时刻关注企业软件开发领域的变化与创新

架构师

2月 ARCHITECT

特别专题

SOA这些年，我们的沉淀

关于ESB实施的几点建议

国内工作流厂商分析

REST服务开发实战

SOA分析和设计中的错误处理要点

Silverlight之重

使用Dijit实现界面组件化开发

用Dojo实现Ajax请求：XHR、跨域、及其他

2011年的HTML 5愿景：Michael Mullany专访

Java线程：基本概念、可见性与同步

InfoQueue

Enterprise Software Development Community

每月8号出版

篇首语 | Editor's letter

知其然，知其所以然

为什么那么多优秀的框架没有一个是中国人设计出来的？为什么我们的网站都在大量翻译外国人的内容（包括 InfoQ）？为什么我们的技术似乎总是跟在别人的后面？这其中当然有很多原因，创新意识，社会大环境等都是可能的理由……然而，我要谈的一个原因是"知其然，不知所以然"。我们的杂志是《架构师》，所以我谈的话题不能偏题太远。我的观点是——要做好架构，就要知其然知其所以然。

单纯看事物表面是不能做好架构的。首先，你会在技术选择面前容易丢失自我。如果仅看事物表面，你就会被各种宣传牵着鼻子走，REST、NoSQL、HTML5、移动计算、Cloud……你能统统拿来主义吗？不能，所以只能了解其皮毛；让我们感觉旧的没有学好，新的又冒出来了，实在是目不暇接，防不胜防。其次，我们会错失事物/技术的本质。只看事物的表面往往造成“一叶障目，不见泰山“，平时很爱看《动物世界》，有些动物有着天然的保护色，不论是为了躲避天地，还是为了欺骗猎物，其结果都是一样，让对方忽视其本质存在。技术上也是一样，如果只看到表面的宣传，就无法深入技术细节，从而无法正确地完成架构设计。这正是许多 SOA 实施无法落地或踉跄落地的原因。

所谓条条大路通罗马，对于同一个问题，可选的编程语言、方法、框架、平台多之又多。架构师的职责是找到那条满足客户最本质需求的最优解。洞悉深层次需求、知其所以然，是架构师们必须培养的两项基本素质。只有了解深层次需求，才能做好每一个架构决策、才能不受技术潮流、个人私欲（如：我最好用 SOAP）的牵制；只有直其所以然，才能分辨出谁优谁劣。打个比方，为什么有时选择 JMS 而非 Native MQ，优势却要反过来？为什么此处选择 SOAP，而那里却选择 JMS 呢？要想回答好这些问题，二者不可或缺。

开篇的那一串问题有点偏激，我们不却乏优秀聪慧激情进取之辈，他们有崇高的理想，并且在路上。InfoQ 做为社区的推动者，正竭尽全力为本土优秀的"知其所以然"的人们创造展示自己的舞台。从去年开始，InfoQ 大力推行本地新闻，本地深度内容。从 2011 年开始，《架构师》的专题技术部分全部选用本地原创。本期专题"SOA 这些年，我们的沉淀"的几篇原创文章里，读者即将看到，在实施 SOA 的过程中，我们有太多的选择，但是如何才能选择最优的路呢？里面有你的答案。InfoQ 的读者们，“知其所以然”，我们在路上，你呢？

本期主编：马国耀

目　　录

[篇首语]

知其然，知其所以然 1

[人物专访]

江枫谈淘宝“双十一”事件中的数据库架构优化 4

[热点新闻]

2010 年 INFOQ 中文站高性能架构内容回顾：在路上 10

NOSQL 开篇——为什么要使用 NOSQL 13

GIT 历险记（一） 17

APPLE、GOOGLE、微软大比拼 22

阅读者（二）：与孩子一起学编程 26

[特别专题]

关于 ESB 实施的几点建议 31

国内工作流厂商分析 40

REST 服务开发实战 48

SOA 分析和设计中的错误处理要点 53

[推荐文章]

SILVERLIGHT 之重 60

使用 DIJIT 实现界面组件化开发 66

用 DOJO 实现 AJAX 请求：XHR、跨域、及其他 72

2011 年的 HTML 5 愿景：MICHAEL MULLANY 专访 80

JAVA 深度历险（三）——JAVA 线程：基本概念、可见性与同步 83

[新品推荐]

APACHE ANT 1.8.2 发布了 88

JBOSS APPLICATION SERVER 6 发布了 88

HTML5 LABS 上线、WEBSOCKET 更新 88

ACTIVITI 5.0 GA 发布——基于 APACHE 2 许可的 BPMN 2 引擎 89

MONO 发布 MONO PACKAGER FOR MAC 89

JRUBY 1.6 发布首个 RC 版本 89

AMAZON 发布 APPSTORE 开发者门户 90

GOOGLE 发布移动 WEB 性能工具 PCAP WEB PERFORMANCE ANALYZER 90

HP 的云产品公告：混合云 90

[推荐编辑]

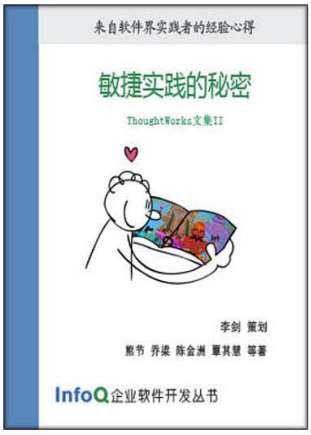
翻译团队编辑 张善友 91

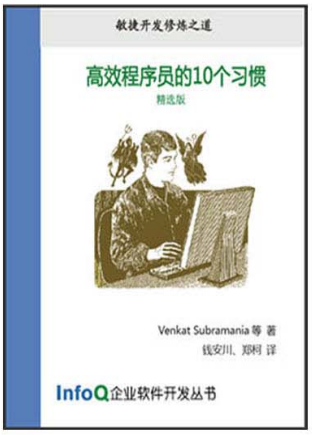
[封面植物]

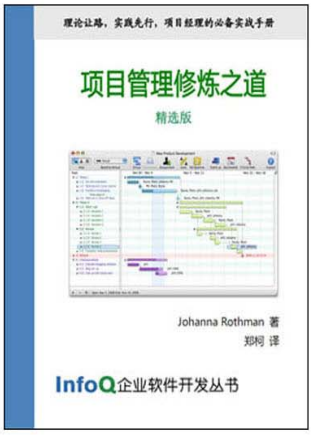
独花兰 92

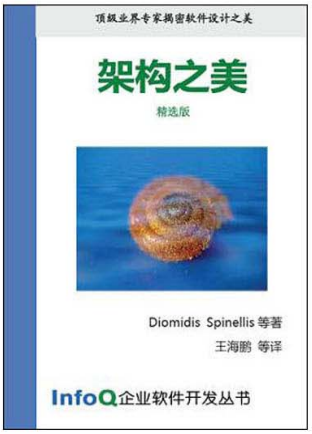
InfoQ企业软件开发丛书

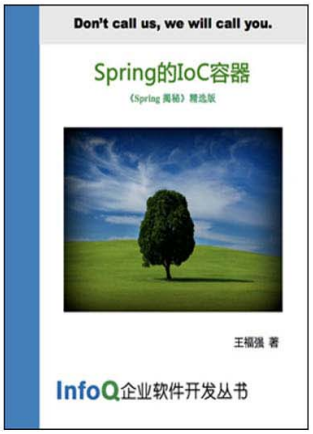
欢迎免费下载











商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

人物专访 | Interview

江枫谈淘宝“双十一”事件中的数据库架构优化

本采访是淘宝双十一事件深度报道的一部分，被采访者是淘宝技术保障部的江枫，主要负责淘宝数据库的稳定性。"双十一"期间负责协调数据库团队提供稳定性保障。就淘宝"双十一"事件，我们邀请江枫介绍了淘宝数据库的架构和硬件组成和近几年的演变过程，以及在事件前和当天，为保证数据库正常运行采取了哪些措施。

江枫，本名宁海元，淘宝系统 DBA/MySQL DBA 团队负责人，双 11 期间主要负责数据库的稳定性。从 2003 年开始超过 7 年的 DBA 经验，于 2007 年加入淘宝，伴随淘宝业务的发展，和淘宝 DBA 团队一起成长，从集中式到分布式，从小型机存储到 PC Server，从 Oracle 到 MySQL，专注于构建高可用高可扩展的数据库系统。

InfoQ：江枫先给我们介绍一下自己，和你在这次淘宝“双十一”事件中所扮演的角色？

江枫：大家好，我是淘宝技术保障部的江枫。目前主要负责数据库的稳定性这一块。双十一这一天，我主要是负责协调整个数据库团队和保障整个数据库在“双十一”过程中的稳定性不受任何影响。

InfoQ :那给我们详细的谈一下淘宝网现在整个数据库整体的一个架构，包括它硬件的组成。

江枫：淘宝的数据库发展到今天，已经是一个非常复杂的系统。我大概算了一下，淘宝目前所有的数据库服务器加起来可能已经超过 800 台。那在这么一个规模底下，淘宝的数据库团队这么多年也是随着淘宝的业务发展一起成长起来的，但淘宝数据库目前核心的数据库还在小型机和高端的存储上面，还有很多的数据库现在是用的是 MySQL，我们逐步在从 Oracle 到 MySQL 这个方向在转移，所以我们 MySQL PC server 硬件也是非常多的了。

InfoQ：我们也了解到，现在淘宝的整个的数据库团队在逐渐的把一些数据库从 Oracle 迁移到 MySQL，然后呢，把一些服务器由小型机转到 PC server，那你们整个转变的动机是什么？

江枫：主要是因为业务压力给了我们最大的动力。07 年我来到淘宝的时候，当时只有三个主要的数据库，全部在小型机和存储上面。以当时的压力来看，它跑起来是非常顺利的，而

且大家也知道小型机它从 Unix 操作系统到硬件，稳定性都会比 PC server 其实要高很多，当时的情况下淘宝用小型机是一个非常自然的选择。

从 07 年开始淘宝的业务量保持每年自然翻一番的增长，数据库质量感觉到非常大的压力。那么前端业务量增长一倍，在数据库上有可能增长是好几倍，它有一个放大效应在里边。当时我们第一步能够想到很自然的架构，就是把三个数据库拆成更多的数据库，或每一个数据库支持一个比较单一的业务。比如用户、商品和交易，都会分成独立的数据库，然后放到独立的小型计算中去，这是我们 08 年做的很大的事情就是垂直拆分，然后 08 年的业务我们就顶住了。

当时我们就预估 09 年、10 年会有更大的压力增长，这个时候我们应该怎么办？当时我们从业界能看到很多的经验分享，包括 eBay、亚马逊这些国外的大公司，他们的经验分享里面，水平拆分是我们数据库涨到一定程度后的架构选择。我们从 Oracle 到 MySQL 转移，主要是用水平拆分，这是我们未来的一个弱点，那水平拆分后机器、数据库的数量都会多很多，那 Oracle 它本身的成本也是我们考虑的一个重要因素，所以当时从成本考虑的话，那个时候我们自然会选择用 MySQL 数据库。

InfoQ：给我们再简单总结一下这几年，淘宝整个数据库的演变过程？

江枫：刚才说到 08 年我们做完垂直拆分以后，09 年到今年我们主要做的工作其实就是水平拆分。今年在十月份之前我们全部完成了淘宝最核心的三个系统：交易数据库、商品数据库和用户数据库的水平拆分。所以到“双十一”之前，在我们内部采访中，我一直跟采访人员说，当时数据库情绪稳定。基本上我们没有做什么事情，只是在不停的看报表，看数据，然后很开心的看到交易曲线以超过 45 度的趋势往上涨。

InfoQ：那前期还是做了非常完善的准备。据我们了解在整个从小型机到 PC server 的迁移，包括从 Oracle 到 MySQL 数据库的迁移，你们在做这个事情的时候，都做过好几个月的压力测试。你讲讲这个背景和故事。

江枫：是这样的，今年我们年初决定，我们商品库从小型机迁到 PC server 上面去，这是淘宝压力最大的一个数据库，当时是用四台小型机加两个高端存储来支撑的。要把这么大一个数据库进行迁移，我们心里面也是没有底的，因为不知道要多少台 PC server 能够支撑，需要什么样的配置来支撑这个压力？当时我们能够想到一个很直观的想法就是模拟线上完全一样的压力，甚至加上几倍的压力来测它的极限值。

我们和开发团队、我们的性能测试团队，加上 DBA 团队和 ops 团队，成立了一个非常大的项目组，然后做了接近两个月的性能测试，在整个测试过程中发现了非常多的问题，包括我

们给 Oracle、MySQL 等厂商都提交了很多 Bug，有些 Bug 也得到厂商回应，进行修复。

InfoQ：那整体的转变的过程到现在进行到了什么样的程度？包括你在整个转变的过程中遇到哪些问题？

江枫：我们现在最核心的用户数据库今年已经彻底完成了从小型机、存储和 Oracle 切入到 PC server 加 MySQL 的架构。

我们内部有一个提法叫做去 O、去 I、去 E，其实就是我们要从高端硬件 Scale up 模式到低端硬件的 Scal out 水平扩展的模式，这是淘宝内部最大最核心的系统，今年已经顺利完成了全部区的水平扩展。其他几个系统，比如说交易和商品已经完成了一部分，完成了水平拆分的一部分，但是没有达到我们希望的进度，这可能是明年我们需要做的事情。

InfoQ：在转型过程中主要遇到哪些问题？

江枫：让我们觉得比较大的问题就是我们从可靠的小型机迁移到大规模，大数据量的 PC server 上来，从架构上就对我们就是一个非常大的挑战。大家都知道，每一个 PC server 的稳定性肯定和单台小型机会有一定的差距，再加上我们一个机群有可能是 32 台或者 64 台 PC server。每一台 PC server 即使有四个 9 的可用性，但如果我们整个系统合在一起，可能它最后的两个 9 的可用性都达不到。这就需要我们从软件层、架构层要做非常多的改进，能够要让单点的一些失效对整体的系统不造成任何影响，因为我们和架构部门、开发部门一起做了很多事情，才能保证我们的集群稳定上线。

InfoQ：其实"双十一"这个时间应该说是对过去的技术转变的检验，现在回头来看，这个检验的结果怎么样？

江枫：当时是有点提心吊胆的，之后又觉得相对来说今年我们做的很多事情还是非常成功的。但是现在再回头仔细想想还是有点后怕，"双十一"那天的凌晨零点不是有一次 Ipad 的秒杀吗，当天晚上我们都在线上观察数据，在零点的一瞬间，就看到所有数据库指标已经达到了以前正常时候最高峰的指标，有些甚至还超过了。

当天晚上睡觉的时候心里就有点在打鼓：才零点就这个样子了，明天下午明天晚上最高峰的时候我们应该怎么渡过？所以第二天早上八点多的时候我们一进到指挥部里面就看到所有的指标， 包括 CDN 的指标、各个业务线的指标、数据库的指标都是噌噌的往上涨，这时心里面其实是很忐忑不安的。

但是我们比较放心的是这三大核心系统，商品、用户和交易，在我们今年所有的水平扩展项目做完了以后，比如说商品功能做完了以后，从我们的机械压测里面它是有十倍的流量的，

所以当天百分之一百，百分之两百的流量基本上对数据库没有造成太大的影响，所以当时还是很开心的看到这个指标快速的往上涨，希望交易能够通过 10 个亿、20 个亿，我觉得都是能够承受的。

InfoQ：那对于整个数据库架构的演进下一步有什么打算？

江枫：下一步其实就是刚刚说的我们有几个核心系统还没有完全的做到这个水平扩展，加上“双十一”那天我们还是有一个小惊险：我们有一个数据库，跟交易核心有一点点联系的，但它还是放在小型机上面，当时已经提前为它准备了百分之一百的余量，就是说它可以承担平时最高压力的两倍。

但是那天已经达到平时最高压力的 1.8 倍左右的时候，把我们吓出了一身冷汗。如果当时淘宝的交易最高峰的流量再增长 20%的话，有可能数据库就会到瓶颈了。所以我们明年是要把更多这种 Scale up 能够看到天花板的数据库全部要拆分成水平库存这种数据库。

InfoQ：那你刚才所提到的去 Oracle，去小型机，去高端存储，这个“三去”的整体思路给淘宝网带来了哪些经济上的效应？

江枫：当时我们知道小型机和存储的价格是非常昂贵的，还是拿我们刚才说压力最大的商品数据库举个例子，当初我们数据库是用了四台高端的小型机，两套高端的存储，成本加起来起码都是三千万以上。那目前我们用的是 32 台 PC server 来搭建的一个机群，价格也就是 300 万~500 万的级别。相对来说我们做完这个事情以后，解决了两三千万的硬件成本。

InfoQ：这样来讲，整体的经济效益还是非常不错的。但是其实刚才我们在前期沟通的时候也提到，你要从 Oracle 转到 MySQL，包括从小型机转到 PC server，其实里面还是会遇到蛮多问题的，包括它的不稳定性等等，那对于这一方面你有没有什么经验可谈？

江枫：在这一方面，我觉得有两个很重要的因素。第一个是我们需要和我们的开发前端应用架构部门能够紧密的合作，能够让我们的应用融入刚才说的整个机群的单点失效和容灾的问题。都需要我们和架构部门一起来考虑的；第二个比较大的经验就是目前我们在做的，深入研究 MySQL 的源代码。我们从研究和压力测试的过程中，发现 MySQL 它本身代码的一些缺陷，可能在高并发大压力下会有很多隐藏的 Bug。

在我们最近的这次测试当中，我们还发现了 Facebook 发布的 FlashCache 二级缓存的软件，当时我们是测出它一个非常大的 Bug：并发压力非常大的情况下，它会导致 MySQL 成为一个僵尸进程。我们发现了以后，很快反馈给 Face book，然后 Face book 很快就修复了这个问题，这也是我们对使用开源软件带来更大的一个信心，就是开源能够在全球得到更多的支持，大家都能够从原代码层面来解决更深层次的一个问题。

InfoQ：我想这也可能是淘宝技术团队现在那么开放，那么注重开源的动力之一。那如果说想对 MySQL 的一些核心代码做编译，就需要对人才的储备，包括各方面资源整合的要求还是蛮大的，那你在这方面有没有什么感触？

江枫：说到人才这个话题，08 年的时候，淘宝当时准备大规模的往 MySQL 方向上转，我们内部也是有一些置疑的声音。他们说淘宝 DDA 团队以前都是在 Oracle 方面比较专精，在业界来说，淘宝的 DDA 团队在 Oracle 方面更加有名气一些。所以我们内部有置疑的声音。就是说你们有 MySQL 专家吗，MySQL 出问题了以后能很快的解决吗？所以从 08 年到现在，我们慢慢的一路走过来，内部培养了很多的 MySQL 的人才，包括这几年我们的应届生的成长，再加上我们从外部招到一些专家，我们对 MySQL 的理解已经越来越深。

刚才说到，我们已经能够给 MySQL 打 Patch，已经能够给 MySQL report 这些 Bug。到现在为止，我觉得 MySQL 的成长已经达到了非常高的一个程度，我们对 MySQL 已经越来越有信心，但是未来淘宝的 MySQL 肯定是要做得越来越大的，淘宝还有很多小型机上面扩展不太容易的系统需要迁移到可扩展的机群上面来，但我们也希望业界能够有更多的 MySQL 伙伴加入我们，和我们一起来做这么一件非常有意义的事情。

InfoQ：我想能够加入到淘宝的技术团队，去经历那么多有大交易量的技术实践还是非常宝贵的。另外一个问题就是虽然说现在我们用的越来越多的是 MySQL，但是现在大家也知道 MySQL 已经被 Oracle 收购了，那对像淘宝这样的团队有什么影响呢？

江枫：大家都知道 MySQL 其实是基于 GPL 的协议来开源的软件，那淘宝在使用过程中，前期是已经考虑到一些风险。所以我们所有的 MySQL 都是自己来做编译做优化的，而且我想 MySQL 被 Oracle 收购了以后，现在看起来 Oracle 应该是给 MySQL 在开发这方面是提供了更大的帮助，像之前在 Sun 的时候，MySQL 的版本相对来说是比较混乱的，包括我们现在在用的 5.0 和 5.1 的正式版本，最近还有包括开发方面就还有两个，一个 6.0，一个 5.4，这些特性会互相交织在一起，让我们选择的时候也有点不知道到底选哪个版本会更好一点。但现在 Oracle 收购 MySQL 以后，他把 5.4 跟 6.0 这些版本已经合成了一个比较规范的 5.5 的版本，并且为它制订了很好的一个 milestone15：31，未来要怎么发展这个里程碑，M1、M2、M3、M4 这种发展方向，而到现在为止这个 5.5 已经发展到 5.6、5.7 的版本，而且已经是 IC 版本了，很快就要 GA 了，那我想这对于 MySQL 来说应该是一个好消息。我们可以用到更多更稳定的新特性，5.5 版本里有几个新的特性是我们非常关注的，比如 Google 已经达到英文 15：57 这个 pach，所以我们觉得对我们未来的这个 MySQL 这个系统非常有用的一个功能。那我们也等着 Oracle 的 5.5 这个版本能够尽快的 GA 出来。

InfoQ：非常感谢江枫接受我们的采访。我们也相信淘宝的整个数据库的一个团队将来会做

得越来越大，会支撑越来越多的交易量。

江枫：谢谢。

原文链接：http://www.infoq.com/cn/interviews/jf-taobao-database

相关内容：

- 视频：与冯大辉谈数据库架构
- iDataForum2010 数据库技术论坛总结
- Martin Fowler看到了数据存储方式的复苏
- Drizzle——数据库的新方向
- 淘宝双十一事件的前中后技术事

QCon 全球企业开发大会（北京站）

QCon北京2011会前培训确定

——大会课程持续更新

QCon北京2011 会前培训

敏捷开发Scrum培训

鲍央舟

鲍央舟有多年的电信软件研发经验以及3年实施Scrum敏捷开发经验。

领域驱动设计

Eric Evans

DDD之父，大型业务系统方面的领域建模和设计专家。

Android平台技术培训

高焕堂

台湾软件架构设计大师，现为自由的专业架构设计技术顾问。

QCon北京2011课程更新

周伟然

金融IT工程师，系统分析员/PMP，现就职于兴业银行总行福州研发中心负责核心系统平台。

潘晓良

百姓网技术总监，联合创始人。曾任职于国信朗讯，毕博全球开发中心和eBay。

肖伟

百度基础平台部高级软件工程师，负责基础库和网络编程框架的设计开发。

杨卫华

新浪产品部技术经理，目前工作以新浪微博技术平台为主，曾负责IM等通讯服务端架构设计。

- Evan Weaver：对话Twitter架构师
- Adrian Cockroft：基于云计算架构构建Netflix
- 吴凯华：构建企业自己的自动化测试平台
- 吴毓雄：淘宝网线上线下性能跟踪体系和容量规划
- 周伟然：银行应用中的稳定性考量
- 路宁：百度持续集成中的工具战争
- 肖伟：揭秘百度应用开发引擎(Baidu App Engine)
- 潘凡：MongoDB开发应用实践（PHP/Perl)

时间：2011.4.8-2011.4.10
地点：北京-京仪大酒店
网站：www.qconbeijing.com
主办方：InfoQ
电话：13911797020
010-89880682
010-64738142
E-mail：qcon@cn.infoq.com

现在报名门票**8**折，可省720元　团购（5人以上）更多优惠　**火热报名中**

热点新闻 | News

2010 年 InfoQ 中文站高性能架构内容回顾：在路上

作者 李明（nasi）

时光荏苒，转眼已是 2011 年。在过去的一年里，中国的网民数量已达 4.5 亿，年增长 20% 以上。而当当和优酷的成功上市，也让网络视频和 B2C 行业登上主流舞台。无论是网民数量的日益增加，还是网络服务的发展变化，都让互联网行业对高性能架构有了更高的要求和期许。

我们都知道，看上去光鲜亮丽的一个个网站和服务的背后，是成百上千台服务器二十四小时不停地运转，是无数行代码织就的应用程序时时刻刻响应着各种请求。当你遭遇到高可用性和可伸缩性问题的时候，当你需要处理和索引海量数据的时候，当你发现目前的数据库和文件系统已经无法满足需求的时候，或许接下来的内容和案例分享会帮助你找到解决问题的思路和方法：

1. 百万级访问量网站的技术准备工作

   这是一篇入门级的文章，作者介绍了网站发展到百万级访问量将会遭遇的问题，以及所对应的解决办法。或许你会觉得内容并不华丽，但却句句言实。相信读了作者的建议后，你会少走很多弯路。

2. 2010 年大规模技术架构的思路

   本文概括地介绍高性能架构在技术方案上的选择思路。从编程语言到分布式存储，再到可扩展架构，如何在新技术和传统方案之间做出选择，也许本文会给你一个答案。

3. 服务器端编程的十大性能问题

   这是 Web 性能诊断专家 Bernd Greifeneder 发表在《瑞士计算机世界》杂志上的一篇文章中，所列举的在过去几年工作中所遇到的服务器端编程的十大性能问题，而 Andreas Grabner 所给出的补充阅读内容也非常值得一读。

4. 迷你书：构建高性能 Web 站点（精选版）

这是一本迷你书，精选了《构建高性能 Web 站点》中的一些章节。这本书凝聚了作者多年的经验和总结，深入地介绍了构建高性能 Web 站点时的方方面面。借用某豆瓣网友的评价："既全面又极具实操性"，我认为此言不虚。

实践篇

理论离不开实践，在高性能架构方面更是如此。很多解决方案都是在实践中反复摸索和试错之后，经过了长时间的考验，才逐步为人所了解的。去年很多公司都分享了自己在高性能架构领域的知识和经验：

1. 分布式计算开源框架 Hadoop 介绍

   Hadoop 是 Apache 开源组织的一个分布式计算开源框架，在很多大型网站上都已经得到了应用，如 Amazon、Facebook 和 Yahoo 等等。岑文初的分布式计算开源框架 Hadoop 入门实践系列文章对于想要应用 Hadoop 的读者来说，是一个很不错的起点。

2. 又拍网架构中的分库设计

   本文主要介绍了又拍网（Yupoo）在 MySQL 数据库方面的发展历程，从一主一从的基本结构，发展到后来多主多从的架构。文中介绍了各种 Sharding 的方式，以及在实践中遇到的问题，深入细节，值得一读。

3. 百度大规模数据处理

   这是百度分布式系统高级工程师马如悦在 QClub 上的演讲，探讨了百度大规模分布式存储、分布式计算、分布式索引的应用需求和设计模式，以及遇到的问题和解决办法。相信在海量数据处理领域，百度是为数不多有发言权的公司之一。

4. LinkedIn 数据基础设施简介

   来自 LinkedIn 的 Jay Kreps 介绍了 LinkedIn 每天是如何处理 1.2 千亿个关系，并通过高容量、低延时的服务来完成大规模数据计算的。在计算海量连接关系时，通过引入 bloomfilter，他们将性能提升了 10 倍。

5. 揭秘淘宝自主研发的文件系统——TFS

   这是淘宝自主研发的分布式文件系统，针对海量小文件的随机读写访问性能做了特殊优化，承载着淘宝主站所有图片、商品描述等数据存储。当现有方案无法满足需求的时候，针对特定问题自主研发，并回馈给开源社区，淘宝做到了。

6. 豆瓣数据存储实践

豆瓣网的三千多万用户上传的照片、写的日记和评论，以及小组话题和订阅的博客内容是非常庞大的，采用传统的关系型数据库和网络文件存储技术很难满足数据的持续增长和高可用性的需求。BeansDB 的作者刘洪清分享了豆瓣网在数据存储上的实践经验。

高性能架构是个永恒的话题。高性能意味着用户满意度的提高，意味着成本的降低，更意味着 IT 技术不断的发展和革新。这条路很长，我们正在路上。

原文链接：http://www.infoq.com/cn/news/2011/01/infoq-2010-high-performance-sum

相关内容：

- 2010 年InfoQ中文站工具内容回顾
- 2010 年InfoQ中文站前端技术总结
- 2010 年敏捷社区回顾：蓬勃发展的一年
- InfoQ中文站.NET社区 2010 年度大盘点
- 2010 年InfoQ中文站SOA社区回顾：SOA稳步前行,REST和云计算新招连连

热点新闻 | News

NoSQL 开篇——为什么要使用 NoSQL

作者 孙立

【编者按】NoSQL 在 2010 年风生水起，大大小小的 Web 站点在追求高性能高可靠性方面，不由自主都选择了 NoSQL 技术作为优先考虑的方面。今年伊始，InfoQ 中文站有幸邀请到凤凰网的孙立先生，为大家分享他之于 NoSQL 方面的经验和体会。

非常荣幸能受邀在 InfoQ 开辟这样一个关于 NoSQL 的专栏，InfoQ 是我非常尊重的一家技术媒体，同时我也希望借助 InfoQ，在国内推动 NoSQL 的发展，希望跟我一样有兴趣的朋友加入进来。这次的 NoSQL 专栏系列将先整体介绍 NoSQL，然后介绍如何把 NoSQL 运用到自己的项目中合适的场景中，还会适当地分析一些成功案例，希望有成功使用 NoSQL 经验的朋友给我提供一些线索和信息。

NoSQL 概念

随着 web2.0 的快速发展，非关系型、分布式数据存储得到了快速的发展，它们不保证关系数据的 ACID 特性。NoSQL 概念在 2009 年被提了出来。NoSQL 最常见的解释是“non-relational”，“Not Only SQL”也被很多人接受。（“NoSQL”一词最早于 1998 年被用于一个轻量级的关系数据库的名字。）

NoSQL 被我们用得最多的当数 key-value 存储，当然还有其他的文档型的、列存储、图型数据库、xml 数据库等。在 NoSQL 概念提出之前，这些数据库就被用于各种系统当中，但是却很少用于 web 互联网应用。比如 cdb、qdbm、bdb 数据库。

传统关系数据库的瓶颈

传统的关系数据库具有不错的性能，高稳定型，久经历史考验，而且使用简单，功能强大，同时也积累了大量的成功案例。在互联网领域，MySQL 成为了绝对靠前的王者，毫不夸张的说，MySQL 为互联网的发展做出了卓越的贡献。

在 90 年代，一个网站的访问量一般都不大，用单个数据库完全可以轻松应付。在那个时候，更多的都是静态网页，动态交互类型的网站不多。

到了最近 10 年，网站开始快速发展。火爆的论坛、博客、sns、微博逐渐引领 web 领域的潮流。在初期，论坛的流量其实也不大，如果你接触网络比较早，你可能还记得那个时候还有文本型存储的论坛程序，可以想象一般的论坛的流量有多大。

Memcached+MySQL

后来，随着访问量的上升，几乎大部分使用 MySQL 架构的网站在数据库上都开始出现了性能问题，web 程序不再仅仅专注在功能上，同时也在追求性能。程序员们开始大量的使用缓存技术来缓解数据库的压力，优化数据库的结构和索引。开始比较流行的是通过文件缓存来缓解数据库压力，但是当访问量继续增大的时候，多台 web 机器通过文件缓存不能共享，大量的小文件缓存也带了了比较高的 IO 压力。在这个时候，Memcached 就自然的成为一个非常时尚的技术产品。

Memcached 作为一个独立的分布式的缓存服务器，为多个 web 服务器提供了一个共享的高性能缓存服务，在 Memcached 服务器上，又发展了根据 hash 算法来进行多台 Memcached 缓存服务的扩展，然后又出现了一致性 hash 来解决增加或减少缓存服务器导致重新 hash 带来的大量缓存失效的弊端。当时，如果你去面试，你说你有 Memcached 经验，肯定会加分的。

Mysql 主从读写分离

由于数据库的写入压力增加，Memcached 只能缓解数据库的读取压力。读写集中在一个数据库上让数据库不堪重负，大部分网站开始使用主从复制技术来达到读写分离，以提高读写性能和读库的可扩展性。Mysql 的 master-slave 模式成为这个时候的网站标配了。

分表分库

随着 web2.0 的继续高速发展，在 Memcached 的高速缓存，MySQL 的主从复制，读写分离的基础之上，这时 MySQL 主库的写压力开始出现瓶颈，而数据量的持续猛增，由于 MyISAM 使用表锁，在高并发下会出现严重的锁问题，大量的高并发 MySQL 应用开始使用 InnoDB 引擎代替 MyISAM。同时，开始流行使用分表分库来缓解写压力和数据增长的扩展问题。这个时候，分表分库成了一个热门技术，是面试的热门问题也是业界讨论的热门技术问题。也就在这个时候，MySQL 推出了还不太稳定的表分区，这也给技术实力一般的公司带来了希望。

虽然 MySQL 推出了 MySQL Cluster 集群，但是由于在互联网几乎没有成功案例，性能也不能满足互联网的要求，只是在高可靠性上提供了非常大的保证。

MySQL 的扩展性瓶颈

在互联网，大部分的 MySQL 都应该是 IO 密集型的，事实上，如果你的 MySQL 是个 CPU 密集型的话，那么很可能你的 MySQL 设计得有性能问题，需要优化了。大数据量高并发环境下的 MySQL 应用开发越来越复杂，也越来越具有技术挑战性。分表分库的规则把握都是需要经验的。虽然有像淘宝这样技术实力强大的公司开发了透明的中间件层来屏蔽开发者的复杂性，但是避免不了整个架构的复杂性。分库分表的子库到一定阶段又面临扩展问题。还有就是需求的变更，可能又需要一种新的分库方式。

MySQL 数据库也经常存储一些大文本字段，导致数据库表非常的大，在做数据库恢复的时候就导致非常的慢，不容易快速恢复数据库。比如 1000 万 4KB 大小的文本就接近 40GB 的大小，如果能把这些数据从 MySQL 省去，MySQL 将变得非常的小。

关系数据库很强大，但是它并不能很好的应付所有的应用场景。MySQL 的扩展性差（需要复杂的技术来实现），大数据下 IO 压力大，表结构更改困难，正是当前使用 MySQL 的开发人员面临的问题。

NOSQL 的优势

易扩展

NoSQL 数据库种类繁多，但是一个共同的特点都是去掉关系数据库的关系型特性。数据之间无关系，这样就非常容易扩展。也无形之间，在架构的层面上带来了可扩展的能力。

大数据量，高性能

NoSQL 数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。这得益于它的无关系性 数据库的结构简单。一般 MySQL 使用 Query Cache ,每次表的更新 Cache 就失效，是一种大粒度的 Cache，在针对 web2.0 的交互频繁的应用，Cache 性能不高。而 NoSQL 的 Cache 是记录级的，是一种细粒度的 Cache，所以 NoSQL 在这个层面上来说就要性能高很多了。

灵活的数据模型

NoSQL 无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。这点在大数据量的 web2.0 时代尤其明显。

高可用

NoSQL 在不太影响性能的情况，就可以方便的实现高可用的架构。比如 Cassandra，HBase 模型，通过复制模型也能实现高可用。

总结

NoSQL 数据库的出现，弥补了关系数据（比如 MySQL）在某些方面的不足，在某些方面能极大的节省开发成本和维护成本。

MySQL 和 NoSQL 都有各自的特点和使用的应用场景，两者的紧密结合将会给 web2.0 的数据库发展带来新的思路。让关系数据库关注在关系上，NoSQL 关注在存储上。

参考阅读

NoSQL、NoSQL 在 wiki 上的介绍、NoSQL 相关博客、NoSQL 相关博客、新浪微博 NoSQL 微群

关于作者

孙立，目前在凤凰网负责底层组的研发工作。曾就职于搜狐和 ku6。多年互联网从业经验和程序开发，对分布式搜索引擎的开发，高并发，大数据量网站系统架构优化，高可用性，可伸缩性，分布式系统缓存,数据库分表分库（sharding）等有丰富的经验，并且对运维监控和自动化运维控制有经验。开源项目 phplock，phpbuffer 的作者。近期开发了一个 NOSQL 数据库存储 INetDB,是 NoSQL 数据库爱好者。他的新浪微博是：http://t.sina.com.cn/sunli1223

原文链接：http://www.infoq.com/cn/news/2011/01/nosql-why

相关内容：关系数据库还是 NoSQL 数据库、数据库在现代搜索技术中的应用、解读 NoSQL 技术代表之作 Dynamo、图形数据库、NOSQL 和 Neo4j、潘磊谈阿里巴巴国际站发展历程

热点新闻 | News

Git 历险记（一）

作者 刘辉

【编者按】作为分布式版本控制系统的重要代表——Git 已经为越来越多的人所认识，它相对于我们熟悉的 CVS、SVN 甚至同时分布式控制系统的 Mercurial，有哪些优势和不足呢。这次 InfoQ 中文站有幸邀请到《Git Community Book》的译者刘辉，在 InfoQ 开辟《Git 历险记》专栏，分享他使用 Git 的经验，以及他对 Git 的看法。

Git 是 Linus.Torvald 为了管理 Linux 内核发起并开发的一个开源分布式版本控件系统（DVCS）。从 2002 年起，Linux 内核一直使用 BitKeeper 来进行版本管理，但是在 2005 年 BitKeeper 和 Linux 内核开源社区的合作关系结束，BitKeeper 再也不能免费使用了，这迫使 Linus 决定开发一个开源界自已的版本控制系统。

传统的 SVN、CVS 等版本控制系统，只有一个仓库（repository），用户必须要连上这个仓库才能开始提交；而 Git 之类的分布式版本控制系统（当然也还包括 BitKeeper、Mercurial 等 等），它的每个工作目录都包含一个完整的仓库，它们可以支持离线工作，先把工作提交到本地仓库后再提交上远程的服务器上的仓库里。分布式的处理也让开发更 为便捷，开发人员可以很方便的在本地创建分支来进行日常开发，每个人的本地仓库都是平等且独立，不会因为你的本地提交而直接影响别人。

老实说，Git 的速度是我用的版本控制系统中最快的（SVN Mercurial Git）。我这里说的速度，包括本地提交(commit)、本地签出(checkout)、提交到远程仓库(git push)和从远程仓库获取（git fetch ，git pull）；它的本地操作速度和本地文件系统在一个级别，远程仓库的操作速度和 SFTP 文件传输在一个级别。这当然和 Git 的内部实现机制有关，这里就不 多展开了，有兴趣的朋友可以看一下这里：Git is the next Unix。

我们在学一门新的语言时，往往是从一个“hello world” 程序开始的，那么 Git 历程也就从一个“hello Git”开始吧。

在这里假设各位同学的电脑都装好了 Git，如果没有装好，可以先看一下这里（安装 Git）。

当然，后面的章节我会专门讲安装可能会碰到的问题。

我们首先打开 Git 的命令行：windows 下是点击“Git Bash 快捷方式”；Linux 或是 Unix like 平台的话就直接打开命令行界面就可以了。

备注：$符号后面的字符串代表的是命令行输入；命令行输入后的以#开始的**黑体字符串**代表注释；其它的部分则是命令行输出。

我们先用建一个仓库吧：

```
$mkdir testGit #建立仓库目录
$cd testGit #进入仓库目录
$git init #这会在当前的目录下建一个仓库
Initialized empty Git repository in e:/doc/Git/test/testGit/.git/
```

好的，前面的三行命令就建立了一个本地的 Git 仓库。这个仓库现在是一个空的仓库。

我们在命令行下执行：

```
$ git status #查看当前仓库的状态
# On branch master (在master分支上)
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
(现在没有任何台被提交的文件，复制或创建新的文件，再用” git add” 命令添加到暂存区中)
$ git log #查看当前仓库的历史日志
fatal: bad default revision 'HEAD'
(由于仓库里没有任提交在里面，所以它会报这个错。BTW: 这种提示是不是有点不友好呀：) )
```

现在就让我们在这个仓库里添加点内容吧。

```
$ echo “hello Git” > readme.txt #建立一个含有 hello Git 的文本文件
$ git add readme.txt #将readme.txt添加到暂存区中
$ git status #查看当前仓库的状态
# On branch master
#
# Initial commit
#
# Changes to be committed:(暂存里下次将被提交的修改)
# (use "git rm --cached <file>..." to unstage)
#
# new file: readme.txt
#
```

好的，文件即然被暂存到暂存区中，我们现在就可以把它提交到仓库里面去：)

```
$ git commit -m "project init" #将刚才的修改提交到本地仓库中
```

```
[master (root-commit) 8223db3] project init
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 readme.txt
$ git status
# On branch master
nothing to commit (working directory clean)
(现在这个工作目录里没有什么要提交的东东，它是整洁的)
```

现在你执行一下 git log 命令就会看到刚才的提交记录

```
$ git log
commit 8223db3b064a9826375041c8fea020cb2e3b17d1
Author: liuhui998 <liuhui998@gmail.com>
Date: Sat Jan 1 18:12:38 2011 +0800
project init
```

“8223db3b064a9826375041c8fea020cb2e3b17d1”这一串字符就是我们这次创建的提交的名字。看起来是不是很熟，如果经常用电驴的朋友就会发现它就是和电驴里内容标识符一样，都是 SHA1 串。Git 通过对提交内容进行 SHA1 Hash 运算，得到它们的 SHA1 串值，作为每个提交的唯一标识。根据一般的密码学原理来说，如果两个提交的内容不相同，那么它们的名字就不会相同；反之，如果它们的名字相同，就意味着它们的内容也相同。

现在我想改一下仓库里文件的内容，现提交到仓库中去

```
$ echo "Git is Cool" >> readme.txt #在文件的最后添加一行
$ git status #查看当前仓库的状态
# On branch master
# Changed but not updated: (修改了，但是还没有暂存的内容)
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: readme.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

(没有修改可以被提交，使用 “git add” 命令添加文件到暂存区，或是使用“git commit -a” 命令强制提交当前目录下的所有文件)
```

OK，即然我们修改了仓库里被提交的文件，那么我想看一下我们

到底改了哪些地方，再决定是否提交。

```
$ git diff #查看仓库里未暂存内容和仓库已提交内容的差异
diff --git a/readme.txt b/readme.txt
index 7b5bbd9..49ec0d6 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1 +1,2 @@
hello Git
+Git is Cool
```

很好，正如我们所愿，我们只是在 readme.txt 的最后一行添加了一行“Git is Cool”。

好的，我们现在再把 readme.txt 放到暂存区里：

```
$ git add readme.txt
```

我们现在看一下仓库的状态：

```
$ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# modified: readme.txt
#
```

可以提交了：

```
$ git commit -m "Git is Cool"
[master 45ff891] Git is Cool
1 files changed, 1 insertions(+), 0 deletions(-)
(一个文件被修改，一行插入，零行删除)
```

再看一下新的日志：

```
$ git log
commit 45ff89198f08365bff32364034aed98126009e44
Author: liuhui998 <liuhui998@gmail.com>
Date: Sat Jan 1 18:17:07 2011 +0800
Git is Cool
commit 8223db3b064a9826375041c8fea020cb2e3b17d1
Author: liuhui998 <liuhui998@gmail.com>
Date: Sat Jan 1 18:12:38 2011 +0800
project init
```

“45ff89198f08365bff32364034aed98126009e44” 这个就是我们刚才提交修改时创建的提交。

大家这么一路看过来，是不是有点糊涂了。不过没有关系，如果你的电脑装了 Git，那么你把上面的这些命令全部执行一下遍就会对它有感性的认识了。

下面的的章节，我会讲一下如何在 windows 和 Linux 安装配置 Git，以及需要注意的问题：)

参考

- GitCommunityBook 中文版
- ProGit 中文版
- Git is the next Unix

关于作者

刘辉，开源技术爱好者，现居长沙，从事软件开发行业 7 年。关注 Linux 平台的相关技术，曾经参与开发过高性能网框架 spserver。2009 年起开始关注 Git 相关应用，与朋友合作翻译了《Git Community Book 中文版》。个人博客是： liuhui998.com。

原文链接：http://www.infoq.com/cn/news/2011/01/git-adventures-1

相关内容：

- 孙以义开发的用于VS2008/2010 的Git源代码控制提供器
- .NET和Mono项目版Git--Git#
- Git历险记（二）--Git的安装和配置
- GitHub发布问题跟踪系统和新API
- Sharpen助NGit实现托管Git命令

热点新闻 | News

Apple、Google、微软大比拼

作者 Abel Avram 译者 张龙

近日，美国咨询公司高德纳举办了一场网络研讨会，比较了当今软件业的 3 个重量级选手——Apple、Google 与微软——主要讨论了他们今天的地位以及 IT 的各种决策是如何被他们之间的竞争所影响的。TheOpenSourcery 则从不同的视角比较了这 3 家公司：敏捷性与开放性。

近日，高德纳公司的副总裁 Tom Austin 与 David Mitchell Smith 举办了一场名为"Google vs. Microsoft: The Battle for Future Dominance (and Apples Sneak Attack)"的网络研讨会，比较了 Apple、Google 及微软的主要特质、财务现状、未来发展前景、企业对他们的看法、主要产品以及 IT 的各种决策是如何被他们之间的竞争所影响的（该研讨会需要注册参加）。

Tom Austin 强调了 Apple 的如下特质：

- Apple 是个面向消费者的公司。他们尽最大努力促进消费，其次才将兴趣放到企业上。
- Apple 关注于设计与用户体验。
- 两个主要的操作系统：Mac OS X 与 iOS。
- iOS 使用了两个不同的平台：App Store（封闭且严格控制的）以及 HTML 5（Apple 贡献的开放技术）。
- 坊间谣传 Apple 正在构建自己的云数据中心。
- Apple 还在试水广告业，以此作为对 Google 踏入移动领域的回击。

关于 Google，Tom Austin 说到：

- Google 所从事的业务是大众化信息
- 他们关注的是"接下来的 20 亿"用户，企业则是第二考虑。
- 他们非常关注于达成目标的过程中可能遇到的各种障碍：网络提供商、RIA 解决方案提

供商如微软的 Silverlight、OS 提供商——微软、Apple，Google 通过广告实现了信息的货币化，对于阻碍他们的任何公司、任何人都会引起 Google 的高度重视。

- Google 是个彻底的创新者，构建了一整套全新的云基础设施。
- 其他服务所用的 Google 搜索基础设施是从零开始构建的。他们设计了自己的母板、使用自己的网络解决方案，并且使用分派机制将工作分发给遍布全球的成千上万台机器。由于这种技术，Austin 称其为“云计算领导者”。

David Mitchell Smith 谈到了微软：

- 微软的主营业务是大众化技术。其“PC in every home was the company”愿景由来已久，并且已经实现了这个目标。
- 微软还将目标瞄向了企业并且处理得很好。
- 微软是个平台公司，创建了“一个巨大的生态圈并且从中盈利”。
- 微软正试图超越 Windows 与 Office，将触角伸向搜索与广告市场，以迎头赶上 Google。作为回应，Google 通过踏入企业与 OS 领域打响了一场不对称的战役，为的就是迫使微软将精力放在其传统的领域上而放弃搜索与广告。之所以说这场战役是不对称的，主要是因为微软投入了比 Google 更多的金钱以赢得企业用户。
- Google 很容易就可以开始新的项目，如果无法为潜在用户所接受则可以将项目抛弃，整个过程没有什么痛苦，而微软的做法则不同，他们无论做什么都坚持到底，希望把事情做好，因此他们做事情的原则是“三思而后行”。

根据 Austin 与 Smith 所述，微软与 Google 的财务都很不错，但增长速度很缓慢，因此他们正在寻求新的增长机会。微软希望在搜索与广告业务上分一杯羹，其次是云计算，而 Google 的计划则不那么明确。他们已经构建了一整套工具——Chrome、Check-out、Analytics、YouTube、Search 以及 GMail 等等，他们拥有了大量用户的数据——IP、位置、搜索历史、Web 站点访问、观看过的视频以及 tweets 等等，因此高德纳公司的分析师认为 Google 可以通过用户的“社会与统计趋势、行为目标、暗藏的社交档案、个性化设置”等内容将这些“信息”货币化。

Apple 与以上两家公司都不同，因为他们在过去几年的增长速度十分强劲。iPod、iPhone 与 iPad 都是公司健康发展的信号，也为公司创造了越来越多的利润。

下图是高德纳公司调查的众多企业对这 3 家公司的认知情况：

Enterprise Perceptions

	Apple	Microsoft	Google
Focus	Consumer	Enterprise	Consumer
Cost	Expensive	Inexpensive	Free
Coolness	Coolest	Chilly-corporate	Cool-aid
Consumer-ization	Partner with others (MSFT?)	Control and Web 2.0	Enterprises will come around
Openness	Tight control	Control	Aggressive
Innovation	Savvy elegance	Detailed follower	Geeky
Cloud	Investing	Empire striking back	Putative leader
Emerging device	iPad	TBD	Chrome-based devices

Gartner

TheOpenSourcery 则从不同的视角比较了 Apple、Google 和微软 :软件开发敏捷性与开放性。Apple 被认为是做事情是秘密、长期迭代的，但必要情况下有能力实现快速反应的公司：

> **Apple**——提出大体的方向，但不会向外界透露整体的计划细节和步骤。相反，这些细节信息是极度保密的。与客户一起进行的测试和设计也是秘密进行的。其产品与技术的更新周期一般为一年，但 Apple 可以在极短的时间内实现主要的升级与 Bug 修复，这在前不久的 iPhone 与 iPad 上已经 可见一斑了。

Google 则介于微软与 Apple 之间：有时很开放，有时却像 Apple 一样保密：

> **Google**——也提出大体的方向，但却在 1 到 2 个迭代之后转到其他方向或是推迟主要项目。Google Wave 与 Nexus One 智能手机就是佐证。Google 在详细计划上有着双重性格，有的项目很开放，有的项目却像 Apple 一样保密。然而，一旦发布了基本的 API，项目就会变得更加开放。通常，Google 会邀请用户进行 Beta 测试（像是 Google Books 与 Google Maps），并且征询用户对软件的使用效果。Google 的项目的更新周期比较短，通常在 3 到 6 个月（比如 Android 与 Google Docs），但有时也需要 1 年甚至更长时间。

微软有时很开放，其开发周期通常会很久：

> 微软——也会公布策略规划和方向。Steve Ballmer 在 CES 2011 展上再一次改变了其 Windows 体系，不再排斥 Intel x86 平台之外的其他平台，而是增加了 ARM CPU+System 架构。但这个 OS 是 Windows 7 而非 Windows Phone 7（构建在几乎相同的语言工具之上，但采用了一些创新的 OS 设计与实现方法）。
>
> 与 Google 一样，微软在细节规划上也是一个双面胶——有时开放，有时保密。但微软总是拥有大量的 Beta 过程，会邀请广大用户和业界颇具声望的早期客户/组织来试用。然而，微软的软件的交付周期都很长，一般都要以年为单位。颇具代表性的就是 Windows 发布周期，从 XP 到 Vista 用了 4 到 5 年时间，从 Windows 7 到 Windows 8 估计需要 2 到 3 年时间。即便从 Vista 升级到 Windows 7 也用了一年多时间。一些小的软件项目如 IE 浏览器的开发周期也是很长的：IE6 到 IE7 花了 6 年多时间。即便比尔盖茨承诺 IE7 之后的升级周期会大大缩短，但 IE8 还是花了 2 年多时间，IE9 的发布周期估计也是 2 年左右。

每个公司都有自己的优势和劣势。他们将共存很长一段时间，但我们很难预测他们的发展。如果有人选择了 Apple、Google 或是微软，那么我们的建议就是在可能的情况下不要在一棵树上吊死。

原文链接：http://www.infoq.com/cn/news/2011/01/Apple-Google-Microsoft

相关内容：

- Google期望使用Chrome和GAE商用版在企业级市场中更进一步
- Google将WindowBuilder Pro与CodePro AnalytiX捐赠给Eclipse
- Oracle与Apple联合宣布针对Mac OS X的OpenJDK项目
- 苹果所谓的"HTML 5 和Web标准"演示本质上是披着标准外衣的垄断
- 微软会统一他们的操作系统吗？

热点新闻 | News

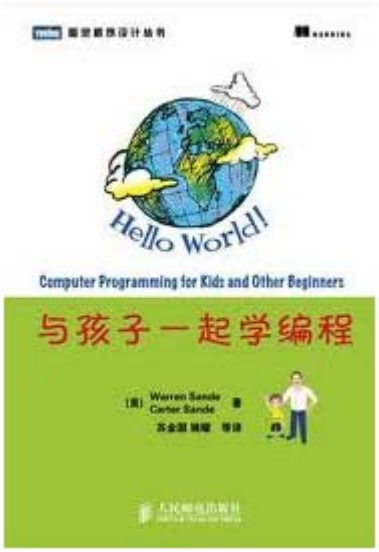
阅读者（二）：与孩子一起学编程

作者 张逸

【编者按】那些对于软件开发技术极端狂热的奇才们，大约希望这门可能解开星空奥秘的技艺，能够薪火相传。一方面是因为他的沉迷与热爱；另一方面，大约是想弥补过去的遗憾，让自己的编码人生在孩子身上重生，并奇迹般地避开自己过去在学习中所曾犯下的错误。其中一个错误，大约是在学习一门语言时，没能选好一本适合的书。

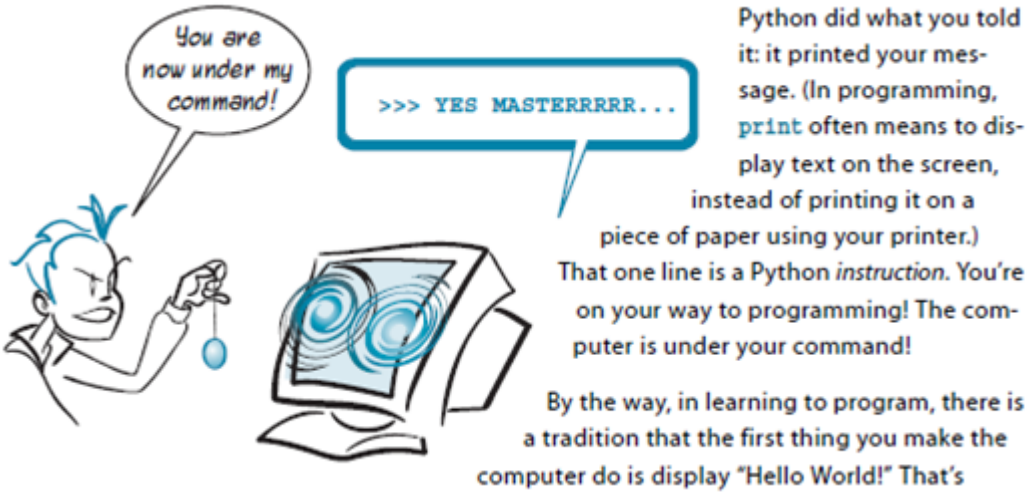
我们几位编辑不约而同选定了这本 2010 年 Jolt 获奖书籍《与孩子一起学编程》。是的，我、凯峰、陈宇还有伯薇，争先恐后想要阅读本书，目的是否想着要培养自己的孩子成为编程高手呢？若能如此，大约 InfoQ 二代就可以继续为 InfoQ 的发展做贡献了。

闲话休提，让我们来看看张凯峰与陈宇阅读此书后的感想。



“这本书的英文名是《Hello World! Computer Programming for Kids and Other Beginners》，比起中文名《与孩子一起学编程》来，区别在于本书不仅仅适合于训练孩子对编程的兴趣，对于任何对编程有兴趣的人，都是不可多得的一本编程入门书。Amazon 评分四星半，今年 Jolt 图书生产效率大奖，足以说明这本书得到了广泛的认可。

这样的中文书名，有夺人眼球的嫌疑。我知道很多对本书感兴趣的人，会是一个程序员爸爸（妈妈），有一点让自己的孩子学习编程思维的意向，但这样也流失了一部分其他可能对编程有需求的门外汉。现在更多的年轻人在选择自己创业，一个好的创意，几个好友一拍即合，缺少的却是把创意实现出来的情投意合、惺惺相惜的程序员。其实，编程世界的门槛较之以前，以前已经下降了很多，从语言到框架到平台，都有现成的选择。创业需要争分夺秒，快速地推出原型，聪明的人们为了满足这样的需要，在不停地推出能快速学习和应用的技术。而本书就是个开始。



图文并茂的写作风格

本书图文并茂的风格，让我想起了 Head First 系列图书，丰富有趣的插图不仅仅让孩子感觉到寓教于乐的轻松，同样可以让成年人过目不忘，达到最好的学习效果。我特地翻阅这本书关于"多态"、"继承"这类关于面向对象的讲解，想看看这本书如何为孩子和门外汉讲解这些极易曲解的概念。

比如"多态"，书中是这么讲的：

非常简单，多态是指对于不同的类，可以有同名的两个（或多个）方法。取决于这些方法分别应用到哪个类，它们可以有不同的行为。

说实话，这个概念讲解的不是很清楚，但下面很快给出了一段代码来说明一个方法在两个不同子类中的实现，对我来说很明了。但对于入门者来说是不是可以很快理解，我就不得而知了。这里的"两个"似乎会让人误解，我以为只要有一个方法同名不同实现，应该就是多态了。

我相信，简单的代码示例是最好的办法。学技术学编程，一开始照猫画虎写代码，是必不可少的实践，写的多了，对代码和语言的理解也会逐步加深，最终达到融会贯通。这方面这本书做的不错。

我想起我在大学学习的第一门语言，Turbo Pascal，号称最适合教学的语言。可惜毕业之后这样的语言几乎没有用武之地，Delphi 也在一时间转投.NET 阵营。一方面学校在输出一些学无致用的学生，一方面社会却在期望得到一些拥有扎实编程基础和语言经验的人才。如果我们当初有这样一本既让人对编程发生兴趣、夯实编程基础，又学到流行的编程语言的教材，该是多么幸福的一件事情。

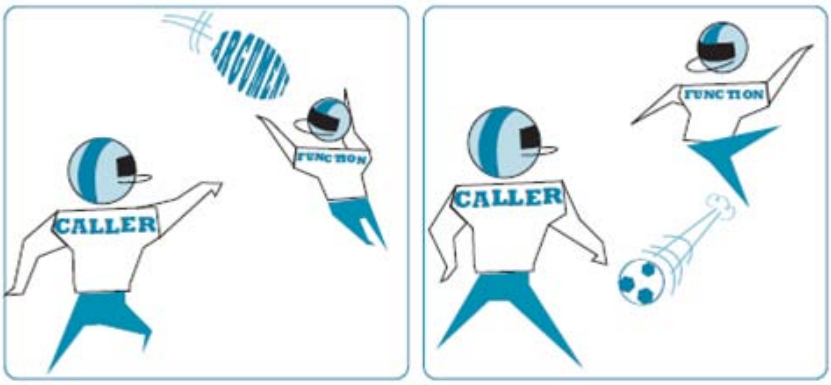
我会向更多的年轻学生去推荐这本编程书。

——张凯峰

“《与孩子一起学编程》，看到书名就忍不住好奇，想知道如何与孩子一起学编程，怎么才能把编程中那些抽象的概念让孩子们正确的理解，并不觉得枯燥呢。看完之后，觉得本书做到了这点，甚至超出了我的想象。其中不但清晰明确的解释了几乎所有编程中会遇到的基本基本概念：内存、变量、数据类型、函数、对象、模块等。还包括一些更进一步的内容，比如：动画精灵、碰撞检测，事件处理及文件读写等。

Int 函数的取整

图文并茂，是刚浏览这本书时的第一印象。对于孩子和其他一些编程的初学者，这种方式能浅显而清晰的对各种概念进行解释，非常有助于理解。比如，Python 中 int() 函数取整方式的配图，就让人印象十分深刻，立刻就能明白这一取整方式且不易忘记。又如对函数参数与函数返回值的插图也格外有趣。还有，用乐高积木来对什么是模块做比喻，对于孩子来说会非常的容易理解和接受。



函数的参数传递与返回值

丰富而有趣的插图让本书如虎添翼。

本书对一些编程的细节把握的很好，对孩子学编程中需要了解的概念和可能遇到的问题都做出了解释。除了上面提到过的一些概念之外，书中还介绍了判断语句、循环语句、列表、命名空间、多态和继承、GUI 编程等。书中在介绍碰撞检测时，不仅介绍了什么是碰撞检测及如何做碰撞检测，还列出了进行碰撞检测时可能会遇到的问题及解决办法。最后还告诉了读者什么是完美碰撞及如何获取相关信息。这可能与作者 Warren Sande 长期进行软件基础课程教育有关，当然也少不了小作者 Carter Sande 的贡献。

穿插于书中的例子都是些小的游戏程序，在作者手把手的指导下，任何人都能实现它们。这很有吸引力，尤其是对孩子们。这能让他们迅速对编程产生兴趣，并把这种兴趣一直维持下去。

本书的另一个功臣就是 Python。很难找出另一门比 Python 更适合本书的语言了。它简单、易学且功能强大。

——陈宇

本书的作者是父子“二人转”Warren a Sande 和 Carter Sande。Python Magazine 的主编 Doug Hellman 在他的博客上评价说：

> “对于那些热爱编程的年轻人而言，我强烈推荐这本书。写作这本书的作者是父子俩，阅读本书好像欣赏父子俩暑期娱乐活动一般的轻松。

图为加拿大电视台在 CTV News 栏目中对父子二人的采访。主持人正拿着本书在访问可爱的小作者。

原文链接：http://www.infoq.com/cn/news/2011/01/readers-learn-programming

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

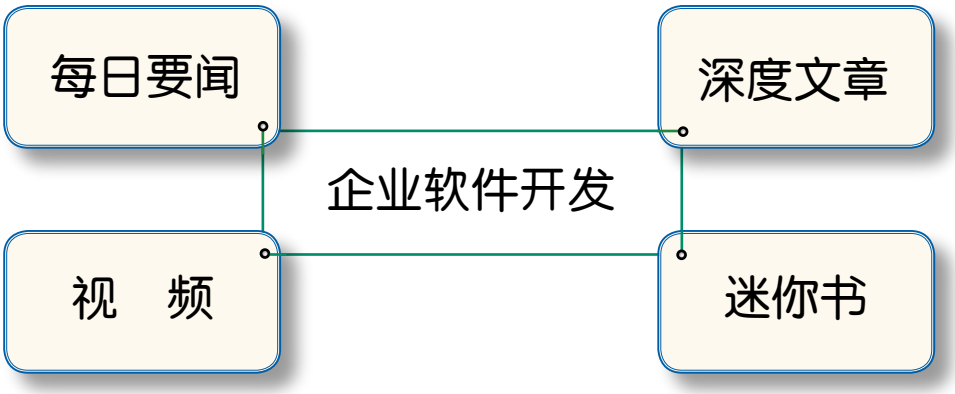
我们的**定位**：关注高级决策人员和大中型企业

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

……

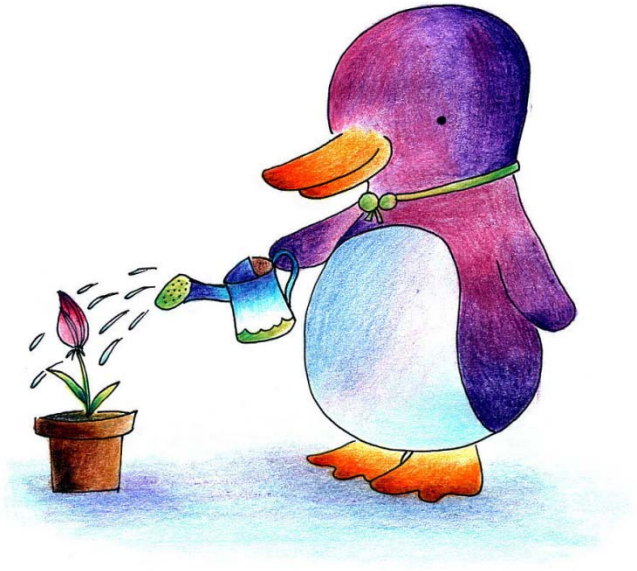


讨 论 组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
广告合作：sales@cn.infoq.com

特别专题 策划：InfoQ 中文站； 执行：马国耀

SOA 在中国也走过了若干年，它经历过热捧，面临过质疑，最终回归平淡。即使面临“SOA 已死，服务万岁”，“云计算即将替代 SOA”等犀利言论的冲击，它依然坚强地活下来了，而且越来越成熟、厚重。这么些年过去了，我们积累了些什么呢，有哪些经验是可以分享的呢？

SOA 这些年，我们的沉淀。



关于 ESB 实施的几点建议 — 国内工作流厂商分析

REST 服务开发实战 — SOA 分析和设计中的错误处理要点

特别专题

关于 ESB 实施的几点建议

作者 马国耀

前言

谈及企业服务总线（ESB），在有面向服务的架构（SOA）实施经验的开发者眼中一定不会陌生。这些年，人们一直在谈论它，以至有些人认为“实施 SOA 一定需要 ESB”，或“只要将 ESB 架起来了，我们就 SOA 了”。这些说法有可取之处，也存在片面之嫌，由于业界对于 ESB 没有统一、标准的定义，所以一千个人眼中有一千个“ESB”也就成了情理中的事情了。然而，怎么才能将 ESB 用好？我们需要清楚地认识 ESB 在 SOA 中所扮演的角色，理解哪些工作是 ESB 的职责之内，哪些却不是。只有正确地认识了 ESB 的职能，并委以恰当的任务，才能将它用在刀刃上、发挥其巨大的能量。

本文是这样安排的：

第一部分，简单介绍 ESB 在 SOA 中所起的作用。

第二部分：介绍三种常见的 ESB 的误用。

第三部分，介绍三种推荐实施。

第四部分，对 ESB 产品发展方向的展望。

申明：文中的出现所有观点仅代表作者个人看法，不代表作者所供职公司的观点。

正确理解 ESB 在 SOA 中的作用

平心而论，ESB 的确是 SOA 中一个非常重要的集成层组件（Integration Layer），不论是 OpenGroup 发布的 SOA 参考架构，还是几大主流 SOA 供应商（参考 IBM 通过参考架构实施 SOA 解决方案 ，Oracle 与 F5 的 SOA 参考架构，Microsoft 的 BizTalk ESB Toolkit 中对 ESB 的定义），都将 ESB 置于 SOA 架构的核心位置。

事实上，ESB 在 SOA 中扮演着重要的角色，在技术层解决了 SOA 的整合问题，耦合了应用与应用之间的集成逻辑，使得 SOA 更灵活，更易于扩展，更敏捷。有了 ESB，新建的服务消

费者应用程序不需要关心服务的提供者在哪里，使用何种通讯协议，与其交互的数据是怎样的……，它只需向 ESB 发出请求，使用开放的、标准的通讯协议。相反，若某个可重用的价值较大的服务位于某一个遗留系统中，而由于种种原因，该遗留系统不能在短期内重写，此时 ESB 可以架起该服务与其使用者之间沟通的桥梁。当然，ESB 的作用远不止这些，业内也有很多讨论，本文不再赘述。读者可在 Google 上搜索 ESB Patterns 获得相关资料。

然而，ESB 并非"救世主"，它注定也不可能解决应用系统整合中出现的所有问题。道理很简单，计算机发展历史长从没有出现过一个产品/工具可以满足所有的应用需求，技术发展得很快，需求发展更快，所以技术永远跟不上需求。此外，ESB 或 ESB 产品有其特定的适用范围，它是基础设施层的概念/产品，解决的是整合中的常见问题，比如服务连通、路由、消息丰富、服务的注册/查找/发布等服务的管理、服务监控和质量保证等。ESB 不能解决的问题比其能解决的问题多很多。比如，让它去做人工流程的编排是不合适的，让它提供门户类产品那样的用户交互也是极其困难的……

笔者支持过许多客户项目。在这些项目中，有的客户将 ESB 用的好，有的则勉强用上，用的功能很简单，有的则用 ESB 做一些原本不属于它该做的工作。在这里，笔者仅从个人的立场，分享自己这些年来积累的 ESB 实施经验。下面列出笔者常看到但不推荐的实施和笔者在实施 ESB 的过程中积累的一些较好的实践方式，供读者参考。同时欢迎批评指正。

不推荐实施

挟 ESB 以令外围应用

- **现象：**

ESB 的架构师在 ESB 上设计一套标准的数据接口（通用的 XML 格式），规定使用统一的协议（如 Web Service/HTTP）。所有的 ESB 服务消费者和接入 ESB 的服务必须符合该标准。其目的是为了简化 ESB 上的开发工作。这就是一种"挟天子以令诸侯"的做法，因为在实际情况中，可能领导规定了"所有的服务必须要经过 ESB，即便是透传"。

- **分析：**

国内的 ESB 实施者大多数是一些 SI/ISV，出于成本/人力或其他个方面的原因，总会有一些架构师希望达成这样一个目标：我能否设计/实现一个一劳永逸的 ESB 中间平台，将来不论哪种服务都可以方便地接入到 ESB 上？

这是一个很浪漫很理想的目标，但这个目标是不可能实现的。原因有二：

- 其一，仲裁逻辑一般是非常具体的，具体的服务有具体的整合需求。譬如，一个服务提供者基于 HTTP 提供了一个 Web 服务 WS1，而我们希望将这个服务通过 ESB 暴露给两个客户端 ClientA，Client B；Client A 使用 XML/HTTP 访问服务，而 Client B 却使用 SOAP/JMS 访问该服务，如图 1 所示。有过 ESB 实施经验的人都能看出里面的仲裁逻辑是非常具体的，我们要考虑的不仅仅是协议之间的差别，还需要考虑消息格式的差别。
- 其二，如果有这样的设计/实现存在，ESB 产商为何不把这个特性直接实现了呢？你也许会说产商不理解具体的业务，而具体的业务是复杂的，SI/ISV 却了解这些复杂业务。而事实上，ESB 解决的更多是技术层面的工作，业务层面的工作大多数不属于 ESB 的范畴，复杂的业务逻辑不是在业务系统中实现，就是在其它 SOA 组件中实现，如流程管理引擎。

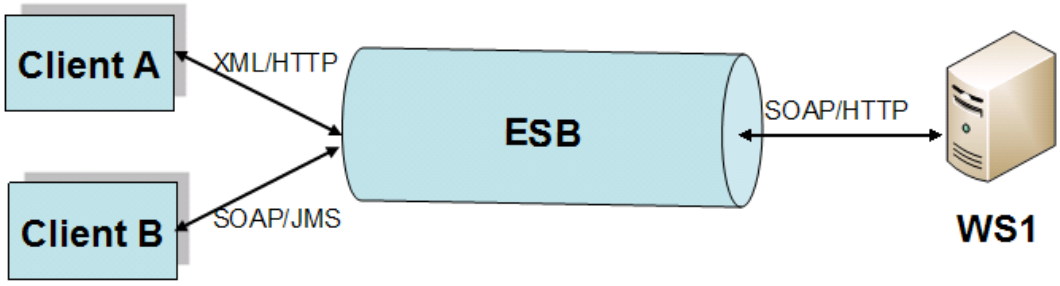


图 1. ESB 的主要功能之一是连接异构协议和数据。

- **原因及危害：**

该实践出现的根本原因是：没有认清 ESB 本身的作用，过分看重了 ESB 之上的设计能力。其实，ESB 针对的是一个个功能各异的整合逻辑，服务之间的整合逻辑也是迥异的。所以，一劳永逸的 ESB 之上的架构是不存在的。

其危害是：丢失了 ESB 本身最强大的连通性方面的功能，正式因为需要联通的应用之间的差异性，才逐渐发展出了 ESB 这样的组件。如果要求所有的外围应用的协议标准和数据标准都统一了（事实上，这是不可能实现的，特别在与第三方应用整合时），那么 ESB 的必要性就逊色不少。

- **建议：**

不要试图实现一个一劳永逸的 ESB 之上的架构，这个架构不存在也没有其存在的必要性。不过，笔者并非反对设计。实际上，ESB 上可以设计，也可以通过一定的设计实现某种程度的自动化，灵活性和通用性。举个例子，某一组功能类似的服务，对这一组服务的所有请求必须进行审计、安全加固等操作，对于这一类需求，我们可以事先在 ESB 上设计好相应的技术

组件/服务，然后在 ESB 的仲裁流中间调用该组件/服务即可。

用 ESB 实现业务流程

- **现象：**

有些架构师看到 ESB 支持服务组合（Service Composition）模式，进而认为可用该模式来实现业务流程。因此，ESB 产品就演变成了 BPM 产品。

- **分析：**

如果读者关心过 ESB 的通用使用模式，就会发现 ESB 的服务组合模式(Service Composition ）与 BPM 中的服务编排（Service Cherography）非常相似，二者都是将细粒度的服务组合/组装起来，形成大粒度的服务，或者业务流程。

事实上，二者有着本质的区别。其一：ESB 是一个偏向技术层整合的组件，比如将“客户资料查询”服务与“日志”服务组装起来，得到的结果还是“客户资料查询”服务，只是在仲裁流中间插入了一个新的附加功能，即“日志”服务。BPM 中的服务编排的含义很侧重于业务流转的概念。比如“项目立项审批流程”，该流程的实现可能需要来自几个相关系统中的服务的参与，它们可能是“立项申请”，接着是“一级职能经历审批”，然后是“部门经理审批”，“财务审批”等。这些服务流转起来形成的是一个完整的业务流程。其二：ESB 上的服务组合是无状态的，ESB 运行时会为每个请求建立一个实例来执行其仲裁逻辑，请求与请求之间没有时序关系，是互不相干的、各自独立的。相反，BPM 上的服务编排这不一样，未完成的业务流程实例一直会存活在 BPM 运行时中，存活期可能为一天、一周、甚至 1 个月或更长；请求与请求之间可能存在着一定的关联性，比如对与一个项目（相同的项目 ID）的立项审批流程，一级职能经理、部门经理和财务对流程发出的请求都是针对同一运行时实例的。

- **原因及危害：**

除了其他非技术的因素外，导致该实践的一个重要原因是：混淆了 ESB 的服务组合与 BPM 的服务编排两个概念。

危害：让 ESB 实现 BPM，特别是长运行的流程时，虽然在技术上可以实现，但是这违背了 ESB 产品的设计理念，会大大影响其 ESB 运行时的整体运行效率。

- **建议：**

认清技术上的服务组合与服务编排之间的差异，分析每个产品所属的概念范畴，选择合适的产品解决合适的问题。

用 ESB 实现大数据传输

- **现象：**

将 ESB 用于在两个系统间传输大量数据，比如传输视频文件、大文档等。在这些场景中传输的信息/文件/消息有一个共同的特点：只使用了 ESB 提供的连通性功能，而没有使用其他功能，如消息解析，转换，路由等。

- **分析：**

为了在 ESB 市场上抢占有利位置，各大厂商提供的 ESB 产品的能力越来越强。ESB 产品的设计是支持消息从一个系统传到另一个或几个系统的。所以，一些架构师利用 ESB 产品本身提供的此项功能架设了一个数据传输总线。

可是 ESB 并不适合完成该项功能，虽然它可以实现这一功能，但这并非最佳实践。ESB 作为企业级的服务联通、管理平台，需要穿透 ESB 的服务应该是企业内重用可能最大、价值最大的那些服务，应用程序对这类服务的访问应该非常频繁，因此同一时刻需要 ESB 支撑的业务可能非常繁重。所以，ESB 产品的设计初衷是实现一个无状态、高吞吐的服务总线，为企业内重要的业务服务提供透明、标准、开放的接入能力。

- **原因及危害：**

这种实践的原因是过分放大了 ESB 对数据的传输能力，如果在 ESB 传输巨大的信息，可能会导致 ESB 整体性能的下降，损害其他重要服务的 QoS。

- **建议：**

如果需要传输的数据有解析或转换的需求、或者需要根据数据里的信息进行某种路由或其他仲裁逻辑需求时，则建议使用 ESB；但如果同时消息体又过与庞大，可以考虑将需要解析的消息与不需解析的数据部分进行拆分，通过其他专门的数据传输平台去传输不需解析的部分，如 FTP，数据库备份机制等，而在 ESB 中实现该传输任务的控制流。

推荐实施

服务要管理起来

ESB 的一个重要功能是将企业内/合作伙伴处的服务以开放的、标准的服务方式暴露出来，使得服务消费者能够便利地查找到服务，以促进服务的重用、管理。而许多 ESB 产品不包含

服务的注册、存储、发布、订阅等功能，这些功能包含 UDDI 库中。所以，ESB 一般需要一个服务注册/存储库与之协作。ESB 执行各种仲裁逻辑，而注册库为 ESB 提供必要的元数据信息。

服务注册库可以存储服务的元数据，包括服务的描述、功能、版本、输入/输出消息的模式、服务端点、SLA、安全策略等内容，这些信息可被 ESB 用来进行消息格式验证、服务的动态路由、执行安全策略和 SLA 保证等。

ESB 可以多级实施，可以实现整个企业级/全国级的 ESB，也可以实现部门级/地市级的 ESB。服务消费者应该能在注册库中找到自己所需要的服务，获得调用该服务所需的位置、服务的描述文件、请求/相应消息格式等信息。若是没有注册库，则随着 ESB 接入的服务越来越多，仲裁逻辑不断增加，必将造成服务管理的混乱。最后抵消了 ESB 带来的价值。

复杂的动态路由规则应服务化

路由是 ESB 中非常重要的仲裁逻辑之一。路由场景是非常普遍的。譬如，针对不同的客户提供不同 QoS 的场景，执行时需根据客户的类型将其路由到不同执行能力的服务提供者；再比如当响应消息到达 ESB 时，总是需要将该响应消息送回最初的服务请求者处。

路由可分为静态路由和动态路由。静态路由指得是设计时已经明确路由分支的情况，而动态路由的路由分支选择是在运行时动态确定的。不论是静态路由还是动态路由，路由分支的选择一定伴随着一个或一系列决策依据。决策依据可简单如一个 If-Else 语句，也可以复杂到需要通过多维决策表并通过多次判断才能得到最后的路由分支。

对于复杂的路由，推荐将路由规则的逻辑外部化，并将它服务化，如图 2 所示。如此，仲裁逻辑在分发消息前可先访问该路由规则服务，从而得到明确的路由结果。至此，可能有人会问到，“复杂”与“简单”如何界定。这个问题没有统一的规定，根据笔者以往的实施经验，如果符合以下条件之一，就可以认为是复杂的路由规则：

1．判断维度大于等于 2

2．连续决策节点大于等于 3

3．路由分支可能随时增加或减少

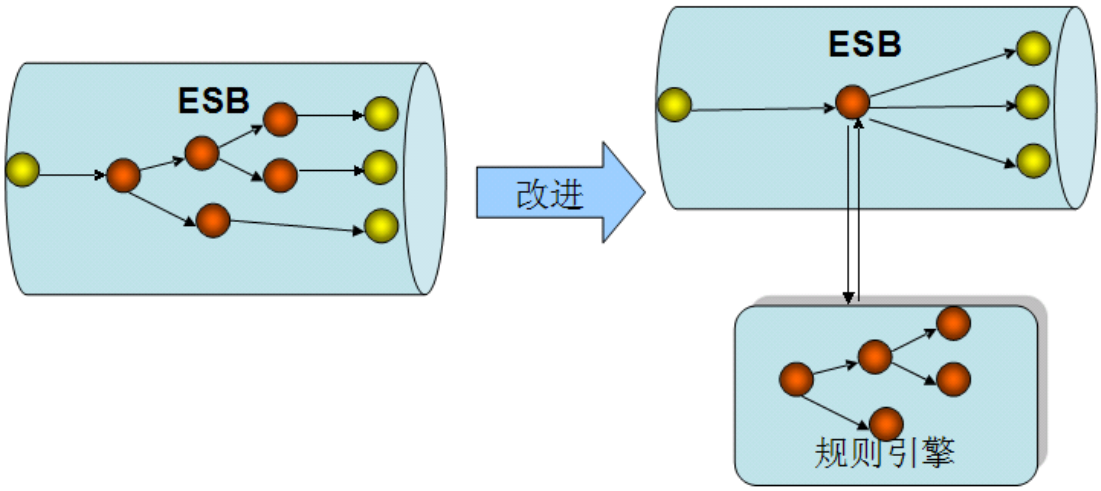


图 2. 动态路由规则服务化。

将复杂的动态路由规则服务化的好处简单明了。首先，让复杂的规则服务化，可以降低仲裁逻辑的变化性。当规则发生改变时，不需要修改仲裁逻辑，而只需更改规则服务即可。进一步，如果该规则服务是由某种专业的规则引擎实现的，更改起来将更加简单。其二，由于 ESB 产品的实现与设计并无业界的标准，所以基于 ESB 产品的配置开发无法在不同产品间互相移植。将规则服务化之后，还可减少仲裁逻辑在不同的 ESB 产品间移植时所需的工作量。最后，规则的服务化符合 SOA 原则，企业内业务规则应该以服务的形式管理起来，以促进规则的重用。

转换逻辑的实现尽量选用 XSLT

数据转换也是仲裁逻辑中非常常见并且重要的功能。同一数据在不同的业务系统中表现方式可能不一样，如上文例子中的 Client A 访问服务 WS1 的情况，仲裁逻辑需要将 XML 消息转换为调用服务 A 的 SOAP 消息。

一般而言，大多数 ESB 产品都提供了多种数据转换的方式，很多产商宣传中力推的都是“拖拽式”可视化映射的转换方式。该功能听起来的确很酷，看上去很直观。但是正如我们前面所说的，ESB 是一个偏向技术层整合的组件，业务人员一般不会关心 XML 是如何转换成 SOAP 的。所以，对于开发者来说，这种“炫”功能并无太大吸引力。更重要的是，他们可能非常习惯于自己的编程语言，如 Java、XSLT、ESQL 和 PHP 等，这些语言操作起来要简单很多。笔者本人就不太喜欢使用可视化隐射的方式去实现数据转换的需求，尤其碰到需要循环运行、复杂计算等转换逻辑时，在可视化映射上实现起来更加不容易。

在此，笔者推荐的方式是使用 XSLT 实现数据转换的功能，如图 3 所示。原因如下：

- 首先，XSLT 是一个标准的 XML 转换语言，使用 XSLT 实现的转换逻辑可很轻易地在不同 ESB 产品间移植。据笔者调查，几乎所有主流商业 ESB 产品都支持 XSLT 的转换机制。
- 其二，选择 XSLT 还可增加 ESB 之上的架构灵活性。因为仲裁逻辑中的转换逻辑是最计算密集型的逻辑，它最消耗 CPU，影响整体性能。在这种架构中，我们很容易地将此转换逻辑剥离出仲裁逻辑，由一些专门的 XML 转换加速软件/硬件（如 IBM WebSphere DataPower）来完成这一工作。这样的设计可提高架构的灵活性，通过分布式计算的方式提高整体计算性能。

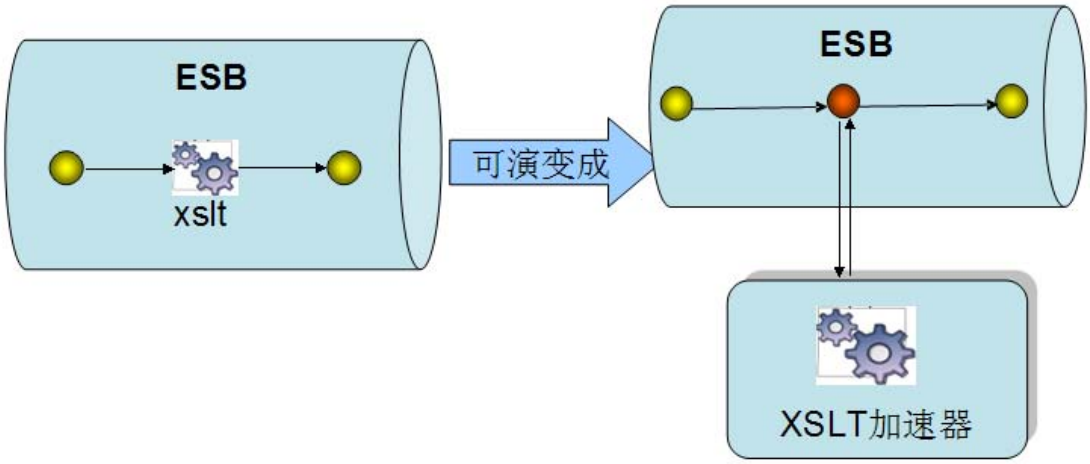


图 3. XSLT 实现数据转换逻辑及架构扩展

展望

在 SOA 架构中，ESB 扮演的重要角色是毋庸置疑的。随着企业架构的不断演变和进化，ESB 也不能一层不变。它只有与时俱进，才能顺应技术和思想的发展和进步。随着 RESTful Web 服务的兴起，SOAP 服务不再是最受宠爱的标准；SOA 架构也在不断突破企业防火墙，延伸到云计算的世界里。在这样的历史大环境下，ESB 至少要需要以下向两个方向发展，才能在竞争中立于不败之地，否则就可能被新产品所替代。

首先，**ESB 需加强对 RESTful Web 服务的整合支持。**RESTFul Web 服务相比于 SOAP 服务有着简单、易学、易扩展等方面的优势。将 RESTFul Web 服务作为 SOA 的基础构建元素的呼声非常高。虽然不通过 HTTP 传输的服务无法转变成 RESTFul 服务，但是这种对 RESTFul 服务的呼声至少代表了人们向往简单的愿望。笔者认为这种趋势不可逆转，ESB 产品必须加强对 RESTFul 服务的整合支持。譬如，可通过 ESB 将后台遗留的 MQ 应用的功能转变成一个 RESTFul

的服务，供 RESTFul 客户端访问。

其次，**ESB 需加强与 SaaS 云应用的整合支持。**随着云计算的铺天盖地席卷而来，人们对云的认识越来越清晰，对云计算经济越来越认可，云端应用变得炙手可热。在 SaaS 应用越来越受到热门的欢迎同时，出于安全、性能、已投资资产等方面的原因，企业不可能将所有的数据/应用部署在云端，所以本地应用与云端 SaaS 应用之间的整合必将成为未来几年最热的集成需求，这就催生了笔者对 ESB 第二个方面的展望。

相关内容：

- 《Open Source ESB in Action》作者谈开源ESB
- 使用JBoss ESB和JBPM实现垂直市场解决方案（VMS）
- ESB综述 1：定义ESB
- ESB是通向SOA的简单解决方案吗？
- 使用NetKerne实现REST风格的ESB

特别专题

国内工作流厂商分析

作者 荣浩

尽管在企业应用中工作流应用的越来越多，但对国内的工作流厂商们来说，这并没有给他们带来期望中的快速增长，这并不奇怪，因为国内工作流产品基本上全部面向开发者和系统集成商，解决的是编程问题，旨在简化对流程进行支撑的软件创建，这个定位决定了当越来越多的系统集成商开始自己研发工作流和越来越多的开发者采用开源工作流时，原有的工作流厂商发现生存日益艰难。

在这篇文章里，我们将一起回顾一下国内主要工作流厂商的产品以及发展策略，接着讨论他们当前所面临的困难以及未来的机会。这里分析的工作流厂商包括了东方易维、西安协同、普元、炎黄动力、有生博大、华创动力、携创、天翎、博汇数码、中创、浪潮以及台湾的华芩。

现状

大部分的工作流产品都实现了 WFMC 工作流参考模型（参见附录）的接口 1、接口 2、接口 3 和接口 5：

- 接口 1，流程设计器：包括了两种类型的设计器，一种是基于 Web 的设计器，实现技术包括了 Swing 和 Flex，一种是基于 Eclipse 插件的本地应用实现。除去普元之外，大部分工作流产品都选择实现了一种类型的设计器。Web 设计器的好处在于对最终用户友好，基于 Eclipse 的设计器的好处在于对开发人员友好，能够比较容易的进行单元测试和流程测试，缺点则是基本上隔绝了最终用户对工作流的使用，将工作流死死限制在开发者的层次上。普元同时实现了两种类型的设计器，是做得最好的厂商，东方易维和西安协同实现了基于 Web 的设计器，通过流程仿真来弥补测试的不足。
- 接口 2，工作项客户端接口：通过 API 暴露调用和交互接口，完成工作项的列表展现、拾取、退回和提交。
- 接口 3，外部应用调用接口：基本上都没有对主流 ERP、企业管理软件和财务软件进行集成的专有支持，这和国内工作流产品应用的场景有关系，工作流多作为支持单个应用

的嵌入式使用，在这一点上天翎提供有与 SAP 的集成接口。大部分通过支持 Web 服务调用进行支持。

- 接口 5，管理控制台：包括两部分，一部分是对运行中的案例进行监控和干预，包括了案例的中止、挂起与恢复，任务的中止、跳过、挂起与恢复，参与者的重新指定和催办，工作流变量的修改查看等；一部分是对案例的统计与分析，包括了针对案例、任务的时间统计，针对参与者的任务效率统计等。

除去对工作流参考模型的支持，基本上所有的工作流产品都实现了电子表单，在处理以数据填报及数据收集为主的应用中（数据不需要过多的逻辑处理和没有复杂的关系关联），电子表单能够显著的增加生产力，但是更现实的情况是企业应用大都具有复杂的业务逻辑，在这一方面，电子表单不是银弹。

通观所有的工作流产品，尽管有些产品不乏闪光点，但是整体用乏善可陈来形容是合适的，有些产品甚至可以用非常初级来描述，这不能责怪厂商，因为他们所要解决的问题域决定了他们产品所能解决的问题域。

乏善可陈表现在 6 个方面：

- 除去华芩（支持 XPDL，WFMC 成员），大部分的工作流产品都不支持流程定义规范（东方易维对 XPDL 进行了很多扩展），主流的规范包括了 XPDL、BPMN 和 BPEL。
- 对工作流模式的支持不够。工作流模式反映出产品对业务流程的建模能力，选择一个工作流产品最重要的考量就是其对工作流模式的支持程度。在这一点上老牌的厂商如东方易维、西安协同、携创和普元做得好一点，而一些以平台为销售重心的厂商做的就差一些，甚至有产品非常初级，建模只提供开始、结束和任务三种节点类型。
- 流程可视化差。流程定义作为企业重要的资产，并没有看到有产品对其可视化管理的直接支持。这里的管理包括了在组织结构内进行不同层次之间的流程可视化导航、流程权限和职责在组织中的分配、流程的全文检索和查找、与流程相关的文档管理等。流程定义更多的是作为了开发者和 IT 部门的私有财产。
- 流程建模与测试没有很好的统一。如何对建好模型的流程进行测试？两种方式，一种方式是进行流程仿真，将流程模拟的跑起来，这种方式效率低，另一种方式是将流程定义导入 IDE 进行单元测试/调试。除去普元，大多数产品都没有提供对 IDE 的支持。
- 整合能力弱。这个在上面已经提到了，基本上都是通过支持 Web 服务调用进行支持。
- 版本更新迟缓，没有持续投入。很多工作流产品早已停止更新。

尽管存在上述种种不足，但是也不乏亮点：

- 业务流程治理（BPG）的提出。东方易维首先明确提出了 BPG 的概念，并将企业流程应用清晰分为了三个层次，分别是 BPG、BPMS 和工作流。BPG 解决的是企业流程是什么的问题，这涉及到企业流程的设计、梳理和协调；BPMS 和工作流解决的是企业流程如何执行的问题，区别在于 BPMS 的要点在于打破各个应用系统（CRM、ECM、ERP、SCM）之间的界线，将分散在这些系统中的流程集中管理；而工作流则旨在简化对流程进行支撑的软件创建。在此之前，大部分厂商将工作流产品与 BPMS 产品混为一谈。

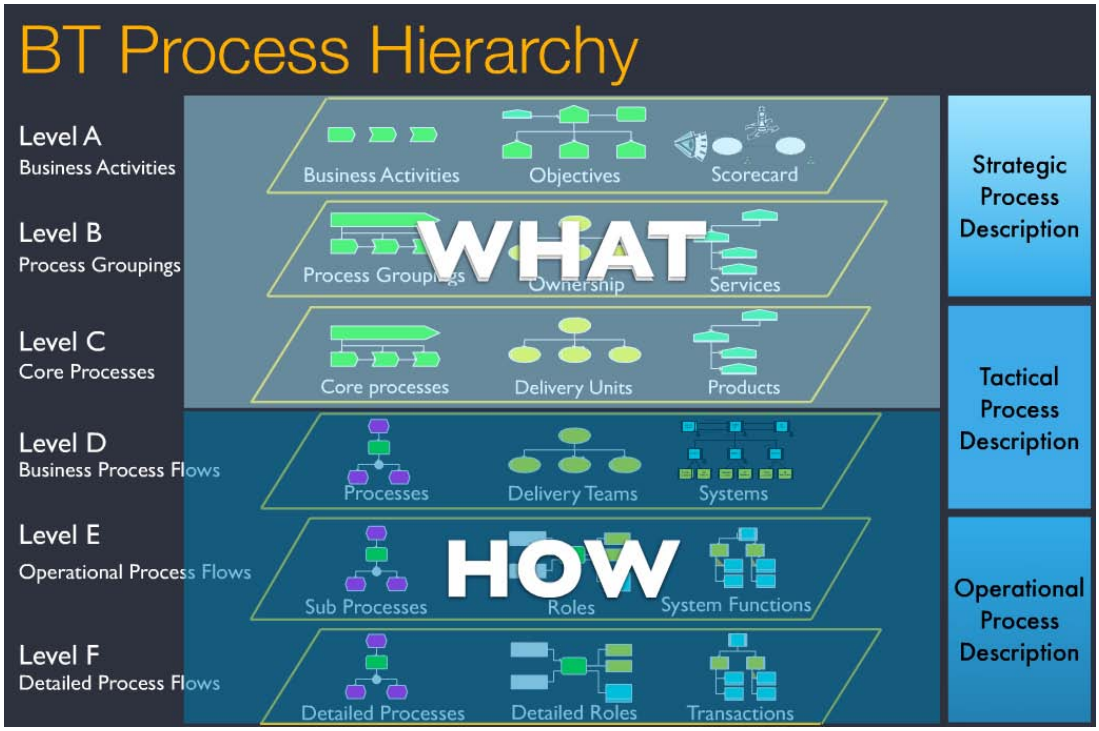


图 1 不同层次流程所对应的问题

- 云平台的提出。炎黄盈动提出了搭建流程私有云平台和将流程平台部署到公有云中。观念在国内厂商中比较超前，但是仔细思考，私有云其实是工作流产品向 BPMS 产品提升的一次尝试：工作流不再嵌入满足单个应用的使用而是暴露出流程服务让所有应用可以调用，但这项工作绝不是暴露出几个 API 可以完成的，涉及的方面非常多。至于公有云，国外产品如 Cordys 早已有成熟的流程服务，这能够显著降低企业的流程应用成本，特别是对中小企业，炎黄盈动提出了这个想法，但是并不是直接提供类似的流程服务。

- 高可靠性。西安协同和普元的产品都有很高的可靠性，这和他们的客户有关系，电信、金融。

工作流与平台

几乎所有的工作流厂商都提供平台，携创是个例外，但是这让他的市场非常狭窄。为什么会提供平台？我们先来看看工作流要解决的问题域，工作流旨在简化对流程进行支撑的软件创建，既然是简化编程，那么更进一步提供平台就显得水到渠成了。

平台分为两种，一种是快速开发平台，一种是业务平台（或者提供相关的业务套件）。

- 快速开发平台主要包括了电子表单、一套开发框架还有为宣传所需要的 ESB 和 SOA 设施（这些所谓的支持都是浮云）。
- 业务平台则包括了文件管理、在线编辑、即时通讯、电子印章、门户、内容管理、人力资源、客户服务、行政管理等套件/模块。

与单纯的快速开发平台相比，业务平台显然站在了一个更高的层次上。在软件开发中，最大的浪费往往并不在于技术本身，而是在于对业务的不熟悉，在于核心领域模型的频繁变动。对用户而言，根据需要选择合适业务平台和相关服务无疑能够产生最大的价值。

为什么有的厂商提供快速开发平台，而有的厂商提供业务平台呢？这取决于两个方面，一是厂商切入工作流市场的年限，年限越长，越积累有丰富的项目经验，这些经验很容易转化成业务套件；二是厂商的客户定位，不难发现提供快速开发平台厂商的客户都是系统集成商，自己并不承接相关项目。而反观这些快速开发平台，很难发现有特别突出的技术优势，大部分都是对 Struts、Spring、Hibernate、Ibatis 简单封装后的 CRUD 框架，加上代码自动生成和 Eclipse 插件支持，没有任何闪光点（普元是个例外，但是我们同样对其平台发展持不乐观态度，这可以再开一个话题）。

我们的观点是：快速开发平台没有太大价值，提供行业应用的业务套件/模块才是正道。

客户

观察台湾地区的工作流应用和国内的工作流应用，我们能够明显的发现：台湾地区的应用大部分集中在制造业（私企），而国内的应用则集中于政府、电力和金融行业（国企）。为什么会出现这种情况，作为制造业的大国，为什么工作流的应用却只集中在国企和政府，答案不得而知。

抛开工作流应用，工作流厂商的客户包括了以下 2 种：

- 系统集成商。这包括了两部分需求，一部分是系统集成商采购工作流产品或平台自己进行开发，一部分是系统集成商直接将项目外包给工作流厂商。这条路限制很多，越走越窄，但对于小的工作流厂商来说没有选择（没有资质和足够的资金）。
- 政府和企业。也就是自己直接接项目，这基本上是目前最好的一条道路。有些原有的工作流厂商早就转向（或部分转向）项目型公司，工作流产品不再更新和销售。

厂商的层次分类

根据上面的讨论，我们不难将工作流厂商分为 3 类：

- 只提供工作流产品。这类厂商产品单一，尽管产品质量能够得到保证，但是发展最为困难。
- 提供工作流产品和快速开发平台。这类厂商在工作流的基础上提供开发框架进一步简化编程，相比第一类厂商会更有竞争力，但是其发展受到系统集成商的限制。同时需要注意的是，部分厂商着力点在于开发平台，工作流产品水平非常一般甚至初级。
- 提供工作流产品和业务套件/平台，同时自己接项目。这是目前生存状态比较好的厂商，多是老牌厂商或是有充足的资金。业务套件/平台能够给用户提供最大的价值。在任何时候，直接面对最终用户都是王道。

挑战与机遇

尽管在企业应用中工作流应用的越来越多，但对国内的工作流厂商们来说，这并没有给他们带来期望中的快速增长，单纯的工作流产品甚至面临销售越来越困难的窘境。

困境

工作流厂商面临的压力来自于 3 方面：

- 系统集成商由购买转向自主开发。这是工作流厂商发展受阻最重要的原因，工作流应用越来越多，没有集成商愿意将这个重要的中间件依赖于他人。大多数集成商选择的方式是直接购入现有工作流厂商的源代码，也有基于开源工作流进行开发的，但是不多。
- 开源工作流的竞争。对于中小软件公司来说，如果遇到有流程的需求，他们的第一反应是采用开源工作流，而事实是开源工作流做的并不差，除去对国情的支持较弱外，开源甚至比一些商业产品还要好，尤其在对标准和模式的支持上。

- 不能面向最终用户。这是最根本的问题。工作流解决问题域的限制让最终用户根本感觉不到工作流产品价值的存在，而又没有一家工作流厂商能够做到像英特尔公司那样：组装电脑时指明要一颗奔腾的心。于是发展严重受限于系统集成商。

机会

那么，工作流厂商的机会在哪里？最重要的就是：将产品面向最终用户。

- 提供行业应用业务套件。能够做到对某类应用的快速实施，例如对于电子政务和协同办公，东方易维和西安协同都积累了丰富的经验，很多功能能够做到开箱即用。这样的好处是显而易见的，一是对集成商和中小软件公司来说，能够最大程度的节约成本，二是能够直接面对企业进行快速的项目实施。是时候抛弃单纯的快速开发平台了。
- 转向 BPMS。在工作流的基础上开发 BPMS，这需要两方面的努力：一是业务方面需要与业务流程咨询公司进行合作，二是产品需要增加 BPMS 特性，这些特性包括了对 BPMN 的支持、实现流程及相关文档可视化的内容存储仓库、提供在组织结构内进行不同层次之间的流程可视化导航、更好的业务活动实时监控预警与控制以及对流程执行的统计分析与反馈。转向 BPMS 的目的也很简单：产品面向最终用户，不再隐藏在某个应用内部。但是如何在市场上推广产品，这又是个问题，几乎所有的工作流厂商都宣称自己的工作流产品是 BPMS。
- 提供云中的流程服务，降低中小企业的流程应用成本。这个是未来的趋势，但是从目前工作流应用集中在政府和国企的情况来看，并不乐观。
- 自己做项目。大公司靠财务，小公司靠销售。有很多原有的工作流厂商转变成了项目性公司，在有工作流实施经验的情况下，开发成本相比其他一些公司会更有竞争力。

总结

国内工作流产品全部面向开发者，解决的是编程问题，旨在简化对流程进行支撑的软件创建。

国内工作流厂商分为 3 类，分别是工作流、工作流+开发平台、工作流+业务平台/套件。

工作流厂商面临困境的主要原因在于产品不能面向最终用户，这样当越来越多的系统集成商开始自己开发工作流和越来越多的开发者采用开源工作流时，生存日益艰难。

工作流厂商的机会与困境相对，就是将产品面向最终用户，这包括了自己实施项目、转向 BPMS 以及提供云中的流程服务。

附录

WfMC 之工作流参考模型

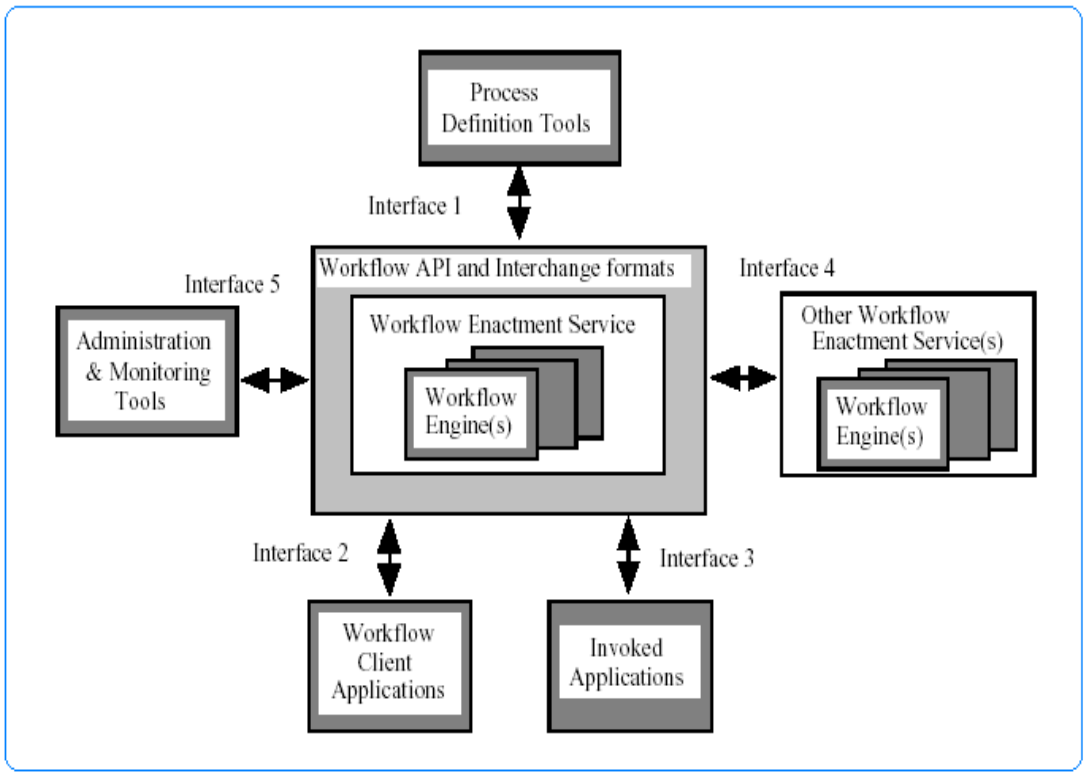


图 2 WFMC 工作流参考模型

图 2 是工作流管理联盟（WFMC）提出的工作流管理系统参考模型，工作流执行服务器周围的接口是 WAPI（Workflow APIs），通过这些接口可以访问工作流管理系统的服务，这些接口还控制工作流控制软件与其他系统组件间的交互。在这 5 个接口中的许多功能，都是被 2 个或更多个接口同时拥有的，因此 WAPI 可以看作是统一的服务接口，可以交叉使用这 5 个接口来支持工作流管理功能，而不是单独的使用其中某个接口，其中各个接口的具体含义如下：

- 接口1：工作流定义接口，为用户提供一种可视化的，可以对实际业务进行建模的工具，并生成业务过程的可被计算机处理的形式化描述即流程定义。
- 接口2：工作流客户应用接口，它给用户提供一种手段，以处理过程案例运行过程中需要人工干预的任务。每一任务的最小工作单元就被称为一个工作项(workitem)。工作流管理系统为每一个用户维护一个工作项列表，它表示当前需要该用户处理的所有任务。

- 接口3：工作流调用应用接口，指工作流执行服务在案例的运行过程中，调用的、用以对应用数据进行处理的程序。在过程定义中包含这种应用程序的详细信息，如类型、地址等。
- 接口4：工作流引擎协作接口，在大型的分布式的工作流管理系统中，工作流需要多个工作流引擎共同完成，甚至需要其他异质的工作流执行服务来辅助完成，此接口为不同的工作流管理系统之间的协作提供了一种标准。
- 接口5：管理接口，其功能是对工作流管理系统中案例的状态进行监控与管理，如组织机构管理、实例监控管理、统计分析管理、资源控制等。

工作流引擎：它是工作流管理系统的核心，工作流引擎对使用工作流模型描述的过程进行初始化、调度和监控过程中每个任务的执行，在需要人工介入的场合完成计算机应用软件与操作人员的交互。另外它的另外一个重要的功能是完成与应用软件及操作人员的交互。

关于作者

荣浩，ThoughtWorks 咨询师，关注敏捷和企业流程改进过程，目前正与辛鹏合著《Head First Process-深入浅出 IT 流程》一书。博客地址 http://ronghao.javaeye.com 。

相关内容：

- 过程组件模型：下一代工作流？
- 开源文档《OSWorkflow开发指南》发布
- 用工作流方式构建应用
- 工作流、服务和模型
- 使用Spring AOP和AspectJ编排工作流

特别专题

REST 服务开发实战

作者 邓涛

REST 介绍

如果要说什么是 REST 的话，那最好先从 Web（万维网）说起。

什么是 Web 呢？读者可以查看维基百科的词条(http://zh.wikipedia.org/zh-cn/Web)，具体的我就不多说了。总之，Web 是我们在互联网上最常用的服务，甚至在某些人的心中，互联网就是 Web。当然，Web 只是互联网的一部分而已，只是大家用的最多而已，我们访问的所有网站都是基于 Web。

那么，Web 和 REST 之间究竟有什么关系呢？我们接下来将聊聊组成 Web 的几大基础技术，URI（统一资源标识符，用来标识资源）、HTTP（超文本传输/转移协议，用来操作资源）、Hypertext（超文本，用来描述资源的内容与状态，我们可以用 HTML、XML、JSON 或者自定义格式的文本来描述任何一个资源）。

那我们再来看看什么是 REST 呢？其实 REST 并不是一种新兴的技术语言，也不是什么新的技术框架。准确来说说 REST 只是一种概念、风格或者约束，是回归 HTTP 本身的建议。

REST 是由 Roy Thomas Fieding 在他的博士论文《Architectural Styles and the Design of Network-based Software Architectures》(《架构风格与基于网络的软件架构设计》)中提出的一种架构思想。Roy Fielding 是 Apache 基金会的合作创作者，同时也是 HTTP、URI 等 Web 基础协议的主要设计者。从 Roy Fielding 的背景，我想大家就应该能了解到 REST 与 Web 之间的关系了吧。的确，在 REST 中我们关注技术实际上也只是 URI、HTTP、Hypertext 而已。

Roy 在他的论文中提出了一个 RESTful 应用应该具备的几点约束。

- 每个资源都应该有一个唯一的标识
- 使用标准的方法来更改资源的状态
- Request 和 Response 的自描述

- 资源多重表述
- 无状态的服务

Roy 认为，只有具备了上面的约束的应用才能算是 REST 应用，其实现在好多所谓的 REST 应用或服务，其实并不能算是真正的 REST 应用。

我发现，其实目前很多所谓的 REST 应用，只是 RPC 而已，出现这样的情况其实很正常，因为 RPC 实际上更符合一般程序员的思维。其实 REST 和 RPC 之间还是有很大的差异的，下面我们说一说 REST 和 RPC 之间的区别。

- REST 强调资源有唯一的 URI；而 RPC 更加强大过程（动词），由统一的接口来调用它们。
- REST 回归 HTTP 最初的设计；RPC 仅仅只是把 HTTP 作为传输协议来使用。
- REST 是由超文本驱动的；RPC 是由方法驱动的。
- REST 强调 HTTP 通信的语义可见性，通过消息头和标准的 HTTP 方法来体现；RPC 把语义封装在 HTTP 消息体中。

REST 的应用场景

通过上面的介绍，大家应该对 REST 有一些最基本的了解，由于 RESTf 应用的这些约束，我们可以很轻易的了解和使用 REST 的服务（只要你了解 HTTP）。

其实，我们经常容易犯一个错误就是，当我们了解了一个新的技术，就会用这个技术来解决所有的问题。有一句谚语是这么来说的："在锤子的眼里，所有的东西都是钉子"，其实 REST 也只是我们工具箱里面的其中的一个工具而已，希望不要把它当做我们唯一的工具。那么我们就来聊聊适合使用 REST 的应用场景和不适合使用 REST 的应用场景。

在我看来 REST 最适合的应用场景其实是需要对外暴露服务的时候，这个时候，我们可以充分利用 REST 的自描述、无状态、唯一标识等特性来提供清晰、友好的 API，而且现在的 Jesery、RESTEasy 等 JAX-RS 框架也提供了 OAuth 的支持，基本上能够保证服务安全。

最不适合的应用场景是对性能要求高的系统内部之间的服务调用，当你在这个时候使用 REST 的话，那么 REST 所有的特性都会变成拖累。这个时候，还是需要选择更底层的通信协议和方式会更好一些，比如 ICE。这样的错误，我曾经犯过，后来通过很长时间的努力才慢慢的将这个错误改过来。

规划 REST 服务

当我们要规划一个 REST 服务的时候，其中最关键的概念其实就是“资源”。

资源是什么呢？广义上讲，任何事物只要它有用，那么它就是资源。狭义的讲（在 Web 环境中），它是一个可以存放、连接在计算机上，可以通过比特流进行操控的实体。一个实体想成为资源，它必须有一个 URI。在这里 URI 包含了两重含义：1)它是资源的名称、2)它是资源的地址。

在我们规划 URI 的时候，有几点希望大家能够注意一下：

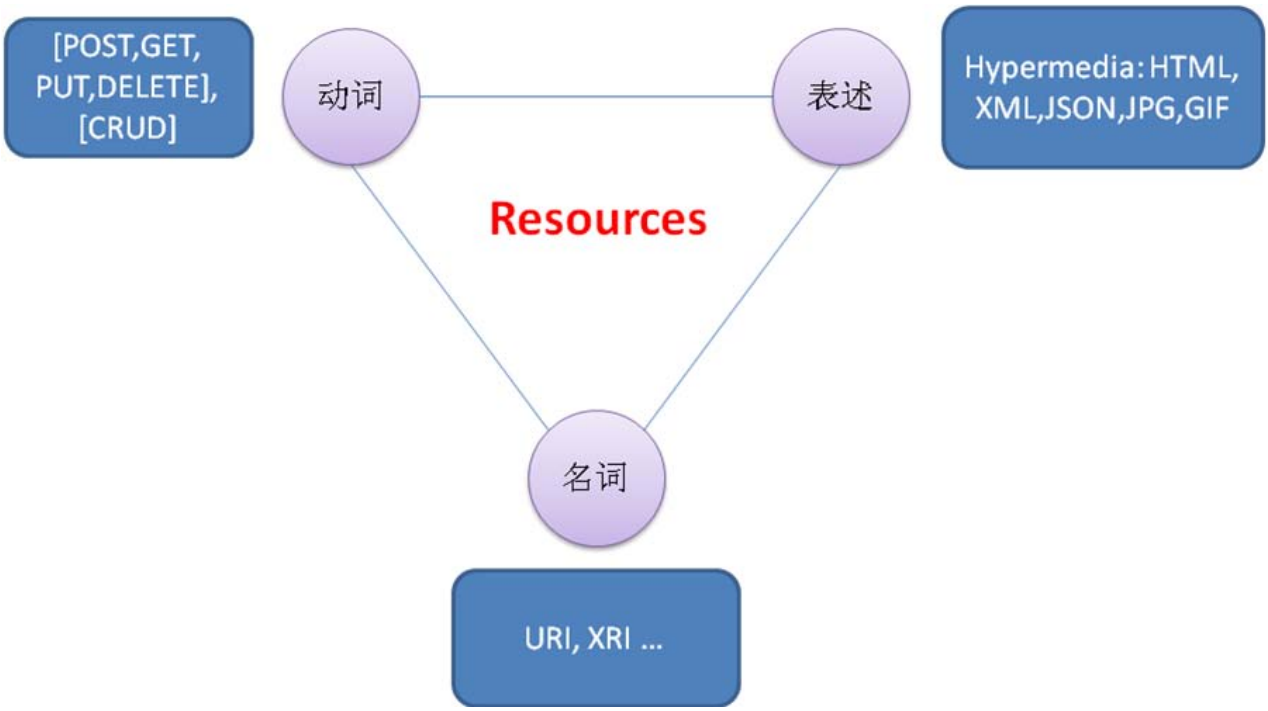
- 一个 URI 标识一个资源，但是一个资源可以被多个 URI 标识。
- 资源也是有层次的，这个层次应该在 URI 上充分的体现出来。
- 在规划 URI 的时候，需要定义一些团队内部确认的关键字或符号，这些关键字或符号是有特殊意义的，不能随便使用。
- 需要有一个 URI 定义的文档，以备以后的查询和维护。
- 可以使用 URI Template 来描述 URI 的定义。如何使用 URI Template 也看看这篇文章

当我们定义好资源之后，接下来要做的事情就是定义操作资源的方法以及资源的表述格式了。

使用 HTTP 提供的基本方法来对资源进行操作，一般的操作定义如下：POST（创建资源）、GET（获取资源）、PUT（修改资源）、DELETE（删除）。它们正好对应了 CRUD。

对资源的表述，一般的选择会是 XML，但是我更加推荐使用 JSON 来表述资源。在网络中的传输量也小，而且也便于 JavaScript 来解析，而且现在其他语言解析也是非常方便的事情。不过，最关键的还是占用更少的资源，让同样的资源能够服务更多的人。

下面的这张图就很好的说明了 REST 中最重要的围绕资源的三角关系。



在 InfoQ 上有一篇很好的文章来介绍如何规划 REST 服务《如何获取（GET）一杯咖啡——星巴克 REST 案例分析》,估计大家看完这篇文章 ,应该对如何规划 REST 服务会有更深的认识。

选择一个快速方便的 REST 框架

现在 REST 的框架也非常多，推荐大家使用 Jersey 和 RESTEasy 来创建自己的 REST 服务。

这两个框架都出自名门，Jersey 是由 SUN 提供的 JAX-RS 实现参考，对 JAX-RS 支持的最为充分和快速，基本上所有的 JAX-RS 的新特性都会在 Jersey 里第一个体现出来，而且提供了相当全了例子让你学习。RESTEasy 则是有 JBoss 开源的项目，它同样有很多优点，而且文档也比 Jersey 更好一些，但是和他 JBoss 应用服务器绑定的比较紧密，这点我个人不太喜欢，如果是熟悉 JBoss 应用服务器的人可以选择 RESTEasy ,它给人的感觉更加成熟一些 ,不像 Jersey 会很快的加入新的特性。不过，需要根据个人自己的喜好来选择。

如何使用 Jersey 来快速创建 REST 应用，参见通过 Jersey 快速构建 REST 应用，如何使用 RESTEasy 快速创建 REST 应用，参见使用 RESTEasy 快速创建 REST 应用。

发布 REST 服务需要注意的地方

之前我也提到了使用 REST 的最佳的场景是对外提供公开的服务，也就是所谓的 OpenAPI。

一旦开放了 API，我们就很难控制这些 API 的使用及其调整了，如果在开放这些 API 之前考虑的不周到的话，那么后期的维护那就会是一个非常麻烦的事情了。所以，当我们决定要开放 API 的时候，那么我们一定要注意一些事情，下面的这些算是我的经验总结。

对外暴露 API 时，需要注意版本规划，以便以后 API 的升级和维护。API 的版本规划，在开始开放 API 的之前，是一件很容易被忽视的。但是一旦你的 API 开放之后，那么你就会发现，没有对开放的 API 进行版本规划，是一件非常愚蠢的事情。当你的 API 使用的人越来越多，当你的开放的 API 越来越多，一旦某个 API 要升级，输入和输出发生变化的时候，你更不不知道该通知谁来升级，解决问题的时候也非常麻烦。

同样，由于对外暴露 API 之后，你很难控制 API 被调用的次数和意图，需要在一些关键的 API 被调用的次数和频率上进行控制，以免受到恶意的攻击。但是，到底次数和频率应该控制在一个什么样的程度，就要看你的 API 的关键程度以及负载能力了，每个系统都会有自己的评判，只要你掌握好了这个尺度，应该都不会有问题的。

另外，由于现在浏览器的限制，只能使用 HTTP 的 GET 和 POST 方法，如果通过 AJAX 直接调用 REST 服务，当你的服务中需要使用 HTTP 的 PUT 或者 DELETE 方法来调用的话，最好是考虑使用重载 POST 方式将需要使用 PUT 和 DELETE 方法调用的服务能够通过 POST 来调用。

关于作者

邓涛（Tony Deng），目前从事移动互联网领域，关注高性能、高并发的互联网架构。

相关内容：

- 面向资源的架构：REST的另一面
- 解答有关REST的十点疑惑
- 如何获取（GET）一杯咖啡——星巴克REST案例分析
- 宣称支持REST的Web框架到底有多REST？
- David Nuescheler谈JCR和REST

特别专题

SOA 分析和设计中的错误处理要点

作者 Hari Poolla 译者 吴宇

在 SOA 分析和设计阶段进行全面的错误处理需求分析对于正确完成设计和实施服务至关重要。缺少具体的需求以明确出错处理场景，或者缺乏将这些场景纳入 SOA 分析及设计阶段的理解，都将导致在开发阶段致力于将设计服务放在首位，只要完成功能，走正确路径（不考虑出错场景）即可。这种方法往往藏有隐患，会导致项目超支，因为之后补加的错误处理方案会产生大量的返工甚至需要对某些组件进行重新设计。

本文探讨了多个与可重用服务设计相关的错误处理要点，提供了一个框架用以说明在 SOA 分析和设计阶段应该使用何种错误处理，另外也描述了一组最佳实践，以及如何将这些最佳实践应用到设计中以确保最终的服务得以最完整的设计和实施。

简介

不同于单一整体应用，在 SOA 环境中，错误处理已成为应用设计中一个非常重要的步骤，因为 SOA 应用集成了各种异构（跨各组织边界、厂商及合作伙伴 IT 资 产）的 IT 系统。在分析和设计的初期就关注错误的处理分析，能够确保在不同的平台中为不同的模块提供适合的错误处理标准/准则。本文所罗列的是一般性的错 误处理要点，也是架构师和设计师们在 SOA 方案设计中需要解决的。SOA 分析和设计任务大致可分为三大阶段：服务标识、服务规范、和服务实现，具体可参考 Ali Arsanjani 的面向服务的建模和架构。后续关于该话题的讨论所围绕的错误处理要点都适用于这三大阶段。

服务标识中的错误处理

服务标识的目的是为了确定某一侯选服务组合，使其能够识别可重用的服务组合。该阶段包含了对业务工件包（其中包括关键需求、业务目标、功能模块、业务流程分析模块、用例等）的分析。

错误类型

错误大致可分为两类：

- **可恢复性错误**——可恢复性错误是指那些可通过恰当的可选执行路径来恢复用户程序的错误。这些错误是由于不遵守某一特定业务规则所导致的。
- **不可恢复性错误**——这些错误是客户程序无法恢复的。此类错误是在运行时中由某些不可预期的错误产生的，诸如空指针、资源不可用等编程性错误。

业务错误的标识

分析整个业务工件包为发现与服务关联的业务错误提供了很多机会。假设已存在某资产正用于某业务服务，那些组件接口可用于发掘那些通过自上而下分析所不能发现的其他业务错误。所谓业务错误实际上是一种可恢复性出错。一旦服务组合进入了最后的定稿阶段，那么请从以下几个错误处理的方面来评估可重用服务：

- **业务错误场景**——详细描述会标记某业务操作无效的条件。
- **错误概述**——当某一业务错误发生时服务消费者收到的业务出错提示，它提供对业务出错的简单描述。
- **错误代码**——可用于查询和某错误相关的附加信息的代码。
- **建议**——给服务消费者的反馈，诸如有效输入、或显示有关出错的具体信息等等。
- **服务区域**——标识某一服务区域，该区域可接收到与服务系统错误相关的所有通知。

这些定义了业务出错的属性既可以涵盖在服务合同中也可以按照需求打包进服务响应中。

流程错误恢复场景

- **标识新操作**——业务流程（长流程）或者任何微流程（短流程）都需要根据各自服务在其流程中可能抛出的业务错误来加以分析。这种分析可以引导发现更新的操作，而这些操作在传统的自上而下的流程任务分解中是无法被发现的。
- **更新流程模型**——服务操作模型/依赖性可随着在前一步中发现更新的操作而得到更新。

服务规范中的错误处理

服务规范阶段包含以下任务：定义输入及输出消息、服务和操作名称、模式、服务组合、非功能需求以及其他诸如同步/异步、调用方式等被标记出来作为服务的特征。

错误处理相关的特征

与错误处理相关的一般服务特征有：

- 有保障的交付——决定了某一服务是否要求有保障的 QOS（服务质量）交付类型。该需求帮助设计者们设计合适的异步消息处理设计模式，或者如果采用 Web 服务来实现的话使用可靠的消息处理机制。
- 监控需求——决定了是否需要为服务业务关键性错误建立事前监控。
- 错误映射/转换规则——建立由服务提供者所返回的错误代码/信息的转换规则，以及这些返回的信息该如何提供给服务消费者。就处理服务出错而言，标准的业务错误代码有助于应用程序更得心应手地使用这些服务。
- 更新处理流程——随着较新的操作发生或者在标识阶段处理业务错误时所发现的可选执行路径，现有的处理流程将得到进一步更新。
- 事务属性和边界——各种错误各具特色，例如系统出错与程序出错，这种错误的特性影响不同的运行时平台如何处理自动回滚。在某一流程中的事务属性和边界将根据发生在单个服务调用/事务时的错误进行分析。

错误的传递

可以使用多种方法来向服务消费者们传递错误，而其中非常重要的就是要给企业作出一个架构设计决定用以选择一个最为合适的传递方式。

两种最受欢迎的返回错误信息的方式就是使用 SOAP fault 或者定制化错误消息体。两种方式各有利弊，人们选择时大多基于现有的开发和运行时平台。使用面向 Web 服务的平台所实现的服务找到 SOAP fault 作为其必然选择是因为有大量基于 Web 服务工具的支持，而定制化错误消息体则更加适合那些使用传统的面向消息的中间件（MOM）平台所实现的服务。有关这个问题的进一步的探讨可参见 Boris Lubinsky 所做的比较。 总而言之，鉴于客户工具在各种不同程度上的支持以及服务器对各种不同概要（profiles）的支持，不可恢复性的系统

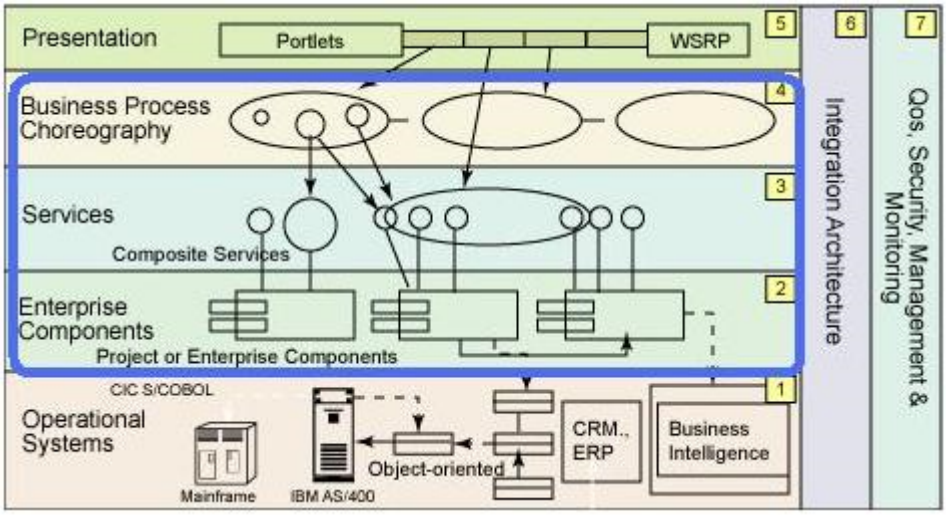
类错误更宜作为 SOAP faults 返回，而可恢复性的业务错误则更适合使用定制化错误消息体进行描述，因其能灵活和扩展地定义定制化错误模式（schema）。尽管如此，使用定制化错误消息体仍然要求服务消费者做额外的客户端处理程序来解析响应消息从而判断调用是否成功。

一般企业范围的自定义模式

通过标识元数据及一般模式来持续描述企业中出现的各类错误。该数据可包含常用的属性，这些属性包括日期、时间、错误代码、描述、严重级别、消息来源、相关 ID 等等。对于该元数据的全方面分析将十分有助于建立有效的服务监测。

服务实现中的错误处理

服务实现阶段是指服务模型影射于服务组件和运行时/部署模型时期。该步骤一般包括设计服务组件、分配组件至 SOA 各堆栈层、选择组件交互方式、运行时平台，以及制定架构设计决策(ADD)。关于此话题的后续讨论将着重围绕一些最佳实践，其中将错误处理方案实施到典型的企业 SOA 堆栈三个层次中：业务流程或编排、中介/总线以及组件，如下图所示



业务流程/编排层的错误处理

该层所部署的组件实施业务流程或编排。以下是可用的错误处理要点：

- 错误处理器——错误处理器是解决编排向外调用过程中返回的错误的最受欢迎的方式之一。错误处理器在某一流程中与特定任务绑定，或者作为整个流程的一种全方位错误处理工具。当流程中有错误发生时，错误处理器就会被调用，由它实施更正的任务。补偿

事务和手动回滚都设置了错误处理器，所以恰当的纠正措施可应用于处理流程错误。使用错误处理器时，不应将其用于可选执行路径上，而应该用于对流程抛出的错误进行恢复。

- 服务状态信息——流程编排场景通常包含调用多个服务。这些在流程内部所调用的服务可能会因多种严重程度等级（从信息、警告、错误到致命）的出错而终止。从每个调用中收集状态描述是很好的实践，例如将错误代码等返回到一个可重复的数组中然后原样返还给服务消费者。这样的做法使服务消费者能够判断整个流程是否包含由某些服务中流程所调用所引起的任何警告/错误。
- 阀值错误严重级别——定义错误严重级别阈值，并在服务编排中根据这些阈值设计容错级别。阀值级别可根据任意属性或某个定义错误的组合（如出错严重级别、定制状态代码等）来确定，而这些确定因素不能仅仅依靠 SOAP faults 来判断流程的失败。

服务/协调/ESB 层的错误处理

企业服务总线(ESB) 层是典型企业 SOA 堆栈的核心。该层支持所需的转换及路由能力独立于企业的可复用服务。该层组件通过使用恰当的消息和协议转换为各种提供商实现（如现有相关资产及基于合作伙伴或厂商的服务）提供了一种明确的接口。由协调组件完成的错误处理一般都包含将提供者错误结构转换成在业务域的上下文中明确定义的错误结构。这些组件也可以处理一些复杂的转换和错误的影射规则，这些错误由后台功能组件返回，在企业内部为服务消费者们提供了更为简单的错误信息。

- 转换提供者错误代码——不同的服务提供者会使用不同的语义来返回服务错误。范围涵盖很广，从流行的 SOAP faults 到相当专用的结构。适当的转换规则可应用于此，以便可重用的企业服务能使企业应用程序以更一致的方式返回错误，轻松解析和实施合适的处理器。
- 过滤敏感信息——当内部服务组件抛出致命错误时，堆栈跟踪往往包含敏感信息，比如所使用的协议、服务器 IP 等等。这一层将设立适当的过滤规则来过滤所有在堆栈里跟踪的敏感信息。当通过可信赖的网络传达服务响应时这一策略变得尤为重要。
- 捕获应用错误——服务组件所经历的任何技术性错误（如资源不可用或是一些运行时异常等）会转换成某种简单的技术错误消息。如果本地组件不记录这些出错，那么协调层可以将所有堆栈跟踪信息传递到日志，但却只是将通用文本消息返还给服务消费者并通知服务暂时不可用。

这一层所提及的许多出错处理的方案也可以应用于组件层。但是市场上有大量 ESB 和框架已经包含这些东西，其配置性和灵活性比起独立平台开发人员实现其功能组件实现方案要更佳。ESB 提供的出错处理仲裁关注点的分离使平台开发者们摆脱了必须应对各种各样出错处理方案的窘境，使他们把注意力更多地放在实施业务功能，提高生产力。

组件层的错误处理

这一层的组件出错处理包括处理异常操作的情况（如某一资源不可用或者运行时组件不按指令处理又或者背离逻辑）。组件需处理类似事件以通知用户程序，并且做适当的日志以便促进排除故障和服务监测。在 Java 编程语言中，类似事件的抛出称作异常，并且 API 提供了两种不同的异常：已检查异常和未检查异常。已检查异常继承自 Exception 类并用于处理可恢复性出错诸如业务出错场景。未检查异常作为运行时 Exception 类是理想的异常侯选处理不可恢复性出错，如资源不可用或者空指针。

组件层出错处理的第二个部分就是要做适当的日志。在离错误源发生最近的地方做日志是很好的做法。当组件抛出应用出错时，他们可以在组件边界内通过适当的接口记录异常，然后再抛出异常。使用关联性 id 来标识事件并将其传递给调用应用程序将可以通过链接不同平台日志的方式大大增强出错跟踪和监测。

事务回滚和补偿

设计合理的容错机制以维持流程中的 ACID（原子性、一致性、隔离性和持久性）特性是设计 SOA 解决方案的一大挑战。这些解决方案一般包含业务流程，可调用的服务可跨不同平台、多种交互模式及由不同的资源提供者提供。一个业务流程中所参与到的所有服务多半不是事务性的。一旦有某个特定的事务在某一流程中失效，就会有相应的恢复性措施用以维持数据的完整性。事务回滚和补偿事务是解决问题的两种方法。

事务回滚可通过一个事务监测器从而协调各事务来实施，前提是业务流程跨越一个限定域且所有资源都是事务型的。如业务流程更复杂些，无法完成业务流程不仅需要回滚，将数据返回至一致状态，而且需要流程调用一定的补偿事务（诸如发送警告，或在调用撤消操作来撤消前一步的服务调用等）。这远远超出了本文所论述的问题的范围。读者们可参考此空间中即将上传的 Web 服务标准：Web 服务协作（WS-C）和 OASISWeb 服务事务（WS-T）。

小结

本文提供了一些 SOA 架构技巧以用于从业务工件包中发现错误处理的需求以及在整个 SOA 分析和设计阶段如何分析这些需求。同时也提供了一些在 SOA（解决方案堆栈的）三层（如服务编排、协调及组件层）之中进行错误处理的最佳实践。

对各种错误处理要点的全面直接的分析有助于架构师们在设计和实施阶段、平台和 SOA 堆栈产品方面作出正确的决策。

关于作者

Hari Poolla 有着丰富的 SOA 实践经验，就职于位于美国中西部的一大型的保险金融服务公司。他主要关注于应用架构、BPM、解决方案设计、以及企业应用领 域的集成与合作，专攻建立企业可重用服务。为企业 SOA 采用路线图及设计定制 SAO 解决方案发展理论提供专业咨询。他具有 IBM 的 SOA 解决方案设计师认 证以及 Sun 认证的 Java2 平台企业架构师。他曾经参与电子移动基金(EFT)在各个不同行业构建和实施一个基于 SOA 的业务解决方案。你可以通过 hari_poolla@yahoo.com 与他联系。

原文链接：http://www.infoq.com/cn/articles/error-handling-soa-design

相关内容：

- 在SOA中实现异常
- 对SOA实施者的实践忠告
- 2010 年SOA发展趋势
- 十年SOA：当前的位置和未来的方向
- 实施SOA治理

开始报名啦！

Adobe在线课堂第三期

Adobe

随着AdobeFlash平台技术的不断发展及开发工具的不断更新，Flash 平台应用的开发变得越来越轻松和高效，这些应用也被越来越多的行业及企业广泛地采用。为达到帮助开发者理解Flash平台技和技术以及Flash应用的开发技巧的目的，特邀请来自Adobe的技术专家在线讲解Adobe平台技术。

黄竣：Adobe平台技术推广经理

国内最早一批Flash开发者之一，Rich Interactive Entertainment（RIE®）创始人。从事近10年的互联网互动应用及Flash游戏及应用的开发，对Flash游戏、视频及移动应用开发具有丰富的经验。目前是Adobe平台技术传教士，致力于Flash技术在游戏/视频方面的应用及推广。

课程：Flash游戏优化技巧

越来越多的游戏开发者运用Flash将自己的作品通过互联网传播，Flash游戏已经从早期单一的小游戏，逐步扩展到社交游戏、大型网页游戏领域。游戏开发者在不断的挑战Flash本身的极限，游戏优化是开发过程中永不休止的话题。本讲座将介绍一些Flash在代码、图像、内存及显示等方面的优化技巧和相关演示。

本次课堂共五期

点击浏览相关课程

1月11日　李鹏
1月18日　马鉴
2月22日　黄竣
3月 1 日　董龙飞
3月 9 日　朱建华

分享知识！
相聚Adobe在线课堂！

推荐文章 | Articles

Silverlight 之重

作者 吴磊

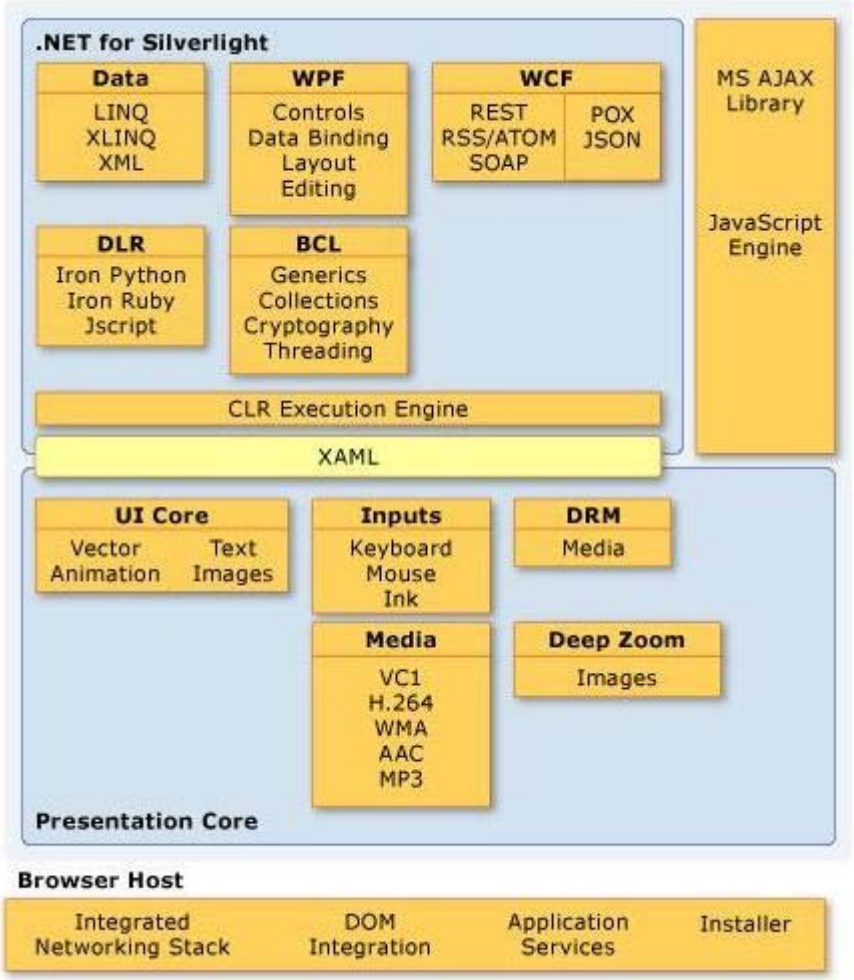
上期笔者刚提出 Silverlight之轻 ， 本期就要讨论silverlight之重？请大家不要觉得奇怪，架构的轻与重永远都是相对而言，虽然Silverlight相对于传统C/S架构有更加轻 型的运行时环境，但相较于Html+JavaScript等传统B/S架构确实具有更加庞大的基础类库支持。Silverlight依靠.Net平台，使 用C#高级语言代替JavaScript脚本语言，实现了强大的客户端计算能力、支持多线程，继承了WPF丰富的样式、控件、特效与动画，更可控的浏览器 适应性，更安全的沙箱模式，客户端嵌入式数据库等，这就是本期要给大家介绍的重点——**Silverlight之重。**

自上世纪 90 年代以来，Html+JavaScript 几乎占据了 Web 应用的全部，在近 20 年的发展过程中，一大批工具化的 JavaScript 框架应运而生，比如轻量级的 JQuery、重量级的 ExtJs 和 Dojo 等都为开发者建立 Web 应用提供了多种选择。 ExtJs 框架有强大的 UI 展现层，已经在 Turbo CRM 等企业级应用中有实际案例，而 Dojo 有 IBM、Oracle 等大公司撑腰，Struts2 更将 Dojo 集成在客户端框架中，该框架甚至还支持离线浏览。

虽然以上框架已经具有相当可观的基础库支持，但 Silverlight 的架构却拥有更加重量级的高级语言支持，使得它可以完成 JavaScript 框架所不能实现的功能，Silverlight 之重就体现在以下几方面：

- **基于.Net 平台，拥有丰富的基础类库支持，可以快速构建复杂的企业级应用**

  Silverlight 拥有强大的.Net 平台支持，自 Silverlight1.1 使用.Net 运行时环境以来，Silverlight 类库已经得到极大的扩充，并集成了 WPF、WCF、LINQ、DLR、COM 等.Net 平台的优秀技术，下图是 MSDN 上关于.Net for Silverlight 的支持图例。



由于 Silverlight 使用了 WPF 展现层控件库，使得 Silverlight 与 WPF 共享代码不存在任何技术屏障，事实上 Silverlight 就是一种桌面应用在 Web 上的轻量级实现，即便存在一些差异，我们也可以通过下面的代码将 Silverlight 应用与桌面应用粘合在一起。

```
#if SILVERLIGHT
    dynamic excel = AutomationFactory.CreateObject("Excel.Application");
#else
    dynamic excel = Microsoft.VisualBasic.Interaction.CreateObject(
        "Excel.Application", string.Empty);
#endif
```

目前已经有很多重型的企业级架构实现了 Silverlight 与 WPF 共享类库，比如 Prism 项目。

Silverlight 运行时环境目前已经有近千个可供使用的类，而 ToolKit 与大量的第三方工具程序集可以为 Silverlight 提供更加强大的后援支持。因此在 RIA 企业级应用开发过程中，团队可以通过.Net 强大的基础库与工具集快速构建出复杂的企业级应用。（敬请期待本专栏第四期《Silverlight CoreCLR 结构浅析》）

- **支持多线程，支持 UI 线程与后台线程分离，具有更大的性能优势**

Silverlight 具有彼此分离的 UI 线程与后台线程，可以将客户端的耗时操作放在后台线程中处理。这样做有两方面的收益：第一，由于没有在 UI 线程中执行耗时操作，避免了 UI 线程阻塞，提高了 UI 响应速度；第二，由于没有将操作放在服务器端运行，减少了网络的调用开销，同时通过使用客户端资源降低服务器端压力。Silverlight 多线程可以通过多种方式实现，比如 Dispatcher、BackgroundWorker、 ThreadPool 都可以轻松创建后台线程，并可以通过事件句柄实现线程间异步通信。（敬请期待本专栏第五期《Silverlight 的多线程能力》）

- **丰富的服务器端通信能力**

目前 Silverlight 支持 TCP、UDP、Socket 等网络协议，以及 HttpHandler、WebServices 等多种应用层协议，还可以通过 WCF RIA Service 数据“透明”访问方式（通过向客户端公开数据访问层，屏蔽了通信层手工编码）快速实现实体通信。

在原有应用集成方面，如果服务器端使用.Net 平台，那么 Silverlight 客户端程序可以重用原有通信方式，而不必对服务器端程序有较大改动，如果服务器端使用 Java 平台，Silverlight 也提供标准 WebServices、Http、Socket 多种交互方式进行通信，服务器端只要进行简单的服务封装就可以实现与 Silverlight 客户端的数据通信。因此，Silverlight 基于.Net 提供了快速与服务器端程序集成的能力。事实上，在笔者所开发的项目中，已经实现了 Silverlight 客户端与.Net、J2EE（通过 REST 接口）、Tuxedo（通过 SALT 接口）的数据连通性。（敬请期待本专栏第六期《Silverlight 与 J2EE 通信》）

- **LINQ 语言的支持**

LINQ 是一种开发高效的声明式语言，开发者通过可以这种类似 SQL 的语言快速构建数据逻辑，而避免了原有面向对象操作中的复杂过程，当然这只是 LINQ 的外在表现形式。其实 LINQ 申明式语言依赖于 Lamda 表达式，而 Lamda 表达式的背后是强大的表达式树 ExpressionTree，LINQ to Any 的真正强大在于将数据逻辑的具体实现方式与书写方式分离开，使开发者使用一种方式就可以访问和操作各种数据对象。而 Silverlight 集成了 LINQ 语言，在实际的应用开发中，开发者可以通过更加简单的方式来实现复杂的业务逻辑；另一方面，Silverlight 同样集成了强大的表达式树，使得开发者可以通过自行扩展表达式树来构建 LINQ To Any 的适配器，也可以使用表达式树来构建更加复杂的动态语法和动态 LINQ。当然，如果未来微软将 PLINQ 的并行查询语言集成到 Silverlight 中，那将使 Silverlight 的查询性能更近一步。（敬请期待本专栏第七期《Silverlight 轻量级查询对比》）

- **客户端嵌入式数据库支持**

Silverlight 内置 IsolatedStorage 可以操作独立存储中的数据，但这还不足以实现复杂的数据操作，我们需要在 Silverlight 项目中引入第三方嵌入式数据库程序集，实德 Silverlight 可以通过客户端独立存储来建立客户端嵌入式数据库，实现在脱机状态下存储与管理数据。开发者可以通过嵌入式数据库快速构建纯客户端（包括桌面、移动设备等）的信息管理应用，甚至实现基于嵌入式数据库缓存的“云与端”应用。目前支持 Silverlight 的嵌入式数据库选择已经相当广泛，比如 Siaqodb、EffiProz、Perst、Rapid Repository（WP7 DataBase）等都有专门针对 Silverlight 的嵌入式数据库与管理工具。（敬请期待本专栏第八期《Silverlight 嵌入式数据对比》）

- **动态语言支持**

Silverlight 对动态语言的支持分为：对 JavaScript 的互操作性、对 dynamic 动态类型的支持和 DLR 的集成。

在对 JavaScript 的互操作性方面，Silverlight 可以通过 System.Windows.Interop 类集将其应用程序中的控件暴露成 Html DOM，实现内部控件对外部 JavaScript 的可见性；Html 反向可见性是通过 System.Windows.Browser 类集来实现的，Silverlight 可以与 Html 对象或者 JavaScript 对象进行交互。Silverlight 中支持 dynamic 动态类型，允许在运行时动态加载对象。通过 Silverlight 集成了动态语言运行时环境 DLR，目前支持 IronPython、IronRuby、JScript 等动态语言在 Silverlight 应用程序中的执行。

- **与 COM 的互操作**

Silverlight 脱浏览器应用（OOB）可以在客户端创建并操作 COM 对象，比如打开 Word、Excel 等所有在客户端操作系统中注册的 COM 组件，Silverlight5 甚至允许在浏览器中调用 Office COM 对象，在 OOB 模式下更可以调用任何外部 Windows 应用程序，并支持对 USB 设备与条形码扫描器等更多外设的 API 调用。

- 3D 引擎与丰富的多媒体控制

Silverlight 基础类库包含 MediaElement 控件，可以在 Silverlight 程序中控制媒体对象，也可以通过 System.Windows.Media 类集对绘图、文本和音频/视频内容进行编程，比如通过 Brush、ImageBrush、VideoBrush 绘制更加丰富的媒体内容；通过 BitmapImage 类自定义点阵图片，实现诸如粒子效果的应用；通过 System.Windows.Media.Animation 类集来创建动画；通过 System.Windows.Media.Effects 实现模糊、阴影等多种视觉特效；通过 System.Windows.Media.Media3D 操作 3D 投射矩阵。

Silverlight5 还将增加 GPU3D 加速的 API 支持，增加 3D 视觉特效，增加即时模式图形 API，

允许将图形图像直接呈现到 GPU，这些特性的加入将使 Silverlight 具有更强大的媒体展现能力。

- **MEF 扩展性管理框架的支持**

  MEF （Managed Extensibility Framework ）扩展性管理框架，是.NET 平台下用于创建可扩展的轻型应用程序的库。在 Silverlight SDK（C:\Program Files\Microsoft SDKs\Silverlight\v4.0\Libraries）中我们可以找到 System.ComponentModel.Composition.dll 与 System.ComponentModel.Composition.Initialization.dll 这两个动态链接文件。在 Silverlight 项目中引用后，就可以使用 MEF 扩展性管理框架，该框架以声明方式指定其功能，在运行时通过 MEF 组合引擎发现部件。应用程序无需硬编码的引用或脆弱的配置文件即可利用相关部件 其依赖项注册的实现方式甚至比基于配置的Spring还要简单，Silverlight 完全可以利用 MEF 构建扩展性极强的企业级应用。

由此可见，Silverlight 拥有比 JavaScript 更为重型的框架支撑，通过 Silverlight 可以快速移植桌面应用到 Web 端，也可以将 Web 应用脱开浏览器执行，或者构建轻型的桌面应用、移动设备应用。应该说 Silverlight 不仅仅是 Web 应用，在 MSDN 的知识库中，Silverlight 已经从 Web 开发节点转到了.Net 开发节点下，从这一点上可以看出微软正在进行 Silverlight 转型。

而未来的 Windows 8 操作系统的瘦客户端 UI-“Jupiter”将采用 Silverlight，该界面将允许运行在基于 ARM CPU 的平板设备上。Silverlight 4 凭借优秀的框架已经获得了 InfoWorld 评出的

2011 年度最佳 RIA 平台奖项，未来 Silverlight 还将担负 Web 应用、移动应用和桌面应 用的统一大业，Silverlight 之重也在于此！

原文链接：http://www.infoq.com/cn/articles/wl-silverlight-heavy

相关内容：

- Silverlight 之轻
- Silverlight和Java的互操作
- Silverlight、JavaFX、Flex/Flash技术比较
- 微软设计产品市场总监Forest Key谈Silverlight
- WPF对决Silverlight：为项目选择最佳技术

推荐文章 | Articles

使用 Dijit 实现界面组件化开发

作者 成富

对于组件化的软件工程设计， 很多开发人员都比较熟悉。组件化的设计适合于复杂的软件系统和团队协作开发。把软件系统划分成若干个组件，组件之间通过预先定义好的接口和协议进行通讯和 协作，共同完成整个软件系统的职责。团队中的开发人员可以各自负责不同的组件。组件化的思想在桌面应用和 Web 应用后台开发中比较流行，相关的技术和实践 都比较成熟。而在 Web 应用的前端部分，组件化一直进展得比较缓慢。这其中的原因有很多，最主要的是 Web 应用的前端在开始的时候比较简单，没有组件化和 设计的必要。随着 Ajax 应用的流行，Web 前端部分越发复杂，用户对 Web 应用的要求不断向桌面应用靠拢。HTML 语言的基本界面元素不能单独地满足这 样的需求。菜单、树形控件、对话框和进度条等组件，在现在的 Ajax 应用中十分常见，但是并不是 HTML 默认提供的。HTML 5 规范中引入了一些新的元素，但还是不够。组件化对于 Web 应用本身的代码共享和团队分工也是很有意义的。

Web 应用前端组件化的发展也是渐进的。开始的时候，只是一些简单的 HTML、CSS 加上 JavaScript 的代码示例。比如当需要实现一个多级菜单的时候，就下载相关的代码示例，就根据自己的需要进行修改。这样的组件比较难以复用。后来 JavaScript 框架开始流行的时候，有些框架本身就提供了组件的支持，包括 Ext JS、jQuery UI 和 Dojo 等。不过不同框架提供的组件模型不尽相同。

Dijit 组件模型概述

Web 应用的前端组件的定义比较宽泛。一个组件占据 Web 页面上的某个区域，并负责完成某项具体的任务。Web 组件有时候也被称为小部件（widget）。在 Dijit 组件模型中，一个 Dijit 组件是一个 JavaScript 类，可以在页面上通过 new 操作符来创建组件的实例。每个组件实例都需要与页面上的某个 DOM 元素绑定在一起。这个 DOM 元素就是该组件的根节点。在 Dijit 组件的逻辑中，就可以对该根节点进行操纵来构建用户界面。组件 JavaScript 类暴露出来的属性和方法就是该组件的接口。

Dijit 组件的使用

Dijit 组件的使用方式非常简单。首先需要在页面上加载组件的 JavaScript 代码，这通过 dojo.require 函数就可以完成。接着在页面上找到或创建一个 DOM 元素作为该组件的根节点。最后通过 new 操作符创建即可。如 new dijit.form.ComboBox({}, node)就可以用 node 作为根元素创建一个 dijit.form.ComboBox 组件，即一个下拉列表选择框。可以看到创建 Dijit 组件的时候，使用了两个参数：第二个参数是组件的根元素，如果创建的时候不指定该根元素，会自动创建一个新的 DIV 元素作为根元素。不过该新创建的根元素一般没有加入到当前的文档树中，可以通过组件的 placeAt 方法来设置该组件在页面文档树中的位置。第一个元素则是一个 JavaScript 对象，包含了组件的配置属性。通常来说，一个 Dijit 组件是可以复用的。因此一般都会提供一些属性供使用者进行配置。通过这个参数，就可以修改这些配置。

上面提到的是程序式的方式创建 Dijit 组件，还有另外的一种方式来进行创建，即通过在 HTML 代码中以声明式的方式创建，如

```
<div dojoType="dijit.Dialog" id="myDialog" title="示例对话框">
    <h3>对话框标题</h3>
    <div>对话框内容</div>
</div>
```

声明式的方式在一定程度上简化了开发人员使用 Dijit 组件的方式。声明式的方式与编写 HTML 代码的形式类似，只需要在一般的 HTML 元素上添加一些额外的属性就可以把 HTML 片段转换成 Dijit 组件。这对于只熟悉 HTML 语言的人来说非常方便，相当于在 HTML 语言的基本元素之上，增加了更多的可用组件。

Dijit 深入分析

Dijit 组件基本类

所有的 Dijit 组件都继承自 dijit._Widget 类。dijit._Widget 类中定义了与组件相关的一系列方法。这些方法中有一些是与组件生命周期相关的，有一些则是所有组件都需要的通用方法。了解 Dijit 组件的生命周期有利于理解 Dijit 组件的运行方式，从而更好的使用已有的组件或开发出自己的组件。

创建 Dijit 组件的过程开始于 dijit._Widget 类中的 create 方法。create 方法采用了典型的模板方法设计模式，即在该方法中封装了创建组件的基本流程。该方法会执行一些重要的操作，并依次调用其它的方法来完成整个创建过程。具体的流程包括：

- 把创建时的配置参数混入（mix-in）到组件中。比如在创建组件的时候使用的方式是 var myWidget = new TestWidget({prop : "Hello"}, node);，那么在创建完成之后就可以通过 myWidget.prop 来获取到"Hello"，在组件中也可以通过 this.prop 来访问。
- 调用生命周期方法：postMixInProperties。该方法在配置参数混入之后调用，可以对混入的参数进行修改。
- 把该新创建出来的组件添加到全局的组件对象注册表中。Dijit 组件都会被分配一个惟一的标识符。添加到注册表中之后，就可以用 dijit.byId 来根据标识符获取组件对象。
- 调用生命周期方法：buildRendering。该方法用来完成构建组件的用户界面。该方法负责设置 this.domNode 的值，表示的是创建完成的组件的根元素。
- 调用生命周期方法：postCreate。该方法在用户界面构建完成之后被调用。一般是组件内部行为逻辑的起点，类似 HTML 页面中的 onload 方法。

对于 Dijit 组件开发人员来说，创建一个新的 Dijit 非常简单。只需要用 dojo.declare 声明一个 JavaScript 类并继承自 dijit._Widget，在该类中覆写感兴趣的 JavaScript 方法即可。最简单的情况是覆写 postCreate 方法并添加组件的逻辑。

对于用来包含其它子组件的容器类组件来说，一般会覆写 startup 方法来让其调用者显式的启动这个组件。这是因为在 postCreate 被调用的时候，只是保证了组件的 DOM 节点已经被创建成功了，但是这些 DOM 节点可能并没有被添加到当前文档树中，因此不能在 postCreate 中包含与 DOM 节点大小和位置相关的代码。如果要添加这样的代码，应该在 startup 中添加。很多容器类组件都使用该方法来对其子节点进行布局。

使用 HTML 模板

如果只是使用 dijit._Widget 的话，编写 Dijit 组件会比较繁琐。比如在构建用户界面的时候，可能会需要很多的 DOM 操作，编写起来也不方便。Dijit 提供了 dijit._Templated 用来使用 HTML 片段来定义组件的内容。HTML 片段是作为组件的内容模板。如：

```
dojo.declare("TempWidget", [dijit._Widget, dijit._Templated], {
   templateString : "<div><span>Hello</span></div>"
});
```

TempWidget 继承了两个 JavaScript 类，除了必需的 dijit._Widget 之外，还有 dijit._Templated 的。需要保证 dijit._Widget 是父类数组的第一个元素，只有它是真正意义上的父类，其余的是混入类。dijit._Templated 类已经覆写了 buildRendering 方法来从 HTML 模板中创建组件内

容的 DOM 元素，并作为组件的 this.domNode 的值。在 HTML 模板中，除了可以使用基本的 HTML 元素和属性之外，还有一些附加的实用功能：

- 在 HTML 模板中直接引用组件中的属性。比如组件中有个属性叫 title，在 HTML 模板中想引用该属性的值，可以直接写${title}。如果属性 title 的值是"Hello"，那么上述模板在运行时刻会变成Hello。
- 通过 dojoAttachPoint 来声明在组件对象中可见的 DOM 节点。当需要在组件中引用某个内部的 DOM 节点时，不需要再次进行查询，通过 dojoAttachPoint 即可。如声明 ，就可以在组件对象中通过 this.myNode 来引用该 SPAN 元素。
- 通过 dojoAttachEvent 来进行事件绑定。这种方式比先手工查询 DOM 节点，再通过 dojo.connect 来绑定要简单得多。如声明 <button dojoAttachEvent="onclick:test">Test</button> ，就意味着将组件的 test 方法绑定到按钮的 onclick 事件上。

dijit._Templated 的模板机制的这些实用功能减少了构建用户界面时的一些繁琐代码。

作为容器

如果组件是作为其它组件的容器来使用的话，就可以混入 dijit._Container 类。该类提供了对子组件的基本管理功能，包括查询、添加和删除等。使用该类的时候，需要在组件中声明一个 containerNode 的属性作为子组件的父节点。创建出 Dijit 组件之后，就可以通过 addChild 方法来添加子组件了。

销毁过程

组件在创建并运行之后，就可能需要被销毁。销毁一个 Dijit 组件很简单，只需要调用 destroyRecursive 方法即可。该方法会负责销毁当前 Dijit 组件及其包含的子组件。当一个组件被销毁的时候，其 uninitialize 方法会被调用，类似于析构函数。因此可以把组件特有的销毁逻辑添加在 uninitialize 方法中。

Dijit 组件的接口与交互

前面提到，组件之间通过设计好的接口和协议进行通讯。对于 Dijit 组件来说，它所提供的接口一般有下面这几类：

- 公开的属性和方法。这些属性和方法类似于 Java 类中的公开的域和方法，在获取到组件对象之后可以直接使用。
- 通过 dojo.connect 进行连接。有些组件提供了一些占位符方法用来允许其使用者监听其内部状态的变化，类似于 DOM 事件的处理。

当组件之间进行通讯和协作的时候，一般有下面几种交互的模式：

- 传递组件对象的引用。这种做法一般是在创建新组件的时候，将其需要引用的组件对象传递进去，如 var anotherWidget = new MyWidget({parent : oneWidget}, node);。
- 不传递对象引用，而是进行查找。这种情况适用于所依赖的组件的 ID 已知的情况。可以通过 dijit.byId 来直接进行查找。
- 使用 Dojo 提供的全局通讯机制：dojo.publish 和 dojo.subscribe。一个组件通过 dojo.publish 来发布消息，另外一个组件则通过 dojo.subscribe 来监听相关的消息并做出处理。

一般来说，比较推荐的做法是第一种，即通过传递组件对象的引用来完成。不过当组件之间的关系比较复杂的时候，有可能需要将一个对象的引用进行多次传递。这个时候也可以考虑后两种做法。

Dijit 开发最佳实践

编写 Dijit 组件并不是一件复杂的事情，只需要按照一般的流程依次完成即可。不过 Dijit 组件本身的设计和实现比较复杂，包含了比较多的内容。下面对一些重点的地方进行讨论。

编程式和声明式的创建方式选择

这两种方式的区别只是在于开发人员的使用方式上。用声明式方式声明的 Dijit 组件，在运行时刻也是通过程序式的方式来进行创建的，由 dojo.parser.parse 方法来完成。因此，声明式的方式更像一种语法糖衣。不过声明式方式的一个好处是可以实现优雅地退化（graceful degradation），即当 JavaScript 不被支持的时候，仍然可以在页面上显示出部分内容。对于简单的和容器类的组件来说，声明式创建的方式比较好。简单的组件用声明式 的方式比较简洁。而容器类的组件在创建的时候一般都需要指定所包含的内容。使用声明式的时候，组件的子节点会自动作为容器类组件的子组件来添加。而如果以程序式的方式来完成的话，需要手工创建子组件，并通过 addChild 方法来逐个添加。代码会比较繁琐。

组件状态变迁与外观样式

在开发 Dijit 组件中，经常会遇到的一个场景是根据组件内部的状态变化改变其外观样式。比如对于一个单选按钮组件来说，选中和未被选中的外观样式是不同的。典型的做法是通过切换不同的 CSS 样式名称来转换外观。比如未被选中适合的 CSS 样式名称可能是 MyRadioButton，被选中之后则变成 MyRadioButtonChecked。Dijiti 提供了 dijit._CssStateMixin 混入类来抽象这种行为。开发人员的组件只需要继承此类，并通过属性 this.baseClass 设置基础的 CSS 样式名称，就具备根据状态的变化动态修改 CSS 样式名称的能力。比如设置了 baseClass 为 myWidget，当鼠标移动到该组件上的时候，其根元素的 CSS 样式名称会自动变成 myWidgetHover。该类所支持的状态变化包括鼠标进入/离开、焦点获取/失去、选中/未选中、启用 /禁用等。

参考资料

- Dojo 中文博客
- Dojo 参考文档
- Dijit 开发最佳实践

原文链接：http://www.infoq.com/cn/articles/cf-dojo-dijit

相关内容：

- Dojo：不容忽视的RIA框架
- Dojo框架：误解与现实
- Dojo的联合创建者Dylan Schiemann就AJAX、Comet、Bayeux、RIAs和Dojo Toolbox的问答
- Cells：将组件开发带入Rails 2.3
- 组件组合的策略与技巧

推荐文章 | Articles

用 Dojo 实现 Ajax 请求：XHR、跨域、及其他

作者 朱小文

在任何浏览器上方便地实现 Ajax 请求是每一个 Ajax 框架的初衷。Dojo 在这方面无疑提供了非常丰富的支持。除了 XMLHttpRequest 之外，动态 script、iframe、RPC 也应有尽有，并且接口统一，使用方便，大多数情况下都只需要一句话就能达到目的，从而免除重复造轮子的麻烦。而且，Dojo 一贯追求的概念完整性也在这里有所体现，换句话说，在使用 Dojo 的 Ajax 工具的过程中不会感到任何的不自然，相反更容易有触类旁通的感觉，因为 API 的模式是统一的，而且这里涉及到的某些概念（如 Deferred 对象）也贯穿在整个 Dojo 之中。

Dojo 的 XHR 函数

Dojo 的 XMLHttpRequest 函数就叫 dojo.xhr，除了把自己取名美元符号之外，这好像是最直接的办法了。它定义在 Dojo 基本库里，所以不需要额外的 require 就能使用。它可以实现任何同域内的 http 请求。不过更常用的是 dojo.xhrGet 和 dojo.xhrPost ,它们只不过是对 dojo.xhr 函数的简单封装；当然根据 REST 风格，还有 dojo.xhrPut 和 dojo.xhrDelete。

这些函数的参数都很统一。除了 dojo.xhr 的第一个参数是 http 方法名之外 ,所有的 dojo.xhr* 系列函数都接受同一种散列式的参数，其中包含请求的细节，例如 url、是否同步、要传给服务器的内容（可以是普通对象、表单、或者纯文本）、超时设定、返回结果的类型（非常丰富且可扩展）以及请求成功和失败时的回调。所有 dojo.xhr*函数（实际上是所有 IO 函数）返回值也都一样，都是一个 Deferred 对象，顾名思义，它能让一些事情“延迟”发生，从而让 API 用起来更灵活。

下面的两个例子可能会带来一点直观感受：

```
dojo.xhrGet({
    url: "something.html",
    load: function(response, ioArgs){
        //用response干一些事
        console.log("xhr get success:", response);
    return response; //必须返回response
    },
```

```
    error: function(response, ioArgs){
        console.log("xhr get failed:", response);
        return response; //必须返回response
    }
});
//Deferred对象允许用同步调用的写法写异步调用
var deferredResult = dojo.xhrPost({
    url: "something.html",
    form: formNode, //Dojo会自动将form转成object
    timeout: 3000, //Dojo会保证超时设定的有效性
    handleAs: "json" //得到的response将被认为是JSON，并自动转为object
});
//当响应结果可用时再调用回调函数
deferredResult.then(function(response){
    console.log("xhr get success:", response);
    return response; //必须返回response
});
```

首先解释一下 timeout。除了 IE8 之外，目前大多数 XMLHttpRequest 对象都没有内置的 timeout 功能，因此必须用 setTimeout。当同时存在大量请求时，需要为每一个请求设置单独的定时器，这在某些浏览器（主要是 IE）会造成严重的性能问题。dojo 的做法是只用一个单独的 setInterval，定时轮询（间隔 50ms）所有还未结束的请求的状态，这样就高效地解决了一切远程请求（包括 JSONP 和 iframe）的超时问题。

值得一提的还有 handleAs 参数，通过设置这个参数，可以自动识别服务器的响应内容格式并转换成对象或文本等方便使用的形式。根据文档，它接受如下值：text（默认）、json、json-comment-optional、json-comment-filtered、javascript, xml。

而且它还是可扩展的。其实 handleAs 只是告诉 xhr 函数去调用哪个格式转换插件，即 dojo.contentHandlers 对象里的一个方法。例如 dojo.contentHandlers.json 就是处理 JSON 格式的插件。你可以方便地定制自己所需要的格式转换插件，当然，你也可修改现有插件的行为：

```
dojo.contentHandlers.json = (function(old){
    return function(xhr){
        var json = old(xhr);
        if(json.someSignalFormServer){
            doSomthing(json);
            delete json.someSignalFormServer;
        }
        return json;
    }
```

```
})(dojo.contentHandlers.json);//一个小技巧，利用传参得到原方法
```

如果要了解每个参数的细节，可以参考 Dojo的文档 。

虚拟的参数类

这里特别提一下 Dojo 在 API 设计上的两个特点。其一是虚拟的参数“类”概念：通过利用 javascript 对象可以灵活扩展的特点，强行规定一个散列参数属于某个“类”。例如 dojo.xhr* 系列函数所接受的参数就称为 dojo.__XhrArgs。这个“类”并不存在于实际代码中（不要试图用 instanceof 验证它），只停留在概念上，比抽象类还抽象，因此给它加上双下划线前缀（Dojo 习惯为抽象类加单下划线前缀）。这样做看起来没什么意思，但实际上简化了 API，因为它使 API 之间产生了联系，更容易记忆也就更易于使用。这一点在对这种类做“继承”时更明显。例如 dojo.__XhrArgs 继承自 dojo.__IoArgs，这是所有 IO 函数所必须支持的参数集合，同样继承自 dojo.__IoArgs 的还有 dojo.io.script.__ioArgs 和 dojo.io.iframe.__ioArgs，分别用于动态脚本请求和 iframe 请求。子类只向父类添加少量的属性，这样繁多的参数就具有了树形类结构。原本散列式参数是用精确的参数名代替了固定的参数顺序，在增加灵活性和可扩展性的同时，实际上增加了记忆量（毕竟参数名不能拼错），使得 API 都不像看起来那么好用，有了参数类的设计就缓解了这个问题。

这种参数类的做法在 Dojo 里随处可见，读源码的话就会发现它们都是被正儿八经地以正常代码形式声明在一种特殊注释格式里的，像这样：

```
/*=====
dojo.declare("dojo.__XhrArgs", dojo.__IoArgs, {
    constructor: function(){
    //summary:
    //...
    //handleAs:
    //...
    //......
    }
 });
 =====*/
```

这种格式可以被 jsDoc 工具自动提取成文档，在文档里这些虚拟出来的类就像真的类一样五脏俱全了。

Deferred 对象

另一个 API 设计特点就是 Deferred 对象的广泛使用。Dojo 里的 Deferred 是基于 MochiKit 实现稍加改进而成的，而后者则是受到 python 的事件驱动网络工具包 Twisted 里同名概念的

启发。概括来说的话，这个对象的作用就是将异步 IO 中回调函数的声明位置与调用位置分离，这样在一个异步 IO 最终完成的地方，开发人员可以简单地说“货已经到了，想用的可以来拿了”，而不用具体地指出到底该调用哪些回调函数。这样做的好处是让异步 IO 的写法和同步 IO 一样（对数据的处理总是在取数据函数的外面，而不是里面），从而简化异步编程。

具体做法是，异步函数总是同步地返回一个代理对象（这就是 Deferred 对象），可以将它看做你想要的数据的代表，它提供一些方法以添加回调函数，当数据可用时，这些回调函数（可以由很多个）便会按照添加顺序依次执行。如果在取数据过程中出现错误，就会调用所提供的错误处理函数（也可以有很多个）；如果想要取消这个异步请求，也可通过 Deferred 对象的 cancel 方法完成。

dojo.Deferred 的核心方法如下：

```
then(callback, errback); //添加回调函数
callback(result); //表示异步调用成功完成，触发回调函数
errback(error); //表示异步调用中产生错误，触发错误处理函数
cancel(); //取消异步调用
```

Dojo 还提供了一个 when 方法，使同步的值和异步的 Deferred 对象在使用时写法一样。例如：

```
//某个工具函数的实现
var obj = {
    getItem: function(){
        if(this.item){
            return this.item; //这里同步地返回数据
        }else{
            return dojo.xhrGet({   //这里返回的是Deferred对象
                url: "toGetItem.html",
                load: dojo.hitch(this, function(response){
                    this.item = response;
                    return response;
                })
            });
        }
    }
};
//用户代码
dojo.when(obj.getItem(), function(item){
    //无论同步异步，使用工具函数getItem的方式都一样
});
```

在函数闭包的帮助下，Deferred 对象的创建和使用变得更为简单，你可以轻易写出一个创建 Deferred 对象的函数，以同步的写法做异步的事。例如写一个使用 store 获取数据的函数：

```
var store = new dojo.data.QueryReadStore({...});
function getData(start, count){
    var d = new dojo.Deferred(); //初始化一个Deferred对象
    store.fetch({
        start: start,
        count: count,
        onComplete: function(items){
            //直接取用上层闭包里的Deferred对象
            d.callback(items);
        }
    });
    return d; //把它当做结果返回
}
```

用 dojo.io.script 跨域

dojo.xhr* 只是 XmlHttpRequest 对象的封装，由于同源策略限制，它不能发跨域请求，要跨域还是需要动态创建<script>标签。Dojo 没有像 JQuery 一样把所有东西都封装在一起（JQuery 的 ajax()方法可以跨域，当然用的是 JSONP，所以它不敢把自己称为 xhr），而是坚持一个 API 只干一件事情。毕竟在大部分应用中，同域请求比跨域请求多得多，如果一个应用不需要跨域，就没必要加载相关代码。因此与 xhr 不同，dojo 的跨域请求组件不在基本库，而在核心库，需要 require 一下才能使用：

```
dojo.require("dojo.io.script");
```

这个包里面基本上只需要用到一个函数：dojo.io.script.get()。它也返回 Deferred 对象，并接受类型为 dojo.io.script.__ioArgs 的散列参数。受益于虚拟参数类，我们不用从头开始学习这个参数，它继承了 dojo.__IoArgs，因此和 dojo.xhr*系列的参数大同小异。唯一需要注意的是 handleAs 在这里无效了，代之以 jsonp 或者 checkString。

前者用于实现 JSONP 协议，其值由服务器端指定，当 script 标签加载后就按照 JSONP 协议执行这个函数，然后 Dojo 会自动介入，负责把真正的数据传给 load 函数。需要指出的是在 Dojo1.4 以前，这个参数叫 callbackParamName，冗长但意义明确。毕竟 Dojo 太早了，它成型的时候（2005）JSONP 这个词才刚出现不久。现在 callbackParamName 还是可用的（为了向后兼容），不过处于 deprecated 状态。

下面的例子从 flickr 获取 feed 数据：

```
dojo.io.script.get({
    url: "http://www.flickr.com/services/feeds/photos_public.gne",
```

```
    jsonp: "jsoncallback", //由flickr指定
    content: {format: "json"},
    load: function(response){
        console.log(response);
        return response;
    },
    error: function(response){
        console.log(response);
        return response;
    }
});
```

与 jsonp 不同，checkString 参数专门用于跨域获取 javascript 代码，它其实是那段跨域脚本里的一个有定义的变量的名字，Dojo 会用它来判断跨域代码是否加载完毕，配合前面提到的 timeout 机制就能实现有效的超时处理。

```
dojo.io.script.get({
    url: "http://.....", //某个提供脚本的URL
    checkString: "obj",
    load: function(response){
        //脚本加载完毕，可以直接使用其中的对象了，如obj。
        Return response;
    }
});
```

用 dojo.io.iframe 传数据

dojo.io 包里还有一个工具就是 iframe，常用于以不刷新页面的方式上传或下载文件。这个很经典的 Ajax 技巧在 Dojo 里就是一句 dojo.io.iframe.send({...})。这个函数接受 dojo.io.iframe.__ioArgs，相比 dojo.__IoArgs，它只多了一个 method 参数，用于指定是用 GET 还是 POST（默认）方法发送请求。下面的例子就实现了通过无刷新提交表单来上传文件：

```
dojo.io.iframe.send({
    form: "formNodeId", //某个form元素包含本地文件路径
    handleAs: "html", //服务器将返回html页面
    load: onSubmitted, //提交成功
    error: onSubmitError //提交失败
});
```

目前 send 函数的 handleAs 参数支持 html, xml, text, json, 和 javascript 五种响应格式。除了 html 和 xml 之外，使用其他格式有一个比较特别的要求，就是服务端返回的响应必须具有以下格式：

```
<html>
    <head></head>
    <body>
```

```
        <textarea>真正的响应内容</textarea>
    </body>
</html>
```

这是因为服务器返回的东西是加载在 iframe 里的，而只有 html 页面才能在任何浏览器里保证成功加载（有个 DOM 在，以后取数据也方便）。加一个<textarea>则可以尽量忠实于原始文本数据的格式，而不会受到 html 的影响。

试试 RPC(远程过程调用)

如果 dojo.xhr*函数以及 Deferred 机制仍然无法避免代码的混乱，那 RPC 可能就是唯一的选择了。dojo.rpc 包提供了基于“简单方法描述”语言(SMD)的 RPC 实现。SMD 的原理类似于 WSDL（Web 服务描述语言），不过是基于 JSON 的，它定义了远程方法的名称、参数等属性，让 Dojo 能创建出代理方法以供调用。

Dojo 提供了两种方式实现 rpc：XHR 和 JSONP，分别对应 dojo.rpc.JsonService 类和 dojo.rpc.JsonpService 类，用户可以根据是否需要跨域各取所需。

一个简单的例子：

```
var smdObj = {
    serviceType: "JSON-RPC",
    serviceURL: "http://...."
    methods: [
        {name: "myFunc", parameters: []}
    ]
};
var rpc = new dojo.rpc.JsonpService(smdObj); //传入SMD
var result = rpc.myFunc(); //直接调用远程方法，返回Deferred对象
dojo.when(result, function(result){
    //得到结果
});
```

SMD 还没有一个被广泛认可的官方标准，一直以来都是 Dojo 社区领导着它的发展，以后这个模块也有可能发生改变，但整个 RPC 基本的框架会保持稳定。

结语

Ajax 请求这个主题太大，本文只能挂一漏万地介绍一点 dojo 在这方面设计和实现的皮毛，包括基本 XHR 请求、动态 script、iframe 请求、以及 RPC，并特别强调了几个有 Dojo 特色的设计，如 timeout 机制、虚拟参数类、Deferred 对象等。

Dojo 由 Ajax 领域的先驱们写就，相信从它的代码中我们一定能学到更多的东西。

参考

- Dojo中文博客
- Dojo 权威指南, Mattbew A. Russell 著，李松峰、李丽 译， O'Reilly, 2009.4
- 精通 Dojo, Rawld Gill,Craig Riecke,Alex Russell 著 ,DOJO 中国 译 ,人民邮电出版社 ,2009.9
- XHR Plugins with Dojo using handleAs, nroberts，2008.4
- Robust Promises with dojo.Deferred 1.5, Kris Zyp，2010.5

原文链接：http://www.infoq.com/cn/articles/dojo-ajax-xhr

相关内容：

- 契约的版本管理、兼容性和可组合性
- W3C发布XML新标准：XProc
- 选择你的Ajax框架
- 用JSF/DWR/DOJO创建动态Web应用
- Ajax应用开发：实践者指南

推荐文章 | Articles

2011 年的 HTML 5 愿景：Michael Mullany 专访

作者 Dionysios G. Synodinos 译者 张龙

近日，来自 Sencha 的 Michael Mullany 发布了一个列表，列举出了 2011 年会从 HTML 5 中获益的诸多内容。InfoQ 有幸采访到了 Michael 以深入了解其愿景。

Michael 最初的列表包含如下内容：

- 持续不断地推进 Webkit 迈向 CSS 3
- CSS 3：富效果工具箱
- 面向移动的高性能定位
- GPU 加速
- 更深层的设备访问
- 面向移动浏览器的更好的调试工具
- Web Sockets 趋于稳定
- IE 9 将完整支持 CSS 3
- HTML5 codec 大战宣告结束
- WebSQL 标准化进程的再次启动

InfoQ 能否说说你认为应该添加到 CSS 3 规范中的最重要的特性列表？这些效果会向开发者提供哪些新功能呢？

Michael：目前，最重要的事情就是实现跨浏览器的特性。但我们最大的期望是将更多 WebKit 所独有的效果（并不属于标准栈）推进到 CSS 草案中。这包括高级的文本样式、遮罩以及字体效果。

InfoQ：由于 IE 采用新 Web 标准和技术的速度非常慢，你觉得这些效果何时才能真正为桌面

开发者所用呢？

Michael：我认为 IE 在功能上越拖后腿，越来越多的人们，甚至是保守派的企业就会开始转向 Chrome 和 Firefox 以利用其现代化的功能。我们认为框架（如 Sencha Ext JS）会得到越来越多的应用，因为他们能够通过 IE 所独有的技术如 VML 和 Filter 创建兼容于 IE 的实现，无需开发者学习 IE 所独有的技术。

InfoQ：你提到目前需要像 Sencha Touch 这样的框架以便实现固定位置的 UI 元素，为什么呢？

Michael：Mobile WebKit 并不支持固定位置的 UI 元素；支持该特性的 CSS 已经被浏览器关闭了。这是因为浏览器针对面向桌面屏幕的网页进行了优化。

InfoQ：你提到需要更好的调试工具，而 Senche 已经发布了用于 Android 设备远程脚本调试的解决方案。不久的将来，针对 HTML 5 特有的功能如 Canvas、Animation 以及 Websockets 等还会出现哪些工具呢？

Michael：现在已经有很多面向桌面的调试工具了，因此访问移动浏览器内核是最为重要的，而非创建具体的工具。

InfoQ：Websocket 的使用和规范本身有什么问题呢？

Michael：问题在于浏览器不断升级携带的 Websockets 协议，同时规范本身也在不断演进。比如说，Chrome 4 所带的 WebSockets 现在就已经过时了（幸好，现在几乎没人使用 Chrome 4）。iPad 最初带有一个 WebSockets 实现，然后又移除了，后来又加回来了。Firefox 4 Beta 最初也带有一个 WebSockets 实现，然后由于安全问题被移除了：最近，有人发现少数代理服务器（< 5%）并不能正确处理 Websockets 所使用的 HTTP 机制。这样就存在这种可能：攻击者可能会利用这些服务器。虽说这可能是代理服务器的 Bug，但文章作者还是建议 Websocket 不应该再使用这种机制了，应该使用其他方案。

InfoQ：你关于视频 codec 的愿景似乎很快就要实现了，因为 Google 宣布将要放弃 H.264。另一方面，目前 H.264 得到了广泛的应用，并且对于免费的互联网内容是免费的，那内容提供者们为何还会费力迁移到 WebM 上呢？如果发布的是付费内容，难道就不需要数字版权管理、高级的流选项么，而这些内容只有 Flash 才有。

Michael：H.264 对于发布视频来说是免费的，但编码与转换却是收费的；在这种情况下我们仍然需要支付专利费。Flash 和 Silverlight 都提供了高级的 DRM 和流视频选项（我相信 Xbox 的 Netflix 流客户端是基于 Silverlight 的，可以动态调整流速率而无需重新缓冲）。问题在于我们需要向那些对 DRM 不感兴趣的用户提供没有任何阻碍的 codec。

InfoQ：你认为 W3C 放弃对关系浏览器存储（WebSQL）的支持而转向层次存储（IndexedDb）的原因何在？为何认为应该重提 WebSQL 标准呢？

Michael：原因很明确：为了让技术成为标准，需要有两个根据规范所构建出来的独立实现才行。完整规范的中心概念是人们所编写的浏览器应能处理遵循该规范的任何内容。对于 WebSQL 来说，所有实现只不过是嵌入到浏览器中的 SQLite 而已，没人想从头开始编写全新的 SQL 实现。此外，Mozilla 阵营认为 SQL 太过重量级、查询机制不够优雅，我们不该再使用它了。

InfoQ：对于 2011 年的 HTML 5，你还有何展望呢？

Michael：我希望更快的 Canvas 能够出现在列表中。

对 HTML 5 感兴趣的读者可以参加我们的 QCon London，其中有个主题是"HTML5，the Platform"！

原文链接：http://www.infoq.com/cn/news/2011/01/html5-wish-list-for-2011

相关内容：

- HTML5 已经起飞
- Flash vs. HTML5：RIA领域当前的混战
- Google Wave加速HTML5 发展？
- 当HTML5 来敲门
- HTML 5 案例研究：使用WebSockets、Canvas与JavaScript构建noVNC客户端

推荐文章 | Articles

Java 深度历险（三）——Java 线程：基本概念、可见性与同步

作者 成富

开发高性能并发应用不是一件容易的事情。这类应用的例子包括高性能 Web 服务器、游戏服务器和搜索引擎爬虫等。这样的应用可能需要同时处理成千上万个请求。对于这样的应用，一般采用多线程或事件驱动的架构。对于 Java 来说，在语言内部提供了线程的支持。但是 Java 的多线程应用开发会遇到很多问题。首先是很难编写正确，其次是很难测试是否正确，最后是出现 问题时很难调试。一个多线程应用可能运行了好几天都没问题，然后突然就出现了问题，之后却又无法再次重现出来。如果在正确性之外，还需要考虑应用的吞吐量 和性能优化的话，就会更加复杂。本文主要介绍 Java 中的线程的基本概念、可见性和线程同步相关的内容。

Java 线程基本概念

在操作系统中两个比较容易混淆的概念是进程（process）和线程（thread）。 操作系统中的进程是资源的组织单位。进程有一个包含了程序内容和数据的地址空间，以及其它的资源，包括打开的文件、子进程和信号处理器等。不同进程的地址 空间是互相隔离的。而线程表示的是程序的执行流程，是 CPU 调度的基本单位。线程有自己的程序计数器、寄存器、栈和帧等。引入线程的动机在于操作系统中阻 塞式 I/O 的存在。当一个线程所执行的 I/O 被阻塞的时候，同一进程中的其它线程可以使用 CPU 来进行计算。这样的话，就提高了应用的执行效率。线程的概 念在主流的操作系统和编程语言中都得到了支持。

一部分的 Java 程序是单线程的。程序的机器指令按照程序中给定的顺序依次执行。Java 语言提供了 java.lang.Thread 类来为线程提供抽象。有两种方式创建一个新的线程：一种是继承 java.lang.Thread 类并覆写其中的 run()方法，另外一种则是在创建 java.lang.Thread 类的对象的时候，在构造函数中提供一个实现了 java.lang.Runnable 接口的类的对象。在得到了 java.lang.Thread 类的对象之后，通过调用其 start()方法就可以启动这个线程的执行。

一个线程被创建成功并启动之后，可以处在不同的状态中。这个线程可能正在占用 CPU 时间运行；也可能处在就绪状态，等待被调度执行；还可能阻塞在某个资源或是事件上。多个就绪状态的线程会竞争 CPU 时间以获得被执行的机会，而 CPU 则采用某种算法来调度线程的执行。不同线程的运行顺序是不确定的，多线程程序中的逻辑不能依赖于 CPU 的调度算法。

可见性

可见性（visibility）的问题是 Java 多线程应用中的错误的根源。在一个单线程程序中，如果首先改变一个变量的值，再读取该变量的值的时候，所读取到的值就是上次写操作写入的值。也就是说前面操作的结果对后面的操作是肯定可见的。但是在多线程程序中，如果不使用一定的同步机制，就不能保证一个线程所写入的值对另外一个线程是可见的。造成这种情况的原因可能有下面几个：

- CPU 内部的缓存：现在的 CPU 一般都拥有层次结构的几级缓存。CPU 直接操作的是缓存中的数据，并在需要的时候把缓存中的数据与主存进行同步。因此在某些时刻，缓存中的数据与主存内的数据可能是不一致的。某个线程所执行的写入操作的新值可能当前还保存在 CPU 的缓存中，还没有被写回到主存中。这个时候，另外一个线程的读取操作读取的就还是主存中的旧值。
- CPU 的指令执行顺序：在某些时候，CPU 可能改变指令的执行顺序。这有可能导致一个线程过早的看到另外一个线程的写入操作完成之后的新值。
- 编译器代码重排：出于性能优化的目的，编译器可能在编译的时候对生成的目标代码进行重新排列。

现实的情况是：不同的 CPU 可能采用不同的架构，而这样的问题在多核处理器和多处理器系统中变得尤其复杂。而 Java 的目标是要实现“编写一次，到处运行”，因此就有必要对 Java 程序访问和操作主存的方式做出规范，以保证同样的程序在不同的 CPU 架构上的运行结果是一致的。Java 内存模型（Java Memory Model）就是为了这个目的而引入的。JSR 133 则进一步修正了之前的内存模型中存在的问题。总得来说，Java 内存模型描述了程序中共享变量的关系以及在主存中写入和读取这些变量值的底层细节。Java 内存模型定义了 Java 语言中的 synchronized、volatile 和 final 等关键词对主存中变量读写操作的意义。Java 开发人员使用这些关键词来描述程序所期望的行为，而编译器和 JVM 负责保证生成的代码在运行时刻的行为符合内存模型的描述。比如对声明为 volatile 的变量来说，在读取之前，JVM 会确保 CPU 中缓存的值首先会失效，重新从主存中进行读取；而写入之后，新的值会被马上写入到主存

中。而 synchronized 和 volatile 关键词也会对编译器优化时候的代码重排带来额外的限制。比如编译器不能把 synchronized 块中的代码移出来。对 volatile 变量的读写操作是不能与其它读写操作一块重新排列的。

Java 内存模型中一个重要的概念是定义了“在之前发生（happens-before）”的顺序。如果一个动作按照“在之前发生”的顺序发生在另外一个动作之前，那么前一个动作的结果在多线程的情况下对于后一个动作就是肯定可见的。最常见的“在之前发生”的顺序包括：对一个对象上的监视器的解锁操作肯定发生在下一个对同一个监视器的加锁操作之前；对声明为 volatile 的变量的写操作肯定发生在后续的读操作之前。有了“在之前发生”顺序，多线程程序在运行时刻的行为在关键部分上就是可预测的了。编译器和 JVM 会确保“在之前发生”顺序可以得到保证。比如下面的一个简单的方法：

```
public void increase() {
    this.count++;
}
```

这是一个常见的计数器递增方法，this.count++实际是 this.count = this.count + 1，由一个对变量 this.count 的读取操作和写入操作组成。如果在多线程情况下，两个线程执行这两个操作的顺序是不可预期的。如果 this.count 的初始值是 1，两个线程可能都读到了为 1 的值，然后先后把 this.count 的值设为 2，从而产生错误。错误的原因在于其中一个线程对 this.count 的写入操作对另外一个线程是不可见的，另外一个线程不知道 this.count 的值已经发生了变化。如果在 increase() 方法声明中加上 synchronized 关键词，那就在两个线程的操作之间强制定义了一个“在之前发生”顺序。一个线程需要首先获得当前对象上的锁才能执行，在它拥有锁的这段时间完成对 this.count 的写入操作。而另一个线程只有在当前线程释放了锁之后才能执行。这样的话，就保证了两个线程对 increase()方法的调用只能依次完成，保证了线程之间操作上的可见性。

如果一个变量的值可能被多个线程读取，又能被最少一个线程锁写入，同时这些读写操作之间并没有定义好的“在之前发生”的顺序的话，那么在这个变量上就存在数据竞争（data race）。数据竞争的存在是 Java 多线程应用中要解决的首要问题。解决的办法就是通过 synchronized 和 volatile 关键词来定义好“在之前发生”顺序。

Java 中的锁

当数据竞争存在的时候，最简单的解决办法就是加锁。锁机制限制在同一时间只允许一个线程访问产生竞争的数据的临界区。Java 语言中的 synchronized 关键字可以为一个代码块或是方法进行加锁。任何 Java 对象都有一个自己的监视器，可以进行加锁和解锁操作。当受到 synchronized 关键字保护的代码块或方法被执行的时候，就说明当前线程已经成功的获取了

对象的监视器上的锁。当代码块或是方法正常执行完成或是发生异常退出的时候，当前线程所获取的锁会被自动释放。一个线程可以在一个 Java 对象上加多次锁。同时 JVM 保证了在获取锁之前和释放锁之后，变量的值是与主存中的内容同步的。

Java 线程的同步

在有些情况下，仅依靠线程之间对数据的互斥访问是不够的。有些线程之间存在协作关系，需要按照一定的协议来协同完成某项任务，比如典型的生产者-消费者模式。这种情况下就需要用到 Java 提供的线程之间的等待-通知机制。当线程所要求的条件不满足时，就进入等待状态；而另外的线程则负责在合适的时机发出通 知来唤醒等待中的线程。Java 中的 java.lang.Object 类中的 wait/notify/notifyAll 方法组就是完成线程之间的同步的。

在某个 Java 对象上面调用 wait 方法的时候，首先要检查当前线程是否获取到了这个对象上的锁。如果没有的话，就会直接抛出 java.lang.IllegalMonitorStateException 异常。如果有锁的话，就把当前线程添加到对象的等待集合中，并释放其所拥有的锁。当前线程被阻塞，无法继续执行，直到被从对象的等待集合中移除。引起某个线程从对象的等待集合中移除的原因有很多：对象上的 notify 方法被调用时，该线程被选中；对象上的 notifyAll 方法被调用；线程被中断；对于有超时限制的 wait 操作，当超过时间限制时；JVM 内部实现在非正常情况下的操作。

从上面的说明中，可以得到几条结论：wait/notify/notifyAll 操作需要放在 synchronized 代码块或方法中，这样才能保证在执行 wait/notify/notifyAll 的时候，当前线程已经获得了所需要的锁。当对于某个对象的等待集合中的线程数目没有把握的时候，最好使用 notifyAll 而不是 notify。notifyAll 虽然会导致线程在没有必要的情况下被唤醒而产生性能影响，但是在使用上更加简单一些。由于线程可能在非正常情况下被意外唤醒，一般需要把 wait 操作放在一个循环中，并检查所要求的逻辑条件是否满足。典型的使用模式如下所示：

```
private Object lock = new Object();
synchronized (lock) {
    while (/* 逻辑条件不满足的时候 */) {
       try {
           lock.wait();
       } catch (InterruptedException e) {}
    }
    //处理逻辑
}
```

上述代码中使用了一个私有对象 lock 来作为加锁的对象，其好处是可以避免其它代码错误的使用这个对象。

中断线程

通过一个线程对象的 interrupt()方 法可以向该线程发出一个中断请求。中断请求是一种线程之间的协作方式。当线程 A 通过调用线程 B 的 interrupt()方法来发出中断请求的时候，线程 A 是在请求线程 B 的注意。线程 B 应该在方便的时候来处理这个中断请求，当然这不是必须的。当中断发生的时候，线程对象中会有一个标记来记录当前的中断状态。通过 isInterrupted()方法可以判断是否有中断请求发生。如果当中断请求发生的时候，线程正处于阻塞状态，那么这个中断请求会导致该线程退出阻塞状态。可能造成线程处于阻塞状态的情况有：当线程通过调用 wait()方法进入一个对象的等待集合中，或是通过 sleep()方法来暂时休眠，或是通过 join()方法来等待另外一个线程完成的时候。在线程阻塞的情况下，当中断发生的时候，会抛出 java.lang.InterruptedException，代码会进入相应的异常处理逻辑之中。实际上在调用 wait/sleep/join 方法的时候，是必须捕获这个异常的。中断一个正在某个对象的等待集合中的线程，会使得这个线程从等待集合中被移除，使得它可以在再次获得锁之后，继续执行 java.lang.InterruptedException 异常的处 理逻辑。

通过中断线程可以实现可取消的任务。在任务的执行过程中可以定期检查当前线程的中断标记，如果线程收到了中断请求，那么就可以终止这个任务的执行。当遇到 java.lang.InterruptedException 的异常，不要捕获了之后不做任何处理。如果不想在这个层次上处理这个异常，就把异常重新抛出。当一个在阻塞状态的线程被中断并且抛出 java.lang.InterruptedException 异常的时候，其对象中的中断状态标记会被清空。如果捕获了 java.lang.InterruptedException 异常但是又不能重新抛出的话，需要通过再次调用 interrupt()方法来重新设置这个标记。

参考资料

- 现代操作系统
- Java 语言规范（第三版）第 17 章：线程与锁
- Java 内存模型 FAQ
- Fixing the Java Memory Model, Part 1 & Part 2

原文链接：http://www.infoq.com/cn/articles/cf-java-thread

相关内容：Java 深度历险（一）--Java 字节代码的操纵、Java 深度历险（二）--Java 类的加载、链接和初始化、在不改变语言的前提下如何推进 Java 的不断演进、代码之丑（十）--条件编译那些事儿、书评：《代码之道》

时刻关注企业软件开发领域的变化与创新

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

新品推荐 | New Products

Apache Ant 1.8.2 发布了

作者 张龙

近日，Apache Ant 开发团队发布了最新版的 Ant 1.8.2，在引入了诸多新特性的同时又修复了之前版本中存在的众多 Bug。

原文链接：http://www.infoq.com/cn/news/2011/01/apache-ant-1.8.2-release

JBoss Application Server 6 发布了

作者 张龙

在 JCP 批准 Java EE 6 后，经过一年多的开发，JBoss Application Server 6.0 终于发布了。此次的 GA 版全面支持 Java EE 6 所引入的激动人心的新技术：JPA 2、Bean Validation、CDI、EJB 3.1、Servlets 3.0 以及 JSF 2.0 等等。

原文链接：http://www.infoq.com/cn/news/2011/01/jboss-as6-release

HTML5 Labs 上线、WebSocket 更新

作者 James Vastbinder 译者 张龙

近日，微软发布了 HTML5 Labs， 旨在为.NET 开发者提供一个平台，可以运行使用了不稳定的 Web 标准规范的应用原型。其意图在于拉紧与开发者社区的关系，并且尽最大可能实现草案规范。 首先发布的两个原型是 IndexDB（基于 Web 规范草案的一个应用，用于在浏览器中存储大容量的结构化数据）和一个示例性的 WebSocket 实现。

原文链接：http://www.infoq.com/cn/news/2011/01/HTML5-WebSocket

Activiti 5.0 GA 发布——基于 Apache 2 许可的 BPMN 2 引擎

作者 Josh Long **译者** 张龙

上个月，Alfresco 发布了 Activiti 5.0 GA——基于 Apache 2 许可的 BPMN 2.0 实现。InfoQ 有幸采访到了该项目领导 Tom Baeyens 一探究竟。Tom 还是 jBPM 项目的创建者和前任项目领导。

原文链接：http://www.infoq.com/cn/news/2011/01/activiti_5_released

Mono 发布 Mono Packager for Mac

作者 James Vastbinder **译者** 张龙

本周初，Mono 团队发布了面向 OS X 的 Mono Packager 并更新了 MonoMac 库与模板。开发者现在可以创建自我包含的 Mono 应用，并且可以通过 Apple App Store 发布。通过此次更新，开发者可以在 MonoDevelop 中创建包、安装器并且可以进行签名。

原文链接：http://www.infoq.com/cn/news/2011/01/mono-mac-packager

JRuby 1.6 发布首个 RC 版本

作者 丁雪丰

JRuby 在众多 Ruby VM 社区中称得上是比较活跃的一个社区，几乎每个月都会发布新版本，继去年 12 月发布了 1.5.6 之后，近日，JRuby 团队“如期”在其官方网站发布了 JRuby 1.6 的首个 RC 版本，即 1.6.0.RC1。

原文链接：http://www.infoq.com/cn/news/2011/01/jruby-1.6-rc1

Amazon 发布 Appstore 开发者门户

作者 Abel Avram **译者** 杨晨

Amazon 最近发布了 Appstore 开发者门户，为即将到来的 Android Appstore 做准备。即将使用的这个商店模型在审查过程和价格设定上均异于 Google 的 Marketplace。

原文链接：http://www.infoq.com/cn/news/2011/01/Amazon-Appstore

Google 发布移动 Web 性能工具 PCAP Web Performance Analyzer

作者 崔康

最近，Google 的 Page Speed 团队一直在关注如何更好的优化移动 Web 应用性能。虽然我们拥有类似 Firebug、Page Speed、YSlow!和 Chrome/Safari Developer Tools 等优秀工具，但是移动浏览器缺少这样的性能分析利器。为此，Page Speed 团队发布了一个分析移动浏览器网络信息的工具——PCAP Web Performance Analyzer。

原文链接：http://www.infoq.com/cn/news/2011/01/google-pcapperf

HP 的云产品公告：混合云

作者 James Vastbinder **译者** 马国耀

HP 日前公布了它的 **HP 混合云产品**，这是一整套云计算基础设施。HP 旨在通过它与 Amazon AWS 和 Rackspace 的产品抗衡。HP 称之为**"聚合的基础设施"**架构模型。它提供整套的计算及管理功能表以支持云中的应用托管。HP 试图从几个方面标新立异以区别于现有的产品。

原文链接：http://www.infoq.com/cn/news/2011/01/hp-hybrid-delivery

推荐编辑 | 翻译团队编辑 张善友

大家好，很高兴在这期《架构师》和大家见面。我叫张善友，英文名 Geff Zhang，在 InfoQ

担任.NET 社区编辑 ,职业是一名程序员 ,目前任职于腾讯科技企业 IT 部，高级程序员。关注前沿技术 ,并致力于开源社区与微软平台的组合应用，致力于 Mono 平台的在社区中的推广。

大学就读的专业是计算数学与应用软件，受到严格的数学训练为我日后从事软件开发打下坚实的基础。软件工程全跟抽象相关，它的每一个概念，观点，以及方法，都是完全抽象的。毕业论文是设计一个软件，当时使用的语言工具是 PowerBuilder，大学刚毕业的时候从事的工作并不是软件开发，而是当时最火的系统集成。在工作中开始接触到 Linux 等开源软件，也接触到.NET，Mono。Miguel de Icaza 创立 Mono 项目的目标是通过这些创建出来的工具让用户可以在更短时间内创建出更多的应用程序从而缩短开发周期和降低开发成本。在工作两年后全面转向软件开发，在工作中也一直保持对开源软件的关注和应用。

.NET 技术发展很快，经常在社区听到这样的声音：微软经常淘汰自己的技术，跟着微软后面程序员很累，难以积累。微软的.NET 技术替代 COM 技术，经过了 10 年的发展，.NET 已经在企业应用开发领域站稳，全面带入云计算领域，随着 Mono 在近两年的创造性的发展，.NET 环境已经进入了 iOS，Android，以及 Mac OS 原生平台。可以说.NET 正处于最辉煌的时代，将我们的.NET 技能带入现今最流行的领域。

.NET 的整个发展历程，我们可以看到微软的务实的策略，它的每一步都是渐进的、踏实的，富有底蕴，不断地拥抱不同的技术领域。在我的工作用使用 Iron Python/Ruby，Mono，Linux，Windows 等等的技术，他们相互配合，享受快乐的程序员生活。

InfoQ 是一个综合性的技术媒体，不局限于某个特定技术领域，这有助于帮助我们保持开放的眼界和心态。在朱永光的介绍之下，有机会加入到 InfoQ 的建设中来，加入到 InfoQ 以后认识了许多各个领域优秀的朋友，从他们的身上也学到很多。

封面植物

独花兰

独花兰，稀有种。陆生直立草本，为中国特有的单种属植物。因连年乱砍林木和采挖入药，生境渐趋恶化，植株日益减少，浙江和江苏产地已趋于灭绝状态。被列为国家二级保护植物。

生长分布：喜云雾多，湿度大，气温低的气候类型。分布于江苏、浙江、安徽、江西、湖南、湖北、陕西和四川等地部分海拔 400 至 1500 米阴坡落叶常绿阔叶混交林下潮湿的沟边、山谷岩壁下。

植物形态：陆生兰。假鳞茎具节，淡黄白色，顶生 1 枚叶。叶具柄，叶片宽卵形或宽椭圆形，长 7-11 厘米，宽 4.5-8 厘米，全缘，下面带紫红色；叶柄长 5.5-9.5 厘未。3-4 月开花，花葶上有 2-3 枚鞘状退化叶；花单朵顶生，直径 5.5-7 厘米，常淡紫色，罕为白色；萼片 3，长圆状披针形；花瓣倒卵状匙形；唇瓣宽大，3 裂，侧裂片斜卵形，直立，围抱蕊柱，中裂肾形，边缘稍皱波，内面唇盘上具 5 条纵褶片，具紫红色疣点和细点，基部具长约 2 厘米稍弯曲的距。花期 3-4 月。

通常情况下，独花兰于 9 月中下旬地下鳞茎萌动并开始展叶(单叶)，保持这个状态度过冬季，次年 3 月中下旬开花。花期约 3–4 周，至 4 月中、下旬花谢，4 月下旬至 5 月上旬叶片逐渐枯死。此时如果独花兰已经受粉、结实，其花葶和果实继续发育，到 8 月底果实成熟、散播。独花兰种子数量众多、细小如尘。开花期间群落内的林木尚未完全展叶，郁闭度较低，其鲜艳的花在群落内很显眼，对吸引传粉者有重要作用。

用途：优良的野生花卉，假鳞茎是治疗疮毒与蛇伤的良药。

1kg.org 多背一公斤

爱自然 | 更爱孩子

架构师 2 月刊

每月 8 日出版

本期主编：马国耀

总编辑：霍泰稳

流程编辑：葛明见

美术编辑：胡伟红

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：
http://t.sina.com.cn/infoqchina

商务合作：sales@cn.infoq.com 13581658359

本期主编：马国耀，InfoQ 中文站翻译团队编辑

马国耀，关注企业级应用相关的开发、架构及思想的发展。尤其对 Java EE、SOA、ESB 和 Cloud Computing 等领域持有浓厚兴趣。入行 8 余载，虽道行尚浅，但执着于每天进步一点点。现就职于 IBM 北亚太战略合作部，为合作伙伴提供产品培训、技术支持和架构咨询等服务。他曾获得吉林大学学士学位和北京大学硕士学位。译著有《云计算与 SOA》。您可以通过 maguoyao[at]gmail.com 联系到他。其在 InfoQ 的编辑特辑为：马国耀。

InfoQ
Enterprise Software Development Community

《架构师》月刊由 InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ neue
Enterprise Software Development Community

商务合作：sales@cn.infoq.com
读者反馈/投稿：editors@cn.infoq.com