InfoQ 时刻关注企业软件开发领域的变化与创新

架构师 Architect

试刊号

Alexandru Popescu谈InfoQ.com网站架构

InfoQ首席架构师Alexandru Popescu谈论了InfoQ的架构、WebWork与DWR的集成、Hibernate与JCR、Hibernate可扩展性、MySQL拷贝、最新 InfoQ视频流系统、视频编码过程、网站搜索和InfoQ未来规划。

一种正规的性能调优方法：基于等待的调优

企业Java应用的性能调优是一项艰巨的、有时甚至是徒劳的任务，本文尝试把性能调优活动变成一种"科学"范畴内的行为。

经济困局中的SOA

当日子紧张的时候，越是该紧迫地重新对业务进行思考。而改进低效的业务流程，为企业挽回成本和带来业务价值，正是 SOA的机会。

InfoQ中文站
www.infoq.com/cn

每月8日出版
InfoQ中文站制作发布

篇首语

合适就好

最近我参加了一个由中欧商学院举行的交流活动，主题是讨论当前经济形势下软件外包产业的发展方向。期间，有位老师分享了一个很有意思的案例，他提到有次他参加另外一个大型外包论坛时，听到有的城市外包产业发展的非常快，单子非常多，包括国外大公司和国内公司的；而有的城市相关负责人对此非常不满，说外包就是接国外的单子，那些不守“规矩”的城市对“外包”的定义有误。结果大家也能猜得到，即这些守规矩的城市外包产业是一直缓步不前的。其实道理很简单，城市发展外包产业的目的是增加就业机会和经济收入，只要是符合这些需求的单子不就很好吗？

又想起从前和台湾的一位知名技术作者聊天时，一旁有朋友请教说，现在动态语言那么多，应该学习哪一种好呢？那位作者微微一笑说，我现在在用 Lua。朋友很吃惊地再问，这个不流行啊，为何要学习这个？“能解决我的问题就好啊”，作者回答说。不知这位朋友最终有没有明白作者的意思，希望他能理解“合适就好”这几个字的含义。语言、框架、工具等当然有好坏之分，但是如果只是将目光放在孰优孰劣上，而不能潜心研究并将其付之于实践，不就沦为“空谈”了吗。在目前所运行的软件系统中，我们可以看到其背后的平台、语言等是各种各样，MySpace 是基于.NET 平台的，淘宝网是基于 Java 的，而 Google 则推崇使用 Python 等，我还听说现在许多大型的电力系统还依然运行在 C++平台上，这有什么关系吗？每门技术自有其缺点，但它们也都自有其优点，如果它的优点恰好能符合你的需要，用它就好了。重要的是，你有没有使用好它的能力。

还有个例子，是从前和 BEA（现在已经被 Oracle 收购）的销售人员聊天时了解到的，他说现在 BEA 的 WebLogic 产品在日本市场很好，但是他们用的多是 5.0 或者 6.0 的版本，我们试图说服他们更换到最新的 10.0 版本上，他们丝毫不为之所动，还很纳闷地问我们：现在系统运行的很稳定，为什么要换？另外，你会发现这些产品的支持工程师对产品的特性、

功能和管理等理解的非常深入，每一个能够优化的地方都进行了调整优化。

这儿提“合适就好”并不是说让我们不再追求进步，而是强调对任何一个策略、技术平台或者语言、工具，如果我们没有做选择，那么就根据自己的系统选择最合适的（而不是最好的），而一旦做了选择，那么就深入地研究，发掘它们的潜力，而不是在选择面前犹豫徘徊。

霍泰稳

目　录

[人物专访]

[热点新闻]

[推荐文章]

[新品推荐]

[人物专访]

Alexandru Popescu

谈 InfoQ.com 网站架构

摘要：在 QCon 伦敦 2008 会议的采访中，InfoQ 首席架构师 Alexandru Popescu 谈论了 InfoQ 的架构、WebWork 与 DWR 的集成、Hibernate 与 JCR、Hibernate 可扩展性、MySQL 拷贝、最新 InfoQ 视频流系统、视频编码过程、网站搜索和 InfoQ 未来规划。

Alexandru Popescu，InfoQ.com 首席架构师和联合创始人。同时，他作为 TestNG 框架的联合创始人、WebWork 和 Magnolia 项目的提交者，参与了许多开源工程和前沿技术。在 AspectWerkz 项目合并到 AspectJ 之前，Alexandru 曾经是三个提交者之一。他的博客地址是 http://themindstorms.blogspot.com/。

Ryan：大家好，我是 Ryan Slobojan，坐在我旁边的是 InfoQ.com 的首席架构师 Alexandru Popescu。Alexandru，能否告诉我们 InfoQ 网站的一些架构信息—它是什么样子的？又是如何构建的？

Alex：你可以从两种不同角度审视 InfoQ 的架构：从我们读者的角度看，InfoQ 就像是一个普通的网站；但是对于我们的编辑和在后台工作的人员来说，它则是一个地地道道的 CMS（内容管理系统）。因此，你所看到的 InfoQ 建立在一个自制的 CMS 的基础之上，它把内容与用户账号、跟踪系统、广告机制等等集成。我们可以从一个更易于理解的角度来描述 InfoQ 网站—它是一个 Web 应用，即使是 CMS，你也可以看作是一个 Web 应用，它有通常的分层结构：表现层、服务层和持久化层。

两年半之前，当我启动这个项目的时候，面临着很多有趣的选择。例如，持久化方面不但基于关系型数据库，而且使用 JCR API 存储内容。同时，我们不得不在基于组件的 Web 框

架和基于动作的框架中二者选其一，并最终选择了后者。我们认为它更贴近我们解决方案的设计，即使我们可能需要一些基于 portlet 的东西……我想说那时侯 portlet 规范非常差，希望以后我不会让大家太失望。你可以想象作为一个三层结构是多么的简单，你应该能够猜到：一点 Spring、一点 WebWork、一点 Hibernate 和 JCR API。

Ryan：能否给我们描述一下，当你作为一个用户和一个作者发出请求时，内部会发生什么变化？

Alex：当然，希望我没有记错一些细节。让我们从浏览器开始。通常有两种方式访问我们的应用，要么是通过浏览器正常访问，要么是通过 AJAX 请求，如 XMLHTTPRequest，然后请求进入 WebWork 或者 DWR。如果是普通请求，则它会经过 WebWork 处理。如果是 AJAX 请求，则进入 DWR，然后分派到服务层，这层的全部家当只不过是 Spring 和一些采用 AspectJ 的 AOP，目的是增强我们的模型。然后，请求会进入持久化层，我刚才已经提到这层被分割为 Hibernate 和 JCR。

因此，最后我们拥有两种不同的存储。此时你可能会问为什么我们选择了两种解决方案来存储信息，这些信息本可以采用同一种存储方式。问题是，当我们设计 InfoQ 的时候，我们并不确定模型会是什么样子的，也不确定我们的内容随着时间会如何变化。同时，在关系型模式下处理这些变化非常困难，在不同版本之间迁移和维护数据等等是非常复杂的。而 JCR API 明确支持非结构的内容和很多其他特性，比如版本化、全文索引，我们充分利用了这些功能。

同时，对于编辑工具，它与你看到的 InfoQ.com 几乎完全一样，除了不太花哨。因为我们设计的是同一个应用，所以使用相同的栈、几乎相同的 API，在构建时我们把 API 分为两部分，对外开放的部分使用只读 API，而对于编辑工具，我们使用可读/写存取 API，不过本质上它们都是基于同一份源代码。

Ryan：你刚才提到使用 WebWork 和 DWR 处理前端。请问它们能够无缝集成吗，或者存在哪些挑战吗？

Alex：起初我们像往常一样启动了这个工程。我是说我们过去有一个处理 DWR 和 WebWork 应用的模型。但是最终我意识到，如果存在一个通用的方式访问和判断我们是应该通过 DWR 还是 WebWork 处理请求的话，对我和开发人员都省力。于是，我建立一个模型把这两个框架集成在一起。同时，通过这种方式我也对 DWR 贡献了代码，所以现在大家都可以使用它，它非常通用，你可以立刻把它应用到 Struts 2 或类似的技术。如今，我们在编写代码处理 HTTP 时，终于能够延迟决定如何处理请求：是通过普通的请求/响应周期还是通过 AJAX 方式。

Ryan：如果你有机会从头重新设计 InfoQ.com，你会保留哪些，改变哪些？

Alex：很多人提过这个问题，这可能是对我最具挑战性的问题。你能够想象，在同一个项目上工作两年半之后，你会有很多不同的想法来改变和提高一些东西。现在，我可能会说我不打算改变任何事情。我可能会尝试不同的方法来看一看它们的效果，但是到目前为止，我们在项目开始时选择的解决方案都工作的非常好。

我可能会研究一下如何标准化访问存储的 API，在 Hibernate 和 JCR 之上创建一个通用的 API，这样开发人员不再费心思考真正的数据到底存储到何处。这可能会涉及到内部 API，不会变化很大。

Ryan：能否提供一些关键的数据，比如 InfoQ 每天处理多少用户请求？其可扩展性呢？

Alex：目前我能够对外公布的数据就是每月的独立访问用户量。你可以通过网站的左上角看到这个信息。目前我们每月的独立访问用户数大约是 25 万。

Ryan：Hibernate 真的可以扩展吗？这种扩展性有用吗？它是一个适合扩展的框架吗还是……还有一个问题是你对数据库分区吗？

Alex：我们一个问题一个问题的看。到目前为止，我还没有在 Hibernate 的层面上发现任何问题。我是说我们甚至都没有优化查询。我们使用的就是 Hibernate 自动生成的东西，性能也非常非常好。其次，由于性能不错目前我们还没有对数据分区，即使我们需要在后台处理海量的数据。我们一直在关注网站的性能，但是现在还不需要做些什么。另外一件关于架构的趣事是，唯一可能的瓶颈是我们使用的关系型数据库，因为其他存储内容的数据库位

于外部服务器上，所以在内容存储方面可以线性扩展。如果我们遇到与关系型数据库相关的性能问题，我们可以很容易的创建一个 MySQL 数据库集群。

Ryan：你们在使用 MySQL 是吗？

Alex：是的，我们创建了几个只读访问的实例和一个可写的实例。

Ryan：当数据量变得太大，你遇到过拷贝问题吗？比如从 master 拷贝到 slaves？

Alex：目前我还没有注意到。是会有一点延迟，但不明显。通常我们采用逻辑划分数据。而不是物理划分。这样我们不需要针对每一个请求都访问数据库。我们能够在真正需要处理一个请求的时候缓存大量信息。访问数据库的通常都是跟踪信息或者处理广告。即使在集群上发布数据的时候存在一些延迟，也影响不到前端的性能。

Ryan：你们使用了多少缓存？在何处缓存数据，只有一个吗？使用分布式缓存吗？

Alex：我们使用本地缓存，单节点，对象缓存。

Ryan：那么是在 Hibernate 之上还是之下？

Alex：在 Hibernate 之上。事实上，如果你说我们存在两个缓存也是正确的，因为我们使用了 Hibernate 缓存，但是我们把 Hibernate 对象混合到了我们的对象中，因为它们太复杂了。我们采用合理的缓存并通过自己的 API 访问这些定制的对象。

Ryan：最近视频流系统重新做了设计。你能详细介绍一下吗，比如新的架构是什么样子的？

Alex：最初我们使用了基于流的解决方案并由第三方实现。不幸的是，在方案设计完并开始动工之后不久，我们就发现第三方提供的服务要求我们和客户开放特定端口来访问 Flash 流。这对我们的大客户来说是一个很大的问题，例如像 IBM 这样的大公司，完全处在防火墙后面，他们绝不会为你打开特定的端口，而只是为了收看 InfoQ 上的视频，哪怕这些视频很有价值。因此，我们开始考虑替代方案。

那时，我们注意到 YouTube 和其他视频服务提供商正在迁移到基于下载的视频方案上。与此同时，Amazon 启动了目前很有名的服务，如 S3 和 EC2。我们考虑使用这些开放服务（希

望它们真的可靠）建立一个解决方案，新的架构就是基于 Amazon S3 和 EC2 服务。部署非常简单—你只需要一个 web 服务器让你能够访问被索引的视频，和一些存储，仅此而已。如果你开始考虑这样一个解决方案，你可能几天之内就能创建。现在就是这么简单。确信 Amazon 服务可靠对我们非常重要，它们为 S3 服务提供的 SLA 让我们决定采用 S3。现在我们正在等待 EC2 的相同服务。

Ryan：当你获得视频的时候：InfoQ 不做其他工作吗，所有的视频都是适合 Flash 播放的编码格式吗？有时你是否需要使用第三方或者内部、外部的编码转换机制？

Alex：简单地说，这个视频处理是一个工作流。首先是获取原始视频，交给视频编辑专家来索引和创建元数据，然后我们拥有一个或者说我们正试图拥有一个更加自动化的管理工作流的方法。所以，就你的问题而言，所有的一切都是在公司内部完成的。目前不是全自动化的，我们会在几个月之后争取实现，以方便编辑的工作，这些小步骤现在都是手工的，但它是一个内部流程。

Ryan：你提到你把视频存储到 Amazon 服务上。你得到的是一个放了一些数据的容器，不管它多大、是什么，你只是把数据放进去，他们负责传递。有没有一个 URL 可以提供给客户或者用户，在他的浏览器上使用？从内部键值到 URL 的映射关系存在何处？你如何知道你把视频存在哪里了？

Alex：我们有 S3 存储还有 EC2 服务器。为了能够提供视频服务，我们需要从 S3 上获取视频。因此，我们在 S3 容器和本地存储之间建立了一个同步机制，然后一切都通过此处访问。现在，解释一下如何获取资源。我们的内容数据库会提供资源的名字，因此所有我们存储在 JCR 中的元数据和与内容相关的信息都存在该数据库中。然后，我们提供一个 ID，数据库里给出获取该视频的映射关系。即使是 S3 或者 VitalStream 第三方支持，都是一样的。说到底，就是基于 ID 的资源查找。

Ryan：你刚才说把 Hibernate 对象映射到其它对象上，为什么要这样做？

Alex：抱歉让你误会了。我刚才想说的是，我们的模型要比只从 Hibernate 得到的更丰富。因此我们把不同的对象组合到一起建立一个代表一个页面或者类似事物的对象。这是一

个聚合过程，而不是从模型到 DTO 的迁移。

Ryan：你是否使用了 Hibernate 提供的关联机制？例如，我创建了一个用户。一个用户可以有多种角色（你可以配置 Hibernate 来获取用户和全部角色）。Hibernate 提供了这种功能。你说在更高一层作了聚合，这是否意味着你只能在更高的层次上获取单实体或者实体集合？

Alex：我提到过我们采用了不同的存储。我需要从所有存储中获取数据并组合成一个页面。我们使用了 Hibernate 的全部特性，比如延迟取、快速取、联合取等一切特性。

Ryan：所以聚合意味着你不得不组合来自不同存储的数据。

Alex：完全正确。如果你看一看网页，你会努力把它描述成一个模型，页面有内容组成、广告元素、图片和其他类似数据—所有这些代表了我们模型的一部分。为了表示整个页面，我们需要聚合所有这些小部件，比如广告元素、内容，聚合的方式很有趣，因为首先使用内容，然后在与内容相关的元数据的基础上，我们努力推断出适合发布何种广告。简单的说，我们有一个核心模型、带有元数据的内容，然后利用其他的数据来修饰这个核心模型。

Ryan：InfoQ 未来有什么规划吗？如何进行开发的？是围绕一个需求清单吗？

Alex：考虑到我们公司非常虚拟化，我是说全球的工作人员，分布在不同的地点和时区，我们围绕着需求清单建立了一个定制过程，清单上按照优先级列举了未来几年内我们需要实现的事情，然后推动几个迭代过程，我们会讨论细节。针对你的问题，我的需求清单有七页之长。这些新功能迟早都会实现。我们还有一些新的想法没有写在清单上，但是我想给大家一个惊喜，我们现在有很多竞争者，所以我们将保守秘密。上一次，视频系统的重新实现，我们做了初稿并邀请用户浏览和评论，给我们反馈，以后主要的功能我们都会采用相同的流程。如果你在 InfoQ 上注册，就有机会帮助我们在未来实现新特性。欢迎注册。

Ryan：你如何实现网站搜索？采用了哪些技术？

Alex：我在采访开始的时候曾提到过 JCR API 提供了全文索引。因此，我们具备这项功能。但是目前我们使用 Google 搜索，因为我们发现这样性能会稍微好一点，运行的也非常好。我们正在考虑将来把这两项技术结合在一起提供高级搜索，能够使用特定的查询语言来

搜索网站，你知道，我们对内容加了标签等，正好可以支持这种搜索。

观看完整视频：http://www.infoq.com/cn/interviews/popescu-infoq-architecture-cn

相关内容：

- 可伸缩性原则
- InfoQ 案例研究：纳斯达克市场回放
- 构建的可伸缩性和达到的性能：一个虚拟座谈会
- 跨企业业务应用的架构
- 在你的企业中使用开源软件：神话与澄清

[热点新闻]

静态分析工具综述：Roodi、Rufus、Reek 和 Flay

作者 Werner Schuster 译者 杨晨

静态分析工具能够保证代码的质量，发现并警告潜在的 bug。静态编译语言的编译器经常运行静态分析检查，然后以警告的形式报告潜在的问题。流行的独立分析工具有 C 的 lint 和 Smalltalk 的 Lint 等等，许多现代的 IDE 同样也能够对代码进行静态分析，还能够随着代码的编辑进行增量的检查。

在很长的时间里，由于没有访问 Ruby 资源中抽象语法树（AST）的标准方法，Ruby 在静态分析工具方面总是不能得心应手。解决方案之一是使用 ParseTree 这个 gem，这个工具使用了原生扩展，来实现对 Ruby 代码解析树的访问。ParseTree 也只是在 Ruby 1.8 中可用，不过似乎 1.9 不会继续支持它（Ruby 1.9 将会支持 Ripper，这个库支持对源文件进行解析，但是不支持实时访问解析树）。ParseTree 现在并不能很好支持那些新的 Ruby 实现。

再来介绍一下 ruby_parser，这是一个 Ruby 的解析器，它是用 Ruby 编写的，并且承诺改进这些问题。这个项目最近发布了 2.0 版本，不但改善了性能，而且将行号作为元数据加入到 AST 中。后者对于静态分析工具是非常重要的，因为这些工具需要报告发现问题的位置。

有一点至关重要，那就是所有现存的 Ruby IDE 都是使用 Java（例如 Aptana 或 3rdRail 等基于 Eclipse 的 IDE、Netbeans 的 Ruby 支持，或者 JetBrains 的 RubyMine） 或者.NET（基于 Visual Studio 的 Ruby In Steel）编写而成的。所有这些 IDE 都包含对 Ruby 代码的静态分析代码，但是它们都不是 Ruby 编写的。基于 Java 或者.NET 语言，并采用 Ruby 解析器和 AST 的静态分析代码，显然不能够支持 MRI 或者其他的 Ruby 实现方式。UnifiedRuby 派生自

ParseTree，对 ParseTree 的输出进行整理加工，还与 ruby_parser 相结合，它现在可以解析 Ruby 的源代码，并且能够通过纯 Ruby 来进行分析。

在过去几个月里，发布了一系列的静态分析工具。

Flay，这是由 Ryan Davis 编写的工具，能够检查重复的 codebase。这个工具使用了 AST 而不是直接分析源代码，从而能够结构化地比较代码。拷贝或者粘贴的代码即使经过了些许修改，也能够被检测到。Ryan 之前曾经发布过另外一个静态分析工具 flog，这个工具主要根据其内置的各种不良代码匹配模式计算 codebase 的得分，例如过多的依赖等等。Flay 和 Flog 都能够使用命令行检查 codebase。Flay 使用 ruby_parser 对 Ruby 代码进行解析。

Reek 由 Kevin Rutherford 编写，是一个“ruby 代码的怪味道嗅探器”。它能够检查非常长的方法体、臃肿的类、错误的名称等等。这些检查是通过继承自 SexpProcessor，并且访问 AST 来实现的。Reek 的代码可以在 Github 上下载。

Roodi 是一个与 reek 非常相似的工具，它能够对 codebase 进行一系列的检查。Roodi 检查方法或模块是否符合命名规则，或者最大参数数目等是否一致等等。其他的检查项目包括提供诸如避免 for 循环之类的建议等等。在 YAML 文件中包含了其他附加功能的配置，而且配置起来非常方便。同样地，编写新的检查类也非常容易。检查器的类是通过注册 AST 的节点，然后监控这个节点的子树来实现检查功能的。

Rufus，这是一个由 John Mettraux 编写的工具，这个工具能够检查 Ruby 中不需要或者不安全的代码。Rufus 的库能够在加载某些 Ruby 源代码之前检查它们。例如，加载一个只有一行（例如 exit）代码的 Ruby 文件可不是什么好主意。这个库是可配置的，能够自定义匹配模式，决定哪些代码将要被排除掉。

你打算把这些工具添加到持续集成的配置中吗？你希望对代码进行什么检查，或者打算自己编写哪些检查功能呢？

原文链接：http://www.infoq.com/cn/news/2008/11/static-analysis-tool-roundup

相关内容：

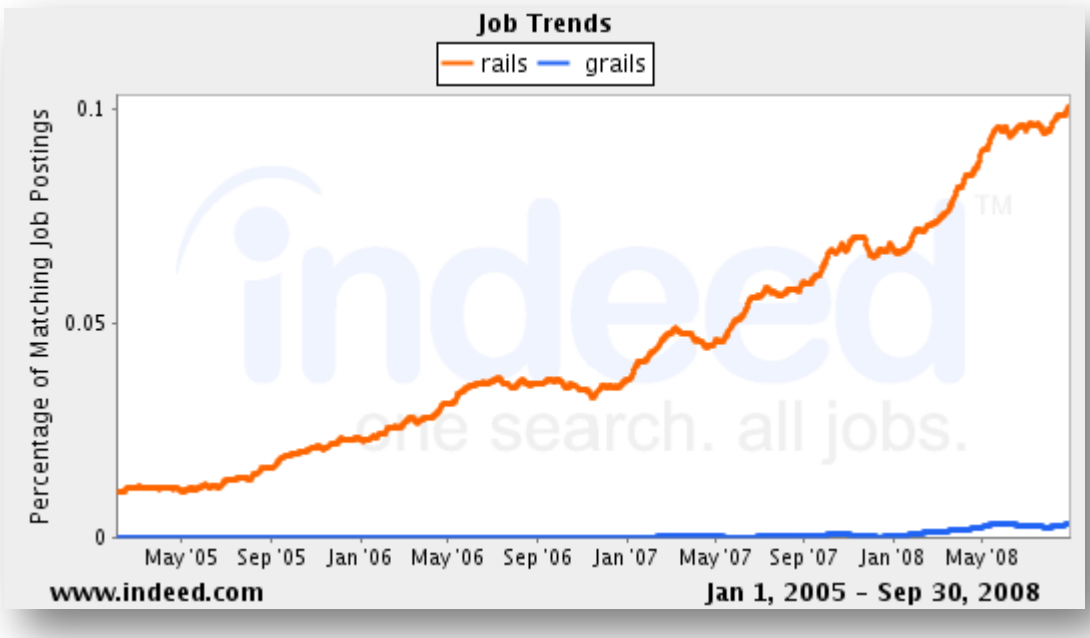
- ParseTree 3.0 发布，众多相关程序库升级
- 洞察动态语言与静态语言之争
- 试着发挥静态类型语言的最大功效
- 静态分析工具可以突出更深层次的问题
- 评论：Exception Hunter

[热点新闻]

如何在Rails和Grails之间做选择？

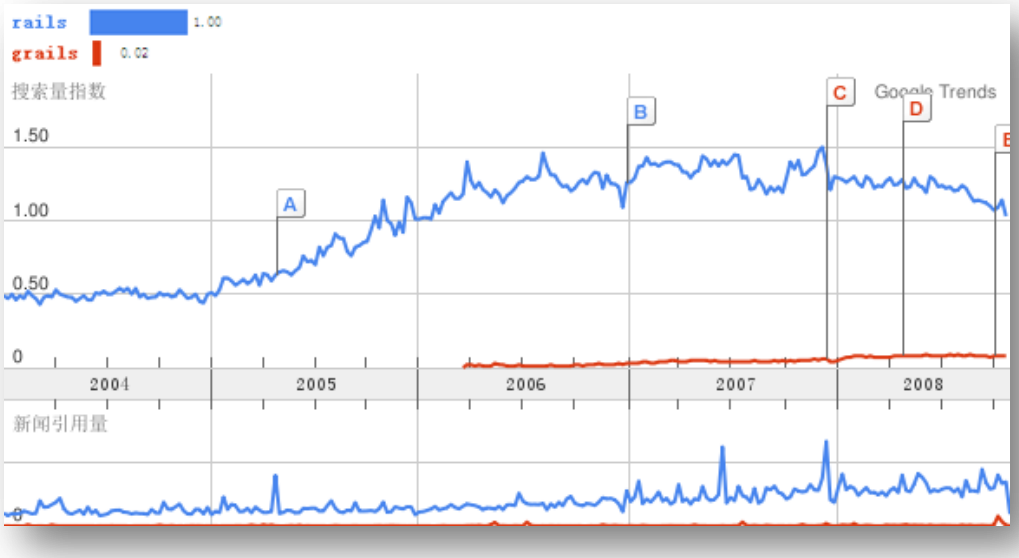
作者 宋玮

自从 Rails 和 Grails 进入人们的视野以来，有关 Rails 和 Grails 之间的各种比较就没有停止过。 最近 Stephan 在其博客上给出了 Rails 和 Grails 的工作趋势图。从 Rails 和 Grails 工作趋势图中可以看出，Rails 正处于快速上升期，Grails 上升的趋势相对较缓。



但是他并没有对此图作出过多评论，正如他所说："如果你想对这一趋势加以评论，我看还是免了吧。已经有很多相关讨论了，每个人都各持己见"。

从另一个张图——Google 趋势图，我们可以看看人们对 Rails 和 Grails 这两个主题的关注度：



图中我们可以看到，Rails 从 2005 年开始就进入“快速上升期”，今年势头开始放慢。而 Grails 从 2006 年以来一直处于相对缓慢的增长过程。

从这两张图中我们可以看出，Rails 还是相对比较火爆的，但 Grails 也在逐步增长。从技术角度来看，这两者又是孰优孰劣呢？我们又该如何选择呢？早些时候 Matt Raible 在其博文《Grails vs Rails——我的想法》中表达了他的看法。

关于成熟度问题，Matt Raible 认为：

> 它们都是优秀的开发架构。Rails 更加成熟一些，但是创建环境是相当痛苦的（特别是在 Windows 上）。对于 Java 开发人 员来说，Grails 非常容易创建起来。Grails 需要提高的地方是热发布和出错记录堆栈，但这些大概是 Groovy 语言的问题，出错记录堆栈是惨不忍 睹的—很少在最初的几行指出类和行数。

关于如何在 Rails 和 Grails 之间进行选择的问题，Matt Raible 说道：

> ……两者都有编程的乐趣，并且有能力大大提高开发效率。如果你熟悉 Hibernate、Spring、SiteMesh 和 JSP，那么你应该学习 Grails。如果你精通这些技术，那么你一个小时之内就能学会 Grails。……
>
> 有没有 Rails 能做而 Grails 不能做的事情？这就不是我能够告诉你的。我想这取决于开发人员的热情和开发团队的选择。如果你是资深的 Java 开发人 员并且喜

> 欢这个生态和它的工具，那么选择 Grails 就更直观一些。如果你是资深的 PHP 开发人员或者在 J2EE 上感觉不好，那么你可能更喜欢 Rails 一些。对于两个开发框架来说，它们都有一个相同的事情——学习一个实际上会教给你另一个的知识。它们在很多方面都如此相似，以至它们之间的知识可以相互转 移。

冯国平（hivon）已经在其博客中将《Grails vs Rails——我的想法》一文翻译成了中文，有兴趣的读者可以点击这里查看完整的译文。

原文链接：http://www.infoq.com/cn/news/2008/11/select-between-rails-grails

相关内容：

- 用 Acegi Security 来保护 Grails 应用
- 使用事件模型定制 Grails 应用的行为
- 迷你书下载：Grails 入门指南
- 用 Groovy 创建领域特定语言
- 辩论：Maven 是正确的构建工具吗？

[热点新闻]

事件流处理：数据仓库的可伸缩替代品

作者 Sadek Drobi 译者 郭晓刚

Dan Pritchett 在博客上提出了一种数据仓库应用的替代方案。 虽然厌恶“只能单一位置及单一存储空间上实现的方案”，他也承认有时候必须先聚合数据才能作分析。他所说的正是数据仓库应用的功能——沿着某些变量轴聚合 信息并转化数据间的关系。而在 Pritchett 看来，数据仓库应用在使用中有许多缺点。数据仓库应用不仅非常昂贵，“比较小的组织一般难以企及”，而且 ETL（Extract, Transform and Load，提取、转换、装载）软件的工作方式意味着要付出可伸缩性和反应能力的代价：

> 首先，ETL 给生产数据库增加了明显的负担。如果你的业务有空窗期可以做 ETL，那是最好的；如果没有，管理可伸缩性就是很大的挑战。第二，数据仓库里的数据新鲜度一般滞后 24 小时或更长，随着业务增长，滞后时间会越来越长。

Dan Pritchett 相信有一种方案更便宜，也更可伸缩：用 ESP（Event Stream Processor）处理事件流。

> ESP 用类似 SQL 的语言处理各种事件流。与数据库和数据仓库通过 SQL 分析数据表类似，ESP 用它们的查询语言分析事件流。要想理解 ESP，可以把事件类比成数据库表中的行，而事件的属性则对应数据库表的列。每一种事件类型就等于是一张表。
>
> [...]
>
> [ESP 分析]数据的变化，而且就在变化发生的当时分析。我们不再进行批量的

> ETL，而是把业务事件变成一连串的数据状态变化。这就创造出一种更易于管理的生产系统的伸缩模型。
>
> [...]
>
> ESP 可以做水平伸缩，因此可以达至一种更具成本效益的业务方案。而且由于 ESP 执行分析是实时的，因此得到的业务指标更加应时，并且不受业务增长的影响。

Dan 也特别指出这种方法的弱点，就是不能进行历史性的分析，不能从当前以外的角度去观察业务活动。Pritchett 提出用一种捕捉并重演事务的 框架去克服此弱点，不过该方案相当昂贵。Tahir Akhtar 在帖子的留言中提出另一种弥补方法：用 ESP 替代 ETL，但在享用 ESP 的可伸缩性和反应能力优势的同时，继续使用数据仓库应用以保留历史分 析能力。

原文链接：http://www.infoq.com/cn/news/2008/11/scalable-datamining-alternative

相关内容：

- 关于复杂事件处理和事件驱动架构的争论
- Gartner 论述平台中间件中的分裂趋势
- Esper 近况：事件流处理框架
- 使用 WebLogic 事件服务器构建复杂事件处理应用
- 通过 Esper 探索事件驱动架构

[热点新闻]

Amazon EC2,Google App Engine, Microsoft Azure 大比拼

作者 Abel Avram 译者 黄璜

当 Microsoft 在 PDC2008 上携 Azure 平台驾云而来时，天气预测注定将发生改变。将当前市场上分别来自 Amazon、Google 和 Microsoft 的三大主流产品作一比较会是一件非常有意思的事儿。第一眼看上去的印象是它们彼此之间似乎实际上并不存在真正的相互竞争。

Ziff 兄弟投资的副总裁 Michael J. Miller 对云计算的三大角逐者进行了比较，这是他关于 Amazon EC2 的发现：

> 备受关注的云平台无疑就是 Amazon Web 服务了，它是多种工具 的集合，大部分都位于相当底层。其中又以 Amazon 弹性计算云(EC2)最为出众，这一 Web 服务能你将应用分配给任意多的“计算单元”。简单的说，一 个标准的单一实例包括了一个“虚拟核心” ,1.7GB 的内存 ,以及 160GB 的存储实例(只针对该对话的存储), 价格为每小时 10 美分。在这个服务之上， 你可能会考虑使用该公司提供的“简单存储服务”(S3) ,对于前 50TB 的数据 ,其价格是每 GB 每月 15 美分 ,之后价格递减，同时要收取一定的交易费用。 同时你也可能考虑使用该公司的“Simple DB”数据库，或者是用于存储消息的队列服务，包括一些附加的收费。
>
> Amazon 平台的基本优势很简单：在你需要的时候，仅仅使用你所需要的存储量。

关于 Google 的 App Engine，Michael 如是说：

> Google 的 App Engine 是新来 者。它仍处在免费的 beta 阶段 ,并且其工具集到

> 目前为止仍有一定局限性。具我的理解，如果说 Amazon 给了你一台可以在其上安装许多软件的虚拟机的 话 ,Google 更应该说是给了你一个基于 Python 语言 ,Django 框架，Google 的 BigTalbe 数据库/存储系统和 Google 文件系统 (GFS)的确定环境。目前，开发者可以免费获得 500MB 的存储，以及最多每月 500 万页面访问的计算能力，并且该公司对更活跃的站点宣布了收费策略。 举例来说，该公司表示开发者每 CPU 核心/小时可能会需要支付 10 到 12 美分。
>
> 因为其与 Google 自己的操作环境联系非常紧密，所以对于了解这些框架的开发者而言 ,起步相对会容易一些。但一些开发者却选择了避开它 ,因为相比 Amazon 的解决方案它显得过于局限了。

当谈到 Microsoft 的 Azure 时，Michael 证实：

> 与 Amazon Web 服务相似，Azure 实际上是由一个公共平台上的多种不同服务来组成的。.....NET 服务是目前最受关注的部分，因为开发者初始时将用它来为平台作 开发。实际上，从我所参加的会议看来，将一个为.NET 框架编写并用 Visual Studio 开发的应用将会很容易的迁移到“云”上。
>
> Azure 的一大不同之处是在于 ,尽管 Microsoft 打算提供其自主的 Azure 托管服务 ,但该平台同样也是为运行于本地工作站和企业服务器而设计的。这使得测试应用变得方便 ,也同样得以支持企业应用既能运行于公司的内部网也能运行于外部环境。

Michael 将其比较概括为：

> 考虑这三个角逐者，你会发现它们每个都发挥着自己的强项。Amazon 是市场的先行者，并利用因特网标准与开源平台打造了一个十分灵活的平台。 Google 利用了其对于大型数据库的研究成果并借助其内部的开发方法创建了一个强大但略显局限的环境。而 Microsoft 凭借其在开发者方面的传统强 势与其宽泛的工具集提供了可能是最庞大的一系列服务。随着时间发展 ,我猜测我们会看到它们会开始互相靠拢 - 从 Amazon 引入 Windows 服务实例就是 一个预示。

道琼斯新闻电报的 Jessica Hodgson 就 Microsoft 参与这场游戏所产生的新财务等式撰文一篇。她引用 Directions 研究公司的分析员 Matt Rosoff 的言论：

> 我以为 Microsoft 想要做的是冻结这个市场。他们想让那些打算尝试按需产品的人们停下来，并等着看他们的产品将会如何。

根据 Jessica 的说法，对于云计算市场的价值有多种不同的意见：

> 虽然大家都同意云计算正在增长，关于它是否会取代本地授权软件，或是与其结伴同行的意见却各有千秋。Deutsche 银行的分析师 Tom Ernst 表示，软件包模式的市场已达到饱和。不出五年，Ernst 估计云计算服务将占据价值 600 亿美元的应用软件市场的半壁江山。与此相 反，Oracle 公司的首席执行官(ORCL)Larry Ellison 却嘲笑云计算是"胡言乱语"，并声称没什么公司可从中获益。
>
> Microsoft 对于业务模型的承诺将会助长期待，推动那些不愿拥抱云计算的大公司采纳它。位于温哥华的 Web 开发公司 Strangeloop Networks 的共同创始人 Richard Campbell 表示，他的许多客户都在询问云计算，虽然出于对安全和可靠性的考虑很少真正转向它。

Jessica 接着分析 Microsoft 的行动：

> Microsoft 审慎的前进着，一边小心的保护着其既有的软件包模式特权，一边打量着云计算市场将如何发展。它采纳了一种混合的方式，因为它声称多数消费者将会继续需要本地授权软件的产品，就是大部分分析师所支持的立场……。
>
> 如果 Microsoft 进展太慢，它将面临让诸如 Google 以及 Amazon 等革新者占据市场分额的风险。这些公司不存在 Microsoft 所面对的利益两难境地，因为他们没有什么实体的软件特权需要去维护。
>
> "如果它(云计算)成为了主流业务的流行方式，很难看到他们如何去避免去蚕食自己桌面服务的销售，"Nickolas Carr 表示，他曾任哈佛商业评论的编辑并出版了一本关于云计算的书。"这将是一场 scale 的游戏。"

原文链接：http://www.infoq.com/cn/news/2008/11/Comparing-EC2-App-Engine-Azure

相关内容：

- 亚马逊 EC2 服务：从 beta 版转换到生产环境
- 亚马逊 EC2 云计算计划支持 Windows

- Amazon FPS：可定制的支付服务 & DSL
- 用 Amazon Web Service 实现视频文件转换程序
- 亚马逊为弹性计算集群（EC2）开发机器映象市场

[热点新闻]

James Shore：敏捷的衰落

作者 Chris Sims 译者 李剑

James Shore 声称敏捷正在走向衰落。他说，很多团队在用“sprints”和每日例会，但是却不采用那些可以在长期内产出高质量软件的技术实践。在他的估计中，已有无数个 Scrum 团队将敏捷用的如此之烂，不仅失败已成必然，而且会将敏捷的发展跟他们一起拖入泥潭。

James 的文章中，大部分都是在指责 Scrum 和 Scrum 的误用。他将 Scrum 和 XP 进行对比，指出 Scrum 故意把 XP 中包含的技术实践抛 在一边。在一些技术话题上——例如结对编程、测试驱动开发、持续集成、自动化测试——Scrum 保持了缄默。但是如果没有这些实践，团队很快就会造出一个 庞大且蠢，问题多多难以维护的代码库。然后这就会变成他们身上重重的禁锢，使他们无法像敏捷团队一样快速应对变化。

James 认为，这也不能说是全都是 Scrum 的错，因为团队必须要为自己的成败负责。很多团队都只选用 Scrum 中浅显简单的部分应用，例如短迭 代和每日例会，更困难而且也是更重要的实践——如回顾和改进——就不管不顾了。在这个过程中，团队本应有能力识别并且采用一些工程实践，帮助他们在每个迭 代中交付可用软件，但不幸的是，很多团队都没能做到这一点。

很多人评论说，这个问题不是源于 Scrum 本身，而是那些把 Scrum 用的惨不忍睹的人造成的。例如，Dustin Whitney 说道，“我觉得你因为那些庸人失败了就来指责 Scrum，这相当不公平。”

James 的观点是，无论失败的原因是什么，这些失败都有可能把敏捷变成一种风潮，随风而逝。

> 不幸的是，有很多自称敏捷的项目在走向失败。他们正在失败。最后 Agile 将

> 承受这后果，它会离我们而去，正如一切流行时尚一样。

Simon Kirk 的回应则十分乐观：

> 我赞同作者的这个前提，很多冠以“敏捷”之名所行之事的确名不副实。不过我也相信，这是普及敏捷（我是说真正的做好敏捷）的过程中无可避免的一步。

敏捷是时尚么？它真的难度很大，大多数团队都没法有效实施？或者它只是正在经受成长的烦恼，即将迎来更广泛更加成功的应用？请留下你的看法，与其他读者共享。

查看英文原文：James Shore: The Decline and Fall of Agile

译者注：

在 InfoQ 英文站上，James 也留下了评论：

> 很多人都把我的这篇文章视作对 Scrum 的责难，但这不是我的本意。我只是想着重指出我所见的失败案例，还有导致失败的成因。最大的问题不在于 Scrum 或是 CSM，是那些买椟还珠的人。

Bob 大叔则以诙谐辛辣的笔吻写道：

> 现在我们总算找到答案了。我们知道是谁的问题了。是 SCRUM！SCRUM 是敏捷运动失败的原因。SCRUM 是敏捷团队把事情搞糟的原因。SCRUM 是一切问题和罪恶的根源。SCRUM 带来了“敏捷衰落”。
>
> 你被我玩了。
>
> Scrum 不是问题，它过去从来不曾成为问题，将来也永远不会成为问题。亲爱的工匠们，这个问题是我们自己的懒惰啊。
>
> 既然我们不写测试，不能保证代码的干净，那埋怨 SCRUM 做什么呢？我们不能将技术债归咎于 Scrum。在 Scrum 出现之前，技术债就存在已久了，而且它还将继续存在下去。不，Scrum 不应该被骂。罪魁祸首还是跟从前一样：我们自己。
>
> 当然，两天的认证课程不足以构成一个优秀软件领导的充要条件。而且在参加完 CSM 课程以后得到的证书，除了能够说明你花钱参加了两天的 CSM 课以外，也没 有别的用途。而且在工程实践方面，Scrum 也有很多欠缺。但无论是 Scrum 还

是 CSM ,它们的目的都不是从我们中间培养出工程师 ,或是给我们灌输工匠 守则。那是我们自己该干的活！

有些人还说要是那些 Scrum 团队都用的是 XP，而不是 Scrum，那就不会有那些技术债了。**扯淡**！

让俺说的更明白一些： ASININE, INANE, ABSURDITY. BALONEY. DINGOES KIDNEYS.（**荒谬！扯屁！蠢驴！XX……**）

让俺告诉你们，在这，从现在到以后，不管到什么时候，你永远都有可能把 XP 搞烂。用 TDD 想留下技术债真他妈的容易。没脑子的家伙跟人结对也会把代码搞成荒地。而且，我告诉你，你会在做出**简单设计**以后，**不再维护它。**

你想知道写出优秀软件的秘诀么？你想知道怎样保证代码干净吗？你想要银弹吗？私家汤料？万事万物间那唯一的真相？

好，我现在就给你。你准备好了吗？秘诀就是……

秘诀就是……

干好自己的活。

够了，别再埋怨一切，你自己别那么懒就行了。

原文链接：http://www.infoq.com/cn/news/2008/11/decline-of-agile

相关内容：

- Scrum 实施情况调查之案例分析
- 通过游戏学习敏捷
- “Sprint”一词对过渡敏捷不利？
- 一个敏捷教练中止越轨列车的故事
- 视频：Jeff Sutherland 论什么是真正的 Scrum

[热点新闻]

SEI 报告——将 CMMI 和敏捷合二为一

作者 Manoel Pimentel 译者 Felipe Rodrigues 郑柯

上星期业界颇不宁静。SEI 发布报告——**“CMMI 或 Agile：为何不能彼此相拥！”**，提出了在软件开发项目中如何将 CMMI 和 Agile 的理念及实践相结合的问题。

报告的编写过程中，有 David Anderson 等数位著名敏捷专家的参与。报告中这样讲述编写主要动机：

> 无论是对用户、还是这两种开发范式，或是更广阔的社区来说，更多对话可以让整个环境变得更健康，对大家也更有益。

文中的另一句话，解释了期望达到的结果：

> 我们希望这份报告可以激励 CMMI 和敏捷的提倡者们（理想状况下，包括软件相关行业的每一个人），让他们能够：
>
> 认识到两种范式的价值。
>
> 避免常见的错误认识。
>
> 继续尝试、学习和分享，说明哪一种可以在什么样的上下文中取得成效。

报告中还提出如下其他议题：

- 敏捷方法的起源
- CMMI 的起源
- 缺少准确的信息

- 术语方面的困难
- 两种范式各自的价值
- 使用敏捷要注意的问题
- 使用 CMMI 要注意的问题
- CMMI 和敏捷都没有解决的问题

敏捷社区对此有不同反应，特别是在巴西（可以从 Visão Ágil 和 Scrum-Brazil 看到葡萄牙语的讨论）。很多人认为这纯粹是投机主义的做法，因为他们觉得这两者根本是水火不相容。有人认为这对大家来说是一次好机会，在敏捷和 CMMI 的理解和实践方面，有助于修复过去产生的某些问题。

不过实际上，这还只是试图让**敏捷**和 **CMMI** 融合的第一步。只有通过持续不断的实验和调整，很多问题才有可能得到答案，我们才有可能知道两者能否和谐共存，更重要的是：这么做是否有益于成功开发软件应用。

原文链接：http://www.infoq.com/cn/news/2008/11/report-integrating-cmmi-agile

相关内容：

- 敏捷了，而且还是离岸敏捷，是自找麻烦吗？
- 风险投资家已认识到加班对 Scrum 的损害
- 用例还是用户故事？
- 视频：C++项目的敏捷实践
- 视频：熊节谈敏捷在软件开发中的实践

[热点新闻]

揭示 Visual Studio 2010 发展路线图

作者 Jonathan Allen 译者 霍泰稳

Rico Mariani，Visual Studio 的首席架构师，近期谈到了有关 Visual Studio 2010 长期计划的情况。在我们跟进此事之前，Rico 先来了个预防针：

> 我是首席架构师，但是我还“只是”个首席架构师，目前并没有为该产品的方向最终拍板，甚至也没有和其他的架构相融合。虽然我们提出了长期技术路线图，也只 是表明为了产品的长期发展需要，哪些关键问题应该被解决，然而这些问题通常不能和某个具体发布版本中的功能一一对应。

首先提到的是扩展性。尽管 Visual Studio 的核心是可扩展的，许多人们真正想扩展的高级组件还是很有限。另外，可扩展的功能点大多是基于 COM 架构的。

> 为了满足这些需要，根据相应的标准，我们采用了 MEF（Managed Extensibility Framework，托管扩展框架）和 Visual Studio 2010 中两个主要的扩展域——输入和输出。当然，现在 MEF 已经时过境迁，但是根据我们在 PDC 大会上所演示的内容，你可以了解到我们已经走了很长的一 段路程。在我们新的文本编辑器和新型 C++ 项目系统上，我们都采用了主要的 MEF 技术。

未来，Visual Studio 会更多依赖于 Windows Presentation Foundation（WPF）。但人们对这一方向褒贬不一：

> 听上去好像简单之极，其实有很多的障碍。我来谈一下 VS2010 中我很喜欢的一个地方——使用 WPF。很多人认为，至少是一开始这么认为，我选择依赖于 WPF

是多么抓狂，“你负担的起吗？那个某某场景怎么样？我听说 WPF 在那个场景中表现的很不理想。”对于这些意见相左的情况，我一般是沉默以对：

“你们真的认为在计算机图形领域，GDI（图形设备接口）会是以后 10 年的发展顶点吗？”

他接着说道：

我知道 WPF 目前还有一些问题。我们需要对它们进行修正，但是有比 WPF 更好的方案吗？我们已经实现了一些中型的 WPF 应用（比如 Blend），现在我们 也在推动一个旗舰应用，也许是目前世界上第三大的套件（不是很确定，但是确实很大）。沿着 WPF 大道我们会走下去，而且还要取得成功。对我们自己来说，这 件事情很酷，对 WPF 也是如此，然后其他人就有信心跟进。现在还没有什么其他可替代方案，因为我们不能就那么坐下来，还是用着老的 UI，然后幻想着接下来 的 10 年会奇迹般地出现很炫的界面。其实我们在 WPF 领域的一些朋友和我们一样，也是非常激动的……如果最终成功了，也许会更加兴奋！

纵观本文，一个连贯的主题是关于 VS 2008 和 VS 98 之间的对比：

去年我给我的副总裁做演示时，所采用的场景就是在 VC98 和 VS2008 中进行简单的 MFC 应用构建和调试——不要误会，我认为 VS2008 目前已经取得了很大的进步，它是一款非常棒的产品。但是坦白说，做同一件事情时，VS2008 要比 VC98 耗费更多的内存。

当然，VS2008 的功能要比 VC98 强的多，不过严肃地说，我认为它还有很大的提升空间。要知道，从 C6.0 的时候我就已经参与了，一路走来啊:)

在被问及一些 Visual Studio 64 位的事情时，Rico 微微一笑：

有时候人们告诉我说，我们应该推出 64 位的解决方案，以迎合形势发展的需要。我想这是错误的，我认为我们所需要的是使用更少的内存，而不是更多；我认为在 某些关键的地方我们要使用聪明勤快的算法；我们需要朝这个方向走，而这也是我正在努力推进的。我不想我们在做每一个行为时，看上去都好像有很多内存一样 ——如果这样做，那么方向也许已经错了。但是我们确实需要 64 位版本计划，

不过这儿不再讨论。

原文链接：http://www.infoq.com/cn/news/2008/11/VS2010-Roadmap

相关内容：

- Visual Studio 2010 特性聚焦：分析和调试并行应用程序
- 迷你书免费下载：Visual Studio .NET 使用技巧手册
- 微软正式推出云服务平台——Windows Azure
- 在 Visual Studio 中对 Linux 应用进行远程调试
- Boo Lang Studio 简介

[热点新闻]

C#特性聚焦：

动态类型化对象、Duck 类型和多重分配

作者 Jonathan Allen 译者 朱永光

在我们要深入研究第一个 C#特性之前，有必要知道微软许诺，任何在 C#中有的功能在 VB 中也会具通过某种形式来提供，反之亦然。不过他们没有必要以同样的方式来提供这些功能，语言之间还是希望继续有所区别。

随着动态语言和 DLR 日益增加的重要性，C#也需要能处理动态类型化的对象（Dynamically Typed Objects）。目前，通过对静态类进行反射，虽然能够实现后期调用，但这种方式却需要大量的代码。此外，对 DLR 对象的调用需要一个完全不同的，使用 了 DLR 反射函数的调用方式。

在 C#中，你可以简单地声明对象的静态类型为“dynamic”。就像 VB 的 Option Explicit Off 选项一样，它告诉编译器忽略必要的代码来解析运行时调用的方法绑定。在 IL 层面，被声明为 dynamic 的变量是一个 System.Object 类 型，附加了一个额外标签来标明它使用动态调用语义。

在运行时，所有普通重载解析规则都是基于对象的运行时类型执行的。这意味着，你能够直接地执行多重分配，而不用借助反射或访问者模式。

每个动态语言都具有它们自己的成员查找规则。为了支持这个功能，对象需要实现 IDynamicObject 接口。如果这个接口存在于运行时对象上，那么对象就能处理它自己的成员查找过程。在示范中，Ander 演示了如何在 C#中定义一个动态对象。

当然，这就意味着你可以在 C#中的任何地方使用 duck 类型。

原文链接：http://www.infoq.com/cn/news/2008/11/CSharp-Dynamic

相关内容：

- C#特性聚焦：可选和命名参数、COM 互操作性
- C#特性聚焦：协变和逆变
- 动态 C#实战
- 书评：C# Annotated Standard
- 以 C#观点探索 IronRuby

[热点新闻]

经济困局中的 SOA

作者 黄璜

Gartner 九月份一份题为 2008 SOA 用户调查：采用趋势与特征 的调查表明，计划采纳 SOA 的组织数量首次出现大幅下滑。分析中指出：

> 自 2008 年伊始，计划采纳 SOA 的组织数量首次出现了急剧的下滑。在 2008 年，调查中这一数字的比例降了一半还多，从 2007 年的 53%降到了 25%，同时，不打算采纳 SOA 的组织比例比 2007 年翻了一倍，从 2007 年的 6%上升到了 2008 年的 16%。

对于这一现象产生的原因，分析中提到：

> 总体来说，阻碍组织追随 SOA 的两大主要因素一是缺乏相应的技术和经验，二是没有可行的业务案例。如若业务案例被验证是不可行 的，那就没有理由去执行它。然而，通过与 Gartner 的众多客户沟通所反映出来的问题是，实际上他们对于如何构造一个 SOA 的业务案例困惑重重。就算一 个有效的业务案例存在，自身也不具备所需要的技术 ,而开拓自身技术和获取外部经验所需要的成本和努力常常令人望而却步。

无疑，外部的不利因素更是加剧了这一影响。独立分析师 Joe McKendrick 这样看待经济拐点对 SOA 采纳所带来的冲击：

> 我们将看到两个不同的 SOA 故事，一种能真正给业务带来变化，并将继续进行下去；一种被我叫做“次级(subprime)”SOA，它在组织财政拮据时很快就会窒息。

然而与上面的现象相反，Joe 要指出的是“**不管是不是低谷，这确是投资于架构的绝佳时机。**”

与此遥相呼应的正是 ZapThink 的 Ronald Schmelzer，他就这一系列的问题，在纽约，伦敦和拉斯维加斯开展了多次行业专注的研讨会。并著文一篇专门加以讨论，给出了中肯的建议。

Ronald Schmelzer 首先指出，“何时是投资于企业架构的正确时机?**现在，是的，现在。**”因为：

> 笨拙低效，冗余又难以交互，并且维护成本高企，却对未来的需求无能为力的系统什么时候最让你无法忍受？你没有钱的时候。什么时候你必须对架构做出投资？当你真正体会到切肤之痛而决定致力于短期内能让企业架构有效运转的时候。

关于如何开展 SOA，他给出了两个关键的意见：

停止长年累月的 SOA 项目。把精力集中于迭代的，流程驱动的 SOA 项目。

> 由识别出一个通过面向服务化能够从成本或时间的角度得以提升的单一业务流程开始。别一上来就抱着整个系统不放，别刚开始就去买个 ESB，别动不动就来个长达两年 ,企业范围的组织性的自顶向下服务分析实践。从关注于业务本身做起，更明确一点，从一个业务流程着手。

没有预算？让 SOA 来挽回成本。

> 简单地通过构建一个能被组织内广泛消费，更重要的是，能解决一个关键业务流程中与变更相关的问题的服务，你就立即能获得 SOA 所带来的收益。你如何知道一个问题值得以面向服务的方式处理？一旦你发觉这一业务流程牵涉的任一方面更改都会不断地增加成本或消耗时间。
>
> ...简单地通过改进业务流程你就能为业务挽回成本，同时可将这些资本再次投资于企业架构，达到良性循环。一个成本补偿（cost- recovery）的 SOA 预算如何工作？关键是从你能找到的效率最低的最小的业务流程开始 ,这一低效是由持续的变更（缺乏灵活性）而引起的，然而业务却 不得不被迫持续地投入该低效的业务流程。

我们再次明白了，SOA 是服务于业务的架构风格。正是好的 SOA 实现，才能达到节约成本，优化流程，高效整合，俨然成为抵御寒冬的最佳武器。在援引 的文章里，ZapThink

概括到，当日子紧张的时候，越是该紧迫地重新对业务进行思考。而改进低效的业务流程，为企业挽回成本和带来业务价值，正是 SOA 的机会。你的企业准备好用 SOA 来应对寒流了吗？对于你的企业，哪一个流程才是最需要关注的切入点呢？

原文链接：http://www.infoq.com/cn/news/2008/11/SOA-down-eco

相关内容：

- 用面向服务架构改进医疗系统表现
- 争论：为什么大多数社交软件会失败，又该如何避免
- 对软件架构和企业组织结构的思考
- 揭秘 SOA 成功的主要因素
- 预先设计的大架构——过早考虑伸缩性之一例？

[热点新闻]

一封普通的 SOA 检讨书

作者 Mark Little 译者 黄璜

近来有好几篇文章，主题都是关于 SOA 是否应当被看作是一个失败。Gartner 分析师们也参与了这场争论，写了一封虚拟的信，以项目经理、企业架构师或首席开发工程师的名义，致“CIO、CEO、CFO、CTO 和所有股东”，表明为什么作者承认 SOA 完全是场失败：

> 作为下述情况的结果，我只能得出 SOA 是场失败，对于 SOA 的任何尝试**都会**以失败收场。在我的领导下：

尽管下列失败的原由都是以调侃的口吻来叙述的，但它们却与人们在考虑 SOA 时所识别出的可能的失败原由息息相关：

- 我忘记了将 SOA 项目与我们的业务需求联系起来，因此我不能证明所创建的这成百上千的服务价值何在，
- 我做不到合理的创建和支持一个 SOA 卓越中、指导委员会或是能力中心
- 我没办法将决策层招集进来，让其作为我们 SOA 进展真正的支持者和倡导者
- 我还没真正搞明白我们 SOA 基础设施的需求就草草地购买了 ESB(实际上真的不怪我嘛，供应商说它超级牛逼，无比重要)
- 我从未让我的工程师们尝到过重用成果物的甜头
- 我也没有义务去关心隔壁那堆做 BPM 的家伙在干嘛啊，实际上我们是两个不同的项目嘛
- 我坚信 SOA 就是超酷的 CORBA 或 COM

显而易见的是，为了获取成功，上述的部分或全部都应该被考虑周详并好好实现。

> 尽管我啥也没做，SOA 还是挂了。对于被全世界很多公司都成功证明的最佳实践，我却疏于确认并实现，这又给了我的 SOA 一刀。

正如一条评论所说：

> 我告诉我的客户，SOA 是处于一个关系逆转、分手埋怨的境地。当事情变糟的时候，SOA 会看着你的眼睛，怀着对这段破裂的关系的诚意，轻轻的对你说“真的，别怪我，都是你不好。” 我们有足够的例子来说明现在的 SOA 并不差，但仍有着**太多**拙劣的 SOA。这些真的是非常好的提醒。

尽管如另一条评论所指出，SOA 绝非太上老君的仙丹，也绝不该被当作一样：

> SOA 在某些情况是管用的，而有的时候就不灵了 - 并且，并不仅仅因为是组织或人员的过错。你得面对它，在有些时候它对于你的公司 架构真是一点意义也没有。是的，作为概念来说它非常棒 - 而且，它可能适用于一些口袋，这取决于你的组织是如何组织的，但这并不意味着所有的都可以。

这封信结尾时对这一片儿（相对而言）刚来的新生儿也狠狠给了一下：

> 谢谢你们的理解，我得提前说，对于云计算、虚拟化和 SaaS，我也是绝佳杀手哦～！

那么等着收到“云计算是个恶梦”或者“SaaS 是个谎言”这样的邮件，又会需要多久呢？

原文链接：http://www.infoq.com/cn/news/2008/11/soa-failure

相关内容：

- 观点：REST 在 SOA 中适合哪个位置？
- 移动 SOA 的门柱
- 克服 SOA 实施过程中的障碍
- 如何开始你的 SOA 治理
- 调查显示，SOA 失败？

[热点新闻]

Sun 将培训带入 Second Life 虚拟平台

作者 刘申

今年的“Sun 培训开放日”将于 12 月 18-19 日及 21-22 日相继 在深圳、北京展开。此活动由 Sun 中国培训部门主办，面向企业 CIO、CTO、IT 经理、软件开发商、公家单位、Sun 合作伙伴和 Java 社群 ,及广大对 Sun 有兴趣的社会各界人士及学生 ,讲授包括 Java、Solaris、Web2.0 应用特性、灾难恢复计划、业务连续性及风险管理等课程。今年的培训开 放日活动，除了在课程内容上较之去年进行了相应的调整外 ,还将把培训与 Linden 实验室开发的 3D 虚拟网络平台“Second Life”（第二人生）相结合。

Second Life 是一个基于因特网的虚拟世界，通过由 Linden 实验室开发的一个可下载的客户端程序，用户，在游戏里叫做"居民"，可以通过可运动的虚拟化身互相交互。这套程序还在一个通常的元宇宙的基础上提供了一个高层次的社交网络服务。居民们可以四处逛逛，会碰到其他的居民，社 交，参加个人或集体活动，制造和相互交易虚拟财产和服务。

Second Life 的出现引起了很多“现实”公司的关注，纷纷在虚拟的世界中安营扎寨，这其中不乏 IBM、Sun 这类的 IT 巨头。当被问及 Sun 在 Second Life 和虚拟化培训方面的投入时，Sun 中国区首席教育官张瓒介绍到：

> Sun 正在探索虚拟培训的模式，我们在第二人生中作了很大的投入，现在已经购入了 7 个小岛，把其中的一个岛专门作为培训开放日的 虚拟基地。在岛上你可以看到全部培训开发日的嘉宾、课程以及日程介绍。让参与者可以在岛上自由的与所有参加开放日的朋友进行交流，这将是培训的全新体验。 在那里，大家不仅能

够看到课程以及演讲嘉宾的介绍，还能找到视频，并实时的观看 PPT。

虽然 Second Life 为我们带来了很多传统 E-learning 所不具备的特性，但是它是基于 3D 的图形网络平台，对计算机的配置和网络带宽要求都比较高，如果你的电 脑是老式的或者你的网速很慢，在其中的体验就会被电源波动所损毁，画面转换迟缓甚至有些动作要等很长时间，针对这个问题，张瓒解释到：

> 传统的 E-learning 是平面的，最多是双向交流，存在很多局限，参与者无法感受到“实时”的便捷。而 Second Life 是完全模拟 3D 的世界，是一种全方位的实时体验，参与者就是其中的一个个体，他可以去选择任何感兴趣的主题并实时地与他人交流，不受任何地理位置 的限制，大大的提高了培训的效果以及便捷性。至于机器配置以及网速的问题，这可能是现在为止最大的一个障碍，特别是网络带宽，对很多中小城市而言，如果带 宽不足的话，“第二人生”里的操作会受到很大的影响，但对国内的一线城市来说，这个问题还是可以克服的。

3D 虚拟世界的到来，对传统的 E-learning 方式进行了延伸，HiPiHi 政策与研究部高级经理张安定对此也发表了自己的看法：

> 对于教育来说，3D 虚拟世界把 2D 网络已经拓展的个体经验空间再次延伸。3D 环境并不适合存储文字和传统意义上的知识。全球各地的人，实时的以 3D 化的主体形式存在一个完整的虚拟空间，带来了 3D 虚拟化的感知，认知，心理等多方面伸延的体验。这包括在场感，分享感，沉浸感，参与感，即时合作与创造，面对面交流，虚拟物品与环境创造，可视化和社会化的体验。

Second Life 等虚拟平台的到来，将为我们的教育培训带来哪些变革，让我们拭目以待吧。

原文链接：http://www.infoq.com/cn/news/2008/11/sls2008-secondlife

相关内容：

- Scrum 认证测试
- 通过游戏学习敏捷
- 个人回顾——提升你的“wetware”

- 简析 Sun 在中国的 Java 认证培训策略
- Sun TechDay 看 GlassFish 最新进展

InfoQ中文站

www.infoq.com/cn

我们的**使命**：成为关注软件开发领域变化和创新的专业网站

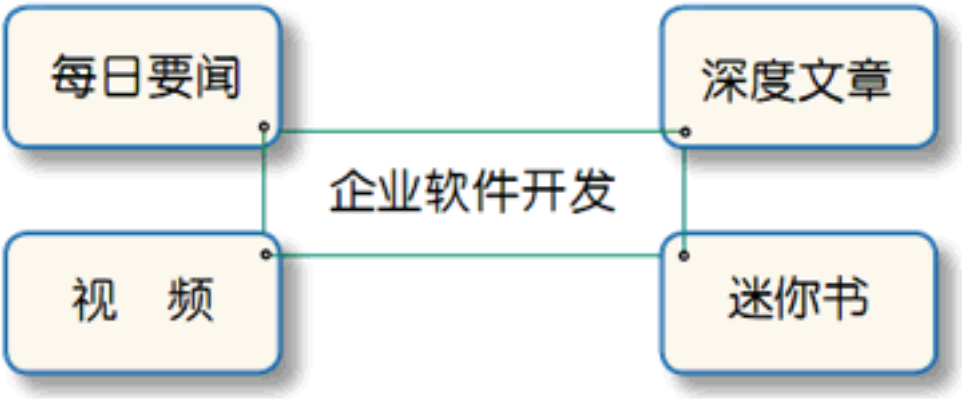
我们的**定位**：关注高级决策人员和大中型企业

我们的**社区**：Java、.NET、Ruby、SOA、Agile、Architecture

我们的**特质**：个性化RSS定制、国际化内容同步更新

我们的**团队**：超过30位领域专家担当社区编辑

……



讨 论 组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
广告合作：sales@cn.infoq.com

[推荐文章]

一种正规的性能调优方法：基于等待的调优

作者 Steven Haines 译者 崔康

推荐理由：*企业 Java 应用的性能调优是一项艰巨的、有时甚至是徒劳的任务，本文尝试把性能调优活动变成一种"科学"范畴内的行为。*

推荐人：郭晓刚，InfoQ 中文站架构社区首席编辑，独立开发者。在经过了 10 年修练之后，总算是懂得一点编程了。目前主要关注以 Spring Framework 和 Hibernate 为主干的 Java Stack 和 Adobe Flex。Microsoft Office 的插件开发也是关心的方向之一。同时也在尽力做一些技术翻译工作，把知识分享给更多的人。

企业 java 应用的性能调优是一项艰巨的、有时甚至是徒劳的任务，这是由现代应用的复杂性和缺少正规的调优方法导致的。现代企业应用与十年前的应用 相比差距很大，如今这些应用支持多输入、多输出、复杂的框架和业务处理引擎。而十年之前，基于 web 的企业应用只是通过网络浏览器获得输入信息 ,然后与数 据库或者遗留系统交互进行后台处理，最后把输出结果返回给浏览器(HTML)。现在 ,输入信息可以来自 HTML 浏览器、富客户端、移动设备或者网络服务， 它可以跨越运行在不同架构下的 servlets 或者门户容器，这反过来又可能调用企业 bean ,外部 web 服务或者把处理委托给业务规则引擎。每一个这样 的组件都可能与内容管理系统、缓存层、众多数据库和遗留系统交互。输出的信息通常以独立于展现层的形式保存，随后转化为 HTML、XML、WML 或者其他 任意客户端需要的格式。现

代应用比过去包含更多移动部分和“黑盒子”，这对性能调优提出了巨大的挑战。

除了复杂性提高，性能调优技术其艺术性要大于科学性，还因为大多数性能调优指南都侧重于性能指标，有时晦涩难懂，也可能影响用户体验。本文尝试把性 能调优活动变成一种“科学”范畴内的行为，提供了一种可重用的关注用户体验的方法，利用“等待点”（也就是应用中引起某请求等待的部分）分析应用架构。总 之，基于等待的调优方法允许性能工程师们通过优化用户体验快速实现可度量的性能提高。

性能调优过程

在详细介绍基于等待调优和等待点分析方法之前 ,本节首先对有效的性能调优过程做一个概述。性能调优可以简单的概括为四步：

1. 负载测试
2. 容器调优
3. 应用调优
4. 迭代

像大多数计算机科学一样，性能调优是一个迭代的过程。首先，创建一个合适的负载测试，其中包含了均衡的、具有代表性的服务请求，这都是容器调优实践 可以满足的。随着容器被不断调优和测试压力的增大，应用程序的瓶颈逐渐显现出来。随着应用的瓶颈被定位和解决，应用行为会发生变化，这就要求容器再次调 优。在容器和应用之间的迭代过程会一直进行到性能到达可以接受的条件（或者直到项目已经到期必须发布时）。

负载测试方法

启动一个性能调优实践的先决条件是创建一整套合适的负载测试集合。每一个负载测试必须满足以下两点：

- 代表性，必须体现最终用户的业务场景（或期望的场景）
- 均衡性，必须符合最终用户不同行为的比例分配

也就是说，负载必须能够按照最终用户的实际操作比例来模拟用户动作。为了说明均衡最终用户动作的重要性，请看下面这个例子：在保险索赔部门，员工执行以下操作：

1. 用户上午八点登陆系统。
2. 上午每人平均处理五个索赔请求。
3. 大约 80%的用户忘记在吃饭之前注销账号，导致 session 过期。
4. 午饭后，用户重新登录系统。
5. 下午每人平均处理五个索赔申请。
6. 下班之前生成两个报告。
7. 80%的用户回家前注销账号。

这个例子是一个真实应用的简化版，但是它足够在这些服务请求建立一个平衡。这个场景展现的均衡是：两次登陆，十次索赔处理，两次报告和一次注销。

如果负载生成器把压力均匀分布在不同的服务请求上又会怎么样呢？在本例中，用户登陆和注销功能会接收与处理与理赔请求相同的负载。如果是 1000 个 并发用户，登陆功能会很快崩溃，导致企业投资建立一个能够处理这种负载的登陆组件，而实际上这种负载根本不会发生。更糟糕的是，本例中由于最大的瓶颈看似 存在于登陆功能上，所以调优的努力会侧重该功能，而忽视索赔处理功能。总之，一个非均衡的负载可能导致调优过程错误的关注于支持那些绝不会发生的负载的组 件，而不是那些真正需要调优的部分！

判断一个负载对于应用是均衡的和代表性的标准，对于测试一个已存在的应用（或者一个新版本）还是一个全新的应用是不同的。

已存在应用

已存在应用跟一个全新应用相比，一个明显的优点是：真实的用户行为可以在实际生产环境中观察获得。根据请求的本质和它们如何被应用定义，可以通过两个选择定义最终用户行为：

- 访问日志
- 最终用户体验监视器

对于大多数基于 web 的应用来说，访问日志提供了足够的资源分析服务请求的本质和它们的均衡关系。Web 服务器可以配置成抓取最终用户请求信息并存 储在一 个日志文件中，称之为“访问日志”（因为该文件通常命名为“access.log”）。使用访问日志定位用户行为的关键是应用交互需要能够通过不同的 URI 来区分。例如，如果之前例子的动作采用类似“/login.do”、“/processClaim.do”、“/logout.do”的 URI，那 么我们可以非常容易的在访问日志中发现这些行为并确定它们的比例。更进一步，通过最频繁的 URI 来排序访问日志可以快速发现占比例前 n%的的若干请求，这 个“n”%应该在 80%左右。

访问日志是文本文件，可以手动检查（不是一个很有效率的任务），可以通过编程解析，也可以通过工具来分析。对此有很多商业解决方案，不过 Quest Software 有一个产品 Funnel Web Analyzer，虽然多年以前已经终止开发，但是由于其很受欢迎的命令，公司就作为将其作为自由软件重新发布。Funnel Web Analyzer 可以分析大多数访问日志并显示用于创建合适负载测试的信息。

一些应用不像上面提到的那样简单，其用户交互无法很容易的通过一个 URI 来定位。例如，考虑一个包含前端控制器 servlet 的应用，该 servlet 接受一个 XML 有效信息——并且其业务处理逻辑就存在于该信息中。在本例中，需要另外的工具来侦测其有效信息以判断其符合哪个业务场景。这 可以通过使用 servlet 过滤器或者一个称为最终用户体验监视器的硬件设备来完成。

不管用户行为是如何获得的，它都是开始任何性能调优实践之前的关键先决条件。

全新应用

由于全新的应用没有任何最终用户行为可以分析，所以对我们提出了一个独特的挑战。定位新应用的用户行为需要三个步骤，如图 1 所示。

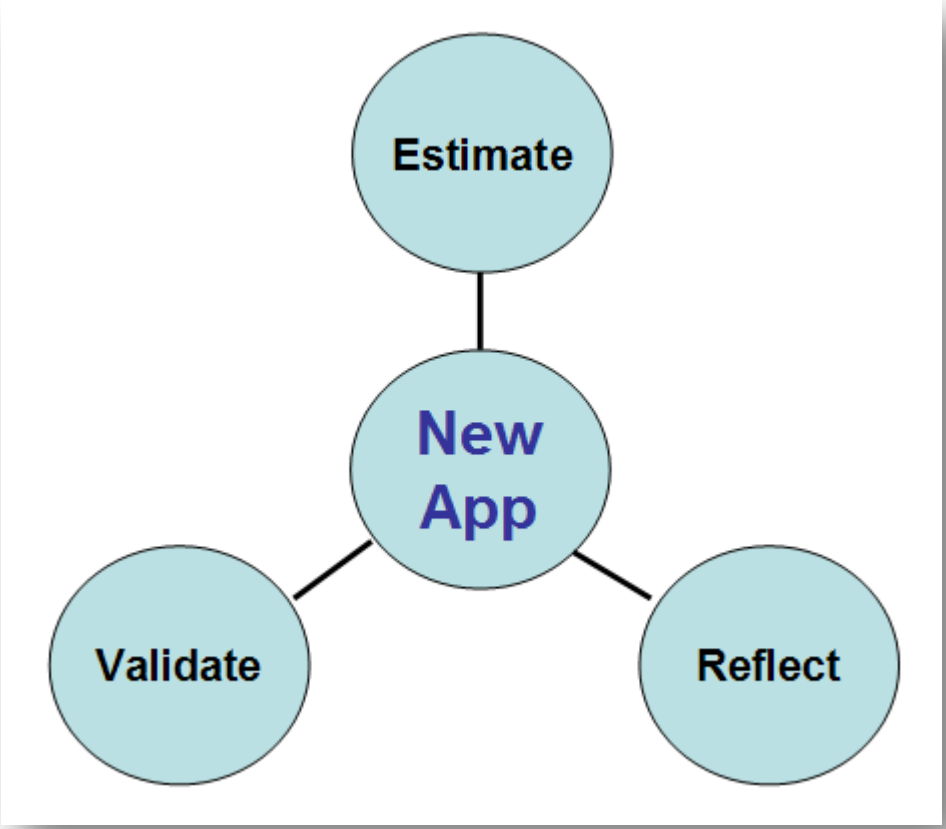


图 1 评估新应用的最终用户行为

第一步，评估最终用户应该会做什么。这步是“猜一猜”的正式说法，但应该是一个经过培训的猜测过程。评估结果应该来自于以下双方的讨论：应用业务负 责人和应用技术负责人。应用业务负责人 ,通常是一个产品经理 ,负责细化最终用户应该如何使用该应用程序——例如，他可能报告说最终用户会登陆、处理五个索 赔请求、过期、处理五个索赔请求、生成两个报告、然后注销。应用技术负责人，一般是架构师或是技术 lead，负责把业务交互的抽象列表翻译成用于生成负载 测试的技术步骤——例如 ,他可能报告说登陆通过“/login.do” URI 完成而处理一个索赔请求需要五个 URI。这些人（或者小组或者一些大型项目里的委员会）应该一起提供足够的信息来建立一个基准负载测试。

我们建立负载测试，用其调整应用和容器，应用程序部署到生产环境中，这一切做完之后，调优工作并没有结束。下一步是验证负载测试集。这通常是一个多阶段的活动：

- 冒烟测试验证：在实际运行的一两周之内验证原先的评估值是否符合真正生产环境下最终用户的行为。这步验证是为了确认在评估过程中没有明显的错误。
- 生产验证：一些应用需要用户花时间才能形成统一的使用方式。这个适应的时间长短因应用而异，可能是一个月或者一个季度，不过一旦用户的行为稳定下来，就需要验证最终用户行为是否与评估一致。
- 回归验证：最好在应用的生产周期中阶段性的验证用户行为，以防止用户行为改变或者引入新的功能或工作流导致用户行为改变。

最后一步，也是经常被忽视的一步，就是反思。根据实际用户行为来反思评估的精确性是很重要的，因为只有通过反思，用户行为才能更便于理解，评估在以后的应用中才能得到提高。没有反思，相同的错误会一犯再犯，最终会增加调优的工作量。

基于等待的调优方法

建好了负载测试，接下来就是决定把调优精力放在何处。大多数调优指南都会提到“性能比率”或者度量之间的关系。例如，某调优指南可能强调说缓存命中 率应该达到 80%或者更高，因此负载测试应用时调整缓存大小直到命中率达到 80%。然后处理列表上的下一个度量值，但是不要忘记验证调整新的参数不会影响 之前已经调好的其他参数。

这项工作不仅困难而且效率很低。例如，调整缓存命中率到 80%可能是件好事，但是存在一些更重要的问题：

- 该应用如何依赖于缓存（与该缓存交互的请求比例是多少）？
- 这些请求对应用中的其他请求有多大影响力？
- 被缓存的条目的本质是什么？它们真的需要缓存吗？

基于等待的调优方法提出了一个新的思想 ,即分析应用的业务交互和实现这些交互的底层结构，然后优化这些业务交互的处理。第一步是分析应用的架构以定 位实现业务请求的相关技术。每一个技术代表一个“等待点”，或者说在应用的这个地方，请求可能需要等待一些事情才能继续处理。例如，如果一个请求执行了一 次数据库查询，则它必须从连接池中

获取一个数据库连接—如果连接池里没有可用的连接，则该请求必须等待直到有连接可用。同样，如果某请求调用了一个 web 服务，而那个 web 服务维护了一个请求队列（对应一个线程池处理请求），这会潜在的导致请求等待直到一个处理线程可用。

从以上称之为等待点分析的方法中，可以定位两种类型的等待点：

- 基于层次的等待点
- 基于技术的等待点

本节首先概述了基于等待点的架构分析方法，然后分别研究了不同类型的等待点。

等待点架构分析

从上面讨论中得出的最重要的结论就是性能调优必须在应用架构的环境中执行。这也是调优性能比率为何如此低效的一个原因：主观的调整一个性能参数到一个最佳值，对应用可能是好事也可能是坏事——因为这可能会也可能不会有利于最终用户体验。

基于等待点的分析是一个分析应用中主要请求处理路径的过程，借此定位潜在影响该请求可能会等待的资源。在等待点分析实践中最有效的办法是定位并标出应用中核心处理路径。这包括一个请求可能访问的所有层次、请求交互的所有外部服务、被做成池的所有对象和全部缓存对象。

基于层次的等待点

一个请求穿过一个物理层，比如在 web 层和业务层之间切换，或者调用外部服务器，比如 web 服务，每当这种时候，都意味着伴随着转换，这里存在一个 隐式等待点。如果服务器在某一时刻只处理单个请求是没有效率的，因此他们使用了多线程方法。典型的，一个服务器在某个 socket 端口监听访问的请求—— 每当收到一个请求它就会把请求放在队列中，然后返回监听其他请求。请求队列对应着一个线程池，负责从队列中提取请求并处理。图 2 描述了对于三层结构 （web 服务器、动态 web 层和业务层）的执行过程。

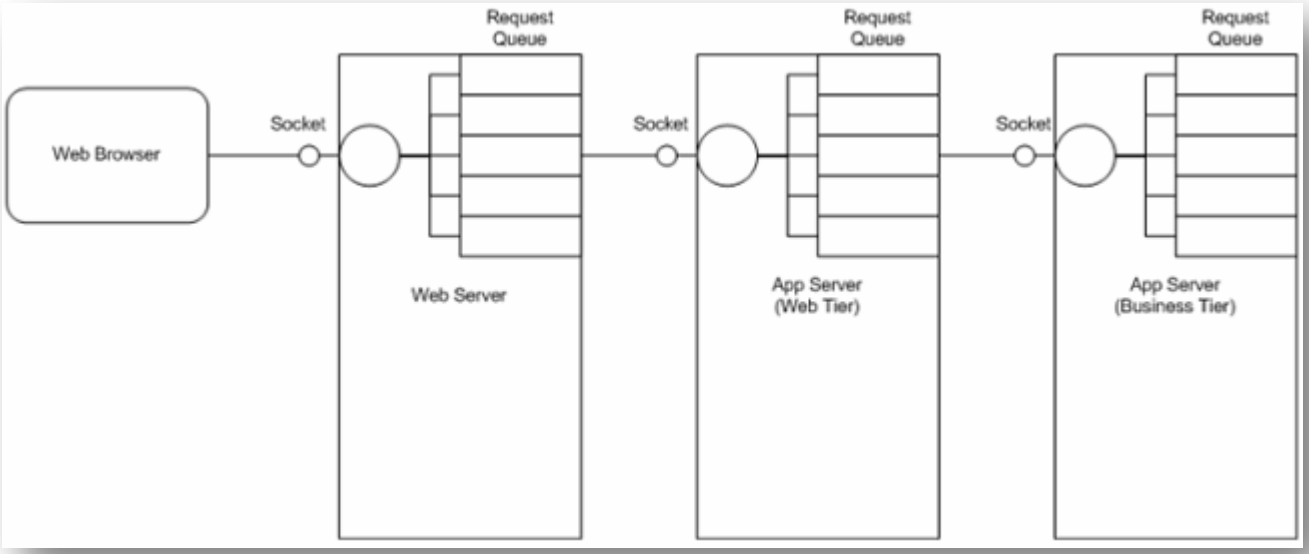


图 2 基于层次的等待点

因为请求穿越层次的动作会引起请求队列，由相应的线程池处理，这种线程池也是一个潜在的等待点。每一个线程池的大小的调优必须基于以下考虑：

- 池必须足够大使得访问的请求无需不必要的等待一个线程。
- 池不能大到耗尽服务器。过多的线程会迫使服务器花费大量时间用于在不同的线程上下文中切换，真正处理请求的时间反而更少了。这种情况通常表现为 CPU 使用率很高，但是处理请求的吞吐量却降低了。
- 池的大小不能透支与之交互的后台资源。例如，如果数据库对于单个服务器只支持 50 个请求，那么服务器不应该向数据库发送超过 50 个数量的请求。

服务器线程池的最佳尺寸的标准是 :对受限制的依赖资源产生足够的负载—最大化它们的使用率而不让其透支。下面的“后退调优”一节有更多调整受限制资源池大小的内容。

基于技术的等待点

基于层次的等待点考虑的是在不同服务器之间传递请求 ,而基于技术的等待点关注的则是在单个服务器中如何通过有效地内部工作来传递请求。基于层次的调优，类似于 IBM 的

队列调优， 只是调整应用的有效第一步，如果忽略了调优应用服务器的内部工作，则会对应用性能产生巨大的影响。这就类似于调整 JDBC 连接池以发送最佳数量的负载给数 据库，但是忽略了检查执行的 SQL 语句——如果查询需要连接十个表单，每个表单有一百万个记录，则最佳负载可能是两个连接，但是如果我们优化了查询，则数 据库可能支持二百个连接。深入研究应用服务器和应用使用的潜在技术，可能存在以下通用的基于技术的等待点：

- 池对象（比如无状态 session bean 或者其他应用放入池的业务对象）
- 缓存设施
- 持久化存储或外部依赖池
- 通讯基础设施
- 垃圾收集

大多数情况下，无状态 session bean 池的大小被应用服务器优化，不会是一个明显的等待点，除非池大小被手工错误的配置了。但是也存在一些池对象必须手动配置大小——这些可能成为有效 的等待点。当一个应用需要一个池化的资源，它必须从池里获取一个资源实例，使用它，然后释放给池。如果池太小，所有的对象实例都在被使用，则请求不得不等 待一个实例可用。显而易见，等待一个池化的资源会增加响应时间，如果越来越多的请求被堵塞在等待池化资源，会导致明显的性能下降。另一方面，如果池很大， 它可能消耗过多内存，对总体 JVM 的性能产生差的影响。池的最佳大小需要权衡，只能在对池的利用情况做彻底的分析才能决定。

池化对象是无状态的，这意味着应用从池中得到哪个实例都无所谓——任何实例都行。另一方面，缓存的对象都是有状态的，也就是说当应用从缓存中请求一 个对象时，它的目标是一个特定对象。举一个很粗糙的类比说明一下区别 :考虑人们生活当中两种常见的活动：超市购物和接孩子放学。在超市中，任何收银员都可 以接待每一位顾客，无论顾客选择哪位收银员都可以顺利结账。因此收银员可以池化。但是在接孩子放学时，每一个父母只想要他们自己的孩子，别的孩子是不行 的。因此孩子可以被缓存。

如前面所述，缓存提出了一个新的调优挑战。简单说，缓存的目的是在本地内存中存储对象，使应用可以随时读取它们，而不是在需要的时候才获取他们。一 个合适大小的缓存可以对通过远程调用加载对象的行为提供明显的性能改善。但是，一个不合适大小的缓存，可能产生明显的性能阻碍。因为缓存维护有状态的对 象，所以重要的一点是在缓存中存储最频繁访问的对象，同时保留额外的空间来处理非频繁访问对象。试想如果缓存太小，请求会怎样：

1. 请求检查缓存中是否存在某对象，结果失败。
2. 请求需要查询外部资源获取对象数据。
3. 因为缓存通常维护最频繁访问的数据，所以这个新对象需要添加到缓存中（它正在被访问）。
4. 但是如果缓存满了，必须利用“最少最近访问”算法选择一个对象移除。
5. 如果缓存对象的状态与外部资源不一致，则缓存对象移除之前必须更新外部资源。
6. 新的对象此刻添加到缓存中。
7. 新的对象最终返回给请求。

这是一个低效的过程，如果大多数请求都要执行这些步骤，那么缓存肯定会降低性能。缓存必须调整到足够大以最小化缓存的“不命中率”，一次不命中意味 着需要执行前面提到的七个步骤，但是也不能太大导致占用太多 JVM 内存。如果缓存需要非常非常大才能满足性能需要，那么最好是重新考虑被缓存对象的实质和 它们到底是否值得缓存。

类似对象池，外部资源池，比如数据库连接池，也必须足够大以满足请求不会被迫等待池中的一个连接变为可用状态，但是也不能太大，导致应用浪费外部资源。“后退调优”一节讨论了如何决定这些池的最佳大小，但是在本节的上下文中，牢记它们代表了一个明显的等待点。

调优通讯基础设施远远超出了本文讨论的范围 ,因为其具体实现因产品不同而存在明显

的区别，这包括诸如 MSMQ、MQSeries、TIBCO 等主流产品。但是请记住，如果一个应用与某消息服务器交互，它必须经过合适的调优或者它也代表了一个等待点。

最后一个明显影响 JVM 性能的等待点是垃圾收集。它不太适用本文中描述的等待点分析过程（检查一个请求，定位导致该请求等待的技术），但是由于它可 以对性能产生显著的影响，所以把它列在这里。不同的 JVM 实现和不同的垃圾收集策略决定了垃圾收集如何执行，但是在很多情况下，一次主垃圾收集（或者说标 记—清除—压缩垃圾收集）可能导致整个 JVM 暂停直到垃圾收集完成。一个显著提高 JVM 性能的办法就是优化垃圾收集。如果想了解更多垃圾收集的信息，请加 入 GeekCap 讨论应用基础设施调优。

后退调优

现在关于基于层次的和基于技术的等待点的一切都介绍完了 ,最后一步就是优化每一个等待点的配置。这一步有时被称为“后退调优”，其思想非常简单：

1. 开放所有基于层次的等待点和外部依赖池——也就是配置它们允许过多的负载经过服务器。
2. 根据应用生成均衡的和具有代表性的服务请求。
3. 定位首先透支的等待点，通常是外部依赖，比如数据库。
4. 减小配置以控制等待点允许足够的负载经过外部依赖而不透支。
5. 调整所有其他基于层次的等待点，发送足够的负载经过服务器，最大化受限制的等待点，但是也不让请求等待。
6. 允许所有其他请求在业务逻辑层之上等待，比如 web 服务器端。

此处的原则就是应用应该发送一定数量的负载给它的外部依赖资源以最大化它们的使用率又不导致透支—并且所有其他等待点应该合理配置以发送足够的负载 给这些受限制的等待点。例如，如果数据库对每一个应用服务器最多支持 50 个连接（例如，配置池容纳 40 或 45 个连接）。接下来，如果 80 个线程产生 40 个 数据库连接，则应用的线程池应配置为 80。最后，web 服务器在任意时刻应该发送不超过 80 个请求给每一个应用服务器。

所有基于技术的等待点，比如对象池、缓存和垃圾回收，应该调整到最大化请求的吞吐量使得尽可能快的穿越服务器或者基于层析的等待点之间。

总结

性能调优曾经是“艺术性”多于“科学性”，但是通过结合抽象分析和尝试并产生错误，基于等待的调优方法已经证明能够使该过程更具科学性和更有效率。 基于等待的调优首先执行一个应用架构的等待点分析，以此定位有可能导致请求等待的某个技术。等待点来自两方面：基于层次的等待点，代表着跨越应用层次的转 换；基于技术的等待点，代表着可能提高或降低性能的技术，比如缓存、池和通讯基础设施。一旦定位好了一系列等待点，调优过程就此开始 :开放所有基于层次的 等待点和外部依赖池 ,产生均衡的和具有代表性的负载，然后后退调优，收紧等待点以最大化该请求最薄弱的一环的性能，但是不要透支。基于等待的调优方法在生 产环境中已经一次又一次得到了证明，不仅仅是高效的，而且允许性能工程师快速实现可度量的性能优化。

关于作者

Steven Haines 是 GeekCap 公司的创立者和 CEO ,该公司致力于向全球的开发社区提供电子学习解决方案 ,尤其侧重于诸如性能测试和性能调优这样生僻的领域。除了电子学习方案，GeekCap 还提供了免费的技术论坛、 学习路线图和其他资源以帮助开发人员发展自己的技术事业和学习新技术。GeekCap 提供的服务包括：市场营销资料，比如白皮书和技术概要；业务服务，比 如市场分析和战略定位。Steven 的著作包括：许多白皮书、Java EE 5 性能管和优化(Apress ,2006),Java 2 Primer Plus （ SAMS， 2002)和从零学习 Java 2(QUE ,1999)。他是 InformIT.com 的 Java 版主，同时也是 InfoQ.com 的 Java 社区编辑。平时他作为一名承包商在 Disney 团队工作，负责实现下一代 Disney 网站的架构。他之前在 Quest Software 公司工作了七年，职责是作为 Java 领域专家设计性能监控和分析软件。他先后在 California 大学 Irvine 分校和 Learning Tree 大学接受过全面的 Java 开发培训。您可以通过 steve@geekcap.com 联系他。

原文链接：http://www.infoq.com/cn/articles/Wait-Based-Tuning-Steven-Haines

相关内容：

- 现代应用性能管理深度概览
- 37signals 使用 New Relic 提供的 Rails 性能监控器
- Ruby 基准评测套件初探
- 测算团队，而不是个人
- 采访 XRuby 开发者：“有趣的” Ruby 实现

[推荐文章]

剖析短迭代

作者 Dave Nicolette 译者 郑柯

推荐理由：作者从利弊两个角度，对短迭代周期进行了透彻分析。凡事有利必有弊，重构、测试驱动、持续集成、短迭代周期、现场客户等各种敏捷实践都需要在成本和收益之间权衡。这篇文章不仅仅是对短迭代的一篇分析，更是帮助读者清楚的看到，要学会权衡，不要盲从。实践出真知。

推荐人：李剑，中国 Eclipse 社区插件开发版版主，北航软件工程硕士。热衷于设计模式，敏捷软件开发的研究与实践。他的个人 Blog 地址为：http://dear.blogbus.com。

很多人都觉得：迭代的长度应该由发布周期的长短确定。我不同意，我认为这两个周期之间不应有关系。相对于长迭代来说，短迭代可以提供更为频繁的客户 反馈， 同时也给予团队机会，让他们可以反思并改进自己的工作实践。短周期可以形成“心跳节奏”，这样的快节奏也足以展现更多意义。由于短周期的本性使然，团队不 大有机会创建过于冗长的工作项目，而这样的项目会使得人们很难产生成就感，除非等到大量的工作完成之后。即使发布的周期很长，下面这些好处仍然存在。

好处

1. **快速响应。**在不影响正在进行的工作的情况下优先快速响应变化。产品负责人、客户或是代理人在迭代中期改变优先 级或是添加新功能，这样的情况很多见。如果迭代时间足够 短，这种状况就可以得到更好的处理，因为变更在下个迭代中就可以容纳进来了，这样也可以避免打乱当前迭代本不应受影响的正常节奏。

2. **问题检测。** 成熟的敏捷团队能够发现流程上的问题并马上处理。然而，目前看来，很多敏捷团队仍然在学习曲线上前行，他们还没有成熟到可以自己 发现并解决问题的地步。他们需要根据项目度量数据的变化来识别问题。由于趋势要靠三个点的连线才能体现出来，而项目数据每个迭代才能收集一次，因此更短的 迭代可以更快地暴露问题。

3. **范围管理。**如果待办事项列表中的条目都很小，那就可以灵活移动。较长的迭代会产生较大的用户故事。如果产品负责人需要变更待办事项列表条目的优先级，如果用户故事较大，那么变更这些用户故事造成的影响也更大。较短的迭代则趋向产生较小的用户故事。遵循 INVEST 原则 ，产品负责人也更容易变更用户故事的优先级。

4. **迭代规划和跟踪。** 长迭代产生的较大的用户故事，经常要被分解为“任务”，也就是要将大块儿的开发工作拆分为可操作性更强的明细任务。接下 来，为了让团队知道所有用户故事的状态，这些任务要在迭代中跟踪，要么使用类似于“看板”的系统，要么使用迭代的燃尽图。很多团队每天都会停下来重新估算 尚未完成的个人任务。使用短迭代，可以去除所有这些内部流程的管理成本；用户故事变成了更小的工作单位，而人们也能够以更简单的方式跟踪迭代状态。

5. 成为转向“**无迭代流程**” 的基础。迭代式开发保留了一些瀑布开发过程固有的管理成本，即使我们付出代价想去掉它们也是如此。如果将每个迭代从 头到尾画一个价值流累积图(cumulative flow diagram) ,这些管理成本就会以“在途时间(lead time)”的形式体现出来。我参与过的一些团队，他们在自己承受范围之内尽量压缩迭代的时间。我注意到他们可以消除大量类似的管理成本。迭代时间越短， 让一切工作顺利进行所需花费的流程管理成本就越少。

从另一方面来看，在有严格时间限制的迭代中工作，也可以带来一些敏捷方法的附加价值，包括频繁和有规律的演示和回顾、用来交付增量开发结果的一致性 时间 表、频繁得到客户反馈的机会、以及对于“心跳”或是“脉搏”类似节奏的感觉，这样的感觉可以让团队在长期的开发过程中保证认真投入。使用时间盒的方式工作 是有一些额外的好处的，而有些团队在采纳无迭代的流程时会把这些好处丢掉，这就等于是“连孩子带洗澡水一起泼出去了”。

而使用短迭代，可以减少转向超轻量 级、无迭代的流程所带来的痛苦；这也是可以预期的。

人们在转向无迭代流程时经常会犯一个错误，他们会将所有与“迭代”有关系的实践都抛弃掉。我们要将“迭代”的概念与有附加价值的敏捷开发特定实践区分开，并寻找能够减少流程管理成本、同时还可以保留有价值实践的解决之道。

潜在问题

有人在使用短迭代时遇到了困难。短迭代的拥护者 Mishkin Berteig 也提到一些潜在的问题：

- “密集的工作回让人筋疲力尽。”我想这是团队选择何种工作方式的问题。周期短，不一定意味着工作就一定密集。短迭代可能仅仅 意味着小时间盒；也就是说，每 个时间盒承诺交付的工作更少了。在工作密度上不一定有什么变化。其他的敏捷原则（特别是“可持续的步调”）就是为了防止发生筋疲力尽的情况。
- “战略层面的思考很难跟日程相结合。”战略层面的思考跟每个迭代要做的具体工作没有太大干系。迭代是战术层面的。战略层面的思考是……呃，非战术层面的。这听起来更像是管理上的问题，而不是短迭代的特性之类的东西。
- “ 每个迭代必须完成的、耗费管理成本的相关任务占用了短迭代的大部分时间。”这似乎又是一个团队如何选择工作方式的问题。我曾观察到挤压迭代时间长度而引发的一个有趣结果：人们首先“发现”一些并不是非常必要的管理任务，然后就不再做它们了。最后，团队只做必要的事情，换句话说，他们去除了流程中的浪费。 实际上，这些观察让我对 Jim Shore 在 Java Ranch 上的发言持 保留意见。他认为更长的迭代给团队的压力更小，因此有经验的敏捷团队更适合用长期的迭代。我觉得我们不必在迭代规划上花费更多时间， 我认为迭代规划还可以更少些。我支持更短的迭代，如果客户可以采取拉式的方法以单件流 （ single piece flow ）的方式提出需求，这些迭代甚至可能逐渐消弭。
- “对团队之外的资源或是人员的等待，这会使得工作的完成要跨越多个 迭代。”组织上的约束造成了此类状况。如果试图采取的迭代长 度过短，以至于组织不能应

对，这样做并不合适。如果真这么做了，也就不能称之为“迭代”了，因为不可能在那样短的时间内交付工作结果，而组织也无法吸收这样的结果。要想有所进步，我们必须识别出组织的约束。我并不认为临时的组织约束（它们是临时的，只要你真心愿意改变）就会使得短迭代不可行。简单么？没人会这么想。但如果组织的变革很容易的话，那就没什么乐趣了，不是么？

原文链接：http://www.infoq.com/cn/articles/short-iterations-argument

相关内容：

- InfoQ 书评：敏捷模式
- 坚定实施回顾中列举的改变
- 提供给每个人的行为驱动开发
- 视频：Segundo Velasquez 与客户眼中的敏捷
- 迷你书下载：硝烟中的 Scrum 和 XP

时刻关注企业软件开发领域的变化与创新

InfoQ中文站

www.infoq.com/cn

Java — .NET — Ruby — SOA — Agile — Architecture

Java社区：企业Java社区的变化与创新

.NET社区：.NET和微软的其它企业软件开发解决方案

Ruby社区：面向Web和企业开发的Ruby，主要关注Ruby on Rails

SOA社区：关于大中型企业内面向服务架构的一切

Agile社区：敏捷软件开发和项目经理

Architecture社区：设计、技术趋势及架构师所感兴趣的话题

讨 论 组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
广告合作：sales@cn.infoq.com

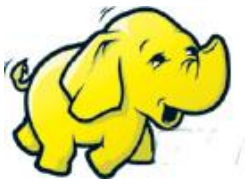
[推荐文章]

分布式计算开源框架 Hadoop 的配置与开发

作者 岑文初

推荐理由：*这一系列从介绍、配置到应用，给读者一个了解 Hadoop 的多方面视角，三者合并起来可以使读者对 Hadoop 有一个总体认识。*

推荐人：宋玮，有多年软件开发经验，从 2002 年开始使用 Java，在各个项目开发过程中先后使用过 Struts、Oracle ADF、Spring Framework、AspectJ 等。他长期担任技术管理和项目管理工作，一直关心开源软件的发展动态以及软件过程和敏捷开发的实践探索。他的 Blog 为 http://www.donews.net/victorsong。



在 SIP 项目设计的过程中，对于它庞大的日志在开始时就考虑使用任务分解的多线程处理模式来分析统计，在我从前写的文章《Tiger Concurrent Practice --日志分析并行分解设计与实现》中有所提到。但是由于统计的内容暂时还是十分简单，所以就采用 Memcache 作为计数器，结合 MySQL 就完成了访问 控制以及统计的工作。然而未来，对于海量日志分析的工作，还是需要有所准备。现在最火的技术词汇莫过于“云计算”，在 Open API 日益盛行的今天，互联网应用的数据将会越来越有价值，如何去分析这些数据，挖掘其内在价值，就需要分布式计算来支撑海量数据的分析工作。

回过头来看，早先那种多线程，多任务分解的日志分析设计，其实是分布式计算的一个单机版缩略，如何将这种单机的工作进行分拆，变成协同工作的集群， 其实就是分布式计算框架设计所涉及的。在去年参加 BEA 大会的时候，BEA 和 VMWare 合作采用虚拟机来构建集群，无非就是希望使得计算机硬件能够类似 于应用程序中资源池的资源，使用者无需关心资源的分配情况，从而最大化了硬件资源的使用价值。分布式计算也是如此，具体的计算任务交由哪一台机器执行，执 行后由谁来汇总，这都由分布式框架的 Master 来抉择，而使用者只需简单地将待分析内容提供给分布式计算系统作为输入 ,就可以得到分布式计算后的结果。

Hadoop 是 Apache 开源组织的一个分布式计算开源框架 ,在很多大型网站上都已经得到了应用，如亚马逊、Facebook 和 Yahoo 等等。 对于我来说，最近的一个使用点就是服务集成平台的日志分析。服务集成平台的日志量将会很大，而这也正好符合了分布式计算的适用场景（日志分析和索引建立就 是两大应用场景）。

当前没有正式确定使用，所以也是自己业余摸索，后续所写的相关内容，都是一个新手的学习过程，难免会有一些错误，只是希望记录下来可以分享给更多志同道合的朋友。

什么是 Hadoop？

搞什么东西之前，第一步是要知道 What（是什么），然后是 Why（为什么），最后才是 How（怎么做）。但很多开发的朋友在做了多年项目以后，都习惯是先 How，然后 What，最后才是 Why，这样只会让自己变得浮躁，同时往往会将技术误用于不适合的场景。

Hadoop 框架中最核心的设计就是 :MapReduce 和 HDFS。MapReduce 的思想是由 Google 的一篇论文所提及而被广为流传的， 简单的一句话解释 MapReduce 就是“任务的分解与结果的汇总”。HDFS 是 Hadoop 分布式文件系统（Hadoop Distributed File System）的缩写，为分布式计算存储提供了底层支持。

MapReduce 从它名字上来看就大致可以看出个缘由，两个动词 Map 和 Reduce，“Map（展开）”就是将一个任务分解成为多个任 务，“Reduce”就是将分解后多任务处理的结果汇总起来，得出最后的分析结果。这不是什么新思想，其实在前面提到的多线程，多任务的设

计就可以找到这 种思想的影子。不论是现实社会，还是在程序设计中，一项工作往往可以被拆分成为多个任务 ,任务之间的关系可以分为两种 :一种是不相关的任务 ,可以并行执 行；另一种是任务之间有相互的依赖，先后顺序不能够颠倒，这类任务是无法并行处理的。回到大学时期，教授上课时让大家去分析关键路径，无非就是找最省时的 任务分解执行方式。在分布式系统中，机器集群就可以看作硬件资源池，将并行的任务拆分，然后交由每一个空闲机器资源去处理，能够极大地提高计算效率，同时 这种资源无关性，对于计算集群的扩展无疑提供了最好的设计保证。（其实我一直认为 Hadoop 的卡通图标不应该是一个小象，应该是蚂蚁 ,分布式计算就好比 蚂蚁吃大象 ,廉价的机器群可以匹敌任何高性能的计算机，纵向扩展的曲线始终敌不过横向扩展的斜线)。任务分解处理以后，那就需要将处理以后的结果再汇总起 来，这就是 Reduce 要做的工作。

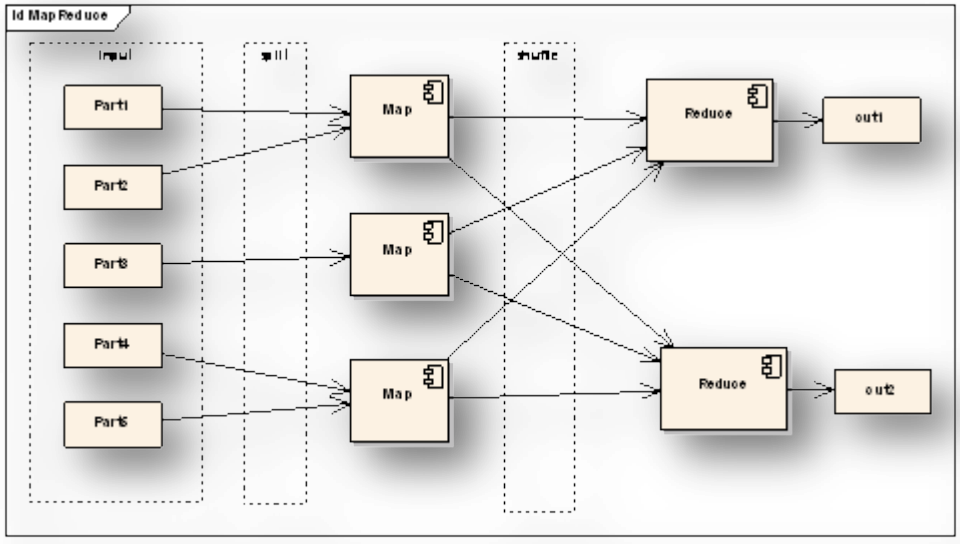


图 1：MapReduce 结构示意图

上图就是 MapReduce 大致的结构图，在 Map 前还可能会对输入的数据有 Split（分割）的过程，保证任务并行效率，在 Map 之后还会有 Shuffle（混合）的过程，对于提高 Reduce 的效率以及减小数据传输的压力有很大的帮助。后面会具体提及这些部分的细节。

HDFS 是分布式计算的存储基石 ,Hadoop 的分布式文件系统和其他分布式文件系统有很多类似的特质。分布式文件系统基本的几个特点：

1. 对于整个集群有单一的命名空间。

2. 数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。

3. 文件会被分割成多个文件块，每个文件块被分配存储到数据节点上，而且根据配置会由复制文件块来保证数据的安全性。

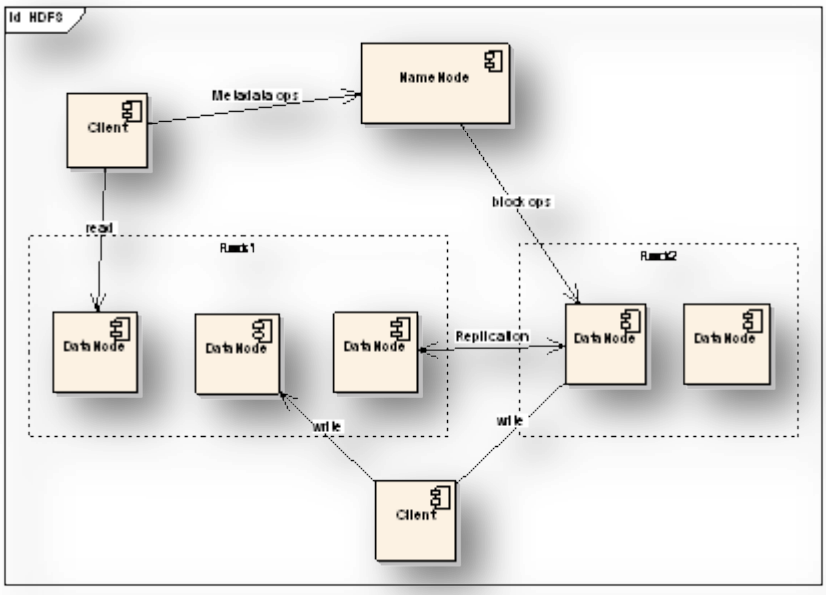


图 2：HDFS 结构示意图

上图中展现了整个 HDFS 三个重要角色：NameNode、DataNode 和 Client。NameNode 可以看作是分布式文件系统中的管理 者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode 会将文件系统的 Meta-data 存储在内存中，这些信息主要包括 了文件信息、每一个文件对应的文件块的信息和每一个文件块在 DataNode 的信息等。DataNode 是文件存储的基本单元，它将 Block 存储在本地 文件系统中，保存了 Block 的 Meta-data，同时周期性地将所有存在的 Block 信息发送给 NameNode。Client 就是需要获取分布式文 件系统文件的应用程序。这里通过三个操作来说明他们之间的交互关系。

文件写入：

1. Client 向 NameNode 发起文件写入的请求。

2. NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode

的信息。

3. Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。

文件读取：

1. Client 向 NameNode 发起文件读取的请求。
2. NameNode 返回文件存储的 DataNode 的信息。
3. Client 读取文件信息。

文件 Block 复制：

1. NameNode 发现部分文件的 Block 不符合最小复制数或者部分 DataNode 失效。
2. 通知 DataNode 相互复制 Block。
3. DataNode 开始直接相互复制。

最后再说一下 HDFS 的几个设计特点（对于框架设计值得借鉴）：

1. Block 的放置：默认不配置。一个 Block 会有三份备份，一份放在 NameNode 指定的 DataNode，另一份放在 与指定 DataNode 非同一 Rack 上的 DataNode，最后一份放在与指定 DataNode 同一 Rack 上的 DataNode 上。备份无非就是为了 数据安全，考虑同一 Rack 的失败情况以及不同 Rack 之间数据拷贝性能问题就采用这种配置方式。
2. 心跳检测 DataNode 的健康状况，如果发现问题就采取数据备份的方式来保证数据的安全性。
3. 数 据复制（场景为 DataNode 失败、需要平衡 DataNode 的存储利用率和需要平衡 DataNode 数据交互压力等情况）：这里先说一下，使用 HDFS 的 balancer 命令，可以配置一个 Threshold 来平衡每一个 DataNode 磁盘利用率。例如设置了 Threshold 为 10%，那么 执行 balancer 命令的时候，首先统计所有 DataNode 的磁盘利用率的均值 然后判断如果某一个 DataNode 的磁盘利用率超过这个均值 Threshold 以上，

那么将会把这个 DataNode 的 block 转移到磁盘利用率低的 DataNode，这对于新节点的加入来说十分有用。

4. 数据交验：采用 CRC32 作数据交验。在文件 Block 写入的时候除了写入数据还会写入交验信息，在读取的时候需要交验后再读入。

5. NameNode 是单点：如果失败的话，任务处理信息将会纪录在本地文件系统和远端的文件系统中。

6. 数 据管道性的写入：当客户端要写入文件到 DataNode 上，首先客户端读取一个 Block 然后写到第一个 DataNode 上，然后由第一个 DataNode 传递到备份的 DataNode 上，一直到所有需要写入这个 Block 的 NataNode 都成功写入，客户端才会继续开始写下一个 Block。

7. 安全模式：在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安全模式的情况下，文 件系统中的内容不允许修改也不允许删除，直到安全模式结束。安全模式主要是为了系统启动的时候检查各个 DataNode 上数据块的有效性，同时根据策略必 要的复制或者删除部分数据块。运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只 需要等待一会儿即可。

下面综合 MapReduce 和 HDFS 来看 Hadoop 的结构：

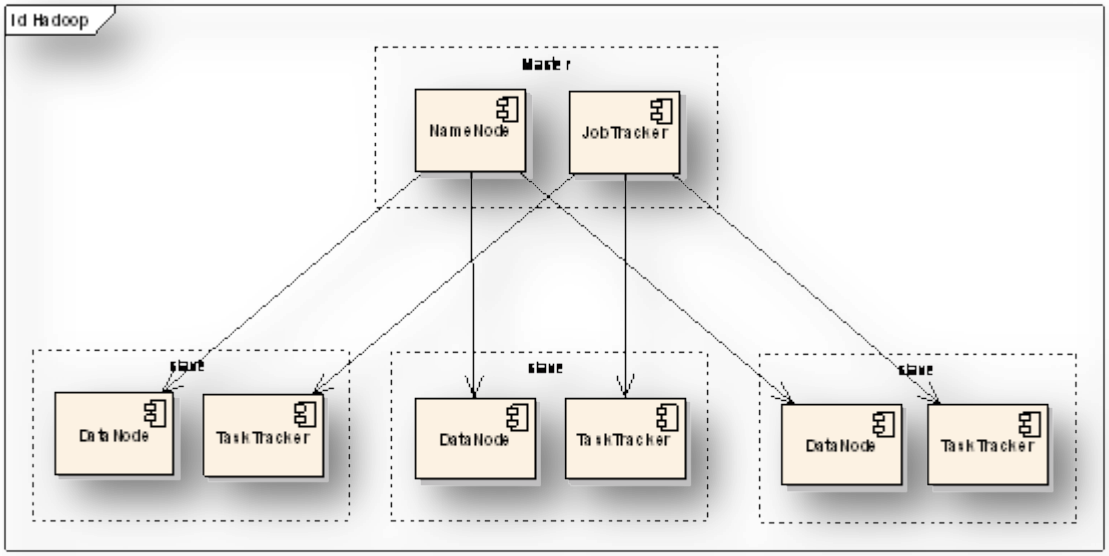


图 3：Hadoop 结构示意图

在 Hadoop 的系统中，会有一台 Master，主要负责 NameNode 的工作以及 JobTracker 的工作。JobTracker 的主要职责 就是启动、跟踪和调度各个 Slave 的任务执行。还会有多台 Slave，每一台 Slave 通常具有 DataNode 的功能并负责 TaskTracker 的 工作。TaskTracker 根据应用要求来结合本地数据执行 Map 任务以及 Reduce 任务。

说到这里，就要提到分布式计算最重要的一个设计点：Moving Computation is Cheaper than Moving Data。就是在分布式处理中，移动数据的代价总是高于转移计算的代价。简单来说就是分而治之的工作，需要将数据也分而存储，本地任务处理本地数据然后归 总，这样才会保证分布式计算的高效性。

为什么要选择 Hadoop？

说完了 What，简单地说一下 Why。官方网站已经给了很多的说明，这里就大致说一下其优点及使用的场景（没有不好的工具，只用不适用的工具，因此选择好场景才能够真正发挥分布式计算的作用）：

1. 可扩展：不论是存储的可扩展还是计算的可扩展都是 Hadoop 的设计根本。
2. 经济：框架可以运行在任何普通的 PC 上。

3. 可靠：分布式文件系统的备份恢复机制以及 MapReduce 的任务监控保证了分布式处理的可靠性。

4. 高效：分布式文件系统的高效数据交互实现以及 MapReduce 结合 Local Data 处理的模式，为高效处理海量的信息作了基础准备。

使用场景：个人觉得最适合的就是海量数据的分析，其实 Google 最早提出 MapReduce 也就是为了海量数 据分析。同时 HDFS 最早是为了搜索引擎实现而开发的，后来才被用于分布式计算框架中。海量数据被分割于多个节点，然后由每一个节点并行计算，将得出的结果归并到输出。同时第一阶段的输出又可以作为下一阶段计算的输入，因此可以想象到一个树状结构的分布式计算图，在不同阶段都有不同产出，同时并行和串行结 合的计算也可以很好地在分布式集群的资源下得以高效的处理。

其实参看 Hadoop 官方文档已经能够很容易配置分布式框架运行环境了，不过这里既然写了就再多写一点，同时有一些细节需要注意的也说明一下，其实 也就是这些细节会让人摸索半天。Hadoop 可以单机跑，也可以配置集群跑，单机跑就不需要多说了，只需要按照 Demo 的运行说明直接执行命令即可。这里 主要重点说一下集群配置运行的过程。

环境

7 台普通的机器，操作系统都是 Linux。内存和 CPU 就不说了，反正 Hadoop 一大特点就是机器在多不在精。JDK 必须是 1.5 以上的，这个切记。7 台机器的机器名务必不同，后续会谈到机器名对于 MapReduce 有很大的影响。

部署考虑

正如上面我描述的，对于 Hadoop 的集群来说，可以分成两大类角色：Master 和 Slave，前者主要配置 NameNode 和 JobTracker 的角色，负责总管分布式数据和分解任务的执行，后者配置 DataNode 和 TaskTracker 的角色，负责分布式数据存储以及任 务的执行。本来我打算看看一台机器是否可以配置成 Master，同时也作为 Slave 使用，不过发现在 NameNode 初始化的过程中以及 TaskTracker 执行过程中机器名配置好像有冲突（NameNode 和 TaskTracker 对于 Hosts 的配置有些冲突，究竟是把机器名对应 IP 放在配置前面还是把

Localhost 对应 IP 放在前面有点问题，不过可能也是我自己的问题吧，这个大家可以根据实施情况给我反馈）。最后反正决定一 台 Master，六台 Slave，后续复杂的应用开发和测试结果的比对会增加机器配置。

实施步骤

1. 在所有的机器上都建立相同的目录，也可以就建立相同的用户，以该用户的 home 路径来做 hadoop 的安装路径。例如我在所有的机器上都建立了/home/wenchu。
2. 下载 Hadoop，先解压到 Master 上。这里我是下载的 0.17.1 的版本。此时 Hadoop 的安装路径就是/home/wenchu/hadoop-0.17.1。
3. 解压后进入 conf 目录，主要需要修改以下文件：hadoop-env.sh，hadoop-site.xml、masters、slaves。

Hadoop 的基础配置文件是 hadoop-default.xml，看 Hadoop 的代码可以知道，默认建立一个 Job 的时候会建立 Job 的 Config，Config 首先读入 hadoop-default.xml 的配置，然后再读入 hadoop-site.xml 的配置（这个文件初始的时候配置为空），hadoop-site.xml 中主要配置你需要覆盖的 hadoop-default.xml 的系统级配置，以及你需要在你的 MapReduce 过程中使用的自定义配置（具体的一些使用例如 final 等参考文档）。

以下是一个简单的 hadoop-site.xml 的配置：

```
    <?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
  <name>fs.default.name</name>//你的 namenode 的配置，机器名加端口
  <value>hdfs://10.2.224.46:54310/</value>
</property>
<property>
  <name>mapred.job.tracker</name>//你的 JobTracker 的配置，机器名加端口
  <value>hdfs://10.2.224.46:54311/</value>
</property>
```

```
<property>
  <name>dfs.replication</name>//数据需要备份的数量，默认是三
  <value>1</value>
</property>
<property>
   <name>hadoop.tmp.dir</name>//Hadoop 的默认临时路径，这个最好配置，如果在新增节点或者其他情况下莫名其妙的 DataNode 启动不了，就删除此文件中的 tmp 目录即可。不过如果删除了 NameNode 机器的此目录，那么就需要重新执行 NameNode 格式化的命令。
   <value>/home/wenchu/hadoop/tmp/</value>
</property>
<property>
  <name>mapred.child.java.opts</name>//java 虚拟机的一些参数可以参照配置
  <value>-Xmx512m</value>
</property>
<property>
 <name>dfs.block.size</name>//block 的大小，单位字节，后面会提到用处，必须是 512 的倍数，因为采用 crc 作文件完整性校验，默认配置 512 是 checksum 的最小单元。
 <value>5120000</value>
 <description>The default block size for new files.</description>
</property>
</configuration>
```

hadoop-env.sh 文件只需要修改一个参数：

```
    # The java implementation to use.    Required.
export JAVA_HOME=/usr/ali/jdk1.5.0_10
```

配置你的 Java 路径，记住一定要 1.5 版本以上，免得莫名其妙出现问题。

Masters 中配置 Masters 的 IP 或者机器名 ,如果是机器名那么需要在/etc/hosts 中有所设置。Slaves 中配置的是 Slaves 的 IP 或者机器名，同样如果是机器名需要在/etc/hosts 中有所设置。范例如下，我这里配置的都是 IP：

```
    Masters:
10.2.224.46
```

```
Slaves:
10.2.226.40
10.2.226.39
10.2.226.38
10.2.226.37
10.2.226.41
10.2.224.36
```

4. 建立 Master 到每一台 Slave 的 SSH 受信证书。由于 Master 将会通过 SSH 启动所有 Slave 的 Hadoop，所以需要建立单向或者双向证书保证命令执行时不需要再输入密码。在 Master 和所有的 Slave 机器上执行：ssh-keygen -t rsa。执行此命令的时候，看到提示只需要回车。然后就会在/root/.ssh/下面产生 id_rsa.pub 的证书文件，通过 scp 将 Master 机器上的这个文件拷贝到 Slave 上（记得修改名称），例如：scp root@masterIP:/root/.ssh/id_rsa.pub /root/.ssh/46_rsa.pub，然后执行 cat /root/.ssh/46_rsa.pub >>/root/.ssh/authorized_keys，建立 authorized_keys 文 件即可，可以打开这个文件看看，也就是 rsa 的公钥作为 key，user@IP 作为 value。此时可以试验一下，从 master ssh 到 slave 已经不需要密码了。由 slave 反向建立也是同样。为什么要反向呢？其实如果一直都是 Master 启动和关闭的话那么没有必要建立反 向，只是如果想在 Slave 也可以关闭 Hadoop 就需要建立反向。

5. 将 Master 上的 Hadoop 通过 scp 拷贝到每一个 Slave 相同的目录下，根据每一个 Slave 的 Java_HOME 的不同修改其 hadoop-env.sh。

6. 修改 Master 上/etc/profile：

   新增以下内容：（具体的内容根据你的安装路径修改，这步只是为了方便使用）

```
    export HADOOP_HOME=/home/wenchu/hadoop-0.17.1
export PATH=$PATH:$HADOOP_HOME/bin
```

   修改完毕后，执行 source /etc/profile 来使其生效。

7. 在 Master 上执行 Hadoop namenode –format，这是第一需要做的初始化，可以看作格式化吧，以后除了在上面我提到过删除了 Master 上的 hadoop.tmp.dir 目录，否则是不需要再次执行的。

8. 然后执行 Master 上的 start-all.sh，这个命令可以直接执行，因为在 6 中已经添加到了 path 路径，这个命令是启动 hdfs 和 mapreduce 两部分，当然你也可以分开单独启动 hdfs 和 mapreduce，分别是 bin 目录下的 start-dfs.sh 和 start-mapred.sh。
9. 检查 Master 的 logs 目录，看看 Namenode 日志以及 JobTracker 日志是否正常启动。
10. 检查 Slave 的 logs 目录看看 Datanode 日志以及 TaskTracker 日志是否正常。
11. 如果需要关闭，那么就直接执行 stop-all.sh 即可。

以上步骤就可以启动 Hadoop 的分布式环境，然后在 Master 的机器进入 Master 的安装目录，执行 hadoop jar hadoop-0.17.1-examples.jar wordcount 输入路径和输出路径，就可以看到字数统计的效果了。此处的输入路径和输出路径都指的是 HDFS 中的路径，因此你可以首先通过拷贝本地文件系统中的目录到 HDFS 中的方式来建立 HDFS 中的输入路径：

hadoop dfs -copyFromLocal /home/wenchu/test-in test-in。其中/home/wenchu/test-in 是本地路径，test-in 是将会建立在 HDFS 中的路径，执行完毕以后可以通过 hadoop dfs –ls 看到 test-in 目录已经存在，同时可以通过 hadoop dfs –ls test-in 查看里面的内容。输出路径要求是在 HDFS 中不存在的，当执行完那个 demo 以后，就可以通过 hadoop dfs –ls 输出路径看到其中的内容，具体文件的内容可以通过 hadoop dfs –cat 文件名称来查看。

经验总结和注意事项（这部分是我在使用过程中花了一些时间走的弯路）：

1. Master 和 Slave 上的几个 conf 配置文件不需要全部同步，如果确定都是通过 Master 去启动和关闭，那么 Slave 机器上的配置不需要去维护。但如果希望在任意一台机器都可以启动和关闭 Hadoop，那么就需要全部保持一致了。
2. Master 和 Slave 机器上的/etc/hosts 中 必须把集群中机器都配置上去，就算在各个配置文件中使用的是 IP。这个吃过不少苦头，原来以为如果配成 IP 就不需要去配置 Host，结果发现在执行 Reduce 的时候总是卡住，在拷贝的时候就无法继续下去，不断重试。另外如果集群中如果有两台机器的机器名如果重复也会出现问题。
3. 如果在新增了节点或者删除节点的时候出现了问题，首先就去删除 Slave 的

hadoop.tmp.dir，然后重新启动试试看，如果还是不行那就干脆把 Master 的 hadoop.tmp.dir 删除（意味着 dfs 上的数据也会丢失），如果删除了 Master 的 hadoop.tmp.dir，那么就需要重新 namenode –format。

4. Map 任务个数以及 Reduce 任务个数配置。前面分布式文件系统设计提到一个文件被放入到分布式文件系统中，会被分割成多个 block 放置到每一个的 DataNode 上，默认 dfs.block.size 应该是 64M，也就是说如果你放置到 HDFS 上的数据小于 64，那么将只有一个 Block，此时会被放置到某一个 DataNode 中，这个可以通过使用命令：hadoop dfsadmin –report 就可以看到各个节点存储的情况。也可以直接去某一个 DataNode 查看目录：hadoop.tmp.dir/dfs/data/current 就 可以看到那些 block 了。Block 的数量将会直接影响到 Map 的个数。当然可以通过配置来设定 Map 和 Reduce 的任务个数。Map 的个数通常默认 和 HDFS 需要处理的 blocks 相同。也可以通过配置 Map 的数量或者配置 minimum split size 来设定，实际的个数为：max(min(block_size,data/#maps),min_split_size)。Reduce 可以通过这个公式计算：0.95*num_nodes*mapred.tasktracker.tasks.maximum。

总的来说出了问题或者启动的时候最好去看看日志，这样心里有底。

Hadoop 中的命令（Command）总结

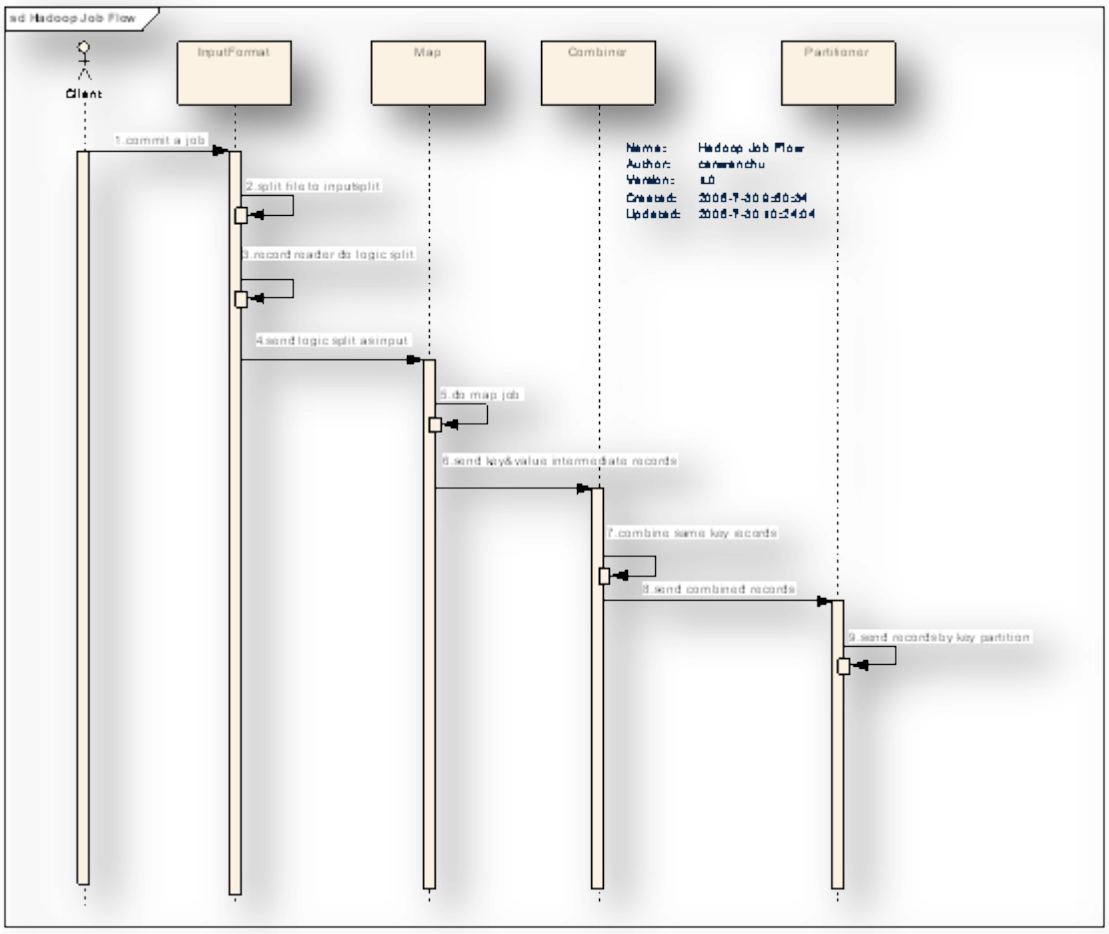
这部分内容其实可以通过命令的 Help 以及介绍了解，我主要侧重于介绍一下我用的比较多的几个命令。Hadoop dfs 这个命令后面加参数就是对于 HDFS 的操作，和 Linux 操作系统的命令很类似，例如：

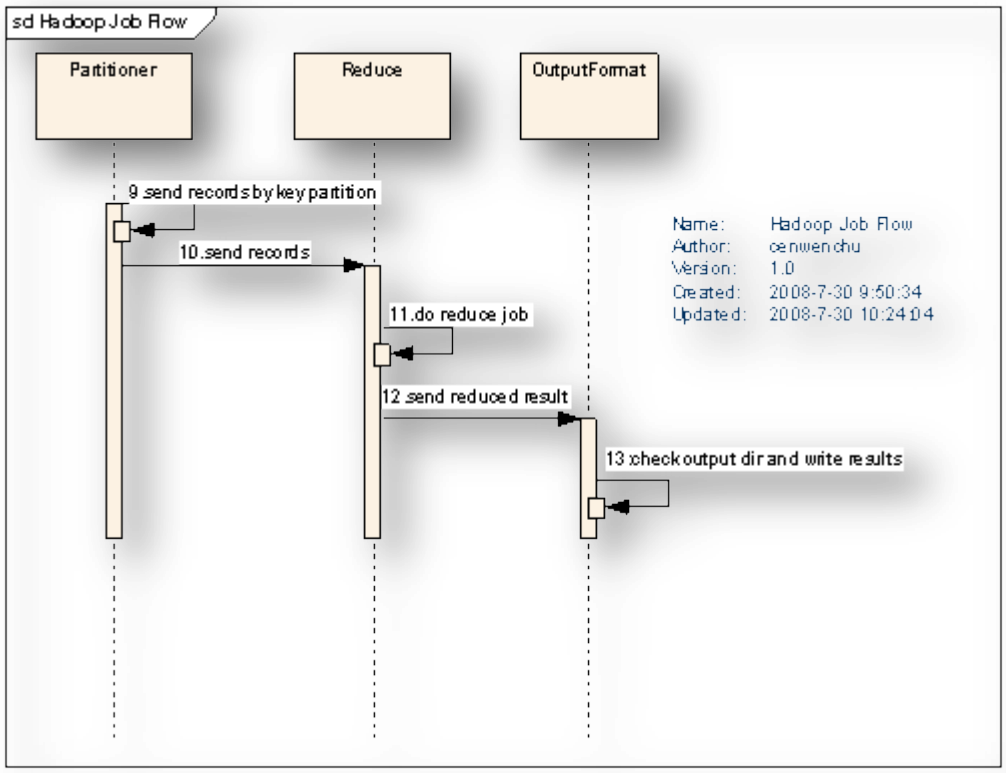
- Hadoop dfs –ls 就是查看/usr/root 目录下的内容，默认如果不填路径这就是当前用户路径；
- Hadoop dfs –rmr xxx 就是删除目录，还有很多命令看看就很容易上手；
- Hadoop dfsadmin –report 这个命令可以全局的查看 DataNode 的情况；
- Hadoop job 后面增加参数是对于当前运行的 Job 的操作，例如 list,kill 等；

- Hadoop balancer 就是前面提到的均衡磁盘负载的命令。

其他就不详细介绍了。

Hadoop 基本流程





一个图片太大了 ,只好分割成为两部分。根据流程图来说一下具体一个任务执行的情况。

1. 在分布式环境中客户端创建任务并提交。
2. InputFormat 做 Map 前的预处理，主要负责以下工作：
    1. 验证输入的格式是否符合 JobConfig 的输入定义，这个在实现 Map 和构建 Conf 的时候就会知道，不定义可以是 Writable 的任意子类。
    2. 将 input 的文件切分为逻辑上的输入 InputSplit，其实这就是在上面提到的在分布式文件系统中 blocksize 是有大小限制的，因此大文件会被划分为多个 block。
    3. 通过 RecordReader 来再次处理 inputsplit 为一组 records 输出给 Map。（inputsplit 只是逻辑切分的第一步，但是如何根据文件中的信息来切分还需要 RecordReader 来实现，例如最简单的默认方式就是回车换行的切分）
3. RecordReader 处理后的结果作为 Map 的输入，Map 执行定义的 Map 逻辑，

输出处理后的 key 和 value 对应到临时中间文件。

4. Combiner 可选择配置，主要作用是在每一个 Map 执行完分析以后，在本地优先作 Reduce 的工作，减少在 Reduce 过程中的数据传输量。

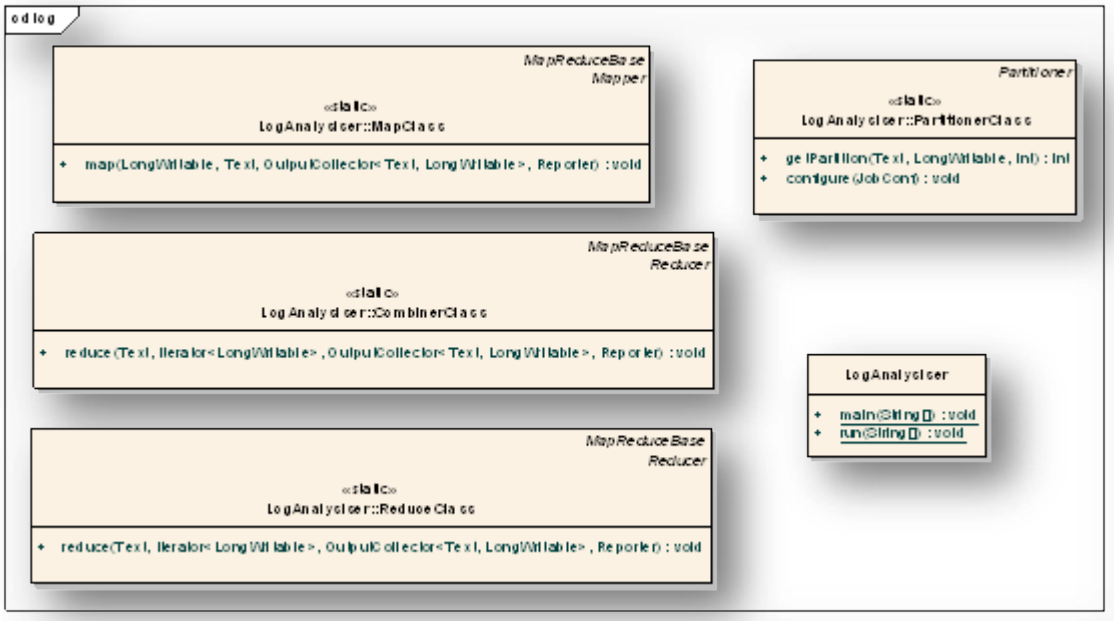
5. Partitioner 可选择配置，主要作用是在多个 Reduce 的情况下，指定 Map 的结果由某一个 Reduce 处理，每一个 Reduce 都会有单独的输出文件。（后面的代码实例中有介绍使用场景）

6. Reduce 执行具体的业务逻辑，并且将处理结果输出给 OutputFormat。

7. OutputFormat 的职责是，验证输出目录是否已经存在，同时验证输出结果类型是否如 Config 中配置，最后输出 Reduce 汇总后的结果。

业务场景和代码范例

业务场景描述：可设定输入和输出路径（操作系统的路径非 HDFS 路径），根据访问日志分析某一个应用访问某一个 API 的总次数和总流量，统计后分别输出到两个文件中。这里仅仅为了测试，没有去细分很多类，将所有的类都归并于一个类便于说明问题。



测试代码类图

LogAnalysiser 就是主类，主要负责创建、提交任务，并且输出部分信息。内部的几个子类用途可以参看流程中提到的角色职责。具体地看看几个类和方法的代码片断：

LogAnalysiser::MapClass

```
        public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable>
  {
    public void map(LongWritable key, Text value, OutputCollector<Text,
LongWritable> output, Reporter reporter)
            throws IOException
    {
      String line = value.toString();//没有配置 RecordReader，所以默认采用 line 的
实现，key 就是行号，value 就是行内容
      if (line == null || line.equals(""))
        return;
      String[] words = line.split(",");
      if (words == null || words.length < 8)
        return;
      String appid = words[1];
      String apiName = words[2];
      LongWritable recbytes = new LongWritable(Long.parseLong(words[7]));
      Text record = new Text();
      record.set(new StringBuffer("flow::").append(appid)
                .append("::").append(apiName).toString());
      reporter.progress();
      output.collect(record, recbytes);//输出流量的统计结果，通过 flow::作为前缀
来标示。
      record.clear();
      record.set(new
StringBuffer("count::").append(appid).append("::").append(apiName).toString());
      output.collect(record, new LongWritable(1));//输出次数的统计结果，通过
count::作为前缀来标示
    }
  }
```

LogAnalysiser:: PartitionerClass

```
        public static class PartitionerClass implements Partitioner<Text, LongWritable>
    {
        public int getPartition(Text key, LongWritable value, int numPartitions)
        {
            if (numPartitions >= 2)//Reduce 个数 ,判断流量还是次数的统计分配到不同
的 Reduce
                if (key.toString().startsWith("flow::"))
                    return 0;
                else
                    return 1;
            else
                return 0;
        }
        public void configure(JobConf job){}
}
```

LogAnalysiser:: CombinerClass

参看 ReduceClass，通常两者可以使用一个，不过这里有些不同的处理就分成了两个。在 ReduceClass 中蓝色的行表示在 CombinerClass 中不存在。

LogAnalysiser:: ReduceClass

```
        public static class ReduceClass extends MapReduceBase
        implements Reducer<Text, LongWritable,Text, LongWritable>
    {
        public void reduce(Text key, Iterator<LongWritable> values,
                OutputCollector<Text, LongWritable> output, Reporter reporter)throws
IOException
        {
            Text newkey = new Text();
            newkey.set(key.toString().substring(key.toString().indexOf("::")+2));
            LongWritable result = new LongWritable();
            long tmp = 0;
            int counter = 0;
            while(values.hasNext())//累加同一个 key 的统计结果
            {
                tmp = tmp + values.next().get();
```

```
            counter = counter +1;//担心处理太久，JobTracker 长时间没有收到报告会
认为 TaskTracker 已经失效，因此定时报告一下
            if (counter == 1000)
            {
                counter = 0;
                reporter.progress();
            }
        }
        result.set(tmp);
        output.collect(newkey, result);//输出最后的汇总结果
    }
}
```

LogAnalysiser

```
    public static void main(String[] args)
{
    try
    {
        run(args);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}
public static void run(String[] args) throws Exception
{
    if (args == null || args.length <2)
    {
        System.out.println("need inputpath and outputpath");
        return;
    }
    String inputpath = args[0];
    String outputpath = args[1];
    String shortin = args[0];
    String shortout = args[1];
    if (shortin.indexOf(File.separator) >= 0)
        shortin = shortin.substring(shortin.lastIndexOf(File.separator));
    if (shortout.indexOf(File.separator) >= 0)
```

```
            shortout = shortout.substring(shortout.lastIndexOf(File.separator));
        SimpleDateFormat formater = new SimpleDateFormat("yyyy.MM.dd");
        shortout = new StringBuffer(shortout).append("-")
            .append(formater.format(new Date())).toString();


        if (!shortin.startsWith("/"))
            shortin = "/" + shortin;
        if (!shortout.startsWith("/"))
            shortout = "/" + shortout;
        shortin = "/user/root" + shortin;
        shortout = "/user/root" + shortout;
        File inputdir = new File(inputpath);
        File outputdir = new File(outputpath);
        if (!inputdir.exists() || !inputdir.isDirectory())
        {
            System.out.println("inputpath not exist or isn't dir!");
            return;
        }
        if (!outputdir.exists())
        {
            new File(outputpath).mkdirs();
        }

        JobConf conf = new JobConf(new Configuration(),LogAnalysiser.class);//构建
Config
        FileSystem fileSys = FileSystem.get(conf);
        fileSys.copyFromLocalFile(new Path(inputpath), new Path(shortin));//将本地文
件系统的文件拷贝到 HDFS 中

        conf.setJobName("analysisjob");
        conf.setOutputKeyClass(Text.class);//输出的 key 类型，在 OutputFormat 会检
查
        conf.setOutputValueClass(LongWritable.class); //输出的 value 类型，在
OutputFormat 会检查
        conf.setMapperClass(MapClass.class);
```

```
        conf.setCombinerClass(CombinerClass.class);
        conf.setReducerClass(ReduceClass.class);
        conf.setPartitionerClass(PartitionerClass.class);
        conf.set("mapred.reduce.tasks", "2");//强制需要有两个 Reduce 来分别处理流量和次数的统计
        FileInputFormat.setInputPaths(conf, shortin);//hdfs 中的输入路径
        FileOutputFormat.setOutputPath(conf, new Path(shortout));//hdfs 中输出路径

        Date startTime = new Date();
            System.out.println("Job started: " + startTime);
            JobClient.runJob(conf);
            Date end_time = new Date();
            System.out.println("Job ended: " + end_time);
            System.out.println("The job took " + (end_time.getTime() - startTime.getTime()) /1000 + " seconds.");
            //删除输入和输出的临时文件
        fileSys.copyToLocalFile(new Path(shortout),new Path(outputpath));
        fileSys.delete(new Path(shortin),true);
        fileSys.delete(new Path(shortout),true);
    }
```

以上的代码就完成了所有的逻辑性代码，然后还需要一个注册驱动类来注册业务 Class 为一个可标示的命令，让 hadoop jar 可以执行。

```
    public class ExampleDriver {
  public static void main(String argv[]){
    ProgramDriver pgd = new ProgramDriver();
    try {
      pgd.addClass("analysislog", LogAnalysiser.class, "A map/reduce program that analysis log .");
      pgd.driver(argv);
    }
    catch(Throwable e){
      e.printStackTrace();
    }
  }
}
```

将代码打成 jar，并且设置 jar 的 mainClass 为 ExampleDriver 这个类。在分布式环境启动以后执行如下语句：

```
hadoop jar analysiser.jar analysislog /home/wenchu/test-in
/home/wenchu/test-out
```

在/home/wenchu/test-in 中是需要分析的日志文件，执行后就会看见整个执行过程，包括了 Map 和 Reduce 的进度。执行完毕会 在/home/wenchu/test-out 下看到输出的内容。有两个文件：part-00000 和 part-00001 分别记录了统计后的结果。 如果需要看执行的具体情况 ,可以看在输出目录下的_logs/history/xxxx_analysisjob ,里面罗列了所有的 Map ,Reduce 的创建情况以及执行情况。在运行期也可以通过浏览器来查看 Map,Reduce 的情况：http://MasterIP:50030 /jobtracker.jsp

Hadoop 集群测试

首先这里使用上面的范例作为测试，也没有做太多的优化配置，这个测试结果只是为了看看集群的效果，以及一些参数配置的影响。

文件复制数为 1，blocksize 5M

Slave 数	处理记录数(万条)	执行时间（秒）
2	95	38
2	950	337
4	95	24
4	950	178
6	95	21
6	950	114

Blocksize 5M

Slave 数	处理记录数(万条)	执行时间（秒）
2（文件复制数为 1）	950	337

Slave 数	处理记录数(万条)	执行时间（秒）
2（文件复制数为 3）	950	339
6（文件复制数为 1）	950	114
6（文件复制数为 3）	950	117

文件复制数为 1

Slave 数	处理记录数(万条)	执行时间（秒）
6(blocksize 5M)	95	21
6(blocksize 77M)	95	26
4(blocksize 5M)	950	178
4(blocksize 50M)	950	54
6(blocksize 5M)	950	114
6(blocksize 50M)	950	44
6(blocksize 77M)	950	74

测试的数据结果很稳定，基本测几次同样条件下都是一样。通过测试结果可以看出以下几点：

1. 机器数对于性能还是有帮助的（等于没说^_^）。
2. 文件复制数的增加只对安全性有帮助，但是对于性能没有太多帮助。而且现在采取的是将操作系统文件拷贝到 HDFS 中，所以备份多了，准备的时间很长。
3. blocksize 对于性能影响很大，首先如果将 block 划分的太小，那么将会增加 job 的数量，同时也增加了协作的代价，降低了性能，但是配置的太大也会让 job 不能最大化并行处理。所以这个值的配置需要根据数据处理的量来考虑。
4. 最后就是除了这个表里面列出来的结果，应该去仔细看输出目录中的_logs/history 中

的 xxx_analysisjob 这个文件，里面记录了全部的执行过程以及读写情况。这个可以更加清楚地了解哪里可能会更加耗时。

随想

“云计算”热的烫手，就和 SAAS、Web2 及 SNS 等一样，往往都是在搞概念，只有真正踏踏实实的大型互联网公司，才会投入人力物力去研究符合自 己的分布式计算。其实当你的数据量没有那么大的时候，这种分布式计算也就仅仅只是一个玩具而已，只有在真正解决问题的过程中，它深层次的问题才会被挖掘出 来。

这三篇文章（分布式计算开源框架 Hadoop 介绍，Hadoop 中的集群配置和使用技巧）仅仅是为了给对分布式计算有兴趣的朋友抛个砖，要想真的掘到金 子，那么就踏踏实实的去用、去想、去分析。或者自己也会更进一步地去研究框架中的实现机制，在解决自己问题的同时，也能够贡献一些什么。

前几日看到有人跪求成为架构师的方式，看了有些可悲，有些可笑，其实有多少架构师知道什么叫做架构？架构师的职责是什么？与其追求这么一个名号 ,还不如踏踏实实地做块石头沉到水底。要知道，积累和沉淀的过程就是一种成长。

原文链接：http://www.infoq.com/cn/articles/hadoop-intro

http://www.infoq.com/cn/articles/hadoop-config-tip

http://www.infoq.com/cn/articles/hadoop-process-develop

相关内容：

- 雅虎架构师谈 MapReduce 和 Hadoop 的未来
- 使用 EhCache Server 部署 1TB 缓存
- 解决云计算安全问题的虚拟专用网络——VPN-Cubed
- JBoss Cache 分布式缓存：Manik Surtani 访谈
- 高效分布式 Session 管理

[推荐文章]

用消费者驱动的契约进行面向服务开发

作者 Ian Robinson 译者 徐涵

推荐理由：SOA？说起来容易，做起来难！向 SOA 过渡给软件开发生命周期带来了许多新的挑战，而本文则针对这些挑战提出了一种解决之道。

推荐人：胡键，2000 年硕士毕业后从事软件开发工作，在实际的项目中长期担任项目经理和技术经理。关心软件技术和相关工具的动态，将其中相对成熟的技术和工具应用到实际的项目之中。已在 IBM developerWorks 中国网站发表数篇文章。目前醉心于服务器端软件的设计和开发，并致力于研究 SOA 方面的规范、技术和工具。他个人 Blog 为：http://foxgem.javaeye.com/

向 SOA 过渡给软件开发生命周期带来了许多新的挑战：机构只有形成一种明确面向服务的开发能力，才能战胜这些挑战。

本文为提高机构的服务开发能力提供了一些实践性建议。本文将先概述 SOA 给软件开发生命周期带来的一些挑战，然后讲述以“服务故事（stories for services）”和服务开发线（service development streams）间交换的单元测试为形式的消费者驱动的契约（consumer-driven contracts）何以能够增强面向服务开发的生命周期。

SOA 给开发带来的挑战

面向服务（service orientation）不仅仅是采纳一种新的架构这么简单。若机构仅使其架

构变得更加面向服务、而不对其开发技术作相应改变的话，那么 SOA 行动肯定要失败。

在启动、构建及运营服务方面的一些挑战包括：

- 在启动阶段，服务功能的描述必须能在多种场合下被重用：粒度既不能粗到仅在一种特定场合下能被重用，也不能细到要做大量补充工作方可在不同场合下被重用。
- 在构建服务时，我们必须确保提供者与消费者可彼此独立地进行演化。如果一项服务功能的消费者们必须随提供者改变而改变，那么该服务就不是真正松耦合的。
- 在运营服务时，我们需要理解各个服务之间的关系，以便我们可以诊断问题、评估服务可用性（availability）发生变化（常常是去除）时将产生的影响、并为各服务针对新的或变化的业务需求而演化设计计划。

从整体资产的角度来看，各服务的具体开发生命周期必须彼此协调一致，且没有不恰当地将各方拴在一个一体的、极其缓慢的、最终无法实行的活动时间表之上。

SOA 给软件开发生命周期带来的挑战源于服务资产的联邦特性。有价值的业务成果不再是由那些“具有单向时间活动表，以及所有权、预算和运营边界都分 散”彼此相隔的应用实现了 ;相反 ,它们是通过那些“拥有各自独特的服务开发生命周期的”服务之间的交互实现的。然而，协调好这些生命周期并不是一次搞定就 完事了的。相反，正如我们将看到的，我们需要识别出一组代表着“对交付一个共同结果的承诺”的协作活动与制品（collaborative activities and artefacts）。这些活动与制品为时刻（若简单地看）将各条开发线联合起来提供了基础。

用敏捷方法实现 SOA，意味着及早交付高价值业务成果且常常需要频繁地发布版本。若机构试图采用这种方法，那只会进一步加剧开发挑战。尽管其名称如 此，但许多人认为敏捷的软件交付方式对与 SOA 相关的机构敏捷效益是不利的。由于在一个服务间存在着很多已知关联的环境中发布新版本的服务会有运营风险， 所以敏捷方法频繁发布版本这一做法常被免去。而且，虽然敏捷鼓励各方进行密切及时地协作，但这种活动难以跨越多个不同服务及其生命周期进行协调。

联邦期望

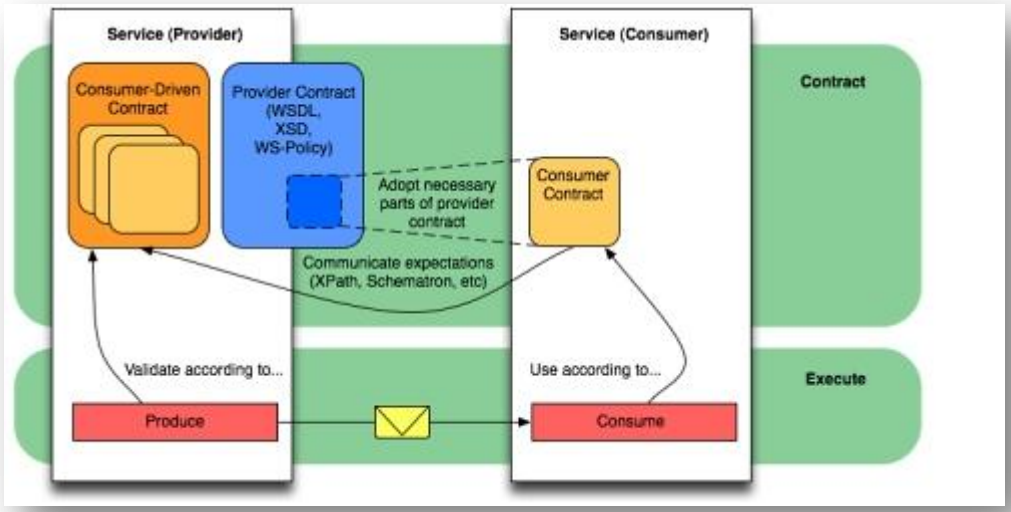
传统的烟囱式应用开发把彼此的交付线（delivery streams）排除在了业务所有权、预算及运营边界之外。面向服务的开发与传统的烟囱式应用开发的不同之处在于，它强调外部的依赖和约束，并把它们整合 到了开发生命周期的每一步中去。这属于 SOA 治理领域：管理交付关联性，并围绕着关于"服务资产及该资产所支持的业务活动、流程及结果"的一个公共表示来 安排交付线。SOA 治理有赖于洞察与反馈——即对服务目录当前状态的洞察，以及关于演化中的服务提供者与消费者所产生影响的反馈。我们可以将联邦的期望和 约束形成制品，这样便可增强洞察与反馈的能力。然后，这些制品就可以在一个特定服务开发线（service development stream）里的各阶段之间转移，以及/或者在不同开发线之间交换了。

要掌握并交换期望与约束，一个实际的做法就是使用所谓的消费者驱动的契约（consumer-driven contracts）。消费者驱动的契约是存在紧密联系的服务契约三件套——提供者契约、消费者契约及消费者驱动的契约——中的一员，它从期望与约束的角度描述了服务提供者与服务消费者之间的关系：

- **提供者契约（Provider contracts）**——提供者契约是我们最为熟悉的一 种的服务契约，参考 WSDL+XML Schema+WS-Policy。顾名思义，提供者契约是以提供者为中心的。提供者规定了它要提供什么；然后，各消费者便将自己绑定到这个一成不变的契 约上。不论消费者实际需要多少功能，消费者接受了提供者契约，就将自己与该提供者的全体功能耦合起来了。

- **消费者契约（Consumer contracts）**——另一方面，消费者契约是对一个消费者的需求更为精确的描述。消费者契约描述了，在一次具体交互场合下，提供者功能中消费者需要的特定部分。消费者契约可被用来标注一个现有的提供者契约，另外消费者契约也有助于发现一个现今尚未规定的提供者契约。

- **消费者驱动的契约（Consumer-driven contracts）**—— 消费者驱动的契约描述的是服务提供者向其所有当前消费者承诺遵守的约束。一旦各消费者把自己的具体期望告知提供者，消费者驱动的契约就被创建了。在提供者 方面创建的约束，确定了一个消费者驱动的契约。若提供者接受了一个消费者驱动的契约，那么它只需保证已有约束仍能得到满足，即可自行改进与修改其服务。



面向服务开发的生命周期

如何将消费者驱动的契约对应到面向服务开发的生命周期上去呢？它们如何帮助解决由联邦的服务资产造成的难题？

启动

在服务开发的第一阶段，消费者驱动的契约很像有关服务的用户故事（user stories for services）。服务只有被消费了才有用；所以，要规定服务的效用，最好的方式就是描述消费者对它们有何期望。通过从角色、功能及利益等方面（参见 Dan North 的《What's in a story》及 Mike Cohn 的《User Stories Applied》）描述高优先级业务结果，用户故事（user stories）有助于我们制订服务关键功能的交付计划。以用户故事（user stories）实现的消费者驱动的契约，描述了一个服务消费者的角色、该消费者所寻求实现的结果或利益、以及它为了实现该结果而对提供者在功能或行为方 面的要求。

这种详细说明具有业务含义的服务行为的方法明显是由外向内的，它跨越业务、机构及技术边界，捕获来自其他开发线及开发生命周期的依赖、期待和约束。实际上，它跟行为驱动的开发（Behaviour-Driven Development，BDD）很 相像，行为驱动的开发也是注重以由外向内的视角来看待业务目标、与那些目标相应的利益、以及为满足那些目标不同参与者需展现出的活动与行为。通过识别出有 助于实现整个业务结果的行为与交互，消费者驱动的契约（就像 BDD 范本一样）指出了一个服务对业务有重大贡献"正好"所需的功能。

关于这些外部化的交互与行为，关键之处在于它们表示的是对业务有意义的东西，它们若不发生，业务活动的某部分就无法继续或完成。在各方之间发生的业 务事件、文档交换和对话过程中，服务符合之处就是系统内在价值显露的地方。消费者驱动的契约反映了一个业务团体、功能或能力为完成其工作而对另一个伙伴的 期望。总之，这些契约提供了一个高层的业务活动视图，它对业务结果有极大的帮助。

消费者驱动的契约解决了一个 SOA 常被问到的难题，即：采用过去的烟囱式方法也能解决当前的业务需求（而且说不定还更简便），那为何还要用服务来支 撑一个活动或流程呢？在一个经过良好构造的服务资产中，真正重要且有用的结果是通过若干对等服务之间的交互实现的，将一个服务与一组分散的业务目标与利益 对应起来并非易事。尽管烟囱式应用常常可直接跟一个具体的利益或结果对应起来，但许多服务具有“流程不可知性”，它们不知道自己在为业务提供什么价值。为 了认识到服务所提供的具体利益与结果，我们需要在其协作上下文之中来理解该服务。这就是消费者驱动的契约发挥作用之处：消费者驱动的契约描述了对服务群落 的协作期望，它通过其更为直观的成对关系、有效地把全体参与者间接感知的价值串连了起来。

构建

在构建阶段，消费者驱动的故事可被转换为可执行的期望。因为它们描述了可接受的结果，所以可以把故事（stories）转换为接受标准，并最终成为 程序断言或测试。要把消费者驱动的契约整合到开发流程中来，破坏性最小的做法是：让提供者团队把消费者驱动的故事转换为程序契约。这无疑提供了一种在服务 开发生命周期各阶段间转移制品的方式，不过这不太符合服务资产的联邦特性。更好的做法是：让负责交付消费者应用与服务的团队来编写他们自己的消费者测试， 并把这些测试交给服务提供者。各个消费者把自己的一个基于测试的消费者契约交给服务提供者——提供者从各消费者处收到的契约集合便构成了它的消费者驱动的 契约。接着，消费者可以参照它们自己的契约进行开发，并相信提供者也将参照同样的期望进行开发。这样做，便可以把契约整合到双方的开发线之中。

为保存服务资产的适当联邦特性，我们要认识到，消费者的期望和契约正是消费者的财产。若我们剥夺消费者对期望的所有权，我们就降低了 SOA 在组织和 架构方面的协同力。

通过把消费者契约的所有权分配给消费者开发线，我们确保了提供者方消费者驱动的契约是由消费者制品直接得到的，而不是提供者自己对消费者期望的理解。以这种方式交换测试，不但把开发与构建过程连结了起来，还降低了出现提供者消费者间理解偏差的风险。

有时，不同消费者要求的契约会彼此冲突。将消费者契约明确化有助于我们及早发现这种冲突。假如冲突相对较小，我们可以跟相关团队商量看是否可以调解——比方说请某个消费者放松一些规则。假如冲突比较严重，可能我们得重新检查我们的设计，以期提取出某些操作与消息，以面向更窄范围的问题。这在一定程度上是个有关判断力的问题——消费者驱动的契约可以指导我们生产出可重用、流程不可知的最小行为集合，但我们仍需确保结果对业务有意义且实现了关注点分离。

最简单的消费者契约，就是对服务提供者与消费者之间交换的消息作断言。对于基于XML的消息来说，消费者可以用诸如XPath或Schematron这样的语言来表达他们的期望，在异质技术环境下这两种语言都比较好用。消费者也可以使用XML Schema和RELAX NG，不过由于这些语言的验证结果常常是"要么对要么错"，因此用XML Schema或RELAX NG编写的消费者契约必须提供扩展点，以说明目前消费者不感兴趣的提供者契约部分——而这某种程度上是有违初衷的。

一个服务暴露什么样的提供者契约，直接受到消费者契约的影响。如果一个服务用XML Schema描述它发送与接收的消息结构，那么它发布的模式（schema）必须符合其消费者的期望。对一个新的服务来说，这可能涉及到提供者与消费者双方的开发团队联合设计消费者断言以及符合那些断言的提供者模式（schema）。对已有的服务来说，消费者可以选择用自定义的断言对模式的一个副本进行标注。对提供者而言，消费者驱动的契约提供了一个明确的开发方向。跟其他单元测试一样，消费者测试可用于驱动开发活动：开发者创建服务行为，以满足测试的期望。也可以把测试添加到一个持续集成环境中，然后拿每个版本对照服务功能进行断言。若消费者以原始XPath或Schematron表达式的形式来提供契约，那么提供者就可以用单元测试框架（如JUnit、NUnit或XMLUnit）对那些它生成的消息执行底层测试。

提供者甚至可以把消费者驱动的契约整合到它自己的运行时活动中，以便当真实世界的

消息未能满足消费者期望时及早提供反馈。如果必须完全遵守所有消费 者契约，那么提供者可能需要对发送管道里的消费者期望进行断言、以验证它发送的消息。根据要验证的契约大小与数目，这可能会对响应速度和吞吐量有影响。或 者，提供者可以实现侦听或对已发送的消息作某种另外的订阅，并对发送管道以外的约束进行断言。

在消费者方面，消费者契约鼓励特别的开发实践。若提供者必须恰好符合消费者驱动的契约，那么消费者必须确保自己也符合对方的期望。消费者不是把提供 者契约整体导入，而是只用它告诉提供者它需要的那部分。这样一来，消费者可以保护自己免受它不关注部分的变化的影响。只用必要的部分意味着不在消费者方面 进行模式验证——相反，假定提供者会遵守被告知的期望，并会根据那些期望发送有效的消息。根据消费者方面反序列化（deserialisation）机制 能够承受多余、丢失或乱序字段的程度，消费者也可以希望避免将收到的消息反序列化为强类型的表示（representations），而是选择用 XPath 提取出消息中所感兴趣的部分。这种策略常被称为 WS-DuckTyping。

基于测试的消费者契约的另一个用处是对厂商套件及 COTS 应用进行评估。采取消费者驱动的方式，我们从“它们将参与的协作”及“系统里其他部分对它 们的期望”的角度来描述我们对这些套件的需求。通过将消费者期望编制成表，我们从外部协作点的角度描述了一个套件实现应该是什么样子；通过用消费者驱动的 契约（也许是以一套自动化测试的形式）来表达这些期望，我们可以判断该应用是否有希望成为的一个优秀资产。

运营

在服务开发团队之间交换的测试，使得服务资产里的较重要的外部协作点凸现了出来。这些测试及它们所代表的相互关联，为那些负责运营服务的人提供了有用的可执行档案。消费者驱动的契约指出了谁依赖于一个服务、以及该依赖的特性是什么，这增强了运营对资产日常行为的洞察。

除了令服务资产之中的关系更加透明，一套以共享测试实现的消费者驱动的契约还增强了关于服务演化将产生影响的反馈。消费者驱动的契约提供了一个已断 言行为的基础，我们可以根据它对一个服务契约的变动产生的可能的影响进行评估。消费者测试构成了一个跨服务开发线的回归测试集的基础，只要提供者能够继续 满足测试集的现有约束，提供者就

可以演化。

消费者驱动的契约可用于规划服务资产的变更，它们对交付机构的版本化策略有帮助。虽然向后/向前兼容性原则依然对版本化服务极为重要，但消费者驱动 的契约有助于根据现有的约束与关系来在大环境中考虑兼容性问题。不必总把必备元素的变化看成是破坏性变化；相反，可以根据它们所违反的消费者契约来找出需 要版本化的变化。若现有消费者未用到某必备消息元素，那么修改或去除该元素就不用被看成是一种破坏性变化。

总结

实施 SOA 给软件开发生命周期带来了新的挑战，这些挑战源自服务资产的联邦特性。一个成功的服务开发能力有赖于对多开发线的洞察和收集关于多开发线 的反馈。以“服务故事（stories for services）”和开发线间交换的测试来使用消费者驱动的契约实现了开发生命周期中各个活动的有机联系和开发线间各个活动的协调，从而让我们部分解决 了目标。消费者驱动的契约支持面向服务的系统的开发与测试，而且支持负责服务生命周期的各方之间的协作。

更多关于消费者驱动的契约的论述，请参看《The ThoughtWorks Anthology》及作者的博客。

关于作者

Ian Robinson 是 ThoughtWorks 公司的首席顾问，他专门研究面向服务及分布式系统的设计与实施交付。他为微软公司编写了用微软技术实现集成模式的指南，并发表了一些关于面向业务的开发方法及分布式系统设计的文章——最近的文章收录在《The ThoughtWorks Anthology》（Pragmatic Programmers，2008）里。他目前正在与人合写一本关于面向 Web（Web-friendly）的企业集成方面的书籍。

原文链接：http://www.infoq.com/cn/articles/consumer-driven-contracts

相关内容：

- 一匙治理即帮助 SOA？

- SOA 治理的业务流程
- SOA 治理：在流程与机动性之间取得平衡
- 克服 SOA 实施过程中的障碍
- 如何开始你的 SOA 治理

[新品推荐]

Rails 2.2 发布：新特性抢鲜

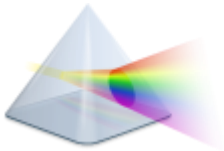
作者 Mirko Stocker 译者 李剑

在两个 RC 版以后，Rails 2.2 最终发布了。虽然从版本号上没有迈出很大一步，但仍然有着很多新特性，比如国际化、线程的安全、文档的完善等等。

原文链接：http://www.infoq.com/cn/news/2008/11/rails-22

跨平台的 Delphi 回归

作者 Jonathan Allen 译者 张龙

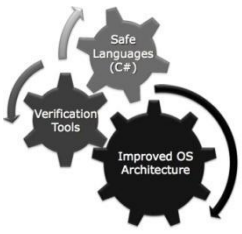


通过将 Visual Studio Shell 作为起始点并使用 Mono 运行时，Embarcadero Technologies 推出了面向.NET 的新版 Delphi——Delphi Prism。该项目主要面向那些想将.NET 生态圈带到 OS X 上的跨平台开发者。

原文链接：http://www.infoq.com/cn/news/2008/11/Delphi-Prism

Singularity：微软的开源操作系统

作者 Jonathan Allen 译者 黄璜



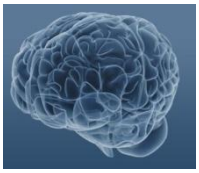
Sigularity 研究开发包的 2.0 发布版现在已经可以通过源代码或者可启动 CD 的方式获得了。Sigularity 操作系统整个依赖于进程之间高度隔离的代码管理方式。一反微软

常规的做法，Singularity 热情招徕补丁，并在 CodePlex 向开发者提供了完整的权限。

原文链接：http://www.infoq.com/cn/news/2008/11/Singularity-Open

Tasktop 1.3：增加对 Firefox 和 Linux 的支持

作者 Ryan Slobojan 译者 宋玮



Tasktop Technologies，创建 Eclipse Mylyn 并领导其开发的公司，11 月 12 日发布了 Tasktop 1.3 版本。InfoQ 采访了 Tasktop 的 CEO 及 Eclipse Mylyn 项目的领导者 Mik Kersten，以了解这一发布的更多信息，以及它给最终用户带来了什么变化。

原文链接：http://www.infoq.com/cn/news/2008/11/tasktop-13

JackBe 发布 Presto Mashup 平台的免费开发版

作者 Srini Penchikala 译者 张龙

企业级 mashup 软件供应商 JackBe 发布了 Presto 企业级 Mashup 平台的免费开发版。这是 Presto 企业级 Mashup 软件的社区版，可用来创建并发布 Mashup 组件。该产品套件还包含一个 Mashup 服务器、一个基于 Eclipse 的 Mashup Studio IDE 及 Mashup 设计器。

原文链接：http://www.infoq.com/cn/news/2008/11/jackbe-presto-dev-edition

Apache Solr：基于 Lucene 的可扩展集群搜索服务器

作者 Ryan Slobojan 译者 崔康

Apache Solr 项目，是一款基于 Apache Lucene 的开源企业搜索服务器，最近发布了 1.3 版。InfoQ 采访了 Solr 的创建者 Yonik Seeley，了解了新版本的更多信息和 Solr 提供给最终用户的功能。

原文链接：http://www.infoq.com/cn/news/2008/11/apache-solr

应用架构指南 2.0 Beta1 发布

作者 Abel Avram 译者 张龙

微软 patterns & practices 组发布了应用架构指南 2.0 Beta1，这是一本讲述在.NET Framework 上设计应用架构时需要遵循的原则、模式及实践的书。其读者定位在解决方案架构师和开发经理。

原文链接：http://www.infoq.com/cn/news/2008/11/App-Architecture-Guide-2.0

JRuby 1.1.5 发布

作者 Werner Schuster 译者 李明（nasi）

JRuby 1.1.5 正式发布，包括大量的 Bug 修正和性能提升，加入了 ruby-ffi 的回调支持，并引入了对 Ruby Gems 1.3.1 的支持。

原文链接：http://www.infoq.com/cn/news/2008/11/jruby-115


1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 试刊号

每月 8 日出版

总编辑：霍泰稳

总编助理：刘申

编辑 :宋玮 朱永光 李剑 胡键 郭晓刚 李明

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

交流群组：
http://groups.google.com/group/infoqchina

商务合作：sales@cn.infoq.com
13911020445 010-84725788

《架构师》月刊由 InfoQ 中文站出品（ www.infoq.com/cn ），Innobook 制作并发布。

更多精彩电子书，请访问 Innobook。

InfoQ中文站
www.infoq.com/cn

innobook
创造 · 共享 · 传播