时刻关注企业软件开发领域的变化与创新

架构师

7月 ARCHITECT

Dan Farino谈MySpace架构

SOAP基于Java消息服务

云彩聚天边

Java创新的未来

Google Wave加速HTML5发展？

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语

我们的新闻理想

很抱歉让大家等了这么久。感谢众多热心读者的督促，让我们终于在这个不算炎热的夏天，发布了《架构师》的创刊号。作为《架构师》第一期，我想和大家分享的是 InfoQ 中文站在新闻方面的价值观，解释我们为什么要和为什么能持续地为读者提供高质量的原创内容。

简而言之，InfoQ 中文站的新闻价值观就是以架构师、项目经理、团队领导者和高级开发人员为目标读者群，努力做好社区的镜子，将技术社区优秀的内容通过 InfoQ 中文站曝光。有一点需要提醒，InfoQ 中文站不做社区的领导者，不将自己的意志强加于读者，所以在新闻中，多是对社区内容的总结之作，让读者自己做最终判断，而非编辑一个观点的陈述。

先解释我们为什么要为读者提供高质量的原创内容。在传统媒体领域，“内容为王”是一个不争的事实。可是在互联网时代，同样的一句话却有了不同的实现方式，虽然还是“内容为王”，但是却不是自己原创的内容，而是复制其他网站的内容，以量取胜。这种方式的好处就是能够比较轻易地获取更多的内容，不足之处是鼓励转载，不鼓励原创，其结果就是创新性的内容愈加稀少。大家都知道，创新的内容和产品对任何一个产业和生态链的作用。

再解释我们为什么能为读者提供高质量的原创内容。浏览 InfoQ 中文站的“关于我们”，我们可以看到目前有近 40 位编辑每天负责而热情地创造着内容，这些编辑均来自于一线的架构师或者项目经理，而且都有着多年的研发经验，这也是为什么我们能够提供高质量原创内容的根本。专业的人做专业的事情，一直是我所欣赏的原则。通过这些对研发有着深刻理解的编辑，不论是翻译还是本地报道，这些内容都更容易让读者找到同感。

总结来看，我们的新闻价值观就是努力做好中国技术社区的镜子，通过专业的编辑为读者提供高质量的内容，这也是 InfoQ 中文站的新闻理想，我们会坚持！

目　录

[人物专访]

DAN FARINO 谈 MYSPACE 架构 8

[热点新闻]

微软决定弃用 ORACLECLIENT 命名空间 17

使用 ADOBE FLEX 进行模型驱动开发 19

SOAP 基于 JAVA 消息服务 21

WINDOWS 7 的图形引擎将超越 VISTA 24

云彩聚天边 26

GOOGLE WAVE 加速 HTML5 发展？ 29

ACTIONSCRIPT 3 网站构建框架 32

主题演讲 ：JAVA 创新的未来 36

RIA 平台：除了 FLEX、SILVERLIGHT，还有 LASZLO 38

SOA 依旧已死? 40

[推荐文章]

融合思想：深入探索 S#ARP 架构 44

使用 HTML5 构建下一代的 WEB FORM 54

使用 SPRING AOP 和 ASPECTJ 编排工作流 61

使用 PERF4J 进行性能分析和监控 77

JAVA 程序员学习 FLEX 和 BLAZEDS 的十三个理由 88

MOCK 不是测试的银弹 95

[新品推荐]

RUBY 1.9.2 的发布计划宣布 103

ZK 发布 3.6.2 版本：性能提升，INCLUDE 模式 103

ECLIPSE GALILEO 发布啦 103

RIP：一个全新的 RUBY 包管理系统 104

OPERA UNITE：超越强大 104

FIREATLAS：ASP.NET AJAX 查看器 104

[封面植物]

异形玉叶金花（白纸宝） 106

本月推荐编辑 107

专家贺词

IT 专业媒体！

—— YouKu 首席架构师，邱丹

祝 InfoQ 健康发展，帮助更多程序员成就梦想！

—— 豆瓣网首席架构师，洪强宁

InfoQ = High Quality

—— 台湾架构大师，高焕堂

InfoQ —— 企业发展原动力！

—— 《走出软件作坊》作者，阿朱

祝 InfoQ 中文站：坚持自己的风格，越办越好！

—— 著名 Blogger，冯大辉（Fenng）

祝 InfoQ 越办越好，蒸蒸日上！

—— Adobe 平台技术经理，马鉴（7yue）

祝 InfoQ 中文站一如既往坚持自己的理想，提供高品质的内容，越办越好！

—— 支付宝首席架构师，程立

牛！

—— UMLChina 首席专家，潘加宇

编辑贺词

既能高瞻远瞩，又可见微知著。

——InfoQ 中文站敏捷社区编辑、《程序员》杂志架构板块责任编辑 郑柯

理论与实践在这里碰撞，愿《架构师》伴随大家一起成长

——InfoQ 中文站敏捷社区编辑、ThoughtWorks 咨询师，金明

英雄出少年，丰姿焕发扫狼烟

豪气干云天，哪怕世道人心险

——InfoQ 中文站敏捷社区首席编辑、ThoughtWorks 咨询师，李剑

武林至尊，宝刀屠龙，倚天不出，谁与争锋

《架构师》帮你找到你自己的倚天屠龙

——InfoQ 中文站 .Net 社区编辑、ThoughtWorks 咨询师，王瑜珩

Ethics of Software Craftsman 中就有一条关於 "We Learn" ,这当然少不了阅读《架构师》

——InfoQ 中文站敏捷社区编辑、Agile Coach @ Odd-e，Steven Mak

张开翅膀，《架构师》与你一起乘风飞向更远的地方。

——InfoQ 中文站架构社区编辑，王丽娟

在学习中进步，在实践中成长。让《架构师》助你进步，伴你成长。

——InfoQ 中文站敏捷社区编辑，张晓庆

《构架师》，给你前沿的资讯，给你创新的技术，更重要的是与你分享原创的经验。

——InfoQ 中文站.NET 社区首席编辑、环境保护实践者，朱永光

穿越浮躁和喧嚣，沏茶剪烛，让《构架师》和你谈谈三分心事。

——InfoQ 中文站敏捷社区编辑，金毅

运筹于架构之中，决胜于代码之外

——InfoQ 中文站 Ruby 社区编辑，杨晨

架海金梁，构厦多材，师法自然

——InfoQ 中文站.NET 社区编辑，张逸

《架构师》，架构你的人生

——InfoQ 中文站 Java 社区编辑，张龙

眼光准，技术就好。做事倍儿快，干嘛嘛强。您瞅准喽，《架构师》杂志！

——InfoQ 中文站.NET 社区编辑，赵劼

聚焦架构不放松，立根原在社区中。

千磨万击还坚劲，任尔东西南北风。

——InfoQ 中文站 Ruby 社区首席编辑，李明

[人物专访]

Dan Farino 谈 MySpace 架构

InfoQ：大家好，我是 Ryan Slobojan，这位是 MySpace 的 Dan Farino。Dan，你能介绍一下你在 MySpace 的工作吗？

Ryan：没问题。我是 MySpace 的首席系统架构师。简单地说，我开发了我们使用的许多后台自定义性能监视和排错工具。当初我刚到那里时，所遇到的问题主要是系统依赖大量的手动配置，大量的手动管理，以及需要大量管理员来创建各种脚本来执行例如重启服务器等工作。你可能要在一个非常简单的问题上花费 30 分钟甚至一个小时的时间。因此，一开始我主要关注于从系统管理的角度出发构建一些自动化工具，希望可以让排错或性能诊断等问题变得简单一些。

InfoQ：你能否想我们解释一下，对如此大型的站点进行调试和排错时遇到了哪些挑战呢？

Ryan：很多时候问题的关键在于，如何从几千台服务器中找到出现问题的那台。所以我开发了一个性能监视系统，可以实时获取整个服务器场中每台机器的 CPU、队列中的请求数以及每秒处理的请求数量等这种类型的信息。这样我们就可以直观地看着屏幕上的一台红色的服务器："喂，我的队列中累积请求了"。然后问题就变成了："好吧，我们已经知道哪台机器有问题了，不过现在该怎么做？" 如果我们获取一个内存快照（memory dump）并发送给微软，可能要过一个星期才会有反应说 "嗨，你们的数据库服务器挂了"。不过我们希望能在两个小时内发现这个问题，所以我们在右键菜单里放了一些简单的工具，可以让一些运维人员——他们不是开发人员——发现说 "我看到几百个线程在数据库访问时阻塞了，我估计是数据库的问题"。我们关注于为出错的服务器提供高度可视化的方案，还开发了一些工具，可以让一些技术不那么强的管理员快速发现问题。

InfoQ：很有意思。你能否从技术角度大致描述一下 MySpace 的架构呢？

Ryan：当然。可能从前端开始谈这个问题最为合适。我们有大约三千台 Windows 2003 IIS 6 服务器。这些代码运行在.NET 2.0 上，少部分是.NET 3.5。不同的.NET 的版本会造成各种麻烦问题，不过前端还是运行了.NET 2.0 和 3.5。我们使用经过调节的 SQL Server 作为后端存储。不过数据库查询的性能不足以应对站点对伸缩性的要求，因此我们开发了一个自定义的缓存组件。这是个简单的对象存储部件，通过自定义的 socket 层进行通信，全部存储在 64 位.NET 2.0 机器中的非托管内存中。因为我们最早遇到的问题之一便是.NET 的垃圾收集器。

就算在 64 位平台上，回收和压缩数以亿计的对象还是会造成较为明显的延迟。所以我们面向伸缩性的问题之一便是“必须自己编写非托管的内存存储”。我们在存储的前端使用.NET 进行通讯，并与这一存储层进行交互——我们称之为缓存层，这大大降低了数据库的压力。现在我们的数据库从 SQL 2000 升级至 64 位 SQL 2005，这样内存中可以存放更多的数据，这带来了很大的性能提升。起初我们使用 Windows 2000 中的 CodeFusion 5 以及少量的数据库。我们最早在数据库伸缩性方面做的努力是进行了纵向划分——或横向划分，我不知道你们把这叫做什么方向——把每 100 万个用户放在不同的数据库中。

这么做提高了伸缩性，也让我们能够轻松地添加硬件，并对错误进行隔离。如果一个数据库当机了，只有一小部分用户会在我们处理这个问题之前得到错误信息或正在维护的消息。我们在 Linux 上构建了一个自定义的分布式文件系统，用于存放用户上传的媒体内容，并作负载均衡。所有的视频或 mp3 等内容都放在这个自定义的 DFS 层上，实现了跨数据中心的冗余，并通过 HTTP 直接从磁盘中获取数据。

InfoQ：你们在扩大网站规模时遇到了哪些挑战？你提到，一开始你们使用了 CodeFusion 服务器，不过现在使用了数以千计的 IIS 服务器。你们是怎么进行切换呢？你们扩大网站规模时只是通过增加服务器数量吗？

Ryan：许多服务器用来构建中心缓存了，这可以大大降低数据库服务器的压力。还有，如果你使用大量后端数据库，就可能会遇到一个问题：如果其中一个数据库服务器当机了，那么它会很快拒绝来自 Web 服务器的请求，问题不大。不过，假如只是这个后端服务器变慢了，试想某台 Web 服务器正在为大量不同的用户提供服务，很可能其中一个缓慢的请求就会阻塞在这个坏的数据库上。

不过此时其它请求依旧工作正常，不过迟早就会有另一个请求阻塞在坏数据库上。可能过了十几秒钟之后 Web 服务器上所有的线程就被这个数据库阻塞了。所以我到那里之后第一个架构的东西便是错误隔离系统，它运行在每台 Web 服务器上说："我只允许同时向同一范围的服务器发起 x 个请求"。这避免了单个坏服务器让整个站点停止响应。原本我们会在扩展的同时可能会遇到越来越糟糕的情况，因为添加了更多可能会造成单点失败的地方，整个站点停止响应的可能性也提高了。现在我们在 IIS 上部署了错误隔离模块，这样增加了网站的正常运行时间。再加上缓存层的功效，正常运行时间进一步增加了。

InfoQ：听上去你们做得很多事情都是在现有工具的基础上构建扩展，那么在微软平台上这么做的感觉如何？

Ryan：感觉非常好。我们使用微软的 Powershell。它早先的代码名(code name)是 Monad，提到这个名称的原因是我们从 Monad beta 3 就开始在生产环境中使用它了。它出现的正是时候，因为那段时间我们几乎已经把所有推荐的 Windows 管理技术，例如 VBScript、命令文件等，都用到极限了。

那些东西在本机上工作的很好，如果你想要看另一台机器上的注册表配置也不错。不过如果要在数千台服务器运行命令的话，你就会遇到瓶颈。首先你只能每次访问一台服务器，其次，如果遇到了一台当机的服务器，那么整个工作就停止了。为了解决这个问题，我自己写了一个简单的远程服务层，于是问题就又变成了"我怎么样才能把它做的通用呢？"我们希望可以轻松地对一系列的服务器进行操作，比如执行一条命令，获得一些结果，进行处理，然后可能运行另外一条命令——也就是说，这些操作形成了一个管道。

就在这时候 PowerShell 出现了，帮了很大的忙。PowerShell 完全用.NET 编写，它在同一个进程里运行所有的命令，这个好处在于命令 A 可以输出各种东西，而不仅仅是字符串，并让命令 B 可以接着进行处理。它们之间可以顺着管道传递任意的.NET 对象，这样就形成了一种非常强大的编程模型，让那些只愿意坐下来写一点点命令的管理员们也乐于使用。于是我们就决定使用那些命令，例如我对 MySpace 说"VIP"，意思运行特定功能的一组虚拟 IP。如果我说"Profile VIP"，意思就是指运行 profiles.myspace.com 的所有几百台服务器，然后可以对它们统一进行处理。于是我们构造了一个叫做"Get VIP"的命令。

我们使用 RunAgent 命令的管道，可以并行地在远程机器上执行任何命令，并把任何对象放进管道里。例如你可以用 PowerShell 的语法来描述“告诉我这个东西是 true 还是 false，某个文件是不是存在”，然后把结果传递到另一个管道中，不断传递，这样就可以快速地处理一些特定的管理任务。人们在写脚本的时候就不需要担心网络是否失败，也不用担心多线程执行的状况，因为我已经把这些问题处理掉了。我抽象了网络连接，抽象了并行特性，这样可以把一些本来要花 30 分钟甚至 1 小时的工作用大约 5 秒钟就完成了，更可靠，可容易控制，更容易进行帐户管理或日志记录。

InfoQ：真厉害。我想你之前一定也听到过这样的问题，为什么你们会选择使用微软的技术而不是其他一些常见的选择呢？

Ryan：嗯，在我到来之前其实就已经做出这个决定了，我刚到的时候系统运行在 CodeFusion 平台上，有大约两千万用户。我们有非常非常优秀的.NET 开发人员，不过我不确定我们到底是专门找了.NET 方面的人才，还是找了最好的开发人员，然后他们正好谈到“.NET 是个不错的东西，我们就试试看吧”。不过我认为，事实上我们很早就已经使用微软平台了，因此延续使用.NET 是一件很自然的事情。我也不是一个搞 Java 的人，我的背景和微软技术比较接近，微软也给了我们不少帮助，让我们在使用.NET 这个相对较新的技术时度过了一个个难关，所以它给我的感觉还是相当不错的。

InfoQ：刚才你提到，你们混合使用了.NET 2.0 和.NET 3.5？它们配合的怎么样？有没有计划完全迁移到 3.5 上？

Ryan：微软现在在版本号的使用上有些疯狂。我早先看了看他们的路线图，其实 3.5 不过是在 2.0 的基础上增加了一点扩展。所以你安装了 3.5 之后自然就有了 2.0 的运行基础。这种做法有点奇怪，不过我觉得 3.5 中的有些东西似乎还不错，例如 Windows Communication Foundation 提供了很方便的 web service 功能，还包括了事务控制等特性，让我们放弃了老旧的 remoting 和 web service 编程模型。我们对 WCF 还没有使用太久，不过看上去这些东西很令人兴奋，它可以让我们摆脱不少已经存在多年的负担，以前我们不得不编写自己的网络通信类库，只是因为 remoting 和 web service 功能上存在一些小毛病。

InfoQ：所以说，整个网站是建立在 2.0 和 3.5 之上的，你们使用的工具基于 PowerShell。

不过还有一点，你们整个系统里有没有配合使用的调试工具呢？

Ryan：这些工具里我最喜欢的是一个叫做“Profiler”的工具。它使用 C++编写，基于微软 CLR 侦测（profiling）接口，所以我们只要说“OK，我们的堆栈转储接口可以展示一些实时的信息，比如当前各个部件都在做什么事情”，然后侦测接口就能说“好吧，我会每 10 秒对这个线程进行一次检查，我会从头至尾监测一遍，我会告诉你每个调用花费了多少时间，告诉你每个异常的信息，每次内存分配，锁的竞争状况，还有各种依靠调试器查看静态状态等做法获取不到的信息”。

这个 Profiler 的作用不光是在错误的情况下告诉我们发生了什么，还可以在正常状况下告诉我们那些请求对系统产生了什么样的影响。我们的系统中有大量不同的人写的各式各样的模块，所以对我这样的人来说，就算已经对系统有所了解了，查看那些跟踪记录有时候也会有新发现，比如“我不知道我们居然做了这些事情”。它可能是我们开发的技术最复杂的工具，这也是仅有的几个不使用.NET 开发的工具之一，这是因为你无法编写.NET 代码来监测.NET 代码。这里我们用了很多 C++，有意思的是它利用了微软研究院的 Detours 类库，在技术上类似于他们用来偷偷摸摸为内核打补丁的方法：把现有代码导向别处，再调用回来，这样程序并不知道自己已经被打上了补丁了。我们也用这种方式来获得一些微软的接口无法提供但有时候特别有用的一些信息。

InfoQ：你提到了 C++，我记得你之前也谈到过 VB 和 VBScript？

Ryan：大约两年前我们使用 VBScript。现在我们使用 C#作为.NET 开发语言。

InfoQ：在 MySpace 内部使用什么语言呢？

Ryan：我必须提到 CLR 本身基本上完全是由 C#编写的。我们内部也有一些“先驱”在尝试一些诸如 F#之类的东西。F#来自微软研究院，看上去是个相当不错的东西。我在不少工具里嵌入了 IronPython 脚本，因为我觉得就配置工作而言，它可以带来更多控制能力。与钩选几百个选择框相比，我现在只要几句 IronPython 脚本就能完成工作了。就总体来说，我们用 C#做前台和后台开发，在各种必要的时候就会用到 IronPython 和 PowerShell。

InfoQ：你们打算迁移到 LINQ 等各种.NET 3.5 新特性上吗？

Ryan：事实上 LINQ 给我的感觉可谓一天比一天深刻。我找不到任何理由不让所有的服务器升级，或者不让开发人员使用它。LINQ 看上去是个非常巧妙的技术，对于像我这种喜欢写 SQL 查询的人来说，可以用 LINQ 来处理 XML，各种对象或内存中的数据，这种感觉实在是太酷了。可能有一天我们还可以使用自定义的 Provider，谁知道呢？真是个不错的东西，我希望很快就能在服务器上看到 LINQ 的使用。

InfoQ：对于最近发布的 MySpace 开发者工具，你们有没有打算让开发人员可以使用一些不同的语言，例如 C#？还是让他们继续基于 JavaScript 或 VBScript 这样的脚本语言进行开发？

Ryan：其实我还没有确定未来的打算。我知道刚发布的开发者平台是网站上的小部件（widget），可惜目前只支持 JavaScript。也许您自己的富客户端应用程序可以调用系统中完整的 API，不过我实在无法确定这方面内容。

InfoQ：对于 MySpace 这样的大型网站来说，如果你遇到了问题，例如服务器的负载到达了峰值，有没有办法可以创建一个补丁让问题直接消失，还是只能一点一点地改变，把问题分成几个部分依次解决？

Ryan：这个问题很好，我觉得要从两方面来看这个问题。一是发现问题，二是解决问题。很多时候我们会发现“OK，站点的请求排队了，我们该怎么办？”我们其实不知道为什么会发生这个情况，然后只是增加了几台服务器。好，似乎问题有所改善，不过其实你并不知道这时是否应该增加服务器，有可能只是后台的数据库刚好在进行备份。这不仅在于你不知道哪里出现了错误，还有目前的做法可能在今后就不起作用了。所以一开始的困难在于，我们要如何开始识别这些问题，我们该如何让前端网络操作中心（NOC，Network Operation Center）的人有能力指出这些问题。一旦这个困难解决了，那么他们就可以比较容易确定该向谁汇报问题。

当然，有可能这的确是一个代码上的问题，我们努力希望让中间层或后端的代码与前台兼容。如果我们发布了一个东西，结果在 QA 时正常，测试时正常，但是发布到网站之后却有问题了但又不清楚是什么原因，那么最简单的方法是回滚这次发布或是禁用这个功能。一般来说，使用了正确的工具包之后，很少会出现找不出问题所在的情况。一开始我们的工具

包比现在要小得多，不过如果每天都会发现类似的问题，那么我们会说“好，那么我们用调试器检查一下”，第二天说“我再用调试器看看这个问题”，而第三天：“算了，我要自己写一个工具，现在的那些派不上用场”。于是我们的工具包越来越完整，这样快速发现和诊断问题的能力也就提高了。

InfoQ：整个排错的过程是什么样的呢？系统是如何适应伸缩要求，这一点上理论和产品上有没有什么区别呢？它又是如何影响系统架构的呢？

Ryan：嗯，目前系统架构模型主要分为三层：前端，缓存层和数据库。所以要大幅度改进其伸缩性会是件困难重重的事情。我们一开始会说“我们现在的这条数据库查询不太好，那么我们试试看简单的修复一下，把它放到缓存中去”。好吧，这没啥效果，那么我们可能要建立另一套系统了。我们就开始琢磨这个问题，我们有不少这样的系统，如果性能出现问题了那么我们可以对它们分别进行优化，比如把这部分移出数据库，放到磁盘上，或者就放在内存中。如果某个地方出现了问题，那么我们会开发一个新的系统。这在 MySpace 中并不多见，因为我们在一开始在水平分割上的考虑十分有效。不过当你打算获得更多 9【译注：即 99.99.....%】的可用时间时，就会在决策上有很大的改变。

InfoQ：这可能是个有 2-3 年历史的老问题了，不过在提及“.NET 无法伸缩”的问题时，你会怎么答复呢？

Ryan：我觉得不存在这方面问题，关键在于你有没有使用正确工具，是否有这方面的专家。事实上我觉得现在.NET 已经是一个非常成熟的平台了。显然 Java 在这方面领先于.NET，不过我想我们还是互联网上最大的.NET 站点。我不知道你会把我们的可用时间和性能与同样规模的 Java 站点进行什么样的比较，我们遇到的各种问题都不是.NET 平台本身造成的。可能是我们自己的 bug，可能是硬件问题，但是我没有真正遇到过.NET 的问题，除了在垃圾收集方面我可能会说“这的确很难进行伸缩”。

不过.NET 上最好的地方莫过于它的可扩展性了，所以如果垃圾收集器在你 16GB 的机器上进行内存管理时表现不好，那么你可以使用 Berkeley DB 或你自己的非托管存储进行替换。要享受.NET 的优势也不必把自己完全局限在.NET 中。我想说.NET 有良好的伸缩性，而且我们的规模也会越来越大。你不妨过几年再来问我这个问题，看看我会怎么说。

InfoQ：你们在服务器或伸缩性方面有没有使用其他的微软企业产品？

Ryan：应该说，有新东西出来的时候我们就会试试看。例如 Enterprise Application Block、Remoting 和 Web Service 我们都尝试过。如果你真想做一个大规模的应用程序，那么这就是我的建议：一般来说，我们最后会使用自己的实现，因为我们不需要如此通用的解决方案，我们真正需要的是性能。我想说的是，我们尝试了某个东西之后就从中学到些东西，然后可能就会把其中有价值的地方给剥离出来，再为了达到我们的性能或伸缩性需求重新写一个。我不觉得微软会在我们这种规模的应用程序中进行测试，所以好像是我们在为他们测试一样。所以，如果有些东西可以在几十台服务规模的程序中工作正常，但是无法满足 MySpace 的伸缩性要求，我觉得这是一件稀松平常的事情。

InfoQ：你能否给出一个例子，关于对.NET 或微软告诉你应该做的东西，却反其道而行之？

Ryan：我觉得我写的 Profiler 算是一个吧。如果他们不会发现的话，我会使用很多肯定违反服务器担保条例的东西，例如因为性能为 CLR 代码打补丁。我们看了很多 C++运行时的代码，也一直用 Reflector 来查看内部情况，可能有时候还会在产品代码中用到反编译来修补一些我们觉得微软做的不好的东西。不过我们尽可能避免这种做法，所以我举不出一个真正的例子来说明我们在什么地方反其道而行。不过每次遇到这些小问题时，我们会说“不如稍微修改一下，希望能够有用”，而不是“好吧，我们重新用一个新的技术”。

InfoQ：你能多谈一写缓存层的东西吗？我对你处理一些常见的缓存问题的方法很感兴趣，例如更新之类的事情。

Ryan：我们正好在处理这方面的问题。目前，缓存层既不是 write through 也不是 read through 的。基本上 web 服务器做的事情会分两步走。首先检查缓存，如果没有东西，那么 web 服务器会从数据库里取出对象并序列化，发送给用户页面，然后异步地提交给缓存，然后下一次再重复这个过程。我相信在某些时候我们会使用一个更加传统的三层模型，这样 web 服务器不会直接连接数据库，不过目前我们仍然基于简单的两层：web 服务器和数据库，而缓存层只是简单的附属物。当以后规模越来越大时，我们就会着手把访问转移到缓存中，最终会让 web 服务器连接缓存服务器，而不是数据库。

不过目前看来，对象存储工作得相当不错，我们也在这方面思考了很多。它只是用来存储对象，它并不知道保存了什么或者那些东西从哪里来，这对性能可能有些好处，我不确定。不过从几年前我们需要缓存的时候，就把它设计成可以轻松增加的服务。我们现在有 400 台运行飞快的服务器，如果有东西变慢了，那么我们就会进行升级。没什么特别的。

本文由赵劼翻译，朱永光审校。

观看完整视频: http://www.infoq.com/cn/interviews/MySpace-Architecture-Dan-Farino-cn

相关内容：

- MacRuby 放弃 GIL，实现并发线程
- Twitter，架构的变迁
- 使用 Perf4J 进行性能分析和监控
- 如何对企业数据进行架构设计？
- 与冯大辉谈数据库架构

[热点新闻]

微软决定弃用 OracleClient 命名空间

作者 Al Tenhundfeld 译者 王波

微软宣布.NET 4.0 以后的版本将弃用 System.Data.OracleClient。该命名空间的类将会在.NET 4.0 中标识为弃用并在未来发布的版本中移除。OracleClient 是微软针对 Oracle 开发的 ADO.NET 提供程序并且作为.NET Framework 类库的一部分。

这项决定引起了使用 Oracle 的.NET 社区开发人员之间的争论。与此同时许多企业级.NET 应用程序使用第三方 Oracle 提供程序，System.Data.OracleClient 通常会在小型应用程序中使用，它与微软开发工具的集成性非常好。

微软坚称这项决定是经过研究和深思熟虑之后做出的：

> 经过慎重地考虑各方的观点以及和我们的客户、合作伙伴和 MVP 沟通之后，我们决定把 OracleClient 从 ADO.NET 路线图中移除。

这项决定有部分原因是基于目前 Oracle 的第三方 ADO.NET 数据提供程序的广泛应用和不断完善。流行的 Oracle 提供程序在性能上有着显著提升以及不同版本的兼容性得到完善：

- Oracle 发布针对.NET 的免费 Oracle 数据提供程序（ODP.NET） ODP.NET 11g 与所有版本的 Oracle 数据库兼容（包括 9.2 版本），在同一个操作系统上可安装多个版本。
- DataDirect ADO.NET 针对 Oracle 数据提供程序 100%托管代码，免费试用
- Devart 公司开发的 Oracle 数据提供程序 dotConnect 正式名称是 OraDirect.NET，免费使用，支持 Entity Framework、LINQ to Oracle 和 ASP.NET 数据源提供程序模块

尽管遭到社区成员的反对，微软并没有改变该决定的意思，任何人都不应抱有幻想。微软官方的解释是“大部分第三方提供程序都提供同样的性能并能够满足顾客的需求”。因此，不值得对 OracleClient 进行投入，以到达第三方提供者的水平，这样可让微软集中资源在 ADO.NET 的开发上。

一些人把这项举动称作针对 Oracle 以提高.NET 开发门槛的不正当打击，但也有很多人对这项决定持谨慎乐观的态度，也许微软在重新审视对其它公司产品的偏见。例如，ALT.NET 开发人员普遍抱怨微软毫无原因就重新发明轮子，在已经有很好的替代产品的情况下重新开发自己的版本。看看，MSTest 和 NUnit，或者 Entity Framework 和 NHibernate 就知道了。最近官方支持 jQuery 和提供 ASP.NET MVC 源代码的背景下，这项决定可解释为微软进一步承认它无需控制所有的技术。他们可以依赖合作伙伴和社区来提供部分支持。

再者，播客 Connected Show 就这次的改动，进行了节目讨论并分析为什么这不是一场灾难。

原文链接: http://www.infoq.com/cn/news/2009/06/oracleclient_deprecated

相关内容：

- 微软发表.NET RIA Services 的路线图
- Entity Framework 4.0 Beta 新特性
- ADO.NET 数据服务将提供离线功能
- SQL Server 中的表值型参数
- 迈向 Data 2.0——在客户端操作 ADO.NET 数据服务

[热点新闻]

使用 Adobe Flex 进行模型驱动开发

作者 Jon Rose 译者 张龙

近日 Adobe 发布了 Adobe LiveCycle Data Services 3（LCDS）beta 版，这也是 Adobe Flex 生态圈的一个重要事件。LCDS 可以看作是 Adobe 开源的数据服务产品 Blaze Data Services 的商业版，它包含了高级的数据管理和消息处理特性，而这些特性在开发企业级 Flex 应用时是不可或缺的。

Adobe 的 Damon Cooper 专门为此次发布撰写了博文：

> 该公共预览版是 LiveCycle Data Services 工程团队近 16 个月的努力成果，在最终版发布前我们还有很多事情要做，不久的将来你会看到此次发布的深远影响，与此同时我们还会根据社区的反馈不断调整。

此次发布中最具革命性的一个增强就是模型驱动开发与部署。LCDS 产品经理 Anil Channappa 在最近的一篇文章中与广大读者分享了该新的模型特性：

> Adobe 开发了一个名为 Fiber 的新技术，以此将模型驱动开发带给广大的 Flex 开发者。借助于 Fiber，开发者首先创建好应用模型，然后以此为基础开发 Flex 用户界面以及服务器端业务逻辑。现在我们可以通过 Flash Builder 4 以及 LiveCycle Data Services 3 又好又快地开发 Flex 应用了。
>
> LiveCycle Data Services 3 beta 版提供了一个 Fiber 模型运行时，这样我们就可以轻松实现数据持久化了。借助于 Fiber，数据管理得到了内在的支持，同时也无需开发者创建客户化的程序或是使用复杂的特定于 LiveCycle Data Services 的配置了。很多时候我们无需编写任何 Java 或是 Flex 代码就能创建好功能性的应用骨架或是原型。

该新的模型特性向开发者提供了从前端到后端的完整解决方案，包括构建产品应用所需的工具。最主要的工具就是 Flash Builder 4 IDE 的 LCDS 插件，最近该插件发布了 beta 版，内置了建模工具以及强大的服务集成以开发客户端的 Flex 应用。该发布还支持可靠通信及数据节流（ data throttling ）。请查看 Anil 的完整文章以了解关于此次发布的更多信息。同时 Adobe 还发布了两个视频（ 1 ，2 ），对相关新特性进行了介绍。

原文链接：http://www.infoq.com/cn/news/2009/06/model-driven-dev-with-flex

相关内容：

- 将架构作为语言：一个故事
- SOA 编程模型
- 采用模式和泛型技术为应用增加策略控制
- 与林昊一起探讨 OSGi
- 建模语言应该是什么样子？UML 又处于何种位置？

[热点新闻]

SOAP 基于 Java 消息服务

作者 Boris Lublinsky 译者 黄璜

W3C 刚刚发布了推荐候选 SOAP 基于 Java 消息服务 1.0 详细描述了 SOAP(SOAP 1.1 和 SOAP 1.2)应当如何绑定到一个支持 Java 消息服务(JMS)的消息系统。这一推荐的主要目的是：

> ...是在于保证不同的 Web 服务供应商实现之间互操作性。同时还应当支持消息者实现他们自己的 Web 服务来作为其基础设施的一部分，并让它们与供应商提供的 Web 服务进行互操作。

这一规范由两个主要部分组成：SOAP/JMS 底层协议绑定-规范一致实现所要求的-以及 WSDL(1.1 和 2.0)用于 SOAP/JMS 绑定-规范一致实现的可选项。

SOAP 底层传输绑定定义了使用 Java 消息服务发送和接收 SOAP 消息的规则。SOAP/JMS 底层协议绑定包括了哪些 JMS 特性/属性在 SOAP 格式层次是“可见的”以及需要哪些 JMS 调用来支持它们；消息内容，包括属性和报头，比如优先级，soapAction，targetService 等等，是如何被处理的；以及 JMS 服务的连接细节。

这一规范将连接到服务目的点一般化了，使用 soapjms:lookupVariant，详细描述了用于查找指定目的名称的技术。该规范所要求的唯一一个 lookupVariant 是基于 JNDI 的 -jms-variat:jndi。其余的连接属性-soapjms:destinationName，soapjms:jndiConnectionFactoryName，soapjms:jndiInitialContextFactory，soapjms:jndiURL 以及 soapjms:jndiContextParameterType-都是为基于 JNDI 的 lookupVariant 作支持的。对于非 JNDI 的查找，作出合适的映射就要取决于其实现者了。

该规范同时还引入了一个 SOAP 参数集合，允许将 JMS 特定的消息报头暴露出来。这包括了 soapjms:deliveryMode，soapjms:timeToLive，soapjms:priority，soapjms:replyToName 以

及 soapjms:topicReplyToName。最后两个看似完全不合时宜 ,因为它们是属于 WS-Addressing，而不属于 SOAP 基本定义。

额外的灵活性来自于对 SOAP 特定的 JMS 消息属性的定义：

- soapjms:targetService -可以被一个目标实现用于分发服务请求。这可以支持在单个队列上复用多个服务访问。
- soapjms:bindingVersion - 被用于指示 SOAP 绑定所使用的版本。
- soapjms:contentType - 允许指明主要消息负载的 MIME 类型。同时它还认定了该消息负载是使用 SOAP1.1，SOAP1.2，SOAP 带附件消息，还是 MTOM 来作为其主要的负载。
- soapjms:soapAction - 它的使用跟 SOAP/HTTP 中的使用一模一样。
- soapjms:isFault - 被用于指明一个消息是错误消息。
- soapjms:requestURI - 用于指明服务的 JMS URI。该规范将这一 URI 定义为一个带有查询参数的 JMS 目的地 URI，表示目的地与参数属性。

该规范同时还讨论了 JMS 消息类型，安全考虑以及消息交换模式。

该规范的第二部分描述了 WSDL 将如何被用来指明使用以及控制 JMS 绑定的操作。

就 WSDL1.1 来说，该规范定义了如下的扩展：

- wsdl11soap11:binding 元素的传输属性获取一个反应了 JMS 传输的新 URL。
- 允许使用 SOAPAction 报头，尽管它是 WSDL 规范显式禁止的。
- 定义了如何设置各种属性来控制绑定的行为(连接参数，运行时设定)。
- 使用 JMS URI 来定位服务。

就 WSDL2.0 来说，有如下的扩展：

- 绑定元素的 wsoap:protocol 属性获取一个反映 JMS 传输的新的 URL。
- 定义了如何设置各种属性来控制绑定的行为(连接参数，运行时设定)。

- 使用 JMS URI 来定位服务。

尽管 SOAP 基于 HTTP 允许可操作的消息实现，在很多需要达到零宕机时间和零数据丢失的任务关键系统里，消息是一个更合适的底层传输支撑。因此，许多 Web 服务实现，不管是在 Java 领域还是.NET 领域，都提供了私有的基于消息的 SOAP 支持。从这个角度来看，一个 SOAP 绑定消息的规范已经缺失太久了。

另一方面，这一将 SOAP 映射到 JMS 的规范看上去却并没有为其目标提供一个解决方案“...是在于保证不同的 Web 服务供应商实现之间互操作性”。首先，它将 SOAP 绑定到 JMS，仅此一项就排除了.NET 和/或大型机实现。同时，它将 SOAP 拴在了 JNDI 上，这就算在单个企业里都有可能造成互操作性的问题-典型的是多个不同的应用服务器集群会有它们自己的 JNDI。最后，除非使用了 高级消息队列协议(AMQP)，否则不同供应商的消息实现之间不会有线上的互操作性。

原文链接：http://www.infoq.com/cn/news/2009/06/SOAPJMS

相关内容：

- SOAP 协议栈是令人尴尬的失败？
- SAAJ - - 理论上很美，实践中受阻?
- 李锟谈 Fielding 博士 REST 论文中文版发布
- 表现层模式新探
- Java、.NET，为什么不合二为一？

[热点新闻]

Windows 7 的图形引擎将超越 Vista

作者 Abel Avram 译者 王波

Windows 7 图形引擎修改了 Vista 引入的 DWM 工作方式 ,同时它还带来了新的 API、D2D 和 DWrite、新的 Direct3D 11 并能更好的处理多路输出设备。

微软的桌面和图形小组高级架构师 David Blythe 在 Channel 9 的采访中解释了，什么是桌面窗口管理器（DWM）及其如何运作。应用程序不会直接在屏幕上绘图而是在内存的位图上进行，DWM 负责把图像绘制到屏幕。这项技术的使用是为了在图形出现在屏幕之前对其应用各种变换。这就是动画任务栏缩略图如何创建的机理或者说桌面缩放是如何运作的。

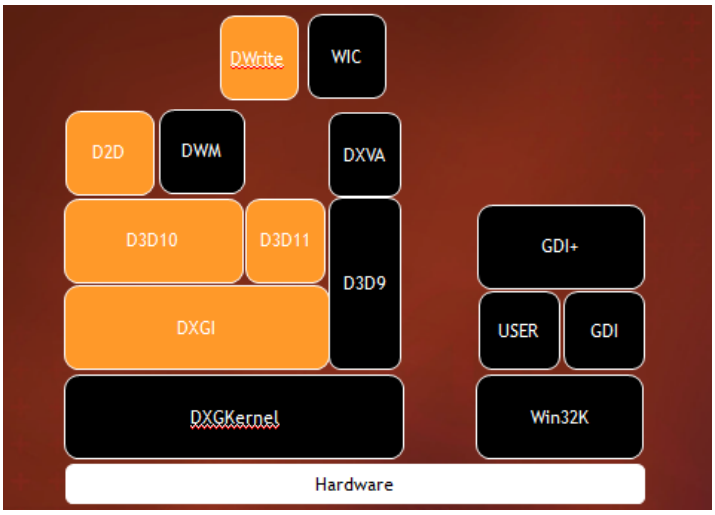
Vista 的 DWM 最大的问题是它的效率问题。直到 Vista 为止，应用程序使用 GDI 在屏幕上进行绘图，同时 GDI 使用硬件图形加速器来完成实际的工作。在 Vista 中，GDI 在内存中进行绘图，DWM 使用已绘制好的位图并用 GPU 把它绘制到屏幕上。这会导致有两份同样的图像。Windows 7 中的改进意味着 GDI 将直接在图形内存中进行绘制，这将在 GDI 中再次引入最小的硬件加速层。这项改变会极大的降低内存的消耗和让图形呈现速度加快。

两项新的 DirectX API 是 Direct2D 和 DirectWrite。根据 MSDN 的资料，Direct2D 是：

> 硬件加速的即时模式 2 维图形 API，它在 2 维几何图形、位图和文本方面有着较高的性能和质量。Direct2D API 是设计与 GDI、GDI+和 Direct3D 之间进行互操作的。

而 DirectWrite 则是：

> DirectWrite 提供高质量的文本呈现、具有独立分辨率的轮廓字体、完整的 Unicode 文本以及布局支持。在使用 Direct2D 的时候，DirectWrite 是硬件加速的。



Windows 7 带的 Direct3D 11 是 D3D10 的超集，它还可运行在 D3D9 和 D3D10.x 硬件之上。D3D11 展现的一些新特性，如下所示：

- 分割：在运行时，在底层细节的多边形模型的基础上，提升直观的多边形数量
- 多线程呈现：利用多核 CPU 从不同的线程来呈现相同的 Direct3D 设备对象
- 计算着色器 - 公开着色器管道的非图形任务，例如，流式处理和物理加速（类似于 NVIDIA 的 CUDA 实现的效果），包括 HLSLShader Model 5

在 Vista 中运行得不太顺畅的一项特性已经在 Windows 7 中得到解决，该特性可管理多个显示器和类似于放映机的显示设备。为了处理大量的潜在输出设备配置，GDI API 进行了扩展，对于所需的输出配置可以简单地通过按 Win + P 键来进行选择。

原文链接：http://www.infoq.com/cn/news/2009/05/should-devs-learn-aspnetmvc

相关内容：

- Comet 框架 Atmosphere 发布 0.2 版本：支持注解和新容器等
- SharePoint 对象模型性能考量
- 如何对企业数据进行架构设计？
- 与冯大辉谈数据库架构
- 王雷谈开源以及新兴市场计划

[热点新闻]

云彩聚天边

作者 Abel Avram 译者 王丽娟

Adobe 准备好了 Acrobat.com，IBM 已提出自己的云服务，Oracle 也将利用 Sun Cloud 加入他们的行列。云计算是企业 IT 的未来，如果过去对此还有疑问的话，那现在这一点就显而易见了。

Acrobat.com 已经结束了其公开测试阶段，推出了 Premium Basic 和 Premium Plus 两种收费服务，而免费版本仍然可用。Basic 和 Plus 产品之间的主要区别在于能创建 PDF 文件的个数，每月 10 个还是无个数限制。此外，Basic 的网络会议功能允许 5 个人，而 Plus 则允许 20 人。继五月份宣布了 Acrobat.com Presentations 之后，Adobe 又发布了在线电子表格程序 Acrobat.com Tables。这两个应用都来自于 Acrobat.com 实验室，所以它们还不是很完善。Acrobat.com 目前提供的服务有：

- Buzzword：文档创建/编辑器。
- ConnectNow：创建虚拟会议的工具，虚拟会议具备共享屏幕、注释、白板、文件、聊天和视频的功能。
- Create PDF：创建、共享 PDF 文件。
- Presentations：在线协作创建演示文稿。
- Tables：在线协作创建电子表格。

很显然，Adobe 希望在在线协作工具市场里与 Google Docs 和 Zoho 相竞争。Google 的 Docs 面向所有人，Acrobat.com 则定位于企业。Adobe 正在利用他们的 RIA 平台提供完全用 Flash 构建的在线协作体验，有精美的图形界面。其计划是将目前的产品集成到构建在 AIR

之上的一个界面中，并在秋季添加对 Smartphone 和 iPhone 的支持。

Adobe 说，Acrobat.com 自一年前推出以来，已经聚集了五百万订户，其中百分之七八十是企业。目前，每天都在使用的账户有几十万个。

IBM 在进行了五年研发、投资了百亿美元之后，于近日发布了他们的 Smart Business Cloud 组合，其中包括：

- IBM 云上的 IBM Smart Business 标准化服务；
- 防火墙之后的 Smart Business 私有云服务，这由 IBM 创建（由 IBM 或客户运行）；
- IBM CloudBurst 工作负荷优化系统，它针对的是那些想用预先集成的软硬件构建自己的云的客户。

该产品用来处理软件开发和测试，还有虚拟化桌面。云的开发和测试服务有三种形式：

- IBM 云上的 Smart Business 开发和测试 （预览）——IBM 软件的云交付服务 ,通过利用 IBM 的安全、针对应用生命周期管理的可伸缩的云交付模型，能让组织释放资源，以快速回收他们在软件上的投资。
- IBM Smart Business 测试云——处于客户防火墙之后的私有云服务 ,由 IBM 构建、运行。
- IBM CloudBurst——该家族预先集成好了软硬件、虚拟化和网络，内置一个先进的服务管理系统。

虚拟化桌面服务通过 IBM 或客户的云提供：

- IBM 云上的 IBM Smart Business 桌面（预览）——IBM Smart Business 虚拟桌面通过 IBM 安全、可伸缩的公共云交付。
- IBM Smart Business 桌面云——通过客户自己的基础设施和数据中心交付的云服务。

在一次关于 Sun 最近被 Oracle 收购的采访（PDF）中，Larry Ellison 说：

> 我们希望与 Fujitsu 共事 ,在 SPARC 微处理器中设计先进的特性 ,来改善 Oracle

> 数据库的性能。在我看来，这能让 SPARC Solaris 开放系统的主机和服务器叫板 IBM 在数据中心的统治地位。

但收购背后最重要的原因或许是 Oracle 希望借助今年早些时候发布的 Sun Cloud 进入云市场。这样，Oracle 就会加入 Amazon、Google、HP、Microsoft 及其它公司的行列。

原文链接： http://www.infoq.com/cn/news/2009/06/Acrobat.com-IBM-Oracle

相关内容：

- 云计算七问七答
- 基于云的跨企业信息系统会取代企业外部网吗？
- 认识云计算
- Jeff Barr 谈论 Amazon Web 服务
- 云计算比较：EC2, Mosso 和 GoGrid

[热点新闻]

Google Wave 加速 HTML5 发展？

作者 宋玮

最近，Google Wave 闪亮登场了，给大家带来不少惊喜。而 Google 宣称全面采用 HTML5 这一事实是否意味着 HTML5 时代已经来临呢？

HTML4 成为标准已经有十余年了，照理说，HTML5 也该出山了。可是 HTML5 目前并没有标准化。按照 HTML5 的编辑 Ian Hickson 的说法，HTML5 的最终提议草案要等到 2022 年才会发布。没错，这太遥远了。不过，各个浏览器厂商们并没有闲着，在去年第一份草案发布之后，大多数浏览器都实现了该草案的一些特性。

也许是因为 HTML5 标准指定过于缓慢，因此一直不是风口浪尖。然而世事难料，Google Wave 的发布似乎一下引发了许多人对 HTML5 的兴趣，也激起了人们的广泛讨论，使得 HTML5 成为最近的一个热门话题。本月初，InfoQ 一篇名为 Google Wave 会影响 RIA/Silverlight 吗？的新闻，谈到了 Wave 对 RIA 可能造成的负面影响及微软相关人士的回应。无独有偶，Paul Krill 在 Infoworld 上也发表了类似的一篇文章，HTML5:会终结 Flash 和 Silverlight 吗？，Paul Krill 认为：

> HTML 5，一个突破性的 Web 展现规范，可能会改变 Web 应用开发中的游戏规则，甚至可能废弃 Adobe Flash、Microsoft Silverlight 及 Sun JavaFX 这样的基于插件的 RIA 技术。

对此，Adobe 的 John Dowdell 在其博文 Adobe on "HTML5"中给出了他的看法。文中他引用了 Adobe CEO Shantanu Narayen 针对一分析家所提 HTML5 对 Adobe 的机遇与威胁时所做的回答。

> Shantanu Narayen：

> “此标准能够增进 Web 内容的变革及一致性，从工具角度来讲，我们非常支持。我们的工具将支持创建和管理这一层次上的 HTML 内容。
>
> 我认为 HTML5 受到越来越多的关注，表明 RIA 实现及提供具有吸引力的用户体验对我们的客户来说越来越重要了。我觉得对于 HTML5 来说，挑战仍然是跨浏览器如何保持 HTML5 显示一致这一问题。……
>
> 因此，随 HTML5 的发展，我们将在我们的 web 著作工具中对其提供支持。但是从持续推动 Flash 及围绕 Flash 和 RIA 的变革角度来看 ,我们仍认为浏览器的分裂实际上使得 Flash 更加重要了。 ”
>
> ……
>
> Shantanu 的最后一点引起了我的共鸣……整个“HTML5”运动将有益于 Flash，因为已经很少有人会反对“体验更重要”这一观点。现在已经不是 5 年前了。Silverlight 的出现使得 Flash 更加流行……iPhone 的出现增加了支持 Flash 手机的数量……“HTML5”的公布只会使得那些仍在争论图像、动画、音频/视频及丰富交互在 Web 上没什么用的人被边缘化。无论浏览器引擎各自能做到多少，Flash 将会高质量的提供这些特性。

虽然，微软和 Adobe 都否认了 HTML5 会给他们带来冲击，很明显他们都在关注 HTML5 的发展。那么，HTML5 时代真的到来了吗？还不好轻易下结论。但是 Google Wave 对 HTML5 的推广确实起到了作用。或许正如 Pieter 在其博文 Why HTML 5 is going to change the web? 中所总结的：

> 当一个像 Google 这样的大公司在 HTML5 上投巨资以使其应用广泛采用 HTML5 时，其它公司不得不跟进。如果你已经看到了 Google wave 的演示，你就知道它能做什么。
>
> 目前 W3C 并没有推荐 HTML5，因为它仍未完成。我希望它快点搞完而且所有浏览器都来支持它。

如果你想了解 HTML5 的一些出彩的特性展示及相应源码，可以参考 Matt 的 5 Amazing HTML5 Features to Look Forward to。

如果你想让 IE6、IE7 支持 HTML5 标签 ,可以参考 remy sharp 的博文 HTML5 enabling script

以及 Paul Hayes 的博文 Moving markup towards HTML5。

原文链接：http://www.infoq.com/cn/news/2009/06/google-wave-promote-html5

相关内容：

- Google Wave 会影响 RIA/Silverlight 吗？
- Google 凭借 Chrome 2.0 和 Wave 推动 Web 平台
- Google 开源网页加速工具 Page Speed
- 谷歌开发者日：Google Wave 将基于 BSD 协议开源
- Google App Engine 增加了 Java 支持：喜忧参半

[热点新闻]

ActionScript 3 网站构建框架

作者 Moxie Zhang 译者 沙晓兰

开发基于 RIA 的网站有很多种方法。对于和多媒体内容息息相关的业务和个人来说，Flash 形式的网站尤其动人。Fosfr 是一个 ActionScript 3 网站构建框架，专门用来创建 Flash 网站。InfoQ 采访了 Fosfr 的创始人——Jeff DePascale，以深入了解 Fosfr。

Q. 是什么驱使你开始开发这个 Fosfr 框架的？

A. 我大部分工作都跟小型网站开发相关。Fosfr 最早是为了保证各网站的版本能保持一致。最初，AS2 中有组特别的类严格控制站点的构建，Fosfr 作为 AS3 框架主要是以前面提到的那个类库为基础来扩展 给我所有的 Flash 构建——无论是小型网站 还是独立的 SWF，或者是其它项目——提供了稳定、一致的基础。

Q. 你为什么觉得会有 AS3（或者说 Flash）网站的需求？

A. 目前有两种对立的看法，一部分认为 Flash 有益于互联网的发展，另一部分人认为 Flash 只是让整个网络充斥着用户体验很烂而又无法被检索到的网站。就我个人认为，每个技术都有各自发展的天地 ,非要说某种技术在像互联网这样迥异的媒体中比另一种技术更适合，未免有些误导。Flash 有缺点吗？当然有，SEO 方面就是其中一个非常大的缺点。Flash 有没有被用到一些不合适的场合？当然也有。但是，如果能有效应用它的长处的话，Flash 能提供强大的富用户体验，

Q. 你能概括下 Fosfr 的基本构架吗？

A. Fosfr 的整体实现是一个三层 SWF 结构。一个是核心层，或者说是主 SWF。这个核心层包含了预加载器，可以看作是项目的宿体。核心层之上是 shell SWF，这里包含了所有通

用的网站元素，诸如导航栏、脚注等等。最后，那些可变内容则放在子 SWF 文件中。

Fosfr 的优点还在于它有用于生成文档的类库。Fosfr 提供自定义文档类，所有.as 文件都能得到扩展（包括外部类）；它提供直至自动生成文档这个层次的自定义方法，这些方法可以是为了调试、跟踪、导航，也可以是为了访问 Fosfr 的其它控件，比方说 cookies、url 信息等等。项目的任何一个地方，你都能直接索引到 fosfr 对象，快速访问到框架的任何一个方面，可以访问到加载在项目中的所有 SWF。

整个项目由一个外部 XML 配置文件来驱动，根据这个配置文件，能构从该 xml 文件本身直接访问 Fosfr 主要控件的所有属性。借助于自动生成文档的类库的后台工作，只需要在每个.as 文件中添加几行简单的代码就能实例化整个框架。只要几分钟，你就能得到可以立刻运行的一个功能完善的网站了。

Fosfr 还能和 Prequel 集成使用，Prequel 是我开发的一个预加载 API；Fosfr 还能集成 SWFAddress，完全自动完成，不需要任何手动处理。通过 Fosfr 创建文档的类库来开发结合 SWFAddress 的子 SWF 的自定义功能会非常容易。

另外，如果你只需要一个 SWF 文件和一个对应的预加载器，那么可以只实现 Fosfr 的核心文件和 shell 文件，可以省去其中的导航 / SWFAddress 功能；或者通过框架中的 FosfrLite 创建一个独立的 SWF。FosfrLite 也可以选择使用 XML 配置文件。

Q. Fosfr 能和已有的 CMS 甚至诸如 Wordpress 这样的日志平台集成吗？

A. Fosfr 的 XML 配置文件包含了一个专门针对自定义网站元素的结点。CMS 或 XML 编辑插件直接与这部分 xml 配置交互的话，能够很容易操作在 Fosfr 上创建的任何 SWF 文件的各个方面。目前还没有给 Wordpress 及类似平台专门开发插件的计划，但我会在以后的开发中实现此类集成。

Q. 开发 Fosfr，你用的是什么开发工具？还有，Fosfr 和 Flex 有关吗？要是没有，以后会把这两者结合起来吗？

A. 开发 Fosfr，我用的是 FlashDevelop，这也是这段日子以来一直在用的开发工具。在调试方面，我结合采用了集成在 Fosfr 当中自定义调试器和 Flash 播放器的标准调试器。目

前我还没有任何计划要把 Fosfr 集成到 Flex 当中的计划。

Q. 你对 Fosfr 的前景有何规划？

A. 目前，Fosfr 的版本上处于 0.8。在推出 1.0 版本之前，我计划完成处理自定义事件和事件提醒的系统，在调试器上再多加一些功能，而且希望到时候还能集成我正在抓紧开发的、更新、更强壮的 Prequel 版本。处于开发阶段、尚未结合在公共发布当中的模块还有 tab 管理和音频管理。我还计划开发另外的文档管理类库来捆绑 papervision3D 站点的创建。1.0 及其之后的版本将囊括这些特性，而且会不断添加新功能。就眼前来说，更重要的是确定什么是目前要完成的，要编写快速入门指南，最好还要创建一个 AIR 应用把 XML 配置文件转变为向导。我希望开发人员能在半个小时以内就能轻松掌握 Fosfr，而且我觉得 Fosfr 能大幅度缩短产品的开发周期，这无疑是它的一大优势。但是如果没有快速入门指南和向导的话，这个优势也很难实现。Fosfr 非常有用，能够极大地缩短开发周期，但前提是你要懂得怎么去使用这个工具。我在不了解如何使用某个 API 的前提下直接去用这个 API，还确实遇到过很多问题。Fosfr 在 1.0 之后就不会有这些问题。所以编写指南是目前最迫切的一件事。之后，我会发布完全值得信赖的 1.0 版本，会尝试让公众关注到这个项目，目前我有意识地尽量把关注度控制到最小。

Q. 为什么要把这个开发框架叫作“Fosfr”？

A. 从技术上来说，它的全称是“Flash Open Source Framework”。然而，由于之前 SWFObject 和 SWFFit 因为名字当中有“Flash”这个词，在注册商标的时候遇到很多问题，所以现在严格采用 Fosfr。另外，这个名字厅听上去也很酷。

原文链接：http://www.infoq.com/cn/news/2009/06/as3-site-framework-fosfr

相关内容：

- Flex 开源数据可视化框架：Axiis
- 使用 Flash 实现增强现实
- Flex MVC Framework——Flight

- Playr 2.0: 继续做最好的
- AsWing：Java 开发者的 ActionScript 朋友？

[热点新闻]

主题演讲 ：Java 创新的未来

作者 Abel Avram 译者 曹云飞

在 2009 年 SpringOne 欧洲大会开幕演讲上，Rod Johnson 认为最近 Oracle 的收购不会使 Java 创新停止，而且相信 Java 会像过去几年那样在 Sun 之外继续进化。作为证据，他提到 了：Grails，Roo，一个用于提高开发者生产力的工具，一个免费的 STS（SpringSource 工具套件），tc 服务器和 dm 服务器。

请观看：Java 创新的未来（1 小时 55 分钟）

Rod 不相信 Oracle 会对 Java 的创新作出贡献：

- Oracle 不是革新家
- Oracle 的收购是为了取悦华尔街而不是开发者
- Oracle 专注于赚钱而不是创新

但是 Rod 认为 Oracle 不会影响 Java 创新，因为语言本身遵守 GNU GPL 协议，是开源的，而且 Oracle 不会停止 Java，因为他们无法做到，而且停止 Java 不符合他们的利益。

据 Rod 所说，近来 Java 的创新不是来自 Sun，而是来自外部，特别是来自于构建在 JVM 之上的诸如 Grovvy 和 Scala 这样的语言。现在重要的是平台和框架，创新必须在这些方面发展。

Rod 没有试图让参会者确信 Java 创新不会在近期消亡，而是展现了 SpringSource 最新的成果，并阐明了公司的目标：“构建、运行和管理 Java 应用的完整基础设施。”

Rod 认为 JVM 是一个出色的平台，可运行大型的可伸缩的企业应用，但是它缺失了“从

开发者到产品的联合经验”，不像 Ruby on Rails 那样是一个“高度集成的，一站式框架”。重点应该关注集成和生产力，而 SpringSource 为此贡献了 Grails，Roo，STS，tc Server，dm Server。

Grails 被 Rod 吹捧为 JVM 上最佳 web 开发框架。Grails 背后的公司 G2One 去年被 SpringSource 收购了。

Roo 是一个交互式的轻量级的用来提高 Java 开发者生产率的工具。开发者可以使用它来通过 Spring 对象模型进行 Java 编程。Roo 是一个建议的名字，开发者被邀请来投票并选择这个工具的最佳名称，包括：Spring Dart，Spring Spark，Spring Roo，Spring HyperDrive，Spring Boost。

Rod 还宣布他们将免费发布 SpringSource 工具套件（STS），今年晚些时候该套件将包括 Grails 和 Groovy 的工具。

另一项公告是发布 SpringSource 的 tc Server，该产品基本上是 Tomcat 应用服务器加上一些用来管理大量类似应用服务器安装的企业级功能。

Rod 的演讲中也演示了 Roo 和 tc server。

原文链接：http://www.infoq.com/cn/news/2009/06/SpringOne-Keynote-Rod-Johnson

相关内容：

- Java 依赖注入
- Spring 工具套件 2.0 发布
- Spring 2.5：Spring MVC 中的新特性
- Rod Johnson 谈 Spring、OSGi、Tomcat 及企业级 Java 的未来
- 企业中的 OSGi

[热点新闻]

RIA 平台：除了 Flex、Silverlight，还有 Laszlo

作者 霍泰稳

和 Flex、Silverlight 一样，Laszlo 也是一个用于构建 RIA 应用的优秀平台，其突出特性是一次编程，多平台部署的模式。InfoQ 中文站近期就 Laszlo 的有关问题采访了 Laszlo 系统公司北京研发中心的首席代表 Sue Liu 和技术总监赵万里。

提起 RIA 平台，很多人开始想到的多是 Adobe 的 Flex 和微软的 Silverlight，再进一步，可能还会提到 Sun 的 JavaFX，鲜有人会想到 Laszlo。但如果你了解了以下信息，相信对 Laszlo 的看法会有所改变。坊间传言因为源于 Laszlo 的压力，Flex 最终选择了开源；另外美国前五大电信运营商中有四家已经基于 Laszlo 构建了自己的应用，最后一家亦在洽谈之中；最后，Laszlo 还是开源的，其官方下载量已经突破 60 万。

根据 Sue 的介绍，Laszlo 平台主要包括两部分：用于写客户端应用的 LZX 语言和用于编译 LZX 的声明式服务器。LZX 并不是一个全新的语言，可以简单将其理解为 XML 和 JavaScript 的结合体。在实际开发中，LZX 使用 XML 标签定义页和用户接口，而用 JavaScript 处理逻辑和声明变量等。在 Laszlo 创建之初，它是一个收费软件，以出售软件许可证为生，在 2004 年的时候，它顺应潮流选择了开源，遵循通用公共协议（CPL），使得开发人员或者企业可以不受限地基于 Laszlo 构建应用。

在最终的呈现形式上，Laszlo 会将代码编译成 swf 文件，用户通过 Flash 插件在浏览器中即可使用。考虑到目前有有超过 98%的计算机上都安装了 Adobe Flash 软件，所以这不会成为 Laszlo 进一步普及的障碍。在 Laszlo 的成功案例中，目前有大型电信运营商 Verizon、SureWest、NRTC、CableVision、Cox，金融服务提供商 H&R Block、Barclays、Ameritrade，互联网企业 IBM、Sears、Walmart、BEHR 等。另外，据 Sue 透露，目前 Laszlo 研发团队还在努力实现将 LZX 程序编译成 Silverlight 或者其他 Ajax 应用，从而实现一次编写处处运行的目的。

在 Sue 的介绍中，也提到了 Laszlo 这样开源软件的商业模式。和其他较为成功的开源软件一样，比如 Spring，Laszlo 采取的也是软件平台开源，而依靠基于该平台的解决方案或者咨询获取营收。在 Laszlo 公司网站的产品列表中，可以看到 OpenLaszlo 是以开源软件的形式免费提供给用户下载的，目前其最新版本是 4.3，支持 DHTML 和 Flash 9。而帮助企业快速构建基于 Web 2.0 RIA 应用的 Laszlo Webtop 则是收费，面向开发人员的价格是每年 795 美元。说到 Laszlo 在中国的目标，赵万里表示，希望不久的未来，人们在提到 RIA 平台时，不仅想到 Flex、Silverlight，还应该包括 Laszlo！

对于想投入 Laszlo 社区的朋友来说，尤其是国内的开发人员，在打开这扇大门之前有什么需要三思的吗？Java 领域的专家 William Grosso 在 2005 撰写的文章《Laszlo：一个富互联网应用的开源框架》中回答了这个问题，现在看来依然适用：一是 Flash 虚拟机是设计用来显示动画的，对于数学计算或者大型数据处理依然存在不足；另外 LZX 虽然不是一个全新的语言，但是对很多人来说其学习曲线还是比较陡峭，特别是目前 Laszlo 社区不成熟和相关书籍缺少的情况下；但是，不管你是否决定采用 Laszlo，很明显的一点是你绝对应该下载试用它，而且还要好好想想 RIA 会如何改变 Web！最后，如果你想深入了解 Laszlo 系统和 LZX 语言，建议阅读 Manning 公司出版的《Laszlo in Action》。

原文链接：http://www.infoq.com/cn/news/2009/06/laszlo-ria-platform

相关内容：

- OpenLaszlo 正致力于支持 Flash Player 9 运行时
- Laszlo Webtop 支持 Web Services 集成
- Merapi 项目利用 Java 扩展 Adobe Air 的桌面功能
- 虚拟座谈会：RIA 和 Ajax 技术的现状与展望
- Flex 与 JSON 及 XML 的互操作

[热点新闻]

SOA 依旧已死?

作者 Boris Lublinsky 译者 马国耀

在她最初的博文 SOA obituray 发表之后的六个月中，InfoQ 上展开了一场热烈的讨论，伴随着很多的博文和批驳, Anne Thomas Manes 在这篇博文中再次明确了她对 SOA 的看法。

在博文的开头 Anne 就阐述了她对 SOA 的立场：

> 作为一个词汇“SOA”已经凋零，但是作为实践方式“SOA”对于任何的企业的发展而言还是非常重要的。很多企业已为 SOA 投资了数百万，但是可见的收益甚微。还有些企业甚至比开始（实施 SOA）时更差。在经济紧缩的形势下，商界人士不再有特别的兴趣去向一艘看起来快要沉没的船砸钱了。如果今年你打算为 SOA 项目争取资金的话，请别使用“SOA”这个词，而要将精力集中在构建“服务”上，它将会为企业带来更多价值。

Anne 引用 Gartner 最近的一份调查再次证明 SOA 已死，该调查显示 40%的用户无法衡量 SOA 的投资需要多久可以收到回报。调查还显示 50%的尚未开始 SOA 项目的公司正是因为不能说清和展示 SOA 的业务价值而没有实施 SOA。缺乏价值衡量的方法，SOA 项目注定失败。此外，SOA 被分析师和软件提供商过度炒作，以至于：

> .....很多实施 SOA 项目的公司对 SOA 有过高的期望，却忽略了为赢取收益将要付出的努力、资源以及时间等等。

基于很多企业的 SOA（基础设施）的开支，Anne 认为，更多迹象表明 SOA 已死：

> - 最近 Gartner 发布了对应用集成和中间件（AIM）市场的年度评估，评估显示该市场在 2008 年度只有个位数的增长。根据 Application Development Trends 对该评估报告的评论，“2009 年，中间件市场将出现刹车”，Gartner

> 预计中间件市场今年将会出现 0.8 百分比的降幅。
>
> - .....通过对市场的研究，Report Buyer 断言 IBM 占有 SOA 基础设施市场 70% 的份额。因此，IBM 的销售业绩将是 SOA 基础设施市场一个很好的风向标。并且，参考 James Governor 在 IMPACT 大会上所透露的风声，“软件销售 Robert LeBlanc GM 说，现在客户购买 SOA 的单子比先前小很多了”（我把“购买 SOA”理解为“购买 SOA 基础设施软软件”，因为众所周知我们没法买 SOA）

这篇博文和最初的那篇相似，也引起了一些回应，例如，Pierre Fricke 在回复 Anne 这篇博文时写到：

> 我觉得，我们是把经济萧条的初潮对企业（软件）开支带来的正常冲击解读为“SOA 已死”.....（软件）开支就像信用枯竭造成资金流通显著放缓一样撞上了南墙.....SOA 的确受这场初潮的影响很深。而如今企业在第一回合（的较量）之后拭去身上的层土，他们发现他们仍然需要管理业务，企业比以前更需要竞争力.....一个有趣的发现是，金融服务公司比以前对待 SOA 和 BMP/BRMS 项目的态度有所放松.....把 SOA 项目得不到资助看成是企业不愿意实施 SOA 或者 SOA 已死，对此我们相当困惑，（这种理解）到底是深思熟虑呢，亦或是欠考虑?

为回应 Annee，Steve Jones 专门写了一篇博文。Steve 解释到，“SOA 不是一项技术，而是关于实施的方法”。

在 Steve 看来，当前 SOA 失败的主要原因是分析师和提供商都优先把人们推向技术道路上（Anne 通过 SOA 基础设施的开销来评估 SOA 的健康程度）：

> 因此，当提供商不断申明“我们一直在不断改进产品”的同时，当初让人们去购买这些产品的分析师们回头又抱怨他们并没有看到多少成效，这其实是一场闹剧.....SOA 并没有失败，失败的是“涂在猪嘴上的下一代口红”的噱头，技术本身是可以解决问题的，但是做一个大项目来解决所有问题的 IT 风格往往会失败。

Steve 认为，死亡的不是 SOA 而是：

> .....僵尸的幼苗是提供商驱动的策略没能给 IT 部门带来它应有的益处，相同的

> 厄运正在云的周围发生，原因是拥有僵尸的公司在市场推广时寻求颠覆的又一种法，例如，SaaS 很大程度上是业务驱动的，而他们要利用它重新销售已经失败的相同技术。

SOA 始终是关于架构的，而不是获得资金的方法，或是分析师炒作的对象，也不是特定的中间件平台。在过去的 10 年间，IT 界在理解和改进这种架构风格上已经取得了显著的进步。因此 Anne 需要明确死亡的是什么。分析师的炒作？不论如何它总会出现架构发展的道路中的。企业主管的盲目投资？现在，企业架构师必须清楚地解释他们将赢得什么，为什么会赢，而非空头支票，我们应该看到这是件好事。斩断软件提供商的利润？目前开源产品以及更高的企业需求迫使软件商（为我们）带来更加创新的产品。

是的，我们的确看到 SOA 软件开支的下滑，但是我们在其他行业或其他产品也看到了相应的下滑。Pierre Fricke 说到："还想用 2006-07 年的价格来买房？买车？艺术珍品？原油开采租约？" 你对 SOA 的发展是怎么看的？欢迎投票表决。

原文链接：http://www.infoq.com/cn/news/2009/06/SOADead

相关内容：

- 挑战 2009——创造商业价值的架构趋势
- 摩尔定律太慢了
- 争论：SOA 已死？
- 面对巨灾，我们需要什么样的软件？
- 如何在动荡的职场中存活

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

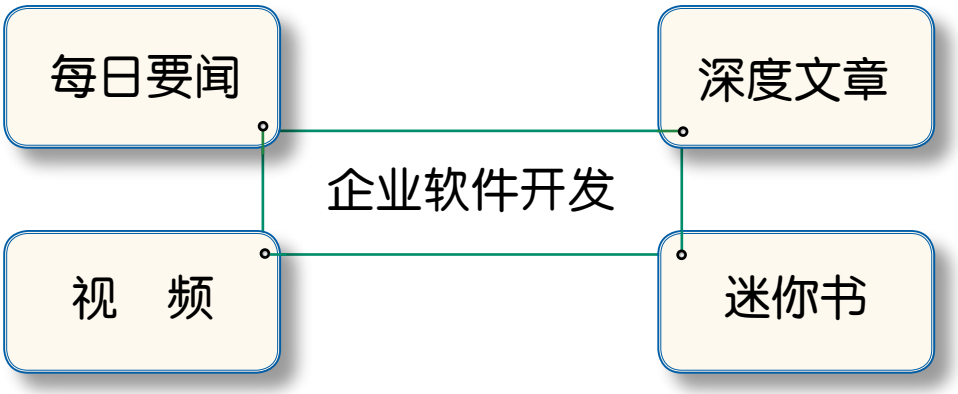
我们的**定位**：关注高级决策人员和大中型企业

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]

融合思想：深入探索 S#arp 架构

作者 Billy McCafferty 译者 张逸

开发 Web 应用程序是痛苦的。无论是组织与测试 AJAX 脚本，还是在无状态环境下模拟状态，开发 Web 应用程序需要在计划和开发的各个阶段中，全神贯注，专心致志。Web 开发人员还要面对开发中的诸多挑战，例如对象与关系的不匹配；在纷繁复杂的选项中选择最合适的工具提高开发效率；在项目中应用良好的架构，在保证代码具有可维护性的同时，不至于影响项目的交付。种种问题都使得开发形势变得越发严峻。

一切都在发展之中。不断涌现出的技术与技能虽然正在逐步解决这些开发中的难题，但没有任何一个可以单独扮演银弹的角色。但是通过对各种技术的权衡 ,精心挑选技术与技能，还是可以在不牺牲质量的前提下，大幅度地提高开发效率与可维护性。本文关注于 Web 开发的主流发展方向，通过使用 S#arp 架构，这是基于 ASP.NET MVC 的一个框架，荟萃了这些技术与技能的精华，从而为客户贡献价值。

赢得关键阶段的技术趋势

如果要用一个词来描述软件开发这个行业，毫无疑问，这个词语就是“变化”。比起那些历史久远的学科如土木工程学，我们这个行业不过是刚刚兴起，还处于萌芽状态。我们正在成长，而成长的烦恼就是行业自身经历的大量变化，而且看起来这样的变化还会持续一段时间。

这种变化可谓屡见不鲜。一个例子是项目管理方法学，由于其走向误区而带来的惨痛经历，它经历了大起大落，从一棵冉冉升起的“明星”，迅速衰落至默默无闻。另一个例子是有关技术的兴衰，长江后浪推前浪，新技术总是后来居上，取代旧的技术。以 ASP.NET 中的 MVP(Model View Presenter)模式为例 ,该设计模式虽然为 ASP.NET 引入了更好的可测试性，

但却增加了复杂程度。最近，微软引入了 ASP.NET MVC 用来取代 ASP.NET，它将之前 ASP.NET 实现的可测试性提升了一个新的台阶。由于要考虑.NET web 开发的可测试性，MVP 模式总是将控制器逻辑放在 ASP.NET 页面后面的代码（code behind）中，而 ASP.NET MVC 则抛弃了 MVP 的这种做法。这并不是说 MVP 隐含的原则到现在已经无效，而是随着这项技术的出现，通过使用适当的关注分离，可以更好的简化 MVP 实现可测试性的目标。

因此，虽然软件行业仍然变化莫测，但技术发展的特定趋势与设计理念，仍然构成了开发与交付高质量可维护项目的基础。或许，这些理念的实现会随着时间的推移而发生变化，但理念本身却是成功软件的坚实基础，能够持续地影响着软件开发。下面，我将简要地回顾这些设计理念，它们在关键阶段所获得的成功，已被开发社区所广泛接受，因而对未来的软件开发产生了深远的影响。

抽象的基础功能

在不久之前，我还在为一个新对象编写 CRUD 功能而被搞得焦头烂额。这项工作就像重新粉刷我的房子一样，费力而不讨好。充斥的大量重复代码，也成为了错误的多发地带。从编写存储过程与 ADO.NET 的适配器，到测试片断的 JavaScript 验证代码，我发现我每天都将大把精力投入到了这些基础功能细节的实现上，以至于在我写完这些代码之后，巴不得赶尽将它们抛诸脑外。

范式的转换在过去十年间已经发展成熟，其中关于基础功能的实现细节，属于最底层的工作，最好能够交给专门的工具来完成。面临的挑战是需要为这项工作找到合适的工具，既要允许软件忽略这些实现细节，又要保证基础功能可以使用。NHibernate 是这类工具的典范。它能够处理普通的.NET 对象与关系数据库之间的持久化。采用这种方式，它就能够让对象自身完全忽略如何实现持久化，又能够解决对象与关系之间的不匹配。而且，它不需要编写任何一行 ADO.NET 代码或者存储过程。NHibernate 是一个非常棒的工具，更重要的是它实现了一个远大目标，通过提供一个固定的解决方案，避免乏味而又琐碎的基础功能实现。要知道，这些实现在过去可是开发活动的重要组成部分。

随着时间的推移，软件行业引入了大量成熟的工具与技术，对这些乏味无趣的基础设施构建进行抽象，然后在开发完成后再进行设置。例如，随着 NHibernate 的逐渐成熟，那些

附加的插件，例如 Fluent NHibernate 具有自动映射的能力，完全能够减轻管理数据访问的负担。这一现象印证了 Douglas Hofstandter 著作 Gödel, Escher, Bach 中提到的预言，即：采用合理的抽象是软件开发的发展之道，需大力提倡。

松散耦合

遗留软件系统最常见的毒瘤是紧耦合。（我在这儿使用的“遗留”一词，指的是那些其他开发人员强加给你的破烂软件，或者是你自己在很多年前开发的破旧系统）紧耦合的例子多不胜数，例如两个对象之间存在双向依赖；具体依赖于服务的对象如数据访问对象；还有在单元测试中，如果依赖的服务断开或不可用，就无法测试服务的行为。紧耦合会导致脆弱的代码，使代码难以测试。修改紧耦合的代码，会让开发人员视如畏途，不战而降，甚至逃之夭夭。毫无疑问，一个成功软件的关键就是松散耦合。

维基百科将松散耦合解释为“两个或多个系统之间的弹性关系。”因此，松散耦合带来的好处就是你能够修改编程关系的任何一方，却不会影响另外一方。举例说明，在我看来，没有哪个设计模式能够比接口隔离，别名为依赖倒置（不要与依赖注入混为一谈）更能够说明松散耦合的思想了。该技术通常会用于将数据访问层从领域层中分离。

例如，在一个 MVC 应用程序中，控制器或者应用服务需要与数据访问的仓储（repository）对象通信，以获取数据库中的项目个数。（本例中的仓储指的是“服务”）要满足这一需求，最简单的办法是让控制器建立一个新的仓储对象的实例。换句话说，控制器创建了一个指向仓储对象的具体依赖；例如通过 new 关键字。遗憾的是，这种方法会导致紧耦合的诸多不良后果：

- 没有真实的数据库支持对仓储的查询，就很难对控制器进行单元测试。要求真实的数据库，会导致单元测试变得脆弱，一旦数据被前一次运行的测试所修改，就会带来问题。在测试控制器逻辑时，我们应集中关注验证控制器的行为，而不是它所依赖的仓储对象能否成功地与数据库进行通信。此外，测试一个“真实的服务”，例如在刚才提及的例子中，与真实数据库通信的仓储服务，会使单元测试的运行变得像老牛拉破车一般的缓慢；结果会让开发人员停止运行单元测试，从而损害了代码的质量。

- 如果不修改实例化服务的控制器，就难以替换服务（仓储服务）的实现细节。假如你希望将仓储从 ADO.NET 改为支持 Web Service，由于包含了一个与前者之间的具体依赖，如果不对实例化以及使用它的控制器进行大量修改，就无法轻易地将其替换为后者。在多数情形下，一旦变化发生，就会导致霰弹式修改——这是说明这一问题的另一种坏味道。
- 无法确定控制器实际拥有的服务依赖的个数。换句话说，如果控制器正在调用一些服务依赖的创建功能，则开发人员很难确定其逻辑边界，或职责范围。作为替代，控制器的依赖可以通过其构造函数传入，这样就能够使开发人员更容易理解控制器的整个职责范围。

替代方案是把服务依赖作为控制器构造函数的参数。这种做法带来的关键改善就是控制器只需要了解服务依赖的接口，而不是具体的实现。为了进一步说明，可以比较下列两段在 ASP.NET MVC 应用程序中的控制器代码。

如下的控制器直接创建了它的服务依赖 CustomerRepository，也就相应拥有了一个具体的依赖：

```
public class CustomerController {
    public CustomerController() {}
    public ActionResult Index() {
        CustomerRepository customerRepository = new CustomerRepository();
        return View(customerRepository.GetAll());
    }
}
```

相反，如下的控制器则将服务依赖作为接口参数传递给它的构造函数。也就是它与服务依赖形成了松耦合关系。

```
public class CustomerController {
    public CustomerController(ICustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }
    public ActionResult Index() {
        return View(customerRepository.GetAll());
    }
```

```
        private ICustomerRepository customerRepository;
    }
```

对比紧耦合的缺陷，这种清晰的分离方式带来了诸多好处：

领域层对于如何创建仓储对象以及它的实现细节一无所知，它只需调用公开暴露的接口。因此，它不需要修改控制器自身，就能很容易地更换数据访问的实现细节（例如从 ADO.NET 切换到 Web Service）。这基于一个假定，两者实现的接口是相同的。

依赖于接口而不是具体实现，在单元测试时更容易将仓储的测试替身（test double）对象注入。这就保证了单元测试能够快速运行，避免维护数据库中的测试数据，从而将测试的注意力放到控制器的行为上，而不是与数据库的集成。

这里并没有详细阐述依赖注入必须支持一个分离的接口和其它松耦合技术。更详细的讨论请参考文章依赖注入实现松耦合。（注意，在这篇文章中描述的“仿（mock）”对象实际上是“桩（stub）”对象。它与“测试替身（test double）”的术语来源于 Martin Fowler 的 Mocks 不是 Stubs）。此外，将服务依赖重构到接口隔离一文详细地介绍了如何将设计转为接口隔离设计模式。

测试驱动开发

简而言之，测试驱动开发（TDD）能够有效交付高质量、可维护的软件，全面地简化设计。作为一项开发技术，测试驱动开发绝不会是昙花一现；它受到了越来越多的吹捧，是能够决定软件开发成败的关键，推动了我们这个行业逐渐走向成熟。

TDD 隐含的基本思想是带着疑问开始软件开发，在开发过程中要不停地询问系统。例如，倘若你正在开发一个银行系统，可能需要询问系统，它有能力成功处理客户的存款业务吗？关键之处在于要在实现行为细节之前提出问题。随之而来的好处就是在编写系统之前，它能够让开发者将注意力放到系统要求的行为上。

若要遵循测试驱动开发的指导原则，需按照如下步骤进行编码：

- 假设目标对象以及要求的行为已经存在，编写测试。
- 编译解决方案，此时会看到编译失败。

- 编写足够的代码使程序通过编译。
- 运行单元测试，测试失败。
- 编写足够的代码使得单元测试通过。
- 若有必要，进行重构！

测试驱动开发虽然没有什么变化，但在日常的开发活动中对它的应用却还在发展之中。例如，测试驱动开发最近的发展方向是行为驱动开发；该方法试图弥补“代码编写的技术语言与商务运用中的领域语言之间”的鸿沟。换句话说，行为驱动开发将 TDD 与后面介绍的领域驱动设计（或者是与你喜欢的其它方法）结合了起来。

领域驱动设计

从近日的发展趋势来看，领域驱动设计（DDD）可谓风光无限。该方法将软件开发的注意力放在了领域与领域逻辑上，而不是技术以及支持技术方案的关系数据库模型。和行为驱动设计一样，DDD 提出了大量的技术与模式，以改善客户与开发团队的协作关系，在描述语言上达成一致。理想情况下，客户应该能够理解 DDD 应用程序的领域层，而编码逻辑也能够完整地反映客户的业务逻辑。

我试验了各种方法，得出的结论是：领域驱动设计是早期编程方法的一种自然进化，例如针对数据库数据进行模型驱动开发，其对应的模型可以视为应用程序的核心，要做的工作就是操作数据。（Castle ActiveRecord 和 ADO.NET 实体框架都是非常稳定和优秀的模型驱动设计工具。）相反，在 DDD 中，数据库则被视为一种必要的基础设施细节，能够支持领域及相关逻辑。事实上，正所谓“万法皆空”，在 DDD 中本来就没有数据库。虽然，领域驱动设计需要数据库，但关键在于领域对于持久化机制（实现数据存储与获取的机制）应该是一无所知的。

而且，领域驱动设计不仅仅是将数据持久化从领域中分离出来。它的主要目的是让领域对象自身拥有领域行为。例如，我们不能分离出 CustomerAccountLogic 类，并由它来决定 CustomerAccount 是否取决于付款日期，而是要求 CustomerAccount 自身维护这一信息。采用这种方式，领域的行为是与模型自身是合二为一的。

以上介绍仅仅是运用 DDD 进行软件开发技术的冰山一角。若要了解领域驱动设计的更多信息，可以阅读文档领域驱动设计快速入门，它是 Eric Evans 经典著作《领域驱动设计》的简明摘要。

在 S#arp 架构中融入这些思想

每个项目都其独特的需求，也没有哪个框架能够尽善尽美，对于开发 Web 应用程序而言，这是机会与挑战共存的局面。但是在面临众多选择时，开发人员很难做出判断，哪些工具和技术适合给定的项目，并让开发中通常遇到的挑战能够迎刃而解。例如，如果你正在寻觅一个.NET 依赖注入工具——或者说控制反转容器——可以选择 Spring.NET（它远远不只是提供 IoC 功能）、Unity、Castle Windsor、Ninject、StructureMap，以及更多。这还只是对于 IoC 的选择！让事情变得更糟糕的是，我们需要明智地规划，恰如其分地在架构中权衡各种工具与技术，这是一项极具挑战的工作。

在.NET Web 开发中，至少在当前还缺乏一个通用的架构与基础，可以在程序开发中对各种技术与技巧进行最优组合，根据已经被证实的实践来选择最近的技术，以及由开源社区开发的优质工具。S#arp 架构正是基于这样的前提应运而生。开源的 S#arp 架构试图利用本文介绍的业已证明行之有效的实践，谨慎地选择工具以提高开发效率，保证系统高质高量，以及良好的可维护性。

S#arp 架构使用的工具与技术如下所示：

- 接口隔离模式，它结合依赖注入模式从数据访问层中移除对领域和控制逻辑的依赖；
- 仓储模式，根据单一职责原则，利用分散的类对数据访问进行封装；
- 模型-视图-控制器模式，通过 ASP.NET MVC 实现，在视图与控制器逻辑之间引入了清晰的关注点分离；
- NHibernate 以及它的扩展 Fluent NHibernate，不再需要开发和维护底层数据存储与获取的代码，保证领域对持久化机制一无所知；
- 使用了 Castle Windsor 默认提供的通用服务定位器，通过开发人员首选的 IoC 容器

实现松散耦合；

- 内存数据库 SQLite 用来运行行为驱动测试，从而避免与持久数据库的集成；
- Visual Studio 模板和 T4 工具箱，为每个领域对象生成项目的基础设施以及通用的 CRUD 脚手架，免除了枯燥乏味的编码工作。

选择的这些技术与工具表明，虽然软件开发没有奇妙的银弹，但选择一个稳定的开发实践，并结合适当的工具，就能带来巨大的价值。

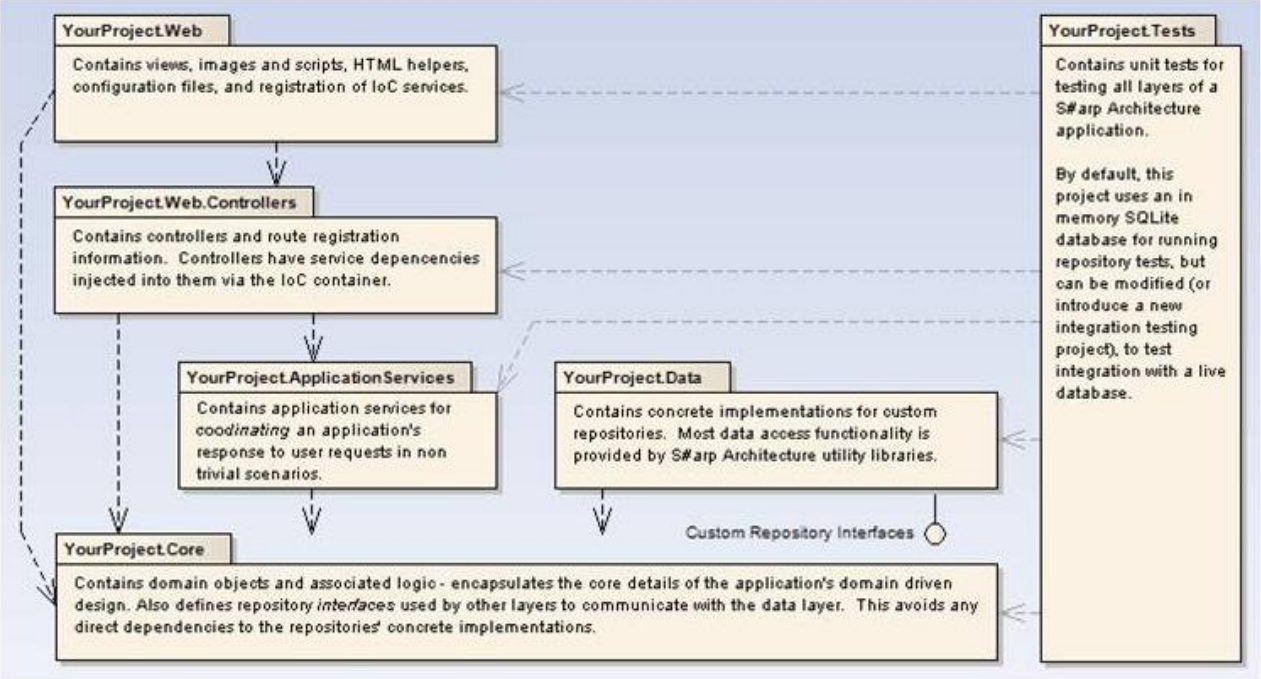
领域驱动架构将它们串联在一起

我认为，被封装到 S#arp 架构中的关键思想，就是颠倒领域与数据访问层之间关系的一种技术。在通常的应用架构中，尤其是与微软推荐的架构紧密相关的，其依赖关系都是从表示层开始自上而下，依赖于业务层，最后依赖于数据层。虽然这是对实现细节的过度简化，但它通常会建议将数据层作为最底层，被其他层依赖；这正是根据设计进行模型驱动。

虽然模型驱动方式自有其优势，也适用于众多解决方案，但却不符合本文提出的领域驱动的目标。让领域直接依赖于数据层，会造成另一种弊端，就是引入领域对象与数据访问代码之间的双向依赖。我的前任教授 Lang 博士说道，只有不断犯错误，最后才能成为这个领域的专家。我正在努力成为一名专家，我已经认识到领域对象与数据访问代码之间存在的双向依赖，它们正是麻烦的渊薮。（这个深刻教训使我在成为专家的道路上能够登堂入室。）

那么，我们该如何冲破这重重迷雾，让设计清晰地呈现这些思想呢？解决之道就是明确地分离各种应用关注点，颠倒领域与数据访问层之间的关系，也就是在领域层定义分离的接口，让数据访问层进行实现。采用这种方式，则应用程序的所有层都只依赖于数据访问（或其它外部服务）层的接口，从而保证对实现细节的一无所知。这会让设计变得松散耦合，更易于进行单元测试，而在项目开发的维护阶段，系统会变得更加地稳定。

下图演示了 S#arp 架构所主张的体系架构，使用接口隔离模式反转了传统的领域与数据访问层之间的依赖关系。每个 box 表示一个单独的物理程序集，箭头则表示了依赖关系及其依赖方向。



注意 ,图中的数据层定义了仓储的具体实现细节 ,它依赖于核心的领域层。在核心层中，除了定义领域模型和逻辑，还定义了仓储接口，该接口可以被其他各层所调用，例如应用服务层会与仓储通信，这就保证了它独立于其实现细节。某些建议认为表示层（在图上方的 YourProject.Web ）不应直接依赖于领域层。作为替代，可以引入数据传输对象（ DTO ），从而在数据传递给视图展现时，更好地将领域层从表示层中分离。

怎么做？

我们工作的底线是，软件交付专家们必须及时交付解决方案，且要符合客户的需求，并具有高质量和可维护性。很多实践都经过了千锤百炼，所谓“前人栽树，后人乘凉”，在个人的应用程序中，我们不必重新创造，只需求助于经过验证而又真实的设计模式，以及基本工具的实现，就能够解决开发中的常见问题，例如数据持久化和单元测试。真正的挑战在于要合理地制定计划，事先做出权衡，既要提高开发效率，又不能扼杀我们的能力，企图以创新的方式去迎合不切实际的需求是不可取的。我希望本文所描述的技术与工具，以及 S#arp 架构对它们的组合，足以说明，虽然条条道路都能够通罗马，然而前车之鉴，若要避免重蹈覆辙，就必须从一开始就要汲取经验教训，并要有足够的智慧来赢得一个稳定的开端。

了解更多信息

S#arp 架构项目已经开发了接近一年的时间，它为快速开发稳定的领域驱动程序提供了简单但又强大的架构基础。你可以从 http://code.google.com/p/sharp-architecture 下载 S#arp 架构的 RC 版本。1.0 的 GA 版则与 ASP.NET MVC 1.0 密切相关。S#arp 架构的论坛欢迎大家踊跃发言，畅谈自己的体验。

关于作者

http://devlicio.us

若要提到编写优美的软件，那么 Billy McCafferty 可以说是经验老到，久经沙场，偏生他又一味追求编程的罗曼蒂克，以至于无可救药。Billy 目前身兼两职，一方面他负责管理一家小规模的培训与咨询公司 Codai（很快会推出新的网站），同时又在 Parsons Brinckerhoff 带领开发人员与架构师团队。在发布 S#arp 架构 1.0 版本之后，Billy 的生活又将重回正规，在不久的 ALT.NET 以及其他开发大会上，你能够见到他的身影。

原文链接：http://www.infoq.com/cn/articles/Sharp-Arch-Billy-McCafferty

相关内容：

- 五问 Eric Hexter 和 Jeffery Palermo
- ASP.NET 开发人员需要学习 ASP.NET MVC 么？
- ASP.NET MVC 开源了
- ASP.NET MVC 或可大大推动 VB 的使用
- 在 ASP.NET MVC 中使用 T4

[推荐文章]

使用 HTML5 构建下一代的 Web Form

作者 蒋博

HTML5 是由 WHATWG (Web Hypertext Application Technology Working Group) 发起的，最开始的名称叫做 Web Application 1.0，而后这个标准吸纳了 Web Forms 2.0 的标准，并一同被 W3C 组织所采用，合并成为下一代的 HTML5 标准。

前言

HTML 语言作为如今编程最为广泛的语言，具有易用、快捷、多浏览平台兼容等特点，但是随着时代的进步，HTML 的标准却停滞不前，这一次还在不断开发中的 HTML5 标准的更新可以说给这门标记语言带来了新的生命力。本文将着重讨论 HTML5 中的 Web Forms 2.0, 即表单的部分。

表单是网页中常见的控件(集)。小到网站注册登录，大到一个企业的数据管理系统，都基本上有表单的身影。表单之所以如此重要，主要是因为它担负大量的用户和网页后台数据更新交互的任务。Web 开发人员，对于网页表单可以说又爱又恨，爱的是它方便的收集、组织数据的功能，恨的是它的功能很大程度上也就仅此而已。一些在最终网站用户看起来稀松平常的功能，比如说输入类型检查、表单校验、错误提示等等，开发人员无不需要花费大量精力利用 JavaScript 和 DOM 编程来满足这些天然所需的功能点，而随着 Ajax 的流行，出现的一些 JavaScript 的工具库，比如 Dojo, YUI 等都提供了方便的 JavaScript Widget 或者 API 来减轻开发人员的负担。

HTML5 的表单新特性

HTML5 Web Forms 2.0 是对目前 Web 表单的全面提升，它在保持了简便易用的特性的同时，增加了许多内置的控件或者控件属性来满足用户的需求，并且同时减少了开发人员的编

程。在我看来，HTML5 主要在以下几个方面对目前的 Web 表单做了改进：

新的控件类型

还在为类型检查犯愁吗，还在为那一长串看不太明白的检验输入的正则表达式而苦恼吗，HTML5 提供的一系列新的控件将天然的具备类型检查的功能。比如说 URL 输入框，Email 输入框等。

```
<input type="url"></input>
<input type="email"></input>
```

当然还有非常重要的日期输入框，要知道使用 JavaScript 和 CSS 来“手工”制作一个日期输入框还是非常花功夫的，类似 Dojo，YUI 这样的类库也无不在这个 widget 上面大做文章。

```
<input type="date"></input>
```

作为我痛苦记忆的一部分，我经常记得我们开发人员要为一个 select 下拉别表动态的添加非常多的选项,这些选项大多数都是来自数据库，比如说国家、省市列表等等。这个事情非常繁琐。HTML5 将支持 data 属性，为 select 控件外联数据源！

```
<select data="http://domain/getmyoptions"></select>
```

改进的文件上传控件，你可以使用一个控件上传多个文件，自行规定上传文件的类型（accept），你甚至可以设定每个文件最大的大小（maxlength）。你看出它和一般操作系统提供的文件上传控件的区别了吗，反正我觉得基本一致了。在 HTML5 应用中，文件上传控件将变得非常强大和易用。

重复（repeat）的模型，HTML5 提供一套重复机制来帮助我们构建一些重复输入列表，其中包括一些诸如 add、remove、move-up，move-down 的按钮类型，通过这一套重复的机制，开发人员可以非常方便的实现我们经常看到的编辑列表，这是一个很常见的模式，我们可以增加一个条目、删除某个条目、或者移动某个条目等等。

内建的表单校验系统，HTML5 为不同类型的输入控件各自提供了新的属性，来控制这些控件的输入行为，比如我们常见的必填项 required 属性，以及为数字类型控件提供的 max、min 等。 而在你提交表单的时候，一旦校验错误，浏览器将不执行提交操作，而会显示相应的检验错误信息。

```
<input type="text" required></input>
<input type="number" min=10    max=100></input>
```

XML Submission，我们一般常见的是 form 的编码格式是 application/x-www-form-urlencoded。开发人员都很清楚这种格式，数据送到服务器端，可以方便的存取。HTML5 将提供一种新的数据格式：XML Submission，即 application/x-www-form+xml。简单的举例说，服务器端将直接接收到 XML 形式的表单数据。

```
<submission>
    <field name="name" index="0">Peter</field>
    <field name="password" index="0">password</field>
</submission>
```

实例分析

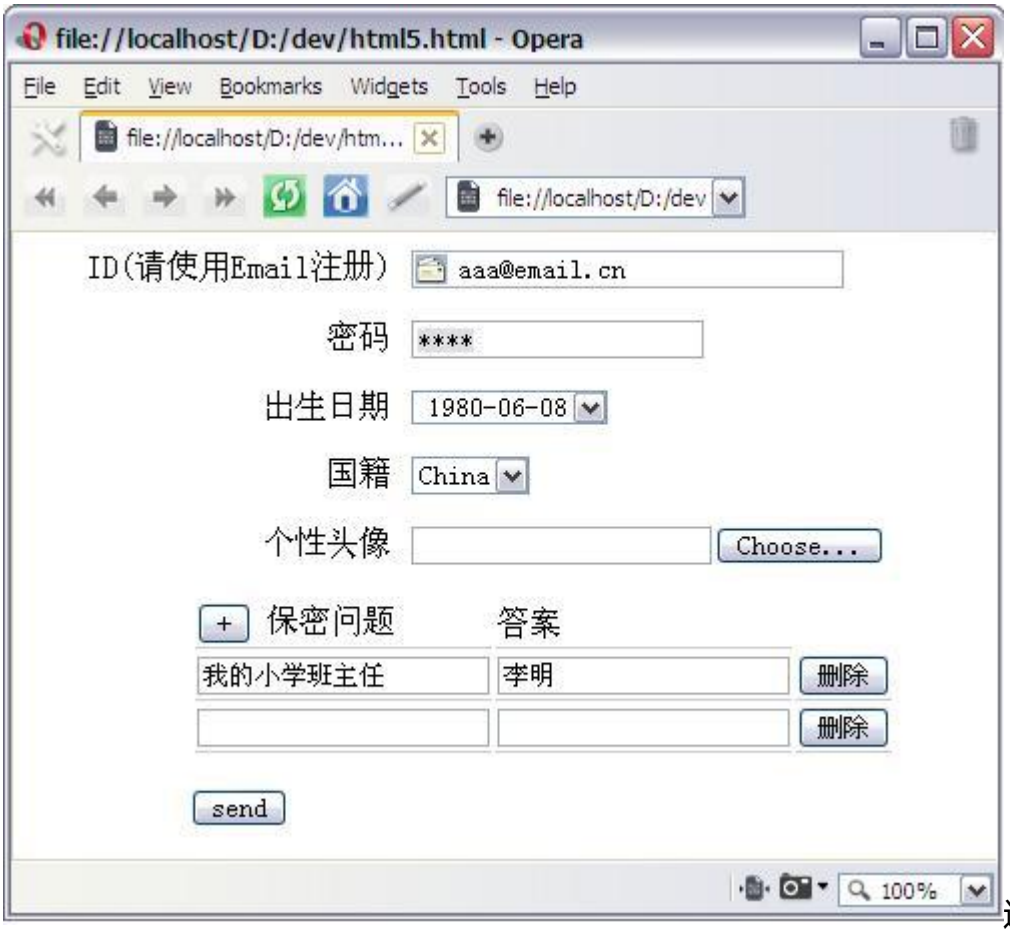
我将利用 HTML5 新的表单系统，做一个简单的用户注册的界面，包括用户名，密码，出生日期，保密问题等内容，代码如下：

```
<! doctype html>
<html>
    <head>
        <style>
            p label {
                width: 180px;
                float: left;
                text-align: right;
                padding-right: 10px
            }
            table {
                margin-left: 80px
            }
            table td {
                border-bottom: 1px solid #CCCCCC
            }
            input.submit {
                margin-left: 80px
            }
        </style>
    </head>
```

```
        <body>
            <form action='/register' enctype="application/x-www-form+xml"
method="post">
                <p>
                    <label for='name'>ID(请使用 Email 注册)</label>
                    <input name='name' required type='email'></input>
                </p>
                <p>
                    <label for='password'>密码</label>
                    <input name='password' required type='password'></input>
                </p>
                <p>
                    <label for='birthday'>出生日期</label>
                    <input type='date' name='birthday' />
                </p>
                <p>
                    <label for='gender'>国籍</label>
                    <select name='country' data='countries.xml'></select>
                </p>
                <p>
                    <label for='photo'>个性头像</label>
                    <input type='file' name='photo' accept='image/*' />
                </p>
                <table>
                    <thead>
                        <td><button type="add"
template="questionId">+</button>  保密问题</td>
                        <td>答案</td>
                        <td></td>
                    </thead>
                    <tr id="questionId" repeat="template" repeat-start="1"
repeat-min="1" repeat-max="3">
                        <td><input type="text"
name="questions[questionId].q"></td>
    <td><input type="text" name="questions[questionId].a"></td>
    <td><button type="remove">删除</button></td>
```

```
                    </tr>
                </table>
                <p>
                    <input type='submit' value='send' class='submit' />
                </p>
            </form>
        </body>
    </html>
```

由于目前 HTML5 标准仍然在开发中，不同的浏览器对 HTML5 特性的支持都相当有限。其中 Opera 在表单方面支持得比较好，本实例在 Opera9 上运行一切正常，效果图如下：



这个实例运用了一些 HTML5 的新的表单元素 ,比如 email 类型的输入框(ID) ,日期类型的输入框(出生日期)。并且使用了重复模型来引导用户填写保密问题 ,而在个性头像的上传中 ,通过限制文件类型，方便用户选择图片进行合乎规范的内容上传。而用户选择国籍的下拉选择输入框中，采用的是外联数据源的形式，外联数据源使用 coutries.xml，内容如下：

```
    <select xmlns="http://www.w3.org/1999/xhtml">
```

```
        <option>China</option>
        <option>Japan</option>
        <option>Korea</option>
    </select>
```

并且 form 的 enctype 是 application/x-www-form+xml，也就是 HTML5 的 XML 提交。所以一旦 form 校验通过，form 的内容将会以 XML 的形式提交。你还会发现，在 ID 输入框如果没有值，或者输入了非 email 类型的字符串时，一旦试图提交表单，就会有提示错误的信息出现，而这都是浏览器内置的。

结语

HTML5 对表单控件的更新，无疑是很振奋人心的。本文描述了一部分表单的新特性，还有一部分新特性同样很令人期待。相信随着标准的深入开发以及浏览器对 HTML5 支持程度的进一步提升，设计一个简单易用的表单的工作，将变得非常轻松。

关于作者

蒋博，主要从事 Web 前端技术的开发工作，在 Web 开发与性能分析以及敏捷实践等领域有较丰富的经验。对 HTML5 的发展以及各种 JavaScript 类库有比较浓厚的兴趣，经常关注社交型的网站发展情况，平常喜欢听音乐，看一些历史类书籍。（本文仅代表个人观点，与公司立场无关。）

原文链接：http://www.infoq.com/cn/articles/html5-web-form

相关内容：

- 位置感知的浏览会成为主流么？
- XHTML 2 与 HTML 5 依旧背道而驰
- RESTful 世界里的 Cool URI
- XSI 对决 URI?
- Web IDL：W3C DOM 规范语言绑定有了新名称

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]

使用 Spring AOP 和 AspectJ 编排工作流

作者 Oleg Zhurakousky 译者 宋玮

1. 简介

如果你需要实现一个流式的流程，特别是嵌入的，并且你想让其易于配置、扩展、管理和维护。你是否需要一个功能齐备的 BPM 引擎呢：引擎都有自己的抽象负载，它对于你正在寻找的简单流程编排来说似乎过于笨重了；或者有什么轻量级的替代方案可以使用，让我们不必采用一个功能齐备的 BPM 引擎？本文说明了如何使用面向方面编程（AOP）技术来构建并编排高可配置、可扩展的轻量级嵌入式流程流（process flow）。目前例子是基于 Spring AOP 和 Aspect J 的，其他 AOP 技术也可实现同样的结果。

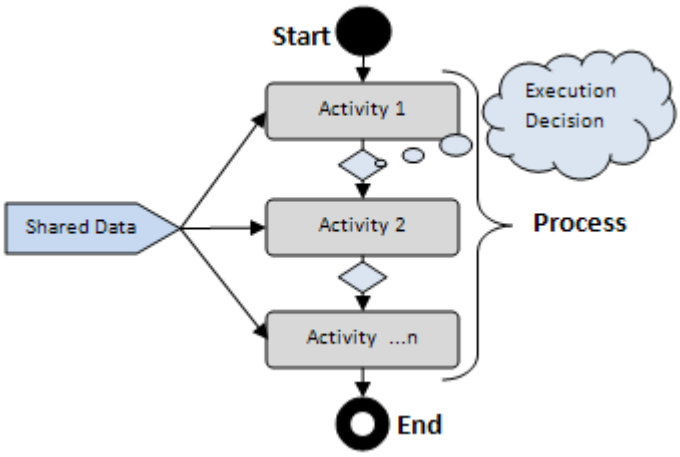
2. 问题

在我们继续深入讨论之前，首先我们需要更好地理解实际的问题，然后试着把我们对问题的理解与一套可用模式、工具和/或技术进行匹配，看看是否能找到一个合适的。我们的问题就是一个流程（process）本身，那么让我们好好理解一下它吧。什么是流程？流程是经过协调的活动的集合，这些活动致使一组目标得到实现。活动（activity）是指令执行的一个单元，它是一个流程的基本组成部分。每个活动操作一部分共享数据（上下文），以实现流程整体目标的一部分。已被实现的流程目标的各部分代表既成的事实（facts），这些事实被用来协调剩余活动的执行。这实质上把流程重新定义为一个在事实集合上进行操作的规则模式，用来协调定义该流程的那些活动的执行。为了让流程协调活动执行，它必须知道如下属性：

- **活动**——定义流程的活动

- **共享数据/上下文**——定义共享数据的机制和活动所完成的事实
- **转移规则——基于已注册的事实**，定义前一个活动结束之后跟着是哪个活
- **执行决策**——定义执行转移规则的机制
- **初始化数据/上下文（可选）**——由该流程操作的共享数据的初始化状态

下图显示了流程的高层结构：



我们现在可以用如下需求集合来形式化一个流程：

- 定义把流程装配为一个活动集合的机制
- 定义各个活动
- 定义共享数据的占位符
- 定义在流程范围内的这些活动协调执行的机制
- 定义转移规则和执行决策机制，根据由活动注册的事实执行转移规则

3. 架构和设计

我们定义架构将从解决头 4 个需求开始：

- 定义把流程装配为一个活动集合的机制

- 定义各个活动
- 定义共享数据的占位符
- 定义在流程范围内的这些活动协调执行的机制

活动是一个无状态工作者，它应该接收一个包含一些数据（上下文）的 token。活动应该通过读写来操作这一共享的数据 token，同时执行由这个活动所定义的业务逻辑。共享的数据 token 定义了一个流程的执行上下文。

为了坚持前面我们制定的轻量级原则，没有理由不把我们的活动定义为实现了 POJI（Plan Old Java Interfaces）的 POJO（Plain Old Java Objects）。

这里是 Activity 接口的定义，它只有一个 `process(Object obj)` 方法，其输入参数代表了一个共享数据（上下文）的占位符。

```
public interface Activity {

   public void process (Object data);

}
```

一个共享数据的占位符可以是结构化或非结构化（比如 Map）对象。这完全取决于你。为简单起见，通常我们的 Activity 接口把它定义为 `java.lang.Object`，但是在真实环境中，它或许被表达为某种结构化对象类型，流程的所有参与者都知道这一结构。

流程

因为流程是活动的集合，我们需要定出装配及执行这种集合的机制。

有许多方式可以达成这一目的。其中之一是把所有活动插入到某种类型的有序集合中，按其预先定义好的顺序迭代它调用每个活动。这种方法的可配置性和可扩展性明显很差，因为流程控制和执行的所有方面（aspects）都将被硬编码。

我们还可以用某种非传统的方式来考虑流程：假定流程是一个行为较少的抽象，并没有具体实现。可是，从一个活动过滤器（Activity Filters）链中过滤这种抽象，可以定义这个流

程的性质、状态和行为。

假定我们有一个叫做 GenericProcess 的类，其定义了 process(..)方法：

```
public class GenericProcess {

   public void process(Object obj){

      System.out.println("executing process");

   }

}
```

如果我们直接传递输入对象来调用 process(..)方法，不会发生什么事情，因为 process(..)方法没有做什么事情，上下文的状态将保持非修改状态。但是如果我们试图在调用 process(..)之前找到一种方法来引入活动，并让该活动修改这个输入对象，那么 process(..)方法仍将保持不变，但是因为输入对象是由活动预先处理过的，流程上下文的整体结果将会改变。

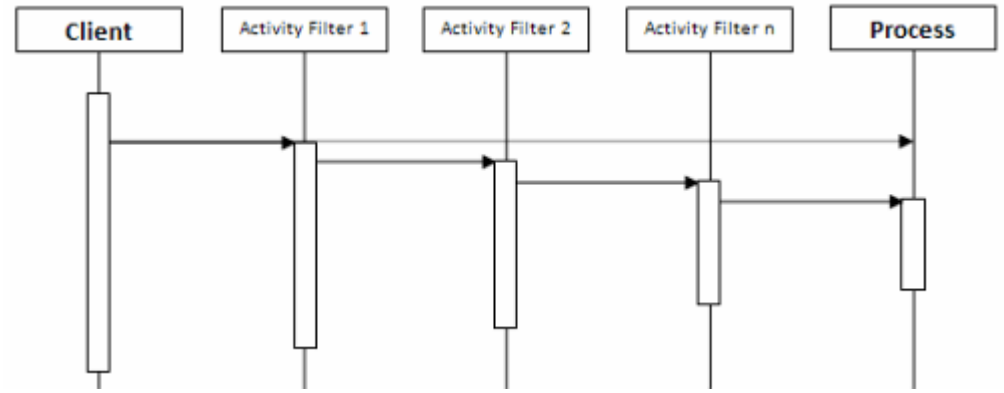
这种把拦截过滤器（Intercepting Filter）应用到目标资源的技术在拦截过滤模式中有很好的文档记录，在今天的企业级应用中被广泛使用。典型的例子是 Servlet 过滤器（Filters）。

拦截过滤器模式用一个过滤器封装了已有应用资源，这个过滤器拦截了请求的接收及响应的传递。一个拦截过滤器可以前置处理（pre-process）或重定向（redirect）应用请求，而且可以后置处理（post-process）或替换（replace）应用响应的内容。拦截过滤器还可以改变顺序，无需改变源代码就可以把一个分离的、可声明部署的服务链加入到现有资源——http://java.sun.com/blueprints/patterns/InterceptingFilter.html。

过去，这一架构通常用来解决非功能关注点（concerns），比如安全、事务等等……但是你可以清楚地看到，通过拦截过滤器（代表各个活动）链来装配类流程结构，同样的方法也可以很容易被应用来解决应用的功能特性。

下图显示了对于一个流程（Process）的调用如何被过滤器链拦截，其中每个过滤器都与流程的某个活动（Activity）相关联，这样，实际的目标流程组件没有什么事情可干，使之

成为一个空的、可重用的目标对象。改变该过滤器链你就得到了自己一个不同的流程。



唯一留下来要做的事情就是看看有没有一个框架 ,可以帮助我们以优雅的方式装配类似这样的东西。

基于代理的（Proxy-based）Spring AOP 看起来是理想的选择，因为它给我们提供了简单的结构和最重要的东西——执行机制。它将使我们能够定义一个有序拦截过滤器集合，来代表一个给定流程的各个活动。输入的流程请求将会由这些过滤器所代理，用活动过滤器（Activity Filters）中的行为实现来装饰流程。

这样留给我们的只有下面唯一的需求：

定义转移规则和执行决策机制，根据由活动注册的事实执行转移规则

基于代理的过滤器的美妙之处是转移机制本身。每当我们调用一个目标对象(流程)时，拦截过滤器将一个接一个的被代理机制所调用。这是自动的，而且在每个活动必须被调用的情况下工作得相当好。但是实际并不总是这种情况。早先我们在问题定义中描述之一是：“已经被实现的流程目标的那部分，表示已完成的事实，它被用来协调剩余活动的执行”——这意味着一个活动完成并不是必须转移到另一个活动。在实际的流程中，转移必须严格地基于前一个活动已完成和/或未完成的事实。这些事实必须向共享数据占位符注册，这样它们就可以被审核。

完成这一点可以简单到在我们的拦截过滤器中放一个 IF 语句：

```
public Object invoke(MethodInvocation invocation){
   if (fact(s) exists){
      invoke activity
   }
}
```

但是这将产生多个问题。在我们研究是些什么问题之前，让我们先搞清楚一件事：在当前的结构中，每个拦截过滤器紧密地与相应的 POJO 活动耦合在一起。正因为如此，我们可以很容易把所有活动逻辑保持在拦截过滤器本身中。唯一能够阻止我们这样做的是我们希望活动保持为 POJO，这意味着在拦截过滤器中的代码将简单地委派给活动回调（Activity callback）。

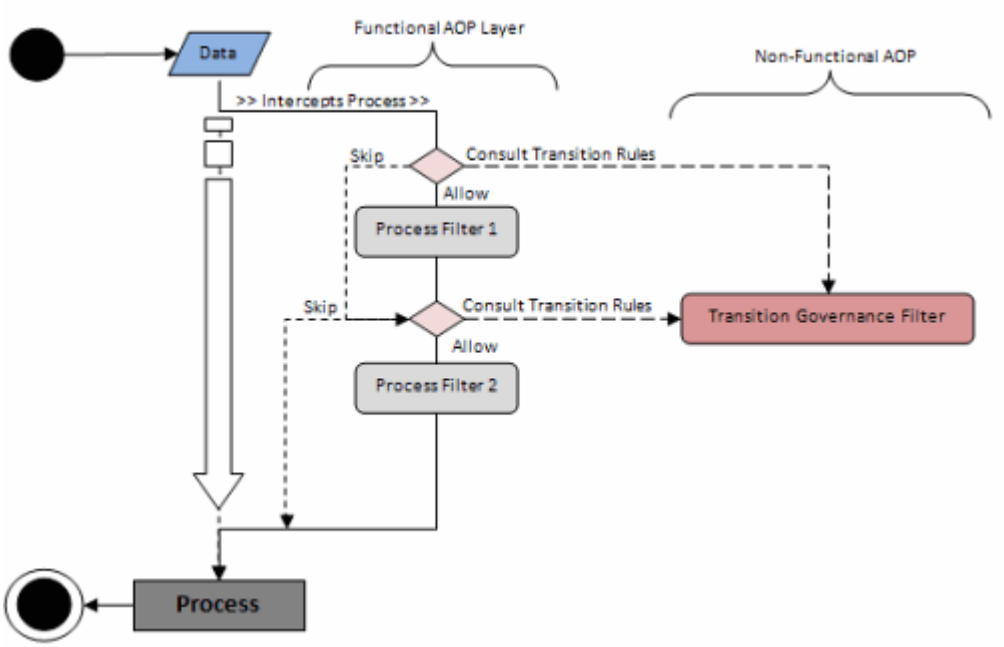
这意味着如果我们把转移求值逻辑放在活动中，我们将从根儿上把这两个关注点耦合在一起（活动转移罗辑和活动业务罗辑），这将违背分离关注点的基本架构原则并将导致关注点/代码耦合（concern/code coupling）。另一个问题则涉及到所有拦截过滤器会重复同一转移逻辑。我们称之为关注点/代码扩散（concern/code scattering）。转移逻辑横贯所有拦截过滤器，你可能已经猜到了，AOP 再次成为所选技术。

我们所需做的一切就是写一个 around advice，它将使我们能够拦截对实际过滤器类目标方法的调用，对输入求值，并作出转移决策，要么允许，要么不允许目标方法执行。唯一要告诫的是我们的目标类本身恰恰就是拦截过滤器。因此本质上我们正在试图拦截拦截器。不幸的是 Spring AOP 不能帮上忙，因为它是基于代理的，因此我们的拦截过滤器已经是代理基础架构的一部分了，我们不能代理这个代理（proxy the proxy）。

但是 AOP 最好的特性之一就是它可以有多种不同的风格和实现（例如，基于代理的，字节码编织等等……）。尽管我们不能使用基于代理的 AOP 来代理另一个代理，但是我们使用字节码编织 AOP,谁也拦不住，它将通过编织（编译时或装载时）我们的转移求值逻辑，来形成我们代理的拦截过滤器，这样就可以保持转移和业务逻辑分离。使用像 AspectJ 这样的框架很容易做到这一点。这样做，我们就给我们的架构引入了第二个 AOP 层，这是非常

有意思的实现。我们使用 Spring AOP 来解决功能关注点，比如用活动编排流程；同时，我们又使用 AspectJ 来解决非功能关注点，比如活动导航和转移。

下图记录了被显示为两个 AOP 层的流程流的最终结构，其中功能 AOP 层（Functional AOP Layer）负责从有序的拦截过滤器集合中装配流程，而非功能 AOP 层（Non-Functional AOP Layer）则解决转移控制的问题。



为证明这一架构的工作情况，我们将实现一个简单的用例——购买物品（Purchase Item），它定义了一个简单的流程流。

4.用例（购买物品）

想象一下你正在在线购物。你已经选择了某项物品，把它放入到购物车中，然后前去结账，给出你的信用卡信息并最终提交购买物品请求（purchase item request）。系统将初始化购买物品流程（Purchase Item process）。

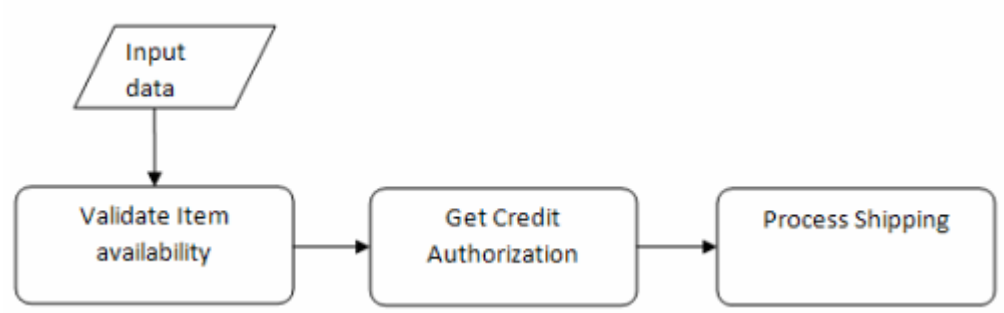
先决条件

流程必须接收包含物品、账单、配送信息的数据

主流程

1. 校验物品数量
2. 获得贷记授权
3. 配送

这个流程当前定义了 3 个活动，如下图所示：



这个图还显示了不受控制（ungoverned）的活动转移。但实际上，如果物品数量不够会发生什么？“获得贷记授权”活动应该执行吗？“配送”应该随之进行吗？

另一个有趣的告诫是：根据条件，贷记授权（credit authorization）可能不会自动完成（授权网络关闭了），而你或客户服务代表不得不直接打电话给贷记公司以获得认证码。一旦该认证码被获得并输入到系统中，这个流程应该从何处重新开始或继续呢？从头来还是直接进入配送环节？我觉得应该进入配送环节，但是怎样才能做到呢？我们怎么才能从中间重新启动该流程，而不用维护和管理许多执行控制呢？

有意思的是，使用 AOP 我们不需要维护执行控制，也不需要维护流程的流向。它是在通过拦截过滤器链时由框架本身来处理的。我们需要做的一切就是提出一种机制，根据注册事实允许或不允许各个过滤器执行。

“校验物品数量”将注册物品数量充足的事实，该事实是“获得贷记授权”的前提。“获得贷记授权”也要注册贷记已授权这一事实，而这又是“配送”活动的前提。存在或缺少事实也将被用来决策何时不执行一个特定活动，这又把我们带回到了“手工贷记审核”场景，怎么才能从中间重新启动流程呢，或者问一个更好的问题： 我们怎样才能重新启动该流程，而不重复该流程上下文中已经执行过的活动？

记得吗，共享的数据 token（上下文）也代表了流程的状态。这一状态包含了该流程登记的所有事实。这些事实被求值以做出转移决策。因此，在“手工贷记审核”场景中，如果我们从最开始重新提交整个流程，我们的转移管理机制，在遇到第一个活动“校验物品数量”之前，会马上意识到物品数数量充足事实（item available fact）已经注册过了，这个活动不应再次重复，因此，它将跳到下一个活动——“贷记审核”。因为贷记已授权事实也已经注册过了（通过某种手工录入方式），它将再次跳到下一个活动“配送”，只允许这一活动执行并完成该流程。

在我们进入实际例子之前，有一个更重要的话题还要讨论一下，那就是活动被定义的顺序。尽管从一开始好像活动的顺序在（这些活动所定义的）流程转移决策中并不起任何作用。

流程中活动的顺序只是代表了流程本身的平衡能力——该策略基于事实的可能性和概率，它们的存在将给下一个活动的执行或不执行创造一个理想的环境。改变活动的顺序永远不应影响整个流程。

实例：

```
    Legend:
  d - depends
  p - produces
Process:
  ProcessContext = A(d-1, 2; p-3) -> B (d-1, 3; p-4, 5) -> C(d-4, 5);
```

按照上面的公式，当流程在给定的 ProcessContext 内开始时，第一个要考虑的活动是 A，在它被调用之前依赖于事实 1 和 2。假定事实 1 和 2 存在于 ProcessContext 中，活动 A 将执行并产生事实 3。在线上的下一个活动是 B，它依赖于事实 1 和 3。我们知道我们的流程在活动 A 执行之前事实 3 发生的可能性和概率非常小。可是在活动 A 被执行之后，事实 3 存在的可能性和概率则相当高，因此活动 B 的顺序是跟在 A 后面。

但是如果我们把活动 B 和 A 的顺序颠倒一下，会有什么变化？

```
ProcessContext = B (d-1, 3; p-4, 5) -> A(d-1, 2; p-3) -> C(d-4, 5);
```

变化不大。当流程被调用时，维护着事实注册表的 ProcessContext 将很快断定已注册的事实不足以允许活动 B 被调用，因此它将跳到下一个活动 A。假定事实 1 和 2 是存在的，对

事实进行评估将确定已注册的事实足以允许调用活动 A，等等。活动 C 也将被跳过，因为它缺少由 B 产生的先决条件。如果流程再次与同一个的 ProcessContext 一起被提交，活动 B 将被调用，因为活动 A 在流程前一次调用过程中已经注册了活动 B 所需的事实，满足了 B 执行的前提条件。活动 A 将被跳过，因为 ProcesContext 知道活动 A 已经做了它的工作。活动 C 也会被调用，因为活动 B 已经注册了足够的事实以满足活动 C 的先决条件。

因此，正如你所看到的，变换活动的顺序不会改变流程行为，但可影响流程的自动化特征。

5.实例

该例子包含了如下制品：

GenericProcess 的实现，正如你所见，它包含的代码没什么意义，事实上它从未会包含任何有意义代码。这个类的唯一目的就是作为应用代表各个活动的拦截过滤器链的目标类。

```
18    public void execute(Object input) {
19        System.out.println("Executing dummy process method");
20    }
```

它相应的 Spring 定义为：

```
14
15    <bean id="purchaseItem" class="process.orchestration.sample.GenericProcessImpl"/>
16
```

PurchaseItem（购买物品）流程的其他配置包括三部分：

```
14    <!--  PART 1 - Process Assembly -->
15    <aop:config>
16        <!-- we are using "bean" pointcut in conjunction with execution pointcut
17             which will allow us to assemble some other process from another instance of
18             GenericProcessImpl -->
19        <aop:pointcut id="purchaseItemPointcut"
20            expression="bean(purchaseItem) and
21                execution(void process.orchestration.sample.GenericProcessImpl.execute(..))"/>
22        <aop:advisor pointcut-ref="purchaseItemPointcut"
23                                              advice-ref="validateItemFlter" order="1"/>
24        <aop:advisor pointcut-ref="purchaseItemPointcut"
25                                              advice-ref="validateCreditFlter" order="2"/>
26        <aop:advisor pointcut-ref="purchaseItemPointcut"
27                                              advice-ref="processShippingFlter" order="3"/>
28    </aop:config>
29
30    <!--  PART 2 - Activity Interceptors (Filters) Configuration -->
31    <!--  Individual Intercepting filters wired with corresponding
32          Activities and simple Fact rules-->
33    <bean id="genericFlter" class="process.orchestration.sample.ActivityFilterInterceptor"
34                                                    abstract="true"/>
35    <bean id="validateItemFlter" parent="genericFlter">
36        <constructor-arg ref="validateItemActivity"/>
37        <property name="facts" value="!VALIDATED_ITEM"/>
38    </bean>
39    <bean id="validateCreditFlter" parent="genericFlter">
40        <constructor-arg ref="validateCreditActivity"/>
41        <property name="facts" value="VALIDATED_ITEM,!VALIDATED_CREDIT"/>
42    </bean>
43    <bean id="processShippingFlter" parent="genericFlter">
44        <constructor-arg ref="processShippingActivity"/>
45        <property name="facts" value="VALIDATED_ITEM,VALIDATED_CREDIT,!PROCESSED_SHIPPING"/>
46    </bean>
47    <!--  PART 3 - POJO Activities -->
48    <bean id="validateItemActivity"
49            class="process.orchestration.sample.ValidateItemActivity"/>
50    <bean id="validateCreditActivity"
51            class="process.orchestration.sample.ValidateCreditActivity"/>
52    <bean id="processShippingActivity"
53            class="process.orchestration.sample.ProcessShippingActivity"/>
```

第一部分（第 14 行）——流程装配 AOP 配置，包含把 GenericProcessImpl.execute(..)方法定义为连接点(Join Point)的 pointcut。你还可以看到我们使用了 bean(purchseItem) pointcut 表达限定我们正在拦截哪个 bean。通过用应用于不同过滤器链的不同 bean 名创建 GenericProcessImpl 的另一个实例，我们可以定义多个流程。它还包含了对实现为 Aopaliance 拦截器的活动过滤器的引用。默认的，过滤器按照从顶置底的顺序排列，然而为了更清楚，我们还可以使用 order 属性。

第二部分（第 30 行）——通过定义 ActivityFilterInterceptor 的三个实例来配置活动拦截器。每个实例将被注入后面定义的相应 POJO 活动 bean 和事实属性。事实属性定义了一个简单的规则机制，允许我们描述一个简单的条件，基于此，下面的活动将被允许或不允许执行。例如：validateItemFilter 定义了 “!VALIDATED_ITEM”事实规则，它将被解释为：如果 VALIDATED_ITEM 事实还未被注册在事实注册表中，则允许调用活动。只要 validatItemActivity 执行了，这一事实将被注册在事实注册表中：如果这一事实还未注册，它将允许这一活动执行；如果事实已经注册，它将在流程与同一执行上下文一起重新提交时保护该活动不会被重

复执行。

第三部分（第 47 行）——为我们的流程配置三个 POJO 活动。

ActivityFilterInterceptor——它所做的所有事情就是调用底层 POJO 活动并把该活动返回的事实进行注册（第 53 行），并且可以让 POJO 活动对事实注册表（Fact Registry）或流程的任何其他底层架构组件的地点保持未知（请见下面代码片断）。可是正如我们后面将要看到的，这一拦截器本身的调用是根据每个拦截器配置中所描述的事实规则由 AspectJ advice 所控制的（第二个 AOP 层），从而控制各个活动的执行。

```
47    public Object invoke(MethodInvocation invocation) throws Throwable {
48        logger.info(">>>> Executing " + activity.getName());
49        Map input = (Map) invocation.getArguments()[0];
50        String[] facts = activity.process(invocation.getArguments()[0]);
51        for (String fact : facts) {
52            List<String> factRegistry = (List<String>) input.get("FACTS");
53            factRegistry.add(fact);
54        }
55        return invocation.proceed();
56    }
```

各个 POJO 活动简单地返回它们要注册的所有事实的 String 数组，然后由自己的拦截器将其注册到事实注册表中（见上面代码片断）。

```
13    public String[] process(Object input) {
14        // implement your business logic here
15        // return Facts
16        return new String[]{"PROCESSED_SHIPPING"};
17    }
```

TransitionGovernorAspect——是一个 AspectJ 组件，拦截对每个 Spring AOP 拦截器（代表各个活动）的调用。它是通过使用 Around advise 做到这一点的，在其中它对事实规则和当前事实注册表进行比较，就执行或跳过下面的活动拦截器调用做出决策。可以通过调用它自己的 invocation 对象（ProceedingJoinPoint thisJoinPoint）的 proceed(..)方法做到这一点，或者调用拦截过滤器的 invocation 对象（MethodInvocation proxyMethodInvocation）的 proceed(..)方法来做到这一点。

```
@Around("execution(public java.lang.Object process.orchestration.sample.*.invoke(..)) " +
        "&& args(proxyMethodInvocation) && target(filter)")
public Object intercept(ProceedingJoinPoint thisJoinPoint,
                        MethodInvocation proxyMethodInvocation,
                        ActivityFilterInterceptor filter) throws Throwable{
    Object returned = null;
    // Now we need to extract Fact registry from the input parameter of
    // the invocation object of the proxy filter
    Map input = (Map) proxyMethodInvocation.getArguments()[0];
    // get the list of registered facts
    List<String> factRegistry = (List<String>) input.get("FACTS");
    // get current activity which is wired to the proxy filter
    ProcessActivity currentActivity = filter.getActivity();
    logger.info(">> Intercepting: " + currentActivity);
    // get the list of fact rules to be evaluated against registered facts
    String[] facts = filter.getFacts();
    // Evaluate the facts in the context of the current activity
    // and make grant/deny decision
    if (this.grantInvocation(currentActivity, factRegistry, facts)){
        logger.info(">> Allowing invocation of the activity");
        // proceeds to the target which is a filter's invoke(..) method
        returned = thisJoinPoint.proceed();
    } else {
        // invoke proceed(..) method of the proxy filter
        // by doing so we'll skip the actual invocation fo the invoke(..)
        // method and obviously the activity itself
        logger.info(">> Skipping invocation of the activity");
        returned = proxyMethodInvocation.proceed();
    }
    return returned;
}
```

由于它是用 AspectJ aspect 实现的，我们需要在 META-INF/aop.xml 中提供配置（见下面配置片断）。

```
<aspectj>
    <weaver>
        <!-- only weave classes in our application-specific packages -->
        <include within="process.orchestration.sample.*" />
    </weaver>
    <aspects>
        <aspect name="process.orchestration.sample.TransitionGovernorAspect" />
    </aspects>
</aspectj>
```

因为我们要使用装载时 AOP，我们需要在 Spring 配置中注册编织器。我们通过使用 context 名字空间来做到这一点：

此时，我们已经做好测试准备了。正如你所见，测试没有什么特殊的，其步骤是：

- 获得 ApplicationContext
- 获得 GenericProcess 对象
- 创建一个事实注册列表
- 创建对象（在我们的用例中是 Map），代表输入数据以及执行上下文

- 调用 process 方法

```
24    public static void main(String[] args) {
25        ApplicationContext ac = new ClassPathXmlApplicationContext("application-config.xml",
26                                                                   PurchaseItemTest.class);
27        GenericProcess process = (GenericProcess) ac.getBean("purchaseItem");
28        List factRegistry = new ArrayList<String>();
29        factRegistry.add("VALIDATED_ITEM");
30        Map input = new HashMap();
31        input.put("FACTS", factRegistry);
32        process.execute(input);
33    }
```

由于我们使用的是 AspectJ 装载时编织 ,因此需要提供-javaagent 选项作为我们的 VM 参数。

VM 参数是：

```
-javaagent:lib/spring-agent.jar
```

spring-agient.jar 已经存在于 lib 目录中了。

在执行之后你应该看到类似下面的输出：

```
>> Intercepting: process.orchestration.sample.ValidateItemActivity@1db9852
>> Allowing invocation of the activity
>>>> Executing process.orchestration.sample.ValidateItemActivity
>> Intercepting: process.orchestration.sample.ValidateCreditActivity@1ed5459
>> Allowing invocation of the activity
>>>> Executing process.orchestration.sample.ValidateCreditActivity
>> Intercepting: process.orchestration.sample.ProcessShippingActivity@3cb075
>> Allowing invocation of the activity
>>>> Executing process.orchestration.sample.ProcessShippingActivity
```

正如你从该测试所看到的，初始的事实列表是空的，但是如果你用已有事实填充它，那么流程流将被改变。

试着把给注册表增加事实这行代码的注释去掉。

在你的测试中的把如下代码行注释去掉：

```
// factRegistry.add("VALIDATED_ITEM");
```

你的输出将变为：

```
>> Intercepting: process.orchestration.sample.ValidateItemActivity@afae4a
>> Skipping invocation of the activity
>> Intercepting: process.orchestration.sample.ValidateCreditActivity@1db9852
>> Allowing invocation of the activity
>>>> Executing process.orchestration.sample.ValidateCreditActivity
>> Intercepting: process.orchestration.sample.ProcessShippingActivity@1ed5459
>> Allowing invocation of the activity
>>>> Executing process.orchestration.sample.ProcessShippingActivity
```

6.结论

该方法说明了怎样使用两层 AOP 来装配、编排并控制流程流（process flow）。第一层是用 Spring AOP 实现的，将流程装配为拦截过滤器链，其中每个过滤器都被注入了相应活动。第二层是用 AspectJ 实现的，提供流程的编排及流控制。通过拦截过滤器链来代理我们的流程，将使我们能够定义和维护流程的流向。而代理机制无需像 BPM 这样单独的引擎的，也提供了执行环境。我们通过使用已有技术（Spring AOP）提供的控制和执行机制做到了这一点。

该方法是轻量级、嵌入式的。它使用已有 Spring 基础架构并建立在流程是已编排的活动集合的前提之上。每个活动是一个 POJO 而且完全不知道任何管理它的底层架构/控制器组件。这有几个优点。除了典型的松耦合架构优点外 ,随着像 OSGi 这样的技术不断的普及和采纳，保持活动和活动调用控制分离，把活动实现为一个 OSGi 服务也成为可能，这使得每个活动都成为独立的单元(部署、更新、卸载等等……)。易于测试是另一个优点。因为活动是 POJO，它们可以在使用它们的应用之外作为 POJO 来测试。他们有定义良好的输入/输出契约(它需要的数据以及它预期产生的数据)。你可以单独测试每个活动。

分离控制逻辑（拦截过滤器）和业务逻辑（POJO 活动）将使你能够给流程事实规则接插更加成熟的规则门面（facade），同样，测试转移逻辑也应该不影响由下面活动实现的业务逻辑。

活动是独立的基本组成部分，可以被在一些其它流程中重复使用。例如“贷记审核”活动可以很容易被重用，将其装配在某些其它需要贷记审核的流程上。

7.参考资料及资源

- 从这里可以下载本文的例程代码
- Spring Framework - http://www.springframework.org
- Spring AOP - http://static.springframework.org/spring/docs/2.5.x/reference/aop-api.html

- AspectJ - http://www.eclipse.org/aspectj/
- AOP 联盟 - http://aopalliance.sourceforge.net/
- OSGi 联盟 - http://www.osgi.org

8.关于作者

Oleg Zhurakousky 是一个 IT 专家，目前是 SpringSource 的高级顾问，拥有 14 年以上跨多领域的软件工程经验，包括：软件架构及设计、咨询、业务分析和应用开发。九十年代初他开始其职业生涯 ,从事 COBOL & CICS 方面的工作。自 1999 年以来他已将精力集中在专业 Java 和 Java EE 开发上。2004 年以后，Oleg 着重使用了几个开源技术和平台（特别是 Spring），同时在数个跨行业的项目中工作，比如：通信、银行、法律实施、美国国防部及其它行业。

原文链接：http://www.infoq.com/cn/articles/Orchestration-Oleg-Zhurakousky

相关内容：

- Ramnivas Laddad 谈 AOP 选型
- John Heintz 谈如何向 Java 注解添加行为
- 使用 AOP 实现应用程序失败转移
- 用 Spring 2.0 和 AspectJ 简化企业应用程序
- 使用 AJDT 简化 AspectJ 开发

[推荐文章]

使用 Perf4J 进行性能分析和监控

作者 Alex Devine 译者 崔康

许多开发人员都很熟悉墨菲法则的一个例子 :他们发现在花费了大量时间确保应用程序在开发环境中快速和灵活之后，在发布到生产环境的时候性能会不可思议的大幅下降。更糟糕的是，应用程序平时运行正常，老板或者重要客户操作应用的时候却反应缓慢。详细的日志记录和分析对于追踪这些间歇性的性能瓶颈尤为重要。

然而，当今世界充满了面向服务的架构和分布式的应用，查找性能瓶颈对应的组件极其困难。考虑一个典型 Web 2.0 风格应用的服务器端的常见场景：

1. 服务器接收一个 Web 请求，分发给负责产生响应的组件。
2. 该请求也许需要通过 LDAP 服务器进行安全验证。
3. 控制器组件对数据库执行查询。
4. 控制器组件也会调用第三方 Web 服务。
5. 控制器组件将所有获得的数据进行汇总，组成一系列业务对象用于显示。
6. 业务对象被展现，响应内容传回用户浏览器。
7. 运行于浏览器的 AJAX 代码产生其他的请求，与服务器端交互。

对于“为何我的网页反应迟钝？”这样问题的回答需要研究多个组件和执行路径，同时需要生产环境中所有应用组件的详细性能数据。

Perf4J 是一款开源工具包，用于添加 Java 服务器端计时代码、记录日志和监控结果。对于熟悉诸如 log4j 日志框架的开发人员来说，可以这样类比：

```
Perf4J is to System.currentTimeMillis() as log4j is to System.out.println()
```

如何利用这个类比理解 Perf4J 呢？回想一下过去还没有广泛应用 Java 日志记录框架的糟糕岁月，我们大多数人如何添加日志记录语句。我们使用 System.out.println()作为一种“简陋的调试器”，利用这种快捷但糟糕的方式记录信息。我们很快意识到，这是不够的。我们希望把记录语句输出到专门的日志文件中（如果可能的话，多个不同文件），也许可以每天覆盖日志。我们需要能够设定重要性的不同级别以输出不用的日志语句，可以选择在不改变代码的情况下在特定环境下只输出特定日志，或者在不同环境中改变日志格式。因此，log4j 提供的丰富功能来源于原始想法，是一种“更好的”System.out.println()日志语句。

类似的，当 Java 新手发现他们需要添加性能监控代码时，他们经常这样做：

```
long start = System.currentTimeMillis();
// execute the block of code to be timed
log.info("ms for block n was: " + (System.currentTimeMillis() - start));
```

但是很快，这些开发人员发现他们需要更多的信息，综合的性能统计数据如平均、最小、最大、标准差和特定时间段内每秒的事务处理量。他们希望将这些数据绘成实时图表监控运行服务器上的问题，或者通过 JMX 输出性能指标以便于启动监控器在性能下降的情况下发出警报。此外，他们还希望计时语句可以和类似 log4j 的框架配合使用。

Perf4J 的目标是通过易于集成（和扩展）的开源软件包提供这些功能。Per4J 是由 Homeaway.com 开发的，其基础设施的分布式特性和网站的高可用性及性能需求需要深入的性能分析。Perf4J 的特点包括：

- 简洁的 stop watch 计时机制。
- 提供命令行工具，从原始的日志文件中生成汇总的统计数据和性能图表。
- 定制的 log4j appender，可以在运行时应用中生成数据和图表，计划在以后的版本中支持 java.util.logging 和 logback。
- 能够以 JMX 属性的形式发布性能数据，在数据超过指定阈值时发送通知。
- 提供@Profiled 注解和一套自定义机制，允许在与 AOP 框架（如 AspectJ 或者 Spring AOP）集成时巧妙的计时。

下面的例子展现了如何轻松利用这些功能。可以通过 Perf4J 开发人员指南来了解集成 Perf4J 的详细信息。

利用 StopWatch 类开发计时代码

org.perf4j.LoggingStopWatch 类用于在代码中添加计时语句并打印到标准输出或者日志文件中：

```
StopWatch stopWatch = new LoggingStopWatch();
//... execute code here to be timed
stopWatch.stop("example1", "custom message text");
```

对 stop()方法的调用记录了执行时间并打印日志信息。默认情况下，基类 LoggingStopWatch 将输出打印到 System.err 流中。但是大多数情况下，你需要使用一个集成到现有 Java 日志框架（如 Log4JStopWatch、CommonsLogStopWatch 或者 Slf4JStopWatch）的子类。下面是一些 stop watch 的输出示例：

```
start[1233364397765] time[499] tag[example1] message[custom message text]
start[1233364398264] time[556] tag[example1] message[custom message text]
start[1233364398820] time[698] tag[example1] message[custom message text]
```

使用 LogParser 创建统计数据和图表

虽然默认的 stop watch 输出相比直接调用 System.currentTimeMillis()来说没有很大的改进，但真正的好处在于能够解析这些输出以生成统计数据和图表。LogParser 通过 tag 和时间片把 stop watch 输出分组，生成详细的统计信息和可选的时间序列图（使用 Google Chart API）。下面是一些使用默认文本格式（也支持 csv 格式）的示例输出：

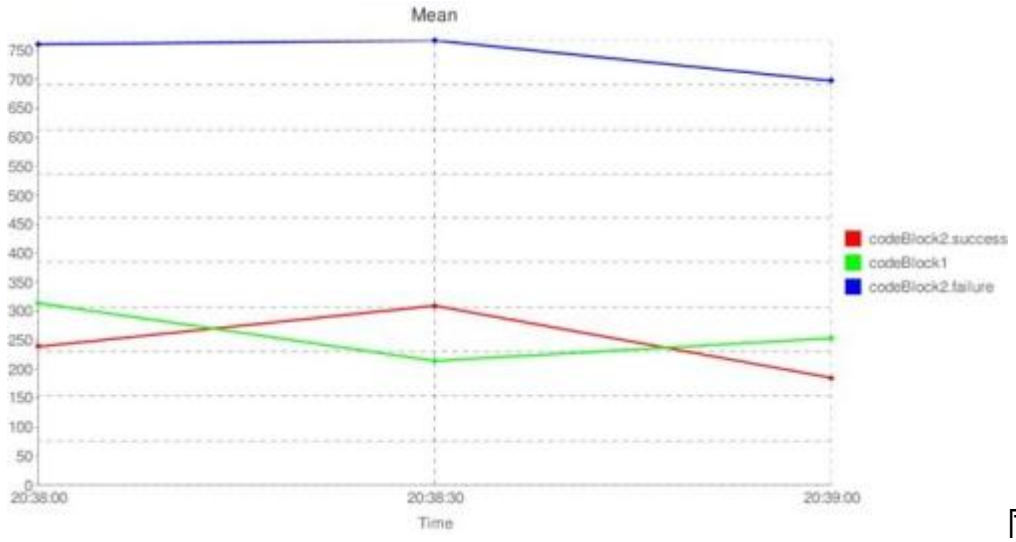
```
   Performance Statistics   20:32:00 - 20:32:30
Tag                                       Avg(ms)        Min        Max     Std Dev
Count
codeBlock1                                  249.4          2        487       151.3
37
codeBlock2.failure                          782.9        612        975       130.8
17
```

```
codeBlock2.success                              260.7          6        500       159.5
20


Performance Statistics   20:32:30 - 20:33:00
Tag                                          Avg(ms)        Min        Max     Std Dev
Count
codeBlock1                                     244.0          7        494       150.6
41
codeBlock2.failure                             747.9        531        943       125.3
21
codeBlock2.success                             224.1         26        398       106.8
21


Performance Statistics   20:33:00 - 20:33:30
Tag                                          Avg(ms)        Min        Max     Std Dev
Count
codeBlock1                                     289.3         10        464       141.1
22
codeBlock2.failure                             781.1        599        947       135.1
8
   codeBlock2.success                              316.2        115        490
112.6        13
```

平均执行时间和每秒事务处理量的图表以指向 Google Chart Server 的 URL 的形式生成。



同时，虽然 LogParser 默认从标准输入中读取数据，但是你也可以指定一个来自运行时服务器的日志文件，用 LogParser 实时输出：

```
tail -f performance.log | java -jar perf4j-0.9.8.1.jar
```

集成 Log4J

Perf4J 的扩展功能大部分通过一套定制的 log4j appender 提供。这样开发人员就能在部署阶段通过非常熟悉的日志框架来零零散散的添加分析和监控功能(未来的 Perf4J 将提供定制 logback appender 和 java.util.logging 处理器)。其中一个重要的功能是允许 Perf4J 作为 JMX MBeans 的属性展示性能数据，同时在性能低于可接受的阈值时发送 JMX 通知。因为 JMX 已经成为处理和监控 Java 应用的标准接口，提供 Perf4J MBean 开启了广泛的由第三方监控应用提供的功能。举例来说 ,我们 Homeaway 的 IT 部门标准化了 Nagios 和 Cacti 用于系统监控，这两个工具都支持把 MBeans 作为数据源查询。

下面的 log4j.xml 文件示例显示了如何启用用于 JMX 的 Perf4J appender：

```
<?xml version="1.0" encoding="UTF-8"    ?>
    <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
    <log4j:configuration debug="false"
xmlns:log4j="http://jakarta.apache.org/log4j/">
        <!-- Perf4J appenders -->
        <!--
            This AsyncCoalescingStatisticsAppender groups    StopWatch log
messages
```

```
                into GroupedTimingStatistics messages which it    sends on the
                file appender and perf4jJmxAppender defined    below
            -->
            <appender name="CoalescingStatistics"
                        class="org.perf4j.log4j.AsyncCoalescingStatisticsAppender">
                <!--
                  The TimeSlice option    means timing logs are aggregated every 10
secs.
                -->
                <param name="TimeSlice"    value="10000"/>
                <appender-ref    ref="fileAppender"/>
                <appender-ref    ref="perf4jJmxAppender"/>
            </appender>
            <!--
                This file appender is used to output aggregated    performance statistics
                in a format identical to that produced by the LogParser.
            -->
            <appender name="fileAppender"
class="org.apache.log4j.FileAppender">
                <param name="File"    value="perfStats.log"/>
                <layout    class="org.apache.log4j.PatternLayout">
                    <param    name="ConversionPattern" value="%m%n"/>
                </layout>
            </appender>
            <!--
                This JMX appender creates an MBean and publishes    it to the platform
MBean
        server by
                default.
            -->
            <appender name="perf4jJmxAppender"
      class="org.perf4j.log4j.JmxAttributeStatisticsAppender">
                <!-- The tag names whose    statistics should be exposed as MBean
attributes. -->
                <param name="TagNamesToExpose"
value="firstBlock,secondBlock"/>
                <!--
                  The    NotificationThresholds param configures the sending of JMX
notifications
```

```
                when statistic values    exceed specified thresholds. This config
states that
                the firstBlock max value    should be between 0 and 800ms, and
the secondBlock max
                value should be less    than 1500 ms. You can also set thresholds on
the Min,
                Mean, StdDev, Count and    TPS statistics - e.g.
firstBlockMean(<600).
            -->
            <param name="NotificationThresholds"
value="firstBlockMax(0-800),secondBlockMax(<1500)"/>
        </appender>


        <!-- Loggers -->
        <!-- The Perf4J logger. -->
        <logger name="org.perf4j.TimingLogger"    additivity="false">
            <level    value="INFO"/>
            <appender-ref    ref="CoalescingStatistics"/>
        </logger>


        <!--
          The root logger sends all log statements EXCEPT    those sent to the
perf4j
          logger to System.out.
        -->
        <root>
            <level    value="INFO"/>
            <appender    name="console"
class="org.apache.log4j.ConsoleAppender">
                <layout    class="org.apache.log4j.SimpleLayout"/>
            </appender>
        </root>
    </log4j:configuration>
```

除了 JMX 插件，Perf4J 也支持生成性能图表的画图 appender，使用前端的 Perf4J 画图 Servlet。定制的 csv 布局可以轻松的导入 Excel 或者其他电子表格应用。

使用@Profiled 注解简化性能日志

在代码中增加性能记录语句(通常的日志语句)的一个缺点是降低了代码的“信噪比”，难以在特定代码块中遵循严格的业务逻辑。为了改善这一不足，Perf4J 提供了@Profiled 注解，可以添加在需要记录执行时间的方法上，允许方法本身不添加 StopWatch 代码：

```
@Profiled(tag = "dynamicTag_{$0}")
public void profiledExample(String tagSuffix) {
    ... business logic only here
}
```

一旦添加了@Profiled 注解，Perf4J 的计时切面会通过一个面向切面的编程框架如 AspectJ 或者 Spring AOP 启用。这个切面在方法调用周围加入建立和停止 StopWatch 实例的字节码。计时切面甚至可以有选择的使用 AspectJ 的加载时编织 (load-time weaving) 功能。因此，通过使用加载时编织你可以保证那些没有启用性能监控的方法绝没有额外的负担。

一个简单的示例：基于 Web 的质数生成器

本示例将带你感受如何创建一个使用 Perf4J 库大多数功能的真实应用。你可以下载 perf4jPrimes.war 文件，然后 Servlet 容器中运行它。在 war 包中也包含该应用的源代码。

一切从简，本例子只包含两个主要的代码文件：primes.jsp 用于显示生成的质数和接受用户指定的参数，PrimeNumberGenerator 类包含真正的生成代码 (大部分委托给 java.math.BigInteger 类)。其中有三处使用了 Perf4J 计时代码块：

1. 在 primes.jsp 创建 Log4JStopWatch 统计整个页面的生成时间。
2. PrimeNumberGenerator.generatePrime()方法具有一个 Profiled 注解。
3. PrimeNumberGenerator.randomFromRandomDotOrg()也含有 Profiled 注解。

如果查看 WEB-INF/classes/log4j.xml 文件，你会看到启用了下面的 Perf4 功能：

1. 所有单独的 stop watch 日志都被写入标准输出(请注意你的 servlet 容器可能把标准输出定向到磁盘的某个文件中)。如果需要的话，你可以直接使用 LogParser。
2. AsyncCoalescingStatisticsAppender 被创建以汇总 stop watch 日志并传递给下游的

GraphingStatisticsAppenders 和 JmxAttributeStatisticsAppender。

3. 两个 GraphingStatisticsAppender 被创建，其中一个表示平均执行时间，另一个输出每秒事务数。

一旦在 Web 服务器上启动了该 Web 应用，你就可以通过 http://servername/<rootContextLocation>/primes.jsp 访问质数生成页面，其中 rootContextLocation 由你的 Web 服务器配置决定（当然，为了方便，你也可以通过 http://servername/<rootContextLocation>/PrimeNumberGenerator.java 查看 PrimeNumberGenerator 源代码）。在 primes.jsp 页面中，你会看到很多用于质数生成的不同参数。你可以改变参数，然后点击“生成质数”按钮，看看这些参数是如何影响质数产生时间的。在生成大量质数之后，你可以通过三种途径来查看 Perf4J 输出：

1. 原始的标准输出日志。
2. 通过 http://servername/<rootContextLocation>/perf4jGraphs 访问图形化 Servlet。你应该能够看到平均执行时间和每秒事务数。
3. JMX 监控也启用了。你可以通过 Java SDK 附带的 jconsole 应用查看 Perf4J MBean 值。这需要在启动 Web 服务器时，使用-Dcom.sun.management.jmxremote 命令行参数。然后，如果在运行 Web 服务 器的同一台机器上启动 jconsole 就可以在“MBeans”标签中看到 “org.perf4j.StatisticsExposingMBean.Perf4J”的数据值。

因为到现在为止你还没有启用任何 TimingAspects 支持@Profiled 注解，你能够看到的唯一 stop watch 输出就是“fullPageGeneration”标记。如果要启用 TimingAspects，你可以使用 AspectJ 加载时编织。你需要做的是，在启动 Web 服务器时使用 AspectJ weaving 代理命令行参数：

```
-javaagent:/path/to/aspectjweaver-1.6.1.jar
```

你可以在这里下载 jar 包：
http://mirrors.ibiblio.org/pub/mirrors/maven2/org/aspectj/aspectjweaver/1.6.1/aspectjweaver-1.6.1.jar

当代理启用时，你可以在“generatePrime”和“randomFromRandomDotOrg”标记中看

到 stop watch 的输出。

陷阱与最佳实践

很多监控应用的方法，不论是针对性能、稳定性还是语义正确性，都无法最有效的体现它们的意图。要么生成了太多的信息以至于难以分析这些数据，要么在需要信息的地方却得不到关键的数据。虽然所有的监控都需要一些额外的开销，但是性能监控应该对这些引入的开销格外小心。以下建议可以帮助你最大限度地利用 Perf4J，同时又将副作用降到最低：

1. 在判断需要分析哪些方法和代码块时，首先把重点放在关键点上。在企业应用中，每当遇到性能瓶颈时，都会存在很多“通常的疑点”：数 据库调用、运程方法调用、磁盘 I/O、针对大型数据集的计算操作。因此，你应该首先集中分析这些类型的操作。同时，因为这些操作花费几十甚至几百毫秒的时 间，Perf4J 所带来的额外开销相对于本地执行时间来说微不足道，在实践中可以忽略不计。事实上，这也是 Perf4J 故意使用 System.currentTimeMillis（而不是 System.nanoTime）计时代码块的原因之一：纳秒的粒度在这种企业应用代码块中意 义并不大。

2. Perf4J 把性能分析的工作委托给独立的线程或者进程。当使用 AsyncCoalescingStatisticsAppender 时，执行线程把日志事件推入到一个内存中的队列，由另外一个独立的线程发送这些日志 消息到下游 appender。因此，即使那些下游 appender 做了大量敏感的工作，如建立图表、更新 MBean 属性或者存储到数据库中，对计时的代码 块的影响微乎其微，而且与这些下游 appender 做的工作多少无关。类似的，当把计时日志写入文件用于解析和分析时（例如，使用 Unix tail 命令），对于计时进程的影响也只与它写日志所花费的时间有关，与 log 分析器的时间无关。

3. 不要忘记性能回归测试的好处。除了监测运行时的性能瓶颈，Perf4J 非常适合创建性能回归测试以判断代码更改是否对性能有显著影响（不论是积极或消极的）。通过创建一个原始代码的基准，你很快就能发现新代码对性能产生了何种影响。

4. 利用@Profiled 注解和 AspectJ 的加载时编织来决定哪些方法应该在部署时计时。如

果使用了 @Profiled 注解，你可以自由的在方法调用周围添加计时，然后决定在使用 AspectJ 的 aop.xml 配置文件进行部署时需要对哪些方法进行计时。没有计时的方法不 会被关注。虽然那些被计时的方法比直接在代码中使用 StopWatches 存在一些细微的额外开销（事实上 AspectJ 在计时方法的周围建立了一个闭 包），这些开销相对于那些需要花费几毫秒的方法来说是微不足道的。

5. 不要忘记应用程序中 JVM 之外执行的部分。举例来说，很多 Web 2.0 应用的大部分都是通过运行在客户端浏览器中的 JavaScript 实现，虽然 Perf4J 可以用于衡量运行在服务器端的方法（响应 AJAX 远程调用），但如果 JavaScript 执行性能下降，你仍然需要其他的客户端调试工具。

展望 Perf4J

Perf4J 目前正在积极的发展，新的功能层出不穷。举例来说，凭借新版本的 Perf4J，我们可以通过单独的配置文件指定要分析的方法，这样即使 无法获得某些方法的源代码也可以对其进行计时。我们始终将用户的反馈和需求放在第一位，如果你想打造它未来的功能，那现在就来尝试 Perf4J 吧。更重要 的是，Perf4J 在 Apache 2 协议下完全开源，代码都充分文档化，你可以随意扩展。

原文链接：http://www.infoq.com/cn/articles/perf4j

相关内容：

- New Relic 升级 RPM，改善了协作和集成性能
- 使用 Perf4j 简化应用分析
- NewRelic 提供免费的 Rails 监测器并增加新特性
- 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

[推荐文章]

Java 程序员学习 Flex 和 BlazeDS 的十三个理由

作者 Ryan Knight 译者 沙晓兰

本文列述了 13 个 Java 程序员应当学习 Flex 和 BlazeDS 的理由，讨论了为什么 Flex 结合 BlazeDS 是开发 RIA 的最佳组合之一。无论是高度交互的网站还是以 Java 为后端的企业应用，这项组合都是最佳选择之一。更重要的是，这项组合能同时为开发员和企业带来高回报（ ROI ）。

在阐述 Java 程序员应当学习 BlazeDS 的 13 条理由时，我以一个假想的苏打分派系统来展示如何让已有的 Java 程序转变为 RIA 应用。通过这个例子，我同时还会讲解到 BlazeDS 在已有 Java 应用或新建 Java 应用中的多种不同用法。

理由一：开源

Flex 软件开发工具箱（ SDK ）的核心是个开源框架，专门用来开发、维护那些在不同浏览器、不同操作系统下界面都相同的 RIA 应用。Flex 发布采用的是 Mozilla 公共许可证（ Mozilla Public License ）。编译后的 Flex 应用在 Adobe Flash 平台下运行。

BlazeDS 是连接 Flex 和 Java 的索桥，是项针对远程调用和消息传递的开源技术。在 Java 应用服务器上，它以 servlet 的形式存在，因此可以在任何标准 Java 网络应用中运用它。BlazeDS 以 LGPL（ Lesser GNU Public License ）公共许可证书发布。在发布 BlazeDS 的同时，Adobe 还公布了 AMF（ ActionScript Message Format ）规格说明，BlazeDS、Java 和 Flex 客户端间以这种简洁的二进制格式实现通信。

理由二：完善的社区支持

Flex 社区非常活跃，社区贡献了大量项目。Flex.org，这个配以社区新闻的 Adobe 站点几乎每天都有新的社区贡献；Yahoo!上的 Flex 用户组的成员也已经超过了 11000。

再比如 Google Code 上的 Flexlib 项目，已经提交了大量的开源 UI 组件。Swiz 和 Mate 项目贡献了优化事件处理的框架；还有 Gorilla Logic 贡献了自动化 UI 测试的 Flex Monkeym 项目。

理由三：带来广阔的就业前景

据 Adobe 的 Flex“传道士”——James Ward 看来，Flex 高级开发员的市场需求非常大，学习 Flex 能让你拥有极具市场竞争力的开发技能。

理由四：更高的业务效益回报

总体上，开发企业 web 应用不是个轻松的活，这基本上是众所周知的事实。Flex 和 BlazeDS 提供的不仅仅是功能强大的开发工具，而且开发技术本身相对也非常简单。开发效率可以得到大幅度的提升，产品因此可以很快推向市场。Flex 和 Flash 带来的用户体验也相对更有魅力，对增加流量、提高用户转化率（conversion rate）很有帮助。

很经典的一个例子是 Borders 连锁书店。他们最近发布了带有“魔法书架”的新网站，这个网站采用 Flash 接口来模拟书籍借阅的过程。Borders 发现这一模拟借阅非常明显地提到了用户转换率：“借助这个 Flash 驱动的接口，用户可以浏览书籍、DVD 和 CD 的封面，用户转换率比其他没有此项功能的网站高出 62%”。

理由五：Flex 是第一个专门为创建 UI 而设计的语言

大部分语言都不是在第一时间设计其对 UI 的支持。Java 中 Swing 包的实现刚好是个很好的证明。也就是这个原因，很多像捆绑数据这样的简单动作在 Swing 当中的实现就非常痛苦。用 Swing 最大的问题在于，要想提高开发效率就必须要对其 API 了如指掌。

Flex 刚好相反，它是专门为创建 web UI 而设计的。正如 Bruce Eckel 所说，Flex 是第一个针对 UI 开发的领域特定语言（DSL）。用 Flex 构建 UI 比其它诸如 JSP、JSF、Swing 等技术简便得多。语言本身糅合了数据绑定、事件处理、控件布局以及其它一些 UI 常用开发技巧，

就算对语言没有深刻的理解也不会影响开发效率。

理由六：编程风格近似于 Java

你可以继续使用现有的 Java 开发工具来开发 Flex 应用。当然也可以采用 SDK 中携带的免费命令行工具，Adobe Flex Builder（一个 Eclipse 插件），或最近的 IntelliJ IDEA 8。

Flex 提供的是一个有状态环境，在这个环境中，数据从客户端加载。这种编程模式更像是开发桌面客户端而非 HTML 编程，这种风格对于用过 Java Swing 编程的开发员来说应该是相当熟悉。

Flex 是 MXML(类似 XML 的 UI 标记语言)和 Adobe ActionScript(面向对象的解析语言) 的结合体。鉴于这种结合方式，Flex 编程与 Java 非常相似，因为两者用的都是熟知的面向对象的概念。

最理想的开发环境是把 Flex 应用创建在 web 部署文件夹下。这样一来，每次更新应用之后都不需要重新部署，只要在浏览器下刷新一下就可以了。用 Flex 和 BlazeDS 开发后，开发效率绝对比之前有很大的提升。

理由七：BlazeDS 可以在任何 Java 应用服务器上运行

BlazeDS 目前已发布了多个版本 ,其中的 turnkey 版本还包含了为 BlazeDS 配置的 Apache Tomcat。本文中，我用的是二进制发布版本，其中含有一个 WAR 用来展示如何把应用部署到各种应用服务器上去。不用这个 WAR 的话，你也可以从中提取 JAR 文件放到自己的项目中去。关于安装 BlazeDS 的各种选项内容，可以参见 BlazeDS 的 wiki。

这里举一个简单的例子，比方说要在已有的一个简单的苏打调配系统中应用 BlazeDS。你只要把 JAR 文件放到项目文件夹下，然后就可以在应用里直接用 BlazeDS，可以部署到能够部署应用的任何地方。

在项目中添加 BlazeDS，只需要完成下面两个步骤：

1. 解压缩 BlazeDS WAR 文件的内容：jar xvf blazeds.war。
2. 把 JAR 文件都拷贝到项目的 lib 文件夹下：cp -R WEB-INF/lib /sodaSample。

理由八：可以在已有 Java 应用中运用

比方说这个简单的苏打调配系统，假设你想要扩展这个已开发好的服务，让其它 Flex 应用可以远程调用。在现成的应用中配置 BlazeDS 的基本步骤有：

1. 修改 WEB-INF/flex 文件夹下的 BlazeDS 配置文件

2. 在该应用对应的 web.xml 文件里定义 MessageBrokerServlet 和 session 监听器

配置好 BlazeDS 之后，再把苏打调配服务添加到 BlazeDS 远程配置文件里，Flex 客户就能远程调用了。这个过程通过在配置文件里定义一个目的地（destination）、一个或多个信道（channel）来传输数据。基本的 AMF 信道定义在 services.xml 文件里。下面这段配置在 remoting-config.xml 里定义了目的地（destination）：

```
<destination id="sodaService" channels="my-amf">
    <properties>
        <source>com.gorillalogic.sodaSample.SodaService</source>
    </properties>
</destination>
```

通过在远程调用配置文件里定义端点（endpoint），Flex 客户端就可以调用任何一个基本的 Java 服务。

要是想把 Java 数据模型也传送到 Flex 客户端的话，只要在 ActionScript 类中定义好两者间的映射：

```
[Bindable]
[RemoteClass(alias="com.gorillalogic.sodaSample.SodaModel")]
```

这段代码告诉 Flex，在远程调用的服务返回 SodaModel 的时候，把它映射到 Flex 的 SodaModel。本例中的 Flex 客户端显示的就是如何调用这个 Java 服务。调用返回一个已经填写好预定信息的 SodaModel：

```
public function callSodaService():void {
    var sodaType:String = type.text;
    var sodaCount:int = parseInt(cnt.text);
    var flag:Boolean = preOpen.selected;
    remoteObject.getSoda(sodaType, sodaCount, flag);
```

```
    }

    private function resultHandler(event:ResultEvent):void {
    var sodaModel:SodaModel = event.result as SodaModel;
    }
```

Flex 返回的结果是通用的 result 变量，可以直接映射到你的 SodaModel。这里我就不深入讨论怎么实现映射了 ,但其中值得提到的是要在编译配置里声明 services-config.xml 路径，像这样：

```
    -locale en_US
-services=/nsource/sodaSample/web/WEB-INF/flex/services-config.xml -context-root /
```

如果不添加这个路径的话，你的 Flex 客户端就没发找到 Java 服务。同样的方式，你还能把一个对象从客户端传递回服务器端。比如，你可以把一个空的 soda model 发回服务器（审校注：原文这里写的是客户端，根据上下文判断这里应该是服务器端）。

理由九：可以通过 Java 来扩展和修改 BlazeDS

假如你想添加特殊的日志来记录苏打调配服务被调用的情况，那么你可以扩展标准的 Java 适配器来添加日志功能。

首先，添加一个继承了 JavaAdapter 的 Java 类：

```
    import flex.messaging.services.remoting.adapters.JavaAdapter.
    public class TimingJavaAdapter extends JavaAdapter    {
```

其次，重载 invoke()方法：

```
    public Object invoke(Message message) {
            RemotingMessage remotingMessage = (RemotingMessage) message;
            String operation = remotingMessage.getOperation();
            String destination = remotingMessage.getDestination();

        Logger.info("calling " + operation + " on destination " + destination);
        Object data = super.invoke(message);
        return data;
    }
```

这个方法中，你可以看到调用之后的操作和调用的目的地（destination）。这种方法也

能用来处理其它一些问题，比如记录向服务器发送调用需要多长时间。

理由十：HTML 和 JSP 也能调用 BlazeDS

从 HTML 和 JSP 也能调用 BlazeDS，这种调用有几种不同的实现方式，比如通过 Browser Manager 或 fflashVarsf 来实现。Flex 应用能够读取由 HTML 页面设置的 fflashVarsf。

比方说你想要通过 HTML 页面来发送你的用户名和准备预定的苏打类型，你可以在 HTML 页面这样设置 flashVars：

```
    <object id='SodaSample' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab'
height='100%' width='100%'>
                <param name='src' value='SodaSample.swf'/>
                <param name='flashVars' value='username=ryan&type=coke'/>
                <embed name='mySwf' src='SodaSample.swf'
pluginspage='http://www.adobe.com/go/getflashplayer' height='100%' width='100%'
   flashVars='username=ryan&type=coke'/>
    </object>
```

然后，在 Flex 应用中，你可以通过读取应用参数来获取这些变量：

```
    var username:String;
    if (Application.application.parameters.hasOwnProperty("username")) {
        username = Application.application.parameters.username;
    }
```

理由十一：Flex 和 BlazeDS 的数据传输性能远胜于其它 Ajax 解决方案

目前使用的远程过程调用（RPC）都默认选择 AMF 二进制协议。AMF 是个开放的标准，而且相当快。James Ward 曾举例比较过多种远程调用解决方案。尽管其它 Ajax 技术——比如 Dojo——已经能够快速处理几百行的数据，但是用 Flex 和 BlazeDS 的话可以轻松搞定成千上万行。（请参考 James Ward's census ,可以了解下各种不同的 RIA 数据加载技术的测评。）

理由十二：Java 客户端能够直接调用 BlazeDS

最新发布的 BlazeDS 当中含有一个 Java 的 AMF 类，通过这个类，你可以在 Java 客户端直接调用 BlazeDS 服务器。对于单元测试和加载测试来说，BlazeDS 的这种调用方式非常实用。

理由十三：Spring 下也能用

Adobe 和 Spring 互相联手，尝试将双方项目集成起来。他们发布的第一个 Spring–BlazeDS 集成版本就向大家展示了他们的良苦用心。Spring Bean 能够以远程服务的方式被调用，因此可以清除很多重复的配置文件。更多这方面的相关信息，可以参考该项目的主页。

结论

开源的 BlazeDS 创建在 Java 基础上，无论是对新的还是已有的 Java 服务器项目来说都是个很好的选择。Flex、BlazeDS 技术能够提供高性能的远程通信，支持 Flex 和 Java 间的对象映射，因此是 RIA 开发的理想选择。Flex 和 BlazeDS 的开发新手，如果曾经是 Java 开发员的话，会发现整个开发过程效率非常高，而且很容易掌握。

Flex 加 BlazeDS 还是开发大型 Java 企业应用的理想选择。我们组开发的上个项目中，应用涉及到 50 多个不同的界面，而且服务器和客户端之间需要规律性地互传几千行的代码。这类应用几乎没法通过传统的 Ajax 技术来实现。但是在引入了 Flex 和 BlazeDS 之后，我们在年内就发布了第一个版本。看，这就是这对动态组合为你的应用开发项目带来的过人之处。

原文链接：http://www.infoq.com/cn/articles/java-flex-blazeds

相关内容：

- Blaze Data Services 还是 LiveCycle Data Services？
- 使用 Grails 和 Flex 开发 JEE 应用
- Merapi 项目利用 Java 扩展 Adobe Air 的桌面功能
- Doug McCune 谈 Flex 开发
- 利用 Clear Toolkit 连接 Flex 与 Java 开发

[推荐文章]

Mock 不是测试的银弹

作者 胡凯

开发者编写高质量测试的征途上可谓布满荆棘，数据库、中间件、不同的文件系统等复杂外部系统的存在，令开发者在编写、运行测试时觉得苦恼异常。由于外部系 统常常运行在不同机器上或者本地单独的进程中，开发者很难在测试中操作和控制它们。外部系统以及网络连接的不稳定性（外部系统停止响应或者网络连接超 时），将有可能导致测试运行过程随机失败。另外，外部系统缓慢的响应速度（HTTP 访问、启动服务、创建删除文件等），还可能会造成测试运行时间过长、成 本过高。种种问题使开发者不断寻找一种更廉价的方式来进行测试，mock 便是开发人员解决上述问题时祭出的法宝。mock 对象运行在本地完全可控环境内，利用 mock 对象模拟被依赖的资源，使开发者可以轻易的创建一个稳定的测试环境。 mock 对象本地创建，本地运行的特性更是加快测试的不二法门。

我所在团队设计开发的产品是持续集成服务器，产品特性决定了它需要在各个平台（ Windows, Mac, Linux 等）与各种版本管理工具(svn, mercurial,git 等)、构建工具(ant, nant, rake 等)进行集成，对于外部系统的严重依赖让我们在编写测试时遇到了很多困难，我们自然而然的选用了 JMock 作为测试框架，利用它来隔离外部系统对 于测试的影响，的的确确在使用 JMock 框架后测试编写起来更容易，运行速度更快，也更稳定，然而出乎意料的是产品质量并没有如我们所预期的随着不断添加 的测试而变得愈加健壮，虽然产品代码的单元测试覆盖率超过了 80%，然而在发布前进行全面测试时，常常发现严重的功能缺陷而不得不一轮轮的修复缺陷、回归 测试。为什么编写了大量的测试还会频繁出现这些问题呢？ 在讨论之前先来看一个真实的例子：

我们的产品需要与 Perforce(一种版本管理工具)进行集成，检测某段时间内 Perforce 服务器上是否存在更新 ,如果有 ,将更新解析为 Modification 对象。将这个需求反应在代码中，

便是首先通过 Perforce 对象检测服务器更新，然后将标准输出(stdout)进行解析：

```
        public class PerforceMaterial {
            private Perforce perforce;
            .....
            .....
            public List findModifications(Date start, Date end) {
                String changes = perforce.changes(start, end);       //检测更新，返回命令行标准输出（stdout）
                List modifications = parseChanges(changes);//将标准输出解析为Modification
                return modifications;
            }

            private List parseChanges(String output) {
                 //通过正则表达式将 stdout 解析为 Modification 对象
            }
        }

        public class Perforce {
            public String changes(Date start, Date end) {
                  //通过命令行检测更新，将命令行标准输出（stdout）返回
            }
        }
```

相应的 mock 测试也非常容易理解：

```
            .....
            .....
            @Test
            public void shouldCreateModifiationWhenChangesAreFound() {
                final String stdout = "Chang 4 on 2008/09/01 by p4user@Dev01 'Added build.xml'"; //设置标准输出样本
                final Date start = new Date();
                final Date end = new Date();

                context.checking(new Expectations() {{
```

```
                one(perforce).changes(start, end);
                will(returnValue(stdout));
            }});//设置 perforce 对象的行为，令其返回设定好的 stdout

            List list = perforceMaterial.findModifications(start, end);//调用被测方法

            assertThat(list.get(0).revision(), is("4"));
            assertThat(list.get(0).user(), is("p4user@Dev01"));
            assertThat(list.get(0).modifiedTime(), is("2008/09/01"));
        }
```

测试中的 stdout 是在真实环境下运行 Perforce 命令行所采集的标准输出(stdout)样本，通过 mock perforce 对象，我们可以轻易的控制 changes 方法的返回值，让验证解析逻辑的正确性变得非常容易，采用 mock 技术使开发者无需顾忌 Perforce 服务器的存在与否，而且可以采用不同的 stdout 来覆盖不同的情况。然而危机就在这看似完美的测试过程中被埋下了，事实上 Perforce stdout 中的时间格式会依用户环境的设定而变化，从而进一步导致 parseChanges 方法中的解析逻辑出现异常。由于测试中的 stdout 全由假 设得来，并不会依照环境变化，即便我们将测试跑在多种不同的环境中也没能发现问题，最终在产品环境才由客户发现并报告了这个缺陷。

真实 perforce 对象的行为与测试所使用的 mock 对象行为不一致是出现上述问题的根本原因 ,被模拟对象的行为与真实对象的行为必须完全一致称之为 mock 对象的行为依赖风险。开发者对 API 的了解不够、被模拟对象的行为发生变化（重构、添加新功能等修改等都可能引起被被模拟对象的行为变化）都可能导致错误假设（与真实对象行为 不一致），错误假设会悄无声息的引入缺陷并留下非法测试。非法测试在这里所代表的含义是，它看起来很像测试，它运行起来很像测试，它几乎没有价值，它几乎 不会失败。在开发中，规避行为依赖风险最常见的方法是编写功能测试，由于在进行 mock 测试时，开发者在层与层之间不断做出假设 ,而端到端的功能测试由于 贯穿了所有层 ,可以验证开发者是否做出了正确的假设，然而由于功能测试编写复杂、运行速度慢、维护难度高 ,大部分产品的功能测试都非常有限。那些通过 mock 测试的逻辑，便如埋下的一颗颗定时炸弹，如何能叫人安心的发布产品呢？

《UNIX 编程艺术》中有一句话“先求运行，再求正确，最后求快”，正确运行的测试是

高质量、可以快速运行测试的基础，离开了正确性，速度和隔离性都是无 根之木，无源之水。那么采用真实环境就意味着必须承受脆弱而缓慢的测试么？经历了一段时间的摸索，这个问题的答案渐渐清晰起来了，真实环境的测试之所以痛 苦，很大程度上是由于我们在多进程、多线程的环境下对编写测试没有经验，不了解如何合理的使用资源（所谓的资源可能是文件、数据库中的记录、也可能是一个 新的进程等），对于我们，mock 测试作为“银弹”的作用更多的体现在通过屏蔽运行在单独进程或者线程中的资源，将测试简化为对大脑友好的单线程运行环境。在修复过足够多的脆弱测试后，我们发现了编写健壮测试的秘密：

要设计合理的等待策略来保守的使用外部系统。很多情况下，外部系统处于某种特定的状态是测试得以通过的条件，譬如 HTTP 服务必须启动完 毕，某个文件必须存在等。在编写测试时，开发者常常对外部系统的估计过于乐观，认为外部系统可以迅速处于就绪状态，而运行时由于机器和环境的差异，结果往 往不如开发者所愿，为了确保测试的稳定性，一定要设计合理的等待策略保证外部系统处于所需状态，之所以使用"等待策略"这个词，是因为最常见”保证外部系 统处于所需状态“的方法是万恶的"Thread.sleep"，当测试运行在运算速度/网络连接速度差异较大的机器上时，它会引起随机失败。而比较合理的方法是利用轮询的方式查看外部系统是否处于所需状态（譬如某个文 件存在、端口打开等），只有当状态满足时，才运行测试或者进行 Assertion，为了避免进入无限等待的状态，还应该设计合理的 timeout 策略，帮 助确定测试失败的原因。

要正确的创建和销毁资源漠视测试环境的清理也常常是产生脆弱测试的原因，它主要表现在测试之间互相影响，测试只有按照某种顺序运行时才会成功/失败，这种问题一旦出现会变的非常棘手，开发者必须逐一对有嫌疑的测试运行并分析。因此，有必要在开始时就处理好资源的创建和销毁，使用资源时应当本着这样一个原则：谁创建，谁销毁。 junit 在环境清理方面所提供的支持有它的局限性，下面的代码是使用资源最普遍的方式：

```
    @After
public void teardown() {
    //销毁资源 A
    //销毁资源 B
}
```

```
@Test
public void test1() {
    //创建资源 A
}

@Test
public void test2() {
    //创建资源 B
}
```

为了确保资源 A 与资源 B 被正确销毁，开发者必须将销毁资源的逻辑写在 teardown 方法中，然而运行用例 test1 时，资源 B 并未被创建，所以必须在 teardown 中同时处理资源 A 或 B 没有被创建的情况，由于需要销毁的资源是用例中所使用资源的并集，teardown 方法会快速得膨胀。由于这样的原 因，我在开源项目 junit-ext 中加入了对 Precondition 的支持，在测试用例运行前，其利用标注所声明的多个 Precondition 的 setup 方法会被逐一调用来创建资源，而测试结束时则调用 teardown 方法销毁资源。

```
   @Preconditions({ResourceIsCreated.class, ServiceIsStarted.class})
@Test
public void test1() {
        //在测试中使用资源
}

public class ResourceIsCreated implements Precondition {
    public void setup() {
            //创建资源
    }
    public void teardown() {
            //回收资源
    }
}

public class ServiceIsStarted implements Precondition {
     public void setup() {
            //创建资源
```

```
        }
        public void teardown() {
                //回收资源
        }
}

public interface Precondition {
        void setup();

        void teardown();
}
```

这个框架可以更好的规范资源的创建和销毁的过程 ,减少因为测试环境可能引起的随机失败，当然这个框架也有其局限性，在 ResourceIsCreated 和 ServiceIsStarted 之间共享状态会比较复杂，在我们的产品中，Precondition 大多用于启动新进程，对于共享状态的要求比较低， 这样一套机制就非常适合。每个项目都有其特殊性，面对的困难和解决方案也不尽相同，但在使用资源时如果能遵守“谁创建，谁销毁”的原则，将会大大减小测试 之间的依赖性，减少脆弱的测试。

要设计合理的过滤策略来忽略某些测试。我们很容易在项目中发现只能在特定环境下通过的测试，这个特定环境可能是特定的操作系统，也可能是特 定的浏览器等，之所以会产生这些测试通常是开发者需要在源码中进行一些特定环境的 hack，它们并不适合在所有环境下运行，也无法在所有环境中稳定的通 过，因此应该设计一套机制可以有选择的运行这些测试，junit 的 assumeThat 的机制让我再次有点失望，本着自己动手丰衣足食的原则，在 junit-ext 我添加了利用标注来过滤测试的机制，标注了 RunIf 的测试仅当条件满足时才会运行 ,除了内置一些 Checker ,开发者也可以很方便的开发自己的 Checker 来适应项目的需要。

```
@RunWith(JunitExtRunner.class)
public class Tests {
    @Test
    @RunIf(PlatformIsWindows.class) //test1 仅运行在 Windows 环境下
    public void test1() {
    }
}
```

```
public class PlatformIsWindows implements Checker {
    public boolean satisfy() {
        //检测是否 WINDOWS 平台
    }
}
```

要充分利用计算资源而不是人力资源来加快测试。对于加快测试，最普遍也最脆弱的方法是利用多线程来同时运行多个测试，这个方法之所以脆弱， 是因为它会让编写测试/分析失败测试变的异常复杂，开发者必须考虑到当前线程在使用资源时，可能有另一个线程正要销毁同一个资源，而测试失败时，也会由于 线程的不确定性，导致问题难于重现而增加了解决问题的成本。我认为一个更好的实践是在多台机器上并发运行测试，每台机器只需要运行(总测试数/机器数)个 测试，这样所花费时间会近似减少为(原本测试时间/机器数)。相对与购置机器的一次性投入，手工优化的不断投入成本更高，而且很多公司都会有闲置的计算资 源，利用旧机器或者在多核的机器上安装虚拟机的方式，可以很经济的增加计算资源。在项目开发的业余时间 ,我和我的同事们一起开发了开源的测试辅助工具 test-load-balancer。在我们的项目中 ,通过它将需要 90 分钟的测试自动划分为数个 10 分钟左右的测试在多台虚拟机上并发运行，很好的解决了速度问题。

对 mock 追本溯源 ,我们会发现它更多扮演的是设计工具的角色而不是测试工具的角色，mock 框架在设计方面的局限性李晓在《不要把 Mock 当作你的设计利器》一文中已经谈的很透彻，在此不再赘述。Mock 不是测试的银弹，世上也并无银弹存在，测试实践能否正常开展的决定性因素是团队成员对测试流程，测试方法的不断改进而不是先进的测试框架。不要去依赖 mock 框架，它的强制约定常常是你改进设计和添加功能的绊脚石，改善设计，依赖一个简洁的代码环境，依赖一套可靠的测试方法才是正途。从意识到 mock 测试带来的负面影响，到从滥用 mock 的泥潭中挣扎出来，我们花费了很多时间和经历，希望这些经验可以对同行们能有所借鉴，有所启发。

Mock，还请慎用。

写在最后

在 ThoughtWorks 工作的三年经历中，对于 mock 框架的使用从无到有，到滥用到谨慎。三年间，和李晓、陶文，李彦辉，Chris Stevenson 等人反复讨论和辩论使我对 mock 有了更多的理解。这篇文章反反复复改了许多稿，在此对陈金洲和李默的耐心帮助致谢。最后，没有与 InfoQ 的编辑李剑和霍泰稳在 Twitter 上的一番谈话，也就没有这篇文章。

作者简介：

胡凯，ThoughtWorks 公司的敏捷咨询师，近 2 年一直从事持续集成工具 Cruise 以及 CruiseControl 的设计开发工作。 创造和参与了开源测试框架 junit-ext 和 test-load-balancer，对于 Web 开发，敏捷实践，开源软件与社区活动有浓厚的兴趣，可以访问他的个人博客进行更多的了解。

相关阅读

[ThoughtWorks 实践集锦（ 1 ）] 我和敏捷团队的五个约定。

[ThoughtWorks 实践集锦（ 2 ）] 如何在敏捷开发中做好数据迁移。

[ThoughtWorks 实践集锦（ 3 ）] RichClient/RIA 原则与实践（ 上 ）、（ 下 ）。

[ThoughtWorks 实践集锦（ 4 ）] 为什么我们要放弃 Subversion。

[ThoughtWorks 实践集锦（ 5 ）] “持续集成”也需要重构。

原文链接：http://www.infoq.com/cn/articles/thoughtworks-practice-partvi

相关内容：

- Web 自动化测试工具 Selenium 1.0 正式发布:Chrome 支持+用户指南
- 使用 JsUnit 和 JSMock 的 JavaScript 测试驱动开发
- 开源自动测试框架 Tellurium
- 选择哪个 SOA 测试工具？
- .NET 平台下 Web 测试工具横向比较

[新品推荐]

Ruby 1.9.2 的发布计划宣布

Ruby 1.9.2 的发布计划现在已经宣布了，包括时间表以及一些可能会添加的特性，例如给 Ruby 添加 SQLite 支持。

原文链接：http://www.infoq.com/cn/news/2009/06/ruby192-plans

ZK 发布 3.6.2 版本：性能提升，include 模式

著名开源 Ajax 框架 ZK 近日发布 3.6.2 版本，该版本引入的特性包括性能的提升（尤其针对 IE6）以及新添包含（include）模式，另外还有针对一些缺陷的修复。

原文链接：http://www.infoq.com/cn/news/2009/06/ZK3.6.2

Eclipse Galileo 发布啦

今天是 Java 社区大喜的日子，因为 Eclipse 基金会发布了 Eclipse Galileo，与之相伴的还有 33 个项目，包括 Eclipse JDT。除了 InfoQ 此前所报道过的新特性外，此次发布的 Galileo 还包含了 PHP 开发工具项目以及 Modelling 项目和持久层 EclipseLink 项目（即大家所熟知的 Oracle TopLink）。

原文链接：http://www.infoq.com/cn/news/2009/06/eclipse-galileo-released

Rip：一个全新的 Ruby 包管理系统

Rip 是 GitHub 团队开发的一个全新的 Ruby 包管理系统。它能够管理不同的安装源，例如目录、文件，Git 版本库以及 RubyGems 等。

原文链接：http://www.infoq.com/cn/news/2009/06/rip

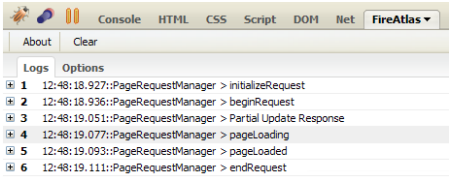
Opera Unite：超越强大

Opera Unite

志在改变互联网的 Opera Software 于近日发布了其最新版的浏览器：Opera 10 Beta 1，该浏览器使用了一项称为 Opera Unite 的服务器端技术，用户可以凭借该项技术直接进行互联以共享数据和交流而无需借助于中间服务。

原文链接：http://www.infoq.com/cn/news/2009/06/Opera-Unite

FireAtlas：ASP.NET AJAX 查看器



FireAtlas 是 Firebug 的一个扩展，提供了 ASP.NET AJAX 调试功能，可以对 PageRequestManager 进行跟踪并检查局部更新。

原文链接：http://www.infoq.com/cn/news/2009/06/fireatlas


1kg.org
多背一公斤
爱自然 | 更爱孩子

[封面植物]

异形玉叶金花（白纸宝）

国家一级保护植物，分布区极狭窄，自 1936 年首次在广西大瑶山采得标本后几近绝迹，近几年在我省东南部重新发现。

形态特征

攀援灌木，丛生，高约 2.5 米，小枝灰褐色，初有贴伏疏柔毛；皮孔明显。叶对生，薄纸质，椭圆形至椭园状卵形，长 13-17 厘米，宽 7.5-11.5 厘米，先端渐尖，基部楔形，两面疏被柔毛，侧脉 8-10 对，叶柄长 2-2.5 厘米，稍被短柔毛；托叶早落。顶生三歧分枝的聚伞花序，被短 柔毛，苞片早落，小苞片长约 1 厘米，迟落；花有毛，花梗长 2-3 毫米，萼筒长约 5 毫米，裂片 5，增大成花瓣状，白色，长 2-4 厘米。宽 1.5-2.5 厘 米。边缘及脉略被毛，花冠长约 1.2 厘米，径约 4 毫米，上部扩大，5 裂，内面有金黄色粉末状小点；雄蕊 4-5，子房 2 室；花柱长约 6 毫米，柱头 2。浆果肉 质，长 6-10 毫米，径 6-8 毫米。

地理分布及省内资源现状

我国特有种。仅见于广西大瑶山及贵州东南部的从江加叶、黎平岩洞、榕江乐里傜人溪和平永，荔波莫干等局部极其狭窄地区。异形玉叶金花几乎都分布在阳光较好，土壤条件较优越的山体下部农地，路边或村寨周围，就说明该种生态幅较宽窄，对环境条件较为苛刻；致使分布区不连续，面积很小，个体数量很稀见。

异地保护及繁殖方法

该种到目前为止 ,尚未进行人工栽植方面研究 ,可用种子育苗或扦插繁殖。就野外所见，果实易受虫害，种子极稀少。

本月推荐编辑

张龙，同济大学软件工程硕士，现就职于理光软件研究所。主要从事文档工作流和协同知识解决方案的研发工作。热衷于 Java 轻量级框架的研究，对敏捷方法很感兴趣。目前对 Ruby 及 Flex 产生浓厚兴趣。曾有若干年的 J2EE 培训讲师经历。

张龙现在为 InfoQ 中文站 Java 社区的编辑，被誉为“最勤劳”的编辑，不但为 InfoQ 中文站贡献了大量优质的内容，还参与撰写/翻译多本技术类著作。

大家好，我是 InfoQ 中文站 Java 社区的编辑张龙，很高兴能有这个机会与大家一起交流。从加入 InfoQ 中文站这个大家庭到现在已将近两年的时间了。两年弹指一挥间，还记得那个秋天，怀着忐忑的心情给 InfoQ 中文站总编泰稳发的邮件，申请加入这个大家庭；还记得第一篇试译的新闻，反反复复修改了无数遍，但最后还是有一句话漏译了；还记得 Java 社区首编宋玮对我的译稿所作的详细修改与指点，还记得为了能按时完成内容的翻译而熬夜加班的情景；还记得....。太多太多了，往事历历在目，却又不知从何说起。我这两年的经历可以用丰富多彩来形容。从来没有想过能有机会与这样一群优秀的人共事，这对我现在，乃至今后都将是一笔宝贵的财富，但不能不提的一个遗憾就是：虽然共事 2 年，但我至今还没有见过这个大家庭中的任何一员，有时也在感叹互联网给我们生活所带来的翻天覆地的变化啊，必须的：-）

从之前没有任何翻译经验的毛头小伙，到现在能及时把最新资讯分享给广大读者的“合格”（姑且这么称呼吧，希望大家少拍砖啦）译者，其中的甘苦只有经历过的人才能品出个中滋味。凭借着在 InfoQ 中文站的锤炼，我也参与翻译了两本技术书籍：《Dojo 构建 Ajax 应用程序》以及《Pro Spring 2.5 中文版》（即将由人民邮电出版社图灵公司出版）。可以这么说，InfoQ 中文站给了我别样的生活，充满了缤纷色彩的生活。

InfoQ 中文站从诞生到现在也两年有余了，两年的时间让这个蹒跚学步的孩子逐渐成长为国内知名的高端技术站点。其使命就是将国外最新的技术资讯及时传递到国内：成功举办

过多场的 QClub、今年四月于北京举办的 QCon 都令国内的技术社区耳目一新，如沐春风。这一切的一切都证明了我们的价值，见证了 InfoQ 中文站蓬勃向上的发展势头——时刻关注企业软件开发领域的变化与创新。

正值《架构师》创刊号发布之际，我谨代表 InfoQ 中文站全体工作人员为《架构师》献上深深的祝福：愿此面向企业架构师及项目经理的电子刊物能为您的企业带来价值，成为您的良师益友。

最后，我们非常欢迎您也能成为我们的一员，开放，自由的 InfoQ 中文站永远敞开双臂，拥抱每一位朋友。

架构师

每月 8 号出版

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

《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