那很适合。你想要完全工作在带有回调的JavaScript中？ 没问题，对Neo4j有很棒的Node.js连接程序。

**InfoQ：图数据库本身在可伸缩性方面有什么限制吗？**

> **Ian and Jim：**当我们谈到伸缩的时候，实际上是在谈三种不同的东西：为大型数据集伸缩、为读取性能伸缩以及为写入性能伸缩。
>
> 关于为大型数据集伸缩 ，没有任何固有的限制。Neo4j当前对图的大小方面有个固定上限（大概是10的10次方，这对于支持我们在现实世界中能够看到的大多数图都足够，包括一个Neo4j部署，它在一个Neo4j簇集中拥有超过一半的Facebook社交图），但这项限制会在本年稍后取消。这会解除人们的顾虑，可以放心地为“大”数据对图进行缩放。
>
> 为读取伸缩同样没有表现出任何问题。当前，在Neo4j中，这是通过主从复制完成的。读取请求会在簇集间做负载平衡，其中它们会针对本地数据执行，它的结构已经针对连接查询做过优化。除了事务透明性之外，Neo4j在历史上一直关注读取的性能。为了支持快速读取，我们拥有本地图的栈，一直到磁盘中。某些其他图存储通过非本地的图存储——像列存储——提供了图接口。这可能对其他缩放特征有所帮助，但它很可能会提高查询的延迟，因为连接是在库代码中执行的，而不是在数据模型的层次执行的。
>
> 为写入伸缩可以通过垂直缩放达到，但在有些时候，对于非常严重的写入负载，就需要跨多台计算机分发数据的能力。这真的是一种挑战。尽管分发数据会有助于提高写入性能，但也会对读取性能造成不利的影响。迄今为止，还没有人实现出一种图数据库，可以优化，并解决本地遍历速度快，而分布式遍历（通过网络）速度慢的问题。
>
> 通过跨多台计算机分发图数据来对其进行伸缩，要比伸缩一些简单的数据模型困难得多，但还是可以实现的。伸缩键-值、列族或者文档存储相对容易，因为在那些情况下，你处理的是离散的聚合，可以完全放在某个位置。缩放一个图要更困难，因为它的数据天生就是相互连接的。当分发一个图的时候，我们想要尽可能避免出现跨计算机的关系；这叫做最小点切问题（minimum point-cut problem）。在平衡图从而有尽可能少的跨计算机的关系的问题之上，情

况会变得更复杂，因为图总是在变化。一个时间点看起来像是很好的分发的情况，可能几秒钟之后就不再是优化的方案了。这一般叫做NP困难问题。

尽管我们无法声称已经解决了NP困难的一般图分区问题，但我们已经找到一些不错的方法来回避它。事实上，我们现在已经让R&D团队忙于试验这些想法，在将来的某个时刻，他们会加入到未来的产品版本中。另外，我们还有一些现成的技巧，可以使用现有的技术来提高写入伸缩限制。

所以，为了回答这个问题，我们会说图中数据相互连接的本质为通过分发图来伸缩写入带来了理论上的挑战。但对于分布式图问题还有一些实践性的解决方案，那也正是我们所研究的。

**InfoQ：图数据库是要存储和管理相互连接的数据。这项特征限制了这些数据库在支持方面——像Sharding和事务——的能力吗？**

**Ian and Jim：**正如我们在上一个回答中所提到的，sharding（或者图分区）是让图可伸缩的关键，特别是对于写入操作。但那很困难，至少在一般的情况下很难。另外，我们还需要使用事务来保护数据的完整性，你会看到我们还有一项有趣的可伸缩性设计挑战。

也就是说，当处理很多并发事务的时候，图的数据结构的特性有助于跨多个图分布事务负载。当图增长的时候，事务争用通常会减少。换句话说，图越大，就越通畅，那非常不错！

**InfoQ：图数据库如何支持数据复制和灾难恢复特性？**

**Ian and Jim：**现在Neo4j使用主从簇集的方案来支持复制和灾难恢复。在Neo4j簇集中，一台计算机会被指定为主，另一台被指定为从。指向到任意服务器的写入操作最终都会传递给主计算机，然后把更新的结果传递给从计算机。

如果主计算机出现了故障，那么簇集会自动通过我们的Paxos实现选取出新的主计算机。再提一下，我们的R&D团队已经做了一些有意义的工作，让Neo4j成为没有主计算机的数据库簇集，那会在将来的版本中发布。

**InfoQ：可否请你说一下图数据库的一些限制？**

**Ian and Jim：**和Apache Cassandra之类产品相比，今天的图数据库还没有同

> 等程度的写入吞吐量——这是由于主从簇集以及合适的ACID事务所决定的。把图数据库用于购物车或者投票系统不是太合理，还有更多合适的数据库可以使用。但是当你为购物车使用了列数据库或者键值（Key-value）数据库，在图数据库中分析客户的购买历史会非常好。这是我们已经见过几次的模式：使用简单的数据库来吸收负载，然后把数据输入到图数据库，作为记录系统来进行提炼和分析。

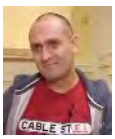
**InfoQ：在图数据库领域有什么样的趋势？ 你认为图数据库的将来会怎样？**

> **Ian and Jim：**从高层次上看，我认为图数据库正在成为主流。五年前，图数据库还在学院中，也无人关注，但现在它们已经无处不在。不久，图数据库会是我们考虑的另一种一般工具。基于如何看待它们今天的应用，我们相信它们会继续取代关系型数据库。随着数据之间的连接变得更杂乱，随着这个领域的成熟，它们的流行度也会猛增。

他们还提到，这是图和图数据库的激动时刻。它们看到更多图工具——既有专门目的，也有一般目的的——进入到市场中。Neo4j正在大幅改进它的数据模型、它的查询语言、对非常大型的数据集的支持以及整体性能。文化也在不断增长，已经有了在线的教程和用例。已经有了一个健康、活跃的用户社区，在世界各地的城市中集会。并且，还有GraphConnect会议，今年会在旧金山、芝加哥、波士顿、纽约和伦敦举办，每个城市都会有技术、理论和实践的话题，会有各个业界和社区的专家参加。

# 关于受访者

**Ian Robinson**是Neo科技的一位工程师，目前正在从事Neo4j图数据库未来版本的研究和开发工作。他是《REST in Practice (O'Reilly)》的合著者，并且是《REST: From Research to Practice”(Springer)》和《Service Design Patterns (Addison-Wesley)》的贡献者。



**Dr. Jim Webber**是Neo科技的首席科学家，该公司支持了流行的开源图数据库Neo4j，他在其中研究和开发分布式图数据库，并编写开源的软件。他是《REST in Practice》一书的合著者。Jim的博客位于http://jimwebber.org，他的推特是@jimwebber。

# 本期专题：NoSQL一览 | Topic

# MetaModel——跨多种数据存储提供统一的数据访问

作者 Kasper Sørensen ，译者 夏雪

最近，Human Inference和Apache软件基金会（ASF）分别宣布了捐赠与接受MetaModel项目的消息，今后MetaModel将成为一个Apache孵化器项目。此前，MetaModel遵循LGPL许可证，由Human Inference的产品开发团队管理，但现在已经转移到了ASF上，将遵循新的许可证、拥有新的社区、接受新的管理。那么这个项目到底是什么呢？它又有哪些用途？

MetaModel是一个Java类库，设计它的目的是提供一个可以与任何数据存储（不论是关系型数据库、NoSQL数据库，还是电子表格或者其他格式的文件）进行交互的单一接口。我们的意思是，通过交互来搜索元数据，查询、写入或更改数据存储中保存的数据。很显然，高层次抽象会丢失一些细节，会带来过度概括和丢失重要特性的风险。我们不想把关系型SQL数据库的功能减化到只有像（SELECT * FROM [table]）这样的全表扫描。但另一方面，我们也不想暴露一些只能在特定品牌特定版本的SQL服务器上才能使用的功能，因为这些功能无法在任何其他数据存储上使用。最终，我们想基于现有的常规技能（比如标准SQL）建立数据交互方式。

## 元数据处理

那么，MetaModel项目为数据存储的方法做了怎样的抽象？该项目通过Java接口发布了一个非常类似于SQL的查询模型，有时该模型也可从字符中解析。由于把查询定义为常规的Java对象，就能容易地解析查询（取决于底层技术），并能根据底层技术细节选择最佳的执行策略。这意味着MetaModel不仅提供了一个接口，还提供了一个完整的查询引擎，这个引擎在查询时可以处理部分或者全部任务。如果你使用的是关系型JDBC数据库，会在数据库的本地引擎上执行99%的查询。但是如果使用了MetaModel，就可以利用它的查询引擎把数据切片、切块，就能在CSV文件或Excel电子表格上执行同样的查询了。同时，你根本不必修改查询语句。

当然，这得假设你数据存储的元数据与数据结构是兼容的。不同的数据存储以

不同的方式暴露或推理元数据。JDBC数据库通常会通过JDBC元数据API暴露元数据。文件格式（比如CSV和Excel表格）并没有很明确地定义，通常可以读取文件的首行内容获取它们的元数据。还有种比较极端的情况，有一些NoSQL数据库甚至根本就没有元数据。MetaModel可以让你选择暴露元数据的方式，你可以指定是以编程方式暴露，或通过检查数据存储的前N条记录的方式来推理出元数据。

MetaModel中最核心的设计是DataContext接口，它表示数据存储，可以用它来浏览和查询该数据存储。此外，它的子接口UpdateableDataContext表示可写的数据存储，可以用它更新数据。一旦你拥有了DataContext实例，就可以或多或少地学习使用类库代码补全的功能了。这里有几个DataContext实现的典型示例，让我们来看看它们是如何实例化的：

```
// a DataContext for a CSV file
UpdateableDataContextcsv = new CsvDataContext(new File
(“data.csv”));

        // a DataContext for an Excel spreadsheet
UpdateableDataContext excel = new ExcelDataContext(new
 File(“spreadsheet.xlsx”));

        // a DataContext for a JDBC database (can use either D
ataSource or Connection)
java.sql.DataSourcedataSource = …
UpdateableDataContextjdbc = new JdbcDataContext(dataSou
rce);

        // a DataContext for an XML file (where metadata is aut
omatically inferred)
DataContext xml = new XmlDomDataContext(new File(“data
.xml”));

        // a DataContext for connecting to Salesforce.com’s d
ata web services
UpdateableDataContextsalesforce =
newSalesforceDataContext(username, pw, securityToken);
```

```
        // a in-memory DataContext for POJOs (useful for testi
ng and mocking)
          Person record1 = ...
          Person record2 = ...
TableDataProvidertableDataProvider = new ObjectTableDat
aProvider(
            "persons", Person.class, Arrays.asList(reco
rd1, record2));
UpdateableDataContextpojos = new PojoDataContext("schem
a", tableDataProvider);
```

对于MetaModel来说元数据极为重要，它不仅用来管理数据结构，而且还用来定义查询。如果你的查询要确保使用了适当的元数据，就需要投入大量的精力保证查询的安全性。因此，作为一名开发人员，你在查询之前就要持有元数据对象。举例来说，如果你知道有一张表，表名是ORDER_LINE，它有一列是price，另一列是order_id，那么就可以用寻常的硬编码方式去查询所需的元数据（显然，只有你熟悉数据存储时才能使用这种方式）：

```
DataContextdataContext = ... // the DataContext object repr
esents the  'connection'
          Table orderLines = dataContext.getTableByQuali
fiedLabel("ORDER_LINES");
          Column price = orderLines.getColumnByName("pri
ce");
          Column orderId = orderLines.getColumnByName("o
rder_id");
```

此外，你还可以使用API基于探索的方式动态获取元数据。当你想为用户提供定制查询时，这种方式就能发挥巨大的作用了，你可以用这种方式为用户提供可用的表和列，让用户自己选择所需的查询。

```
Schema[] schema = dataContext.getSchemas();
Table[] tables = schemas[0].getTables();
Column[] columns = tables[0].getColumns();
```

MetaModel还一个非常重要的思想，它把元数据、查询和其他数据交互都当作是对象。查询在MetaModel中就是一个普通的Java对象，所以你可以在执行之前篡改它，也可以把它分发出去。这使应用程序可以创建由不同代码块协作完成的复合工作流程，在优化查询计划时就不必再面对冗长的SQL字符串操作了。它对类型安全也很有帮助，举例来说，查询模型是基于像列、表等类型安全的结构来描述模型的，而不是采用模糊不清的字符串常量。

# 数据存储的查询

那么，让我们来看看MetaModel是如何查询数据存储的。

你可以用三种不同的方式触发同一个查询：

## 1. 手写所有代码:

手写所有代码是POJO的传统方式。用这种方式要写非常冗长的代码，但你能得到最高的灵活性。

```
Query q = new Query();
q.select(SUM, price);
q.select(orderId);
q.from(orderLines);
q.groupBy(orderId);
q.setMaxRows(100);
DataSetdataSet = dataContext.executeQuery(q);
```

## 2. 使用优雅的Builder API:

使用Builder API是另外一种类型安全的查询方式，用这种方式只需要编写较短的代码。另外，这个API运用了建造者模式builder-pattern），使开发人员可以迭代调用API，一步一步补充子查询逻辑。如果你只想定义一个组件的查询，首先就应考虑这种方式。

```
Query q = dataContext.query().from(orderLines)
                             .select(SUM, price).and(orderId)
```

```
                    .groupBy(orderId).maxRows(100).toQuery();
DataSetdataSet = dataContext.executeQuery(q);
```

### 3. 通过字符串解析：

有时可能你只想快捷简单地完成工作，那么就可以使用编写SQL语句这种更加传统的方式了。MetaModel也能把普通字符串解析成查询，但这种方式只能在运行期验证查询语句，所以存在类型安全的风险。

```
Query q = dataContext.parseQuery(
                    "SELECT SUM(price), order_id FROM order_line
s GROUP BY order_id LIMIT 100" );
DataSetdataSet = dataContext.executeQuery(q);
```

很明显可以看出，这三种方式最终的查询结果都是一个DataSet类型的对象，这个对象把查询结果描述成一个表格。DataSet并没有太多复杂的特性，遍历它的方法非常简单，如下所示：

```
Try {
   while (dataSet.next()) {
    Row row = dataSet.getRow();
System.out.println(row.toString());
   }
} finally {
dataset.close();
}
```

## 数据存储的更新

MetaModel使用类型安全、元数据驱动的方法执行更新。如前文所述，并非所有的数据存储都是可写的，所以若要写入数据，你不仅仅需要一个DataContext对象，还需要实现UpdateableDataContext接口。假设，我们想更新*order*表的数据，代码示例如下：

```
dataContext.executeUpdate(new UpdateScript() {
        @Override
        public void run(UpdateCallbackcb) {
          // insert a new order line
cb.insertInto(orderLines).value(orderId, 123).value(price, 395).e
xecute();

                    //  update the price of orderlines where or
der_id = 122
cb.update(orderLines).where(orderId).eq(122).value(price, 295)
.execute();
              }
        });
```

这里要注意UpdateScript的代码结构，更新操作是放在逻辑事务里的。MetaModel会根据不同的底层数据技术选用适当的事务策略，比如JDBC数据库就会采用ACID事务，对于大多数文件格式的存储就会采用同步写入的策略，不一而足。最终使你只需要用单一的语法就能把数据写入到所有数据存储中。

因为示例代码使用了匿名内部类，所以看起来并不是特别优雅。但Java 8肯定会借助闭包的概念改进这一点。此外，如果你只希望执行一个单独的操作，也可以使用几个预制的UpdateScript类，它们简单易用，极易上手：

```
dataContext.executeUpdate(
        new InsertInto(orderLines).value(orderId, 123).value(pri
ce, 395));
```

而且，你还可以使用executeUpdate方法创建和删除表，也能用它删除表记录。

## 对新的数据存储支持

最后，MetaModel的专家级用户可能会问，*“如果我想连接[XYZ]呢？”*（XYZ是我们暂不支持的其他数据存储）。很显然，我们希望MetaModel易于扩展，出于这个原因和其他方面的考虑，我们设计了可插拔的查询引擎。你只需要自己实现DataContext接口，但是如果你从头开始实现的话，也不是很简单。所以，我们为你提供了一个抽象类实现，它包括了几个扩

展点。以下是全部过程：

- 先让你的类扩展抽象类*QueryPostprocessDataContext*。然后你会发现几个需要实现的抽象方法：
  - getMainSchema()
    通过实现这个方法，暴露你DataContext的模式模型。
  - materializeMainSchemaTable(Table, Column[], int)
    通过实现这个方法，为一张特定的表提供一个相当于全表扫描的实现。
- 现在你的DataContext就可以使用了，你可以在外来数据存储中开始使用MetaModel了！
- 但是，我们还要继续优化！虽然我们的DataContext现在完全可以使用了，但是可能它还无法执行某些查询，因为MetaModel查询引擎依赖于materializeMainSchemaTable（...）方法，会把该方法的执行结果作为数据源来处理几乎所有的查询。所以，你可能还需要重写以下几个方法：
  - executeCountQuery（...）
    很多数据存储都提供了这么一个统计指定表总记录数的简易方法。由于经常会用到这种查询，所以很有必要重写这个方法。
  - materializeMainSchemaTable(Table, Column[], int, int)
    大家很多时候都会采用*分页*查询的方式获取表数据（从第X条记录到第Y条记录）。为了满足这种查询需求，就需要在这个方法中追加int类型的参数，让使用者可以通过这个参数按页查询，而不必总是查出整张表的所有记录。
  - executeQuery(Query)
    如果你想要进一步优化，支持WHERE或GROUP BY子句，就需要重写这个方法。为了满足各种复杂的情况，MetaModel把这个方法的参数（Query）设计为复杂对象类型。在SalesforceDataContext和MongoDbDataContext两个类的源代码中可以找到一些很好的参考示例。

# 结束语

在这篇文章中，我们介绍了MetaModel（它是一个类库，让我们可以用它访问各种不同的数据存储），解释了如何用它处理元数据，如何用它询问存储，以及如何用它更新数据。

即使MetaModel搬了新家（Apache），我们将来还会继续完善它。我们将会

针对HBase、Cassandra和其他流行的数据库增加更多内置的DataContext实现。并且进一步扩展特性，让你可以利用元数据完成更多的工作。除了这些，我们还有一些其他的想法，比如我们正在努力丰富元数据嵌套结构（比如许多NoSQL数据库都会用到的地图和列表）、创建虚拟表（类似于视图，但要建在客户端上，而不是服务端上）的能力，支持把POJO映射为DataSet行，为查询引擎添加更多的功能。

如果您对这个项目感兴趣，可以在Apache Incubator MetaModel页面中查阅到邮件列表、bug跟踪等相关内容。

# 作者简介

**Kasper Sørensen**是Human Inference的首席软件工程师和产品经理。他的专业特长和兴趣爱好是开发数据密集型应用产品。在哥本哈根商学院毕业时，他创办了DataCleaner和MetaModel这两个开源项目，他硕士论文的部分内容就是关于这两个项目的。你可以在他的博客Kasper’ s Source中了解更多内容。

**原文链接：http://www.infoq.com/cn/articles/apache-incubator-metamodel**

**相关内容**

- 使用Power Map Preview for Excel 2013创建和分享Bings 3D地图
- Hadoop + SQL Server + Excel = 大数据分析
- Phoenix：在Apache HBase上执行SQL查询
- Apache Isis：领域驱动设计的Java框架
- SpringOne专访：Chris Richardson谈云时代的Spring发展

# 本期专题：NoSQL一览 | Topic

# 深入浅出Symfony2 - 结合MongoDB开发LBS应用

作者 周攀，洪涛

## 简介

随着近几年各类移动终端的迅速普及，基于地理位置的服务（LBS）和相关应用也越来越多，而支撑这些应用的最基础技术之一，就是基于地理位置信息的处理。我所在的项目也正从事相关系统的开发，我们使用的是Symfony2+Doctrine2 ODM+MongoDB的组合。

我们将这些技术要点整理成文，希望能够通过本文的介绍和案例，详细解释如何使用MongoDB进行地理位置信息的查询和处理。在文章的开头，我们也会先介绍一下业界通常用来处理地理位置信息的一些方案并进行比较，让读者逐步了解使用MongoDB查询及处理地理位置信息的优势。

本文使用了Symfony2和Doctrine2作为Web应用的开发框架，对于想了解Symfony2的数据库操作的读者来说阅读本文也可以了解和掌握相关的技术和使用方法。

## 1. LBS类应用特点

不管是什么LBS应用，一个共同的特点就是：他们的数据都或多或少包含了地理位置信息。而如何对这些信息进行查询、处理、分析，也就成为了支撑LBS应用的最基础也是最关键的技术问题。

而由于地理位置信息的特殊性，在开发中经常会有比较难以处理的问题出现，比如：由于用户所在位置的不固定性，用户可能会在很小范围内移动，而此时经纬度值也会随之变化；甚至在同一个位置，通过GPS设备获取到的位置信息也可能不一样。所以如果通过经纬度去获取周边信息时，就很难像传统数据库那样做查询并进行缓存。

对于这个问题，有读者可能会说有别的处理方案，没错，比如只按经纬度固定

的几位小数点做索引，比如按矩阵将用户划分到某固定小范围的区域（可以参考后文将会提到的geohash）等方式，虽然可以绕个弯子解决，但或多或少操作起来比较麻烦，也会牺牲一些精度，甚至无法做到性能的最优化，所以不能算作是最佳的解决办法。

而最近几年，直接支持地理位置操作的数据库层出不穷，其操作友好、性能高的特性也开始被我们慢慢重视起来，其中的佼佼者当属MongoDB。

MongoDB在地理位置信息的处理上有什么优势？下面我们通过一个简单的案例来对比一下各种技术方案之间进行进行地理位置信息处理的差异。

# 2. 几个地理位置信息处理方案的对比和分析

## 1. 确定功能需求

对于任何LBS应用来说，让用户寻找周围的好友可能都是一个必不可少的功能，我们就以这个功能为例，来看看各种处理方案之间的差异和区别。

我们假设有如下功能需求：

- 显示我附近的人
- 由近到远排序
- 显示距离

## 2. 可能的技术方案

排除一些不通用和难以实现的技术，我们罗列出以下几种方案：

1. 基于MySQL数据库
2. 采用GeoHash索引，基于MySQL
3. MySQL空间存储（MySQL Spatial Extensions）
4. 使用MongoDB存储地理位置信息

我们一个个来分析这几种方案。

**方案1：基于MySQL数据库**

MySQL的使用非常简单。对于大部分已经使用MySQL的网站来说，使用这种方案没有任何迁移和部署成本。

而在MySQL中查询“最近的人”也仅需一条SQL即可,

```
SELECT id, ( 6371 * acos( cos( radians(37) ) * cos( radians( lat ) )
* cos( radians
( lng ) - radians(-122) ) + sin( radians(37) ) * sin( radians( lat ) ) ) )
 AS distance
 FROM places HAVING distance < 25 ORDER BY distance LIMI
T 0 , 100;
```

注：这条SQL查询的是在lat,lng这个坐标附近的目标，并且按距离正序排列，SQL中的distance单位为公里。

但使用SQL语句进行查询的缺点也显而易见，每条SQL的计算量都会非常大，性能将会是严重的问题。

先别放弃，我们尝试对这条SQL做一些优化。

可以将圆形区域抽象为正方形，如下图

根据维基百科上的球面计算公式，可以根据圆心坐标计算出正方形四个点的坐标。

然后，查询这个正方形内的目标点。

SQL语句可以简化如下：

```
SELECT * FROM places WHERE ((lat BETWEEN ? AND ?) AND (
```

```
lng BETWEEN ? AND ?))
```

这样优化后，虽然数据不完全精确，但性能提升很明显，并且可以通过给lat lng字段做索引的方式进一步加快这条SQL的查询速度。对精度有要求的应用也可以在这个结果上再进行计算，排除那些在方块范围内但不在原型范围内的数据，已达到对精度的要求。

可是这样查询出来的结果，是没有排序的，除非再进行一些SQL计算。但那又会在查询的过程中产生临时表排序，可能会造成性能问题。

### 方案2：GeoHash索引，基于MySQL

GeoHash是一种地址编码，通过切分地图区域为小方块（切分次数越多，精度越高），它能把二维的经纬度编码成一维的字符串。也就是说，理论上geohash字符串表示的并不是一个点，而是一个矩形区域，只要矩形区域足够小，达到所需精度即可。（其实MongoDB的索引也是基于geohash）

如：wtw3ued9m就是目前我所在的位置，降低一些精度，就会是wtw3ued，再降低一些精度，就会是wtw3u。（点击链接查看坐标编码对应Google地图的位置）

所以这样一来，我们就可以在MySQL中用LIKE ‘wtw3u%’来限定区域范围查询目标点，并且可以对结果集做缓存。更不会因为微小的经纬度变化而无法用上数据库的Query Cache。

这种方案的优点显而易见，仅用一个字符串保存经纬度信息，并且精度由字符串从头到尾的长度决定，可以方便索引。

但这种方案的缺点是：从geohash的编码算法中可以看出，靠近每个方块边界两侧的点虽然十分接近，但所属的编码会完全不同。实际应用中，虽然可以通过去搜索环绕当前方块周围的8个方块来解决该问题，但一下子将原来只需要1次SQL查询变成了需要查询9次，这样不仅增大了查询量，也将原本简单的方案复杂化了。

除此之外，这个方案也无法直接得到距离，需要程序协助进行后续的排序计算。

### 方案3：MySQL空间存储

MySQL的空间扩展（MySQL Spatial Extensions），它允许在MySQL中直接处理、保存和分析地理位置相关的信息，看起来这是使用MySQL处理地理位置信息的“官方解决方案”。但恰恰很可惜的是：它却不支持某些最基本的地理

位置操作，比如查询在半径范围内的所有数据。它甚至连两坐标点之间的距离计算方法都没有（MySQL Spatial的distance方法在5.*版本中不支持）

官方指南的做法是这样的：

```
GLength(LineStringFromWKB(LineString(point1, point2)))
```

这条语句的处理逻辑是先通过两个点产生一个LineString的类型的数据，然后调用GLength得到这个LineString的实际长度。

这么做虽然有些复杂，貌似也解决了距离计算的问题，但读者需要注意的是：这种方法计算的是欧式空间的距离，简单来说，它给出的结果是两个点在三维空间中的直线距离，不是飞机在地球上飞的那条轨迹，而是笔直穿过地球的那条直线。

所以如果你的地理位置信息是用经纬度进行存储的，你就无法简单的直接使用这种方式进行距离计算。

### 方案4：使用MongoDB存储地理位置信息

MongoDB原生支持地理位置索引，可以直接用于位置距离计算和查询。

另外，它也是如今最流行的NoSQL数据库之一，除了能够很好地支持地理位置计算之外，还拥有诸如面向集合存储、模式自由、高性能、支持复杂查询、支持完全索引等等特性。

对于我们的需求，在MongoDB只需一个命令即可得到所需要的结果：

```
db.runCommand( { geoNear: "places", near: [ 121.4905, 31.2646 ], num:100 })
```

查询结果默认将会由近到远排序，而且查询结果也包含目标点对象、距离目标点的距离等信息。

由于geoNear是MongoDB原生支持的查询函数，所以性能上也做到了高度的优化，完全可以应付生产环境的压力。

### 方案总结

基于MongoDB做附近查询是很方便的一件事情。

MongoDB在地理位置信息方面的表现远远不限于此，它还支持更多更加方便的功能，如范围查询、距离自动计算等。

接下来，我们结合Symfony2来详细地演示一些使用MongoDB进行地理位置信息处理的例子。

# 3. 结合Symfony2演示

## 运行环境

参考环境：Nginx1.2 + PHP5.4 + MongoDB2.4.3 + Symfony2.1

建立coordinate和places两个document文件，前者是作为places内的一个embed字段. 为方便演示效果，这里同时设置了两个索引 2d 和 2dsphere

```
Document/Coordinate.php

/**
* @MongoDB\EmbeddedDocument
*/
class Coordinate {
    /**
* @MongoDB\Field(type="float")
*/
    public $longitude;

    /**
* @MongoDB\Field(type="float")
*/
    public $latitude;

    ...
}

Document/Place.php
```

```
/**
* @MongoDB\Document(collection="places")
* @MongoDB\ChangeTrackingPolicy("DEFERRED_EXPLICIT")
* @MongoDB\Indexes({
*   @MongoDB\Index(keys={"coordinate"="2d"}),
*   @MongoDB\Index(keys={"coordinate"="2dsphere"})
* })
*/
class Place
{
    /**
    *
    * @MongoDB\Id(strategy="INCREMENT")
    */
    protected $id;

    /**
    * @MongoDB\Field(type="string")
    */
    protected $title;

    /**
    * @MongoDB\Field(type="string")
    */
    protected $address;

    /**
    * @MongoDB\EmbedOne(targetDocument="HenterGEO\
GEOBundle\Document\Coordinate")
    */
    protected $coordinate;

    /**
    * @MongoDB\Distance
    */
```

```
        public $distance;

        ...
    }
```

坐标保存以longitude, latitude这个顺序（没有明确的限制和区别，但我们在此遵循官方的推荐）。

另外，为直观显示查询效果，默认使用百度地图标记查询数据。

### 程序说明

我们用到的代码包是doctrine/mongodb-odm-bundle（下文称ODM），这个代码包提供了在Symfony2环境下的MongoDB数据库支持，使用这个代码包，可以让我们更加方便的在Symfony2环境下操作MongoDB数据库。。

ODM封装了MongoDB中常用的一些地理位置函数，如周边搜索和范围搜索。

ODM中的操作默认距离单位是度，只有geoSphere支持弧度单位（必须在参数中指定spherical(true)）

## 4. MongoDB的地理位置查询

### 注意事项

1. 下文大多数直接对MongoDB的数据库操作将使用Mongo Shell进行演示。在演示网站页面和功能时，将结合Symfony2、Doctrine-MongoDB进行演示。
2. 本文演示所用的MongoDB版本为2.4.3，版本号比较新，所以某些查询方式在低版本里面并不支持。
3. 以places这个collection为例，大部分例子都需要类似下面格式的测试数据支持：{ "_id" : 2, "coordinate" : { "longitude" : 121.3449, "latitude" : 31.17528 }, "title" : "仅售75元，市场价210元的顶呱呱田鸡火锅3-4人套餐，无餐具费，冬日暖锅，欢迎品尝", "address" : "闵行区航新路634号" }

### 地理位置索引：

MongoDB地理位置索引常用的有两种。

- 2d 平面坐标索引，适用于基于平面的坐标计算。也支持球面距离计算，不过官方推荐使用2dsphere索引。
- 2dsphere 几何球体索引，适用于球面几何运算

关于两个坐标之间的距离，官方推荐2dsphere：

> MongoDB supports rudimentary spherical queries on flat 2d indexes for legacy reasons. In general, spherical calculations should use a 2dsphere index, as described in 2dsphere Indexes.

不过，只要坐标跨度不太大（比如几百几千公里），这两个索引计算出的距离相差几乎可以忽略不计。

建立索引：

```
> db.places.ensureIndex({'coordinate':'2d'})
> db.places.ensureIndex({'coordinate':'2dsphere'})
```

## 查询方式：

查询方式分三种情况：

1. Inclusion。范围查询，如百度地图“视野内搜索”。
2. Inetersection。交集查询。不常用。
3. Proximity。周边查询，如“附近500内的餐厅”。

而查询坐标参数则分两种：

1. 坐标对（经纬度）根据查询命令的不同，$maxDistance距离单位可能是 弧度 和 平面单位（经纬度的“度”）：

```
db.<collection>.find( { <location field> :
            { $nearSphere: [ <x> , <y> ] ,
              $maxDistance: <distance in radians>
          } } )
```

2. GeoJson $maxDistance距离单位默认为米：

```
db.<collection>.find( { <location field> :
          { $nearSphere :
           { $geometry :
             { type : "Point" ,
               coordinates : [ <longitude> , <latitude> ] } ,
            $maxDistance : <distance in meters>
      } } } )
```

## 案例A：附近的人

查询当前坐标附近的目标，由近到远排列。

可以通过$near或$nearSphere，这两个方法类似，但**默认**情况下所用到的索引和距离单位不同。

查询方式：

```
> db.places.find({'coordinate':{$near: [121.4905, 31.2646
]}})
> db.places.find({'coordinate':{$nearSphere: [121.4905, 3
1.2646]}})
```

查询结果：

```
{
    "_id" : 115,
    "coordinate" : {
        "longitude" : 121.4915,
        "latitude" : 31.25933
    },
    "title" : "仅售148元，市场价298元的星程上服假日酒店全
日房一间入住一天，
```

```
节假日通用，精致生活，品质享受",
    "address" : "虹口区天水路90号"
}

…(100条)
```

上述查询坐标[121.4905, 31.2646]附近的100个点，从最近到最远排序。

默认返回100条数据，也可以用limit()指定结果数量，如

```
> db.places.find({'coordinate':{$near: [121.4905, 31.2646
]}}).limit(2)
```

指定最大距离 $maxDistance

```
> db.places.find({'coordinate':{$near: [121.4905, 31.2646
], $maxDistance:2}})
```

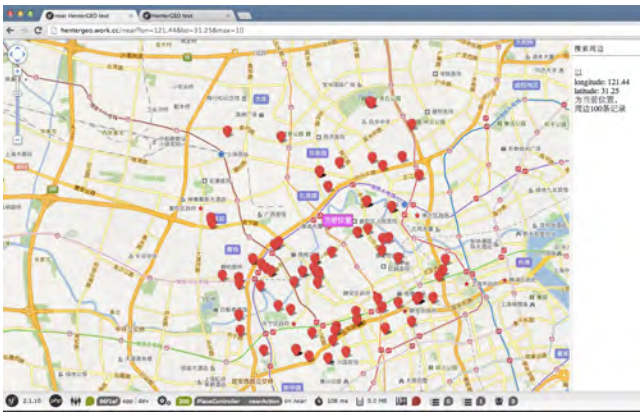
结合Symfony2进行演示：

这里用near，默认以度为单位，公里数除以111（关于该距离单位后文有详细解释）。

```
/**
 * @Route("/near", name="near")
 * @Template()
 */
public function nearAction(){

    $longitude = (float)$this->getRequest()->get('lon',121.
4905);
    $latitude = (float)$this->getRequest()->get('lat',31.264
6);
    //2km
```

```
    $max = (float)$this->getRequest()->get('max', 2);

    $places = $this->getPlaceRepository()->createQueryBui
lder()
        ->field('coordinate')->near($longitude, $latitude)
        ->maxDistance($max/111)
        ->getQuery()->toarray();

    return compact('places','max','longitude','latitude');
}
```

通过 domain.dev/near 访问，效果如下：



longitude: xxx, latitude: xxx为当前位置，我们在地图上显示了周边100条目标记录

## 案例B：区域内搜索

MongoDB中的范围搜索（Inclusion）主要用$geoWithin这个命令，它又细分为3种不同类型，如下：

1. $box 矩形
2. $center 圆（平面），$centerSphere圆（球面）
3. $polygon 多边形

$center和$centerSphere在小范围内的应用几乎没差别（除非这个圆半径几百上千公里）。

下面我们介绍一下这三种查询的案例。

### 矩形区域

这个比较常用，比如百度地图的视野内搜索（矩形）、或搜狗地图的“拉框搜索”

定义一个矩形范围，我们需要指定两个坐标，在MongoDB的查询方式如下：

```
> db.places.find(
    {
     coordinate : {
        $geoWithin : {
           $box :[ [ 121.44, 31.25 ] , [ 121.5005, 31.2846 ] ]
        }
          }
    }
)
```

查询结果：

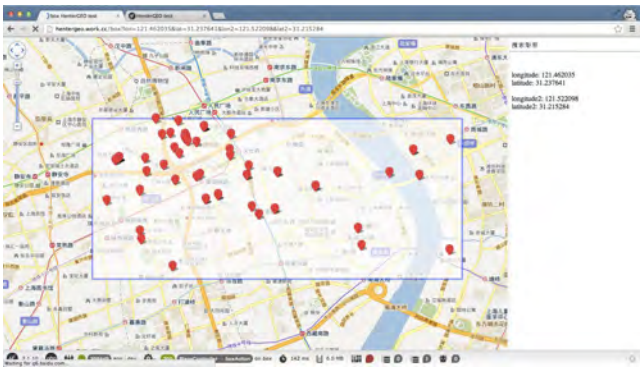
```
{
    "_id" : 90472,
    "title" : "【鲁迅公园】仅售99元！酒店门市价288元的上海
虹元商务宾馆客房一间入住一天（需持本人有效
身份证件办理登记）：大床房/标准房（2选1）！不含早餐！不涉外
！2012年9月29日-10月6日
不可使用拉手券！可延迟退房至14:00！",
    "address" : "上海市虹口区柳营路8号",
    "coordinate" : {
        "longitude" : 121.47,
        "latitude" : 31.27145
    }
}
...
...
```

Symfony2演示代码：

指定两个坐标点

```
/**
 * @Route("/box", name="box")
 * @Template()
 */
public function boxAction(){
    $request = $this->getRequest();

    $longitude = (float)$request->get('lon',121.462035);
    $latitude = (float)$request->get('lat',31.237641);

    $longitude2 = (float)$request->get('lon2',121.522098);
    $latitude2 = (float)$request->get('lat2',31.215284);

    $places = $this->getPlaceRepository()->createQueryBui
lder()
        ->field('coordinate')->withinBox($longitude, $latit
ude,
$longitude2, $latitude2)
        ->getQuery()->toarray();

    return compact('places','longitude','latitude', 'longitude
2', 'latitude2');
}
```

通过 domain.dev/box 访问，效果如下：



**圆形区域**

应用场景有：地图搜索租房信息

查询以某坐标为圆心，指定半径的圆内的数据。

前面已提到，圆形区域搜索分为$center和$centerSphere这两种类型，它们的区别主要在于支持的索引和默认距离单位不同。

2d索引能同时支持$center和$centerSphere，2dsphere索引支持$centerSphere。关于距离单位，$center默认是度，$centerSphere默认距离是弧度。

查询方式如下：

```
> db.places.find({'coordinate':{$geoWithin:{$centerSphere:[
 [121.4905, 31.2646] ,
0.6/111] }}})
或
> db.places.find({'coordinate':{$geoWithin:{$centerSphere:[
 [121.4905, 31.2646] ,
0.6/6371] }}})
查询结果
{
    "_id" : 115,
    "coordinate" : {
        "longitude" : 121.4915,
        "latitude" : 31.25933
    },
    "title" : "仅售148元，市场价298元的星程上服假日酒店全日房
一间入住一天，节假日通用，
精致生活，品质享受",
    "address" : "虹口区天水路90号"
}
...
```

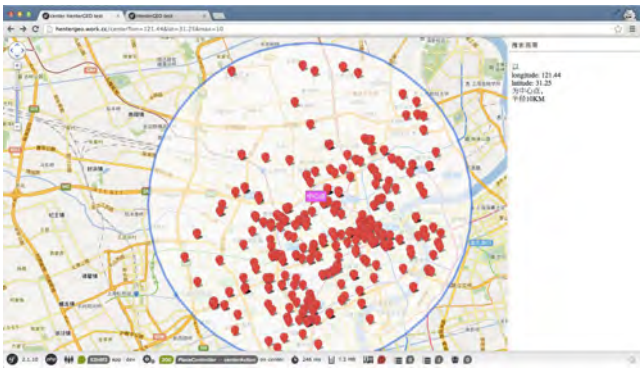
Symfony2演示代码：

指定圆心坐标和半径

```
/**
 * @Route("/center", name="center")
 * @Template()
 */
public function centerAction(){
    $request = $this->getRequest();

    $longitude = (float)$request->get('lon',121.4905);
    $latitude = (float)$request->get('lat',31.2646);
    //10km
    $max = (float)$request->get('max', 10);

    $places = $this->getPlaceRepository()->createQueryBui
lder()
        ->field('coordinate')->withinCenter($longitude, $l
atitude, $max/111)
        ->getQuery()->toarray();

    return compact('places','max','longitude','latitude');
}
```

通过 domain.dev/center 访问，效果如下：

以longitude: xxx，latitude: xxx为中心点，半径10km的圆内



**多边形**

复杂区域内的查询，这个应用场景比较少见。指定至少3个坐标点，查询方式如下（五边形）：

```
> db.places.find( { coordinate : { $geoWithin : { $polygon : [
  [121.45183 , 31.243816] ,
  [121.533181, 31.24344] ,
  [121.535049, 31.208983] ,
  [121.448955, 31.214913] ,
  [121.440619, 31.228748]
]} } } )
```

查询结果

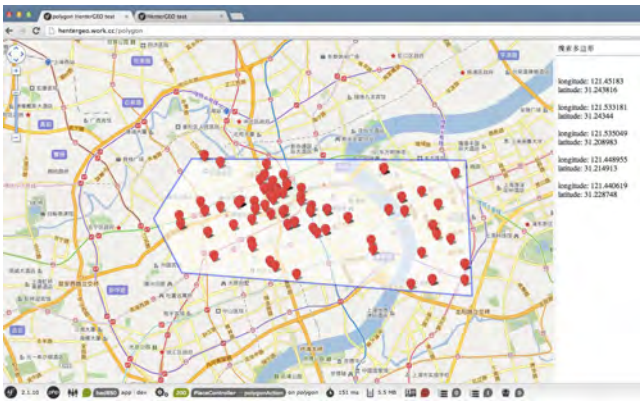
```
{
    "_id" : 90078,
    "title" : "仅售9.9元，市场价38元的燕太太燕窝单人甜品餐，用耐心守候一盅炖品，用爱滋补一生情谊",
    "address" : "河南南路489号香港名都购物广场1F125燕太太燕窝",
    "coordinate" : {
        "longitude" : 121.48912,
        "latitude" : 31.22355
    }
}
...
```

Symfony2演示代码（这里为方便，直接写死了5个坐标点）：

```
/**
 * @Route("/polygon", name="polygon")
 * @Template()
 */
public function polygonAction(){
    $points = [];
    $points[] = [121.45183,31.243816];
    $points[] = [121.533181,31.24344];
```

```
    $points[] = [121.535049,31.208983];
    $points[] = [121.448955,31.214913];
    $points[] = [121.440619,31.228748];

    $sumlon = $sumlat = 0;
    foreach($points as $p){
        $sumlon += $p[0];
        $sumlat += $p[1];
    }
    $center = [$sumlon/count($points), $sumlat/count($po
ints)];

    $places = $this->getPlaceRepository()->createQueryBui
lder()
        ->field('coordinate')->withinPolygon($points[0], $
points[1], $points[2],
$points[3], $points[4])
        ->getQuery()->toarray();

    return compact('places','points', 'center');
}
```

通过 domain.dev/polygon 访问，效果如下：



## 案例C：附近的餐厅

我们假设需要以当前坐标为原点，查询附近指定范围内的餐厅，并直接显示距离。

这个需求用前面提到的$near是可以实现的，但是距离需要二次计算。这里我

们用$geoNear这个命令查询。

$geoNear与$near功能类似，但提供更多功能和返回更多信息，官方文档是这么解释的：

> The geoNear command provides an alternative to the $near operator. In addition to the functionality of $near, geoNear returns additional diagnostic information.

查询方式如下（关于下面的示例用到了distanceMultipler函数，后文会详细解释)：

```
> db.runCommand( { geoNear: "places", near: [ 121.4905, 3
1.2646 ], spherical: true,
 maxDistance:1/6371, num:2 })
{
    "ns" : "mongo_test.places",
    "near" : "1110001100111100001011010110010111001000
110011111101",
    "results" : [
        {
            "dis" : 0.00009318095248858048,
            "obj" : {
                "_id" : 115,
                "coordinate" : {
                    "longitude" : 121.4915,
                    "latitude" : 31.25933
                },
                "title" : "仅售148元，市场价298元的星程上服
假日酒店全日房一间入住一天，
节假日通用，精致生活，品质享受",
                "address" : "虹口区天水路90号"
            }
        },
        {
```

```
                "dis" : 0.00010610660597329082,
                "obj" : {
                    "_id" : 465,
                    "coordinate" : {
                        "longitude" : 121.48406,
                        "latitude" : 31.26202
                    },
                    "title" : "【四川北路】热烈庆祝康骏会馆成立8
周年！仅售169元！市场价399元的
康骏会馆四川北路一店（仅限3星级技师）全身精油按摩一人次！全
程约90分钟！
男女不限！仅限四川北路一店使用，非本市所有门店通用！拉手券消
费仅限每日19:00前！
健康有道，骏越万里！",
                    "address" : "虹口区四川北路1896号-1904号20
1室"
                }
            }
        ],
        "stats" : {
            "time" : 0,
            "btreelocs" : 0,
            "nscanned" : 18,
            "objectsLoaded" : 12,
            "avgDistance" : 0.00009964377923093564,
            "maxDistance" : 0.0001064199324957278
        },
        "ok" : 1
}
```

可以看到返回了很多详细信息，如查询时间、返回数量、最大距离、平均距离等。

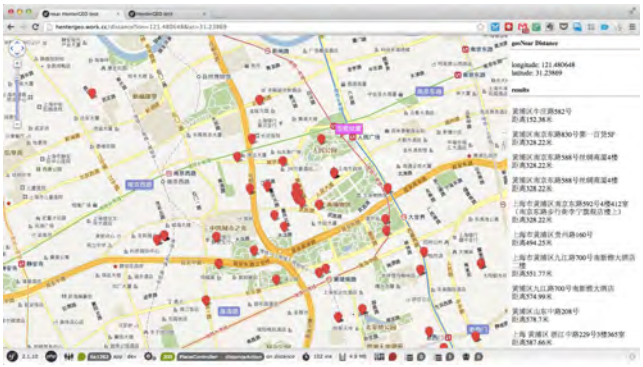
另外，results里面直接返回了距离目标点的距离dis。

Symfony2演示代码：

```
/**
 * @Route("/distance", name="distance")
 * @Template()
 */
public function distanceAction(){

    $longitude = (float)$this->getRequest()->get('lon',121.
4905);
    $latitude = (float)$this->getRequest()->get('lat',31.264
6);
    //2km
    $max = (float)$this->getRequest()->get('max', 2);

    $places = $this->getPlaceRepository()->createQueryBui
lder()
        ->field('coordinate')
        ->geoNear($longitude, $latitude)
        ->spherical(true)
        ->distanceMultiplier(6371)
        ->maxDistance($max/6371)
        ->limit(100)
        ->getQuery()
        ->execute()
        ->toArray();

    return compact('places','longitude', 'latitude', 'max');
}
```

通过 domian.dev/distance 访问，效果如下：

距离xxx米



## 小结

前面演示的查询代码中，坐标都是按照 longitude, latitude这个顺序的。

这个是官方建议的坐标顺序，但是网上很多文档是相反的顺序，经测试发现，只要查询时指定的坐标顺序与数据库内的坐标顺序一致，出来的结果就是正确的，没有特定的先后顺序之分。

但鉴于官方文档的推荐，我在此还是建议大家按照官方推荐的顺序。

案例A的$near和案例B的$center从需求上看差不多，但是$center或$centerSphere是属于$geoWithin的类型，$near方法查询后会对结果集对距离进行排序,而$geoWithin是无序的。

常用的查询方式已经介绍完了，不常用的比如geoIntersect查询，这里不做介绍，但是已经包含在开源的演示程序里了，有兴趣的读者可以自行测试研究。

下面介绍前文提到的距离单位等问题。

# 5. 需要注意的问题

## 索引

$near命令必须要求有索引。

$geoWithin可以无需索引，但是建议还是建立索引以提升性能。

## 距离单位

MongoDB查询地理位置默认有3种距离单位：

米(meters)
平面单位(flat units，可以理解为经纬度的“一度”)
弧度(radians)。

通过GeoJSON格式查询，单位默认是米，通过其它方式则比较混乱，下面详细解释一下。

下面的查询语句指定距离内的目标：

```
> db.places.find({'coordinate':{$near: [121.4905, 31.2646
], $maxDistance:2}})
```

现在$maxDistance参数是2，但是如果我要查询如“附近500米内的餐厅”这样的需求，这个参数应该是多少？

关于距离计算，MongoDB的官方文档仅仅提到了弧度计算，未说明水平单位（度）计算。

关于弧度计算，官方文档的说明是：

> To convert: distance to radians: divide the distance by the radius of the sphere (e.g. the Earth) in the same units as the distance measurement. radians to distance: multiply the radian measure by the radius of the sphere (e.g. the Earth) in the units system that you want to convert the distance to.
>
> The radius of the Earth is approximately 3,959 miles or 6,371 kilometers.

所以如果用弧度查询，则以公里数除以6371，如“附近500米的餐厅”：

```
> db.runCommand( { geoNear: "places", near: [ 121.4905, 31.2
646 ], spherical: true,
 $maxDistance: 0.5/6371 })
```

那如果不用弧度，以水平单位（度）查询时，距离单位如何处理？

答案是以公里数除以111（推荐值），原因如下：

经纬度的一度，分为经度一度和纬度一度。

地球不同纬度之间的距离是一样的，地球子午线（南极到北极的连线）长度39940.67公里，纬度一度大约110.9公里

但是不同纬度的经度一度对应的长度是不一样的：

在地球赤道，一圈大约为40075KM,除以360度，每一个经度大概是：40075/360=111.32KM

上海，大概在北纬31度，对应一个经度的长度是：40075*sin(90-31)/360=95.41KM

北京在北纬40度，对应的是85KM

前面提到的参数111，这个值只是估算，并不完全准确，任意两点之间的距离，平均纬度越大，这个参数则误差越大。详细原因可以参考wiki上的解释：http://en.wikipedia.org/wiki/Latitude

但是，即便如此，“度”这个单位只用于平面，由于地球是圆的，在大范围使用时会有误差。

官方建议使用sphere查询方式，也就是说距离单位用弧度。

> The current implementation assumes an idealized model of a flat earth, meaning that an arcdegree of latitude (y) and longitude (x) represent the same distance everywhere. This is only true at the equator where they are both about equal to 69 miles or 111km. However, at the 10gen offices at { x : -74 , y : 40.74 } one arcdegree of longitude is about 52 miles or 83 km (latitude is unchanged). This means that something 1 mile to the north would seem closer than something 1 mile to the east.

$geoNear返回结果集中的dis，如果指定了spherical为true， dis的值为弧度，不指定则为度。

指定 spherical为true,结果中的dis需要乘以6371换算为km:

```
> db.runCommand( { geoNear: "places", near: [ 121.4905, 3
1.2646 ],
spherical: true, num:1 })
{
    "ns" : "mongo_test.places",
```

```
    "near" : "1110001100111100001011010110010111001000
110011111101",
    "results" : [
        {
            "dis" : 0.00009318095248858048,
            "obj" : {
                "_id" : 115,
                "coordinate" : {
                    "longitude" : 121.4915,
                    "latitude" : 31.25933
                },
                "title" : "仅售148元，市场价298元的星程上服
假日酒店全日房一间入住一天，节假日通用，
精致生活，品质享受",
                "address" : "虹口区天水路90号"
            }
        }

    ],
    "stats" : {
        "time" : 0,
        "btreelocs" : 0,
        "nscanned" : 18,
        "objectsLoaded" : 12,
        "avgDistance" : 0.00009964377923093564,
        "maxDistance" : 0.0001064199324957278
    },
    "ok" : 1
}
```


不指定sphericial，结果中的dis需要乘以111换算为km:

```
> db.runCommand( { geoNear: "places", near: [ 121.4905, 31.
2646 ], num:1 })
{
```

```
    "ns" : "mongo_test.places",
    "near" : "1110001100111100001011010110010111001000110
011111101",
    "results" : [
        {
            "dis" : 0.005364037658335473,
            "obj" : {
                "_id" : 115,
                "coordinate" : {
                    "longitude" : 121.4915,
                    "latitude" : 31.25933
                },
                "title" : "仅售148元，市场价298元的星程上服假日酒店全
日房一间入住一天，节假日通用，
精致生活，品质享受",
                "address" : "虹口区天水路90号"
            }
        }

    ],
    "stats" : {
        "time" : 0,
        "btreelocs" : 0,
        "nscanned" : 18,
        "objectsLoaded" : 12,
        "avgDistance" : 0.006150808243357531,
        "maxDistance" : 0.00695541352612983
    },
    "ok" : 1
}
```


说到这里读者是不是已经有点迷糊了？没关系，在开发中其实你并不需要去知道各种距离单位的历史和使用它的原因，我在此为你总结了一张表，大部分常用的函数和所使用的距离单位都已经被我整理了出来，你只需要参考表上所列的距离单位直接使用即可。

| 查询命令 | 距离单位 | 说明 |
|---|---|---|
| $near | 度 | 官方文档上关于这一点是错的 |
| $nearSphere | 弧度 | |
| $center | 度 | |

| | |
|---|---|
| $centerSphere | 弧度 |
| $polygon | 度 |
| $geoNear | 度或弧度 指定参数spherical为true则为弧度，否则为度 |

如果坐标以GeoJSON格式，则单位都为米。

当然如果你的操作比较复杂，或者希望知道更加详细的对照关系，也可以参考官方的这个更详细的对比表格：http://docs.mongodb.org/manual/reference/operator/query-geospatial/

## 单位自动换算

如上面两个geoNear示例，结果中的dis，前文已经提过这是与目标点的距离，但是这个距离单位是跟查询单位一致的，需要二次计算，不太方便。

而其实可以直接在查询时指定 distanceMultiplier ，它会将这个参数乘以距离返回，如指定为6371，返回的就是公里数。

```
> db.runCommand({ geoNear : "places", near : [121.4905, 31
.2646], spherical : true,
 maxDistance : 1/6371, distanceMultiplier: 6371})
{
    "ns" : "mongo_test.places",
    "near" : "1110001100111100001011010110010111001000
110011111101",
    "results" : [
        {
            "dis" : 0.5936558483047463,
            "obj" : {
                "_id" : 115,
                "coordinate" : {
                    "longitude" : 121.4915,
                    "latitude" : 31.25933
                },
                "title" : "仅售148元，市场价298元的星程上服
假日酒店全日房一间入住一天，节假日通用，
精致生活，品质享受",
                "address" : "虹口区天水路90号"
```

```
                }
            },

            …
            …

        ],
        "stats" : {
            "time" : 0,
            "btreelocs" : 0,
            "nscanned" : 15,
            "objectsLoaded" : 9,
            "avgDistance" : 0.6348305174802911,
            "maxDistance" : 0.00010641993249572 78
        },
        "ok" : 1
    }
```

注意上面的结果中dis的值，已经是km单位的了。

# 结语

通过前面的案例演示，相信大家对MongoDB的地理位置特性已经比较了解。

MongoDB还有很多很酷的功能，地址位置支持仅是其中一项。希望以后能有机会为各位读者介绍如何结合Symfony2使用MongoDB进行应用开发的更多案例。

文中的演示程序已经发布在了Github上，地址是https://github.com/henter/HenterGEO，读者可以直接使用。

参考：

http://docs.mongodb.org/manual/

https://wiki.10gen.com/pages/viewpage.action?pageId=21268367&navigatingVersions=true

http://en.wikipedia.org/wiki/Radian

http://www.scribd.com/doc/2569355/Geo-Distance-Search-with-MySQL

http://www.phpchina.com/resource/manual/mysql/spatial-extensions-in-mysql.html

http://derickrethans.nl/spatial-indexes-mysql.html

http://dev.mysql.com/doc/refman/5.6/en/spatial-extensions.html

http://dev.mysql.com/doc/refman/4.1/en/functions-that-test-spatial-relationships-between-geometries.html#function_distance

http://blog.nosqlfan.com/html/1811.html

http://en.wikipedia.org/wiki/Geohash

# 关于作者

周攀，多年互联网从业经验，精通Symfony2框架，微博阅后即焚应用SnapWeibo的作者，斯诺克爱好者，养猫达人。读者可以通过他的微博 @周攀Henter或邮件（henter at henter dot me）与他取得联系。

洪涛，在互联网、零售、电信领域有多年的从业经验，曾负责中国电信域名纠错平台的开发，也曾为雅虎、腾讯等大型互联网站进行架构设计与开发工作，善于使用开源技术解决技术难题。读者可以通过他的微博@斯考吉恩或邮件（scourgen at gmail dot com）与他取得联系。

**原文链接：http://www.infoq.com/cn/articles/depth-study-of-Symfony2**

**相关内容**

- Jelastic 1.9.2新增GIT、SVS、PHP 5.5和MongoDB备份计划支持
- 10gen发布MongoDB增量备份服务
- NoSQL基准对比Aerospike、Cassandra、Couchbase和MongoDB
- MoSQL：简化MongoDB与PostgreSQL之间的同步
- Simba Technologies正将SQL的强大能力带向Cassandra、Hadoop、BigQuery和MongoDB

# 百度开发者中心

Baidu Developer Center

**应用万象 云创未来**

百度秉持“平等地成就每一个开发者”的理念，围绕百度云平台能力整合开放给开发者四大服务体系：开发支持、 运营支撑、渠道推广、商业变现。

我们希望藉由这些服务的开放，让开发者在开发、运营、推广、变现各环节有所受益，最大程度地保证开发者的利益，，共建开放共赢的局面！

开发支持

运营支撑

渠道推广

商业变现

Baidu 百度

http://developer.baidu.com

# 推荐文章 | Article

# 1688域名切换——看我空中换引擎

作者 冯凌圣

## 一、背景

经过10多年发展、沉淀，阿里巴巴中文站从最初的单个应用系统发展超过100个应用系统，500个以上域名。

早期有不少域名切换案例，但这次切换跟之前项目规模、复杂度要大许多,且要求严格，需要一次性全部干干净净的切换。

## 二、挑战

协调超过100个应用系统、几百个域名同时切换，各个应用之间往往有各种依赖关系，一个地方出错很容易引起连锁效应，造成大面故障。所以，在这种情况下，我们必须保证中文站以及各个兄弟部门的业务正常的情况下，完成这个域名切换乾坤大挪移。

同时在商业上，需要保障客户的权益，防止SEO流量下降，防止P4P、外投广告的收益保持稳定，等等，项目组感到压力重大。

## 三、切换方案

### 1、切换域名面临的技术挑战

1）一次性整体切换

如果按照应用维度逐一切换到1688域，需要考虑cookie双域读写和跨域请求问题。cookie分三类：

a、永久cookie，保存在用户本地电脑上，比如用户标识，访问行为跟踪，识别是否新用户等。

b、登录状态cookie（session cookie），全站共享，在主域名下。

c、业务cookie，各个业务应用读写的cookie，在子域名或者主域名下。

Web标准无法实现cookie的双域直接读写，只能通过302跳转或者内嵌url的方式间接实现。但是这种方案对应用浸入性比较大，非常复杂，难以大面积的实施；且即使使用，首pv的cookie丢失问题同样难以接受。所以，考虑到用户体验和切换效果及实施成本，项目组决定选择一次性整体切换。

2）各应用需要独立修改实施

目前有150个左右web应用，大约600个域名，还有部分数据库保持有模板信息，应用之间的相互依赖关系复杂，很难在同一时刻完成所有应用和数据库内容的修改且一次性发布。所以，必须采取策略保证各应用能够独立发布修改，解除应用之间的改造耦合。

## 2、方案策略

1）保护罩。给整个中文站所有与外界（用户）交互的应用系统（在系统边界）套上一个保护罩，这个保护罩对内部应用系统的修改实施过程进行保护，使应用的修改发布对外界不影响，这样各个应用之间的依赖关系解耦，各个应用可以逐个独立修改发布，在整个过程屏蔽外界用户的感知，保证在实施过程用户保持在同一个主域名及其子域名（*.alibaba.com）下访问，在实施过程中现有功能不会受到任何影响，网站所有功能照常提供服务。

2）为了保证全套环境——包括应用服务器、数据库、配置服务器、文件系统——在保护罩保护之下，线上（生产）环境支持双模式运行：一个是正式模式（对应域名*.alibaba.com*），*正常用户在正式模式下访问；另外一个是测试模式（对应域名*.1688.com），在测试模式下，保护罩不起作用，测试人员看到的效果跟实际代码一致，这样就可以利用线上全套真实环境进行全站联测，线上系统、配置服务器和数据库可以随时进行修改并订正。

3）待所有应用修改完成，全站线上新域名联测，完成联测之后去除保护罩，实现一次性切换。

## 3、方案实现

1）全站实施支持双域cookie工作改造，支持多套域配置，老域名采用老cookie配置，新域名采用新cookie配置。

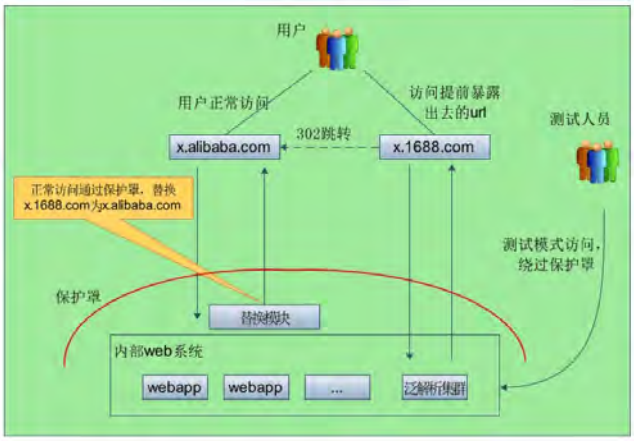
2）web内容输出增加保护罩，在web内容完成渲染输出时对内容以及http head location替换（考虑到静态内容替换，在apache层替换，需要开发一个apache replace module）。这个模块支持双模式工作，在正式模式下，对输出的web内容（已经修的内容），扫描分析并反向替换回老域名模式，即把内

容里面发现新域名的替换回到老域名，因此对用户屏蔽了改变，在实施过程用户不会影响现有功能；在测试模式下不做替换，测试模式下访问返回的内容和已经修改的代码保持一致，这样就可以提供线上全套测试解决方案，且可以逐个应用发布上线，应用之间的依赖完全解耦。

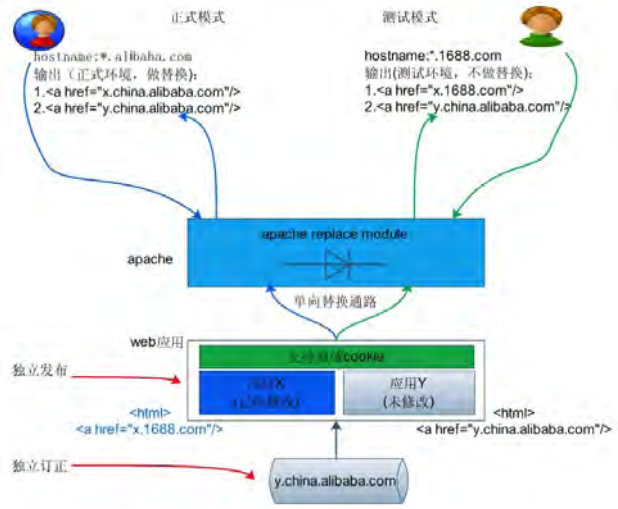
3）双模式支持，除了AliSession和apache replace module对双模式工作支持之外，apache配置也需要增加虚拟主机配置来支持双域名访问，保证外部用户用老域名访问，内部测试人员和实施过程中apache反向代理、httpclient方式，新域名能访问。

一次性切换，完成所有联测之后，把apache替换模块一次性下线（也可以做成开关模式，但最终需要下线），去除系统和用户之间的感知屏蔽层，实现全站一次性切换。

4）切换之后，所有老域名的访问301跳转至新域名。




总体方案设计图




替换工作原理图解

## 4、补偿方案

无论方案多么完美，难免会有些内容（不包含url）绕过保护罩泄露出去，比如邮件内容，这些内容不经过web服务器request/reponse模式出去，因此apache替换模块不起作用。如果逐一去封闭各个出口，成本会比较大，而且这类途径泄露出去的内容占比很小，所以采用泛域名方式来解决。如保护罩设计图中的路径2，所有新域名（新增的*.1688.com域名）会泛解析到static集群，这个集群做302跳转到相应的alibaba.com的url，从而保障了可用性。但是对于先前已经提供了1688域名的应用，如路径3，需要单独采取类似的方案。

### 5、服务器间的访问保护方案

服务器之间通常存在反向代理，或者通过httpclient、rmi等调用方式的数据服务访问，一旦这些访问的域名修改为了对应的1688域名，没有提前支持，肯定是有问题故障的（因为域名没开发，服务不可达）。我们采取的方案是，对内部环境新域名提前开放，保证服务器之间通过新域名访问可达。

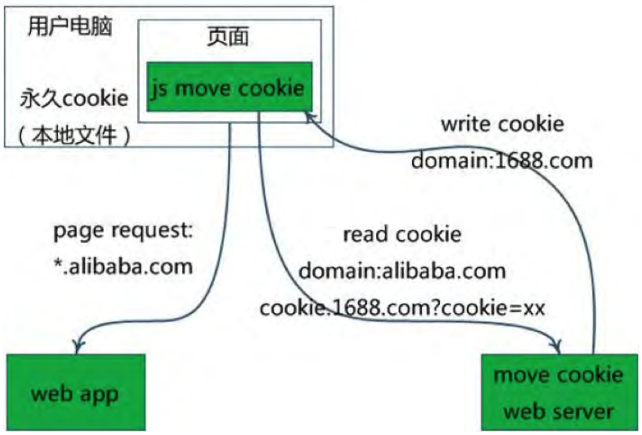
### 6、测试模式识别

线下环境直接提供两种模式：正式模式和测试模式。为了把测试模式和正式模式区分开，我们采用在cookie下记录一个标志来区分。为了方面使用，开发一个简单页面，测试人员在进入测试模式之前，点击这个页面按钮切换模式，这个页面功能在双域下记录一个测试标记cookie，apache根据这个来判断是否属于测试模式。在测试模式下，apache replace module不做替换，apache rewrite rule不会外部重定向至alibaba域的域名。

## 四、其他问题解决方案

### 1、基于cookie数据统计跟踪断层问题解决

采取的方案是将老域名的永久cookie迁移至新域名。实现很简单，成本也很低，在通用js（覆盖大部分页面）埋入迁移js即可。注意在迁移cookie的过程中，有可能改变cookie的值，因为cookie标准的不一样，对于有些值会加引号，使用方没有兼容的话，很容易产生故障。



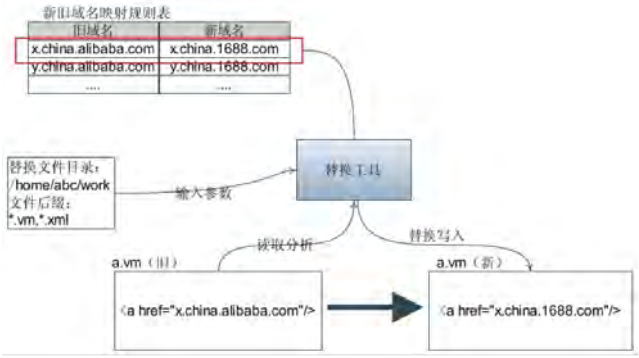

## 2、SEO流量影响的解决

1. 各重要应用重新在百度站长平台，google,bing的webmaster重新验证新的域名权限。
2. 重要应用都在百度站长平台网站改版功能中，提交域名切换规则，同时，调整各大搜索引擎爬虫的抓取压力。
3. 修改sitemap文件生成规则，将sitemap文件中的URL全部替换成新的URL，同时实施sitemap地址安全化。
4. 修改百度开发适配协议，重新调整URL的映射关系。
5. 对于各应用暴露出去的RPC服务，或者URL服务，添加到robots禁止策略中，禁止各搜索引擎抓取，提升爬虫抓取效率。
6. 寻求百度，360，sogou，soso等搜索引擎帮助，请求搜索引擎算法部门实施域名切换，进行数据归一化，索引更新。(依赖新老域名映射清单，便于搜索引擎算法部门实施域名切换)

## 3、代码替换工具

首先建立核心映射关系表，梳理域名切换转换关系表格，老<-->新域名关系，包含一对一和一对多的关系，包括三级域名升级为二级域名，域名换名称等，后续所有工作都围绕这个表格进行。整个梳理过程相当繁琐，对几百个域名逐一沟通确认。

开发代码替换工具，同时可以提供给替换数据库，这个工具依赖的核心是域名映射列表。




# 五、总结

通过三大利器（保护罩、替换工具、补充模式）解决了切换存在的各种技术难题和大大降低了切换实施成本，并通过SEO优化策略确保切换之后影响尽可能小。

1688域名切换项目正式上线一周全站业务系统非常稳定，未产生任何重大故障，客户反应也比较平滑，项目非常成功，达到了我们的预期目的，同时还顺带解决了一些历史包袱：

1、静态资源（图片、js、css等）彻底独立于主域名，迁移至无cookie环境，解决了历史包袱，节省大量网络带宽消耗。

2、域名优化改善，包括域名合并，不合理域名清理，域名升级等。

# 作者简介

冯凌圣，阿里巴巴应用架构师，2004年底加入阿里巴巴，主导过中文站许多重点项目及应用架构、性能优化，独自研制开发json安全解决方案、jmemory（java版嵌入式堆外内存使用，类似简单memcached的嵌入式版本），目前主导阿里巴巴中文站应用架构。

**原文链接：http://www.infoq.com/cn/articles/1688-domain-switch**

**相关内容**

- 对话伯昊、林轩：阿里TAE的发展故事
- 架构师（8月刊）
- 阿里巴巴M-DMP负责人许成访谈：从顾客管理到消费者管理的升级
- GAE 1.8.2引入专属Memcache和模块化
- 基于云计算的产品开发

# 推荐文章 | Article

# 基于AWS技术实现发布/订阅服务

作者 Boris Lublinsky ，译者 王丽娟

AWS提供两种服务——Amazon简单通知服务（Simple Notification Service）和Amazon简单队列服务（Simple Queue Service），两者结合起来可以为完整的发布/订阅服务提供支撑。

## 现有的AWS功能

Amazon简单通知服务（Amazon SNS）是一个Web服务，能让应用、最终用户和设备立即从云端发送和接收通知。简化的SNS架构如下图所示（图1）：

（点击查看大图）

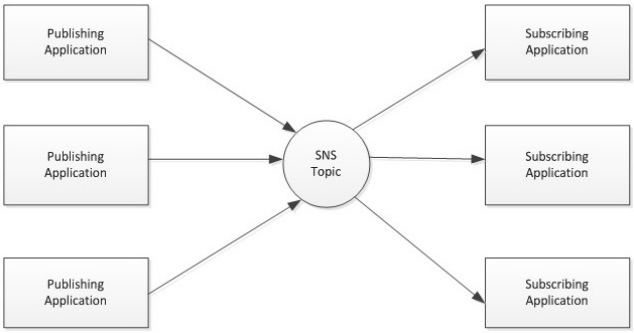



**图1：Amazon SNS的基础架构**

多个发布应用和多个订阅应用可以将SNS主题作为中介互相通讯。这样实现的优点是发布者和订阅者不需要知道对方，因此，应用可以完全动态地进行集成。SNS支持用多种传输协议传递通知，包括HTTP、HTTPS、Email、SMS和Amazon简单队列（Simple Queue）。

Amazon简单队列服务（Amazon SQS）提供可靠、可伸缩的托管队列，用来存储计算机之间传输的消息。使用Amazon SQS，你可以在执行不同任务的应用分布式组件之间移动数据，而不会丢失消息，也不必要求每个组件始终都是可用的。SQS和SNS结合起来会带来两个额外的优势——解除时间上的耦合度，根据消费应用特定的情况提供负载均衡——这是SNS无法单独提供的。要做到第二个附加优势，需要同一个应用的多个实例从同一个队列里读取消息。

下图展示了SNS和SQS结合的总体架构（图2）。其中的一个订阅应用显示为负载均衡的。

（点击查看大图）

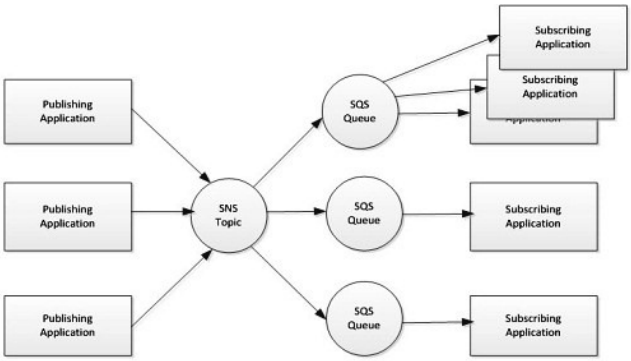



**图2：结合SNS和SQS**

这个实现的主要缺点是，发布者和订阅者需要明确统一SNS主题的名称。此外，如果一个特定的消费者想从多个主题获取信息，那他需要把队列注册到多个主题上。

# 期望中的发布/订阅实现

这个问题的典型解决方案是采用基于树的主题组织，大部分发布/订阅引擎都是这样实现的。OASIS规范的Web Services Topics 1.3概述了这种组织的主要原则。

这个规范将主题定义为：

> “……主题是一组通知的组织和分类方式。主题机制为订阅者推断出感兴趣的通知提供了便捷的方式……发布者可以将通知发布和一或多个主题关联起来。当订阅者创建订阅的时候，可以提供一个主题的过滤器表达式，将订阅和一或多个主题关联起来……每个主题都可以有零或多个**子主题**，子主题本身也可以进一步包含子主题。没有**“父亲”**的主题叫**根主题**。特定的根主题和它所有的后代会形成一个层次结构（称为**主题树**）。”

下面是手机销售的一个主题树例子（图3）。

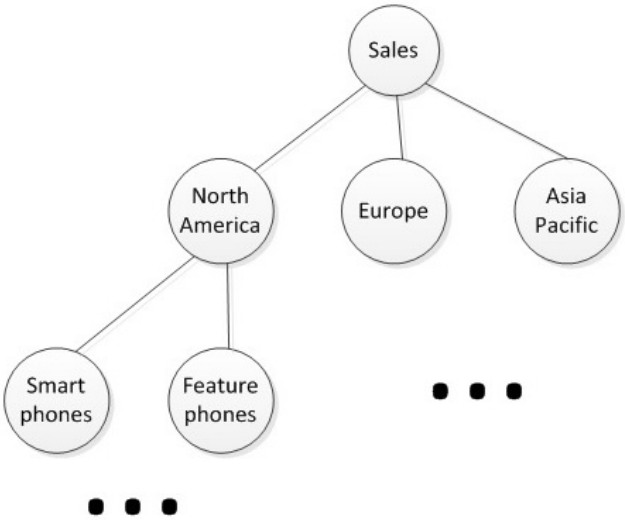



**图3：主题树示例**

主题树的根表示销售。销售可以按区域细分（在我们的例子中有北美、欧洲和亚太地区）。特定区域的销售还可以按照手机类型进一步细分，依此类推。

在发布/订阅系统中，这样的结构之所以重要是因为树反映了数据的组织。如果消费者对北美的智能手机销售感兴趣，他可以监听这个特定的主题。如果他对北美所有的销售都感兴趣，那他就可以监听北美的主题，从子主题获取所有的通知。

当然，这种方法并不能解决所有的问题。比如说，如果消费者想监听所有智能手机销售的事件，他就需要明确订阅所有地区的智能手机销售事件。这种情况通常是主题树设计的问题。树的设计基于信息的组织和典型的使用模式。在某些情况下，会设计多个主题来满足不同的内部需求（参见Web Services Topics 1.3里的主题命名空间）。发布/订阅架构的另一个重要特性就是基于内容的消息过滤：

> “在基于内容的系统中，如果消息的属性或内容与订阅者定义的约束相匹配，消息就只会传递给这个订阅者。订阅者负责消息的分类。”

换句话说，订阅者在这种情况下可以使用正则表达式列表，明确指定他们感兴趣的消息内容。

把这种过滤和结构化的主题结构结合起来，可以创建出非常灵活和强大的发布/订阅实现。

我们将在本文中展示如何用AWS组件轻松构建这类系统。

# 发布/订阅架构建议

建议给大家的架构如下图所示（图4）。在这个架构中，发布/订阅服务器的实现是一个Tomcat容器里运行的Web应用。我们还充分利用了AWS的弹性负载均衡器（Elastic Load Balancer），它可以根据当前的负载动态扩展或缩减发布/订阅服务器集群的大小。此外，架构还用关系型数据服务（Relational Data Service）存储当前的配置，以便动态新增发布/订阅实例。为了提高整体性能，我们在内存里保留了当前的拓扑结构，尽量减少数据库访问的次数。这样的话，实际的消息路由会非常迅速。这个解决方案需要一种机制，能在拓扑结构发生变化的时候去通知所有的服务器（因为任何服务器都能处理负载均衡器）。Amazon SNS能轻而易举地做到这一点。最后，我们用Amazon SQS将通知分发给消费者。需要注意的是，一个消费者可以监听多个队列。

（点击查看大图）

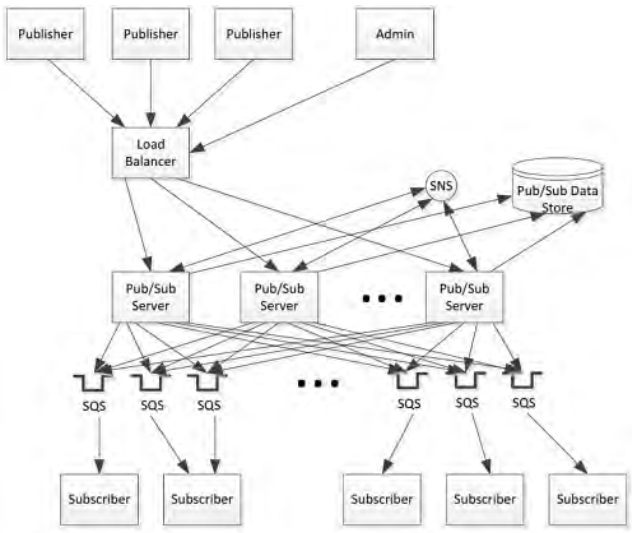



**图4：整体架构建议**

# 发布/订阅服务器

这个实现的核心是一个自定义的发布/订阅服务器。服务器实现包括三个主要的层——持久化、域和服务。

## 持久化

服务器持久化层采用JPA 2.0实现，定义了三个主要的实体——主题、订阅和语义过滤器。

主题实体（清单1）描述了特定主题要存储的相关信息，包括主题ID（数据库的内部ID）、主题名称（标识主题的字符串）、一个布尔变量（定义该主题是否是个根主题）、到父主题和孩子主题的引用（以便对主题层次结构进行遍历），以及与给定主题关联的订阅列表。

```
@Entity
@NamedQueries({
    @NamedQuery(name="Topic.RootTopics",
                query="SELECT t FROM Topic t where t.root='true'"),
    @NamedQuery(name="Topic.AllTopics",
                query="SELECT t FROM Topic t")
})
@Table(name = "Topic")
public class Topic {

 @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
 private long id;     // 自动生成的ID

 @Column(name = "name",nullable = false, length = 32)
 private String name;                          // 主题名称

 @Column(name = "root",nullable = false)
 private Boolean root = false;            // 根主题标识

 @ManyToOne(fetch=FetchType.LAZY)
 @JoinColumn(name="TOPIC_ID")
 private Topic parent;

 @OneToMany(mappedBy="parent",cascade=CascadeType.ALL,orphanRemoval=true)
 private List<Topic> children;

 @OneToMany(mappedBy="topic",cascade=CascadeType.ALL,orphanRemoval=true)
 private List<Subscription> subscriptions;
 ……
```

**清单1：主题实体**

我们定义了两个命名的查询，用来访问主题：RootTopics获取从根开始的主题结构，AllTopics获取所有现有的主题。

这个实体提供了一个完整的主题定义，也可以支持多个主题树（而不是实现示例的一部分）。

订阅实体（清单2）描述了订阅相关的信息，包括订阅ID（数据库的内部ID）、队列名称（SQS队列的ARN，ARN即Amazon Resource Name）、对订阅关联主题的引用，还有一个语义过滤器列表。只有所有的过滤器都接受消息（见下文），通知才会分发给给定的队列（客户端）。如果通知不包含语义过滤器，那来自于关联主题的所有消息都会直接传递给队列。

```
@Entity
@NamedQueries({
    @NamedQuery(name="Subscription.AllSubscriptions",
                query="SELECT s FROM Subscription s"
)
})
@Table(name = "Subscription")
public class Subscription {

        @Id @GeneratedValue(strategy=GenerationType.IDE
NTITY)
        private long id;     // 自动生成的ID

        @Column(name = "queue",nullable = false, length
 = 128)
        private String queue;

        @ManyToOne(fetch=FetchType.LAZY)
        @JoinColumn(name="TOPIC_ID")
        private Topic topic;

        @OneToMany(mappedBy="subscription",
                                cascade=CascadeType.AL
L,orphanRemoval=true)
        private List<SemanticFilter> filters;
```

```
        ..............................................................
```

**清单2：订阅实体**

我们还定义了一个命名的查询，获得所有存在的订阅。

最后，语义过滤器实体（清单3）描述了特定语义过滤器的信息，包括语义过滤器ID（数据库的内部ID）、该语义过滤器测试的属性名称、使用的正则表达式，以及对语义过滤器关联订阅的引用。

```
@Entity
@NamedQueries({
    @NamedQuery(name="SemanticFilter.AllSemanticFilters",
                query="SELECT sf FROM SemanticFilter
sf")
})
@Table(name = "Filter")
public class SemanticFilter {

        @Id @GeneratedValue(strategy=GenerationType.IDE
NTITY)
        private long id;     // 自动生成的ID

        @Column(name = "attribute",nullable = false, leng
th = 32)
        private String attribute;                    // 属
性名称

        @Column(name = "filter",nullable = false, length =
 128)
        private String filter;                    // 正则表达
式过滤器

        @ManyToOne(fetch=FetchType.LAZY)
        @JoinColumn(name="SUBSCRIPTION_ID")
        private Subscription subscription;
        ..............................................
```

**清单3：语义过滤器实体**

我们一样定义一个命名的查询，用来获取所有现有的语义过滤器。

除了实体，持久化层还包含一个持久化管理类，负责：

管理数据库访问和事务

从数据库读取、写入对象

对域对象（见下文）和持久化实体进行相互转换

发送拓扑结构变化的通知

# 域模型

域模型对象的主要职责是支持服务操作，包括数据的订阅和发布，并把通知真正发布到订阅的队列上。在这个简单的实现里，域模型和持久化模型是合在一起的，但为了阐述得更清楚，我们分开介绍。这两层的数据模型是一样的，但域对象会多一些明确支持发布/订阅实现的方法。

过滤器处理的实现（清单4）利用了Java String里对正则表达式处理的内置支持。

```
public boolean accept(String value){
        if(value == null)
                return false;
        return value.matches(_pattern);
}
```

**清单4：过滤器处理方法**

发布实现（清单5）是订阅类的一个方法。请注意，这个方法对语义过滤器进行了或操作。如果给定的客户端能有多个订阅，或者对订阅实现进行扩展、让它支持Boolean函数，那就可以突破这个限制了。

```
public void publish(Map<String, String> attributes, Strin
g message){
```

```
    if((_filters != null) && (_filters.size() > 0)){
        for(DomainSemanticFilter f : _filters){
            String av = attributes.get(f.getField());
            if(av == null)
                return;
            if(!f.accept(av))
                return;
        }
    }
    SQSPublisher.getPublisher().sendMessage(_queue, mes
sage);
}
```

**清单5：发布实现**

这个实现利用了基于现有AWS Java API的SQSPublisher类（清单6）。

```
import java.io.IOException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.sqs.AmazonSQSClient;
import com.amazonaws.services.sqs.model.CreateQueueRequ
est;
import com.amazonaws.services.sqs.model.DeleteQueueRequ
est;
import com.amazonaws.services.sqs.model.SendMessageRequ
est;

public class SQSPublisher {

  private static SQSPublisher _publisher;

  private AmazonSQSClient _sqs;

  private SQSPublisher()throws IOException {
            AWSCredentials credentials = new Properties
```

```
Credentials(
                this.getClass().getClassLoader().
getResourceAsStream("AwsCredentials.properties"));
            _sqs = new AmazonSQSClient(credentials);
  }

  public String createQueue(String name){
            CreateQueueRequest request = new CreateQueue
Request(name);
            return _sqs.createQueue(request).getQueueUrl(
);
  }

  public void sendMessage(String queueURL, String messag
e){
            SendMessageRequest request = new SendMessage
Request(queueURL,
message);
            _sqs.sendMessage(request);

  }

          public void deleteQueue(String queueURL){
            DeleteQueueRequest request = new DeleteQueue
Request(queueURL);
            _sqs.deleteQueue(request);
  }

  public static synchronized SQSPublisher getPublisher(){
            if(_publisher == null)
                      try {
                                _publisher = new SQSPubl
isher();
                      }catch (IOException e) {
                                e.printStackTrace();
                      }
```

```
            return _publisher;
    }

}
```

**清单6：SQS发布者**

订阅者可以利用这个类的其他方法创建/销毁SQS队列。

除了SQS队列，我们的实现还利用SNS进行数据库变化的同步。与SNS的交互由SNSPubSub类实现（清单7），这个实现也利用了AWS SNS Java API。

```
import java.io.IOException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.sns.AmazonSNSClient;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.SubscribeRequest;
import com.amazonaws.services.sns.model.SubscribeResult;
import com.amazonaws.services.sns.model.UnsubscribeReque
st;

public class SNSPubSub {

    private static SNSPubSub _topicPublisher;
    private static String _topicARN;
    private static String _endpoint;

    private AmazonSNSClient _sns;
    private String _protocol = "http";
    private String _subscriptionARN;

    private SNSPubSub()throws IOException {
            AWSCredentials credentials = new Propertie
sCredentials(
                this.getClass().getClassLoader().
```

```
getResourceAsStream("AwsCredentials.properties"));
            _sns = new AmazonSNSClient(credentials);
    }

    public void publish(String message){
            PublishRequest request = new PublishReques
t(_topicARN, message);
            _sns.publish(request);
    }

    public void subscribe(){
            SubscribeRequest request = new SubscribeRe
quest
(_topicARN, _protocol, _endpoint);
            _sns.subscribe(request);
    }

    public void confirmSubscription(String token){
            ConfirmSubscriptionRequest request = new
 ConfirmSubscriptionRequest(_topicARN, token);
            ConfirmSubscriptionResult result = _sns
.confirmSubscription(request);
            _subscriptionARN = result.getSubscriptionAr
n();
    }

    public void unSubscribe(){
            if(_subscribed){
                    UnsubscribeRequest request = new
 UnsubscribeRequest(_subscriptionARN);
                _sns
.unsubscribe(request);
            }
    }

    public static void configureSNS(String topicARN, Strin
g endpoint){
```

```
                _topicARN = topicARN;
                _endpoint = endpoint;
    }

    public static synchronized SNSPubSub getSNS(){
                if(_topicPublisher == null){
                        try{
                                _topicPublisher = new
 SNSPubSub();
                        }
                        catch(Exception e){
                                e.printStackTrace();
                        }
                }
                return _topicPublisher;
    }
}
```

**清单7：SNS Pub/Sub**

**使用SNS**

使用SNS的时候要谨记：订阅主题并不意味着你已经准备好监听主题。SNS订阅的过程包含两个步骤。向SNS发送订阅请求时，SNS返回的响应表明确认订阅的必要性。这正是清单8既有subscribe方法又有confirmSubscription方法的原因。

```
<xsd:complextype name="NotificationType">
    <xsd:sequence>
        <xsd:element name="Type" type="xsd:string" />
        <xsd:element name="MessageId" type="xsd:string" /
>
        <xsd:element name="Token" type="xsd:string" mino
ccurs="0" />
        <xsd:element name="TopicArn" type="xsd:string" />
        <xsd:element name="Message" type="xsd:string" />
```

```
        <xsd:element name="SubscribeURL" type="xsd:string" minoccurs="0" />
        <xsd:element name="Timestamp" type="xsd:string" />
        <xsd:element name="SignatureVersion" type="xsd:string" />
        <xsd:element name="Signature" type="xsd:string" />
        <xsd:element name="SigningCertURL" type="xsd:string" />
        <xsd:element name="UnsubscribeURL" type="xsd:string" minoccurs="0" />
    </xsd:sequence>
</xsd:complextype>
```

上面的Schema描述了两种消息类型——确认请求和实际的通知。两种类型通过Type元素进行区分。如果元素值是“SubscriptionConfirmation”，那它就是订阅确认的请求，如果是“Notification”，就表明是个真正的通知。

主题类实现了两个方法（清单8），以便支持发布。

```
public void publish(Map<String, String> attributes, String message){

    if(_subscriptions == null)
        return;
    for(DomainSubscription ds : _subscriptions)
        ds.publish(attributes, message);
}

public void processPublications(List<DomainTopic> tList, StringTokenizer st) throws PublicationException{

    tList.add(this);
    if(!st.hasMoreTokens())
        return;
    String topic = st.nextToken();
```

```
    for(DomainTopic dt : _children){
        if(topic.equalsIgnoreCase(dt.getName())){
            dt.processPublications(tList, st);
            return;
        }
    }
    throw new PublicationException("Subtopic " + topic +
 " is not found in topic " + _name);
}
```

**清单8：主题对发布的支持**

processPublications方法创建了一个主题列表，这些主题与给定的消息相关联。这个方法有一个标记过的主题树字符串，如果标记和主题名称相对应，就会把当前的主题添加到列表中。主题的publish方法维护一个消息属性的映射，对主题相关的每个订阅来说，publish方法还会尝试着去发布一条消息。

上面的方法都由Domain管理器类的publish方法调用（清单9）。这个方法首先标记主题字符串，然后用processPublications方法创建一个订阅者感兴趣的主题列表。列表一旦被创建好，就会构建一个消息属性的映射（我们假设是一个XML消息），并把这个映射发布给列表里的所有主题。

```
     public void publish (String topic, String message){
       StringTokenizer st = new StringTokenizer(topic,
".");
       List<DomainTopic> topics = new LinkedList<Domain
topic>();
       DomainTopic root = PersistenceManager.getPersiste
nceManager().getRoot();
       try {
                if(!st.hasMoreTokens())
                        return;
                String t = st.nextToken();
                if(!t.equalsIgnoreCase(root.getName()))
                        throw new PublicationException("
Unrecognized subtopic name " + topic);
                root.processPublications(topics, st);
```

```
        }catch (PublicationException e) {
                e.printStackTrace();
                return;
        }
        MessageType msg = null;
        try {
                JAXBElement<MessageType> msgEl = (JAXBEl
ement<MessageType>)
                        _unmarshaller.unmarshal(new ByteA
rrayInputStream(message.getBytes()));
                msg = msgEl.getValue();
        } catch (JAXBException e) {
                e.printStackTrace();
                return;
        }
        Map<String, String> attributes = new HashMap<Str
ing, String>();
        MessageEnvelopeType envelope = msg.getEnvelope();
        if(envelope != null){
                for(MessageAttributeType attribute : env
elope.getAttribute()){
                        attributes.put(attribute.getName(
), attribute.getValue());
                }
        }
        for(DomainTopic t : topics)
                t.publish(attributes, message);
    }
```

**清单9：发布方法实现**

# 服务模型

我们用一组REST服务对发布/订阅功能进行访问（清单10）。

```
@Path("/")
public class PubSubServiceImplementation {

    // 功能方法
    @POST
    @Path("publish")
    @Consumes("application/text")
    public void publish (@QueryParam("topic")String topi
c, String message) throws PublicationException{
            DomainManager.getDomainManager().publish(
topic, message);
    }

    @GET
    @Path("publish")
    public void publishGet (@QueryParam("topic")String t
opic, @QueryParam("message")String message)  throws
PublicationException{
            DomainManager.getDomainManager().publish(
topic, message);
    }

    @POST
    @Path("synch")
    @Consumes("text/plain")
    public void getSynchNotification (Object message){
            PersistenceManager.setUpdated();
    }

    // 配置方法

    @GET
    @Path("root")
    @Produces("application/json")
    public TopicType getRoot()throws PublicationException
 {
```

```
            return DomainManager.getDomainManager().g
etRoot();
    }

    @GET
    @Path("filters")
    @Produces("application/json")
    public FiltersType getFilters() throws PublicationExc
eption {
            return DomainManager.getDomainManager().g
etFilters();
    }

    @POST
    @Path("filter")
    @Consumes("application/json")
    public long addFilter(FilterType filter) throws Publ
icationException {
            return DomainManager.getDomainManager().a
ddFilter(filter);
    }

    @DELETE
    @Path("filter/{id}")
    public void deleteFilter(@PathParam("id")long id) thr
ows PublicationException {
            DomainManager.getDomainManager().removeFi
lter(id);
    }

    @GET
    @Path("subscriptions")
    @Produces("application/json")
    public SubscriptionsType getSubscriptions() throws Pu
blicationException {
            return DomainManager.getDomainManager().g
```

```
etSubscriptions();
    }

    @POST
    @Path("subscription")
    @Consumes("application/json")
    public long addSubscription(SubscriptionType s) throw
s PublicationException {
            return DomainManager.getDomainManager().a
ddSubscription(s, null);
    }

    @DELETE
    @Path("subscription/{id}")
    public void deleteSubscription(@PathParam("id")long i
d) throws PublicationException {
            DomainManager.getDomainManager().removeSu
bscription(id);
    }

    @POST
    @Path("subscriptionFilters/{sid}")
    @Consumes("application/json")
    public long assignFilersToSubscription(@PathParam("si
d")long sid, IDsType ids)throws PublicationException{
            return DomainManager.getDomainManager().a
ssignFilersToSubscription(sid, ids);

    }

    @POST
    @Path("topic")
    @Consumes("application/json")
    public long addTopic(TopicType t) throws PublicationE
xception {
            return DomainManager.getDomainManager().a
```

```
ddTopic(t, null);
    }

    @DELETE
    @Path("topic/{id}")
    public void deleteTopic(@PathParam("id")long id) throws PublicationException {
            DomainManager.getDomainManager().removeTopic(id);
    }

    @POST
    @Path("topicsubscription/{tid}")
    @Consumes("application/json")
    public void assignTopicHierarchy(@PathParam("tid")long tid, IDsType ids) throws PublicationException{
            DomainManager.getDomainManager().assignTopicHierarchy(tid, ids);
    }

    @POST
    @Path("topicsubscription/{tid}")
    @Consumes("application/json")
    public long assignTopicSubscriptions(@PathParam("tid")long tid, IDsType ids)throws PublicationException{
            return DomainManager.getDomainManager().assignTopicSubscriptions(tid, ids);
    }
```

**清单10：发布/订阅服务**

这些服务的使用者有消息发布者（publish方法）、服务订阅者（创建/删除语义过滤器，订阅，还有订阅和主题订阅相关的过滤器）、内部的发布/订阅实现（获取同步的服务）和管理应用。

# 结论

这个实现虽然简单，但创建了一个非常强大、可扩展的发布/订阅实现，同时利用了很多现有的AWS功能和少量的Java定制代码。另外它还充分利用了现有AWS部署功能对负载均衡和容错的支持。

# 作者简介

**Boris Lublinsky博士**是Nokia的主要架构师，参与大数据、SOA、BPM和中间件实现的相关工作。Boris去Nokia前是Herzum软件的主要架构师，负责为客户设计大型、可伸缩的SOA系统；在此之前，他是CNA保险的企业架构师，参与CNA集成和SOA策略的设计及实现，构建应用框架，实现面向服务的架构。Boris在企业技术架构和软件工程方面有二十五年多的经验。他是OASIS SOA RM委员会的活跃成员，和他人一起编著了《Applied SOA: Service-Oriented Architecture and Design Strategies》一书，另外他还写了很多关于架构、编程、大数据、SOA和BPM的文章。

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

**相关内容**

- AWS和Microsoft发布了竞争的移动通知服务
- AWS云搜索的使用：极简Java API
- 拥抱Node？最新的AWS云提供Node.js应用服务
- Microsoft和AWS推出免费的云优化服务
- Vagrant支持Amazon AWS和Rackspace

# 推荐文章 | Article

# Java SE 8: Lambda表达式

作者 成富

Java SE 8在6月13的版本中已经完全了全部的功能。在这些新的功能中，lambda表达式是推动该版本发布的最重要新特性。因为Java第一次尝试引入函数式编程的相关内容。社区对于lambda表达式也期待已久。Lambda表达式的相关内容在JSR 335中定义，本文的内容基于最新的规范和JDK 8 Build b94。 开发环境使用的是Eclipse。

## Lambda表达式

要理解lambda表达式，首先要了解的是函数式接口（functional interface）。简单来说，函数式接口是只包含一个抽象方法的接口。比如Java标准库中的java.lang.Runnable和java.util.Comparator都是典型的函数式接口。对于函数式接口，除了可以使用Java中标准的方法来创建实现对象之外，还可以使用lambda表达式来创建实现对象。这可以在很大程度上简化代码的实现。在使用lambda表达式时，只需要提供形式参数和方法体。由于函数式接口只有一个抽象方法，所以通过lambda表达式声明的方法体就肯定是这个唯一的抽象方法的实现，而且形式参数的类型可以根据方法的类型声明进行自动推断。

以Runnable接口为例来进行说明，传统的创建一个线程并运行的方式如下所示：

```
public void runThread() {
    new Thread(new Runnable() {
        public void run() {
            System.out.println("Run!");
        }
    }).start();
}
```

在上面的代码中，首先需要创建一个匿名内部类实现Runnable接口，还需要实现接口中的run方法。如果使用lambda表达式来完成同样的功能，得到的代码非常简洁，如下面所示：

```
public void runThreadUseLambda() {
    new Thread(() -> {
        System.out.println("Run!");
    }).start();
}
```

相对于传统的方式，lambda表达式在两个方面进行了简化：首先是Runnable接口的声明，这可以通过对上下文环境进行推断来得出；其次是对run方法的实现，因为函数式接口中只包含一个需要实现的方法。

Lambda表达式的声明方式比较简单，由形式参数和方法体两部分组成，中间通过“->”分隔。形式参数不需要包含类型声明，可以进行自动推断。当然在某些情况下，形式参数的类型声明是不可少的。方法体则可以是简单的表达式或代码块。

比如把一个整数列表按照降序排列可以用下面的代码来简洁实现：

```
Collections.sort(list, (x, y) -> y - x);
```

Lambda表达式“(x, y) -> y - x“实现了java.util.Comparator接口。

在Java SE 8之前的标准库中包含的函数式接口并不多。Java SE 8增加了java.util.function包，里面都是可以在开发中使用的函数式接口。开发人员也可以创建新的函数式接口。最好在接口上使用注解@FunctionalInterface进行声明，以免团队的其他人员错误地往接口中添加新的方法。

下面的代码使用函数式接口java.util.function.Function实现的对列表进行map操作的方法。从代码中可以看到，如果尽可能的使用函数式接口，则代码使用起来会非常简洁。

```
public class CollectionUtils {
```

```
    public static  List map(List input, Function processor)
{
        ArrayList result = new ArrayList();
        for (T obj : input) {
            result.add(processor.apply(obj));
        }
        return result;
    }

    public static void main(String[] args) {
        List input = Arrays.asList(new String[] {"apple"
, "orange", "pear"});
        List lengths = CollectionUtils.map(input, (String
 v) -> v.length());
        List uppercases = CollectionUtils.map(input, (Str
ing v) -> v.toUpperCase());
    }
}
```

## 方法和构造方法引用

方法引用可以在不调用某个方法的情况下引用一个方法。构造方法引用可以在不创建对象的情况下引用一个构造方法。方法引用是另外一种实现函数式接口的方法。在某些情况下，方法引用可以进一步简化代码。比如下面的代码中，第一个forEach方法调用使用的是lambda表达式，第二个使用的是方法引用。两者作用相同，不过使用方法引用的做法更加简洁。

```
List input = Arrays.asList(new String[] {"apple", "orang
e", "pear"});
input.forEach((v) -> System.out.println(v));
input.forEach(System.out::println);
```

构造方法可以通过名称“new”来进行引用，如下面的代码所示：

```
List dateValues = Arrays.asList(new Long[] {0L, 1000L})
;
List dates = CollectionUtils.map(dateValues, Date::new
);
```

# 接口的默认方法

Java开发中所推荐的实践是面向接口而不是实现来编程。接口作为不同组件之间的契约，使得接口的实现可以不断地演化。不过接口本身的演化则比较困难。当接口发生变化时，该接口的所有实现类都需要做出相应的修改。如果在新版本中对接口进行了修改，会导致早期版本的代码无法运行。Java对于接口更新的限制过于严格。在代码演化的过程中，一般所遵循的原则是不删除或修改已有的功能，而是添加新的功能作为替代。已有代码可以继续使用原有的功能，而新的代码则可以使用新的功能。但是这种更新方式对于接口是不适用的，因为往一个接口中添加新的方法也会导致已有代码无法运行。

接口的默认方法的主要目标之一是解决接口的演化问题。当往一个接口中添加新的方法时，可以提供该方法的默认实现。对于已有的接口使用者来说，代码可以继续运行。新的代码则可以使用该方法，也可以覆写默认的实现。

考虑下面的一个简单的进行货币转换的接口。该接口的实现方式可能是调用第三方提供的服务来完成实际的转换操作。

```
public interface CurrencyConverter {
    BigDecimal convert(Currency from, Currency to, BigDe
cimal amount);
}
```

该接口在开发出来之后，在应用中得到了使用。在后续的版本更新中，第三方服务提供了新的批量处理的功能，允许在一次请求中同时转换多个数值。最直接的做法是在原有的接口中添加一个新的方法来支持批量处理，不过这样会造成已有的代码无法运行。而默认方法则可以很好的解决这个问题。使用默认方法的新接口如下所示。

```
public interface CurrencyConverter {
    BigDecimal convert(Currency from, Currency to, BigD
```

```
ecimal amount);

    default List convert(Currency from, Currency to, Lis
t amounts) {
        List result = new ArrayList();
            for (BigDecimal amount : amounts) {
                result.add(convert(from, to, amount));
            }
            return result;
    }
}
```

新添加的方法使用default关键词来修饰，并可以有自己的方法体。

默认方法的另外一个作用是实现行为的多继承。Java语言只允许类之间的单继承关系，但是一个类可以实现多个接口。在默认方法引入之后，接口中不仅可以包含变量和方法声明，还可以包含方法体，也就是行为。通过实现多个接口，一个Java类实际上可以获得来自不同接口的行为。这种功能类似于JavaScript等其他语言中可见的“混入类”（mixin）。实际上，Java中一直存在“常量接口（Constant Interface）”的用法。常量接口中只包含常量的声明。通过实现这样的接口，就可以直接引用这些常量。通过默认方法，可以创建出类似的帮助接口，即接口中包含的都是通过默认方法实现的帮助方法。比如创建一个StringUtils接口包含各种与字符串操作相关的默认方法。通过继承该接口就可以直接使用这些方法。

Java SE 8标准库已经使用默认方法来对集合类中的接口进行更新。比如java.util.Collection接口中新增的默认方法removeIf可以删除集合中满足某些条件的元素。还有java.lang.Iterable接口中新增的默认方法forEach可以遍历集合中的元素，并执行一些操作。这些新增的默认方法大多使用了java.util.function包中的函数式接口，因此可以使用lambda表达式来非常简洁的进行操作。

Lambda表达式是Java SE 8在提高开发人员生产效率上的一个重大改进。通过语法上的改进，可以减少开发人员需要编写和维护的代码数量。

**原文链接：http://www.infoq.com/cn/articles/Java-se-8-lambda**

**相关内容**

- Java SE 8: Lambda表达式
- Java ME的重塑：从非智能机到物联网

TURING

微信账号：
图灵教育

# 图灵教育推荐

书名：嗨翻 C 语言
作者：David Griffiths，Dawn Griffiths
译者：程亦超
书号：978-7-115-31884-8
定价：99.00 元

- Head First 系列经典畅销书
- 创意 C 语言　嗨翻你的大脑！

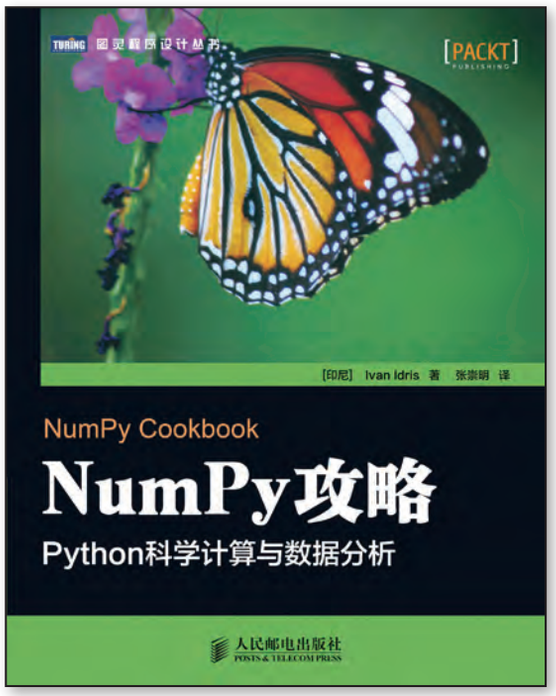
想用 C 代码编写好玩的游戏吗？想成为 C 代码侦探，去破解重重谜案吗？或是让 C 代码变成牵线红娘，帮你找到你的梦中情人？哈哈！别着急，David Griffiths 和 Dawn Griffiths 夫妇的这本《嗨翻 C 语言》将会带你来到这个奇妙的 C 语言世界：这里没有令人乏味的“学究腔”，没有冗长枯燥的老生常谈，也没有令人望而却步的大堆练习。作者用诙谐、幽默的口吻为你讲述了一个又一个关于 C 的故事。无论你是 C 语言学习航程中的新船员，还是经历过大风大浪的老水手，你都能从《嗨翻 C 语言》中体会到无限乐趣。

详情请点击：http://www.ituring.com.cn/book/1004

- 使用 Python 做科学计算所必须掌握的利器——NumPy ！

本书带领读者了解熟悉当下最流行的科学计算库 NumPy 的方方面面。书中不仅介绍了 NumPy 的安装、使用和各种相关概念，还介绍了如何利用这一最新的开源软件库，以尽可能接近传统数学语言的方式，编写可读性好、实现效率高和运行速度快的代码。最后还探究了几个和 NumPy 相关的科学计算项目。此外，本书将为你掌握 NumPy 数组和通用函数打下坚实的基础，也会通过实例教你用 Matplotlib 绘图，并了解和 SciPy 相关的项目。

详情请点击：http://www.ituring.com.cn/book/1183




书名：NumPy 攻略：Python 科学计算与数据分析
作者：Ivan Idris
译者：张崇明
书号：978-7-115-32991-2
定价：49.00 元

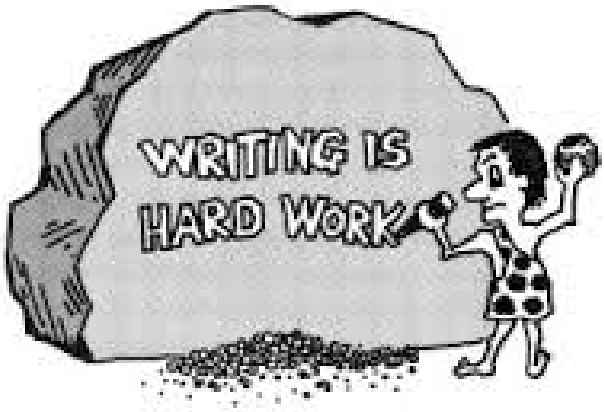
TURING 北京图灵文化发展有限公司

联系我们：
官方微博：@图灵教育
联 系 人：@图灵郭志敏
邮　　箱：guozm@turingbook.com
Q　　Q：52398720

# 推荐语

## 专栏主编有话说

技术改变未来，技术人的特点是闷骚，很抱歉我用这样的话来形容技术人，但是实际的场景可能也是这样的。但是技术人的闷骚 代表着他们的满足点真到的单纯，写出了一个新的类库、算法得到了提升亦或是新的架构在系统中的表现很好，这些都让他们不亦乐乎。同时，技术人乐于分享， 乐于把自己的思考奉献出去，这和InfoQ的Slogan：“促进软件开发领域只是与创新的传播”如出一辙，回顾InfoQ的内容历程，我们发现借助这些技术人的力量， InfoQ也一直努力为大家带来精彩纷呈的内容，也因此我们也希望有越来越多的朋友能够加入到这个行列中。在这几期的专栏中我们将会介绍目前的专栏和内容规划， 如果在这些领域有话想说，不要犹豫，给我们发邮件（editors@cn.infoq.com）来，参与到这个知识传播的过程中吧！



# 特别专栏 | Column

# 主编有话说：云计算、SOA专栏主编马国耀谈专栏建设

作者 马国耀

本人马国耀，是InfoQ中文站负责云计算和SOA相关方面的内容主编。之所以负责这两块内容，一方面，本人对这两个领域感兴趣；另一方面；这两个领域也是我自己熟悉和工作的领域。而促使我成为内容主编的主要原因是：我乐于从分享和服务读者的过程中提升自身，同时我也能从分享中得到快乐。工作近10年以来，我涉足SOA领域和云计算领域分别有6年和3年。现阶段，我主要工作的方向包括：在国内市场推广SOA解决方案和产品；设计云服务整合交付平台（Cloud Service Integration and Delivery Platform），属于PaaS层解决方案。我曾经翻译出版过《云计算与SOA》，而《SOA与REST》的翻译也已经交稿，即将出版。

回到这个专栏的两大领域：云计算和SOA。其中，SOA已经深入人心，从InfoQ的中文及英文社区来看，当前已经很少有人在博客对其进行评头论足了。在这一领域，当下值得关注的内容是实际经验的分享，我们非常欢迎通过SOA的架构思想（风格）建设企业级解决方案的例子，成功的案例、失败的教训都是颇受欢迎的。与SOA相比，云计算则并不那么成熟。同时，云计算领域涵盖的范围又极为广泛：从云服务的类型看，有IaaS/PaaS/SaaS；从部署模型看，有私有云、公有云、混合云；从其技术解决方案栈看，有虚拟化、存储、云操作系统、数据中心管理、云管理等；开源解决方案和项目亦是如雨后春笋般初露锋芒；不论哪个领域，几乎每天都有创新，每天都有新的论战，热闹无比。就InfoQ中文站而言，我们特别关注的方面是：通用云计算解决方案及成功经验、云计算厂商在特点领域里面做出的创新和成绩、企业在云计算采纳方面做出的尝试以及收获的经验和教训、技术领域（安全、控制、管理、治理等）的突破、商业模式的更新等等。

作为相关专栏内容负责人，从理论上来说，专栏内的所有原创内容都会经过我的审阅。这项工作对我提出了很大的挑战，因个人精力和能力有限，我无法深入了解该领域的方方面面。但是，一方面，我把握一个原则，大多数创新都是在巨人的肩膀上完成的，从思想和哲学的层面，许多技术都有相通之处。另一方面，我站在读者角度来审阅文章，把握文章对读者带来的价值和读者的阅读

体验。所以，只要是对我们广大读者有其独特价值的内容，即便存在一些厂商营销和个人品牌营销的成分，也是能通过我的审阅的。今年，我审了许多文章，被我毙掉的文章一类是内容老调重弹，毫无新意可言，另一类是厂商的纯软文；但大多数情况下我都会与作者进行深入沟通，提出修改建议。愿意分享的人并不多，既愿意分享又能把语言组织好的作者就更少了，所以我特别珍惜与乐于分享的作者交流的机会。

在这里，我想与各位读者分享一点自己的体会，“分享”是既能提升自己又能帮助别人的过程，当你把脑中的思想和知识组织成别人能够读懂的文字时，这一过程本身就是对知识和思想的一次提炼和深化，写成了发表在网站之后被读者阅读和评论时，成就感也会油然而生。好文章能够传道解惑、也能启发读者，还能增加作者的知名度，结交许多朋友，何乐而不为？如果有读者动心了，但担心写不出符合InfoQ要求的深度内容，那么请不要担心，我将与您深入交流撰写InfoQ深度内容的方法和心得。

好了，如果你愿意分享自己的创新、经验、实战案例和教训，就不要犹豫了。请与我们联系，当然也可以直接与我联系maguoyao@gmail.com。

近期由我负责的专栏和文章推荐如下：

- 专栏：深入探索REST

  通过一系列深入内容探讨REST的方方面面，希望借此帮助读者全面而深入地认识REST。

- 云计算时代的运维与安全

  云计算时代给运维和安全带来了新的机遇与挑战，听一听中国运维和安全大会上专家们的声音，相信对你有所启发。

- Windows Azure案例分析: 选择虚拟机或云服务？

  如何在Windows Azure提供的三种计算模式（虚拟机、云服务和网站）中做出选择，更好地解决你的问题？

- “林仕鼎谈数据中心计算”系列文章：

  - 林仕鼎谈数据中心计算（一）：整体大于部分之和
  - 林仕鼎谈数据中心计算（二）：存储资源分层
  - 林仕鼎谈数据中心计算（三）：学术界与工业界，以及未来展望

通过对百度基础架构部首席架构师林仕鼎先生的系列访谈，学习他对建设云计算数据中心算的见解与观点、设计原则、对未来的展望等。

# 特别专栏 | Column

# 理解本真的REST架构风格

作者 李锟

本文是“深入探索REST”专栏系列深度内容中的第二篇，它将带您领略REST架构的起源、与Web的关系、REST架构的本质及特性，以及REST架构与其他架构风格之间的比较。

## 引子

在移动互联网、云计算迅猛发展的今天，作为一名Web开发者，如果您还没听说过“REST”这个buzzword，显然已经落伍了。夸张点说，甚至“出了门都不好意思跟别人打招呼”。尽管如此，对于REST这个泊来品的理解，大多数人（包括一些资深的架构师）仍然停留在“盲人摸象”的阶段。常常听到各种各样关于REST的说法，例如：有人说：“我们这套新的API决定不用Web Service（SOAP+WSDL），而是直接使用HTTP+JSON，也就是用RESTful的方式来开发。” 不用SOAP，甚至也不用XML，就自动变成了RESTful了。还有人认为：REST与传统的Web Service其实没有本质区别，只是对于URI的构造方式提出了更多要求，而这些要求Web Service完全都可以实现。潜台词是：既生瑜，何生亮。Web Service已经足够好了，干嘛还要再折腾什么REST。这些对于REST的不同说法，果真如此吗？REST究竟是什么？是一种新的技术、一种新的架构、还是一种新的规范？

对于这些问题笔者先不解答，为了深入理解REST是什么，我们需要回顾一下Web发展的最初年代，从源头上讲讲REST是怎么得来的。

## Web技术发展与REST的由来

Web（万维网World Wide Web的简称）是个包罗万象的万花筒，不同的人从不同的角度观察，对于Web究竟是什么会得出大不相同的观点。作为Web开发者，我们需要从技术上来理解Web。从技术架构层面上看，Web的技术架构包括了四个基石：

- URI

- HTTP
- HyperText（除了HTML外，也可以是带有超链接的XML或JSON）
- MIME

这四个基石相互支撑，促使Web这座宏伟的大厦以几何级数的速度发展了起来。在这四个基石之上，Web开发技术的发展可以粗略划分成以下几个阶段：

1. 静态内容阶段：在这个最初的阶段，使用Web的主要是一些研究机构。Web由大量的静态HTML文档组成，其中大多是一些学术论文。Web服务器可以被看作是支持超文本的共享文件服务器。
2. CGI程序阶段：在这个阶段，Web服务器增加了一些编程API。通过这些API编写的应用程序，可以向客户端提供一些动态变化的内容。Web服务器与应用程序之间的通信，通过CGI（Common Gateway Interface）协议完成，应用程序被称作CGI程序。
3. 脚本语言阶段：在这个阶段，服务器端出现了ASP、PHP、JSP、ColdFusion等支持session的脚本语言技术，浏览器端出现了Java Applet、JavaScript等技术。使用这些技术，可以提供更加丰富的动态内容。
4. 瘦客户端应用阶段：在这个阶段，在服务器端出现了独立于Web服务器的应用服务器。同时出现了Web MVC开发模式，各种Web MVC开发框架逐渐流行，并且占据了统治地位。基于这些框架开发的Web应用，通常都是瘦客户端应用，因为它们是在服务器端生成全部的动态内容。
5. RIA应用阶段：在这个阶段，出现了多种RIA（Rich Internet Application）技术，大幅改善了Web应用的用户体验。应用最为广泛的RIA技术是DHTML+Ajax。Ajax技术支持在不刷新页面的情况下动态更新页面中的局部内容。同时诞生了大量的Web前端DHTML开发库，例如Prototype、Dojo、ExtJS、jQuery/jQuery UI等等，很多开发库都支持单页面应用（Single Page Application）的开发。其他的RIA技术还有Adobe公司的Flex、微软公司的Silverlight、Sun公司的JavaFX（现在为Oracle公司所有）等等。
6. 移动Web应用阶段：在这个阶段，出现了大量面向移动设备的Web应用开发技术。除了Android、iOS、Windows Phone等操作系统平台原生的开发技术之外，基于HTML5的开发技术也变得非常流行。

从上述Web开发技术的发展过程看，Web从最初其设计者所构思的主要支持静态文档的阶段，逐渐变得越来越动态化。Web应用的交互模式，变得越来越复杂：从静态文档发展到以内容为主的门户网站、电子商务网站、搜索引擎、社交网站，再到以娱乐为主的大型多人在线游戏、手机游戏。

在互联网行业，实践总是走在理论的前面。Web发展到了1995年，在CGI、ASP等技术出现之后，沿用了多年、主要面向静态文档的HTTP/1.0协议已经无法满足Web应用的开发需求，因此需要设计新版本的HTTP协议。在HTTP/1.0协议专家组之中，有一位年轻人脱颖而出，显示出了不凡的洞察力，后来他成为了HTTP/1.1协议专家组的负责人。这位年轻人就是Apache HTTP服务器的核心开发者Roy Fielding，他还是Apache软件基金会的合作创始人。

Roy Fielding和他的同事们在HTTP/1.1协议的设计工作中，对于Web之所以取得巨大成功，在技术架构方面的因素做了一番深入的总结。Fielding将这些总结纳入到了一套理论框架之中，然后使用这套理论框架中的指导原则，来指导HTTP/1.1协议的设计方向。HTTP/1.1协议的第一个草稿是在1996年1月发布的，经过了三年多时间的修订，于1999年6月成为了IETF的正式规范（包括了RFC 2616以及用于对客户端做身份认证的RFC 2617）。HTTP/1.1协议设计的极为成功，以至于发布之后整整10年时间里，都没有多少人认为有修订的必要。用来指导HTTP/1.1协议设计的这套理论框架，最初是以备忘录的形式在专家组成员之间交流，除了IETF/W3C的专家圈子，并没有在外界广泛流传。Fielding在完成HTTP/1.1协议的设计工作之后，回到了加州大学欧文分校继续攻读自己的博士学位。第二年（2000年）在他的博士学位论文Architectural Styles and the Design of Network-based Software Architectures中，Fielding更为系统、严谨地阐述了这套理论框架，并且使用这套理论框架推导出了一种新的架构风格，并且为这种架构风格取了一个令人轻松愉快的名字“REST”——Representational State Transfer（表述性状态转移）的缩写。

在笔者看来，Fielding这篇博士论文在Web发展史上的价值，不亚于Web之父Tim Berners-Lee关于超文本的那篇经典论文。然而遗憾的是，这篇博士论文在诞生之后的将近5年时间里，一直没有得到足够的重视。例如Web Service相关规范SOAP/WSDL的设计者们，显然不大理解REST是什么，HTTP/1.1究竟是一个什么样的协议、为何要设计成这个样子。

这种情况在2005年之后有了很大的改善，随着Ajax、Ruby on Rails等新的Web开发技术的兴起，在Web开发技术社区掀起了一场重归Web架构设计本源的运动，REST架构风格得到了越来越多的关注。在2007年1月，支持REST开发的Ruby on Rails 1.2版正式发布，并且将支持REST开发作为Rails未来发展中的优先内容。Ruby on Rails的创始人DHH做了一个名为“World of Resources”的精彩演讲，DHH在Web开发技术社区中的强大影响力，使得REST一下子处在Web开发技术舞台的聚光灯之下。

今天，各种流行的Web开发框架，几乎没有不支持REST开发的了。大多数Web开发者都是通过阅读某种REST开发框架的文档，以及通过一些例子代码来学习REST开发的。然而，通过例子代码来学习REST有非常大的局限性。因为

REST并不是一种具体的技术，也不是一种具体的规范，REST其实是一种内涵非常丰富的架构风格。通过例子代码来学习REST，除了学习到一种有趣的Web开发技术之外，并不能全面深入的理解REST究竟是什么。甚至还会误以为这些简单的例子代码就是REST本身，REST不过是一种简单的Web开发技术而已。就像盲人摸象一样，有的人摸到了象鼻子、有的人摸到了象耳朵、有的人摸到了象腿、有的人摸到了象尾巴。他们都坚信自己感觉到的大象，才是最真实的大象，而其他人的感觉都是错误的。

对于不理解REST的Web开发者，人们习惯于展示一些例子代码来让他们理解REST，笔者不赞同上述做法。如果Web开发者想要深入理解REST是什么，就很难避开Fielding的这篇博士论文。笔者在本文中对于REST是什么的介绍，也是基于Fielding的博士论文的。尽管如此，笔者强烈建议本文的读者亲自去通读一下Fielding的博士论文，就像想要了解孔子的思想应该直接去读《论语》等著作，而不是首先去读其他人的转述一样。笔者在本文中也仅仅是努力不做一个把经书念错了的歪嘴和尚而已。那么，下面我们言归正传。

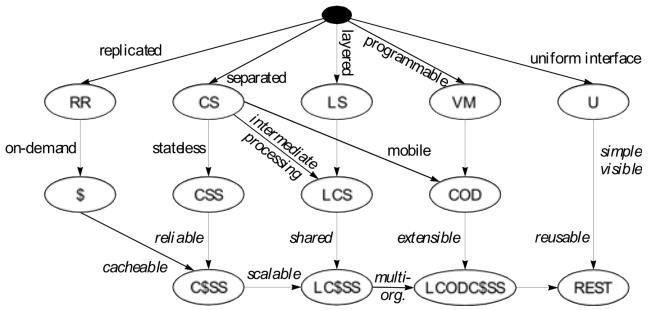
在Fielding的这篇名为Architectural Styles and the Design of Network-based Software Architectures的博士论文（中文版名为《架构风格与基于网络的软件架构设计》）中，提出了一整套基于网络的软件（即所谓的“分布式应用”）的设计方法，值得所有分布式应用的开发者仔细阅读、深入体会。

在论文的前三章中，Fielding在批判性继承前人研究成果的基础上，建立起来一整套研究和评价软件架构的方法论。这套方法论的核心是“架构风格”这个概念。架构风格是一种研究和评价软件架构设计的方法，它是比架构更加抽象的概念。一种架构风格是由一组相互协作的架构约束来定义的。架构约束是指软件的运行环境施加在架构设计之上的约束。

在论文的第四章中，Fielding研究了Web这样一个分布式系统对于软件架构设计提出了哪些需求。在第五章中，Fielding将第四章Web提出的需求具体化为一些架构约束，通过逐步添加各种架构约束，推导出来了REST这种新的架构风格。

REST架构风格的推导过程如下图所示：

图1：REST所继承的架构风格约束（原图可在这里下载）



在图1中，每一个椭圆形里面的缩写词代表了一种架构风格，而每一个箭头边的单词代表了一种架构约束。

REST架构风格最重要的架构约束有6个：

- 客户-服务器（Client-Server）

通信只能由客户端单方面发起，表现为请求-响应的形式。

- 无状态（Stateless）

通信的会话状态（Session State）应该全部由客户端负责维护。

- 缓存（Cache）

响应内容可以在通信链的某处被缓存，以改善网络效率。

- 统一接口（Uniform Interface）

通信链的组件之间通过统一的接口相互通信，以提高交互的可见性。

- 分层系统（Layered System）

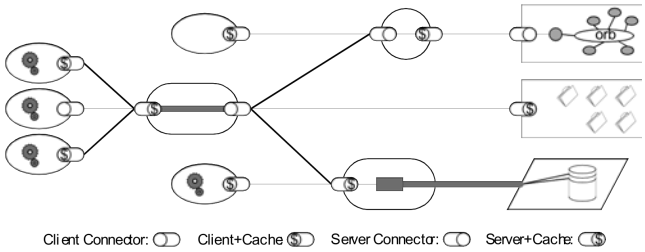
通过限制组件的行为（即，每个组件只能“看到”与其交互的紧邻层），将架构分解为若干等级的层。

- 按需代码（Code-On-Demand，可选）

支持通过下载并执行一些代码（例如Java Applet、Flash或JavaScript），对客户端的功能进行扩展。

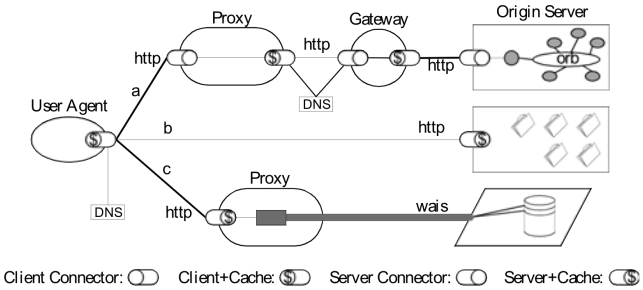
在论文中推导出的REST架构风格如下图所示：

图2：REST架构风格（原图可在这里下载）




而HTTP/1.1协议作为一种REST架构风格的架构实例，其架构如下图所示：

图3：一个基于REST的架构的过程视图（原图可在这里下载）




用户代理处在三个并行交互（a、b和c）的中间。用户代理的客户端连接器缓存无法满足请求，因此它根据每个资源标识符的属性和客户端连接器的配置，将每个请求路由到资源的来源。请求（a）被发送到一个本地代理，代理随后访问一个通过DNS查找发现的缓存网关，该网关将这个请求转发到一个能够满足该请求的来源服务器，服务器的内部资源由一个封装过的对象请求代理（object request broker）架构来定义。请求（b）直接发送到一个来源服务器，它能够通过自己的缓存来满足这个请求。请求（c）被发送到一个代理，它能够直接访问WAIS（一种与Web架构分离的信息服务），并将WAIS的响应翻译为一种通用的连接器接口能够识别的格式。每一个组件只知道与它们自己的客户端或服务器连接器的交互；整个过程拓扑是我们的视图的产物。

通过比较图2和图3，读者不难发现这两张图中的架构是高度一致的。对于HTTP/1.1协议为何要设计成这个样子，读者想必已经有所领悟。

在论文的第六章中，Fielding对于到2000年为止在Web基础架构协议的设计和开发方面的一些经验教训进行了深入的分析。其中，“HTTP不是RPC”、“HTTP不是一种传输协议”两部分值得读者反复阅读。时至13年之后的今日，对于HTTP协议的误解仍然广泛存在。

以上简要介绍了Fielding博士论文中的内容。为了帮助读者仔细阅读Fielding的博士论文，笔者整理了一套Fielding博士论文的导读，将在本专栏后续文章中载出。

# REST详解

REST究竟是什么？因为REST的内涵非常丰富，所以很难用一两句话解释清楚这个问题。

首先，REST是Web自身的架构风格。REST也是Web之所以取得成功的技术架构方面因素的总结。REST是世界上最成功的分布式应用架构风格（成功案例：Web，还不够吗？）。它是为 运行在互联网环境 的 分布式 超媒体系统量身定制的。互联网环境与企业内网环境有非常大的差别，最主要的差别是两个方面：

- 可伸缩性需求无法控制：并发访问量可能会暴涨，也可能会暴跌。

- 安全性需求无法控制：无法控制客户端发来的请求的格式，很可能会是恶意的请求。

而所谓的“超媒体系统”，即，使用了超文本的系统。可以把“超媒体”理解为超文本+媒体内容。

REST是HTTP/1.1协议等Web规范的设计指导原则，HTTP/1.1协议正是为实现REST风格的架构而设计的。新的Web规范，其设计必须符合REST的要求，否则整个Web的体系架构会因为引入严重矛盾而崩溃。这句话不是危言耸听，做个类比，假如苏州市政府同意在市区著名园林的附近大型土木，建造大量具有后现代风格的摩天大楼，那么不久之后世界闻名的苏州园林美景将不复存在。

上述这些关于“REST是什么”的描述，可以总结为一句话：REST是所有Web应用都应该遵守的架构设计指导原则。当然，REST并不是法律，违反了REST的指导原则，仍然能够实现应用的功能。但是违反了REST的指导原则，会付出很多代价，特别是对于大流量的网站而言。

要深入理解REST，需要理解REST的五个关键词：

1. 资源（Resource）
2. 资源的表述（Representation）
3. 状态转移（State Transfer）
4. 统一接口（Uniform Interface）
5. 超文本驱动（Hypertext Driven）

什么是资源？

资源是一种看待服务器的方式，即，将服务器看作是由很多离散的资源组成。每个资源是服务器上一个可命名的抽象概念。因为资源是一个抽象的概念，所以它不仅仅能代表服务器文件系统中的一个文件、数据库中的一张表等等具体的东西，可以将资源设计的要多抽象有多抽象，只要想象力允许而且客户端应用开发者能够理解。与面向对象设计类似，资源是以名词为核心来组织的，首先关注的是名词。一个资源可以由一个或多个URI来标识。URI既是资源的名称，也是资源在Web上的地址。对某个资源感兴趣的客户端应用，可以通过资源的URI与其进行交互。

什么是资源的表述？

资源的表述是一段对于资源在某个特定时刻的状态的描述。可以在客户端-服务器端之间转移（交换）。资源的表述可以有多种格式，例如HTML/XML/JSON/纯文本/图片/视频/音频等等。资源的表述格式可以通过协商机制来确定。请求-响应方向的表述通常使用不同的格式。

什么是状态转移？

状态转移（state transfer）与状态机中的状态迁移（state transition）的含义是不同的。状态转移说的是：在客户端和服务器端之间转移（transfer）代表资源状态的表述。通过转移和操作资源的表述，来间接实现操作资源的目的。

什么是统一接口？

REST要求，必须通过统一的接口来对资源执行各种操作。对于每个资源只能执行一组有限的操作。以HTTP/1.1协议为例，HTTP/1.1协议定义了一个操作资源的统一接口，主要包括以下内容：

- 7个HTTP方法：GET/POST/PUT/DELETE/PATCH/HEAD/OPTIONS
- HTTP头信息（可自定义）
- HTTP响应状态代码（可自定义）
- 一套标准的内容协商机制
- 一套标准的缓存机制
- 一套标准的客户端身份认证机制

REST还要求，对于资源执行的操作，其操作语义必须由HTTP消息体之前的部分完全表达，不能将操作语义封装在HTTP消息体内部。这样做是为了提高交互的可见性，以便于通信链的中间组件实现缓存、安全审计等等功能。

什么是超文本驱动？

“超文本驱动”又名“将超媒体作为应用状态的引擎”（Hypermedia As The Engine Of Application State，来自Fielding博士论文中的一句话，缩写为HATEOAS）。将Web应用看作是一个由很多状态（应用状态）组成的有限状态机。资源之间通过超链接相互关联，超链接既代表资源之间的关系，也代表可执行的状态迁移。在超媒体之中不仅仅包含数据，还包含了状态迁移的语义。以超媒体作为引擎，驱动Web应用的状态迁移。通过超媒体暴露出服务器所提供的资源，服务器提供了哪些资源是在运行时通过解析超媒体发现的，而不是事先定义的。从面向服务的角度看，超媒体定义了服务器所提供服务的协议。客户端应该依赖的是超媒体的状态迁移语义，而不应该对于是否存在某个URI或URI的某种特殊构造方式作出假设。一切都有可能变化，只有超媒体的状态迁移语义能够长期保持稳定。

一旦读者理解了上述REST的五个关键词，就很容易理解REST风格的架构所具

有的6个的主要特征：

- 面向资源（Resource Oriented）
- 可寻址（Addressability）
- 连通性（Connectedness）
- 无状态（Statelessness）
- 统一接口（Uniform Interface）
- 超文本驱动（Hypertext Driven）

这6个特征是REST架构设计优秀程度的判断标准。其中，面向资源是REST最明显的特征，即，REST架构设计是以资源抽象为核心展开的。可寻址说的是：每一个资源在Web之上都有自己的地址。连通性说的是：应该尽量避免设计孤立的资源，除了设计资源本身，还需要设计资源之间的关联关系，并且通过超链接将资源关联起来。无状态、统一接口是REST的两种架构约束，超文本驱动是REST的一个关键词，在前面都已经解释过，就不再赘述了。

从架构风格的抽象高度来看，常见的分布式应用架构风格有三种：

- 分布式对象（Distributed Objects，简称DO）

架构实例有CORBA/RMI/EJB/DCOM/.NET Remoting等等

- 远程过程调用（Remote Procedure Call，简称RPC）

架构实例有SOAP/XML-RPC/Hessian/Flash AMF/DWR等等

- 表述性状态转移（Representational State Transfer，简称REST）

架构实例有HTTP/WebDAV

DO和RPC这两种架构风格在企业应用中非常普遍，而REST则是Web应用的架构风格，它们之间有非常大的差别。

REST与DO的差别在于：

- REST支持抽象（即建模）的工具是资源，DO支持抽象的工具是对象。在不同的编程语言中，对象的定义有很大差别，所以DO风格的架构通常都是与某种编程语言绑定的。跨语言交互即使能实现，实现起来也会非常复杂。而REST中的资源，则完全中立于开发平台和编程语言，可以使用任何编程语

言来实现。

- DO中没有统一接口的概念。不同的API，接口设计风格可以完全不同。DO也不支持操作语义对于中间组件的可见性。
- DO中没有使用超文本，响应的内容中只包含对象本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比DO更高。
- REST支持数据流和管道，DO不支持数据流和管道。
- DO风格通常会带来客户端与服务器端的紧耦合。在三种架构风格之中，DO风格的耦合度是最大的，而REST的风格耦合度是最小的。REST松耦合的源泉来自于统一接口+超文本驱动。

REST与RPC的差别在于：

- REST支持抽象的工具是资源，RPC支持抽象的工具是过程。REST风格的架构建模是以名词为核心的，RPC风格的架构建模是以动词为核心的。简单类比一下，REST是面向对象编程，RPC则是面向过程编程。
- RPC中没有统一接口的概念。不同的API，接口设计风格可以完全不同。RPC也不支持操作语义对于中间组件的可见性。
- RPC中没有使用超文本，响应的内容中只包含消息本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比RPC更高。
- REST支持数据流和管道，RPC不支持数据流和管道。
- 因为使用了平台中立的消息，RPC风格的耦合度比DO风格要小一些，但是RPC风格也常常会带来客户端与服务器端的紧耦合。支持统一接口+超文本驱动的REST风格，可以达到最小的耦合度。

比较了三种架构风格之间的差别之后，从面向实用的角度来看，REST架构风格可以为Web开发者带来三方面的利益：

- 简单性

采用REST架构风格，对于开发、测试、运维人员来说，都会更简单。可以充分利用大量HTTP服务器端和客户端开发库、Web功能测试/性能测试工具、HTTP缓存、HTTP代理服务器、防火墙。这些开发库和基础设施早已成为了日常用品，不需要什么火箭科技（例如神奇昂贵的应用服务器、中间件）就能解决大多数可伸缩性方面的问题。

- 可伸缩性

充分利用好通信链各个位置的HTTP缓存组件，可以带来更好的可伸缩性。其实很多时候，在Web前端做性能优化，产生的效果不亚于仅仅在服务器端做性能优化，但是HTTP协议层面的缓存常常被一些资深的架构师完全忽略掉。

- 松耦合

统一接口+超文本驱动，带来了最大限度的松耦合。允许服务器端和客户端程序在很大范围内，相对独立地进化。对于设计面向企业内网的API来说，松耦合并不是一个很重要的设计关注点。但是对于设计面向互联网的API来说，松耦合变成了一个必选项，不仅在设计时应该关注，而且应该放在最优先位置。

有的读者可能会问："你说了这么多，REST难道就没有任何缺点了吗？"当然不是，正如Fielding在博士论文中阐述的那样，评价一种软件架构的优劣，不能脱离开软件的具体运行环境。永远不存在适用于任何运行环境的、包治百病的银弹式架构。笔者在前面强调过REST是一种为运行在互联网环境中的Web应用量身定制的架构风格。REST在互联网这个运行环境之中已经占据了统治地位，然而，在企业内网运行环境之中，REST还会面临DO、RPC的巨大挑战。特别是一些对实时性要求很高的应用，REST的表现不如DO和RPC。所以需要针对具体的运行环境来具体问题具体分析。但是，REST可以带来的上述三方面的利益即使在开发企业应用时，仍然是非常有价值的。所以REST在企业应用开发，特别是在SOA架构的开发中，已经得到了越来越大的重视。本专栏将有一篇文章专门介绍REST在企业级应用中与SOA的结合。

到了这里，"REST究竟是什么"这个问题笔者就解答完了。本文开头那些说法是否正确，笔者还是笑而不语，读者此时应该已经有了自己的判断。在接下来的REST系列文章中，我将会为读者澄清一些关于HTTP协议和REST的常见误解。

参考资料：

Roy Fielding博士论文英文版

Roy Fielding博士论文中文版

HTTP/1.1协议RFC2616、RFC2617

**原文链接：http://www.infoq.com/cn/articles/understanding-restful-style**

**相关内容**

- REST会是SOA的未来吗？

# 特别专栏 | Column

# 云计算时代的运维与安全

作者 丁雪丰

云计算时代给大家带了很多机遇，同时也带来了很多挑战，有人就认为随着云的普及，运维人员将会最终消失。当然，这个论点不免有些偏激，但云时代的确给运维带来了很多不同，也让运维从业人员开始思考很多问题。在近日举办的中国运维和安全大会上，我们就欣喜地看到了很多乐意迎接挑战的同学，也有很多大牛分享了自己的经验与心得。

中国的第一代黑客，现任UCloud CEO的季昕华为大家分析了云计算时代为运维与安全带来的挑战和机会。首先，运维人员要有一些基本的素质要求，其中包括懂风水，在机房选址时是否处于地震带，吹的什么风向，当地电价如何都是运维要考虑的；懂网络，在国内特殊的网络环境下，要理解南北差异；要有体力，必要时能去机房搬服务器；还要懂操作系统，懂网络攻击防御等等……

可是大多数运维人员在公司中的地位不高，而且在行业中的薪资相对偏低，究其原因还是因为运维的从业门槛低，大家对运维的认知度不高。因此，季昕华认为，除了上述基本知识，运维人员还因具备以下三方面的素质：

1. **懂业务** ，例如要能理解产品的用户是一线城市还是二线城市，是PC端还是移动端，在对业务有足够的了解的情况下，才能让你的工作成为领导关心的事。
2. **运营化** ，将运维中的意外管理变为过程管理，并能持续改进、持续优化；运维要能做到四个“第一” ，即第一时间发现问题，第一时间定位问题，第一时间解决问题和第一时间反馈问题。
3. **系统化** ，要能通过各种系统来辅助运维工作，甚至要能自己开发运维系统。

目前摆在大家面前有几个瓶颈，第一是成长空间有限，在公司的地位不高，行业内的知名度也不高；第二是云计算可能会革掉很多运维人员的名，很多小的初创企业甚至都不需要运维；第三是人员转型困难大。

当然，机会也有不少，比如，互联网正在快速地改变传统行业，之前兴起的O2O浪潮就是很好的例子，运维人员可以帮助那些传统行业快速地成长；大数

据的到来也为大家打开了一扇窗户；另外就是云计算，当你能把一个行业做精做细，就能把它挖掘成一个产业，例如又拍云、DNSPod、监控宝和安全宝都是最好的例子。

季昕华建议大家在使用那些免费的运维服务时，如果可以，就更多地向他们付费，让公司知道运维也是有价值的。当台下有开发的同学问到该如何帮助运维同学时，几位嘉宾都讲到了如果能够做到DevOps那是最好的，不要再出现这样的情况：

> 产品不足，开发补，开发不足运维补，运维不足客服补

既然云是本次大会的一个重要主题，那自然少不了云存储的内容。来自七牛的韩拓为大家介绍了七牛在建设云存储方面的一些做法，他的分享分为两部分——底层存储和构建于前者之上的云存储，两者在设计上有着截然不同的地方。

底层存储有以下难点：

- 元数据管理
- 对冗余度的控制（副本的数量与成本的平衡点）
- 修复速度（直接影响存储系统的可靠性，在七牛恢复是集群任务，盘上数据的副本松散地保存在集群中，目前能做到在十几分钟到几十分钟内修复2到3T的数据）
- 应对容量的增长
- 可接受的访问速度
- 合理、有效的缓存

七牛在网络上采用了常规的千兆局域网，这是考虑到了它的成熟度和成本，在机柜之间无法保证任意两点间随时都是千兆，甚至无法保证全联通，而机房之间的速度，带宽成本很高，速度与连通性都无法保证。因此，数据存储的位置需要有一定的平衡，副本在同一机柜和不同机柜各有利弊，机房亦是如此。

**在故障方面，除了要将故障视为常态，更要能明确地知道要面对哪些故障，它们的成因、概率和影响范围。**

例如，常见的故障有：

- 机房内故障
  - 网卡（断线、降速）
  - 网线（断线、降速）

  - 交换机（整体故障、单口故障、VLAN故障）
  - 机柜级联故障
- 机房间故障
  - 区域性网络故障（机房出口断网）
  - DNS解析故障（服务器之间DNS）

对于机房内的故障，不需要投入太多的资源成本做额外的高可用方案。

在网络安全上，除了必要的基础防御之外，更重要的是业务层面的防护，公有云的基本原则是开放，任何服务可以无条件暴露于公网，机房间的交互与客户无差别，不组VPN。

云存储构建于基础存储之上，它要能提供极高的上传、下载速度，有极高的可用性，有极高的可靠性，有丰富的附加功能（缩略图、水印等等），方便的网络访问。

它的难点在于：

- 云存储属于终端网络，它直接面对用户，情况复杂；它是最外层的接入点，前端没有机会做遮挡，对各种指标要求高。
- 广域网基础设施普遍质量不高，要基于99%可用的基础设施来提供99.999%的服务。

提到基础设施，机房的网络是个大问题，网络延时可以从几毫秒大到几千毫秒，吞吐速度从几十Mbps到几Kbps，而且带宽平均成本也不便宜。机房的可用性并不理想，经常会有链路故障，甚至是大面积、区域性掉线、降速，不仅机房间有问题，机房内也会频繁故障，小城市、小运营商用户会有个例无法访问的现象（七牛为用户提供了下载SDK，在APP和Web上连接到本区域节点下载不到内容时，可通过SDK连接备用域名和IP）。

七牛对数据进行了跨机房冗余，除了可靠性，更多地是为了可用性考虑；数据同步采用了分级异步同步的策略，最热的数据秒级异步同步，而冷数据则会批量同步；成本方面，冗余度的提升并未造成线性的成本提升，同时，异步同步还能智能地利用昂贵的带宽资源。

提供云存储的又拍云，为大家带来了与CDN与DDoS防御方面的一些经验。邵海杨先是介绍了两种DDoS的主要攻击类型，即缓慢性CC攻击和致命流量攻击，在他的日常工作中，遇到较多的是后者，来得快去得也快，不差钱的主经常选择这种方式。他指出：

> 一定要在第一时间发现攻击的征兆，及时作出反应。

黄冬曾经表示过，要防御DDoS，直接交给CDN就行了。邵海杨的观点与他不谋而同，自建CDN有以下考量：

1. 硬件成本（1U的机箱放多块主板，成本大约在一万五到两万之间）
2. 带宽成本（双线带宽贵，做CDN加速不需要双线，只需要单线机房即可，每兆大约只需1块多）
3. 架构设计
4. 配置要点
5. 智能脚本

他对比了Squid、Varnish、Nginx、Apache Traffic Server（ATS）和HAProxy的强弱，目前又拍大量使用了ATS，集群规模已经超过200台，ATS的集群功能现在还不完善，可以通过Nginx在前面做一层一致性Hash的转发，规避ATS的集群问题。另外他也强调了HAProxy强大的HTTP头解析能力，是用来充当防御层的合适选择。可以根据具体的用途进行选择：

- 反向代理（路由加速，隐藏主节点）：HAProxy>Nginx>Varnih>ATS>Squid
- 缓存加速（静态加速、节省带宽、边缘推送）：ATS>Varnish>Squid>Nginx>HAProxy
- 防御功能（快速解析、过滤匹配）：HAProxy>Nginx>ATS>Squid>Varnish

此外，选择的系统最好还要能支持文件读取和匹配，支持热加载生效和可插拔式的缓存组件灵活组合。

架构是需要持续改进的，又拍云的CDN就经过了这样一个过程：

- 智能DNS区域化（又拍云负责部署节点，通过DNSPod实现智能节点选择，自动选择离用户最近的节点，以此实现全网加速）
- 大规模日志分析（如何从日志中提取恶意代码进行分析？又拍云在Nginx中增加了一个模块，将最近的URL保存在内存中，以便实时分析，此外还有一个Hadoop集群分析日志）
- 后端管理不直观（使用OpenCDN来提供多节点CDN管理平台）

CC和DDoS可能会交叉进行，用HAProxy加后端存储，是应对小流量攻击的，如果在承受范围内，可以选择不切节点，但是如果遇到大流量DDoS攻击，可以立刻选择切节点。邵海杨强调到**防御DDoS攻击，要靠技术、靠业务，更要获取高层的支持**。

在讲了很多公有云相关的技术之后，支付宝的章邯为大家带来了一些与支付宝的私有云环境有关的内容，他介绍了支付宝私有云中的以业务为核心的监控产品。

在支付宝，除了常规的运维监控和应用监控，还有更多其他的诉求，例如业务监控、合作伙伴监控和SOA环境监控。

章邯特别强调了一个概念——业务分析，它在支付宝的监控体系中起着至关重要的作用：

- 实时BI——有时不是为了排查故障，而是为了确认没有问题
- 确定故障范围——不同的业务特征，代表了不同的故障影响范围；不同的影响范围，应急人员有不同的策略
- 业务与合作伙伴——比如银行，单个银行下跌，可能是银行的问题，所有银行下跌，可能是支付宝的问题
- 业务与应用的关系——通过监控不同的业务，可以快速定位故障
- 业务与业务的关系——虽然没有系统间的直接关系，但业务直接确实有可能会存在相互的影响
- 业务与运维策略的关系——例如，确定机房引流，流量的分配
- 业务与管控策略的关系——管控策略有很多，比如分组、降级、限流和引流，管控策略的制定和业务是息息相关的

很多公司都会采用在系统中埋点的做法进行监控，而支付宝则采用了业务分析结合现象分析的做法来进行实时故障应急处理。章邯指出：

> 埋点需要对所有服务器做埋点检查，而故障的原因是无穷的，往往可以从现象症状上来判断故障的原因。

随后，他简单介绍了一下支付宝内部基于日志的监控解决方案XFlush，其中借鉴了Percolator、Storm、Spark、HayStack、GFS和RDDS的很多思想。XFlush追求的是低侵入性、增量计算、不保存原始数据、保证时效性、保证数据准确性、保证可扩展性、避免冗余计算和计算逻辑可扩展性。为了实现上述内容，甚至还实现了一套定制的分布式文件系统XStore，它的特点是能够无限扩展，纯粹为周期统计计算和固话监控点常见而定制，能做到极低的IO，提供高速、无IO的元数据检索。

数据库的运维也是运维的重要工作，作为一个运维大会，自然少不了数据库相关的内容，ThinkInLAMP创始人马骏和MySQL技术专家金官丁分别为大家带来了很多MySQL数据库运维相关的经验分享。而来自金山网络的安全专家赵闽还和大家讲述了很多与Android安全相关的故事，在一个个的故事里让大家感

到移动端的安全也是个重要的领域，金山的火眼系统值得关注。

如果您也从事运维行业，或者是对运维感兴趣，那么现在会是个不错的机会，云计算时代中，机遇与挑战并存，如果能选择勇敢地接受挑战，一定会发现运维的领域也可以很精彩。

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

**相关内容**

- 电话录音成为法定证据 云计算保障数据安全
- 运维团队能从橄榄球教练身上学到什么？
- 交付云计算的复杂性
- QCon北京：十亿级PV性能测试，海量SNS社区网站高效运维
- 通过（美国）感恩节晚餐来类比形容云计算

# 避开那些坑 | Void

## 从光大证券交易系统8.16事件说起

作者 郑柯

8月18日，中国证监会新闻发言人通报了8月16日上午的“光大证券交易异常”事件的初步核查情况，认为问题源于光大证券自营策略交易系统存在缺陷。

8月16日11时5分左右，上证综指突然上涨5.96%，中石油、中石化、工商银行和中国银行等权重股均触及涨停。证监会核查发现：主要买入方为光大证券自营账户。后光大证券申请：午后暂停其股票的交易。

事件发生后，多方认为此次事件是一次“乌龙指”事件，也就是说可能是股票交易员、操盘手、股民等在交易的时候，不小心敲错了价格、数量、买卖方向等。

8月18日，证监会初步核查的结果显示：**光大证券自营的策略交易系统包含订单生成系统和订单执行系统两个部分，存在程序调用错误、额度控制失效等设计缺陷，并被连锁触发，导致生成巨量市价委托订单，直接发送至上交所，**累计申报买入234亿元，实际成交72.7亿元。此外，在核查中尚未发现人为操作差错，但**光大证券该项业务内部控制存在明显缺陷，信息系统管理问题较多。**

证券金融机构的系统事故近年来在国内外时有发生。

- 2008年9月8日，伦敦证券交易所当天却因为网络连接技术故障停盘近7个小时，令不少投资者痛失大好时机。有人推测是因为微软平台的缘故。一年半之后，伦敦证交所将系统切换到Linux平台之上。在2011年2月25日上午，伦敦证交所在当天早晨开盘竞价时发现“市场数据”部分出现故障，因此又停盘处理这一问题。他们使用的“千年信息技术”交易系统于2月14号刚刚启用。
- 2012年2月2日，东京证券交易所系统开盘前出现故障，导致241只股票停止交易。东京证交所表示，这次出现故障的是传递交易信息的系统。投资者在发出交易请求后，即使交易成功，股价也无法传递到投资者那里。进入午盘，东京证交所的系统得到恢复。
- 2012年3月，美国第三大证券交易所运营商BATS在上市当日经历一次

严重技术故障后宣布取消上市。随后，由于纳斯达克的系统瘫痪，社交网络巨头Facebook股票上市当天开盘时间被推迟了30分钟，交易故障给投资者造成了前所未有的困惑和混乱。
- 2012年8月，骑士资本由于技术故障在45分钟内损失4.4亿美元。
- 2012年8月6日，由于技术故障，西班牙股票交易所被迫暂停交易近5个小时。
- 2012年11月28日，瑞典斯德哥尔摩证券交易所因为程序bug，导致交易系统中断了数小时。自动交易系统发出了4,294,967,290指数期货购买指令，每份价值约等于16,000美元，总价值69万亿美元，是瑞典GDP的131倍。这项交易后被废止，但它的后遗症导致交易系统中止了数小时。

不仅是证券行业，银行业今年在国内也颇不平静。

之前，InfoQ曾就去年中国银行IBM大型机宕机事件，采访了兴业银行的系统分析师周伟然和支付宝的应用运维架构师。

优秀的金融系统，需要设计、开发、测试、运维等多个环节的配合与反馈。不妨回顾一下这些环节的优秀内容，希望其中的观点和实践能为广大读者提供有价值的参考。

兴业银行的系统分析师周伟然曾在InfoQ中文站主办的全球架构师峰会上做《联机服务平台的安全运行设计》的演讲，其中以几个基于商业中间件的架构、以及轻量级联机服务的架构为例，从不间断运行、流量控制、故障隔离、交易运行时间控制、系统超时等方面对Native Code联机服务系统的高可用性架构设计进行讨论。

中文站还曾有一篇文章——《关注网银系统：安全模型和架构设计》，简要描述了网上银行普遍采用的安全系统架构以及相关技术。文章在分析安全性需求的基础上，引入了目前网上银行最为著名的安全系统架构模型——PPDRR 模型，即策略 (Policy)、防护 (Protection)、检测 (Detection)、响应 (Response) 和恢复 (Recovery)，并认为这是“一种动态的、自适应的安全模型，可适应安全风险和安全需求的不断变化，提供持续的安全保障“。

平安科技应用开发支持部门的测试工程师刘毅，在自己的博客中撰写了一篇文章——《紧耦合金融系统群的测试自动化策略（一）》，其中讲到了单元测试、环境监控、测试数据和持续集成方面的方针和策略。文中的背景部分提到：

> 科技子公司或者IT部门在一个大金融团体里面只能算是个成本中心，对IT团队

> 来说，核心使命就是稳定运营、降低成本。这对于自动化测试来说，意味着非常有限的资源预算、不稳定的测试环境、复杂的系统耦合关系、严苛的测试数据要求，还有那近乎无理的信息安全规范。如此种种，让我们并不能按照自己想象的那样去实施自己的规划，以致会走很多弯路；而再回首你会觉得，有时候只是方法欠妥而不是资源不够，有时候只是导向错误而非技术不够高。

除了技术方面，应用好的项目管理的框架与实践重要，InfoQ中文站曾经发布过一篇虚拟座谈会——《Scrum是否适用于金融电信行业？》，其中对如何在金融行业应用Scrum等框架有深入讨论，时任上海惠普资深敏捷顾问的徐毅提到：

> 金融电信业相对目前蓬勃发展的互联网行业而言，采用的技术相对落后，多采用重型软件开发方法，产品开发所涉及的人员为数众多，这恰恰需要敏捷来解除束缚释放出潜藏的巨大生产力。

当今这个高速发展的全球化和信息社会，金融行业发挥了心脏的作用，不停为其他行业造血——提供资金，而金融行业的IT系统又相当于他们自己的心脏，是重中之重，这也使得金融一直走在IT技术和系统的前列。然而，这些时常出现在耳边的故障，却总令社会公众有种种担心。作为IT从业者，我们有责任、有义务将自己的事情做得更好、更完善、更到位，从而尽可能避免类似问题发生。

**原文链接：http://www.infoq.com/cn/news/2013/08/everbright-8.16-event**

**相关内容**

- 当敏捷遇上销售
- 在敏捷开发过程中，架构师是鸡还是猪？
- 对比其它软件方法评估敏捷和Scrum
- 个人如何采用敏捷的工作方法
- 通过优先级游戏裁剪待办事项列表以避免浪费

# 避开那些坑 | Void

# 提升.NET应用程序启动速度

**作者 Jeff Martin ，译者 马德奎**

.NET框架是Windows系统上所有基于.NET的应用程序的基础。出于性能方面的考虑，Windows会运行.NET框架优化服务（它的可执行程序名称是mscorsvw.exe），优化用户机器上.NET应用程序的性能。初次安装新版本的.NET框架，或者框架稍后收到Windows更新的补丁，均会触发激活该服务。

NGEN是本机映像生成器，已经作为.NET框架的一部分存在了好几年。NGEN创建托管代码应用程序的本机映像，该映像经过优化，可以在运行时直接使用，而不需要使用即时（JIT）编译器。随着时间的推移，NGEN不断发展，它包含了一个作为后台服务运行的组件，试图以此确保.NET应用程序持续优化。通过设置自定义选项，NGEN也可以针对特定程序集在前台运行。

AutoNGEN是该产品的最新成果，它伴随Windows 8出现，并在Windows 8.1中进行了进一步的优化。相比之前的迭代，它体现了数项改进。按照设计，在Windows 8中，AutoNGEN只优化机器上正在使用的.NET应用程序或库。，设计还使它在后台运行时更高效，并且当用户返回机器时它会让出处理器。

在Windows 8.1中，该服务试图利用微软的构建系统。这样，来自微软的基于.NET的包，可以在分发之前使用类似NGEN的服务进行优化。这样做的结果是，AutoNGEN需要在本机上完成的工作更少，也避免了在数百万台运行着.NET的机器上分发程序包时重复同样的优化工作。

**查看英文原文：**Improving NET Application Startup Speed

**原文链接：http://www.infoq.com/cn/news/2013/08/net_AutoNGEN**

**相关内容**

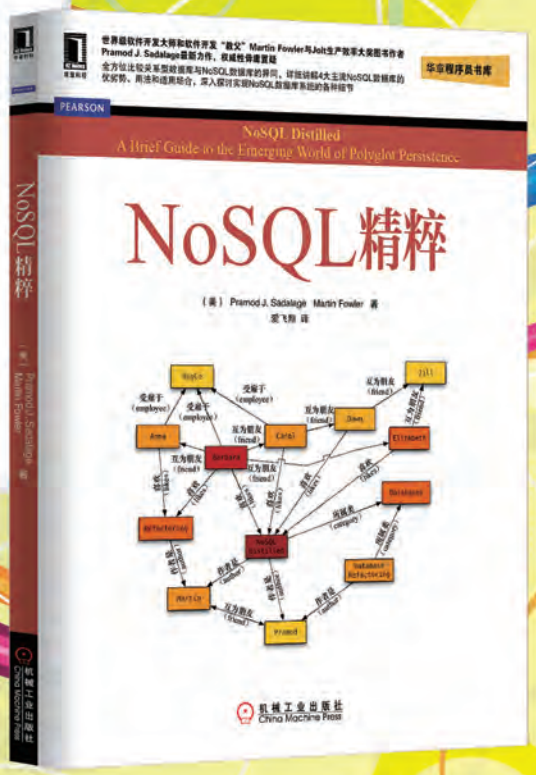
- .NET Reflector 8.2 支持Visual Studio 2013、 局部变量高亮显示和代码地图视图
- Google API的.NET可移植类库1.4.0测试版发布
- 高性能的.NET不可变数组
- .NET、Silverlight和Windows Phone的API变更列表

# 华章精品图书推荐

**苹果的产品设计之道**

深刻解读苹果公司产品设计的真谛与精髓，揭示了苹果公司创造伟大的产品所遵循的7大设计原则

ISBN：978-7-111-42714-8
定价：69.00元
作者：（美）John Edson 著
译者：黄喆 译




**NoSQL精粹**

世界级软件开发大师和软件开发“教父”Martin Fowler与Jolt生产效率大奖图书作者Pramod J. Sadalage最新力作，权威性毋庸置疑

ISBN：978-7-111-43303-3
定价：49.00元
作者：（美）Pramod J. Sadalage, Martin Fowler 著
译者：爱飞翔 译

## 诚征合作，欢迎投稿

如果您有写作或翻译计算机图书方面的计划，欢迎通过以下方式与我们取得联系。
邮箱：linux1689@gmail.com 电话：010-88379525
微信：linux1689 联系人：杨福川（副总编辑）

机械工业出版社
China Machine Press

# 新品推荐 | Product

## Google开放HTML5解析库Gumbo的源代码

**作者 Abel Avram ，译者 马德奎**

Google已经开放了Gumbo的源代码，这是一个用C编写的HTML解析库。该库符合HTML解析算法标准，它通过了所有的html5lib-0.95测试，并且已经在Google索引的25亿个网页上进行过测试。

**原文链接：http://www.infoq.com/cn/news/2013/08/google-gumbo-html5**

## Bootstrap 3拥有全新视觉方案和更多的组件

**作者 Abel Avram ，译者 李彬**

Bootstrap 3.0发布，增加全新方案和更多的组件；此外，它与上一版本相比，除了补丁外，还做出了许多破坏性变更。

**原文链接：http://www.infoq.com/cn/news/2013/08/bootstrap-3**

## 来自百度的移动应用框架Clouda：快速开发，一键部署BAE

**作者 邱彦铭**

在2013年8月22日的百度世界大会上，百度提出了“轻应用”的概念，并发布了一个叫做Clouda的JavaScript开发框架。百度希望通过这个开发框架打破既有的Native App与Web App市場，创造出一個更好的Hybrid App市場。

**原文链接：http://www.infoq.com/cn/news/2013/08/baidu-clouda**

## WinJS 2.0：更快的速度、更多的功能

**作者 Jeff Martin ，译者 孙浩**

Windows JavaScript库（WinJS）首次是和Windows 8一起出现的，它为JavaScript开发者提供了一种有效的方式来创建Windows应用商店应用。2.0版本将作为Windows 8.1发布的一部分，微软也承诺将增加其特性以及大大提高其性能，这将使所有用户受益，特别是那些运行着ARM或者基于Atom CPU的设备。

**原文链接：http://www.infoq.com/cn/news/2013/08/winjs20**

# Vim 7.4发布，带来了更快的正则表达式引擎及1000多项修复和小改进

**作者 Roopesh Shenoy ，译者 马德奎**

经过一个多月的测试后，Vim 7.4于近日发布。该版本更健壮，并且带来了一个更快的新正则表达式引擎。

**原文链接：http://www.infoq.com/cn/news/2013/08/vim-7-4**

# 移动开发技术周报：使用GenyMotion加速Android开发，iOS 7 UI设计技巧

**作者 邱彦铭**

本周在移动方向值得关注的内容：Google Cast SDK Update，Bootstrap 3.0，GenyMotion试用体验，iOS Design Cheat Sheet，Mail App滑动的实现，Hybrid Apps for iOS开发指南，如何设计一个吸引人的广告。

**原文链接：http://www.infoq.com/cn/news/2013/08/mobile-weekly-genymotion**

# Weblogic+Coherence+Tuxedo=云应用程序基础（CAF）

**作者 Bienvenido David III ，译者 马德奎**

Oracle发布了云应用程序基础（CAF）12c，它是一个可以在传统的数据中心和云环境中工作的集成中间件平台。该平台是多款Oracle产品的组合体，包括WebLogic Server for Java EE、内存数据网格Coherence、Tuxedo for C/C++C/COBOL、用于配置和部署的Oracle Virtual Assembly Builder和用

于负载均衡的Oracle Traffic Director。

**原文链接：http://www.infoq.com/cn/news/2013/08/cloud-application-foundation**

# 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

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

商务合作：sales@cn.infoq.com

读者反馈/投稿：editors@cn.infoq.com

# 推荐编辑 | 邱彦铭

大家好，我是邱彦铭，来自台湾的90后开发者，很高兴成为本期的《架构师》推荐编辑，对于刚成为编辑的我真是莫大的鼓励。一切都要从今年七月底说起，因为参加了百度黑客松台北场的活动，与InfoQ结缘，以前在台湾我会关注的科技媒体是本地的iThome以及国外的Engadget，不过参加之后与来协助办活动的InfoQ编辑以及接洽我们到北京的InfoQ成员Yolanda & Jessie 有了交集，便更加瞭解InfoQ的核心精神以及来自InfoQ的热情。在北京办公室与Kevin跟Sai交流并且分享对于科技媒体的想法之后，我鼓起勇气向他们表达了我也想为台湾写稿的想法，就在隔天的百度世界发表会，我拿到了成为社群编辑的机会，也紧紧抓住了这次的机会以一位编辑的身份採访了百度的工程师，深度瞭解了一个新产品的起源以及设计方式，那时的心情已非慷慨激昂可以形容的了，消失了许久的热情如滔滔江水搬的涌现了出来，再飞回台湾的飞机上，我完成了第一篇的报导来自百度的移动应用框架Clouda：快速开发，一键部署BAE，那时我立马把稿子发给了InfoQ的杨赛，她帮我改完稿之后我才发现原来身为编辑写稿除了要前后工整之外还要言简易骇，对于以往都是写流水账的我才是一大考验，不过杨赛很细心地跟我讨论以及修改之后才把稿发出去，那时我才发现当一位编辑不是一件简单的事情。能跟InfoQ的编辑们共事真的可以学到很多，大家都是不同主题的专业编辑，从移动应用、互联网、云技术、大数据等等，从大家发的文章可以快速地去学习以及了解到目前最尖端的技术以及业界趋势。

当我还是学生的时候，就开始订阅了很多资讯相关的RSS(像是Cult of Mac、the next web、VentureBeat、TechOrange)，特别会关于的为MAC跟iOS相关的，我也不讳言地说我是个果粉，我常常会熬夜看Apple的发表会然后兴奋得睡不着觉。在成为编辑之前，只能朋友们分享彼此对于新技术的开发心得以及资讯，闲暇之余也会去参加一些新技术推广的黑客松及研讨会，我发现自己对于新技术充满着热情。渐渐的，我发现自己更适合去当一个编辑或是一个布道者的角色，希望可以把自己所知道的及学习到的新知识或是新技术，透过文字或是视频去分享给更多的人，不仅仅只是身边的朋友，我觉得资讯就应该要透过媒体平台才能更快速更即时的去传递，这样才不会让这项技术因为宣传不佳而因此埋没了。然而，InfoQ就提供了这麽一个平台，让我可以去分享在互联网最新的信息以及业界研讨会最新的技术，让更多的资讯人可以透过这个平台去接收到最新的技术以及瞭解业界的动态。

如果你也有意愿成为InfoQ大家庭的一员,请发邮件到:editors@cn.infoq.com。 也欢迎大家与我(新浪微博@steveicetw、Facebook@stevechiou)交流。

# 封面植物

## 风信子

风信子，为风信子科风信子属中的多年生草本植物，具鳞茎。 原来属于百合科，现在已被提升为新的风信子科的模式属，而风信子也从原来的百合目改到新成立的天门冬目中。其花色彩丰富，有红、蓝、白等多种颜色，培植方式以分球繁殖为主，经过长时间的杂交与繁殖栽培，风信子渐渐产生了一些特殊的重瓣品种。风信子原产于欧洲南部、荷兰，学名得自希腊神话中受太阳神阿波罗宠眷、并被其所掷铁饼误伤而死的美少年雅辛托斯（Hyacinthus）。其花语为：悲伤的爱情、永远的怀念等。 希腊神话中受太阳神阿波罗（Απλων）宠眷、并被其所掷铁饼误伤而死的美少年雅辛托斯（κινθο），是由于西风风神泽费奴斯（Zephyrus）用计害死。在雅辛托斯的血泊中，长出了一种美丽的花，阿波罗便以少年的名字命名这种花——风信子υκινθο（Hyacinthus orientalis）。 多年草本生球根类植物，鳞茎卵形，有膜质外皮，皮膜颜色与花色成正相关。未开花时形如大蒜。叶4～8枚，狭披针形，肉质，上有凹沟，绿色有光。花茎肉质，长15～45厘米，总状花序顶生，小花10～20朵密生上部，横向或下倾，漏斗形，花被筒形，上部四裂，反卷，有紫、玫瑰红、粉红、黄、白、蓝等色，芳香，蒴果。自然花期3～4月。园艺品种有 2 000多个，根据其花色，大致分为蓝色、粉红色、白色、紫色、黄色、绯红色、红色等七个品系。

# QCon

## 全球软件开发大会 [上海站] 2013

International Software Development Conference

上海光大会展中心国际大酒店
2013年11月1-3日

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

**9折优惠抢购中**

## 大会最新精彩内容呈现：

Twitter工程团队VP / Raffi Krikorian
Twitter面向服务架构之路

Google分布式系统与性能专家 / Clive Saha
Google高性能CPU/GPU集群

LinkedIn数据团队首席工程师 / Sam Shah
如何将数据变为产品

Facebook资深前端工程师 / Hedger Wang
Facebook的JavaScript和ReactJS

阿里巴巴网站架构师 / 周涛明
跨境网站性能优化挑战和思路

GitHubber / Ben Lavender
机器人协助的远程协同办公

京东云平台首席架构师 / 刘海锋
京东文件系统与统一数据中心存储

用友畅捷通智能平台总监 / 张俊林
微博环境下利用综合策略构建推荐系统

阿里巴巴高级技术专家 / 温绍锦
为监控而生的数据库连接池Druid

爱立信软件开发高级专家 / 蔡煜
突破思维，建立环境、工具、自动化团队

天猫前端业务负责人 / 舒文亮
天猫以移动优先的跨终端解决方案

东软产品管理中心主任 / 殷坤
顺势而为的Web自动化测试

阿里巴巴集团高级安全专家 / 魏兴国（云舒）
弹性计算云安全的反思与展望

大众点评网研发管理高级经理 / 唐灏
大众点评敏捷转型的三个三角

抢购热线：**010-64738142**　　会务咨询：**qcon@cn.infoq.com**

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

1kg.org 多背一公斤

爱自然 | 更爱孩子

# 架构师 9 月刊

**每月 8 日出版**


本期主编： 水羽哲

顾问：杨赛

美术/流程编辑：水羽哲

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 :http://weibo.com/infoqchina

商务合作：sales@cn.infoq.com

本期主编：水羽哲，InfoQ 中文站内容运营编辑

水羽哲（@麦可思哲），写代码、看书、追美剧：）


《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。