

架构师

11月 ARCHITECT

特别专题

软件测试的方方面面

GUI功能测试自动化模式

测试覆盖（率）到底有什么用？

全程软件测试实践：从需求到运营

AWDC（阿里云开发者大会）

云的蝴蝶效应

阿里云唐洪谈飞天现状以及5K项目

如何挑选合适的大数据或Hadoop平台

HotSpot和OpenJDK入门



InfoQ
架构



每月8号出版

卷首语

第一手知识——切身体验

InfoQ中文站的QCon大会已经为业界的朋友所熟知，三天的日程中，各种演讲干货之多，场面之精彩已经让越来越多的人愿意加入到这个大平台中来交流和分享。而大家不知道的是，为了让各位讲师能够在现场有更出色的表现，InfoQ从今年的QCon北京开始，都会在会前一个月左右为各位讲师安排一场特殊的培训——QCon讲师训练营。

10月12日，我也有幸作为InfoQ的编辑参加了训练营，聆听了杨天颖老师一天的课程，收获非常大。而在10月26日大连本地的QClub活动上，我在演讲中着重实践了杨老师教授的各种技能，包括：手势、J-Cutting的场景转换、放慢语速、把QA环节放在总结之前等等，得到的效果非常不错。

经过活动上的演讲，非常重要的一点就是，我把训练营上获得的知识真正消化吸收了一部分，如果说经过别人总结归纳然后讲述出来的知识是第二手的知识，那么在实践中通过自己的切身体验所获得的知识绝对是第一手的知识，而这种知识也因为其新鲜的程度以及和自己密切相关，所以更容易让我牢记。

在演讲的过程中如此，在作为程序员编写各种各样程序的过程中也是一样。曾经在公司面试新员工的时候，非常注重的一点就是——是否做过真实的项目，相信很多公司中的面试官也是一样。仅仅学习并掌握了书本上的内容是不够的，“纸上得来终觉浅，绝知此事要躬行”，只有在切实利用学到的知识解决了实际的问题之后，即获得了第一手的知识之后，才能够对其有深入的了解，从而可以在以后的工作中更好地应用。这也许就是大家所说的经验吧。

在网络上经常会有各种各样的争论，有时是使用各种语言的程序员争论到底哪种语言才最好、才是王道，有时是开发人员和测试人员相互攻击，都认为对方的工作不重要，只有自己的工作才是重中之重。其实，往往争论的同学对于另一方的知识只是有个简单的了解，掌握的大多是二手的知识，即便有少许一手知识，通常也是浅尝辄止。正因为没有切身体验，才会因为不了解真实的情况而去争论。如果一名程序员掌握了多种语言、框架、工具，那么就不会争论孰优孰劣的问题，而是会根据具体的情况选择最为合适的工具来解决问题；如果一个人既做过开发，也做过测试工作，那么就不会厚此薄彼，因为他很清楚，两种角色都是团队的组成元素，都有其重要的作用，缺一不可。

对于敏捷的方法也同样有很多质疑，不少人总是觉得只有传统的软件工程方法才能够真正保证项目的成功。仅仅比较一下二者的不同就妄下断言。其实，想要知道敏捷的方法是否适合自己的项目，是否能够解决团队管理以及项目开发中的问

题，唯一的方法就是“just do it！”尝试过一次，无论成功与失败，都会获得第一手的知识，成功的话，可以作为经验来推广，在以后的项目中进一步应用和提升；失败的话，也明白到底是什么地方出现了问题，以后可以尽量去避免类似情况的发生。这些都要比单纯的怀疑和无意义的争论强得多。

第一手的知识之所以重要，是因为它只能存在于一个人的头脑之中，体现在一个人的行为之中，而不是简简单单通过在网上Google一下就能够找到的。一个人只有掌握了足够的第一手知识，才能够不断提升，体现出与其他人的不同，才能够达到传说中的高手境界。所以，程序员朋友们，让我们为了掌握更多的第一手知识而努力吧！

本期主编：侯伯薇



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 |

目录

人物 | People

UNIX环境高级编程：Stephen Rago访谈

观点 | View

Kent Beck揭秘Facebook开发部署流程

NoSQL更适合担当云数据库吗

本期专题：软件测试的方方面面 | Topic

GUI功能测试自动化模式

测试覆盖（率）到底有什么用？

全程软件测试实践：从需求到运营

推荐文章 | Article

太多选择——如何挑选合适的大数据或Hadoop平台？

HotSpot和OpenJDK入门

特别专栏 | Column

阿里巴巴集团CTO 王坚博士：云的蝴蝶效应

阿里云计算资深总监唐洪谈飞天现状以及5K项目发展

恒拓开源陈操谈去IOE方案的普及对独立开发商的机遇与挑战

避开那些坑 | Void

双重检查锁定与延迟初始化

从CDN到云计算，在变中寻找不变

新品推荐 | Product

Oracle在JavaOne大会上揭开了Project Avatar的面纱

Java内存数据网格Hazelcast 3.0支持连续查询和条目处理

新的.NET编译器——RyuJIT 项目

使用Ruby开发iOS游戏

Backbone 1.1.0 发布，部分内容与版本1.0不兼容

Rubinius 2.0发布，实现了Ruby 2.1

QClub

我们影响有影响力的人

非盈利、非商业、纯技术交流的技术社区活动

We are QClub

QClub作为InfoQ线下技术沙龙品牌，定期在全国主要城市举办免费的技术沙龙，邀请国内外知名公司技术总监，项目经理，高级研发工程师等走入各地技术社区，分享他们的经验与对行业趋势的预测与讨论，为中国技术人员搭建交流、分享的平台，尽自己微薄之力架起中高端技术人员之间的桥梁，为中国技术社区的发展与价值的传播贡献自己的力量。

非盈利 非商业 纯技术交流

QClub的话题

QClub的话题可能包括任何当地技术人员感兴趣的话题，比如：编程语言、架构设计、企业级开发、运维、基础架构、过程与实践、云计算、大数据、互联网、移动开发、安全、敏捷、性能、业务流程、SOA、.....



参会人群：

开发人员、技术或
团队负责人、技术
爱好者、学生等

活动城市：

北京、长沙、成都、大连、福
州、广州、上海、太原、天津、
温州、西安、郑州、.....

举办周期：

每个城市
每1-2个月一次活动

活动形式多样：

- 讲师演讲 ◦ 线上交流
- Open Space ◦ 粉丝 QQ 群
- 户外活动 ◦ 等等.....

● 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44期

大数据面面观

11月23日，北京车库咖啡



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

Bai^必度

InfoQ

人物 | People

UNIX环境高级编程：Stephen Rago访谈

作者 [Jeff Martin](#)，译者 [臧秀涛](#)

《UNIX环境高级编程》（Advanced Programming in the UNIX Environment, APUE）被誉为基于UNIX的编程环境的圣经。本书全面介绍了UNIX环境上的C语言编程，涵盖文件I/O、进程、信号、线程、进程间通信和套接字（Socket）等主题。第3版结合FreeBSD 8、Linux 3.2.0、OS X 10.6.8和Solaris 10讨论了这些概念。

APUE的[第3版](#)已于近期出版。InfoQ有机会采访了其作者Stephen Rago，谈到了这一最新版本以及UNIX开发。

InfoQ：本书第1版出版于1992年，2005年更新过一版。在第3版中，你的主要目标和动机是什么？

Stephen Rago：在第2版出版之前，我一直受困于没有足够的时间来更新某些平台相关的内容。尤其是我想用Linux 2.6代替2.4，因为2.6对pthread的支持更好一些，而且其表现与其他平台更为接近。但当时2.4仍然有很大的装机量，所以我保留了它。第3版用了我两年的时间才得以出版，因为在这期间，书中覆盖的平台频繁更新了好多次，我感觉自己一直在追赶。

InfoQ：你感觉哪部分写起来最有意思？

Rago：这就好像问我最喜欢自己的哪个孩子。从遗传学角度讲，好父母不能厚此薄彼。我也一样。相对于我的工作，在APUE第3版上的工作是一次让我耳目一新的改变。我本质上是一位C程序员和操作系统开发人员，而UNIX系统的优雅超越了其他所有操作系统，所以每部分工作我都非常喜欢。

InfoQ：与第2版相比，面向的读者有变化吗？比如说，这本书是面向职业开发人员的，还是面向学习相关编程知识的人员的，这方面是不是有所改变？

Rago：APUE最初是作为“Addison-Wesley专业计算丛书”（Addison-Wesley Professional Computing Series）的一部分出版的，所以我假定读者是职业程序员。不过从我最近收到的提问问题的电子邮件来看，很多是来自学术

界（对于赤裸裸地要答案的请求，我尽量不直接回复）。我知道一些系统编程类课程用到了这本书，我猜随着UNIX系统及其克隆产品在业务中越来越常见，越来越多的开发者都掌握了书中材料，所以本书面向的读者某种程度上也向学术界迁移了。或许也可以看做对这种改变的反映，我目前正在编写APUE第3版配套的教师手册，其中包括了书中所有习题的答案，还添加了一些书中没有的新习题。

在描述实际编程问题以及揭示很多UNIX系统接口的背景方面，这本书做得不错。所以我认为这本书可以很好地服务这两类读者。

InfoQ：你有没有发现基于**UNIX**系统比较适合学术研究？这是因为其设计内在的特性，还是说只是因为相对于商业系统，其源代码很容易获得？

Rago：这两方面的原因使基于UNIX的系统用于学术研究非常理想。其设计简洁清晰，各种实现的源代码也可以免费获得，所以我们可以看到抽象的概念是如何映射为实际实现的。你可能需要把UNIX系统包含在“商业系统”中，因为很多业务都运行在UNIX系统之上。

InfoQ：你感觉你的书在哪种环境上更受欢迎，是**UNIX**、**OS X**还是其他系统？

Rago：很难说。尽管我并没有一种很好的方法来衡量各种环境的受欢迎程度，但是从我收到的电子邮件来看，大部分人运行的都是某个版本的Linux。

InfoQ：考虑到**C**语言与**UNIX**的历史关系，所有例子都是用**C**写的。你有没有发现哪种语言能挑战**C**语言？**C**语言足够好了吗，还是说有新语言可以改进系统编程？

Rago：40年来，C语言一直是够用的。我工作过的地方曾经把C++当做更好的C，因为它会进行更强的类型检查。但是C++如此庞大和复杂，使用起来非常困难。编写C程序时，我不需要不断地查参考书，以确定这样那样的特性应该如何使用；编写C++程序就不是这样了。我更喜欢C，因为可以用同一种语言处理高层和低层的东西，语言规范也足够简单，可以记在脑子里。操作系统需要与硬件交互，并基于高层构造提供功能，所以C语言可以很好地满足需求。使用底层操作系统所用的语言来编程会比较轻松。我还没研究过Go语言，但会找机会看一下，因为该语言尝试解决用C和C++之类的语言构建大型项目时所遇到的某些软件工程问题。

InfoQ：在花时间写书的时候，你有没有发现**UNIX**有何不足？

Rago：或许UNIX系统需要一些简单的特性使执行更安全。比如，在IPC通道（如命名管道或套接字）的一端很难获得一个进程的标识信息。但是UNIX系统起源于协同环境，所以它没有提供更多内置的认证基础设施这一点也很容易理解。但是相对于使用该系统能做的所有事情，这只是小瑕疵了。

关于**APUE**一书作者



Stephen A. Rago是《UNIX® System V网络编程》(Addison-Wesley, 1993)一书的作者。Rago是贝尔实验室参与构建UNIX System V Release 4的开发者之一。他曾经是《UNIX环境高级编程》第1版的技术审校者。Rago目前是NEC美国实验室存储系统组的一名研究人员。

W. Richard Stevens是公认的UNIX和网络专家，也是一位备受尊重的作家，还是广受欢迎的讲师和咨询师。他最著名的是《UNIX网络编程》、《UNIX环境高级编程》和《TCP/IP详解》等一系列书籍。

查看英文原文：[Advanced UNIX Programming: An Interview with Stephen Rago](#)

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

相关内容

- [红帽企业版Linux新添SQL Server驱动](#)
- [AIDE 2.0引入对原生C/C++应用的支持](#)
- [今时今日，C还适合当下之所需么？](#)
- [LLVM提议向C语言中加入模块机制](#)
- [网络报文处理的两个模式](#)

观点 | View

Kent Beck揭秘Facebook开发部署流程

作者 [工雪丰](#)

Facebook是世界上最大的社交网站，有超过10亿用户每月至少要登录一次，他们每天要上传超过25亿内容，支持这样一个站点的运行，还要不断发布新的功能，Facebook的工程师是如何做到这一切的？目前就职于Facebook的极限编程创始人[Kent Beck](#)在近期发表的一篇与别人合著的[论文](#)里向大家详细介绍了Facebook的开发与部署流程。

显而易见，Facebook的工程师们不会像传统软件行业那样使用瀑布模型进行开发，他们不断地开发新的功能，并迅速上线，让用户能够访问到这些新功能，这就是大家口中经常提到的持续部署（continuous deployment）。在他们看来，Facebook的开发永远没有到头的那一天，代码库在不停地增长着，目前已经有超过1000万行代码，其中850万是PHP代码，代码随时间呈现超线性增长的趋势。

在Facebook，所有前端工程师都工作在同一个稳定的分支上，这也能加快开发速度，因为省去了繁琐的分支合并过程。在日常开发中，每个人都用git在本地进行开发，当代码就绪之后，就会将它推送到SVN上（之所以是SVN，这是出于历史原因），这样就很自然地区分开了开发中的代码和可以上线的代码。

但是为了保证网站的稳定运行，并非是工程师将代码推送到SVN上，认为可以上线，代码就能发布上线的。Facebook采用了一种兼顾了速度与稳定性做法——将每日发布与每周发布结合到一起。所有的代码变动默认是每周发布，每次发布会包含相对比较多的变更，在每周日的下午，代码会被发布工程师推送到SVN上，随后会进行大量的自动测试，其中包含很多针对正确性和性能的回归测试，这个版本会成为Facebook员工内部使用的默认版本，正式的发布通常被安排在周二下午。

发布工程师会为每个工程师的历史表现打分，内部称为“Push Karma”，比如那些代码经常出问题的人，分数就会相对较低，他们的代码自然也会受到更多的“关照”。这样做的目的是控制发布的风险，而非对某人做出评判，因此这个分数是保密的。除此之外，越是大的变更，或者在Code Review时讨论越是多的代码，也是风险较高的地方，同样会受到更多的“关照”。

在每周发布以外，其他工作日每天会有两次小发布，大多是些非关键性的更新，或者是些Bugfix，极端情况下会进行更多的发布，甚至是在周末进行发布。

在被纳入发布之前，代码已经经过了开发者的单元测试和Code Review，在Facebook，Code Review是非常重要的事情，他们使用名为[Phabricator](#)的工具进行Code Review，该工具是和代码版本管理整合在一起的。

在大量的自动化测试之外，每位员工在内部使用Facebook时也相当于进行了高密度的测试，每位员工都能报告自己发现的问题，写代码的人多了，代码增长的快了，相对而言，对代码进行测试的人也多了。

在性能方面，Facebook使用Perflab对新老代码的性能进行对比，如果新的代码性能不理想，并且开发工程师无法及时修复，那么相关代码就会从本次发布中剔除出去，待问题修复后再进行发布。每个小的性能问题都是不容忽视的，因为小问题会很快累积起来，变成影响容量和性能的大问题，Perflab能通过图表的形式直观地展现系统的性能。

像Facebook这样一个网站，每周发布自然是分阶段进行的，首先是H1，即部署到仅有内部访问的服务器上，进行最后的测试，很多公司也称其为“预发布”；随后是H2，部署到几千台服务器上，开放给一小部分用户；如果H2阶段没有发现问题，则进入H3，部署到全部服务器上。

如果在这个过程中发现问题，工程师会立即进行修复，随后重新开始分阶段的部署。当然，也可以选择回滚代码，有两种回滚方式——常见的是回滚某个变更及其依赖的文件，另一种则是回滚整个二进制包。

Facebook在四个不同的地理位置分布了大量的服务器，整个发布的包大约有1.5G，一般需要20分钟来完成整个分发。为了实现这一点，分发过程中分发使用了BitTorrent，分发时也会考虑到机架和集群的亲缘性。自从Twitter开源了他们的基于BitTorrent的发布方案[Murder](#)后，通过BitTorrent进行发布已然成为了业内的标配。

在发布时，与变更相关的开发者必须在线，发布工程师会通过IRC机器人进行确认，如果人不在，那么他的变更会被回滚。这样保证了问题能够在上线之初就被快速发现并修复，当然，想在这么大的一个系统里及时发现一些问题有时也是很困难的，所以Facebook会结合内部工具[Claspin](#)和外部的信息源（比如Twitter）持续地监控系统的健康状态。

通过Gatekeeper系统，工程师们可以方便地控制多少用户能够访问特定的新功能，筛选的条件可以是地区，也可以是年龄，在遇到问题是也能迅速关闭某个功能的入口。在Gatekeeper的帮助下，工程师们能方便地进行A/B测试，藉此迅速收集用户的真实体验，对产品做出调整。不要忘了，在Facebook，是工程师来选择自己做什么的，那么工程师们肯定是选择把东西做出来，看看用户的反应，而不是坐在会议室里和一堆人开会去猜测用户想要什么。

Kent Beck在文中表示：

仅有方法论和工具是远远不够的，因为它们总是会被误用。所以，拥有鼓励个人责任感的企业文化是很重要的。

现在，Facebook有大约1000名开发工程师，仅有3名发布工程师，没有独立的测试工程师。每位工程师都可以看到全部的代码，并且能提交补丁，或者提交详细的问题描述。工程师们需要自己编写详尽的单元测试，他们的代码还要通过所有的回归测试，并能支持后续的各种运维工作。

除了要对自己的代码负责，他们还要面对各种巨大的挑战，往往要针对多种解决方案进行大量试验。比如，当时为了解决PHP的性能问题，有3个不同的方案同时在进行开发，当某个方案的负责人发现另一个方案更好时，他们就会停下来；最后HipHop胜出了，但另两组人的精力也没白费，他们提供了重要的备份能力。

在文章的最后，还提到了Facebook的新兵训练营制度，关于这一点，Facebook的早期员工王淮在他的《调教你的新工程师 - 谈新兵训练营》中做了详细的描述。

关于Facebook，有很多值得深入学习和探讨的地方，比如他们的工程师文化，比如上文提到的新兵训练营。不知您在看了Kent Beck的文章之后有何感想，能否和InfoQ的读者们一同分享一下呢。

原文链接：<http://www.infoq.com/cn/news/2013/10/facebook-development-deployment>

相关内容

- [Facebook已将HHVM/JIT用于其开发和产品中](#)
- [Sean Lynch谈Facebook Claspin监控工具的由来](#)
- [Facebook元老王淮谈科技公司应有的工具文化](#)
- [移动开发技术周报：Facebook Login重新设计，iOS 7应用更新设计指南](#)
- [Facebook发布用户界面库React，业界褒贬不一](#)

观点 | View

NoSQL更适合担当云数据库吗

作者 崔康

在过去几十年，关系型数据库管理系统一直是数据管理的主要模型，随着Web应用数据规模的显著增长，NoSQL系统逐渐引起关注。领域专家Sherif Sakr[分析](#)指出，NoSQL具备的优势（能够水平扩展数据、支持较弱的一致性模型、能够使用灵活模式和数据模型、支持简单的低级查询接口）使其更适合在云计算领域做数据管理。

Sherif认为，云模型导致了云数据库模型的出现，事实上，有三个主要技术常用于在云平台上部署软件应用程序的数据库层：

- 虚拟化数据库服务器
- 数据库即服务平台
- NoSQL存储系统

对于虚拟化来说，一般而言，资源虚拟化技术在应用程序和应用程序使用的资源之间添加了一个灵活的可编程的软件层。定义虚拟化数据库服务器的概念充分利用了这些优势，特别是当软件应用程序的现有数据库层可直接导入公共云中的虚拟机时，这些软件应用程序是专为在传统数据中心使用而设计的。

这样一个迁移过程通常只需要在部署的应用程序代码或基础架构中进行微小改变。和其他软件组件一样，在该虚拟化数据库方法中，数据库服务器被迁移到虚拟机中运行。为每个数据库副本部署一个VM会带来性能开销，这类开销估计小于10%。实际上，虚拟化数据库服务器方法的一个主要优势是，应用程序可根据需要完全动态控制数据库层（数据库服务器）物理资源的分配和配置。

“数据库即服务”是这样一个概念，第三方服务器供应商托管一个关系型数据库作为一种服务。这类服务缓解了用户购买昂贵硬件和软件、处理软件升级以及雇佣专业人员进行管理和维护的需求。

实践中，如果现有软件应用程序的底层RDBMS与提供的服务相兼容，那么将任何软件应用程序的数据库层迁移至关系数据库服务中预计不会太困难。但是服务供应商可能因为各种原因带来一些限制或约束（比如，托管数据库的

最大容量，可行并发连接的最大数量）。此外，软件应用程序可能在控制其应用程序的分配资源方面不够灵活（可能动态分配过多的资源来处理不断增长的工作负载，或动态减少分配资源以降低操作成本）。整个资源管理和分配过程由供应商控制，对于数据库层来说，这需要一个准确的分配计算资源计划，并通过利用云计算环境的弹性和可伸缩性来限制用户应用程序功能，从而最大限度地增加其获益。

Sherif指出，不断增长的可伸缩性和新应用程序需求为传统 RDBMS 带来了新的挑战，包括某些大规模网络应用程序对这个“万能”方法的不满。这个问题的解决方法是新一代的低成本、高性能数据库软件，专门设计用于挑战关系数据库管理系统优势。NoSQL 运动的一个主要原因是不同的 Web、企业和云计算应用程序的实现有不同的数据库需求，例如，不是每个应用程序都需要严格的数据一致性。再比如：像 eBay、Amazon、Twitter 或 Facebook 这类大流量网站，对可伸缩性和高可用性都是非常严格的要求，不容妥协。对于这些应用程序，即使是最轻微的故障都会带来重大的经济后果，从而影响顾客信任。

Sherif列举了NoSQL 系统具有的主要优势：

- 弹性伸缩：多年来，数据库管理员一直依赖纵向扩展方法，而不是横向扩展方法；随着目前事务级别和高可用性需求的不断增加，横向扩展（特别是在商用硬件上）的经济优势变得极具吸引力。NoSQL 系统设计具有透明扩展能力，可利用新节点添加优势。
- 更少的管理：NoSQL 数据库通常旨在支持诸如自动修复和简单的数据模型等特性；这会导致较低的管理和调优需求。
- 更好的经济效益：NoSQL 数据库通常使用大量廉价商品服务器来管理激增的数据和事务量。
- 灵活的数据模型：NoSQL 数据库具有更为宽松的数据模型限制（如果有的话），因此应用程序和数据库模式改动可能更小。

但是，NoSQL 数据库仍然需要克服许多障碍，Sherif认为需要改进的地方：

- 编程模型：NoSQL 数据库提供一些专用查询和分析设施。即使一个简单的查询也需要大量的编程知识。由于缺少声明表述支持，重要的join 操作一直被认为是这些系统的主要限制。
- 事务支持：事务管理是功能最强大的一个 RDBMS 特性。目前，对于那些被接受用来实现任务关键系统的事务来说，NoSQL 数据库系统上的事务概念的非存在限制（limited-to-non-existent）支持是一个障碍。
- 成熟性：众所周知，RDBMS 系统具有高稳定性和丰富的功能性。相比之下，大多数 NoSQL 替代方案还处于预生产版本阶段，很多功能尚不稳定，且无法实现。这意味着企业在进入新一波数据管理时仍需谨慎。
- 支持：企业还在寻求保障，如果系统出现故障，那么他们能够得到及时的、足够

的支持。RDBMS 供应商竭尽全力提供高级支持。相比之下，很多 NoSQL 系统是开源项目，目前尚未获得支持。

- 专业人士：几乎每个 NoSQL 开发人员都处于学习阶段；当然，这种状况不会持续太长时间。但是目前寻找经验丰富的 RDBMS 程序员或管理员比寻找 NoSQL 专家更容易。

在11月1日开幕的[QCon全球企业开发大会](#)（上海站），有多个主题与云计算、数据管理相关：

- [扩展性、可用性与高性能](#) 讨论大型复杂系统中，如何在架构设计、代码、运维体系等方面达到扩展性、可用性与高性能。
- [大数据应用](#) 从大数据创业公司和互联网公司的创新数据产品角度来了解大数据的最新应用。
- [云计算架构与案例](#) 实例分析、最佳实践，云计算新时代的创造者为何又如何纷纷将各种应用搬至云端？
- [大数据处理技术](#) 关注大数据处理和分析的最新技术和工具以及未来的技术走向。

原文链接：<http://www.infoq.com/cn/news/2013/10/cloud-nosql>

相关内容

- [从关系数据库向NoSQL迁移：采访Couchbase的产品管理主管Dipti Borkar](#)
- [James Phillips谈从关系型数据库转到NoSQL](#)
- [Google开放公众IaaS云并新增NoSQL数据库](#)
- [架构、云、NoSQL、产品设计等QCon演讲确定，29日前8折优惠报名](#)
- [Peter Bell谈NoSQL的发展趋势](#)

专题推荐语

软件测试的方方面面

前几天看到了一篇关于丰田的消息，由于其动力控制系统软件的质量问题，导致在某些情况下会出现刹车失灵的状况。抛去系统架构、设计等方面的原因不谈，我想到更多的是要对系统的测试引起重视。

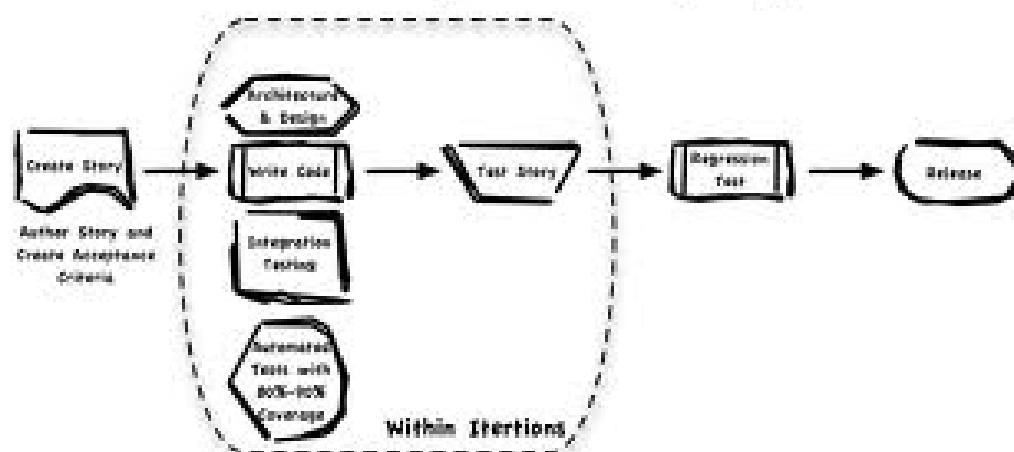
所以本期专栏让我们来看看软件测试的方方面面，我为大家选择了三篇文章，从三个不同的角度来看软件测试的方方面面。

图形化界面上的自动化测试一直困扰着很多测试人员，《GUI功能测试自动化模式》让我们可以了解到，如何使用模式来自动化地完成图形化界面的测试，解决这个老大难问题，从而得到效率上的提升；

测试覆盖率要达到什么样的标准，是不是越高越好呢？《测试覆盖（率）到底有什么用？》会告诉我们应该如何看待“测试覆盖率”这项KPI指标，如何充分利用好时间和精力；

传统软件工程中，测试一直是下游工程，这也导致了很多问题，《全程软件测试实践：从需求到运营》会向我们展示一种不同以往的测试实践，测试人员全程参与到软件的生命周期之中，这样是否能够解决很多问题呢？

Common Agile Testing Approach



本期专题：软件测试的方方面面 | Topic

GUI功能测试自动化模式

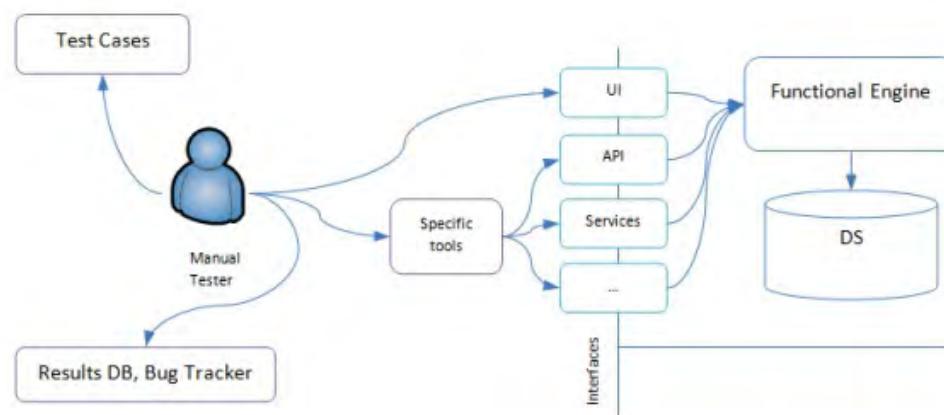
作者 [Oleksandr Reminnyi](#)，译者 李彬

对于某个特定程序，为其开发自动化功能测试解决方案的过程，与创建该程序的过程，二者相较并没有很悬殊的差别。自动化测试是一个非常年轻的领域，它正在不断经历大量的进步、提升和标准化进程。在这个领域中，涌现了许多与“被测系统”（SUT，System Under Test）互动的新工具。

现在，软件开发方面有大量可供选择的方法论和途径，例如：面向对象编程、函数式编程、[领域驱动设计](#)、[测试驱动设计](#)、[行为驱动设计](#)等等。它们拥有明确的声明性概念和理论，并简化了对初始系统架构的定义过程、对系统的理解以及开发者之间的知识交换等方面的工作。

本文将主要针对GUI（图形用户界面）应用的测试自动化进行讨论——从自动化开发人员的角度看，在这种情况下被测系统（SUT）表现为一个黑箱（被测系统，是指一个正在测试是否能够正确操作的系统。对于桌面应用来说，它就是应用本身，而对浏览器系统来说——则代表了网站/Web项目等含义）。在公司的遗留系统占很高比例的环境里，或是在新开发的系统没有考虑可检测质量属性时，这一现象非常常见。

对最佳实践的准备和定义，是开发自动化的测试的关键部分。下图展示了被测系统和测试者之间的传统交互：



测试者与SUT之间的交互

位于该系统中心的，是一个扮演测试者角色的人类个体。测试者使用手动交互和应用的视觉化分析，以及特定的SUT非可视化界面访问工具，将测试用例中所描

绘的场景进行复制。如果失败或是遇到系统意外行为，测试者便将错误行为的有关信息输入到默认的追踪系统中。

自动化测试的主要目的，是消除（或者至少最小化）人类与SUT之间的交互。在持续交付产品开发周期中，这是非常常见的问题。一份文献来源的研究表明，现在自动化测试系统的数量非常多。商业产品一般会公布一系列详细的要求和推荐，特别适用于其产品。但是这些厂家往往不会公布适合与任何自动工具一起使用的一系列工具诊断实践。

此外，这些自动工具软件提供商往往还会运用营销手段，基于小量功能测试来描绘其系统的优势。但是随着自动化测试数量的增长，维护现有测试将成为系统工作流程中最昂贵的部分。

自动化测试框架旨在帮忙解决这些问题。他们定义了基本的系统可复用组件，公布了最佳实践和统一自动方法——为了正确地开发自动化测试框架，我们需要得到独立最佳实践的指导。

自动化功能测试的模式

让我们检查以下Web应用程序自动化方面的问题（图1），这是一个自动化解决方案装置的例子。该应用包含了一幅登录页面，而每项测试都必须经过该页面，才能进行更进一步的测试。

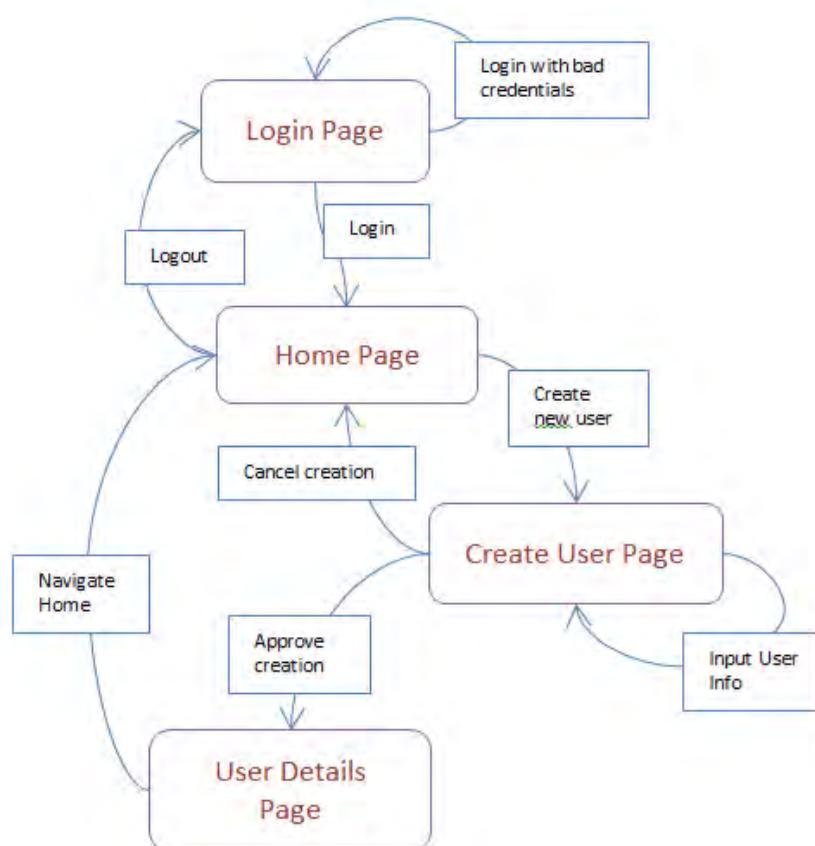


图1：示例——拥有最少页面和功能集的简单Web应用

分类体系（图2）为我们带来了全部功能测试模式的整体视角，本文稍后会对各个模式展开描述。

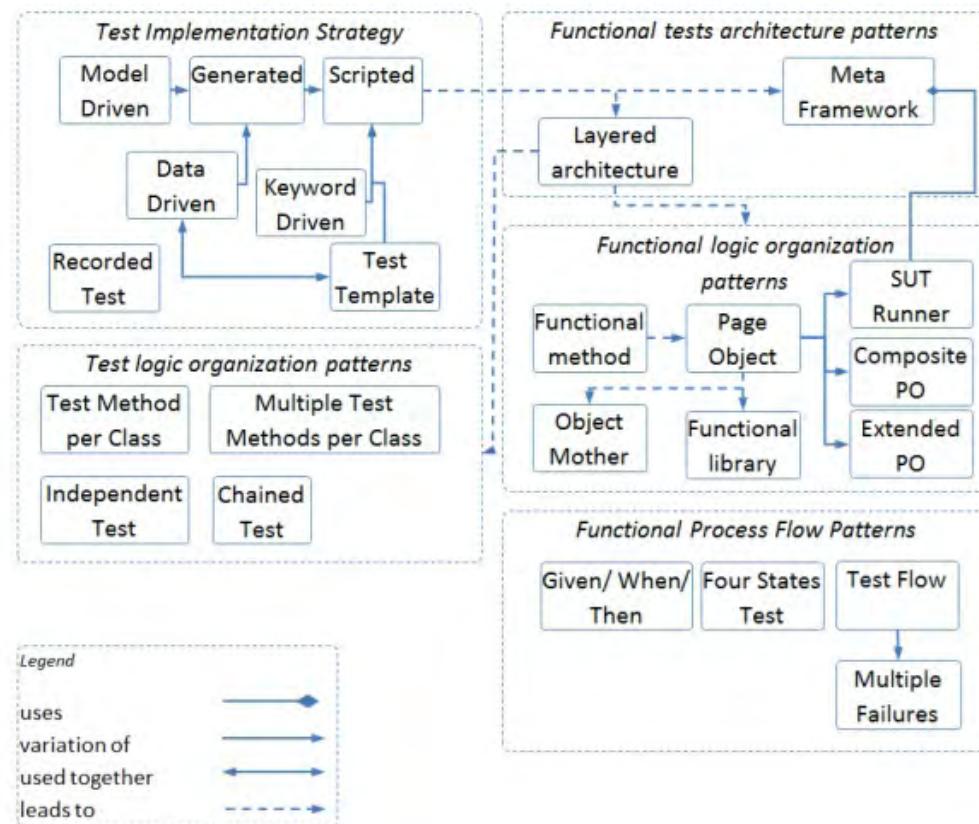


图2：自动化功能测试模式的分类

测试实现的模式

记录

通过自动化测试工具来执行此模式的测试实现。该工具记录并回放手工测试者的操作。某种程度上，考虑到昂贵的维护成本，这种方式被认为是一个糟糕的实践。

脚本化

由程序员使用自动化工具的API来执行测试实现（例如Selenium WebDriver API）。

实现基础模板（测试模板）

该模式将实现基础测试模板类。要实现各种测试的变体，可以通过继承和扩展模

板类的功能来创建。

数据驱动实现

通过测试用例来定义一个基础测试的实现。可以通过一系列不同的输入数据组合，来创建各种测试的变体。

在大部分单元测试框架中都实现了这一方法，例如：MSTest以DataContext（键值集合）的方式，提供对测试内部的属性的访问权限。于是，相同的测试方法主体会运行很多次，但DataContext属性所提供的数据各不相同。

关键字驱动实现

这一测试实现借助了关键字（例如：点击、回车）。测试实现通过特殊的IDE完成，这类IDE可以钩挂到应用的UI上。

目前已有数款软件工具，支持借助关键字来实现测试。测试的步骤，以关键字、屏幕上的控制名和输入参数的结合的形式出现。HP QTP和MonkeyTalk是这类IDE中的典型例子

模型驱动实现

对任何应用来说，在特定的时刻接受特定的输入，都只会有一个特定的状态。根据这样的定义，我们可以将软件程序描绘为有限状态机（有限自动机）。考虑到这一事实，以及状态可用性和转换模型（例如图1），我们可以定义一套特定的页面间转换（工作流）的集合，它将能够覆盖程序的大部分功能。

架构模式

多层的测试解决方案

该模式从逻辑维度将正在测试的系统划分为独立的逻辑层级。

将软件系统以架构性方式分为独立的层级是一个广为流传的实践。第一层封装了表述逻辑，第二层则是业务逻辑层，而第三层负责数据存储。使用这一范式，有助于降低应用维护成本，因为更换每层内部的组件对其他层级的影响将被最小化。而同样的方法也可以运用到测试系统中。

测试代码同样可以分为三层：用来向SUT提供访问的UI自动化工具界面层、功能逻辑层和测试用例层。每一层都肩负特定的责任，而它们都在追求共同的目标——降低测试维护成本，提升创建新测试的便利性。

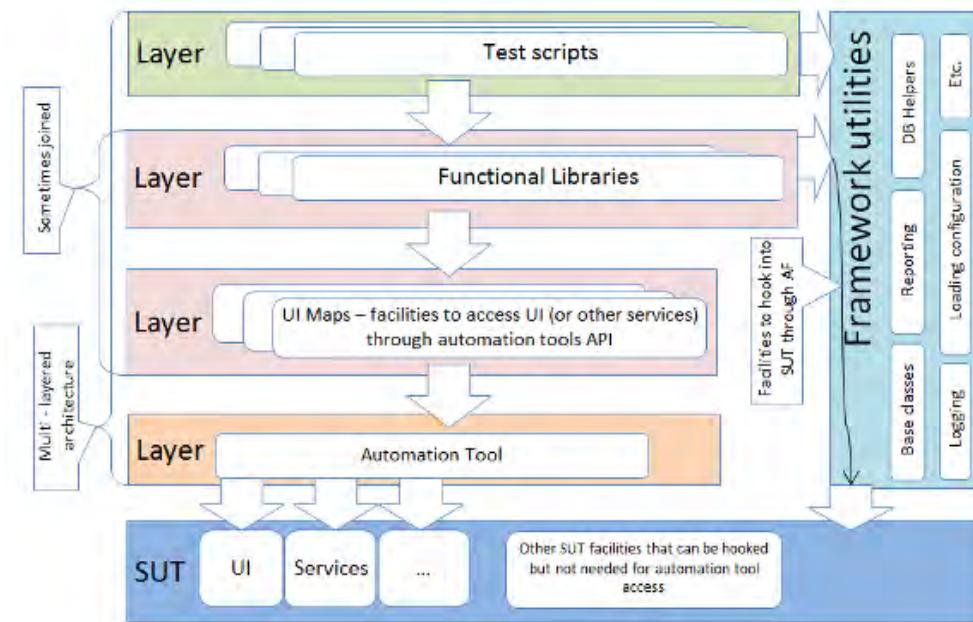


图 3: 架构性原型——测试系统的多层架构

元框架

该模式定义了一组基础的独立工具类。对任何自动化工具来说，它们都是通用的，而且可以在不同自动化项目之间复用。

当需要在一个组织机构中测试不同的项目，而且企业标准要求测试结果采用统一的界面时，或许就需要这样的解决方案。此外，元框架改进了项目间代码复用的度量标准，因为它可能会包含有用的功能方法。有关功能和测试对象的基础类，简化了项目之间知识的转移。元框架展现在图3的右侧。

功能组合模式

功能方法

针对特定于应用的业务功能，从其UI实现、API或其他层面进行了抽象。

用于自动化测试的许多工具，都支持创建被称之为“场景记录”的功能。当一位测试开发者针对特定应用执行特定操作时，这些工具将自动创建一份测试脚本。它可以在稍后进行回放，并检查在程序发生变化后，该脚本是否得到了正确的执行。

例子：改变登录界面的外观，将要求在全部预计的测试场景中的对应部分都进行变更。如果我们将登录方法提取为Application.Login(username,password)，并在全部测试中使用这一方法，那么当登陆页面发生任何变更的时候，我们将只需要修改仅仅这一个功能方法，随后变更就会被自动分发到所有使用该方法的

测试中。

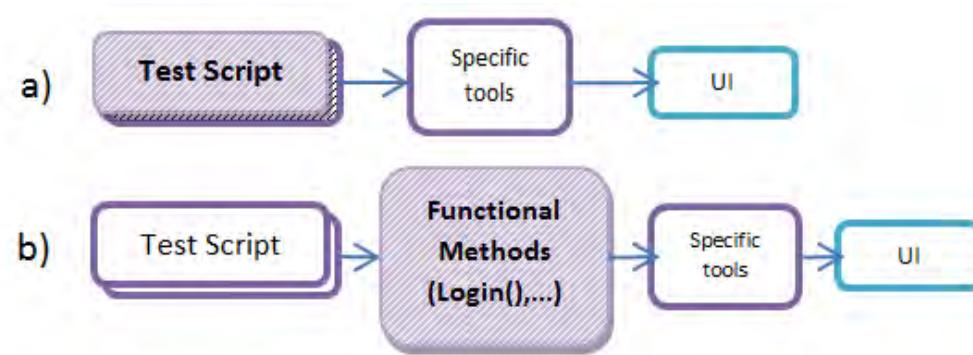


图4：测试脚本与(a)缺乏功能方法的过渡层的用户界面之间的交互；(b) 带有功能方法层的用户界面。当应用发生变更时，阴影部分对象将会被改变。

页面对象

这一模式将某个页面的功能方法组合在一起。

由于图1中展现的用于应用的功能方法数量不多，因此它们可以移入一个单独的类。但是为了提升代码可维护性，此模式建议依据这些方法所代表页面，对其进行分组。例如，这些对应关系可以包括：页面：PageLogin——方法：Login()；页面PageHome——方法：Logout()、CreateUser()。

功能库

它将某个特定应用的功能对象或（和）功能方法分组打包，成为一个适合复用的模块。

SUT的启动和拆卸对象（SUT运行者）

该模式支持被测系统的初始启动，即它的初始化。在此之后，测试对象释放与该系统相关的资源。

在功能方法之中，我们可以区分出与功能测试无关的方法集合：例如启动某个Web浏览器，并访问SUT的登录页面。而在测试之后，Web浏览器应该被关闭。SUT运行者负责以上这些通用活动。

对象源（对象之母、对象精灵、对象工程）

该模式按测试执行所要求的形式，创建并初始化的对象。

运输者（导航器）

根据测试要求，它将被测试系统中的导航控制集中在一起。

该对象封装了与被测试系统内部的导航实现有关的完整逻辑。因此业务逻辑的问题不会影响系统内的导航。

对于图1所描述的情况，我们将拥有一个Transporter类（运输者），它拥有以下方法：NavigateToLogin()、NavigateToHomePage()和NavigateToCreateUser()等等。或者，每个独立的页面（page）对象或许都会拥有自己的运输（transport）方法。在这种情况下，这些方法就作为该对象自身的运输者。

复合页面对象

该模式将复用的页面（page）对象聚合在一个外部对象中。

这一模式支持用更加“面向对象”的方式来组织页面对象——把可以在不同页面上复用的子对象分离，并将其包含到父对象中。



图5：通过在主页和创建用户页面对象中聚集，来使用导航页面（**Navigation Page**）对象

扩展的页面对象

该模式通过继承来扩展基础的页面对象，成为复合页面对象的替代选择。

过程模式

给定条件/时机/结果

此模式将测试执行过程分为三个阶段：

- 给定条件（定义先决条件）
- 时机（设定与上下文协同工作的特定操作）
- 结果（检查结果）

四阶段测试

此模式将测试执行过程分为四个阶段：

- 定义先决条件
- 调用业务功能
- Checking results;
- 拆卸系统

流测试

该模式支持在一个测试内执行业务操作并进行检查。两项内容可以交替进行，直到实现测试的最终目标。

多次失败

该模式定义了一套机制，能够在非致命错误出现之后继续进行测试。

测试依赖模式

独立测试

该模式令被测试的系统回到测试前的状态。

链式测试

在该模式中，初步测试树立起了测试需要跟踪的SUT状态。

测试分组模式

每个测试类封装一个测试方法

在该模式下，每个独立的测试方法，都封装在一个独立的测试类中。

测试类中的分组测试方法

在该模式下，多个测试方法被放置于一个独立的测试类中。

总结

测试解决方案的设计背后的驱动力，是对特定测试实现模式的选择。它扮演了未来所有测试解决方案开发的起点，将在可读性、可维护性和其他诸多特性方面产生影响。而且它也设置了这些实验，使其在能够产生帮助的时候更好地复用项目之间的资源，并减少新项目自动启动的时间。

这篇文章抛出了有关如何参考设计模式，来构建测试解决方案的想法。

参考文献

1. “设计模式：可复用面向对象软件的基础”，作者：Erich Gamma、Richard H elm、Ralph Johnson、John Vlissides，Addison-Wesley Professional于1994年出版；
2. “xUnit测试模式：测试码重构”，作者：Gerard Meszaros，Addison-Wesley于2007年出版；
3. 论文[*Design Patterns for Customer Testing*](#)，作者：Misha Rybalov，一位以质量为中心的开发者；
4. 论文[*Meta-Framework: A New Pattern for Test Automation*](#)，作者：来自Symantec公司Security 2.0团队的Ryan Gerard和Amit Mathur
5. [*Selenium-Wiki*页面](#)

关于作者



Oleksandr Reminnyi是[SoftServe股份有限公司](#)的软件架构师，**SoftServe**是一家全球领先的软件开发、测试和技术咨询服务提供商。**Oleksandr**负责为新客户和已有客户建立自动化项目和流程。他相信自动化的成功与否，完全取决于已经建立起来的流程，以及是否设定正确的目标。**Oleksandr**目前正在致力于完成自动化方面的博士研究。他的联系方式为orem@softserveinc.com。

查看英文原文：[**Functional GUI Testing Automation Patterns**](#)

原文链接：<http://www.infoq.com/cn/articles/gui-automation-patterns>

相关内容

- [Google Espresso：一种用于云中Android UI的快速自动化测试框架](#)
- [采访和书评：Google如何做测试](#)
- [Anthony F. Voellm在Google测试博客上讨论测试2.0](#)
- [软件测试宣言](#)
- [在敏捷项目中实施自动化测试之我见](#)

本期专题：软件测试的方方面面 | Topic

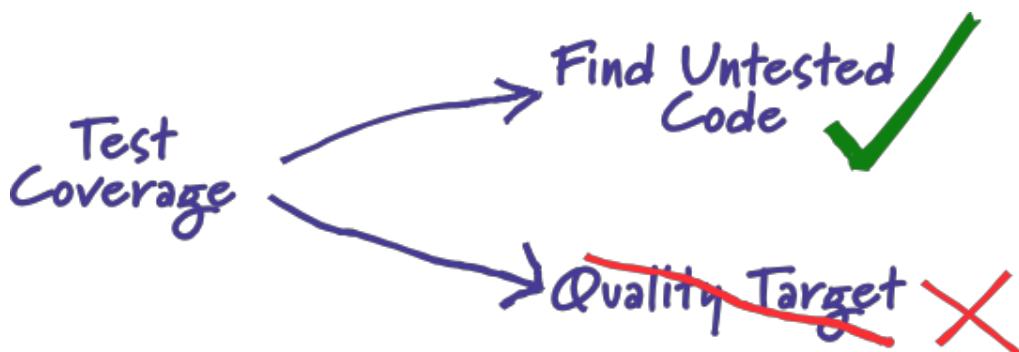
测试覆盖（率）到底有什么用？

引言

作者 姚若舟

经常有人问我这样的问题：“我们在做单元测试，那测试覆盖率要到多少才行？”
。而我的答案很简单，“作为指标的测试覆盖率都是没有用处的。”

Martin Fowler（重构那本书的作者）曾经写过一篇博客来讨论这个问题，他指出：把测试覆盖作为质量目标没有任何意义，而我们应该把它作为一种发现未被测试覆盖的代码的手段。



<http://martinfowler.com/bliki/TestCoverage.html>

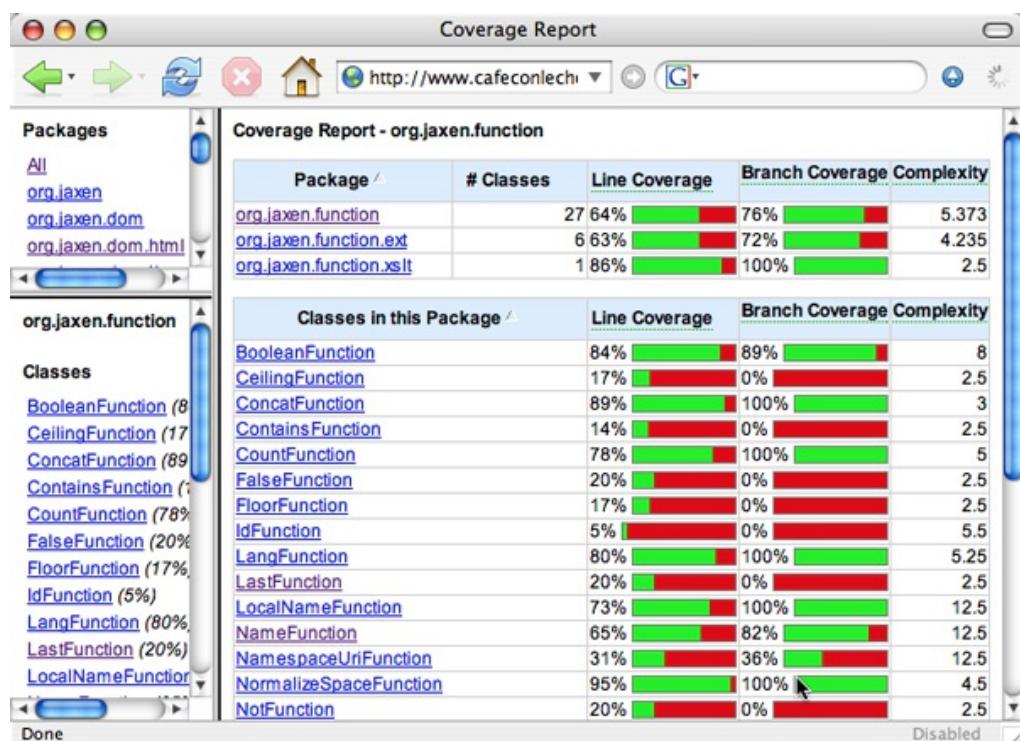
Brian Marick（敏捷宣言最早的17个签署人之一）也说过，作为一名程序员，我当然期望我的代码有较高的测试覆盖率。但是，当我的经理要求这样的指标时，那就有别的目的了（绩效考核？）。

我认为，高的测试覆盖率应该是每个“认真”写单元测试的程序员得到的必然结果，管理者把一个结果作为指标来衡量，本身就是没有意义的。如果你把“万能”的程序员逼急了，他就会从“神秘的工具箱”中拿出一两个“法宝”来，“高效”地达成指标。我就见过很多这样的“法宝”，比如在单元测试中连一个“assert”也没有，或者写很多get和set方法的单元测试（写起来简单啊）来提高整体的覆盖率等等。更何况，测试充分的代码也有可能无法达到100%的覆盖率，本文的后面就有这样的例子。

那你大概会问：“那测试覆盖到底有什么用呢？”。我的答案还是很简单，“测试覆盖是一种学习手段”。学习什么呢？学习为什么有些代码没有被覆盖到，以及为什么有些代码变了测试却没有失败。理解“为什么”背后的原因，程序员就可以做相应的改善和提高，相比凭空想象单元测试的有效性和代码的好坏，这会更加有效。

接下来，我会给大家介绍一些传统的测试覆盖方法和一种称为“代码变异测试”（Mutation Test）的方法。大家将会看到这些方法都可以产生什么样的学习点，以及代码变异测试相比传统方法更有价值的地方。如果你是一名程序员（我不会区分你是开发人员还是测试人员，那对我来说都一样），希望你看完这篇文章之后，可以找到一些提高测试和代码质量的方法。如果你是一位管理者，不论你正在用还是想要用“测试覆盖率”来做度量，希望你看完这篇文章之后，可以放弃这个想法，做点更有意义的事情（比如去写点代码）。

传统的测试覆盖方法



传统的测试覆盖方法常见的有以下几种：

- 函数覆盖（Function Coverage）
- 语句覆盖（Statement Coverage）
- 决策覆盖（Decision Coverage）
- 条件覆盖（Condition Coverage）

还有一些其他覆盖方法，如Modified Condition/Decision Coverage，就不在这里讨论了。

函数覆盖：顾名思义，就是指这个函数是否被测试代码调用了。下面的代码为例，对函数foo要做到覆盖，只要一个测试——如`assertEquals(2, foo(2, 2))`——就可以了。如果连函数覆盖都达不到，那应该想想这个函数是否真的需要了。如果需要的话，那又为什么写不了一个测试呢？

```
int foo (int x, int y){  
    int z = 0;  
    if ((x>0) && (y>0)) {  
        z = x;  
    }  
    return z;  
}
```

语句覆盖：（也称行覆盖），指的是某一行代码是否被测试覆盖了。同样的代码要达到语句覆盖也只需要一个测试就够了，如`assertEquals(2, foo(2, 2))`。但是，如果把测试换成`assertEquals(0, foo(2, -1))`，那就无法达到所有行覆盖的效果了。通常这种情况是由于一些分支语句导致的，因为相应的问题就是“那行代码（以及它所对应的分支）需要吗？”，或者“用什么测试可以覆盖那行代码所代表的分支呢？”。

```
int foo (int x, int y){  
    int z = 0;  
    if ((x>0) && (y>0)) {  
        z = x;  
    }  
    return z;  
}
```

决策覆盖：指的是某一个逻辑分支是否被测试覆盖了。如我上面所说，语句覆盖通常和决策覆盖有关系。还是以上面的代码为例，要达到所有的决策覆盖（即那个if语句为真和假的情况至少出现一次），我们需要至少两个测试，如`assertEquals(2, foo(2, 2))`和`assertEquals(0, foo(-1, 2))`。如果有一个逻辑分支没有被覆盖（比如只有测试`assertEquals(2, foo(2, 2))`），那么我们应该问和上面“语句覆盖”小节中相似的问题。

```
int foo (int x, int y){  
    int z = 0;  
    if ((x>0) && (y>0)) {True/False  
        z = x;  
    }  
    return z;  
}
```

条件覆盖:指的是分支中的每个条件（即与，或，非逻辑运算中的每一个条件判

断)是否被测试覆盖了。之前的代码要达到全部的条件覆盖(也就是 $x > 0$ 和 $y > 0$ 这两个条件为真和假的情况均至少出现一次)需要更多的测试,如`assertEquals(2, foo(2, 2))`,`assertEquals(2, foo(2, -1))`和`assertEquals(2, foo(-1, -1))`。如果有一个条件分支没有被覆盖(比如缺少测试`assertEquals(2, foo(-1, -1))`),那么大家应该想想“那个条件判断是否还需要呢?”,或者“用什么测试可以覆盖那个条件所对应的逻辑呢?”。

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0)) {True/False+
        z = x;
    }
    return z;
}
```

通过上面对几种传统的测试覆盖方法的介绍,大家不难发现,这些方法的确可以帮助我们找到一些显而易见的代码冗余或者测试遗漏的问题。不过,实践证明,这些传统的方法只能产生非常有限的“学习”代码和测试中问题的机会。很多代码和测试的问题即使在达到100%覆盖的情况下也无法发现。然而,我接下来要介绍的“代码变异测试”这种方法则,它可以很好的弥补传统方法的缺点,产生更加有效的“学习”机会。

代码变异测试 (Mutation Test)

```
1 package info.solidsoft.blog.pitest;
2
3 public class FancyClass {
4
5     public boolean doSomethingComplicated(int xFactor, int yFactor) {
6         if (xFactor > 0 || yFactor > 0) {
7             return true;
8         } else {
9             return false;
10        }
11    }
12 }
```

Mutations

```
negated conditional : KILLED -> info.solidsoft.blog.pitest.FancyClassTest_
5 changed conditional boundary : KILLED -> info.solidsoft.blog.pitest.FancyClassTest_
negated conditional : KILLED -> info.solidsoft.blog.pitest.FancyClassTest_
changed conditional boundary : SURVIVED
7 replaced return of integer sized value with (x == 0 ? 1 : 0) : KILLED -> info.solidsoft.blog.pitest.FancyClassTest_
8 replaced return of integer sized value with (x == 0 ? 1 : 0) : KILLED -> info.solidsoft.blog.pitest.FancyClassTest_
```

代码变异测试是通过对代码产生“变异”来帮助我们学习的。“变异”指的是修改一处代码来改变代码行为(当然保证语法的合理性)。简单来说,代码变异测试先试着对代码产生这样的变异,然后运行单元测试,并检查是否有任何测试因为这个代码变异而失败。如果有测试失败,那么说明这个变异被“消灭”了,这是我们期望看到的结果。如果没有测试失败,则说明这个变异“存活”了下来,这种情况下我们就需要去研究一下“为什么”了。

是不是感觉有点绕呢？让我们换个角度来说明一下，可能就容易理解了。测试驱动开发相信大家一定都听说过，它的一个重要观点是，我们应该以最简单的代码来通过测试（刚好够，Just Enough）。基于这个前提，那么几乎所有的代码修改（即“变异”）都应该会改变代码的行为，从而导致测试失败。这样的话，如果有個变异没有导致测试失败，那要么是代码有冗余，要么就是测试不足以发现这个变异。

另一方面，大家可以想一下对于自动化测试（包括单元测试）的期望是什么。我觉得一个很重要的期望就是，自动化测试可以防止“任何”错误的代码修改，以减少代码维护带来的风险。错误的代码修改实际上就是一个代码变异，代码变异测试可以帮助我们找到一些无法被当前测试所防止的潜在错误。

举例来说，我们给之前的那段被测代码增加一行，`sideEffect(z)`。之前的那些可以让传统的测试覆盖方法达到100%覆盖率的测试，在新增这行代码之后，依然会全部通过且覆盖率不变。然而，如果我们再删除那行新代码`sideEffect(z)`，结果会有怎样呢？那些测试还是会全部通过，覆盖率也还是100%。在这种情况下，原来那些测试可以说没有任何意义。相对的，代码变异测试则可以通过删除那一行，再运行测试，就会发现没有任何测试失败。然后，我们就可以根据这个结果想到其实还需要一个测试来验证`sideEffect(z)`这个行为（如果那行代码不是多余的话）。

```
int foo (int x, int y){  
    {  
        int z = 0;  
        if ((x>0) && (y>0)) {  
            z = x;  
        }  
        sideEffect(z);  
        return z;  
    }  
}
```



```
int foo (int x, int y){  
    {  
        int z = 0;  
        if ((x>0) && (y>0)) {  
            z = x;  
        }  
        sideEffect(z);  
        return z;  
    }  
}
```

再举一个例子，还是之前的代码，不做任何修改。我们用`assertEquals(2, foo(2, 2))`, `assertEquals(2, foo(2, -1))`和`assertEquals(2, foo(-1, -1))`这三个测试达到了100%的条件覆盖。然而，如果把 $y > 0$ 的条件改成 $y \geq 0$ 的话，这三个测试依然会通过。为什么会出现这样的问题呢？那是因为之前的测试对输入参数的选择比较随意，所以让这个代码变异存活了下来。可以看到，在条件覆盖100%的情况下，代码变异测试依然可以帮助我们发现这种测试写的不严谨的问题（假设 $y \geq 0$ 这个代码变异是不合理的），从而使修改后的测试可以防止产生这样的错误代码。

```

int foo (int x, int y){  

{  

    int z = 0;  

    if ((x>0) && (y>0)) {  

        z = x;  

    }  

    return z;  

}

```



```

int foo (int x, int y){  

{  

    int z = 0;  

    if ((x>0) && (y>=0)) {  

        z = x;  

    }  

    return z;  

}

```

通过上面两个例子，相信大家已经发现代码变异测试可以给我们提供大量的学习代码合理性和测试有效性的机会。实际上，类似的代码变异还有很多种。下面是常见变异的列表，更详细的内容可以参考 <http://pitest.org/quickstart/mutators/>。

- **条件边界变异 (Conditionals Boundary Mutator)**

对关系运算 (`<`, `<=`, `>`, `>=`) 进行变异，上面第二例子就是这种变异

- **反向条件变异 (Negate Conditionals Mutator)**

对关系运算 (`==`, `!=`, `<`, `<=`, `>`, `>=`) 进行变异，例如把“`==`”变成“`!=`”

- **数学运算变异 (Math Mutator)**

对数学运算 (`+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `>>`, `<<`, `>>>`) 进行变异，例如把“`+`”变成“`-`”

- **增量运算变异 (Increments Mutator)**

对递增或者递减的运算 (`++`, `--`) 进行变异，例如把“`++`”变成“`--`”

- **负值翻转变异 (Invert Negatives Mutator)**

对负数表示的变量进行变异，例如把“`return -i`”变成“`return i`”

- **内联常量变异 (Inline Constant Mutator)**

对代码中用到的常量数字进行变异，例如把“`int i=42`”变成“`int i=43`”

- **返回值变异 (Return Values Mutator)**

对代码中的返回值进行变异，例如把“`return 0`”变成“`return 1`”或者把“`return new Object();`”变成“`new Object(); return null;`”

- **无返回值方法调用变异 (Void Method Calls Mutator)**

对代码中的无返回值方法调用进行变异，也就是把那个方法调用删除掉，

上面的第一个例子就是这种变异。

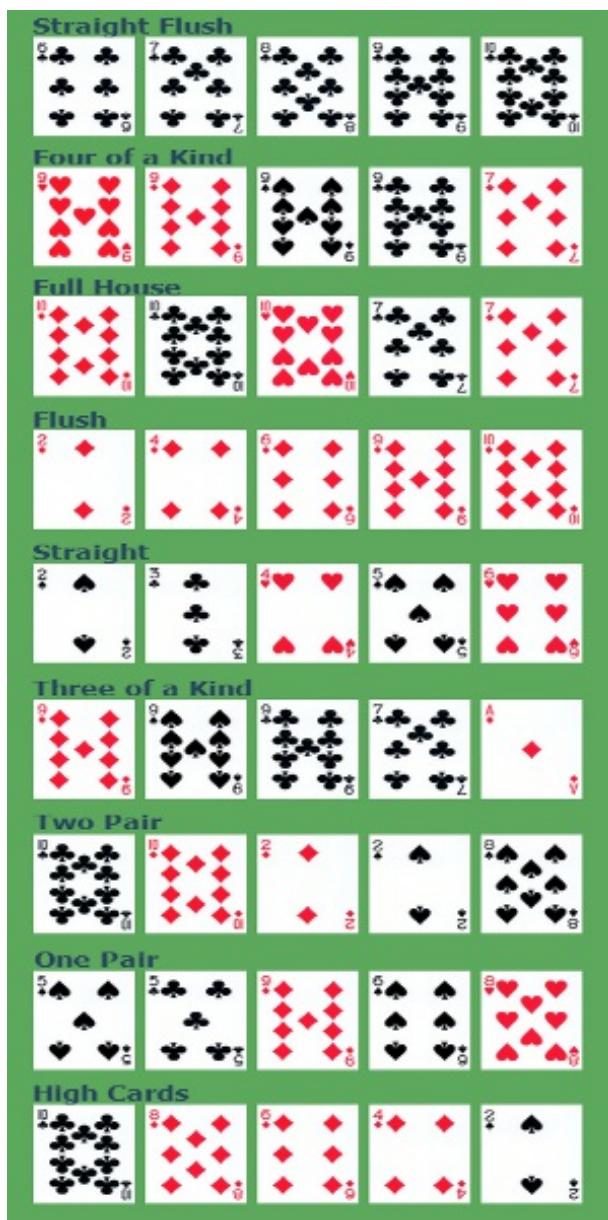
- 有返回值方法调用变异 (**Non Void Method Calls Mutator**)

对代码中的有返回值函数调用进行变异，也就是接收返回值的变量赋值将被替换成为返回值类型的语言默认值，例如把“int i = getSomeIntValue()”变成“int i = 0”

- 构造函数调用变异 (**Constructor Calls Mutator**)

对代码中的构造函数调用进行变异，例如把“Object o = new Object()”变成“Object o == null”

测试驱动开发和代码变异测试



测试驱动开发 (TDD) 是我推崇和实践的写代码 (做设计) 方法。我在前面曾经

提到，代码变异测试的假设是“实现代码是刚好够通过测试的最简单代码”，而这也是TDD中的重要实践之一。大家可能会问，如果做了TDD，代码变异测试的结果又会如何呢？还会产生学习的机会吗？答案是肯定的，一定会。

让我们通过例子来看一下。我经常会做一些Kata来练习编程技巧，PokerHands（如上图）就是其中之一（其实大体就是实现梭哈的五张比较规则<http://codin dojo.org/cgi-bin/wiki.pl?KataPokerHands>）。每次我把Kata做完之后，都会用运行一下代码变异测试（sonar中有插件）。Java的代码变异测试工具有个比较好的叫pitest。下面是我用这个工具跑出来的结果，代码可以在这里找到<https://github.com/JosephYao/Kata-PokerHands>。

如大家所见，红色那一行中有一个存活下来的代码变异。而这个代码变异是把“index < CARD_COUNT - 1”中的“<”换成“>”。看上去很不可思议吧，因为进行这样的代码变异意味着整个for循环都不会被执行了，应该不可能没有一个测试失败吧？

PokerHands

AbstractPairPokerHands

Coverage Dependencies Duplications LCOM4 Mutations Coverage Source Violations

1 violations Blocker: 0 Critical: 0 Major: 1 Minor: 0 Info: 0

Full source Time changes... Survived mutant (1)

```

26
27     private List<Integer> getPairCardRanks(List<Integer> cardRanks) {
28         List<Integer> result = new ArrayList<>();
29
30         for (int index = 0; index < CARD_COUNT - 1; index++)
31             if (isTwoNeighborCardRanksEquals(index, cardRanks))
32                 result.add(cardRanks.get(index));
33
34     return result;

```

Survived mutant | 10 minutes

A conditional expression has been negated without breaking the tests

Comment Assign False-positive More actions

让我们来看一下相关的单元测试。在下面这个测试中有三个assert，它们都是在验证“一对”之间通过对子的点数来比较大小的情况。大家仔细观察就可以发现，其实这三个assert中的牌如果作为High Card（就是比一对小一点的牌组）来比较的话，也都是成立的。这也就是那个代码变异可以存活下来的原因，因为即使忽略了一对之间的比较，通过High Card比较出来的大小关系也是一样的。我从中学到的是，只要把 assertPokerHandsLargerThan("2S 3H 5S 8C 8D", "2S 3H 5S 7C 7D") 改为 assertPokerHandsLargerThan("2S 3H 5S 8C 8D", "2S 3H 9S 7C 7D") 就可以清除这个代码变异了。

```

@Test public void pair_compare_to_pair_by_pair_card_rank() {
    assertPokerHandsSmallerThan("2S 3H 6C 6D 8S", "2S 3H 5S 7C 7D");
    assertPokerHandsLargerThan("2S 3H 5S 8C 8D", "2S 3H 5S 7C 7D");
    assertPokerHandsLargerThan("2S 3H 8S 8C 9D", "2S 7H 7S 8C 9D");
}

```

从这个例子中可以看到，即使以TDD的方法来写代码，也是无法完全避免出现代码变异存活下来的情况的（当然，存活变异的数量要非常明显的少于不用TDD而写出来的代码）。做过TDD的人可能都有这样的感觉，就是有时很难抑制自己写出复杂代码的冲动（也就是说代码不是“刚好够”的）。有时，即使实现代码是最简单的，也可能因为代码过于直接，就会很“随意”的写出一个让当前代码失败的测试。上面的例子就是这种情况，这样不太“有效”的测试通常在TDD过程中很难意识到，从而给之后的代码维护造成隐患。

除了上面那个有学习意义的代码变异之外，其实工具还帮我找到了一个“没意义”但存活下来的代码变异。

The screenshot shows a mutation analysis interface for a Java file named `PokerHands`. The main navigation bar includes `Coverage`, `Dependencies`, `Duplications`, `LCOM4`, `Mutations Coverage`, `Source`, and `Violations`. Below the navigation bar, a summary bar displays **6 violations** with categories: Blocker: 0, Critical: 0, Major: 3, Minor: 3, Info: 0. A dropdown menu shows "Survived mutant (1)". The code editor shows a portion of a `getThreeOfAKindCardRank` method:

```

86     return cardRanks.get(index) == cardRanks.get(index + 1);
87 }
88
89 protected Integer getThreeOfAKindCardRank(List<Integer> cardRanks) {
90     for (int index = 0; index < CARD_COUNT - 2; index++)

```

A tooltip for the line `index < CARD_COUNT - 2` indicates it is a **Survived mutant** that has been active for 6 days. The tooltip text reads: "A relational operator has been replaced by a boundary counterpart without breaking the tests". Below the code editor, there are buttons for `Comment`, `Assign`, `False-positive`, and `More actions`.

这里存活下来的代码变异是指把“`index < CARD_COUNT - 2`”中的“`<`”变成“`<=`”。之所以说这个代码变异没意义，是因为根据代码上下文，在for循环中一定会在`index`等于`CARD_COUNT - 2`之前就找到那个三张的点数。因为工具无法理解上下文，所以产生了这个没意义的代码变异（也叫做Equivalent Mutation）。之所以举这个例子，只是想提醒大家不要迷信代码变异测试工具。对于他产生的结果一定去分析和学习，不然很容易走上考核指标的那条不归路。

小结

总而言之，测试覆盖这种方法是一种不错的学习手段，可以帮助我们提高代码和测试质量。代码变异测试则比传统的测试覆盖方法可以更加有效的发现代码和测

试中潜在的问题，提供更多的学习机会。在这里，我要郑重警告那些妄图把代码变异测试变成一种新的考核指标的管理者们，这样做只会迫使程序员从他的神秘工具箱中找出新的法宝来对付你（比如，修改编译器等等）。

代码变异测试的概念其实早在30年前就被提出了。之所以到目前为止还没有被业界广泛接纳，一个重要原因是由于需要对每个代码变异反复运行测试。如果不是单元测试（运行速度慢），代码变异测试工具执行时将消耗大量的时间。正因如此，单元测试可能是唯一符合代码变异测试要求的一种测试了。如果你对代码变异测试的历史和发展过程感兴趣的话，你可以参考这篇研究报告 http://crestweb.cs.ucl.ac.uk/resources/mutation_testing_repository/TR-09-06.pdf。

关于姚若舟

姚若舟是Odd-e的一名敏捷教练，为团队提供敏捷实践的教导和培训。他在软件业有超过十三年的开发和项目管理经验，熟悉互联网，移动和桌面软件的开发。他是中国敏捷社区的积极参与者和组织者。作为一名追求软件工艺的程序员，他从2011年开始坚持每天通过Coding Kata来不断提高自己的编程技巧。同时，他还在不少公司，社区沙龙和会议中组织过许多次代码道场（Coding Dojo）和Coderetreat的活动。

他有非常丰富的Scrum经验，曾长期担任ScrumMaster，服务于多个开发团队和Product Owner。他擅于在各种Scrum会议和日常工作中通过提问来引导和帮助团队成员发现潜在的问题和改善机会，从而真正实现团队的敏捷转型。

他对软件工艺的各类实践（如编写高可读性代码，单元测试，重构，遗留代码隔离，测试驱动开发（TDD）等）有着深入的了解和丰富的实战经验。同时擅于通过Coding Dojo这种编程练习的方式来帮助开发人员提高编程能力和代码质量。他精通Java语言，熟悉基于Java的Web开发中的常用协议，开源软件和工具。

Kata视频: <http://tudou.com/home/yaoruzhou>

新浪微博: <http://weibo.com/yaoruzhou>

Github: <https://github.com/JosephYao>

感谢[侯伯薇](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/test-coverage-rate-r>

ole

相关内容

- [Google Espresso：一种用于云中Android UI的快速自动化测试框架](#)
- [GUI功能测试自动化模式](#)
- [性能提速-百亿级性能测试](#)
- [测试专栏主编：侯伯薇](#)
- [博文共赏：SAP自动化准备测试数据：一个基于AutoIt VBS XML的实现思路](#)

本期专题：软件测试的方方面面 | Topic

全程软件测试实践：从需求到运营

作者 李乐

之前一篇文章《[软件测试转型之路](#)》介绍过我们转型的一些实践，下文将介绍从2011年3月至今，持续改进的全程软件测试实践活动。

1 全程软件测试图解

传统的软件测试，可以简单描述为下图所示：



图-1-传统交付测试

开发人员完成任务之后，最后交付给测试人员，这种模式下，测试人员不能及早发现需求阶段的缺陷，同时测试工作的开展也滞后了，产品质量得不到有效的过程控制和分析，总体进度可能会由于返工问题造成拖延。

那什么是全程软件测试，如下图所示：

(点击图像放大)

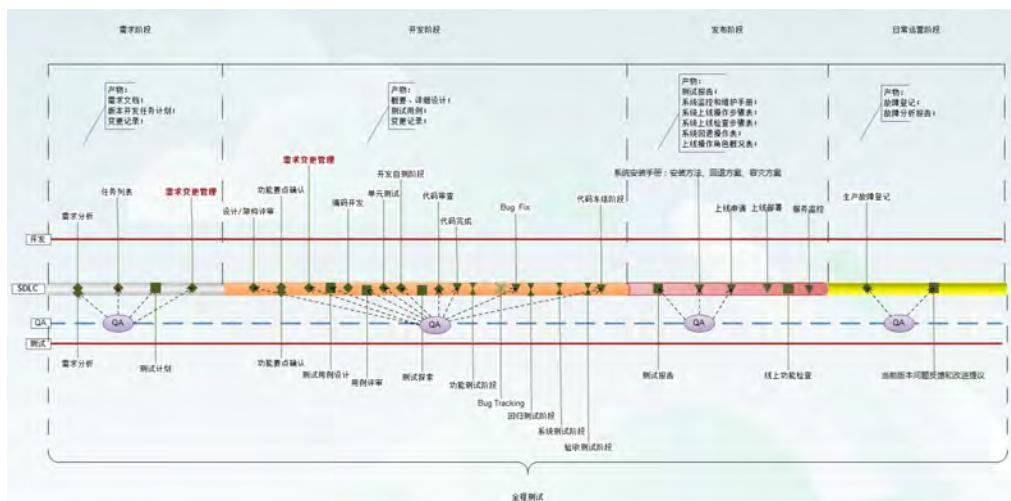


图-2-全程软件测试图

在整个SDLC中，三条角色主线和四个阶段。

三条角色主线：开发、QA、测试，文中主要讲解测试。

四个阶段：需求、开发、发布、日常运营。

简单来说可以归纳为下图所示：



图-3-全程软件测试概述

测试人员贯穿这四个阶段，开展测试活动，试实践活动简单描述如下图所示：

(点击图像放大)



图-4-全程软件测试概述

每个阶段也有开发人员对应的活动，以及QA人员对应的活动。

对于产品而言，每次版本迭代，都会经历：需求、开发、发布，最后推向日常运营，发布阶段虚线指向的需求阶段和日常运营阶段，并不是一个终止阶段，而是不断迭代的过程。

那测试人员是如何开展全程软件测试活动的呢？

2 需求阶段测试

在需求阶段，开发人员、测试人员、QA人员主要做的事情，如下表所示：

阶段	开发人员	测试人员	QA人员
需求阶段	<ul style="list-style-type: none"> • 用户故事分析 • 用户故事估时 	<ul style="list-style-type: none"> • 参与用户故事分析、挖掘故事含混性 • 参考经验库质疑开发的时间估算 	<ul style="list-style-type: none"> • 保证确认需求活动符合需求管理过程 • 管理用户故事评审 • 管理需求变更

作为测试人员的主要实践如下：

参与用户故事分析、挖掘故事含混性

在sprint会议上，对用户故事进行分析，检查功能性需求和非功能性需求是否描述清晰，其中可以将非功能性需求作为验收要点，例如一个用户故事：

“客户希望提高响应时间”

测试人员应当协助开发人员消除故事的含混性：提高什么的响应时间和响应时间为多少？可以建议修改为：

“客户信息普通查询返回结果的响应时间为5s内”

说明在“客户信息”模块，进行“普通查询”操作，返回结果的时间在5s内，这个陈述句已经清晰表达了，也达到了消除含混性的效果。同样，测试人员可以编写提高查询效率的用户故事：

“客户在信息查询模块，进行普通查询，能够在5s内返回结果”

“备注：5s为非功能性需求，也是验收要点”

参考经验库质疑开发的时间估算

在sprint会议上，开发人员根据经验出牌（团队自己定义的规则，用扑克牌）估算时间，当给出最终结果的时候，测试人员应当对其进行质疑。测试人员借鉴历史经验库：开发人员在某方面的技能如何、该模块曾经产生过何种程度的缺陷、修复缺陷的消耗时间是多少等等，综合考虑，提出疑问，让开发估算最终的时间，尽可能考虑这些因素。当然，测试人员能够质疑的其中一个前提是：测试人员具备相关开发经验。

小结：在需求阶段，测试人员要发挥作用，减少含混性需求引入到开发阶段、同

时协助开发做好时间估算。

3 开发阶段测试

在开发阶段，开发人员、测试人员、QA人员主要做的事情，如下表所示：

阶段	开发人员	测试人员	QA人员
开发阶段	<ul style="list-style-type: none"> • 架构评审 • 功能要点确认 • 编码开发 • 单元测试 • 开发自测 • 代码评审 • Bug Fix 	<ul style="list-style-type: none"> • 功能要点确认 • 测试用例设计 • 用例评审 • 测试探索 • 功能测试 • Bug Tracking • 回归测试 • 系统测试 • 验收测试 	<ul style="list-style-type: none"> • 管理评审活动 • 管理文档产物

作为测试人员的主要实践如下：

功能要点确认

Xmind是一个非常好用的脑图工具，通常在开发人员进行编码前，测试人员会针对需求处理的用户故事，与开发人员进行确认，修正理解偏差，确保需求理解一致。



图-5-脑图用例模板

测试用例设计

测试人员主要设计测试故事点，使用DSL(Domain Specific language)，infoq文章（[敏捷测试之借力DSL](#)），对测试用例进行描述，包括三个基本要素：

Feature、Scenario、Example，补充要素：xmind、Requirement。

Feature：把测试分类到某个模块，并对这个特性本身的业务目的进行相关描述

，带进业务目标，传递业务知识。

Scenario: 标明这个Feature的测试场景，可以使用文字描述步骤，或者使用x mind脑图

描述，场景中的数据使用Examples中列出的。

Example: 引出具体的数据表格把用到的数据都展示出来，避免相同步骤因为测试数据的变化而重复若干遍造成冗余。

Xmind: 脑图文件，展示测试故事点

Requirement: 关联需求管理系统的需求id。

用例评审

主要是坚持同行评审的原则，主要在测试组内进行，负责该任务的开发人员也会参与，简单来说就是对测试用例进行查漏补缺的工作。

测试探索

进行了“功能要点确认”和“用例评审”后，为了保证测试场景的覆盖率，需要再进行测试探索。在开发人员完成雏形之后，使用探索式测试的策略，对功能基本流程进行有目的的快速走查，挖掘功能不确定的地方和补充测试场景，避免不确定的因素拖延到开发阶段后期，造成返工。

其中：功能测试、Bug Tracking、回归测试、系统测试、验收测试都是日常测试工作所需环节。

燃尽图发布

另外，测试人员还有一项重要工作，每日发布燃尽图，让团队了解当前进度情况，总结问题

所在，寻求耗时超过预期时间任务的解决办法。



图-6-燃尽图

图形特点：

1) 剩余工时在计划基准上方，代表进度有所延迟，应抓紧进度；

发现此类问题，需要分析总结，原则是保证交付时间，对相应任务进行调整，拥抱变化，发现任务粒度太大，该拆分的继续拆分；对于重构需要慎重，不要过度深入重构，给测试带来额外工作量，影响整个进度，对于整个版本而言，只有开发、测试在承诺的时间内完成任务，才是真正完成，仅仅开发完成交付算不上成功。

2) 剩余工时在计划基准接近，代表进展良好，继续保持；

此时也需要查看在这种进度下，优先级高的任务是否得到时间保证，而不是因为处理完简单任务才使得燃尽图长的好看。往往有些开发人员，喜欢挑着任务来做，把简单易做、优先级的任务先完成了，因为这些总在预期内能够完成，所以前期燃尽图的趋势看起来没有问题。

缺陷经验库

每个团队都存在开发/测试新人和开发/测试老人，当测试人员与开发新人进行需求确认的时候，还需要进行缺陷经验教训的提醒，避免多走弯路。

(点击图像放大)



图-7-缺陷总结

提升开发自测质量

测试人员可以提供相关[checklist](#)（大家可以根据原作者提供的修改为符合团队的）帮助开发人员在编码过程中关注开发自测的要点，从而提升质量。

(点击图像放大)





图-8-web软件测试checklist

持续集成

利用持续集成（Jenkins）平台，做到快速的构建开发代码，自动的单元测试化，来提高开发代码的效率和质量。

负责单元测试的开发人员，会收到失败构建的邮件；

负责集成测试的开发人员，会收到失败构建的邮件；

负责自动化测试（Selenium）的测试负责人，会收到失败构建的邮件；

这种方式，确保单元测试、集成测试、自动化测试，有相关人员关注和维护。

（点击图像放大）

The Jenkins dashboard displays the status of four projects: portal_inte_test, portal_selenium, portal_trunk, and portal_web_test. All projects are currently green (successful). The portal_trunk project has the most recent successful build at 3 hours 32 minutes ago. The portal_web_test project has the shortest duration at 2 hours 32 minutes.

图-9-持续集成

Sonar反馈

Sonar is an open platform to manage code quality. As such, it covers the 7 axes of code quality.

(点击图像放大)



图-10-sonar分析结果

测试人员主要反馈问题如下：

Code coverage: 团队要求代码覆盖率在80%以上；

Test success: 团队要求测试成功率在100%；

Duplications: 团队要求代码重复率在10%以下；

Violations: 团队要求Major类别的代码规则缺陷在20以下；

开发团队必须保证每个环境的质量目标，才能够保证整个的质量目标。

小结：

测试人员与开发人员永远不是敌对关系，而是协助关系，确切来说是质量天枰的两边，任何一边的工作没有做好，都会失去平衡。

4 发布阶段测试

在发布阶段，开发人员、测试人员、QA人员主要做的事情，如下表所示：

阶段	开发人员	测试人员	QA人员
发布阶段	<ul style="list-style-type: none"> • 上线申请 • 上线部署 • 服务监控 	<ul style="list-style-type: none"> • 测试报告 • 线上功能检查 	<ul style="list-style-type: none"> • 管理评审活动 • 管理文档产物

作为测试人员的主要实践如下：

测试报告

完成验收测试，提供测试报告，给出测试数据度量，例如：

- **测试发现缺陷总数：** 测试过程中产生的去除状态为“无效”、“不用改”的缺陷数目。
- **测试发现严重缺陷数：** 测试过程中产生的并去除状态为“无效”、“不用改”的、且严重性为“Major”和“Critical”的缺陷总数目。
- **测试发现缺陷修复数：** 测试过程中产生的状态为“已关闭”的缺陷数量；
- **未解决缺陷数：** 去除状态为“无效”、“不用改”、“关闭”的缺陷总数。
- **缺陷修复率：** $(\text{测试发现缺陷的修复数}) \div (\text{测试发现缺陷总数}) \times 100\%$
- **严重缺陷率：** $(\text{测试发现严重缺陷数}) \div (\text{测试发现缺陷总数}) \times 100\%$
- **严重缺陷修复率：** $(\text{已修复的严重缺陷数}) \div (\text{测试发现严重缺陷数}) \times 100\%$
- **测试需求覆盖率：** $\text{已测试需求个数} \div \text{需求总数} \times 100\%$

缺陷统计分析报告

另外，测试人员还有一项重要工作，对当前版本的缺陷进行统计分析：

按缺陷级别统计：

	Critical	Major	Medium	Minor	总计
首页	0	0	1	0	1
模块一	0	0	0	2	2
模块二	0	1	2	10	13
模块三	0	0	1	4	5
模块四	0	0	1	2	3
模块五	0	0	3	2	5
模块六	0	1	0	1	2
模块七	0	2	0	6	8

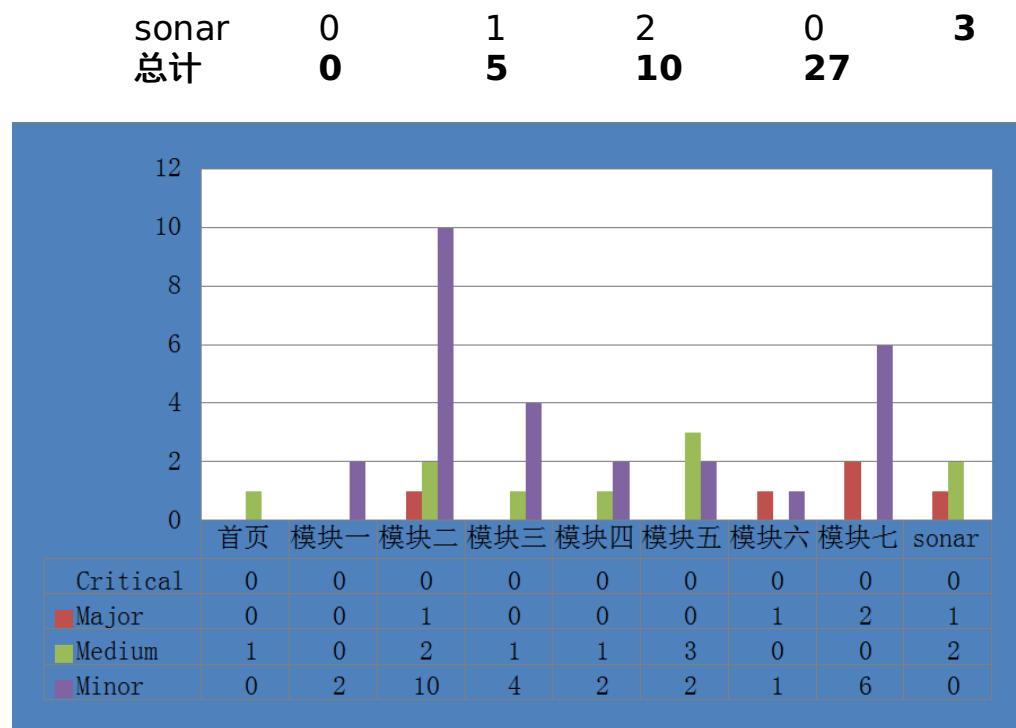


图-11-缺陷统计

按缺陷来源统计：

	开发1	开发2	开发3	开发4	开发5	遗留
Critical	0	0	0	0	0	0
Major	1	2	0	0	0	2
Medium	1	7	0	1	0	1
Minor	1	7	4	6	3	6
总计	3	16	4	7	3	9

按缺陷状态统计：

缺陷总数	已关闭缺陷数	遗留	缺陷修复率	严重缺陷数	严重缺陷率	已关闭严重缺陷数	严重缺陷修复率
42	40	2	95%	5	12%	5	100%

测试进度和问题分析：

- 从BUG的严重级别分布来看，Major级别的BUG占12%，占的比重不高，说明大部分的主要功能已经实现了；
- 其中在sonar定义级别的缺陷，主要集中在代码规范和单元测试覆盖率，说明代码质量有待提高；
- 版本测试的前期时间较充足，后期随着开发提交完成的功能点增多，BUG数量增多，剩余测试时间变得紧张；
- 在版本测试期间，发现测试环境存在一次代码被覆盖、两次因开发人员操作失误影响测试执行的情况；

小结：

测试人员应当持续反馈、改进、总结每个版本发生的问题（不管是缺陷，还是过程中出现的），并对缺陷进行分析，总结出一些规律，帮助开发人员建立良好的习惯，改进代码的质量。

5 日常运营阶段测试

在日常运营阶段，开发人员、测试人员、QA人员主要做的事情，如下表所示：

阶段	开发人员	测试人员	QA人员
日常运营	生产故障登记	<ul style="list-style-type: none">• 版本问题反馈和改进提议• 生产故障分析	管理日常运营活动

日常运营阶段，并不是终止阶段，即便需求、开发、发布阶段暂停活动，只要产品提供服务，日常运营都存在着。

作为测试人员的主要实践如下：

版本问题反馈和改进提议

对日常运营发生的问题，总结反馈，提出改进建议，并且跟踪实施。

生产故障分析

协助开发排查生产故障，避免测试场景的遗漏。

6 总结

软件测试并不是保证产品质量的最后一道防线，测试人员也不是，测试人员的工作完全可以由更加资深的开发人员来完成，不过现实总是残酷的，目前测试与开发的比例为：1:3，在成熟的团队是这样子，另外一些还在持续改进的团队，由于资源不足，可能去到1:7。开发人员在相当长的一段时间内不可能完全替代测试人员，有个关键要素：思维方式不同，有句古话来形容：江山易改本性难移。当开发人员的思维方式改变的时候，那就成为测试人员了，倒不如把测试人员独立出来更好，并且培养给开发人员一定的测试素养，这个对保证产品质量都是有帮助的。

全程软件测试实践，强调的是贯穿每个阶段的测试活动，不论是开发、还是测试，要理解双方的活动价值，什么时候该做什么事情，什么事情该做到什么程度才算好，保证每个环节的质量，才能够保证产品的全程质量，另外产品质量不是测试出来的，而是构建过程中沉淀下来的，开发人员的素养、测试人员的素养、以

及团队对开发测试过程的重视程度，决定了产品质量。产品质量就如同一块蛋糕，应当切分为小块，落实到每个人手里，让每个人尝到甜头，担当起来。

作者简介

李乐（[博客](#)，[微博](#)），测试经理，6年以上工作经验，目前就职于ChinaCache质量部。

原文链接：<http://www.infoq.com/cn/articles/whole-software-testing-practice-requirements-to-operational>

相关内容

- [软件测试中的黑天鹅（三◆◆——测试的平均斯坦与极端斯坦）](#)
- [软件测试魅力何在——QCon北京测试专题出品人高楼访谈](#)
- [Web软件测试中数据输入的检查清单](#)
- [软件测试宣言](#)
- [腾讯云计算的设计与持续运营](#)

推荐文章 | Article

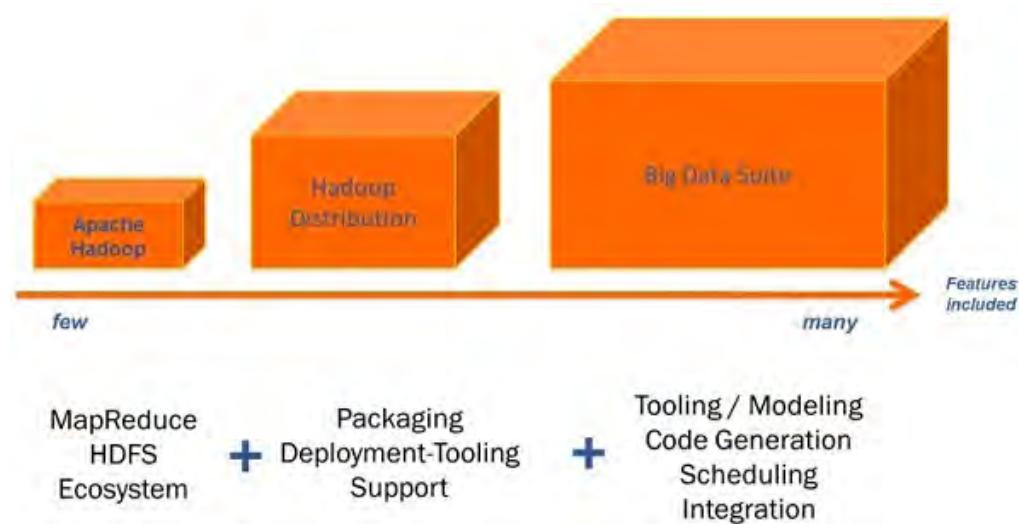
太多选择——如何挑选合适的大数据或Hadoop平台？

作者 [Kai W  hner](#)，译者 王灵军

今年，大数据在很多公司都成为相关话题。虽然没有一个标准的定义来解释何为“大数据”，但在处理大数据上，Hadoop已经成为事实上的标准。IBM、Oracle、SAP、甚至Microsoft等几乎所有的大型软件提供商都采用了Hadoop。然而，当你已经决定要使用Hadoop来处理大数据时，首先碰到的问题就是如何开始以及选择哪一种产品。你有多种选择来安装Hadoop的一个版本并实现大数据处理。本文讨论了不同的选择，并推荐了每种选择的适用场合。

Hadoop平台的多种选择

下图展示了Hadoop平台的多种选择。你可以只安装Apache发布版本，或从不同提供商所提供的几个发行版本中选择一个，或决定使用某个大数据套件。每个发行版本都包含有Apache Hadoop，而几乎每个大数据套件都包含或使用了一个发行版本，理解这一点是很重要的。



下面我们首先从Apache Hadoop开始来好好看看每种选择。

Apache Hadoop

Apache Hadoop项目的目前版本（2.0版）含有以下模块：

- **Hadoop通用模块**：支持其他Hadoop模块的通用工具集。
- **Hadoop分布式文件系统（HDFS）**：支持对应用数据高吞吐量访问的分布式

文件系统。

- **Hadoop YARN**: 用于作业调度和集群资源管理的框架。
- **Hadoop MapReduce**: 基于YARN的大数据并行处理系统。

在本地系统上独立安装Apache Hadoop是非常容易的（只需解压缩并设置某些环境变量，然后就可以开始使用了）。但是这仅适用于入门和做一些基本的教程学习。

如果你想在一个或多个“真正的节点”上安装Apache Hadoop，那就复杂多了。

问题1：复杂的集群设置

你可以使用伪分布式模式在单个节点上模拟多节点的安装。你可以在单台服务器上模拟在多台不同服务器上的安装。就算是在该模式下，你也要做大量的配置工作。如果你想设置一个由几个节点组成的集群，毫无疑问，该过程就变得更为复杂了。要是你是一个新手管理员，那么你就不得不在用户权限、访问权限等诸如此类的问题中痛苦挣扎。

问题2：Hadoop生态系统的使用

在Apache中，所有项目之间都是相互独立的。这是很好的一点！不过Hadoop生态系统除了包含Hadoop外，还包含了很多其他Apache项目：

- **Pig**: 分析大数据集的一个平台，该平台由一种表达数据分析程序的高级语言和对这些程序进行评估的基础设施一起组成。
- **Hive**: 用于Hadoop的一个数据仓库系统，它提供了类似于SQL的查询语言，通过使用该语言，可以方便地进行数据汇总，特定查询以及分析存放在Hadoop兼容文件系统中的大数据。
- **Hbase**: 一种分布的、可伸缩的、大数据储存库，支持随机、实时读/写访问。
- **Sqoop**: 为高效传输批量数据而设计的一种工具，其用于Apache Hadoop和结构化数据储存库如关系数据库之间的数据传输。
- **Flume**: 一种分布式的、可靠的、可用的服务，其用于高效地搜集、汇总、移动大量日志数据。
- **ZooKeeper**: 一种集中服务，其用于维护配置信息，命名，提供分布式同步，以及提供分组服务。
- 还有其他一些项目。

你需要安装这些项目，并手动地将它们集成到Hadoop中。

你需要自己留意不同的版本和发布版本。不幸的是，不是所有的版本都能在一起完美地运行起来。你要自己比较发布说明并找出解决之道。Hadoop提供了众多的不同版本、分支、特性等等。跟你从其他项目了解的1.0、1.1、2.0这些版本

号不同，Hadoop的版本可远没这么简单。如果你想更进一步了解关于“Hadoop版本地狱”的细节，请阅读“[大象的家谱 \(Genealogy of elephants \)](#)”一文。

问题3：商业支持

Apache Hadoop只是一个开源项目。这当然有很多益处。你可以访问和更改源码。实际上有些公司使用并扩展了基础代码，还添加了新的特性。很多讨论、文章、博客和邮件列表中都提供了大量信息。

然而，真正的问题是如何获取像Apache Hadoop这样的开源项目的商业支持。公司通常只是为自己的产品提供支持，而不会为开源项目提供支持（不光是Hadoop项目，所有开源项目都面临这样的问题）。

何时使用Apache Hadoop

由于在本地系统上，只需10分钟左右就可完成其独立安装，所以Apache Hadoop很适合于第一次尝试。你可以试试WordCount示例（这是Hadoop的“hello world”示例），并浏览部分MapReduce的Java代码。

如果你并不想使用一个“真正的”Hadoop发行版本（请看下一节）的话，那么选择Apache Hadoop也是正确的。然而，我没有理由不去使用Hadoop的一个发行版本——因为它们也有免费的、非商业版。

所以，对于真正的Hadoop项目来说，我强烈推荐使用一个Hadoop的发行版本来代替Apache Hadoop。下一节将会说明这种选择的优点。

Hadoop发行版本

Hadoop发行版本解决了在上一节中所提到的问题。发行版本提供商的商业模型百分之百地依赖于自己的发行版本。他们提供打包、工具和商业支持。而这些不仅极大地简化了开发，而且也极大地简化了操作。

Hadoop发行版本将Hadoop生态系统所包含的不同项目打包在一起。这就确保了所有使用到的版本都可以顺当地在一起工作。发行版本会定期发布，它包含了不同项目的版本更新。

发行版本的提供商在打包之上还提供了用于部署、管理和监控Hadoop集群的图形化工具。采用这种方式，可以更容易地设置、管理和监控复杂集群。节省了大量的工作。

正如上节所提到的，获取普通Apache Hadoop项目的商业支持是很艰难的，而提供商却为自己的Hadoop发行版本提供了商业支持。

Hadoop发行版本提供商

目前，除了Apache Hadoop外， HortonWorks、Cloudera和MapR三驾马车在发布版本上差不多齐头并进。虽然，在此期间也出现了其他的Hadoop发行版本。比如EMC公司的Pivotal HD、IBM的InfoSphere BigInsights。通过Amazon Elastic MapReduce (EMR)，Amazon甚至在其云上提供了一个托管的、预配置的解决方案。

虽然很多别的软件提供商没有开发自己的Hadoop发行版本，但它们和某一个发行版本提供商相互合作。举例来说，Microsoft和Hortonworks相互合作，特别是合作将Apache Hadoop引入到Windows Server操作系统和Windows Azure云服务中。另外一个例子是，Oracle通过将自己的软硬件与Cloudera的Hadoop发行版本结合到一起，提供一个大数据应用产品。而像SAP、Talend这样的软件提供商则同时支持几个不同的发行版本。

如何选择合适的Hadoop发行版本？

本文不会评估各个Hadoop的发行版本。然而，下面会简短地介绍下主要的发行版本提供商。在不同的发行版本之间一般只有一些细微的差别，而提供商则将这些差别视为秘诀和自己产品的与众不同之处。下面的列表解释了这些差别：

- **Cloudera**：最成型的发行版本，拥有最多的部署案例。提供强大的部署、管理和监控工具。Cloudera开发并贡献了可实时处理大数据的Impala项目。
- **Hortonworks**：不拥有任何私有（非开源）修改地使用了100%开源Apache Hadoop的唯一提供商。Hortonworks是第一家使用了Apache HCatalog的元数据服务特性的提供商。并且，它们的Stinger开创性地极大地优化了Hive项目。Hortonworks为入门提供了一个非常好的，易于使用的沙盒。Hortonworks开发了很多增强特性并提交至核心主干，这使得Apache Hadoop能够在包括Windows Server和Windows Azure在内的Microsoft Windows平台上本地运行。
- **MapR**：与竞争者相比，它使用了一些不同的概念，特别是为了获取更好的性能和易用性而支持本地Unix文件系统而不是HDFS（使用非开源的组件）。可以使用本地Unix命令来代替Hadoop命令。除此之外，MapR还凭借诸如快照、镜像或有状态的故障恢复之类的高可用性特性来与其他竞争者相区别。该公司也领导着Apache Drill项目，该项目是Google的Dremel的开源项目的重新实现，目的是在Hadoop数据上执行类似SQL的查询以提供实时处理。
- **Amazon Elastic Map Reduce (EMR)**：区别于其他提供商的是，这是一个托管的解决方案，其运行在由Amazon Elastic Compute Cloud (Amazon EC2) 和Amazon Simple Storage Service (Amazon S3) 组成的网络规模的基础设施之上。除了Amazon的发行版本之外，你也可以在EMR上使用Map

R。临时集群是主要的使用情形。如果你需要一次性的或不常见的大数据处理，EMR可能会为你节省大笔开支。然而，这也存在不利之处。其只包含了Hadoop生态系统中Pig和Hive项目，在默认情况下不包含其他很多项目。并且，EMR是高度优化成与S3中的数据一起工作的，这种方式会有较高的延时并且不会定位位于你的计算节点上的数据。所以处于EMR上的文件IO相比于你自己的Hadoop集群或你的私有EC2集群来说会慢很多，并有更大的延时。

上面的发行版本都能灵活地单独使用或是与不同的大数据套件组合使用。而这期间出现的一些其它的发行版本则不够灵活，会将你绑定至特定的软件栈和（或）硬件栈。比如EMC的Pivotal HD原生地融合了Greenplum的分析数据库，目的是为了在Hadoop，或Intel的Apache Hadoop发行版本之上提供实时SQL查询和卓越的性能，Intel的Apache Hadoop发行版本为固态驱动器进行了优化，这是其他Hadoop公司目前还没有的做法。

所以，如果你的企业已经有了特定的供应方案栈，则一定要核查它支持哪个Hadoop发行版本。比如，如果你使用了Greenplum数据库，那么Pivotal就可能是一个完美的选择，而在其他情况下，可能更适合采取更加灵活的解决方案。例如，如果你已经使用了Talend ESB，并且你想使用Talend Big Data来启动你的大数据项目，那么你可以选择你心仪的Hadoop发行版本，因为Talend并不依赖于Hadoop发行版本的某个特定提供商。

为了做出正确的选择，请了解各个发行版本的概念并进行试用。请查证所提供的工具并分析企业版加上商业支持的总费用。在这之后，你就可以决定哪个发行版本是适合自己的。

何时使用Hadoop发行版本？

由于发行版本具有打包、工具和商业支持这些优点，所以在绝大多数使用情形下都应使用Hadoop的发行版本。使用普通的（原文为plan，应为plain）Apache Hadoop发布版本并在此基础之上构建自己的发行版本的情况是极少见的。你会要自己测试打包，构建自己的工具，并自己动手写补丁。其他一些人已经遇到了你将会遇到的同样问题。所以，请确信你有很好的理由不使用Hadoop发行版本。

然而，就算是Hadoop发行版本也需要付出很大的努力。你还是需要为自己的MapReduce作业编写大量代码，并将你所有的不同数据源集成到Hadoop中。而这就是大数据套件的切入点。

大数据套件

你可以在Apache Hadoop或Hadoop发行版本之上使用一个大数据套件。大数

据套件通常支持多个不同的Hadoop发行版本。然而，某些提供商实现了自己的Hadoop解决方案。无论哪种方式，大数据套件为了处理大数据而在发行版本上增加了几个更进一步的特性：

- **工具：**通常，大数据套件是建立像Eclipse之类的IDE之上。附加插件方便了大数据应用的开发。你可以在自己熟悉的开发环境之内创建、构建并部署大数据服务。
- **建模：**Apache Hadoop或Hadoop发行版本为Hadoop集群提供了基础设施。然而，你仍然要写一大堆很复杂的代码来构建自己的MapReduce程序。你可以使用普通的Java来编写这些代码，或者你也可以那些已经优化好的语言，比如 PigLatin或Hive查询语言（HQL），它们生成MapReduce代码。大数据套件提供了图形化的工具来为你的大数据服务进行建模。所有需要的代码都是自动生成的。你只用配置你的作业（即定义某些参数）。这样实现大数据作业变得更容易和更有效率。
- **代码生成：**生成所有的代码。你不用编写、调试、分析和优化你的MapReduce代码。
- **调度：**需要调度和监控大数据作业的执行。你无需为了调度而编写cron作业或是其他代码。你可以很容易地使用大数据套件来定义和管理执行计划。
- **集成：**Hadoop需要集成所有不同类技术和产品的数据。除了文件和SQL数据库之外，你还要集成NoSQL数据库、诸如Twitter或Facebook这样的社交媒体、来自消息中间件的消息、或者来自类似于Salesforce或SAP的B2B产品的数据。通过提供从不同接口到Hadoop和后端的众多连接器，大数据套件为集成提供了很多帮助。你不用手工编写连接代码，你只需使用图形化的工具来集成并映射所有这些数据。集成能力通常也具有数据质量特性，比如数据清洗以提高导入数据的质量。

大数据套件提供商

大数据套件的数目在持续增长。你可以在几个开源和专有提供商之间选择。像IBM、Oracle、Microsoft等这样的大部分大软件提供商将某一类的大数据套件集成到自己的软件产品组合中。而绝大多数的这些厂商仅只支持某一个Hadoop发行版本，要么是自己的，要么和某个Hadoop发行版本提供商合作。

从另外一方面来看，还有专注于数据处理的提供商可供选择。它们提供的产品可用于数据集成、数据质量、企业服务总线、业务流程管理和更进一步的集成组件。既有像Informatica这样的专有提供商，也有Talend或Pentaho这样的开源提供商。某些提供商不只支持某一个Hadoop发行版本，而是同时支持很多的。比如，就在撰写本文的时刻，Talend就可以和Apache Hadoop、Cloudera、 Hortonworks、MapR、Amazon Elastic MapReduce或某个定制的自创发行版本（如使用EMC的Pivotal HD）一起使用。

如何选择合适的大数据套件？

本文不会评估各个大数据套件。当你选择大数据套件时，应考虑几个方面。下面这些应该可以帮助你为自己的大数据问题作出合适的抉择：

- **简单性：**亲自试用大数据套件。这也就意味着：安装它，将它连接到你的Hadoop安装，集成你的不同接口（文件、数据库、B2B等等），并最终建模、部署、执行一些大数据作业。自己来了解使用大数据套件的容易程度——仅让某个提供商的顾问来为你展示它是如何工作是远远不够的。亲自做一个概念验证。
- **广泛性：**是否该大数据套件支持广泛使用的开源标准——不只是Hadoop和它的生态系统，还有通过SOAP和REST web服务的数据集成等等。它是否开源，并能根据你的特定问题易于改变或扩展？是否存在一个含有文档、论坛、博客和交流会的大社区？
- **特性：**是否支持所有需要的特性？Hadoop的发行版本（如果你已经使用了某一个）？你想要使用的Hadoop生态系统的所有部分？你想要集成的所有接口、技术、产品？请注意过多的特性可能会大大增加复杂性和费用。所以请查证你是否真正需要一个非常重量级的解决方案。是否你真的需要它的所有特性？
- **陷阱：**请注意某些陷阱。某些大数据套件采用数据驱动的付费方式（“数据税”），也就是说，你得为自己处理的每个数据行付费。因为我们是在谈论大数据，所以这会变得非常昂贵。并不是所有的大数据套件都会生成本地Apache Hadoop代码，通常要在每个Hadoop集群的服务器上安装一个私有引擎，而这样就会解除对于软件提供商的独立性。还要考虑你使用大数据套件真正想做的事情。某些解决方案仅支持将Hadoop用于ETL来填充数据至数据仓库，而其他一些解决方案还提供了诸如后处理、转换或Hadoop集群上的大数据分析。ETL仅是Apache Hadoop和其生态系统的一种使用情形。

决策树：框架vs.发行版本vs.套件

现在，你了解了Hadoop不同选择之间的差异。最后，让我们总结并讨论选择Apache Hadoop框架、Hadoop发行版本或大数据套件的场合。

下面的“决策树”将帮助你选择合适的一种：

Apache:

- 学习并理解底层细节？
- 专家？自己选择和配置？

发行版本:

- 容易的设置？
- 初学（新手）？

- 部署工具？
- 需要商业支持？

大数据套件：

- 不同数据源集成？
- 需要商业支持？
- 代码生成？
- 大数据作业的图形化调度？
- 实现大数据处理（集成、操作、分析）？

结论

Hadoop安装有好几种选择。你可以只使用Apache Hadoop项目并从Hadoop生态系统中创建自己的发行版本。像Cloudera、 Hortonworks或MapR这样的Hadoop发行版本提供商为了减少用户需要付出的工作，在Apache Hadoop之上添加了如工具、商业支持等特性。在Hadoop发行版本之上，为了使用如建模、代码生成、大数据作业调度、所有不同种类的数据源集成等附加特性，你可以使用一个大数据套件。一定要评估不同的选择来为自己的大数据项目做出正确的决策。

作者简介



Kai Wöhner 是Talend公司的首席顾问。他擅长的主要领域是Java EE、SOA、云计算、业务流程管理（BPM）、大数据以及企业架构管理。他还在像JavaOne、ApacheCon或OOP这样的国际IT会议上做演讲，为专业期刊撰文，并在[博客](#)上分享自己的经验。你可以在他的[网站](#)上找到更多详细信息和参考资料（演示文稿、文章、博客文章），可以点击[这里](#)或是通过Twitter：[@KaiWaehner](#)来联系他。

查看英文原文：[Spoilt for Choice - How to choose the right Big Data / Hadoop Platform?](#)

感谢[马国耀](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/BigDataPlatform>

推荐文章 | Article

HotSpot和OpenJDK入门

作者 [Ben Evans](#) , 译者 [孙镜涛](#)

在本文中，我们将会介绍如何开始使用HotSpot Java虚拟机以及它在OpenJDK开源项目中的实现——我们将会从两个方面进行介绍，分别是虚拟机和虚拟机与Java类库的交互。

HotSpot源码介绍

首先让我们看看JDK源码和它所包含的相关Java概念的实现。检查源码的方式主要有两种：

- 现代IDE能够附加src.zip（在\$JAVA_HOME目录），能够从IDE中访问
- 使用OpenJDK的源码并导航到文件系统

这两种方式都非常有用，但是重要的是哪种方式比较舒适一点。OpenJDK的源码存储在Mercurial（一个分布式的版本控制系统，与流行的Git版本控制系统相似）中。如果你不熟悉Mercurial，可以查看这本名为“[版本控制示例](#)”的免费书，该书介绍了相关的基础内容。

为了检出OpenJDK 7的源码，你需要安装Mercurial命令行工具，然后执行以下命令：

```
hg clone http://hg.openjdk.java.net/jdk7/jdk7_jdk7_tl
```

该命令会在本地生成一个OpenJDK仓库的副本。该仓库含有项目的基础布局，但是并没有包含所有的文件——因为OpenJDK项目分别分布在几个子仓库中。

完成克隆之后，本地仓库应该有类似于下面的内容：

```
ariel-2: jdk7_tl boxcat$ ls -l
total 664
-rw-r--r-- 1 boxcat staff 1503 14 May 12:54 ASSEMBLY_EXCEPTION
-rw-r--r-- 1 boxcat staff 19263 14 May 12:54 LICENSE
-rw-r--r-- 1 boxcat staff 16341 14 May 12:54 Makefile
-rw-r--r-- 1 boxcat staff 1808 14 May 12:54 README
-rw-r--r-- 1 boxcat staff 110836 14 May 12:54 README-builds.html
-rw-r--r-- 1 boxcat staff 172135 14 May 12:54 THIRD_PARTY_README
```

```

drwxr-xr-x 12 boxcat staff    408 14 May 12:54 corba
-rw xr-xr-x 1 boxcat staff 1367 14 May 12:54 get_source.sh
drwxr-xr-x 14 boxcat staff   476 14 May 12:55 hotspot
drwxr-xr-x 19 boxcat staff   646 14 May 12:54 jaxp
drwxr-xr-x 19 boxcat staff   646 14 May 12:55 jaxws
drwxr-xr-x 13 boxcat staff  442 16 May 16:01 jdk
drwxr-xr-x 13 boxcat staff  442 14 May 12:55 langtools
drwxr-xr-x 18 boxcat staff  612 14 May 12:54 make
drwxr-xr-x 3boxcat staff  102 14 May 12:54 test

```

接下来，你应该运行get_source.sh脚本，该脚本是初始克隆内容的一部分。该脚本会填充项目的剩余部分，克隆构建OpenJDK所需要的所有文件。

在我们深入并详细地介绍源码之前，我们必须要有“不惧怕平台源码”的信念。开发者通常会认为JDK源码一定是令人振奋且难以接近的，但这毕竟是整个平台的核心。

JDK源码是固定的、经过良好的审核和测试的，但是并不是那么无法接近。特别是这些源码并不是始终包含Java语言的最新特性。所以我们经常会在其内部找到那些依然没有泛型化的、使用原始类型的类。

对于JDK源码而言，有几个主要的仓库是你应该熟悉的：

jdk

这是类库存在的地方。几乎所有的内容都是Java（本地方法会使用一些C代码）。这是深入学习OpenJDK源码的一个非常好的起点。JDK的类在jdk/src/share/classes目录中。

hotspot

HotSpot虚拟机——这里面是C/C++和汇编代码（还有一些基于Java的虚拟机开发工具）。这些内容非常高级，如果你并不是一个专业的C/C++开发人员那么这些内容会让人有一点难以入手。稍后我们会更加详细地讨论一些入门的好方法。

langtools

对于那些对编译器和工具开发感兴趣的人而言，可以从这里找到语言和平台工具。大部分是Java和C代码——学习这些内容比学习JDK代码要难，但是对于大多数开发者而言还是可以接受的。

还有一些其他的仓库，但是它们可能没有那么重要或者对大多数开发者而言没什么吸引力，这些仓库包括corba、jaxp和jaxws等内容。

构建OpenJDK

Oracle最近开始了一个项目对OpenJDK做了一次全面的修整，并且简化了构建过程。这个项目称为“build-dev”，目前该项目已经完成并且成为了构建OpenJDK的标准方式。对于很多使用基于Unix系统的用户而言，构建过程现在就和安装一个编译器和一个“引导JDK”然后运行三个命令那么简单：

```
./configure
make clean
make images
```

如果你想获取更多与构建自己的OpenSDK相关的信息，那么[AdoptOpenJDK计划](#)（由伦敦的Java社团创建）是一个不错的起点——这是一个由100多位草根开发者组成的社团，他们都工作在警告清理、小bug解决和OpenJDK 8对主要开源项目的兼容性测试等项目上。

理解HotSpot运行时环境

Java运行时环境正如OpenJDK所提供的那样，由HotSpot JVM和类库（大部分都捆绑到了rt.jar里面）组成。

因为Java是一个可移植的环境，所有需要调用操作系统的内容最终都会由一个本地方法处理。另外，还有一些方法需要JVM的特殊处理（例如类的加载）。这些内容也会通过一个本地调用移交给JVM。

例如，让我们看看原始Object类中本地方法的C代码。Object类的本地源码包含在jdk/src/share/native/java/lang/Object.c文件中，它有六个方法。

Java本地接口（JNI）通常会要求本地方法的C实现按照一种非常特别的方式命名。例如，本地方法Object::getClass()使用通用的命名约定，因此C实现被包含在一个具有如下签名的C函数中：

```
Java_java_lang_Object_getClass(JNIEnv *env, jobject this)
```

JNI还有另一种加载本地方法的方式，java.lang.Object类中剩余的5个本地方法就使用了这种方式：

```
static JNINativeMethod methods[] = {
    {"hashCode",     "()I",      (void *)&JVM_IHashCode},
    {"wait",          "(J)V",     (void *)&JVM_MonitorWait},
```

```

    {"notify",           "()V",      (void *)&JVM_MonitorNotify},
    {"notifyAll",        "()V",      (void *)&JVM_MonitorNotifyAll},
    {"clone",            "()Ljava/lang/Object;", (void *)&JVM_Clone},};

```

这5个方法被映射到了JVM的入口点（它们是通过在C方法名上使用JVM_前缀来指定的），——使用registerNatives()的方式（开发人员能够通过这种方式改变Java本地方法到C函数名称的映射）。

Java运行时环境是用Java编写的，仅有很少的与JVM相关的小地方不是。除了代码的执行之外，JVM的主要工作是运行时环境的内务处理和维护，这里是活动Java对象运行时表示赖以生存的地方——Java堆。

OOP和KlassOOP

堆中的任何Java对象都是由一个普通的对象指针（OOP）表示的。在C/C++中一个OOP是一个真正的指针——一个指向Java堆里面某个内存位置的机器字。在JVM进程的虚拟地址空间中，会为Java堆分配一个单独的连续的地址范围，然后用户空间中的这块内存就会完全由JVM进程自己管理，直到JVM因为某些原因需要调整堆大小为止。

这意味着Java对象的创建和收集并不会牵扯到分配和释放内存的系统调用。

一个OOP由两个机器字头组成，它们被称为Mark和Klass字，之后是这个实例的成员字段。对于数组而言，在成员字段之前还有一个额外的字头——数组的长度。

之后我们会更加详细地介绍Mark和Klass字，但是它们的名字也暗示了一些内容——Mark字用于垃圾收集（用于标记——扫描的标记部分），而Klass字则是一个指向类元数据的指针。

在OOP头之后，实例字段会按照它在字节码中的特定顺序进行排列。如果你想了解更精确的细节，可以阅读NitsanWakart的博客文章[“理解Java对象的内存分布”](#)。

基本字段和引用字段都会排列在OOP头后面——当然，对象的引用也是OOP。下面让我们看一个Entry类（java.util.HashMap类中使用了该类）的例子：

```

static class Entry<K,V> implements Map.Entry<K,V> {
    final K key;
    V value;
    Entry<K,V> next;
    final int hash;
}

```

```
// methods...
}
```

现在，让我们来计算一下一个Entry对象的大小（在32位的JVM上）。

头包含一个Mark字和一个Klass字，因此在32位的HotSpot上OOP头会占用8个字节（在64位HotSpot上占用16个字节）。

一个OOP定义的总体大小是2个机器字加上所有实例字段的大小。

引用类型的字段实际上是指针——在所有健全的处理器架构中该指针都将占用一个机器字。

因此，因为我们有一个int字段，两个引用字段（对类型为K和V的对象的引用）和一个Entry字段，所以整个大小为2个字（头）+1个字（int）+3个字（指针）。

存储一个HashMap.Entry对象总共需要24个字节（6个字）。

KlassOOP

Klass字是OOP头中最重要的部分之一。它是指向这个类元数据的指针（它由一个称为KlassOOP的C++类型表示）。在这些元数据当中最重要的是这个类的方法，它们被表示为一个C++虚拟方法表（一个“vtable”）。

我们并不想让所有的实例都携带着方法的所有细节，因为这样做效率会非常低，所以使用了一个vtable在实例之间共享这些信息。

需要注意的是，KlassOOP和类加载操作所产生的类对象是不同的。这两者之间的区别可以概括为下面两个方面：

- Class对象（例如String.class）仅仅是普通的Java对象——它们和任何其他的ava对象（实例OOP）一样都是OOP，和所有其他的对象那样拥有同样的行为，同时它们也能够被放入Java变量中。
- KlassOOP是类元数据的JVM表示——它们通过一个vtable结构携带类的方法信息。我们不能直接从Java代码中获得到KlassOOP的引用——它们存在于堆的Permgen区域。

记住这个区别最容易的方式是，将KlassOOP当作是类对象的JVM级别的“镜像”。

虚拟调度

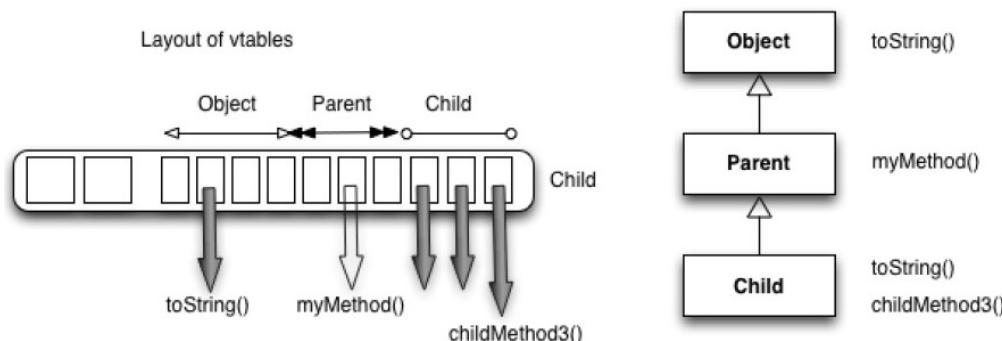
KlassOOP的vtable结构直接与Java的方法调度和单继承相关。要记住，默认情况下Java的实例方法调度是虚拟的（它使用被调用实例对象的运行时类型信息查找方法）。

在KlassOOPvtable中这是通过“常量vtable偏移”实现的。这意味着，重载方法在vtable中的偏移和它所重载的父类（包括祖父等）中的方法实现具有相同的偏移。

在这种情况下虚拟调度就很容易实现了，只需要简单地追溯继承层次（按照类——父类——祖父类的层次追溯）并寻找方法的实现就可以了（在vtable中的偏移始终相同）。

例如，这意味着在所有的类中`toString()`方法在vtable中的偏移始终相同。这个vtable结构有助于单继承，同时在使用JIT编译代码的时候也能够做一些非常好的优化。

(单击图片放大)



OOP头的Mark字是一个到某个结构的指针（实际上仅仅是一个位字段的集合，它们保存着OOP相关的内部处理信息）。

在常见的32位JVM环境中，Mark结构的位字段类似于下面的内容（查看hotspot/src/share/vm/oops/markOop.hpp了解更多内容）：

```
hash:25 ->| age:4 biased_lock:1 lock:2
```

高25位包含对象的`hashCode()`值，紧接着的4位是对象的年龄（存活对象所经过的垃圾收集的次数）。剩下的3个位用于表明对象的同步锁状态。

Java 5引入了一种新的对象同步方式，称为偏向锁（在Java 6中是默认的锁机制）。该方案的灵感来源于对对象运行时行为的观察——在很多情况下对象永远只

会被一个线程锁定。

在偏向锁中，一个对象会“偏向于”锁定它的第一个线程——然后这个线程会实现更好的锁性能。获得偏向的线程会被记录在Mark头中。

```
JavaThread*:23 epoch:2 age:4 biased_lock:1 lock:2
```

如果另一个线程试图锁定对象，那么这个偏向就会被取消（并且不会被重新获得），并且自此之后所有的线程都必须明确地锁定和解锁对象。

对象的状态可能会是：

- 未锁定
- 偏向的
- 轻量级锁定
- 重量级锁定
- 标记的（仅在垃圾收集期间有效）

HotSpot源码中的OOP

HotSpot源中相关的OOP类型层次非常复杂。这些类型被保存在hotspot/src/share/vm/oops中，包括：

- oop (抽象基础)
- instanceOop (实例对象)
- methodOop (方法表示)
- arrayOop (数组抽象基础)
- symbolOop (内部符号/字符串类)
- klassOop
- markOop

有一些稍微奇怪的历史性事件——虚拟调度表（vtable）的内容和klassOOP是分开保存的，markOOP和其他OOP看起来完全不同，但是它依然包含在同样的层次中。

一个非常有趣的地方是，我们可以从jmap命令行工具中直接看到OOP。它对堆中的内容做了一个快照，包括出现在permgen中的所有OOP（包括子类和Klass OOP所需的支持结构）。

```
$ jmap -histo 150 | head -18
num #instances #bytes class name
```

```

1: 10555 21650048 [I
2: 272357 6536568 java.lang.Double
3: 25163 5670768 [Ljava.lang.Object;
4: 229099 5498376 com.jclarity.cenum.dataset.CenumXYDataItem
5: 39021 5470944 <constMethodKlass>
6: 39021 5319320 <methodKlass>
7: 8269 4031248 [B
8: 3161 3855136 <constantPoolKlass>
9: 119759 2874216 org.jfree.data.xy.XYDataItem
10: 3161 2773120 <instanceKlassKlass>
11: 2894 2451648 <constantPoolCacheKlass>
12: 34012 2271576 [C
13: 87065 2089560 java.lang.Long
14: 20897 2006112 [Lcom.jclarity.cenum.CollectionType;
15: 33798 1081536 java.util.HashMap$Entry

```

尖括号中的条目包含了各种类型的OOP，例如[I]和[B]分别指int类型和byte类型的数组。

HotSpot解释器

开发者通常会比较熟悉那种“在一个while循环中切换”的解释器，但是HotSpot比这种类型的解释器要更加先进。

HotSpot是一个模板解释器。这意味着它会构建一个动态的、优化的机器码调度表——特定于用户所使用的操作系统和CPU。大部分的字节码指令都是使用汇编语言代码实现的，仅有非常复杂的指令会被委托给虚拟机处理，例如从一个类文件的常量池中查找一个入口。

这提升了HotSpot解释器的性能，但是代价是难以将虚拟机移植到新的架构和操作系统上。同是对于新开发者而言也增加了他们理解解释器的难度。

对于新手开发者而言，对OpenJDK所提供的运行时环境有一个基础的理解是非常必要的：

- 环境中的大部分都是使用Java编写的
- 通过本地方法实现操作系统的可移植性
- 堆中的Java对象由OOP表示
- JVM中的类元数据用KlassOOP表示
- 有一个先进的高性能模板解释器，哪怕是解释执行模式下的性能

到现在为止，开发者已经能够开始探索JDK仓库中的Java代码了，也能够尝试着积累自己的C/C++和汇编知识去深入学习HotSpot了。

关于作者



Ben Evans 是jClarity公司的CEO，这是一家为开发和运营团队提供性能工具的新公司。他是LJC（伦敦JUG）的组织者之一，同时也是JCP执行委员会中的一员，他帮助Java生态系统定义标准，是一个JavaOne巨星，《Java程序员修炼之道》一书的合著者，他会定期地针对Java平台、性能、并行处理等相关主题发表公开演讲。

查看英文原文：[Getting Started with HotSpot and OpenJDK](#)

原文地址：<http://www.infoq.com/cn/articles/Introduction-to-HotSpot>

相关内容

- [Azul Systems发布面向WebSphere应用服务器的Zing平台版](#)
- [HotSpot JVM的任务控制和Flight Recorder](#)
- [多租户JVM技术前瞻](#)
- [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](#)

推荐语

AWDC（阿里云开发者大会）

2013年10月24、25日，阿里云开发者大会在杭州转塘阿里云创业新基地召开，这次会议以“云计算的蝴蝶效应”为主题，第一天进行大赛的参赛作品评选，第二天围绕“移动互联网”、“游戏”、“开发者服务”和“金融创新”设置了分论坛进行分享。



特别专栏 | Column

阿里巴巴集团CTO 王坚博士：云的蝴蝶效应

作者 [InfoQ中文站](#)



在10月25日的阿里云开发者大会上，阿里巴巴集团CTO 王坚博士做了题为『云的蝴蝶效应』的分享。

王坚博士首先感谢所有人对阿里云的发展：

阿里云大概是09年的9月10号成立，过去那么多年一直想一句话，阿里云是在座的所有开发者用他们的命养活的。所以阿里云的今天是大家用命换回来的，感谢大家。

目前中国除了澳门，所有省分都有阿里云的客户，这对于没有一个销售人员的阿里云来说，是一个希望，同时也是这个时代的变革。王坚博士列举了几个例子来说明他眼中云计算带来的变革：

- 天弘基金从倒数第2到前数第5，这个基金的后台，就是在这么短时间架构在阿里云上；
- 云计算可以让一家传统公司变成一家互联网公司；
- 玩蟹是一帮85后的年轻人，对云计算非常的坚定，也相信阿里云能够做好云计算，这么不到100人团队，变成一个价值17亿公司；

总这些都让阿里云相信，云计算可以让一个人公司与一个很大公司在同一起跑线上竞争。阿里云还在苦练内功，比如今年的8月15号，阿里巴巴集团正式运营服务器规模达到5000台（5K）的“飞天”集群，成为中国第一个独立研发拥有大规模通用计算平台的公司，也是世界上第一个对外提供5K云计算服务能力的公司，对于5K项目，他说：

我今天觉得我们不能说做云计算，因为那不是我们能做的事情，我们今天要做的事情就是把飞天做好，因为在座的所有人把中国的云计算做起来，阿里云能做的就是把飞天做好。所以非常感恩，在阿里巴巴这样的公司，可以让那么多的技术人员专著的做一件事情就是“飞天”。

同时，这次开发者大会上还宣布了阿里云Ali OS的下一步动作：Ali OS联合了HKE、波导等品牌手机厂商推出了搭载最新操作系统的新品手机，阿里巴巴集团副总裁喻策表示，自4月份以来，Ali OS的合作伙伴不断增加，其中除了专注于智能手机生产的多个国内厂商，芯片商、应用开发商也在积极寻求与Ali OS合作的方式，接下来将会拿出10亿元来建立操作系统的生态圈。

最后，王坚说：

云计算的收获一定是在移动互联网，这是硬币的两面，少了一面都不成立。云计算的收获一定是在移动互联网，这是硬币的两面，少了一面都不成立。你永远要相信你坚定的东西，也永远要坚定你相信的东西。

原文链接：<http://www.infoq.com/cn/news/2013/11/alibaba-wangjian-on-cloud>

相关内容

- [阿里巴巴集团数据平台总监张东晖谈阿里云ODPS](#)
- [阿里巴巴集团CTO王坚演讲实录](#)
- [阿里巴巴网站架构师周涛明：跨境网站的优化与挑战](#)
- [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](#)
- [阿里巴巴陶勇谈海量数据技术架构](#)

特别专栏 | Column

阿里云计算资深总监唐洪谈飞天现状以及5K项目发展

作者 [水羽哲](#)

2013年8月，历时4个月，阿里集团涵括云梯1空间优化与跨机房集群扩展、以及云梯2 单集群规模从1500台升级到5000台，同时实现跨集群扩展的5K项目顺利取得阶段性成果。

在[阿里云开发者大会](#)上，我们就飞天的发展现状和这次5K项目取得的突破采访了阿里云计算资深总监唐洪。

InfoQ：现在飞天的整体规模是怎样的？有多少数据中心部署了飞天？

唐洪：大概有几十个数据中心，差不多每个阿里巴巴集团的数据中心里面都会有我们自己部署的飞天集群，单集群的规模也已经到了5000台。整个飞天我们从2008年做了5年的时间，可以看到集团使用基于飞天云平台的比例越来越重，基本上所有阿里集团重量级的应用都会用到阿里云。其他部门没有精确的统计过，比如说大的数据处理方面，我觉得云平台的比例超过一半以上。

我们以前有内部有个说法，叫云梯1，云梯2。云梯1是基于开源系统搭建的，是飞天没有完全成熟之前的云梯过渡阶段，现在使用的资源已经小于飞天的资源。

InfoQ：飞天现在每年版本的更新状况。

唐洪：其实很难界定，版本的概念是传统软件的一种概念，我们的版本首先是一个操作系统，以服务化的方式来提供的，比如说现场发现一个bug我们马上就更新了。我们一般一年一到两次这样大的升级，但是外界用户一般是看不到的。

InfoQ：我看到网上有一些评测，阿里云IO速度有些慢，我不知道您是怎么看的？

唐洪：有一种是一些竞争对手的评测，他们从不同的角度来做类似于苹果和橘

子的比较。其实对一个系统的设定，每个人的取舍不同，比如系统性能的稳定性、数据的可靠性等，大家关注的方面显然不太一样，因此性能快慢也是相对来说的，对于绝大多数的用户来说，不是追求极致的把一个硬盘的性能压到极致，而是要降低他的运维IT的负载、降低他的硬件购置的成本。所以我们没有追求单个虚拟机能够做到多快多快的性能。我们追求在一个合理的，在一个比较好的性能的范围内，提供好的性价比。第三类是碰到一些客户的问题，所谓IO的问题其实是一些我们系统设计里边碰到的比较有挑战性的问题，可以认为是个bug，也就是说它的平均的性能满足期望，但是偶尔性能可能会导致整个系统不响应，我们2012年开始花了很多精力在解决这样的问题，我们从工单的角度来看这方面的问题是解决差不多了。

InfoQ：阿里云基础设施，如防断电、防不可抗的自然灾害等有没有优越于其他服务商的？

唐洪：第一，在阿里巴巴，我们维护淘宝总站的团队和维护阿里云是一个团队，所以在国内是屈指可数的几家公司运营这么大的网站。第二个，从硬件角度来说防断电，我们设有几级的保护措施，比如我们在每一个机房会有柴油机发电机，当它断电之后柴油发电机可以切换进来，确保它有足够的能量供给72小时临时供电。我们会有一个UPS，断电以后UPS切换进来，然后是发电机发动起来，我们会收到报警把一些非关键性的功能关掉。延长整个数据中心的使用时间。另外一方面从软件层，比如杭州前一阵发大水，如果时间再久一点，可能是连柴油机都顶不上了情况下、甚至网络隔绝的情况下，我们的云服务器会提供异地架构的方案，但是需要客户去做一些配合，比如说你可以在两个不同的数据中心购买我们的云服务器，这样发生故障的话你可以切换到另一个那边去，这是无状态的东西。还有有状态的，比如OSS、OTS这些服务，我们会做异地的同步，应该都还没有发布，但是对重点客户我们在做测试，今年或者明年会提供这些东西。

InfoQ：你们在研制飞天的时候，硬件研发到了什么样的阶段？目前飞天面临的最大的挑战是什么？

唐洪：飞天作为一个通用的计算服务，已经标准化硬件，所有的飞天集群机型都是差不多的，比较通用型的带CPU、带内存，有些云服务比如说针对存储型的，全部用的SSD的硬盘，相对来说有比较好的耦合。

对于挑战来说，当你把一个服务从企业内部的人群变成了互联网人群的话，首先要考虑的是安全的问题，企业审核的流程，可以做到一些控制，如果放在公共的互联网上，任何人可以到5K上，他们的作业是并行跑的，安全首先是非

常非常大的挑战，我们是做了很多的工作。

其次企业用户和互联网的量级是不一样的，企业用户数以百计，互联网用户数以千计、数以万计，在整个架构上用户管理上有不一样的大的架构。很多美国的互联网公司也基于hadoop做了很多的东西，为什么从来没有公司可以把hadoop这种能力作为功能服务弄出去？这个本身需要做很大的工作才有可能变成一个公共服务的。

InfoQ：云服务对外界的市场的比例多少？

唐洪：没有数过多少机器用于对内服务，多少机器用于对外服务，我猜想差不多一半的样子，比如数据形态的大规模数据的处理，对内服务多一些，对于长尾的，比如对于移动端的应用的东西可能对外的多一些。

InfoQ：我们了解到飞天实现了**5000**节点的单集群，我想问一下单集群有什么好处，双集群又有什么好处。

唐洪：飞天在单个集群里面，可以说是透明。单个应用，通常来说只能用到一个集群里面的资源，也就是说，属于你的一个集群只有500台的话，你最大的规模也就只能到500台了。所以大集群容纳的作业也就大了。

其次是数据，当你有分多个集群的时候，在本地和远程访问的性能是不太一样的。集群大了之后，可以做的事情就更多了，承载的数据就更加容易访问。当然也不是说多集群就不可以做到这一点，因为很多的集群是用有很多小的集群来做的，包括hadoop也有这样的方式。通常来说，它的效率就会低一点。

InfoQ：单集群升至到**5000**台，跨集群的计算，已经是一个比较大的规模。它在国内甚至在国际上的影响是怎么样的？飞天**5K**影响力谈一下。

唐洪：这是一个技术的东西，我觉得它的影响力单单技术实现到一个东西是没有影响力的，我们今年在开发者大赛的时候，做了一个飞天5K的体验项目，把它的计算能力给外面的人用了，这件事情谷歌，facebook都没有做到过的事情。一个随随便便的开发者，当他有一个需求，想要用5000台机器跑一个小时，做一些事情的时候，可以直接使用。

InfoQ：未来数据可能不断的往外成倍的往上翻，我相信有一天**5000**台会达到极限，那时候怎么做？

唐洪：那时候说不定是10000台一个集群了。正常的话，技术演进也是一个渐变的过程，我们这次大概半年的时间从1500扩展到5000，3倍的扩展，500到一万2倍的扩展，绝对数量是比较可怕的，但是相对而言就小一点的问题了。还有就是硬件的红利：CPU在变快、网速从千兆变成了万兆，核心交换机的容量也可以增大，某种意义上到那个时候简单的支持红利，即使集群规模在5000台，处理能力也会有可能翻倍的。

InfoQ：咱们是单集群5000台往上加，还是说再去加集群？

唐洪：单个集群到5000台，但是会有第二个5000、第三个5000，你通常说的是业务的数目增加，不是单个业务量，通常单个业务量一个5000台支撑可以的，这就是我们为什么要做5000台的量的原因。

InfoQ：咱们5K项目，总目标是咱们提供跨机房的服务，最终这些收益的人群，是定位在什么地方？

唐洪：某种意义上5K是个基础，不是根本的目标。我们把5K作为整个云服务的基础，一方面能提供大规模计算的能力，另外有5K是可以帮助我们做成本的节省，比如有些应用是CPU，有些应用是内存，有些应用是IO，只有池子大了我才能在几个应用之间进行互补，否则池子那么小，资源占满，其他东西跑不上来，所以5K可以降低运维的成本、集群数目少提高了调度的效率、削峰填谷的事情可以做的更多，这样反映在用户上面，对他们来说我们可以提供更低廉的服务，做到同样的资源。

InfoQ：前面几十个数据中心，我想知道，多大的比例部署了5K？

唐洪：我们现在有三个集群部署了5K。

InfoQ：您去年说飞天是谷歌+亚马逊的服务模式，现在跟亚马逊、谷歌比起来有什么优势？

唐洪：我们作为一个公司来说，要看产品和业务的竞争力，跟谷歌比不是非常公平的比较，因为谷歌没有在推云服务。谷歌在云那方面做的一直羞羞答答的，一直没有大踏步向前走，它其实有云服务，但我听到的小道消息的说法是有

两派人，一派人觉得我们不应该做，就是谷歌不应该做这种公共云计算服务商，他觉得谷歌不应该把这种大规模计算的能力公布出去，造就他的很多的竞争对 手，对它的搜索对它的广告造成危险。亚马逊的角度来说，大规模计算服务，它是没有的。对我们的优势，毕竟我们在国内对中国的云计算，或者是移动互联网的市场我们会有第一手精确的了，这样对我们产品的本地化，贴近用户的使用习惯，我们觉得可以比亚马逊做的更好。

InfoQ：您看到前面的路是怎么样的？

唐洪：道路，前景很光明，道路还是很艰苦的，马拉松才跑了5公里。还有很长的路，还有很多人要跑。

原文链接：<http://www.infoq.com/cn/articles/ali-yun-on-feitian-5k>

相关内容

- [云计算成就创业梦想-阿里云开发者大赛盛大开启](#)
- [云计算就是服务：阿里云总裁王坚博士访谈录](#)
- [阿里云开发者大会：企业如何看待云计算](#)
- [云计算与大数据共舞 阿里云开发者盛宴](#)
- [推动云计算应用落地与实践-阿里云开发者大会正式启动](#)

特别专栏 | Column

恒拓开源陈操谈去IOE方案的普及对独立开发商的机遇与挑战

作者 杨赛

阿里巴巴在国内互联网领域是去IOE的强力推动者。今年5月17日，支付宝最后一台IBM型机下线，整个阿里集团告别IBM小机；7月10日，淘宝广告系统的Oracle数据库下线，淘宝告别Oracle。10月，阿里正式启动“聚宝盆”项目，针对金融行业软件开发商和广大中小金融企业推出云迁移服务，把原来采用IOE的技术方案改造成基于阿里云技术的方案，并通过金融软件开发商把金融业务迁移到阿里云。

在传统企业软件领域，也有很多开发商在致力于开源解决方案的输出。在2013年10月的阿里云开发者大会上，恒拓开源作为阿里云行业合作伙伴之一亮相，主持了一个开源技术分论坛。在航空领域，恒拓开源已经积累了不少客户和项目经验。在下面跟恒拓开源技术副总监陈操的对话中，我们将对整个行业的去IOE普及的状态进行一个概述，请陈操分享一下他们推广开源方案的经验，并聊聊独立开发商在这样的一个环境变化中将面对哪些机遇与挑战等话题。

嘉宾简介：

陈操，恒拓开源架构顾问/技术副总监，专注于企业级开源解决方案，在企业级Java开发领域有十多年的经验。积累了丰富的ESB、SOA、JBoss、Drools、分布式计算、高并发高访问量系统架构经验。参与主导了大量中大型企业级信息系统架构设计，具有丰富的实战经验。曾在中国数码集团任职，于2010年加入恒拓开源，目前负责恒拓开源深圳分公司的团队与业务。

InfoQ：能否根据你们目前接触到的客户的情况，描述一下整个行业对去IOE这个概念的接受情况，以及实际实施的进展状态？

陈操：在近几年，我们接触的这些行业形势比之前已经好了非常多，大家对“去IOE”、尤其是对开源的认识，跟过去相比已有显著提高。虽然，能够在实践中真正敢于去尝试的企业为数不多，但是大部分已经有这样的一些规划了，南航算是其中敢于吃螃蟹并获得成功的。我们接触的客户，有航协、航信、东航、国航、中信信托、长安汽车，还有我们在深圳所接触到的深交所、深圳证券通讯有限公司、前海股权交易中心等，我们很高兴看到这些航空行业，制造行业，金融行业的企业纷纷开始想尝试和探索“去IOE”。

目前在航空行业，我们算是走得比较前。对于绝大多数客户，我们目前正在尝试着用开源的解决方案去替换掉IBM的小机和中间件，Oracle的数据库，用我们的分布式存储方案去替换掉EMC的存储设备等等。许多项目都正在接洽或已经在实施，这相比前几年来讲已经是很大的进步和提升。

InfoQ：能否以某个场景为例，简单估算一下去IOE之前的硬件-License-服务运维团队方面的成本总和和比例，以及去IOE之后的成本比例？

陈操：举例来说，在没有“去IOE”之前，很多企业，像我们接触到的一些国企，硬件主要是IBM的小型机或者大机，现在给他们换成几万块钱一台的刀片组成的小集群，用普通的PC硬盘或者磁盘阵列替换EMC存储设备作为大数据的存储方案。仅这一块节省的资金就十分可观。

软件方面，从操作系统、负载均衡、Web服务器、JavaEE容器、消息中间件、到ESB服务器，到工作流，规则引擎，分布式计算等基础设施和框架，均有开源的解决方案且几乎完全免费。我们从2010年开始为我们其中一个大客户服务，2011年到2012年之间，一共为其节省了将近四千万软件License费用。从今年之后，我们的客户已经开始尝试替换商业数据库，我相信数据库存储这块的替换能够带来更多的成本节省。

服务运维这块，如果客户之前用的IBM或Oracle，基本上只能找原厂的工程师或他们的合作伙伴去运维；“去IOE”之后，目前的市面上，熟悉开源技术的工程师越来越多，生态环境越来越好，服务运维较之前更容易，且获得更多的选择。总体说来，“去IOE”之前，成本这块硬件占据大头，软件居中，其次才是服务，当然也有少量较昂贵的。“去IOE”之后，软件几乎没有成本，硬件占据小头，相比之下服务运维——包括定制开发——的比例会稍微高一些，而且服务运维的性价比更优于原厂服务。当然，这也要视项目的具体情形而定。

InfoQ：去IOE的过程，客户那边有没有来自内部的阻力，或者其他阻力？是如何解决这些阻力的？

陈操：这么多年来我们一直在国内市场推广开源，期间确实遇到了非常多的阻力。

我们接触到的企业大多数都是国企和政府。对于大型国企来讲，第一个阻力来自于是观念上的，因为这些企业里面的大部分员工对开源技术了解的不多，虽然比起前几年来说已经有了一个很大的改变，但我们还是要面对他们对开源和商业的一个不要钱、一个要钱的观念上的转换问题。

第二个是学习阻力：对于那些已经有一定的技术背景的客户来讲，他们之前因为被IBM、Oracle已经洗脑了多年，所以如果要他们转换过来，对他们来说会有很大的成本。比如我们接触到的汽车制造企业，他们的技术专家在Oracle方面已经积累了差不多有10年的经验，这个时候让他去转换成开源，自身的挑战和牺牲很大。

第三个阻力是政治阻力，政治阻力主要源于央企和国企，这类企业的中层普遍认为自身没有必要去冒风险尝试开源。他们会认为，无论什么原因导致项目失败，只要是有国际上的知名商业厂商，可以借此免责，但如果用的是开源，那么要承担的责任就大了，严重的还可能影响政治前途。最后一个比较大的阻力就是来自于利益上的，这个就属于不在阳光下的那一部分了。

至于怎么去解决这些阻力，我们最开始尝试的是自下而上的策略去推行“去IOE”，比如说我们去接触对方的项目经理，或者是技术层面的人，同他们宣讲开源和“去IOE”，虽然他们经过这么多年已经非常认可开源，但是你说让他们在自己的企业里面主导实践这个过程，是非常困难的，风险非常大。所以之后我们改变了策略，觉得这个要推广还是应该从高层，甚至从政府的层面去做一些宣传，让我们的央企、国企的高管们有意识去往下推行“去IOE”这样一个理念，下面的人才会容易在项目中真正进行实践。

改变了这个策略之后，我们确实收到了不错的效果，比如我们的一个客户，他们的领导非常重视，直接把“去IOE”放到他们下面员工的KPI当中，接下来的工作开展的就比较顺利了。

除了自上而下的策略以外，我们还会去加强我们的培训，和对“去IOE”这块的宣讲。你也知道，前一阵子的棱镜门引起了包括政府在内很多大企业的重视，这也是我们用来推广“去IOE”的一个很有说服力的案例。

InfoQ：跟阿里云这边的合作具体包括哪些？

陈操：我们很早之前就开始跟阿里云合作。现在企业内部的两个技术型的产品，考拉跟变色龙，以及我们的行业产品——运价魔方都已经迁移到了阿里云上面。我们购买了他们的主机、带宽、存储，其中光我们的运价魔方这一个产品就买了阿里云十多台主机进行运营。

除了这几款产品以外，我们内部的持续交付平台、持续集成平台，全部都已经搭建在了阿里云上。我们的开源中国社区，前不久和阿里一起推出了中国源，主要提供代码托管、Maven仓库管理以及开源软件镜像下载功能，也已经上线了。

InfoQ：有没有客户提出要将应用迁移到阿里云上的需求？你们在这一块上目前提供哪些服务？

陈操：我们接触到的很多客户，尤其是一些政府的客户经常会找到我们，因为阿里云同政府的关系非常不错，政府也有一些意愿去把他们的一些系统从他们原来的机房迁移到阿里云上，以寻求更安全、更高效、更廉价的服务。不少的客户跟我们提起过这些需求，说是不是可以把他们的应用和系统迁移到阿里云上去，我们给他们提供系统迁移和改造服务。这也正是目前我们可以提供服务，即：熟悉业务系统的流程，数据结构，系统架构和设计，以及部署等方面的内容，基于这些，提供迁移方案甚至实施服务。

我们之前已经跟阿里合作把PHP从Windows环境迁移到Linux上。另外我们在一些政府的核心系统迁移上，开始和阿里有这种合作的意向。

InfoQ：就你们的观察，现在云计算的普及、基于x86和Linux架构的主流化，对独立开发商而言，都有哪些机遇和挑战？

陈操：机遇和挑战是并存的。机遇的话，就目前来讲，现在云计算的普及以及开源化，x86结构的主流化，首先能够帮助我们的独立开发商以更低廉的成本去实施他们的项目，以获取更多的利润，进而能承接更多的项目，提升自身的技术能力。如果这些开发商原来是跟IOE紧密合作的，这样也可以帮助他们摆脱大厂商的控制，增强自己的独立性。另外，借助云计算，使得独立软件开发商不用关注基础设施搭建，能够帮助他们更加专注在自己擅长的领域。这些都是机遇。

挑战的话则是，如果摆脱了大厂商，真正脱离了IOE，对于这些独立开发商来讲，就没有了品牌的支撑，这是一个比较大的风险。对他们自身而言，可能需要去培养自己的技术力量，甚至在业务上要更加具有专业性；此外这种改变还有可能对自身的技术体系有一定的冲击，甚至运营模式也可能会随之进行转变，这些我觉得是对软件开发商的挑战。

原文链接：<http://www.infoq.com/cn/news/2013/11/probing-ioe-and-isv>

相关内容

- [58同城开源其轻量级Web框架Argo](#)
- [Netflix继续开源，更多猴子进入视野](#)

避开那些坑 | Void

双重检查锁定与延迟初始化

作者 程晓明

在java程序中，有时候可能需要推迟一些高开销的对象初始化操作，并且只有在使用这些对象时才进行初始化。此时程序员可能会采用延迟初始化。但要正确实现线程安全的延迟初始化需要一些技巧，否则很容易出现问题。比如，下面是非线程安全的延迟初始化对象的示例代码：

```
public class UnsafeLazyInitialization {  
    private static Instance instance;  
    public static Instance getInstance() {  
        if (instance == null) //1: A线程执行  
            instance = new Instance(); //2: B线程执行  
        return instance;  
    }  
}
```

在UnsafeLazyInitialization中，假设A线程执行代码1的同时，B线程执行代码2。此时，线程A可能会看到instance引用的对象还没有完成初始化（出现这种情况的原因见后文的“问题的根源”）。

对于UnsafeLazyInitialization，我们可以对getInstance()做同步处理来实现线程安全的延迟初始化。示例代码如下：

```
public class SafeLazyInitialization {  
    private static Instance instance;  
  
    public synchronized static Instance getInstance() {  
        if (instance == null)  
            instance = new Instance();  
        return instance;  
    }  
}
```

由于对getInstance()做了同步处理，synchronized将导致性能开销。如果getInstance()被多个线程频繁的调用，将会导致程序执行性能的下降。反之，如果getInstance()不会被多个线程频繁的调用，那么这个延迟初始化方案将能提供令人满意的性能。

在早期的JVM中，`synchronized`（甚至是无竞争的`synchronized`）存在这巨大的性能开销。因此，人们想出了一个“聪明”的技巧：双重检查锁定（double-checked locking）。人们想通过双重检查锁定来降低同步的开销。下面是使用双重检查锁定来实现延迟初始化的示例代码：

```

public class DoubleCheckedLocking {           //1
    private static Instance instance;          //2

    public static Instance getInstance() {      //3
        if (instance == null) {                 //4:第一次检查
            synchronized (DoubleCheckedLocking.class) { //5:加锁
                if (instance == null)               //6:第二次检查
                    instance = new Instance();     //7:问题的根源出
在这里
                }                                //8
            }
        return instance;                      //9
    }                                    //10
}                                      //11
}                                      //12

```

如上面代码所示，如果第一次检查`instance`不为`null`，那么就不需要执行下面的加锁和初始化操作。因此可以大幅降低`synchronized`带来的性能开销。上面代码表面上看起来，似乎两全其美：

- 在多个线程试图在同一时间创建对象时，会通过加锁来保证只有一个线程能创建对象。
- 在对象创建好之后，执行`getInstance()`将不需要获取锁，直接返回已创建好的对象。

双重检查锁定看起来似乎很完美，但这是一个错误的优化！在线程执行到第4行代码读取到`instance`不为`null`时，`instance`引用的对象有可能还没有完成初始化。

问题的根源

前面的双重检查锁定示例代码的第7行（`instance = new Singleton();`）创建一个对象。这一行代码可以分解为如下的三行伪代码：

```

memory = allocate();    //1: 分配对象的内存空间
ctorInstance(memory);  //2: 初始化对象
instance = memory;     //3: 设置instance指向刚分配的内存地址

```

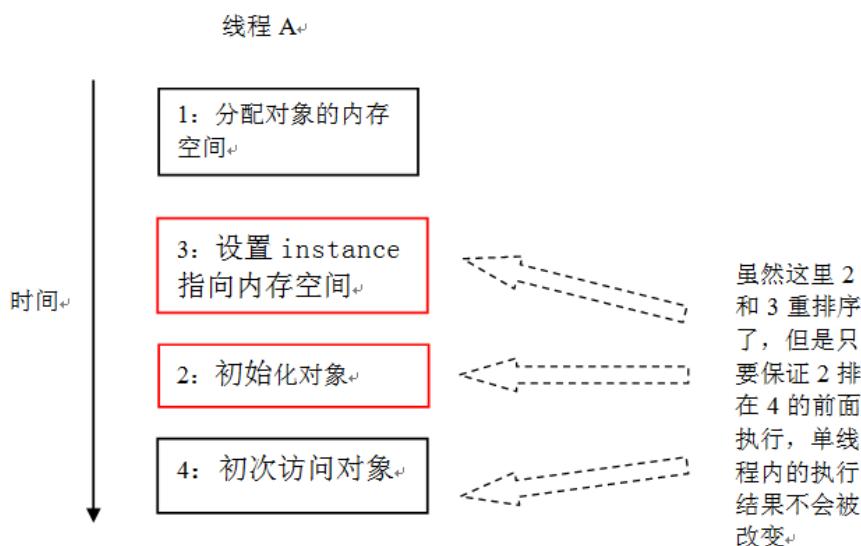
上面三行伪代码中的2和3之间，可能会被重排序（在一些JIT编译器上，这种重

排序是真实发生的，详情见参考文献1的“Out-of-order writes”部分）。2和3之间重排序之后的执行时序如下：

```
memory = allocate(); //1: 分配对象的内存空间
instance = memory; //3: 设置instance指向刚分配的内存地址
                     //注意，此时对象还没有被初始化！
ctorInstance(memory); //2: 初始化对象
```

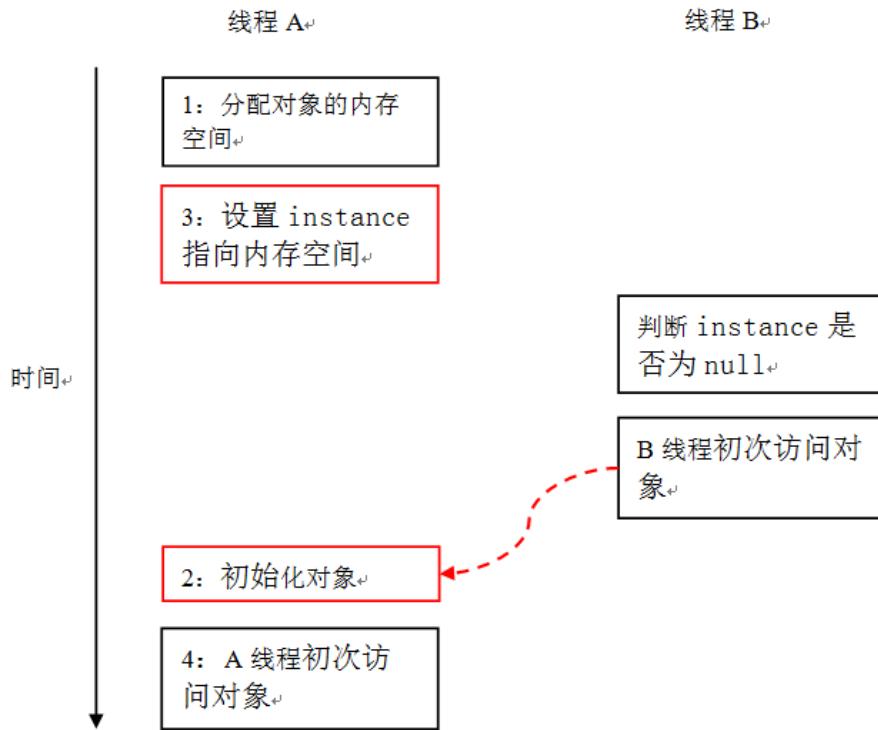
根据《The Java Language Specification, Java SE 7 Edition》（后文简称为java语言规范），所有线程在执行java程序时必须要遵守intra-thread semantics。intra-thread semantics保证重排序不会改变单线程内的程序执行结果。换句话说，intra-thread semantics允许那些在单线程内，不会改变单线程程序执行结果的重排序。上面三行伪代码的2和3之间虽然被重排序了，但这个重排序并不会违反intra-thread semantics。这个重排序在没有改变单线程程序的执行结果的前提下，可以提高程序的执行性能。

为了更好的理解intra-thread semantics，请看下面的示意图（假设一个线程A在构造对象后，立即访问这个对象）：



如上图所示，只要保证2排在4的前面，即使2和3之间重排序了，也不会违反intra-thread semantics。

下面，再让我们看看多线程并发执行的时候的情况。请看下面的示意图：



由于单线程内要遵守intra-thread semantics，从而能保证A线程的程序执行结果不会被改变。但是当线程A和B按上图的时序执行时，B线程将看到一个还没有被初始化的对象。

*注：本文统一用红色的虚箭线标识错误的读操作，用绿色的虚箭线标识正确的读操作。

回到本文的主题，DoubleCheckedLocking示例代码的第7行（`instance = new Singleton();`）如果发生重排序，另一个并发执行的线程B就有可能在第4行判断`instance`不为null。线程B接下来将访问`instance`所引用的对象，但此时这个对象可能还没有被A线程初始化！下面是这个场景的具体执行时序：

时间	线程A	线程B
t1	A1: 分配对象的内存空间	
t2	A3: 设置instance指向内存空间	
t3		B1: 判断instance是否为空
t4		B2: 由于instance不为null，线程B将访问instance引用的对象
t5	A2: 初始化对象	
t6	A4: 访问instance引用的对象	

这里A2和A3虽然重排序了，但java内存模型的intra-thread semantics将确保A2一定会排在A4前面执行。因此线程A的intra-thread semantics没有改变。但A2和A3的重排序，将导致线程B在B1处判断出`instance`不为空，线程B接下来将访问`instance`引用的对象。此时，线程B将会访问到一个还未初始化的对象。

。

在知晓了问题发生的根源之后，我们可以想出两个办法来实现线程安全的延迟初始化：

1. 不允许2和3重排序；
2. 允许2和3重排序，但不允许其他线程“看到”这个重排序。

后文介绍的两个解决方案，分别对应于上面这两点。

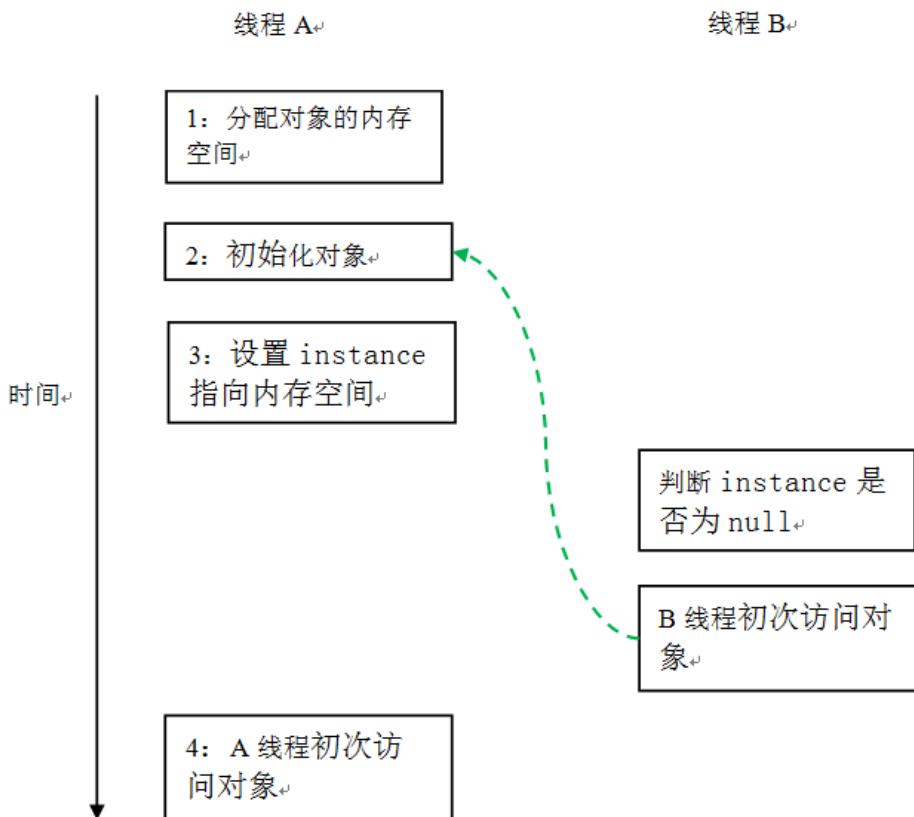
基于volatile的双重检查锁定的解决方案

对于前面的基于双重检查锁定来实现延迟初始化的方案（指DoubleCheckedLocking示例代码），我们只需要做一点小的修改（把instance声明为volatile型），就可以实现线程安全的延迟初始化。请看下面的示例代码：

```
public class SafeDoubleCheckedLocking {  
    private volatile static Instance instance;  
  
    public static Instance getInstance() {  
        if (instance == null) {  
            synchronized (SafeDoubleCheckedLocking.class) {  
                if (instance == null)  
                    instance = new Instance(); //instance为volatile, 现在  
                没问题了  
            }  
        }  
        return instance;  
    }  
}
```

注意，这个解决方案需要JDK5或更高版本（因为从JDK5开始使用新的JSR-133内存模型规范，这个规范增强了volatile的语义）。

当声明对象的引用为volatile后，“问题的根源”的三行伪代码中的2和3之间的重排序，在多线程环境中将会被禁止。上面示例代码将按如下的时序执行：



这个方案本质上是通过禁止上图中的2和3之间的重排序，来保证线程安全的延迟初始化。

基于类初始化的解决方案

JVM在类的初始化阶段（即在Class被加载后，且被线程使用之前），会执行类的初始化。在执行类的初始化期间，JVM会去获取一个锁。这个锁可以同步多个线程对同一个类的初始化。

基于这个特性，可以实现另一种线程安全的延迟初始化方案（这个方案被称之为Initialization On Demand Holder idiom）：

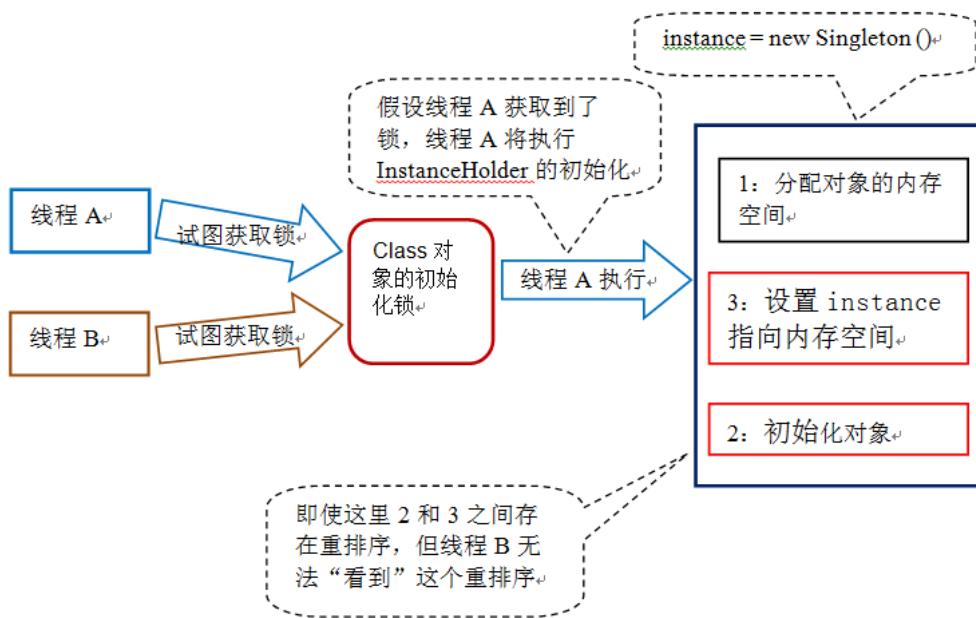
```

public class InstanceFactory {
    private static class InstanceHolder {
        public static Instance instance = new Instance();
    }

    public static Instance getInstance() {
        return InstanceHolder.instance; //这里将导致InstanceHolder类被初
        //始化
    }
}

```

假设两个线程并发执行getInstance()，下面是执行的示意图：



这个方案的实质是：允许“问题的根源”的三行伪代码中的2和3重排序，但不允许非构造线程（这里指线程B）“看到”这个重排序。

初始化一个类，包括执行这个类的静态初始化和初始化在这个类中声明的静态字段。根据java语言规范，在首次发生下列任意一种情况时，一个类或接口类型T将被立即初始化：

- T是一个类，而且一个T类型的实例被创建；
- T是一个类，且T中声明的一个静态方法被调用；
- T中声明的一个静态字段被赋值；
- T中声明的一个静态字段被使用，而且这个字段不是一个常量字段；
- T是一个顶级类（top level class，见java语言规范的§7.6），而且一个断言语句嵌套在T内部被执行。

在InstanceFactory示例代码中，首次执行`getInstance()`的线程将导致`InstanceHolder`类被初始化（符合情况4）。

由于java语言是多线程的，多个线程可能在同一时间尝试去初始化同一个类或接口（比如这里多个线程可能在同一时刻调用`getInstance()`来初始化`InstanceHolder`类）。因此在java中初始化一个类或者接口时，需要做细致的同步处理。

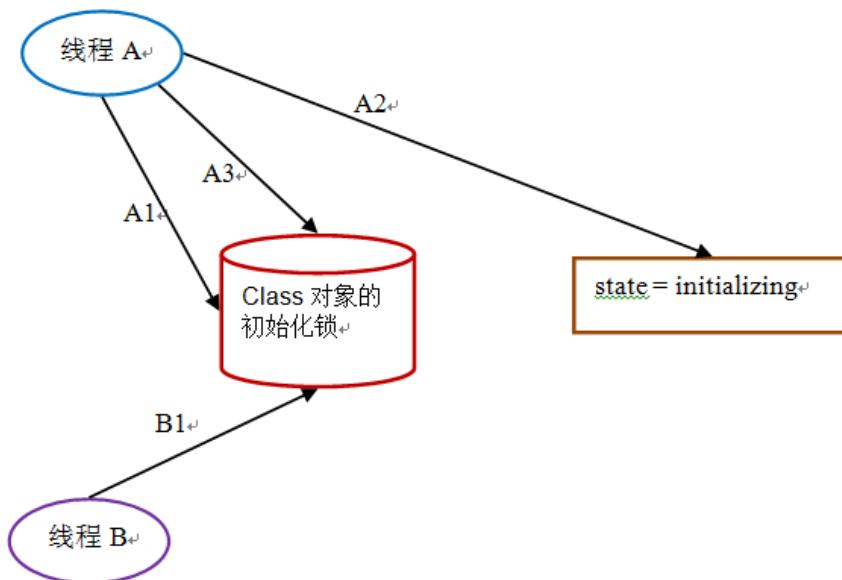
Java语言规范规定，对于每一个类或接口C，都有一个唯一的初始化锁LC与之对应。从C到LC的映射，由JVM的具体实现去自由实现。JVM在类初始化期间会获取这个初始化锁，并且每个线程至少获取一次锁来确保这个类已经被初始化过了（事实上，java语言规范允许JVM的具体实现在这里做一些优化，见后文的说明）。

对于类或接口的初始化，java语言规范制定了精巧而复杂的类初始化处理过程。

java初始化一个类或接口的处理过程如下（这里对类初始化处理过程的说明，省略了与本文无关的部分；同时为了更好的说明类初始化过程中的同步处理机制，笔者人为的把类初始化的处理过程分为了五个阶段）：

第一阶段：通过在Class对象上同步（即获取Class对象的初始化锁），来控制类或接口的初始化。这个获取锁的线程会一直等待，直到当前线程能够获取到这个初始化锁。

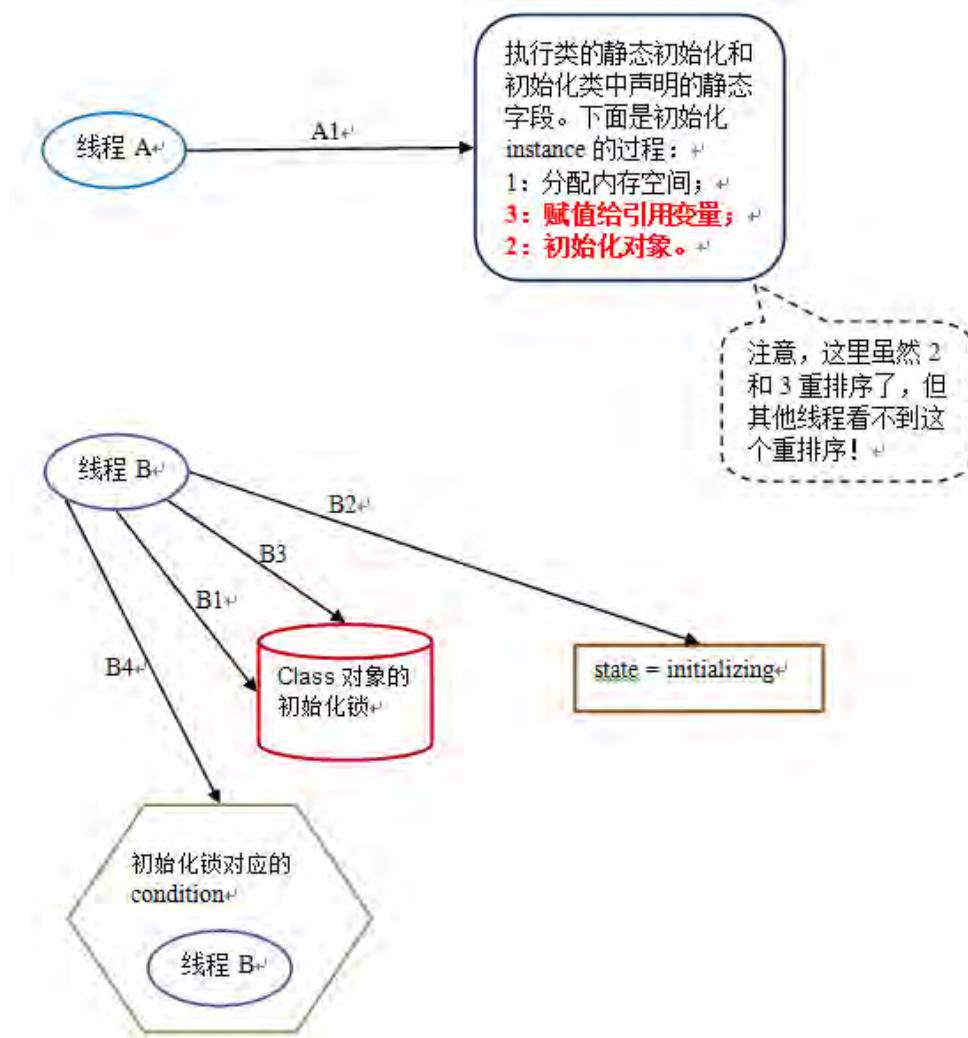
假设Class对象当前还没有被初始化（初始化状态state此时被标记为state = n oInitialization），且有两个线程A和B试图同时初始化这个Class对象。下面是对应的示意图：



下面是这个示意图的说明：

时间	线程A	线程B
t1	A1: 尝试获取Class对象的初始化锁。这里假设线程A获取到了初始化锁	B1: 尝试获取Class对象的初始化锁，由于线程A获取到了锁，线程B将一直等待获取初始化锁
t2	A2: 线程A看到线程还未被初始化（因为读取到state == noInitialization），线程设置state = initializing	
t3	A3: 线程A释放初始化锁	

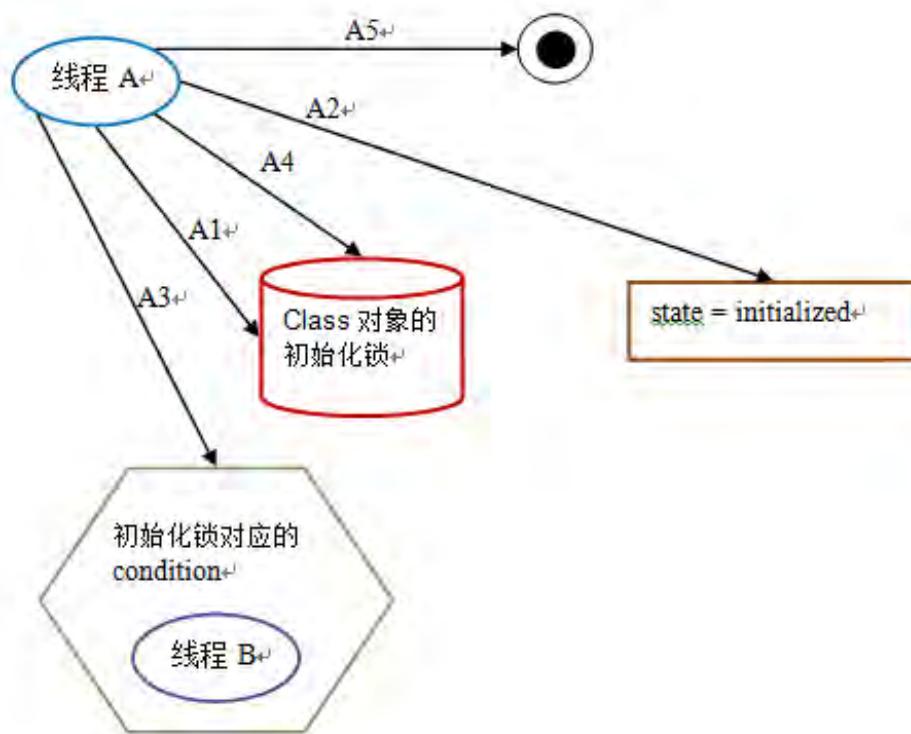
第二阶段：线程A执行类的初始化，同时线程B在初始化锁对应的condition上等待：



下面是这个示意图的说明：

时间	线程A	线程B
t1	A1: 执行类的静态初始化和初始化类中声明的静态字段	B1: 获取到初始化锁
t2		B2: 读取到state == initializing
t3		B3: 释放初始化锁
t4		B4: 在初始化锁的condition中等待

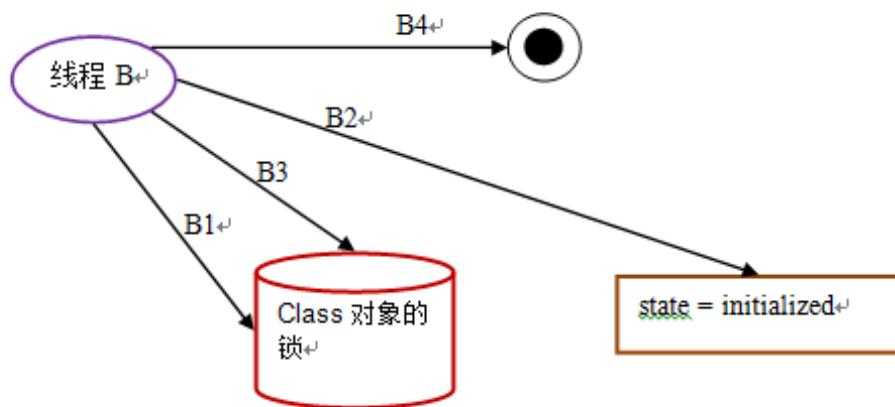
第三阶段：线程A设置state = initialized，然后唤醒在condition中等待的所有线程：



下面是这个示意图的说明：

时间	线程A
t1	A1：获取初始化锁
t2	A2：设置state = initialized
t3	A3：唤醒在condition中等待的所有线程
t4	A4：释放初始化锁
t5	A5：线程A的初始化处理过程完成

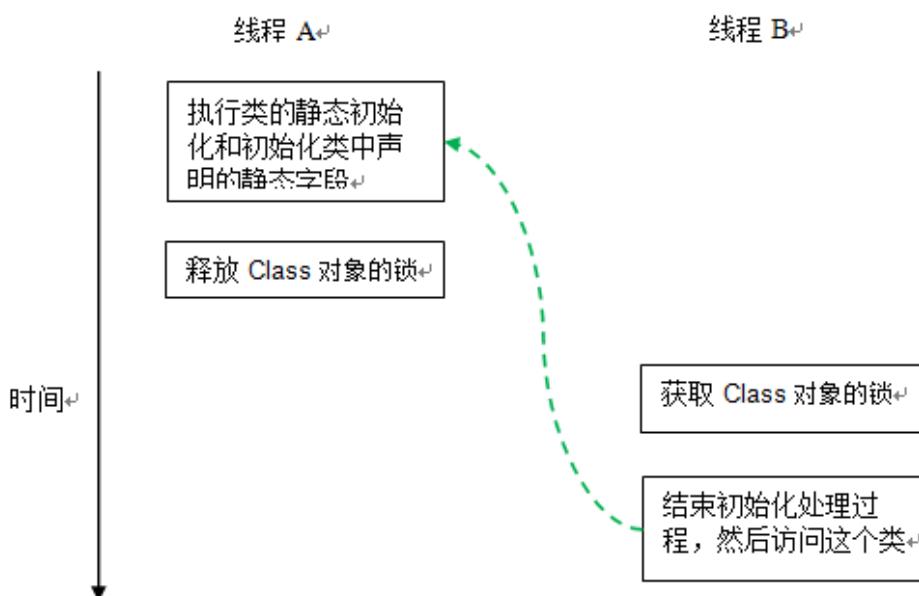
第四阶段：线程B结束类的初始化处理：



下面是这个示意图的说明：

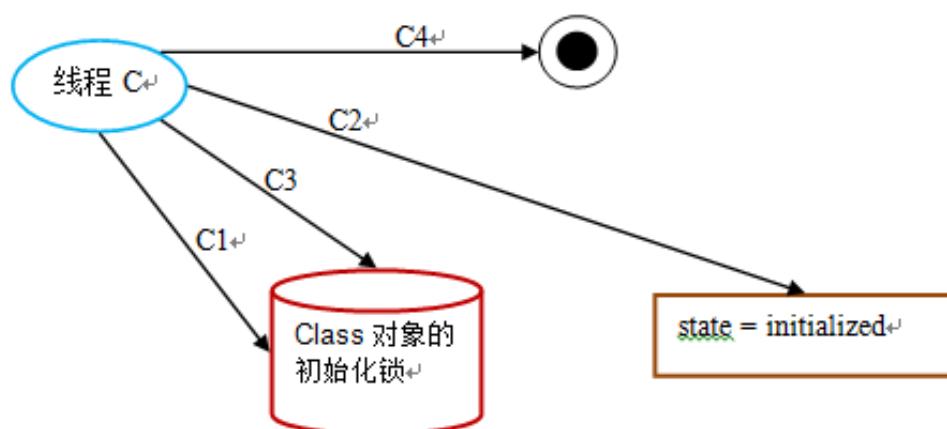
时间	线程B
t1	B1: 获取初始化锁
t2	B2: 读取到state == initialized
t3	B3: 释放初始化锁
t4	B4: 线程B的类初始化处理过程完成

线程A在第二阶段的A1执行类的初始化，并在第三阶段的A4释放初始化锁；线程B在第四阶段的B1获取同一个初始化锁，并在第四阶段的B4之后才开始访问这个类。根据java内存模型规范的锁规则，这里将存在如下的happens-before关系：



这个happens-before关系将保证：线程A执行类的初始化时的写入操作（执行类的静态初始化和初始化类中声明的静态字段），线程B一定能看到。

第五阶段：线程C执行类的初始化的处理：

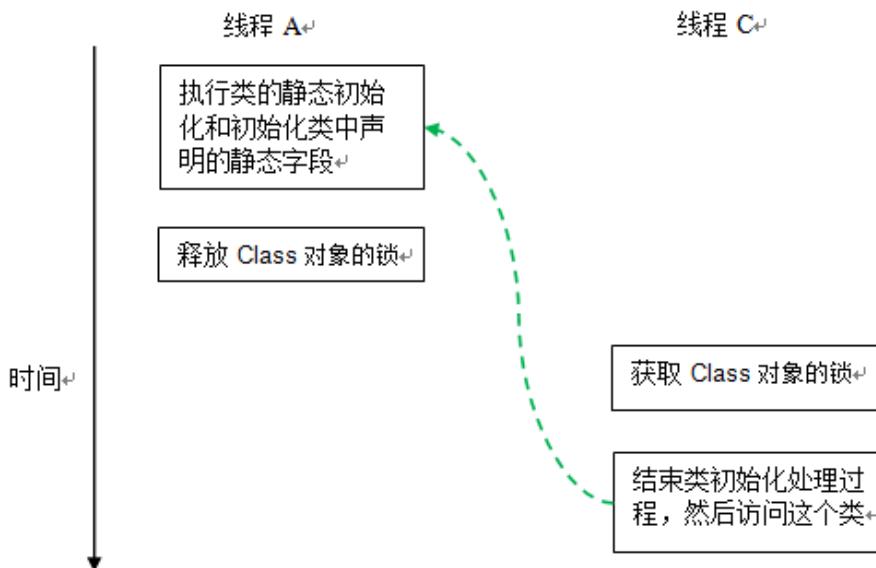


下面是这个示意图的说明：

时间	线程B
t1	C1: 获取初始化锁
t2	C2: 读取到state == initialized
t3	C3: 释放初始化锁
t4	C4: 线程C的类初始化处理过程完成

在第三阶段之后，类已经完成了初始化。因此线程C在第五阶段的类初始化处理过程相对简单一些（前面的线程A和B的类初始化处理过程都经历了两次锁获取-锁释放，而线程C的类初始化处理只需要经历一次锁获取-锁释放）。

线程A在第二阶段的A1执行类的初始化，并在第三阶段的A4释放锁；线程C在第五阶段的C1获取同一个锁，并在第五阶段的C4之后才开始访问这个类。根据Java内存模型规范的锁规则，这里将存在如下的happens-before关系：



这个happens-before关系将保证：线程A执行类的初始化时的写入操作，线程C一定能看到。

*注1：这里的condition和state标记是本文虚构出来的。Java语言规范并没有硬性规定一定要使用condition和state标记。JVM的具体实现只要实现类似功能即可。

*注2：Java语言规范允许Java的具体实现，优化类的初始化处理过程（对这里的第五阶段做优化），具体细节参见java语言规范的12.4.2章。

通过对基于volatile的双重检查锁定的方案和基于类初始化的方案，我们会发现基于类初始化的方案的实现代码更简洁。但基于volatile的双重检查锁定的方案有一个额外的优势：除了可以对静态字段实现延迟初始化外，还可以对实例字段实现延迟初始化。

总结

延迟初始化降低了初始化类或创建实例的开销，但增加了访问被延迟初始化的字段的开销。在大多数时候，正常的初始化要优于延迟初始化。如果确实需要对实例字段使用线程安全的延迟初始化，请使用上面介绍的基于volatile的延迟初始化的方案；如果确实需要对静态字段使用线程安全的延迟初始化，请使用上面介绍的基于类初始化的方案。

参考文献

1. [Double-checked locking and the Singleton pattern](#)
2. [The Java Language Specification, Java SE 7 Edition](#)
3. [JSR-133: Java Memory Model and Thread Specification](#)
4. [Java Concurrency in Practice](#)
5. [Effective Java \(2nd Edition\)](#)
6. [JSR 133 \(Java Memory Model\) FAQ](#)
7. [The JSR-133 Cookbook for Compiler Writers](#)
8. [Java theory and practice: Fixing the Java Memory Model, Part 2](#)

个人简介

程晓明，java软件工程师，专注于并发编程，现就职于富士通南大。个人邮箱：asst2003@163.com。

感谢[方腾飞](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/double-checked-locking-with-delay-initialization>

相关内容

- [Ruby on Rails与Java，哪一个才适合你呢？](#)
- [恒拓开源陈操谈去IOE方案的普及对独立开发商的机遇与挑战](#)
- [书评：Java核心编程卷1——基础](#)
- [Peter Kriens重返OSGi联盟](#)
- [讨论：Java的发展趋势向好向坏？](#)

避开那些坑 | Void

从CDN到云计算，在变中寻找不变

作者 董伟

从2006年7月，我开始在蓝汛工作，在这些年中我历任软件工程师、高级软件工程师和团队主管，参与并主导开发了公司的Squid配置管理系统、资源配置管理系统和计费采集系统这三大系统。同时，经历了蓝汛的带宽从100G发展到1.5个T的过程；也经历了蓝汛在美国纳斯达克上市的时刻。虽然现在离开了那里，转而投身云计算IaaS平台的技术创业和开发，但是在那积累的分布式系统开发的丰富经验，仍然令我受益良多，这篇文章就是从技术角度对这些年的一个总结。

从100G到1.5个T

我刚到公司时，公司对外服务提供的带宽刚到100G，我记得当时还专门做了T恤衫发给大家，以示庆祝。从那年开始，公司的业务每年都以70%到80%左右的速度增长。

业务的高速发展，对我们这些技术人员提出了挑战。当时蓝汛使用基于硬件的专用物理设备，比如3DNS、CacheFlow、NetCache、Array Networks等等。客户如果需要CDN加速需求，运维人员需要登录到多种服务器上手工进行相应的配置维护，既容易出错，又效率低下。后来经过公司不断对研发的投入以及不断的创新实践，我们研发了拥有自主知识产权的流量调度管理系统—SSR（Scalable Service Routing，智能流量路由系统），以及基于Squid的缓存组件，同时开发了用来进行相应配置及下发的系统，慢慢的就不再依赖专用的硬件设备了，在大幅度降低了服务成本的同时，也让服务的开通及运维更加自动化了。这块的工作也就从瓶颈变成了推动力，推动公司的业务向前跑得更快。

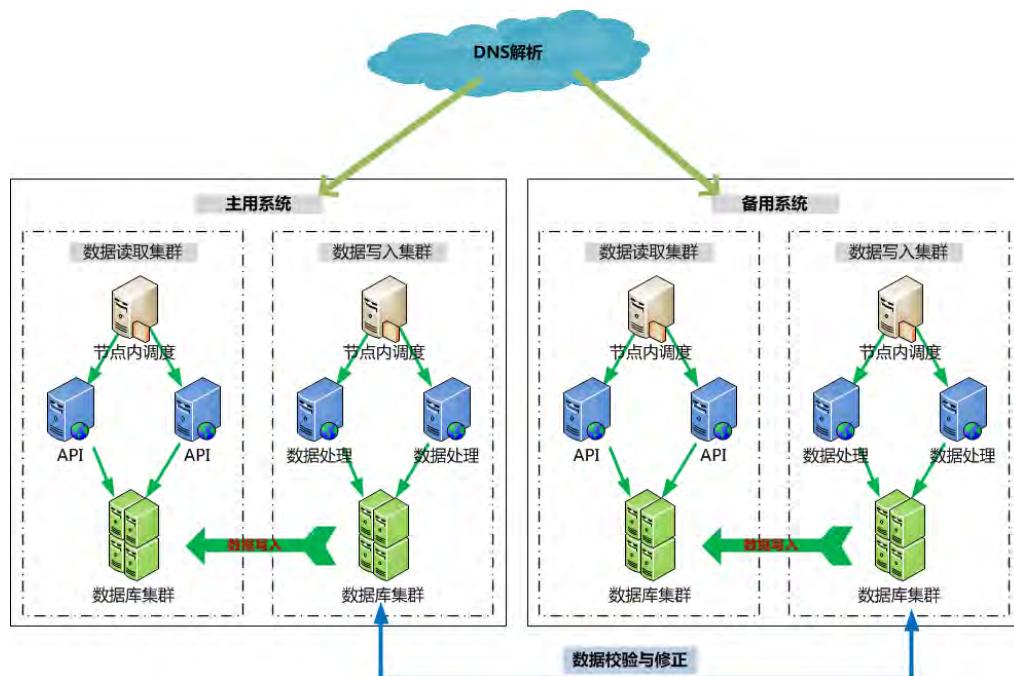
现在做云计算IaaS平台，回头想一想，现在跟当初有许多的相似之处。IaaS，不也是着眼于降低企业IT运营成本，提高企业运营灵活性，同时又将IT支持人员从繁琐、低效、易于出错的物理服务器购买、上架、安装、配置、优化的过程中解放出来么？用基于API的方式，管理、调度虚拟化的资源，只要几分钟，就可以完成几十台甚至上百台云服务器的安装、部署、优化。人，可以因此享有更多自由，完成更有价值、更有创意的事情。这就是技术的力量。

三个大型分布式系统

接下来说说我当时做过的3个大型分布式系统。

- Squid配置管理系统。这个系统的主要作用是：用来管理公司的Squid缓存设备。因为我们对于服务质量的苛刻要求，公司在全国一、二、三线城市基本都做了节点的覆盖，并在部分城市间设有专线，而且在海外一些国家也设置了缓存节点，设备总数上万台。每天运维人员都需要通过该系统来操作这些缓存设备以实现对频道内容的新增、变更、优化等等。配置内容基本上在5分钟之内就可以下发到设备上并生效。
- 资源配置管理系统。这个系统的主要作用是：它是公司IT资源管理的核心，很多系统的运行都需要通过与它进行API的交互来完成，它还管理并配置智能流量路由系统SSR的调度策略，以及管理并配置公司的各种软硬件资源等等。同样该系统要与全网的上万台设备进行数据交互，而且更大的难点在于，这些资源之间存在复杂的运算关系，逻辑复杂度之高可想而知。
- 计费采集系统。这个系统的主要作用是：收集全网用于提供加速服务的设备所产生的计费数据。截止到2012年第三季度，我们的活跃用户数已经超过1千个，每个用户下面的频道数不等，多的几百个，有的甚至上千，少的几个，几十个，全网的频道好几万个，由于我们采用每5分钟采集一次数据的方式，所以一天就是288个采集点，而一个频道可能分布在多台设备上，这样几万个频道一天算下来，所采集到的数据有十几亿条，采集的数据之庞大对系统处理能力的要求很高。

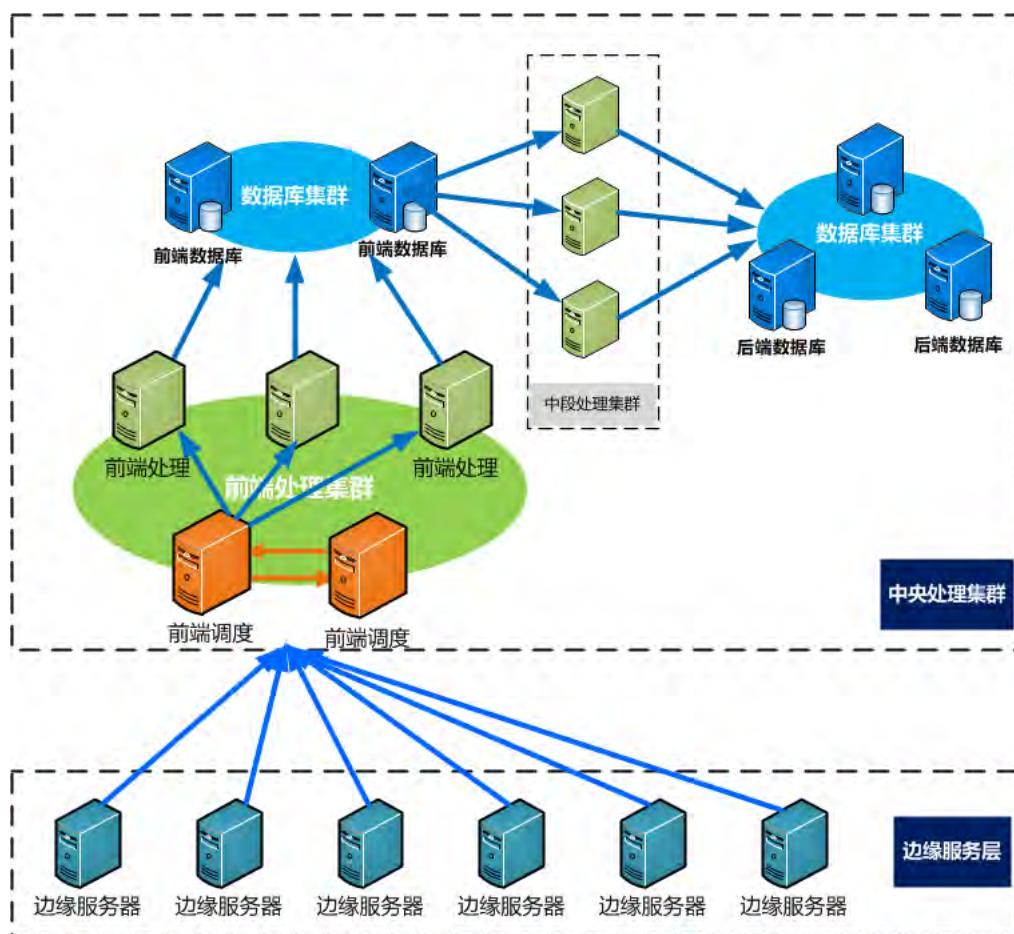
开发这些支持CDN服务的大型分布式系统，主要难点有两个：大并发的处理和大数据量的处理。这两个难点在技术上有一个共同点：如何保证服务的稳定性，保证系统7X24持续可用。当时我规划的一种方式是：边缘设备通过域名解析，以负载均衡的方式（便于容灾切换），将文件和数据收集到集中的集群当中来，同时为了保证系统的持续可用，对整个集群做异地的容灾和定期的轮岗服役及演练。同时为了保证集群两边数据的一致性，定时在集群之间做数据准确性校验和修正工作。如下图所示：



系统简要逻辑图

对于大并发的处理来说，我们要做的：一是优化数据的读写性能，以提高单位时间内的吞吐量，二是横向扩展读写能力以提高总的处理能力。如上图，为了提升读写性能，做了读写分离的架构，而且在数据处理层、API层、数据库层都构建了扩展点。

对于大数据量的处理来说，我们要做的：一是对数据来源进行分类处理，比如会用专门的流程来处理流媒体服务器产生的数据，而用另外的流程来处理Squid服务器产生的数据。二是对数据进行多步加工，每一次加工只做很少的事情，将中间结果保留，然后再由下一阶段的处理程序在此基础上进行加工。这样环环相扣最终形成完整的可扩展的处理链条，如下所示为分阶段处理逻辑架构图：



分阶段处理逻辑架构图

另外我们会根据一些客户的需求做定制。举个例子：比如某些大客户，他们对数据实时性要求就比较高，那么我们就会采用另外的策略，开辟专用的通道，同时尽量缩短中间处理环节以减少总耗时，而且开发专有API来满足客户对于实时性数据的访问需求。

这些难点，有的来自公司内部本身的业务快速增长带来的复杂性以及处理吞吐量的要求，有的来自客户的定制化需求。开发这几个系统的经历，对我现在设计开

发目前这套“云计算综合管理平台系统”很有帮助。举个例子：在开发层面，我会去想办法抽象客户的需求，在个性和通用性上取得一个平衡，对于非常个性化的需求考虑用插件化的方式接入，当该功能不再使用时即从系统中注销掉。

目前整个系统都是用Python写成，在架构上使用异步消息队列RabbitMQ中间件作为服务总线，来完成任务的异步调用以及云主机等的状态同步。由于采用Python的模块化的开发方式，对一个插件的启用或停用，只要一行代码就可以搞定。

回顾这9年多的开发历程，变的是在每个不同阶段从事不同业务的开发，但不变的依然对系统整体不断的完善、思考和探索，经过一番摸爬滚打，积累的成果最终体现在这些代码和架构中。

云+CDN：大规模web服务必行之路

CDN解决静态内容的传送和扩展问题，但是无法解决动态内容的传送和扩展给源站带来的建设、部署和扩展上的难题，而有了云计算的支持，对于源站的动态内容扩展是一个极好的解决方案。静态内容扩展实际上就是存储和网络扩展的问题，动态内容扩展的本质是计算扩展，而计算的扩展分为横向扩展和纵向扩展，这两方面在云平台上都可以在极短的时间内完成。

之前在蓝汛积累了大量分布式系统开发和管理经验，对于我目前搭建迅达云成的云计算综合管理平台系统很有帮助。我们现在想要做到的，是帮助客户搭建一套通过代码和自动化方式驱动的、有高控制权的基础设施。我们通过云计算为客户提供资源，提供编程接口，让客户通过自动化方式完成扩展，从过去繁复的工作中快速解脱出来，实现业务的快速伸缩。

目前，我们正在致力于整合接入一些CDN系统，在此基础上进行一定的封装，屏蔽底层差异性，对外暴露统一的CDN调度的API接口，这样客户可以直接在我们的云平台上跟CDN做结合，调用一站式服务的API，从而达到带宽和计算资源的充分利用。

	计算	存储	网络
集中式	大型机	集中存储	专线
分散式	个人电脑	分散存储	互联网
分布式	云计算	分布式存储	SDN...

其实回顾下IT技术的发展历史，可以看到是围绕着计算、存储和网络这三个方面展开的，而且各自的总体趋势都是从集中式到分散式，从分散式到分布式。在分布式这个层面，必须要通过虚拟化技术，才能实现更纯粹的分布式，因为虚拟化技术会从多个层面来对数据及系统提供更加安全、有效的冗余及保护。计算虚拟化、存储虚拟化目前都已经基本实现，正在走向不断完善的过程，而以OpenFlow和SDN为代表的网络虚拟化，是整个IT业界正在跨越的另一座山峰。在当下，云结合CDN，已经可以满足大规模Web服务开发的绝大部分要求。

作者简介：董伟，9年互联网从业经验。曾就职于天极网、北京蓝汛(ChinaCache)等国内知名IT公司，历任软件工程师、高级软件工程师、团队主管等职位。现任迅达云CTO。喜欢思考，喜欢攻克、钻研技术难点。

原文链接：<http://www.infoq.com/cn/articles/find-the-same-in-the-variable-from-CDN-to-cloud-computing>

相关内容

- [PeerCDN：使用WebRTC构建基于浏览器的P2P CDN](#)
- [阿里云计算资深总监唐洪谈飞天现状以及5K项目发展](#)
- [云计算技术的实践](#)
- [云计算与持续集成——七牛的实践](#)
- [博文共赏：道哥的网站优化指南之数据库缓存、CDN与云存储](#)

新品推荐 | Product

Oracle在JavaOne大会上揭开了Project Avatar的面纱

作者 [Dan Woods](#) , 译者 [孙镜涛](#)

Oracle在JavaOne大会期间发布了Project Avatar的开源版本。Avatar是一个Web应用程序框架，它关注于构建“现代HTML5应用程序”，但是需要应用程序开发者有“轻微的JavaScript知识”。

原文链接：<http://www.infoq.com/cn/news/2013/10/oracle-unveils-avatar>

Java内存数据网格Hazelcast 3.0支持连续查询和条目处理

作者 [Sriini Penchikala](#) , 译者 [马德奎](#)

Java开源内存数据网格Hazelcast的最新版本支持“条目处理（Entry Processing）”、多线程执行、连续查询和延迟索引。它还使用服务提供程序接口（SPI）重新实现了所有现有的分布式对象，如Map、Queue和ExecutorService等。

原文链接：<http://www.infoq.com/cn/news/2013/10/hazelcast-3.0>

新的.NET编译器——RyuJIT 项目

作者 [Jeff Martin](#) , 译者 [孙镜涛](#)

Microsoft正在开发一个代号为“RyuJIT”的新即时（Just-In-Time, JIT）编译器，该编译器最终会被用于运行.NET应用程序。

原文链接：<http://www.infoq.com/cn/news/2013/10/ryujit>

使用Ruby开发iOS游戏

作者 [Manuel Pais](#) , 译者 [孙镜涛](#)

Integrallis网站的创始人Brian Sam-Bodden在Barcelona Ruby大会上演示了在没有任何Object-C知识的情况下，如何使用普通的Ruby语言利用RubyMotion 和开源2D图形类库快速地创建iOS平台上的2D游戏。

原文链接：<http://www.infoq.com/cn/news/2013/10/ios-games-ruby>

Backbone 1.1.0 发布，部分内容与版本1.0不兼容

作者 [Brian Rinaldi](#)，译者 [孙镜涛](#)

在JavaScript社区中Backbone.js 是使用最广泛的前端构建框架之一，该框架最近发布了一个新的更新。尽管这并不是一个非常重要的版本号更新，但是来自于一些用户的报告表明其中的一些变化可能与为了使用版本1.0而设计的应用程序不兼容。

原文链接：<http://www.infoq.com/cn/news/2013/10/backbone-1.1.0-released>

Rubinius 2.0发布，实现了Ruby 2.1

作者 [Mirko Stocker](#)，译者 [马德奎](#)

两年多以后，Rubinius团队发布了2.0版本，带来了经过改进的多线程支持，并实现了即将到来的Ruby 2.1。

原文链接：<http://www.infoq.com/cn/news/2013/10/rubinius-2>

Upyun 又拍云存储

又拍云存储主要为用户提供静态文件云存储、云处理、云分发三块业务，我们致力于让您的创业变得简单。凭着安全、稳定、快速的产品特性，当前已有 30000 多家互联网企业选择使用又拍云存储的服务，客户包括：会说话的汤姆猫，捕鱼达人，one · 一个，百姓网，知乎，36 氪，雪球财经等众多知名企业。

产品优势



省钱

按需付费，用多少付多少，相比传统CDN按峰值带宽计费成本更低。



省心

业务快速增长，只需扩展相应的又拍云套餐即可，可以把更多的精力放在产品运营上。



更安全

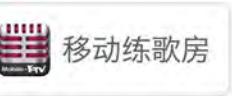
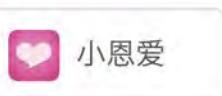
上传文件自动3备份，不用担心数据丢失问题。



多功能

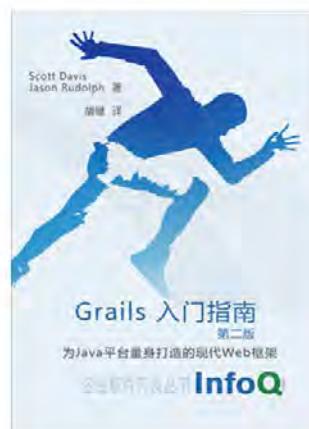
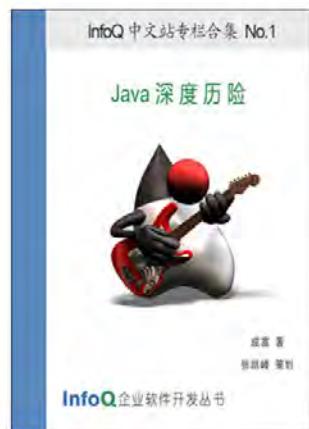
CDN加速，自定义30种缩略图，token防盗链，图片打水印，音视频处理等众多实用酷功能。

客户案例



InfoQ 软件开发丛书

欢迎免费下载



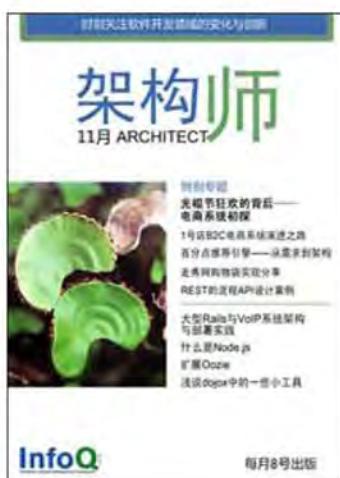
商务合作: sales@cn.infoq.com

读者反馈/内容提供: editors@cn.infoq.com

架构师

www.infoq.com/cn/architect

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

每月8号出版



InfoQ

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

封面植物

山楂



茶花，又名山茶花，山茶科植物，属常绿灌木和小乔木。古名海石榴。有玉茗花、耐冬或曼陀罗等别名，又被分为华东山茶、川茶花和晚山茶。茶花的品种极多，是中国传统的观赏花卉，“十大名花”中排名第七，亦是世界名贵花木之一。分布于重庆、浙江、四川、江西及山东；日本、朝鲜半岛也有分

布。山茶是常绿阔叶灌木或小乔木。枝条黄褐色，小枝呈绿色或绿紫色至紫褐色。叶片革质，互生，椭圆形、长椭圆形、卵形至倒卵形，长4~10cm，先端渐尖或急尖，基部楔形至近半圆形，边缘有锯齿，叶片正面为深绿色，多数有光泽，背面较淡，叶片光滑无毛，叶柄粗短，有柔毛或无毛。花两性，常单生或2~3朵着生于枝梢顶端或叶腋间。花梗极短或不明显，苞萼9~10片，覆瓦状排列，被茸毛。花单瓣，花瓣5~7片，呈1~2轮覆瓦状排列，花朵直径5~6cm，色大红，花瓣先端有凹或缺口，基部连生成一体而呈筒状；雄蕊发达，多达100余枚，花丝白色或有红晕，基部连生成筒状，集聚花心，花药金黄色；雌蕊发育正常，子房光滑无毛，3~4室，花柱单一，柱头3~5裂，结实率高。蒴果圆形，外壳木质化，成熟蒴果能自然从背缝开裂，散出种子。山茶花为常绿花木，开花于冬春之际，花姿绰约，花色鲜艳，郭沫若同志盛赞曰：“茶花一树早桃红，百朵彤云啸傲中。”对云南山茶郭老也曾赋诗赞美：“艳说茶花是省花，今来始见满城霞；人人都道牡丹好，我道牡丹不及茶。”

1kg

多背一公斤
.org

爱自然 | 更爱孩子



促进软件开发领域知识与创新的传播

架构师

11月 ARCHITECT



特别专题

- 软件测试的方方面面
- GUI功能测试自动化模式
- 软件测试转型之路
- 全程软件测试实践：从需求到运营
- AWDC（阿里云开发者大会）
- 云的蝴蝶效应
- 阿里云唐洪谈飞天现状以及5K项目
- 如何挑选合适的大数据或Hadoop平台
- HotSpot和OpenJDK入门

InfoQ 

每月8号出版

架构师 11 月刊

每月 8 日出版

本期主编：侯伯薇

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：侯伯薇，InfoQ 中文站翻译团队主编

侯伯薇，生于丹东凤城，学在春城长春，工作在滨城大连；虽已年过而立，但自问童心未泯；对代码热情不减，愿与天下程序员共同修炼，不断提升。译有《学习 WCF》、《Expert C# 2008 Business Objects》。