时刻关注企业软件开发领域的变化与创新

架构师

2月 ARCHITECT

现在的技术趋势

Flex RIA生态圈现状分析

十年SOA：当前的位置和未来的方向

站点恢复过程中的经验和教训

虚拟研讨会：软件架构文档

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语 | Editor's letter

QCon北京 2010，现在进行式

时间过得真快，总感觉去年 4 月份举办的 QCon 北京 2009 就在昨天，而那些讲师也好像刚从台上走下来和参会者进行交流。也许是值得留念的太多，才让自己对过去那么难以释怀。三个人，一台戏，从头走到尾。很多朋友听过之后直咂舌，说这怎么可能，那么大场面的背后竟然只有三个人操刀？事情就是这样，在很多朋友的帮助下，从讲师的质量，到现场的组织，QCon 北京 2009 的反响都还算不错。比如包括 ThoughtWorks 的首席科学家 Martin Fowler、Spring 的创始人 Rod Johnson、eBay 的资深架构师 Randy Shoup 等，都前来捧场。国的讲师也不逊色，现在依然有人提起豆瓣网首席架构师洪强宁和淘宝网架构师岳旭强两位演讲时间的撞车，使得“鱼和熊掌不可兼得”。也许正因为这些出色的演讲嘉宾，大会在开始前两周门票就售罄，有些朋友希望“走后门”进来，终因场地所限未能成行，不知道现在是否还在遗憾。

社会总是在进步，尽管依然有很多不足。对 InfoQ 中文站来说，我们的团队规模不断扩大，在 QCon 北京大会的组织上，也有了更多的经验，也希望能够比去年做得更好。天助自助者，和去年一样，我们的真诚依然打动了许多知名国际讲师，包括 Eclipse 之父、《设计模式》的作者 Erich Gamma，JSON 作者、Yahoo！资深架构师和《JavaScript 语言精粹》的作者 Douglas Crockford，Jolt 图书大奖获得者 Michael Nygard 等，以及令很多互联网技术从业人员兴奋的——来自 Facebook 和 Twitter 的架构师等。当有好事者在 Twitter 等微博工具上发布这一消息时，引来众人围观，反应出来的直接效果是报名者激增。网站发布还不到一周，已经有数十人报名并确认参会。而这，也更让我们组织者如履薄冰，更加努力，因为我们可不想让大家的期望化为失望，不想让参会者白花了自己的真金白银。

QCon 北京 2010 的演讲嘉宾确认依然在继续，各项组织工作也按部就班地进行，没有喧天的张扬，只有静默的踏实，因为我们认为在静默中才能更加孕育爆发的力量。在互联网领域，很难说要将某一个网站或者品牌做到“百年老店”，但是尽量做到品牌的持久高质量，却是可以的，这也是 QCon 希望能够做到的。QCon 北京 2010，2010 年 4 月 23~25 日，我们期待您的光临，和我们一起见证这一全球知名技术大会品牌的影响力。

马上就是 2010 年春节了，我也代表 InfoQ 中文站的管理团队成员和编辑们，祝读者朋友节日愉快，有一个轻松的心情，并感谢对 InfoQ 中文站一路走来的陪伴！

本期主编： 霍泰稳

QCon是由C4Media媒体集团InfoQ网站主办的全球顶级技术盛会，每年在伦敦、旧金山、东京、北京召开。2010年，QCon全球企业开发大会（北京站）将于4月23日～25日，在北京京仪大酒店举行。大会将开设包括架构、语言、敏捷、SOA、BAAP、案例在内的6大主题，邀请40多名国内外著名讲师授课，预计将有200多家企业、500多名来自业内的资深人士参会。

QCon 全球企业开发大会（北京站）

马上报名

2010年4月23日～25日 北京京仪大酒店

已邀嘉宾（部分）

Erich Gamma
Eclipse之父、JUnit联合作者

Michael Nygard
Jolt大奖图书《Release It》作者

Douglas Crockford
JavaScript开发权威，JSON之父

Marc Kwiatkowski
Facebook资深架构师

Nick Kallen
Twitter系统架构师

Jim Webber
ThoughtWorks全球首席架构师

Paul King
Groovy顶级贡献者

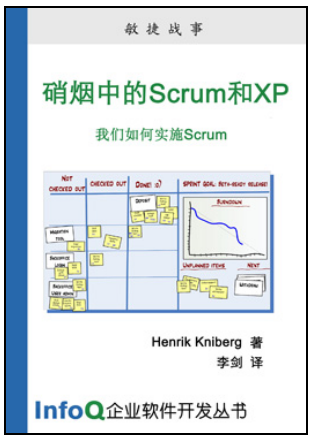
Dylan Schiemann
Dojo Toolkit联合创始人

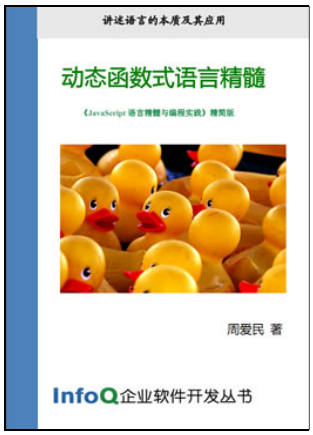
www.QConBeijing.com

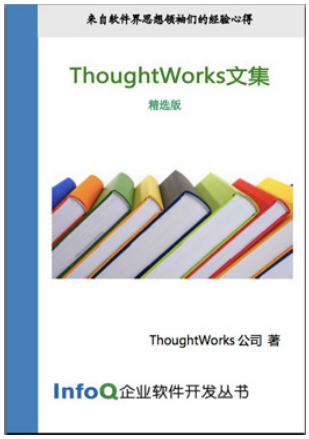
QCon@cn.infoq.com

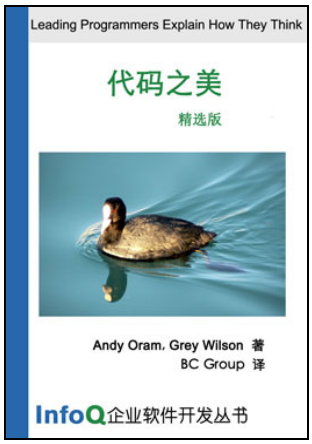
InfoQ企业软件开发丛书

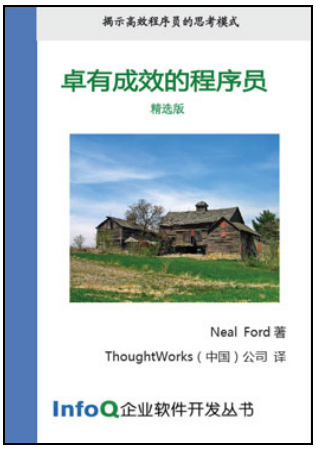
欢迎免费下载

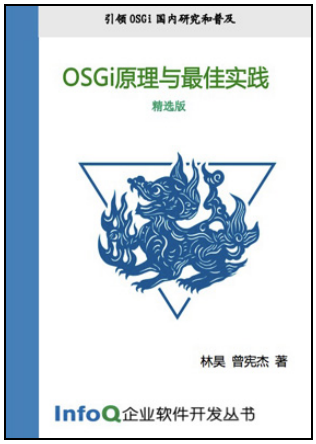












商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

目　录

[篇首语]

[人物专访]

[热点新闻]

[推荐文章]

[新品推荐]

[封面植物]

人物专访 | Interview

十年SOA：当前的位置和未来的方向

很少有人记得微软BizTalk服务器曾经是SOAP、WSDL和UDDI发展的幕后主要推动者之一——它们主要作为一种跟ebXML标准竞争的B2B技术，“互操作性”一词在当时还未成为时髦用语。那个时候的许多专家和分析师都草率地得出结论：

> Web 服务是一种新型模型 ,其本质是基于 Web 的对象的面向对象编程。2000 年 4 月，SOAP1.1 规范朝着这个方向迈出了一 步 ,规范中描述了如何将 XML 格式的消息用于请求和响应。在整个拼图中，UDDI 负责让业务可以发现这些服务。Web 服务描述语言（ WSDL ）则是描述服务和服务提供者的 XML 词汇。因此[……]：Web 服务 = SOAP + UDDI + WSDL。

十年过去了，我们仍然在纠结于服务或者基于Web的对象的含义，甚至不去考虑SOA曾经对信息系统建设是多么的重要。

2009 年，InfoQ 召开了一次虚拟研讨会，与会者都是在这十个年头中大多数时间内体验和实施过 SOA 的企业架构师，我们期望能借这次会议更好地理解 SOA 对于 IT 的意义。

会议的参加者包括 Jeff Andres、来自 MomentumSI 的 Eric Ballou、来自 Saber 的 Dave Hollander 和来自 Carlson Wagonlit Travel 的 William El Kaim。

InfoQ：我们已经花了 10 年来做 SOA。对此，您作何感想？未来十年，SOA 将会是个什么样子？

Eric：哇，我简直无法相信我们从事这个行当已经 10 年了，但这是千真万确。大体来讲，让我感到震惊的是，在厂商和大牛们在花了这 么多精 力去教育人们之后，还有这么多的公司认为只要写个Web服务就是SOA了。那些尚未踏上SOA之路的企业将继续为了提升自己的成熟度而奋斗，由于缺乏坚实 的基础，他们的业务和信息服务团队将付出极大的努力：相对于他们那些已经走上SOA之路并且已经开窍的竞争对手而言，他们面临的是一个非常高的市场技术入 口。所幸，如Gartner指出的，SOA正进入“启蒙的斜坡”。

SOA 的未来将会是：随着厂商将焦点转移到以业务/用户为中心的服务和产品上，它开始被

打上新方法论和实践的标签。这些标签，诸如 Web3.0、业 务驱动 架构和给最终用户授权的用户中心（User Centricity），就是这个方向上的一个极为隐蔽的转变，又或象 Yahoo 给它贴上的：Y!ou。信息服务必须很到位（参见服务架构），才能使授权 到位，SOA/BPM/云只是这个方向上的一小步。

Jeff：看看 SOA，其本质非常类似于 80 年代就已经完成的功能设计和构造。回顾从当时到现在我们已经了解的事实，然后再对未来进 行预 测，会发现"事件"驱动架构、基于流程的架构这些领域也跟着成熟起来，同时 SOA 也变得更轻量级——因为新语言、介质和设备要求它这样。那么，为什么我会 说是这些领域，以及如何到达这些领域呢？让我们先回过头来了解 SOA 今天所处的位置，以及它可能的方向和进入的领域。在我看来，SOA 目前并没有触及实时 和事件驱动领域，因此，"事件驱动"的语言和构造（中断、分支、汇合）以及类似需求就需要被考虑。这可能会引来争论，即这到底是基于语言的，还是它就是架 构。这暂且不管。其次，SOA 已经明确成为一个招牌，许多大厂商已经烙上它的印记（如 SAP）。然而，他们还遗漏了"流程"的交付。因此，可以肯定，在未 来几年，流程和服务的联姻及融合都需要做得更好。最后，技术总在变化。若干年前，象诸如 iPhones、iPods、Smartphones、BookReaders 这样的设备都未曾出现。要是我们想在它们身上构建和交付功能，SOA 也需要变得更轻，成为精简版（SOA-lite）！就像敏捷改 变了瀑布或 OOP 的开发实践，SOA 需要针对新浮现的技术进行改变。

William：SOA 将真正的把数据和业务流程当作服务来关注。在今后几年中，SaaS 供应商将关注自动化和可配置的业务流程，这 些流程 将导致产生随需应变的世界，驱动企业的业务价值链。SOA 还将用来让诸如安全（认证、联邦、身份认证），日志、监测、分析等这样的 IT 服务变得灵活和可持 续。门户将成为面向服务企业的入口点。

Dave: 谈到 SOA 的未来，这让我想起了一个形容词：淹没的。SOA 将会像客户端-服务器曾经的情形一样，无处不在。新的架构方法会引人注目，但更有可能的是，它们都将构建在 SOA 的原则之上。

InfoQ：在 2004 年左右，SOA 治理成了 SOA 一个主要组成部分，您认为它是促进还是阻碍了 SOA？

Jeff：绝对有帮助！但我们要看到任何事物总存在着两面性。首先，治理（取决于它的力度）已经为服务的创建和管理以及生命周期管理 提供了 某种控制级别和结构。要是没有治理，混乱就会到处肆虐，我们只会得到大量"闲置（即无法重用）的"服务。它也让开发者、管理层和其他人明白，监督很重要。 在事物的另一面，它也暗示，由于官僚主义，创新会被扼杀。再次强调，这完全取决于治理水平以及它对组织的意义。我相信，赞同治理有积极影响

的比率会是 65/35。

William：SOA 管理当然阻碍了 SOA。治理不是仅靠安装个工具或是创建一打 Web 服务就能搞定的。你需要管理组织变更、培 训人 员、改变人们的思想。根据公司业务、技术能力和成熟度，差距会很大。然而，值得注意的是，SOA 治理的开源工具已经从纯粹的一个存储库/注册库的单独应 用，演变成了一个面向工作流的平台……治理在未来会变得越来越重要，因为对服务使用、结算、访问和集成的控制需要会变得越来越重要。

Eric：随着企业试图实现 SOA 愿景，他们很快会发现一两的预防要比一斤的治疗好得多。治理正是我们保证工作成熟的手段。SOA 治理的 学习曲线最终会克服人们所说的障碍，“哦，我还没有这么做过，也没有大师可供参考”。因为任何改变，为了支持 SOA 而对公司治理和软件开发生命周期进行修 改都不会那么容易进行——即使公司有老练的 SOA 专家帮助安全地走过这个阶段。这种组织学习曲线对于不怎么能适应变化的公司会伤害 SOA 的成果，但是那些 坚持下来的公司会发现它对 SOA 成果还是有价值的。总的说来，正确地运用 SOA 治理会对 SOA 有帮助，因为它促进了业务的可见性以及项目成果 ROI 的可追 溯性。我曾经一同工作过的很多架构师已经被治理授权 ,真正贴近业务 ,并让开发团队的生活变得更加舒心。

Dave：SOA 治理肯定对 SOA 有帮助。优秀的治理不是 SOA 治理，而是将 SOA 治理包含到现有和可能需要增强的治理进程。

InfoQ：服务治理常被吹捧成让业务/IT 更好对齐的一种方法。您如何看待服务治理中业务的参与？

Jeff：因为刚通过 ITIL 考试，我会对这一概念和服务支持保持警觉，因为从 IT 角度看，它会被滥用和过度利用。然而，至于 SOA 服务治 理，我相信在开发对彼此都有价值的组件时，它为业务和 IT“合二为一”提供了机会。还没有其他情况（好吧，业务流程中可能有）能让 IT 和业务一起合作，而 双方彼此互利。通常，IT 去找业务的目的只是为了理解需求，接着他们就会开始试验，开发解决方案。在服务治理中，他们实际有机会在一个开放的论坛中从对方 的角度进行讨论，抛开“老子天下第一”的念头。这时，业务可以了解一些 IT 的运作，而 IT 则赢得了业务的援助，以及对他们目前所从事的事情的支持。

William：业务不会对服务治理感兴趣，除非你能证明它能给公司带来效益或能加快推向市场的时间。截至到今天，对齐并不存在。业 务的快 速发展和它的需求让 IT 现在就跟它对齐不太可能。当前的 IT 流程和技术还不够成熟。这就是我们为什么要就地改变 IT 范式的原因。业务制定的规范应该被重 用，以业务的形式创建出软件。这是模式驱动的方式。新语言和平台也能进一步的加快这一进程。我们唯一不能加快的变量就是人。

Eric：我在上一个问题中曾经提到，SOA 治理的工作真的能让架构师更接近业务。这种亲密性为业务所有者在决策制订过程中培育出了 一种更 好的控制感 ,通过架构师交付实际所需要的一种更好的能力。治理成功的手段是在业务或架构师分别代表自己提出新想法和新需求时，将两者在一系列快速“自然 的”公司运营步骤中拉到一起，然后将验证/审批作为交付的保证。尽管这听起来很简单并且‘容易’，但是要实现这样的沟通和流程级别，要求公司大量的自我评 价和批判指导。我还没有在市面上看到有多少成熟度提供在该领域下实做的指导，因此 ,业务所有者仍然会因为发现他们所察觉的应该属于信息服务的内部事务而感 到沮丧。

Dave：这种参与至关重要，另一方面，服务和 SOA 仍会继续节省成本和提高效率。当然，在我们跟业务打交道时，还是不要明确点出服务和 SOA，但是它们必须驱动系统提供价值的方式。

InfoQ：业界有很多关于“重用”的争论，您认为重用在行业内取得成功了么？

Jeff：“圣杯”是重用。怎样才能让构建和制造项目更有效和高效呢？当技术人员能够通过 SOA 的手段在此基础上交付的时候，价值是 巨大 的！然而，重用通常不会马上被发现，在部署多年后才能开始看到结果。此外，要让此发生还得要有好的“架构师”和“工程师”。顶级管理层时常期望在 18 月内 就看到结果，不幸的是结果通常不会那么快就自动现身。服务治理能帮点忙，但这需要好的领导，以及从组织（文化）到真正“采用”SOA 的大宗买进。前面说 过，倘若 SOA 和重用只是耍耍嘴皮子，那么它们注定要失败。不管治理是否有帮助，因为它不会在一个“积极的”状态下进行考虑或具有价值提供能力。

William：在讨论重用时，说明其困难程度的最好例子就是 Portlet。在 Portal 中，我们要构建多个 Portlet。 但是大 多数 Portlet 在 Portal 首页中只会使用一次。这里，并未发生重用，但是 Portlet 必须具有可移植性。我想说的是，重用越来越不重要了。真正 重要的是在企业中创建可持续的资产。这些资产是公司的知识产权，具有内在价值。

Eric：由于不止一个，你需要澄清这里讨论的是哪个“重用”。总的来讲——每个企业都关注用尽可能有效的方式来使用他们的资源。然 而，由 于业务和 IT 都不清楚他们拥有的哪些属于 IT 资产，这严重阻碍公司最大化重用的好处，使重用的成功故事成为空白。服务治理已经强调了可见性的需求，并造就 了了解在用 IT 资产、使用人、有意使用者，以及未来使用存在的障碍等需求。在回答可见性问题时，存在有很多风格和技术使用，但是成功故事已经大大增加了， 不仅仅是重用，而且是以一种可理解、可度量的方式将资产度量指标反映到业务上。

Dave：我不确定。我们避免将重用作为一个主要卖点，因为它太难衡量和量化了。

InfoQ：您能否介绍一下 SOA 或者服务治理组织成功的元素？

Jeff：拥抱这一变化的上层领导的大力支持和认可。愿意抓住这次机会进入该领域成为变革推动者的工作人员！开发流程的修改，类似面 向对象 开发 VS 瀑布模型：服务设计上的前期工作对于有意义并有机会在将来重用的服务的界定和构架至关重要。随着时间日渐"成熟"的治理模型；设置一个它要达到的认识水 平，完成之后再改变，然后再来执行。不要一开始就搞专制，否则你不会有任何追随者。

William：我们需要：

- 关于现有服务的一个清晰服务分类，并将其在注册库中实现，以始终提供准确的服务目录；
- 服务的任何配置都通过保存在具体存储库中的元数据来完成；
- 管理服务全生命周期（设计时、运行时、终结）和发布；
- 每个服务都有针对其消费者的 SLA 契约，并有工具可以对其进行度量；
- 公司内有一个集成能力中心来管理 SOA，并有相关的专门预算。

Eric：我在公司中看到的成功元素首先是拥有灵活思维和能够适应流程变化的人。他们大多数都拥有技术 ,这让他 们能够自动治理流程并能得到 IT 资产的联合视图——不 仅限于 SOA 资产，而是整个企业的视图——人、流程和技术/服务。最大的成功因素——具备强大业务本领以及与最高层及高级经理有政治关系的拥护者。

Dave：我会建议将它包含到其他 IT 和开发治理中，并且建立一套统一的标准来进行考量。

InfoQ：要是您能改变 SOA 中一个事物，您会选择哪一个？

Jeff：标准.....我个人没有太密切关注过这个领域，但是我能确认我们需要将采用过程变得简单。此外，这跟其他领域没什么不同，我们 需要建 立一个路线图，明确指出我们要前进的方向。不幸的是，我认为 SOA 标准是"老鼠洞"。对不起...如果我参与了委员会，可能我不会这么想——不幸的是，我不 是，因此我还是会这么看。

技术也是一个它可以发挥作用的领域。没人会喜欢在微软和 Java 之间来一场战争的想法。因而，折磨世界的不是观点统一而是分歧。尽管我知道有好几种方法能让它们一起工作，但没有必要将世界统一成一个 SOA。

William：我会说服大家采用契约优先的方法并定义描述服务 SLA 的唯一标准。我很想有一个 Web 服务管理平台的开源实现。

Eri：我们能否更早地把它创造出来？;)说实话，要是能改变 SOA 中的一个事物，我会选择一开始就击中靶心。企业承受的任何改变都 会导致 公司的成本。尽管 SOA 现在 10 岁了，对于大多数信息服务和企业而言，他们仍然将 SOA 看成开销而不是节约。MomentumSI 已经反复给客户证明了这 一点。

Dave：我会不惜代价阻止市场炒作和夸大其词对 SOA 品牌的消弱。

你已经看到的，即便 10 年过后，SOA 的某些方法和预期收益方面还是很容易产生不同观点。你对这些问题的答案又是什么呢？

原文链接：

http://www.infoq.com/cn/articles/soa-panel

相关内容：

- 回顾之回顾
- Open Group发布新SOA治理框架与服务集成成熟度模型
- Open Group的SOA资料卷
- SOA：我们从这里走向何处？
- 实用的SOA治理

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

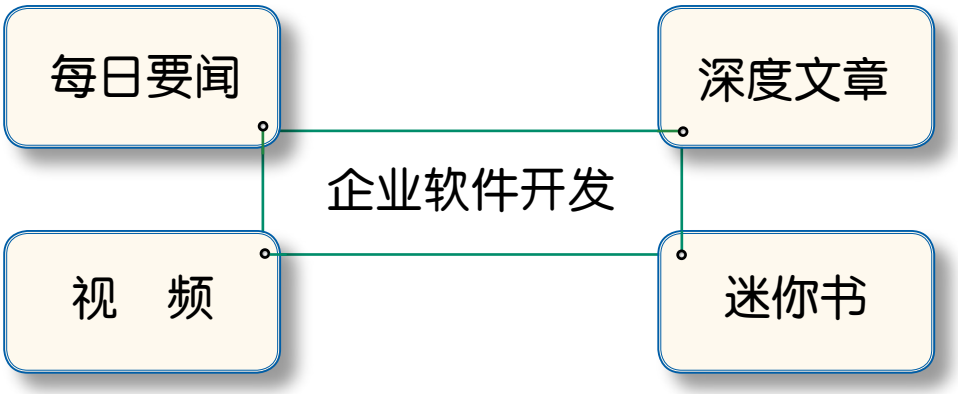
我们的**定位**：关注高级决策人员和大中型企业

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

热点新闻 | News

现在的技术趋势

作者 Abel Avram 译者 张凯峰

在这个月，ThoughtWorks公司发布了2010 年的技术前瞻（PDF），这份白皮书包含了ThoughtWorks在四个主要领域中的技术策略和趋势：技术、工 具、语言和平台。InfoQ仔细考量了这份白皮书，以更好地理解ThoughtWorks主张的思想和建议。

针对每一个领域，ThoughtWorks 提出了一个技术列表，并分为四类：采用、试用、评估和保留。采用意味着在企业中推荐使用这 个技术。试用意思是这个技术值得投资，但应该只用于低风险项目中。评估是指技术值得考虑和学习如何使用，以及它们的潜力所在。保留则意味着目前不值得花费 精力和资源在某些技术上。

Techniques	Tools	Languages	Platforms
Build pipelines	IE6 end of life	JavaScript as a first class language	JVM as platform
User centered design	ASP.NET MVC	C# 4.0	Firefox
Evolutionary database	Visualization & metrics	DSL's	Cloud
Web as platform	Subversion	Java language end of life	iPhone
Emergent design	Distributed version control	Functional languages	Android
Lean software development	Next-generation test tools	Concurrent languages	Non-relational databases
Continuous deployment	Google Wave		HTML 5
Incremental data warehousing	Language workbenches		Rich Internet applications
Evolutionary architecture	Polyglot development environments		RDF & SPARQL
			Google as corporate platform
			Location based services
			Chrome OS
			Chrome
			IE8

Legend:	Adopt	Trial	Assess	Hold

技术

跟一年前相比，所有推荐的技术都提升了一个或多个等级。“构建管道”从试用变成了采用，“精益软件开发”从评估变成了试用，而“自然 设计”和“演进式数据库”则从默默无闻变成试

用。

工具

至于工具方面，相比较一年前，变化显得更加多样。IE6 在去年没有被考虑，而现在则建议停止在产品中使用。ASP.NET MVC 从试用变成了采用。Subversion 已经从采用变回了试用，因为“分布式版本控制”工具从保留变成了试用。白皮书还提到，作为“下一代测试”工 具的 rspec 和 Cucumber 把这个分类推进到了评估。其他之前没有考虑过，但现在在保留类别中初次登场的新工具有：Google Wave、语言工作台、和多语言开发环境。

语言

也许来自这个领域的建议最引人入胜。JavaScript 从不被看好进入采用区域，作者是这么解释的：

> 虽然 JavaScript 最初出现在 1995 年，但直到过去几年，由于 Prototype 和 JQuery 这样的工具库的帮助，这门语 言才开始获得更加广泛的开发群体。在开发者们持续拥抱 JavaScript，开发出具备优秀用户体验的应用的同时，我们也在不断地给予 JavaScript 跟其他生产语言同样的尊重，确保脚本可以得到足够的测试、重构和维护。

C#则从保留转移到了评估和试用两者之间，这缘自去年它所取得的进步：

> C#随着语言特性的增强继续发展，这些特性包括 lambda 表达式、扩展方法、对象初始化器和自动属性 getter 和 setter， 所有这些特性都出现在语言的 3.5 版本中。而在 C#4.0 中，我们还会看到动态关键字、命名和可选参数等特性，这使得 C#能继续跟 Ruby 这样的语言保持 一致，同时越于 Java。

作者提到了最近 Java 在增加新特性方面进展缓慢 ,而基于 JVM 的其他语言更具开创精神 ,“比如像 Groovy、JRuby、 Scala 和 Clojure 这样的语言”。他们期望企业“开始评估在企业应用中减少特定于 Java 的代码数量之可行性，以支持这些更新的语言。”由此，作 者们建议评估 Java 作为编程语言的结局。

平台

最后，平台方面没有什么大的惊喜，除了一点：“富 Internet 应用”从采用降级到评估和试用两者之间。原因在于：

> 我们关于 RIA 的定位在过去的一年发生了变化。经验表明 ,像 Silverlight、Flex 和 JavaFX 这样的平台对数据的丰富 可视化很有帮助，但对简化 web 应用并未带来好处。
>
> 考虑到这些工具集对自动化测试只有有限的支持，所 以建议企业级开发采用更加传统的 web 应用开发栈来提供更大的价值。我们推荐只针对丰富可视化而采用 RIA 平台，以合并到 web 应用中，而不是作为一个复 杂的开发目标。

Firefox 得到采用，因为它提供了“针对 widget 范围的平台级支持”。JVM“作为 Ruby、Groovy、Scale 和 Clojure 等语言的通用虚拟机”取得了同样的成绩。另外一个平台则是 Android，建议试用。iPhone 现在是个热门的技术，建议采用。

原文链接：http://www.infoq.com/cn/news/2010/01/ThoughtWorks-Technology-Radar

相关内容：

- 做一个技术布道者
- 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？
- 适应性——新软件危机的主因

热点新闻 | News

2010 年SOA发展趋势

作者 晁晓娟

过去一年可以说是 SOA 的大战，各种各样的利益组织，云，业务过程管理，企业 2.0，还有服务商们都在试图证明 SOA 的影响力和商业价值。那么接下来的 2010 年，SOA 领域会是什么样呢？

Peter Schooff 提出了一个很好的问题你认为 2010 年SOA的最大趋势或发展是什么，跟贴者甚众。

悲观者如Dave Youkers认为：

> 由于太多的混乱和失误使得 SOA 将不太适合作为具体的方向来发展，SOA 的理论将会被 BPM 社区取代。

而更多对SOA寄予希望的人们则提出了自己对SOA的愿望清单，Kelly Emo和JP Morgenthal说出了他们最期望的一些方面。如：

> ……SOA 最终被认为是一种专注于服务发布的策略工作而不是应用开发。
> 应用程序和运营停止对“服务”一词的争论……服务开发和服务发布的差距能够变小……
> ……质量被认为是基于 SOA 的方案成功的首要工作……IT 组织对 SOA 测试技能和质量管理过程有实际的投入……
> ……需求管理，业务分析，架构和开发团队能够在一起理解商业需求如何映射到服务和复合应用程序的功能性，非功能性需求……
> 最重要的是，IT 和商业更好的提高信任关系并定期在一起工作而确保基于 SOA 的解决方案能够被计划和发布……

SOA分析师Joe McKendrick对其2009 年的帖子进行了总结，并提出了2010 年SOA的十大趋势。

他总结的 2009 年热门帖子涵盖了影响企业信息数据管理的SOX法案、2009 年初的'SOA'已死、SOA的一些经典例子、BPM和SOA是互相需要的、云计算持有兴趣的增长、人们误解SOA的 5 个原因、10 个数据中心的矛盾等等。

而他对 2010 年的预测如下：

- 面向服务的原理将会复苏 ，因为有效的治理和云服务的交付都会需要它们，即使不一定被叫做 SOA。
- 随着经济的复苏，将会涌现出大量的创业项目，无论是个人还是已经建立的团队，很多都会依赖云服务起来展开他们的运营。而面向服务的原则是其基础。
- 随着企业摆脱经济恐慌，并寻求更具侵略性的成长，它们对分析仪表盘的依赖会大大加强，对复杂事件处理（CEP）的兴趣将持续增长，而这需要事件驱动架构(EDA)来支撑。
- 通过 SOA 来改善质量，时间表和数据流的可信赖性将得到更多的关注。数据服务会变成 SOA 工作中越来越重要的组件。
- 厂商将会更少地讨论"SOA"，而更多地提及"云"。不论对不对，实际上在他们看来，这两个词是可以替换的。
- 云治理将会在组织中越来越重要，因为他们发现得到的是一团杂乱的点对点云服务，而没有一个统一可度量的中心，这预计会大大促进 SOA 治理工具和云服务注册库的应用。
- 更多的 SOA 功能和产品都将会在云上发布。如按"集成即服务"等方式。
- 微软，借助 Oslo、 WCF、 BizTalk 和.NET Framework，将会进一步促进中小型企业的面向服务进程——大部分也会是基于云来发布的。
- SOAP 和 WS-*（这些年一直是 Web Service 的核心）会被轻量级协议（如 REST）挤到一边，更多地 SOA 工作将会结合 SOAP 和基于 REST 的服务来做。
- 企业混搭将会是最常见的用户访问后端服务的形式，这也许能帮助商业用户更好的理解 SOA 到底是什么。

ZapThink网站也预测了2010 年SOA的几大趋势：

- 开源 SOA 架构将会成为主流，由于缺乏风险投资的兴趣和五大 IT 厂商统一地支持而导致可选择的 SOA 基础框架很少，用户将会更偏向选择开源的方案。
- RIA 市场战争会结束或者说已经结束了，Adobe Flash 和基于 Javascript 的开源 Ajax 解决

方案占了上风。

- 云的私有和安全性的讨论会慢慢平息。尽管我们看到了很多关于不安全，不可靠，缺乏私密性的讨论，但这里有着巨大的客户群和高额的收入，因此云服务商不会放弃巨大的市场，这样必然会有相应的对策来解决这些问题。

相信每一个相关领域的人对这个问题都有自己的看法，无论从哪个角度来看，接下来云和 SOA 将会越来越紧密地在一起，我们期待未来 SOA 能够更深入到实践和产品中来。

原文链接：http://infoq.com/cn/news/2010/01/2010-SOA-TREND

相关内容：

- REST作为风格 -- WOA作为架构
- 调查表明SOA势头正劲
- 选择哪个SOA测试工具？
- 微软发布.Net RIA Services
- 微软发布BizTalk Server 2009

热点新闻 | News

Flex RIA生态圈现状分析

作者 Moxie Zhang 译者 张龙

2004 年 3 月，Macromedia（2005 年被 Adobe 收购）发布了 Flex 1.0。从那时起，基于 Flex 的 RIA 开发获得了越来越多的动力，RIA 也已经成为广泛接受的 Web 应用开发方式。今年，Adobe 将发布 Flex 4，随之而来的是 Flash Builder 4 以及 Flash Catalyst，他们都将成为 Adobe Flash Platform 技术的组成部分。最近 InfoQ 回顾了当前的 Flex RIA 生态圈以明晰 Adobe Flex 的现状。

Flex 开发环境

Adobe Flex Builder仍然是使用最为广泛的商业Flex IDE。它构建在开源的Eclipse IDE平台之上。在Flex 4 发布后，Flex Builder即将更名为Flash Builder。除此以外，Adobe Flash Catalyst目前还处在beta版，这是一款设计工具，旨在通过集成设计与编程以将Flash设计人员与Flex开发人员联系起来。

除了 Adobe 的工具外，Flex 开发也已经深入到了现有的各种 IDE 中。

- IntelliJ IDEA ,这是一款大获成功的Java IDE ,它已经从v7 开始支持Flex开发了。最新的IntelliJ IDEA 9 提供了更加全面的特性以支持Flex。
- Amethyst是一款构建于Microsoft Visual Studio之上的Flex IDE。它向微软平台的开发者们提供了熟悉的环境进行Flex开发。
- Ensemble Tofino for Visual Studio同样是一款面向Windows开发者的Flex开发工具。
- FlashDevelop是一个开源的ActionScript 2/3 及Web开发环境。它集成了Adobe Flash IDE、Adobe Flex SDK、MTASC（一个开源的Flash ）、haXe（一门开源的编程语言，可以被编译成swf）以及swfmill（一个支持XML到swf双向转换的处理器 ）。

应用框架

开发软件框架的目的在于实现常见的软件开发模式以提高编程生产率及改善质量。InfoQ注意到 2008 年推出的一些Flex/ActionScript框架对于Flex使用率的提升功不可没。他们是Cairngorm、PureMVC、Model-Glue:Flex、Foundry、Guasax Flex Framework、ARP、Flest Framework、EasyMVC以及Adobe FAST。从那以后涌现出了越来越多的框架，这些框架丰富了Flex开发生态圈：

- Ruboss这个Flex框架集成了Ruby on Rails和Merb。它还有一个RESTful接口以与Adobe AIR的嵌入式SQLite数据库进行通信。Ruboss框架与Rails和Merb应用的关系就好像是Adobe LiveCycle Data Services ES与J2EE应用的关系一样。
- Mate Flex框架发布于 2008 年，其目的是简化事件驱动的Flex应用开发。
- Swiz是个面向Flex的IoC框架。它并没有太多的强制要求，比如目录结构或是样板代码等，这一点与其他框架如JEE大不相同。
- Prana是又一个面向ActionScript的IoC框架。它基于Spring框架的XML方式进行开发。
- JumpShip是个ActionScript MVC框架，包含了标准的数据模型以进行自动化的数据分类、枚举以及搜索。它反对在框架中使用单例模式，而单例模式在现代的软件框架中得到了广泛的应用。
- GAIA是个面向Adobe Flash的前端ActionScript框架，支持Flex Builder。
- Razor是个ActionScript组件框架，对常用的Flex组件提供了另一种选择。
- Flight Framework是又一个ActionScript框架，支持MVC及其他设计模式。

Flex 与 AIR 开发工具支持

如果没有调试、测试、日志以及文档，软件开发怎能进行下去。在过去几年中，Flex/ActionScript社区创建了大量的开发支持工具。

- RIATest是个面向Flex的GUI自动化测试工具。它支持Windows以及Max OS X。
- Flexcover是个面向Flex、 AIR以及ActionScript 3 的开源代码覆盖率检测工具。
- Alcon是个轻量级的调试工具，支持ActionScript3、 Flex及AIR开发。
- Fluint（Flex unit and integration的简称）是个面向Flex 2/3 应用的测试框架，无论应用是通过Adobe Flash Player部署在Web浏览器中还是通过Adobe AIR部署在桌面上。
- Arthropod是个面向Flex和AIR开发的调试工具。凭借Arthropod，开发者可以在运行期轻

松调试应用。

- De MonsterDebugger是个面向Adobe Flash、Flex及AIR项目的开源、轻量级，但功能完善的调试器。它完全使用Adobe AIR开发。
- ASTUce是个衰退测试框架，其灵感来源于xUnit架构，如JUnit。它支持对ActionScript 3 的单元测试。
- AsUnit是个面向ActionScript 3 的开源的单元测试框架。AsUnit 2.x已经完全集成了Flash IDE。
- FlexMonkey是个面向Flex应用的测试框架，它可以对Flex UI功能进行捕获、重放以及确认。FlexMonkey可以记录并回放Flex UI的交互并生成ActionScript测试脚本，这些脚本可以轻松集成到持续集成过程中。
- Xray是个Flash应用检测工具，用于在运行期调试应用而不会增加应用负载。
- FlexPMD是由Adobe创建的，旨在通过审查AS3/Flex源代码目录来改进代码质量和检测常见的最差实践。
- Natural Docs是个文档生成工具，支持多种语言，包括ActionScript 3。

Flex 企业级开发

Adobe 在企业应用系统开发上投入了大量的人力物力。大多数企业系统都需要服务端开发和集成，Adobe 的开源产品 BlazeDS 及商业产品 Livecycle DS 在这其中扮演着重要的角色。此外，Flex/ActionScript 社区也开发出了各种服务端集成工具以支持 Flex 企业级 RIA 开发。

- Potomac framework for Flex用来开发大规模的Flex应用，它利用了模块化方法而没有使用Flex模块框架。其灵感来源于OSGi，后者则被众多的应用服务器厂商使用以支持服务端的模块化功能。
- FluorineFx提供了一个Flex/Flash Remoting、Flex Data Service以及实时消息功能的.NET框架实现。
- FxStruts则是一个开源的程序库，提供了与Struts中的bean:write相同的功能，但其输出格式为AMF或是XML。
- X2O是面向Adobe Flex应用的基于Web的数据建模平台。它会生成一个远程托管框架，这样开发者只需编写客户端即可。

- Spring BlazeDS integration是个Spring组件，用于简化以Adobe Flex作为前端的基于Spring的RIA开发。
- Spring ActionScript以前叫做Prana framework（上面提到过）。
- Granite Data Services是个免费（基于LGPL）的组件，作用与Adobe LiveCycle Data Services一样。
- Red5是个开源的Flash服务器，使用Java编写。
- AmFast是个面向Python的Flash Remoting框架，支持NetConnection与RemoteObject RPC。
- Exadel Flamingo可以将Flex、JavaFX、Swing、J2ME以及Android SDK粘合到Seam、Spring及JEE中。
- 还有面向各种脚本语言的Flash Remoting支持：面向PHP5 的PHPObject和SabreAMF、面向JEE的OpenAMF、面向Perl的AMF::Perl以及面向Python的AmFast。

展望未来，Flex RIA 开发的下一领域将是移动平台。一系列事实表明即将发布的 Adobe Flash 10.1 将能够运行在大量的智能设备上，比如将要发布的 Google Nexus One phone 将安会安装 Flash 10.1。一旦 Flash 移动技术横空出世，Flex 社区将会大举进军移动平台。

InfoQ 将会持续关注并报道 Flex RIA 领域的最新进展。

原文链接：http://infoq.com/cn/news/2010/01/state-of-flex-dev-tools

相关内容：

- David Wadhwani访谈：开放屏幕计划与富媒体平台
- Spring BlazeDS Integration简介与入门
- FlexMonkey将单元测试引入Flex用户界面开发
- Flex 4 的新体系
- 深入理解Flash Player的安全沙箱

热点新闻 | News

站点恢复过程中的经验和教训

作者 Abel Avram 译者 侯伯薇

最近Jeff Atwood丢失了两个blog站点：Blog @ Stackoverflow 和 Coding Horror。他设法恢复了这两个站点的内容，但是从这个事件中我们应该得到什么教训呢？

在 Jeff 的博客上，他写了一篇为什么人们需要备份内容以及如何进行该项工作的文章，2008年一月发布的，在其中他做出了以下结论：

> 可以肯定的是：只有在你拥有某种备份策略，并且坚持执行的时候，你才会了解它。如果备份数据看起来有点麻烦，那就对了，因为它的确是那样。**打住！我知道怎么回事儿。听我的。不管怎样都要做。**

Jeff针对Stackoverflow以及Codeing Horror的博客都有备份策略，其中后者是由CrystalTech管理的，他们会经常备份整个站点。但是又怎么可能会丢失站点的内容呢？原因是由于站点是存放在虚拟机中的，而CrystalTech备份的是虚拟机的镜像， 而镜像全都被破坏了。这样，虽然备份包含了大量被破坏了的虚拟机镜像，但是它们没什么用处，因为使用它们无法启动虚拟机。在文章的开始，Jeff将责任归咎于他自己以及提供商：

> Jeff Atwood：呃，CrystalTech的服务器崩溃了。很显然他们通常所用的备份流程在备份虚拟机镜像的时候失败了，而没有给出提示。
>
> Jeff Atwood：对此，我和站点提供商各负 50%的责任（不要相信站点提供商，还是要做你自己的离线备份！）

在一些用户的帮助下，Jeff设法恢复了他的站点中丢失的内容。Rich Skrenta帮助他找回了文本：

> 幸亏有blekko.com的Rich Skrenta，这样我才能几乎立即找到Coding Horror站点上的静态的HTML版本。他热心地提供了该站点上每个爬虫页面的结果。有些人有目标，而有些人有大胆的的目标。Rich的目标尤其令人惊叹：在Google最擅长的搜索领域与其

> 进行较量。这是为什么他能够恰好找到Coding Horror完全的文本存档的原因。Rich，我是否曾经告诉你，你是我的英雄？不管怎样，幸亏有了Rich，你现在还能够浏览Coding Horror的静态HTML版本。令人惊奇的是，对于这个站点，静态的HTML版本和动态的版本没有很大的不同。我想这是作为极简主义者的好处吧。

而图片都来自于 Carmine Paolino，恰好一个用户拥有“这个站点几乎完全的镜像，这些数据都存放在他的 Mac 上！多亏他的镜像，我现在才能够几乎 100%地恢复丢失的图片和内容。”在恢复了站点之后，Jeff 获得了重生，同时也停止了自责：

> 因为我是个笨蛋，我没有对Coding Horror做自己的（最新的）备份。天哪，我多希望曾经读过一些好的关于备份策略的博文啊！

他做出如下结论：

> 从这些悲惨的事件中，我们能够得到什么教训呢？
>
> 1. 我吸取教训。
> 2. 是的，真的，我吸取教训。
> 3. 不要依赖于你的提供商或者任何其他人来备份你重要的数据。而应该自己来做。如果你不能够对你自己的备份负责，那么数据丢失就不会是偶然事件了。
> 4. 如 果你的数据真的出了问题，那么你怎么才能够恢复呢？过程是什么呢？恢复最困难的部分是什么呢？我想在我思想深处对Coding Horror的灾难恢复能力有错误的信心，因为我一直认为它大多数都是文本。当然，后来发现文本是其中最简单的部分。而我曾经认为是很好方式的镜像比我意 识到的更加重要，而且更加难以恢复。有些人认为我们不应该讨论如何备份，而应该讨论如何恢复。
> 5. 定期对你的恢复过程进行检查，以确认它仍然是可用的、有效的并且功能齐备，这是非常有价值的工作。
> 6. 我太伟大了！不，只是开玩笑。我还是要吸取教训。

Joel Spolsky展示了多种其他可能的情况，其中如果没有提供恢复的话，那么备份策略就不起作用：

- 我们使用加密安全密钥对备份文件进行了加密，而密钥位于丢失数据的那台计算机上。
- 服务器有大量存储在 IIS 的元数据库中的配置信息，而它们没有备份。

- 备份文件被复制到 FAT 分区上，并且被截断为 2GB，而没有给出提示
- 你的备份位于 LTO 驱动器上，它和数据中心一起丢失了，并且你无法在三天之内得到另一个 LTO 驱动器。
- 还有其它成千上万种可能出现的错误，即便你拥有备份策略。

可靠服务的最小限度并非是你已经做了备份，而是你已经为恢复做了准备。如果你正在运行一个 Web 服务，那么你需要明白的是，你能对整个站点创建合理 的最新副本，并且是在合理的时间之内，在一个新的服务器上或者过去没有访问过原来的数据中心上任何东西的服务器。底线是你已经做了对恢复的准备。

让我们不再询问人们他们是否做了备份，而要询问他们是否为恢复做了准备。

Jeff转移了他的博客站点，以便于存放Stackoverflow的数据中心和其他在家中的服务器更加可靠，因为它们带有更好的备份策略：

1. 我们在早上四点、下午四点和晚上 12 点会对所有数据库做全备份。（某些数据库可能会更频繁，但这是通常的方式）这些数据库的全备份都存储在PEAK数据中心机架上的NAS RAID-6 设备中。
2. 我 们有一个连接在数据库服务器上的 500GB 的 USB 硬盘。还有一段 C#的脚本，它会在每晚的凌晨一点左右将最新的备份从 NAS 复制到 USB 硬盘上。最旧的文件会被删除，从而为新的文件腾出所需要的空间。（当前的 Stack Overflow 的全备份压缩之后有 7GB 左右，而另一个数据库压缩后大概有 2GB）。新措施是：我们会在服务器上连接两个 USB 硬盘，从而并行地做验证 复制，以免其中之一出现问题。
3. Geoff Dalgas是我们团队的一员，他住在离PEAK数据中心一英里的地方。他每过几周就会顺便到数据中心更换USB硬盘。他在家里放有四块 500 GB的硬盘，还有两个在数据中心。他会不时地持续循环更换。
4. 新措施：Fog Creek 每周都会使用 FTP 接入，然后将最新的数据库备份到他们的伺服设备上，在星期六流量低的时候进行。
5. 我们每个月都会为所有站点（Stack Overflow、Server Fault、Super User）创建共享的数据块（dump）。这是数据的子集，但是是可以控制大小的，并且提供在Legal Torrents上。这些数据块被物理存储，并且由Legal Torrents提供种子。
6. 我们的 Subversion 源代码控制库每天都会被复制到 NAS 上，并且被复制到外部的 USB

硬盘上等等，这都是通过相同的脚本完成的。

7. 我们还运行了几个虚拟机镜像——大多数是为了提供 Linux 辅助服务——他们也是通过同样的过程备份的。因为我们的其他提供商吸取了困难的方式的教训，所以像虚拟机那样备份更有技巧性，而这肯定是你需要小心的事情。

8. 我们有规律地下载最新的数据库备份并在本地恢复它们（我们总是针对真实数据进行开发），从而我们知道我们的备份是有效的。

这个策略听起来比在开始的时候设置更好一些。在这种情况下薄弱的环节在于“Geoff”。如果 Geoff 没有出现并更换驱动器怎么办？或者他丢掉了一块怎么办？或者小偷从他家里把硬盘偷走了怎么办？

Jeff Atwood 不是真的要责怪谁。这对任何人都有可能发生，即便拥有更好的备份策略。问题在于：我们在此能够得到什么样的教训？

原文链接：http://infoq.com/cn/news/2010/01/Backup-Strategy-Restore

相关内容：

- 案例分析：移植大型VB6 应用程序到.NET
- 引领自组织团队就像指挥交响乐吗？
- 诺贝尔奖官网——纯Adobe应用
- 郭晓谈实效敏捷和敏捷在中国的发展
- 敏捷应对“团队的五重机能障碍”

热点新闻 | News

对SOA实施者的实践忠告

作者 Boris Lublinsky 译者 黄璜

在"对目前从事SOA的组织的几点建议"这篇博客中，Ganesh Prasad基于他多年的经验，对于如何开展一个大型的SOA项目作出了建议：

> ...不要因为这一方式不是那么宏大而感到担心。复杂性让新手感受深刻，但结果才是最终能让所有人留下印象。

Ganesh 的方法包括了以下几个要点：

- **眼观全局。**SOA 不是关于集成或者是引入一种新技术来简化现有系统的连接。而是关于：

> ...精简企业的部件并且使它们易于理解和连接...所以始终应当谨记简洁性，并且不要把它与权宜之计搞混了，那是指的阻力最少的道路和。而精简可能需要付出努力。

- **理解数据。**服务互操作性需要用于交互的“语义”数据模型。Ganesh 指出所谓规范的数据模型通常层次较高且对于实践应用来说过于抽象。作为代替，他建议将企业数据划分为几个逻辑域并为各个域定义字典。

> ...所有暴露它们的逻辑域的服务都应该当使用这些定义，而来自其它域的服务消费者有责任理解这些定义。由跨这些域的服务组合起来的流程应当在类似的数据元素之间 执行它们自己的映射。这不象听起来这么恐怖，因为只有一个域所管理的数据元素只有一部分子集会通过服务接口暴露出去...不要尝试[构建]一个单一的规范 数据模型。那只是徒劳之举，根本不要启动。

- **选择正确的中间件。**在 Ganesh 看来，大部分情况下，HTTP 是 SOA 实现最合适的中间件。他建议，除了必须需要的情况下，避免使用消息队列并指出基于 HTTP 的数据库备份的通信模型通常能提供更简单的解决方案。

> HTTP 是一个十分通用的协议，可用作你的 SOA 项目的逻辑基础设施的元素。ESB，服务目录以及其它的“治理”组件通常只在管理它们自身所引入的复杂性时才需要。用简单的 web 服务器群和数据库群所能做到的会让你惊叹，同时还能始终保持简单和明了。

- **选择正确的服务实现手段。**Ganesh 认为基于 SOAP 的 web 服务很大程度上是“供应商提供”的宣传，并推荐尽量予以避免。他建议使用 REST 来代替：

> REST 实际上是实现 SOA 的有效方式，它通常可以以极低的成本和复杂性来交付解决方案。采用 REST 的困难所在是找到用这种方式思考的优秀人员。

- **选择正确的数据合约定义。** 谈到领域模型的正式定义时，Ganesh建议道“标准”的XML方式是重量级的比较笨拙。相反，他建议好好看一下JSON模式提案。

> 在许多高级语言比如 Java 当中，已经有现成的 JSON 模式的库可用。应该能够可以以极低的复杂性 ,如 XML 一样严格的定义数据合约...避免 XML 的那些繁文缛节 ,由 JSON 开始，并且融合日趋成熟的 JSON 模式。你会发现这些与 REST 结合起来会工作得非常棒。

- **解决 SOA 简洁性的悖论。** 尽管 SOA 背后的主要驱动因素是精简企业架构，按照 Ganesh 的说法，典型的 SOA 实现的现实是，因使用重量级方案而导致集成了复杂性，又通过引入工具来管理这一复杂性。

> 当然，如果你有官僚的倾向，你可以沐浴在高预算与大团队的声望中，并且可以基于你所交付的服务和流程和数量发表胜利的宣言。但如果你真的想成功交付 SOA(例如，让你的业务更加灵活并且以一种可持续的低成本来运营)，而这一路上不用烧钱的话，你得务必看看我上面列的这些烦人的，没什么印象的，甚至是 不合潮流的方案和技术。让那些大卖主好好歇歇吧。你不需要买技术(除了你所拥有的 web 服务器和数据库)。你也当然不需要买任何复杂的技术 ,而这正是那些 供应商要倾售给你的。

Prasad 的文章讨论了一个典型的 SOA 实现会遇到的许多问题。它同样通过新的途径，摒除了现有的经常使用的解决方案遇到的问题。这引出一个话题：什么时候更适宜去理解和改进一个现有解决方案要 ,而什么时候又适宜摒弃现有方案而尝试新的途径呢？新事物是否总是最好？

原文链接：http://infoq.com/cn/news/2010/01/Implementers

相关内容：

- 网络爬虫服务 80legs介绍
- Web服务契约的版本控制
- SOA与服务识别
- SOA在互联网系统中的应用
- Jeff Barr谈论Amazon Web服务

热点新闻 | News

NHibernate和Entity Framework 4.0 优劣势争论

作者 Abel Avram 译者 侯伯薇

最近，Oren Eini（也被称为 Ayende Rahein）发表了一个帖子，从而引发了关于 NHibernate 和 Entity Framework 4.0 各自优点和功能的讨论，而这二者都是基于.NET 的对象/关系映射框架。InfoQ 对此讨论进行了深入的探究，以了解其中提到的观点。

Rahien是NHibernate项目的成员之一，他对NHibernate和Entity Framework 4（EF）做了简要的比较。在称赞EF 4 相比EF1.0 所作出的进步之后，Rahien列举了他认为使得NHibernate成为更好的ORM解决方案的特性：

- **批量写入**——我们可以配置 NHibernate，使其对数据库进行批量写入，从而在你需要向数据库中写入多个指令的时候，NHibernate 只需要与其进行一次交互，而不需要在每个指令的执行过程中都要访问数据库。
- **批量读/多重查询特性**——NHibernate 使你可以在与数据库的一次交互过程中批量执行多个查询，而不需要在独立的交互过程中执行每个查询。
- **批量的集合加载**——当你延迟加载集合的时候，NHibernate 能够找到其它相同类型而没有载入的集合，然后只对数据库进行一次访问，就把它们全部载入。这种方法很好，因为这样就可以避免处理 SELECT N+1 的问题。
- **带有 lazy="extra"的集合**——额外的延迟意味着 NHibernate 会适应你可能在集合之上所要执行的操作。这也意味着 blog.Posts.Count 不会强行载入整个集合，而是创建“select count(*) from Posts where BlogId = 1”的指令，然后 blog.Posts.Contains() 会类似地执行单独的查询，而不需要付出将整个集合都载入到内存中的代价。
- **集合过滤器和分页集合**——这让你能够在实体集合上定义附加的过滤器（包括分页！），这意味着你可以很容易地对 blog.Posts 集合进行分页浏览，而不需要将所有的内容都载入到内存中。

- **二级缓存**——管理缓存很复杂，之前我曾经谈过这为什么很重要，所以现在我将跳过它。
- **调整**——当你需要某些框架没有提供的功能的时候，这就显得很重要了。使用NHibernate，几乎在所有的情况下，你都有扩展点，但如果使用的是EF，你是完全并且绝对做不到的。
- **集成和扩展性**——NHibernate有大量扩展项目，像NHibernate Search、NHibernate Validator，NHibernate Shards等等。而在EF中不仅不存在这样的项目，而且大多数情况下也无法编写这样的项目，因为EF没有任何可以使用的扩展点。

Rahien也提到了使用EF 4的优势：

- EF 4.0比当前的NHibernate实现拥有更好的Linq提供程序。这也正是NHibernate正在积极改进的地方，NH 3.0将会弥补这个问题。
- EF属于微软。

作为NHibernate项目知名的贡献者，Rahien的帖子引发了相当数量的正反两方面的响应。一位名叫tobi的读者对NHibernate错误消息的缺少提出了抱怨：

> 我只使用过NHibernate几个小时，对于我来说，手动创建域的类和映射（我使用了FluentNHibernate ）的过程需要太多手动的工作，并且错误信息不是很好。这是我认为相比而言EF 4比较好的地方。

Roy对于错误信息和文档有着矛盾的心情：

> EF的一个额外的优势在于文档组织得更好，并且错误信息能够更清楚地描述问题。
>
> 尽管如此我还是更喜欢NH，但是一旦你遇到问题，那么就需要浏览大量的博客来解决。相反，它的优势在于有很多人你可以请教。

Jimmy Bogard赞赏NHibernate的缺陷修正过程，这使得它更有吸引力：

> NH的另一个主要优势在于它是开源软件。这些年来我多次需要给NH打补丁，以修正缺陷或者添加我所需要的功能。如果使用的是EF，我是不能做这些的。

Alex Yakunin参与了另一个ORM工具ORMBattle.NET测试套件的创建工作，他抱怨说：

> 我想你可以很清楚地发现，在这里只显示了 NHibernate 的优点。而根本没有涉及到它的缺点——即便是你提到的关于 LINQ 提供程序的说法也和事实相 去甚远；另一个众所周知的问题是 EF 支持变更跟踪，而 NH 不支持，这在很多情况下会很大程度上影响性能（事实上，你应该完全忘记 NH 中的特定情况 ——那是"有意地"）。

Radenko Zec对单元测试和设计器的功能进行了比较：

> 我想 NHibernate 最大的优势在于它能够更好地支持单元测试。EF 4 并非为测试而设计，因此很难基于 EF 4 为某些自定义的解决方案编写单元测试。
>
> 另一方面，EF 4 拥有很好的设计器（对于真实世界中的大型项目，这是你所需要的最重要的东西），还有基于该设计器的 POCO T4 模板。我想现在是你应该开始考虑为 NHibernate 建立自己的设计器，而不是拒绝设计器和代码生成器的时候了。如果社区需要 NHibernate 的设计器，那么就给他们好的设计器。第三方的设计器和 EF4 的设计器相差甚远，可能除了 LLBGEN 3 还好一些，但是它还没有发布，而且不是免费的。

Frans Bouma是另一个ORM工具LLBLGenPro的作者，当说到文档时他指出NHibernate在该方面非常欠缺。

> EF 比 NH 好的地方就在于文档、一致的示例以及在每次开发者大会上发表的大量的传播演讲，还有日夜不停发表的文章……NH 应该在这个问题上吸取教训（并且 请不要找借口，它确实应该在文档方面吸取教训。如果你想要知道在那上面应该吸取多大的教训，那么请现在就去查看为_N_hibernate 提供的 DDL SQL 生产文档，看它有多伟大，甚至能够产生……java 类。嗯？），同时还有很多可选择的方法，那真的不是它所拥有的优势。

Felix 建议采用组合式的解决方案：

> 不要相信某人所说的"OR/M 是编码的越南战场"，NH 是老兵，而 EF 是年轻的新兵。不幸的是微软不支持开源，如果可以的话，事情会变得更容易：使用微软提供的设计器和集成工具，使用 NH 作为 OR/M，这会是高生产力的解决方案。

讨论所呈现出来的一般共识是，尽管Entity Framework拥有更好的LINQ提供程序、文档，并且是由微软所支持的，但NHibernate具有大量Entity Framework 4.0 所不具备的特性，像批量读/写、"额外的"延迟、集合过滤器、调整等等。关于这个讨论你的看法如何呢？

原文链接：http://infoq.com/cn/news/2010/01/Comparing-NHibernate-EF-4

相关内容：

- J2EE应用下基于AOP的抓取策略实现
- 数据服务简介
- Blaze Data Services还是LiveCycle Data Services？
- 如何对企业数据进行架构设计？
- 数据库驱动应用程序中影响性能的反模式

热点新闻 | News

Json.NET性能改进，宣称超越其他.NET序列化机制

作者 Jonathan Allen 译者 张龙

JSon.NET提供了更棒的序列化与反序列化机制，其性能要超越.NET中的所有主流序列化机制，包括BinaryFormatter，甚至比大名鼎鼎的DataContractSerizalier还要快。

即便有人说能战胜略微麻烦的 WCF JSON 实现也不值得我们大惊小怪。因为它是基准中所用的唯一程序库，其序列化要比反序列化慢多了，实际上其序列化所需的时间是反序列化的 6 倍多。基准中 让我们感到不可思议的是 BinaryFormatter 竟然也非常慢。大多数人都会觉得二进制格式要更快一些，但 Json.NET 和 WCF 的 DataContractSerializer 的反序列化时间都要比基准快 2 倍而序列化时间则快 3 倍。

大家可以从James Newton-King的博客上查看此次基准比较结果。Json.NET将发布于CodePlex上，基于MIT License。

原文链接： http://infoq.com/cn/news/2010/01/Json-NET

相关内容：

- 为网站和智能手机构建FlightCaster前台应用
- RESTful JSON Web服务最佳实践
- Atom发布协议是一个失败吗？
- JOSH：企业软件组合的新提议
- Flex与JSON及XML的互操作

热点新闻 | News

Java EE 6 Web层综述：Servlet获得异步支持、扩展性改善

作者 Charles Humble 译者 张龙

很多 Java Web 应用都是基于某个框架的，如 Apache Wicket、Java ServerFaces、Struts 或是 Spring MVC 等等。要想使用框架，开发者需要在应用的 web.xml 配置文件中注册框架的切入代码，如 Servlet、Filter 或是 Listener。这么 做的后果就是部署描述符变得很庞大，同时导致框架所用的 XML 与特定于应用的 XML 混杂在了一起。Servlet 3.0 规范的一个主要目标就是让开发者无需编辑 web.xml 部署描述符就能部署 Servlet、Filter 和 Listener，同时可以将 web.xml 文件拆分成多个模块。为了实现这一点，Servlet 3.0 规范增加了基于注解的配置（@WebServlet、@ServletFilter 以及@WebServletContextListener），这使得我们可以不再需要 web.xml 文件，同时规范还引入了一个新的概念：Web 片段（Web Fragment）。

Web 片段可以将框架的“样板”XML 与应用的其他配置分开，并且能够实现应用的自我注册。Web 片段必须放在名为 web- fragment.xml 的文件中，该文件只要位于 Web 应用的 classpath 下即可，但通常都将其放到 META-INF 目录下或是框架的 jar 文件 中。XML 以<web-fragment>元素开始，里面包含的元素与 web.xml 部署描述符大同小异。如下代码所示：

```
<web-fragment>
   <filter>
      <filter-name>MyXSSFilter</filter-name>
      <filter-class> MyXssFilter</filter-class>
   </filter>
   <servlet>
      <servlet-name>myFrameworkServlet</servlet-name>
```

```
        <servlet-class> MyFrameworkServlet</servlet-class>
    </servlet>
    <listener>
        <listener-class> MyFrameworkListener</listener-class>
    </listener>
</web-fragment>
```

容器在部署时会处理 XML 片段并组装成最终的部署描述符。由于容器负责组装 web.xml 文件，因此如果需要按照特定的顺序来调用框架的 Servlet、Listener 或 Filter 时就可能产生问题。为了避免这个问题，Servlet 3.0 API 支持绝对与相对顺序的部署描述符。我们可以在 web.xml 文件中使用<absolute-ordering>元素指定绝对顺序，这样 WEB-INF/lib 下的每个 jar 都可以通过 META-INF/web-fragment.xml 文件的<name>元素获得一个名字。 接下来，Web 应用的 WEB-INF/web.xml 文件可以通过<absolute-ordering>元素按照顺序列举出这些片段名，这 个顺序就是 jar 的调用顺序，同时还有一个可选的<others/>元素用于指定是否以及何时包含那些未命名的 jar 文件。由于部署者可以选 择只列出那些受信任的 jar 以进行部署，这样就可以避免意外情况的发生。除此之外，通过顺序还可以排除那些不需要被扫描的 jar，这样就可以加快应用的部 署速度。最后，如果你不想在产品环境下看到自我注册的情况发生，那就可以在 web.xml 文件中使用 <metadata-complete> 元素，这会告诉 Web 容器只去寻找注解而非 Web 片段。

由于既支持片段，又可以使用注解作为另一种配置机制，Servlet 3.0 可以插入框架的共享拷贝，比如 JAX-WS、JAX-RS 以及 JSF 等，他们都构建在 Web 容器之上，使用了 ServletContainerInitializers。这些框架是通过 jar services API 被检测到的，同时还可以指定其处理的类型列表。对于 WEB-INF/lib 下的任何 jar 来说，只要其中包含的类被检测到都会传递给 ServletContainerInitializer。这样，我们还可以将同样的 API 作为 ServletContextListeners。

从 Servlet API 首次发布以来，构建 Web 应用的方法发生了翻天覆地的变化，尤其是使用越来越多的异步 Web 技术。这些技术（一般统称为 Ajax 或是 Web 2.0）对于 Web 客户端（比如浏览器）与服务器端之间的传输机制产生了重要的影响，因为客户端会在一个页面中向服务器端发出更多的请求而不是每次请求都 刷新一次页面。

长时间的服务器端处理会恶化这一情况，比如等待 JDBC 连接池中的连接，或是等待 JMS 队列中的消息等。在 Servlet 中等待实在是太低效了，因 为这种阻塞会消耗线程以及其他有限的系统资源。鉴于此，Servlet 3.0 引入了异步处理请求的功能，这样线程就可以返回到容器中并执行其他任务。在请求上的异步处理开始时，其他线程或是回调既可以生成响应，也可

以分发请 求以便通过 AsyncContext.dispatch 方法在容器上下文中执行请求。

由于异步 Servlet 的行为与同步的差别非常大，因此 Servlet 3.0 要求开发者指定 asyncSupported=true 以表示 Servlet 支持异步请求。不仅是 Servlet，Filter 也可以异步执行。Servlet 3.0 通过新的 ServletRequest 方法来支持异步处理 ,比如 startAsync() 会返回一个 AsyncContext 对象 ,该对象用于持有 传递给方法的 request 与 response 对象。这里，处理原始请求的线程还可以执行其他操作。此外，API 还引入了一个新的 Listener 类：AsyncListener，它会告诉我们异步操作何时结束或者是否超时了。AsyncContext 类拥有一个 complete()方法，凭借该方 法我们可以在异步操作结束后提交响应。AsyncListener 类拥有一个 dispatch()方法，它会将异步请求转发给容器，这样其他框架（比如 JSP）就可以生成响应了。

除了引入大量的新技术和新方法外，Servlet 3.0 规范还对其他地方进行了大量的增强：HttpServletRequest 终于获得对 multipart/form-data MIME 类型的内置支持了、Cookie 类开始支持“HttpOnly” cookie 以避免某些跨站点的脚本攻击、ServletContext API 也得到了更新，我们可以通过编程的方式将 Servlet 和 Filter 加到上下文中了。

原文链接：http://infoq.com/cn/news/2010/01/ee6_servlet30

相关内容：

- Tiobe编程语言排行 12 月份榜单公布
- Jave EE 6 特性：依赖注入、Bean验证和EJB增强
- Gavin King谈JSR-299 和Weld 1.0 对Java EE与JBoss的影响
- Java EE 6 最终草案暗示了平台的未来发展方向
- GraniteDS不断发展

热点新闻 | News

Sun发布的Java 6 第 18 次更新大大提升了性能并增加了对Window7 的支持

作者 Charles Humble 译者 马国耀

Sun发布了Java 6 的第 18 次更新，这次更新着重强调了性能的改进，包括Hotspot(16.0)新版，对UI应用程序的启动和运行时的改进。该发布还包括了对 Ubuntu 8.04，Red Hat企业版Linux 5.3 和Windows 7 的支持，此外她还解决了 357 个bug。

企业开发者们特别感兴趣的改进当然是Java 7 中提出的垃圾回收站的改进。Garbage First(G1)垃圾回收站（不再是OpenJDK 7 中的实验品） 提升了可靠性和性能，而且，并行扫描垃圾回收站（Parallel Scavenger garbage collector）还包含了对改进的NUMA体系结构的支持。大多数现代计算机都是基于NUMA体系结构的，这该体系结构中，访问不同区域的内存所需的时间是不同的。Java HotSport虚拟机实现了NUMA感知的内存分配器，由它为Java应用提供自动的内存分配优化。比如，每个系统中的处理器都有一个访问低延迟高带宽 的本地内存和存取很慢的远程内存。NUMA感知的分配器是为Solaris (>= 9u2)和Linux(kernel >= 2.6.19, glibc >= 2.6.1) 操作系统而实现的 ,并且可以通过 -XX:+UseNUMA标记打开或关闭并行扫描垃圾回收站。服务器的并行扫描器（Parallel Scavenger）默认使用其缺省值，也可通过指定-XX:+UseParallelGC选项的值打开它。此改动所产生的影响非常大：当在 8 芯片Opteron机器上的使用SPEC JBB 2005基准进行评估时，NUMA感知的操作系统能够带来 30%（32 位操作系统）到 40%（64 位操作系统）的性能提高。

在此次更行中的其他的 Hotspot 变更包括代码生成的改进，如优化了通用字符串合并模式和删除了不必要的整形基本类型之间转换，还有其他新增选 项，比如在完全垃圾回收之前或之后请求堆内存映射或类历史图。遗憾的是，在第 14 次更新中激活的内存泄漏分析的优化，在这次更行中被关闭了，只能等待将来 的某天被再次激活了。

Sun 对于桌面和 RIA 市场的持续关注见证了桌面应用和 Java Web Start 的性能的大大提升，

其中包括：

- 更可观的垃圾回收改进，其中包括新增的客户端和服务端 Java 虚拟机的缺省堆配置。
- 类加载优化让启动更快。
- 应用启动的改进，包括在 Direct 3D 使用时带来的 100 至 200 毫秒的系统时间的节省。
- 对 JavaFX 运行时的预验证的修正支持，它可以加速 JavaFX 应用程序的预热启动，提升空间达 15%。
- 为 Web Start 应用和 applet 并行下载 jar 包。
- 更行了Java Web启动，实现了JSR-56（6.0.18）版并解决了一些关键的bug（6888118, 6800992, 6863499）。

本次发布中的其他变更有：

- 创建 jar 文件的时间降低了 20%
- JavaDB 更新到 10.5.3 版本
- VisualVM 更新到 1.2.1
- StaX 的细微更新（面向流的 XML 处理 API）

在本次发布中没有涉及安全相关的更新，但下一次与安全相关的更新有望在本季度发布。

原文链接：http://infoq.com/cn/news/2010/01/java6u18

相关内容：

- JDK 7 M5 包含了并发性和性能更新，但功能并不完整
- Sun将在Java 7 中摒弃Swing Application Framework
- Project Coin发布语言变化最终列表
- 为Java和Flex编写Mock对象
- Jigsaw蓄势待发

热点新闻 | News

RESTful API认证模式

作者 Boris Lublinsky 译者 黄璜

“所有人都觉得编写客户化认证协议是有必要的”， George Reese说，这也是他在使用云提供者和SaaS提供商们提供的API进行编程的过程中的领悟之一。在一篇博文中他提出了一组旨在适用于任何REST的认证需求的标准。

George 曾开发过各种各样的 Web 服务 API，他发现每一种 API 都需要一种特定的认证机制。

> 我已经疲于在这种事情上上浪费脑细胞了，比如某提供商 A 的要求是在 URL 编码之前或之后为查询字符串进行签名。我也早已厌倦了提供商们诸如要求使用用交互用户的凭证进行 API 调用的这样的认证要求了。

他勾勒了 REST API 的认证机制的设计规则。他说，“让我们变得简单些：如果你不加密 API 调用，你甚至连假装安全都做不到”。

1．所有的 REST API 调用必须运行在使用可信的 CA 签名过的证书的 HTTPS 之上。所有客户端与服务端交互之间必须要验证服务端证书。

> 通过使用由可信机构签名的证书，SSL可以保护你免受“中间人”攻击。中间人攻击的手段是在客户端和服务端之间插入一个代理进而窃听“加密的”通信。
>
> 如果你不验证服务端的 SSL 证书，你就无法知道谁在接收你的 REST 查询请求。

2．所有的 REST API 调用应该通过专门的 API 密钥完成，该密钥由标识成分和共享密钥两部分组成。系统必须允许一个指定客户端拥有多个活动的 API 密钥并能方便地让个别密钥失效。

> 前半部分的重点是发起 REST 请求的系统不应是某个交互用户……REST 认证的的是程序而不是人，它支持比人使用的用户名/密码更强大的认证手段。
>
> 后半部分的意思是，每个 REST 服务器应该支持每个客户端拥有多个 API 密钥。该需

> 求使得孤立潜在危害和当危害发生时解决问题更为简单。[...] 当应用被破坏时，你也需要一种完善的方式铺开替换的 API 密钥。

3．所有的 REST 查询必须通过签名令牌签名的方式进行认证，该过程通过对按小写的字母顺序排序的查询参数使用私钥进行签名。签名应在查询字符串的 URL 编码前完成。

> 换言之，你不能将共享密钥作为查询串的一部分进行传递，而应使用它进行签名。签名后的查询串看起来应该是这样的：
>
> GET /object?timestamp=1261496500&apiKey=Qwerty2010&signature=abcdef0123456789
>
> 被签名的串是“ "/object?apikey=Qwerty2010&timestamp=1261496500" ”，而签名是应用 API 密钥的私钥所得到的 HMAC-SHA256 哈希值。

他承认在大部分类 REST 的 RESTFul API 方案中，认证几乎肯定被看作是次要的问题。然而，在文章的结论中他建议读者“最好参照别人的例子，而不应自创认证模式”。

InfoQ的读者们，请别吝惜你的意见。最初的博文地址是：O’Rielly社区博客。

原文链接：http://infoq.com/cn/news/2010/01/rest-api-authentication-schemes

相关内容：

- JAX-RS，或者说RESTeasy不是RESTful?
- Restfulie作者Guilherme Silveira专访
- IBM WebSphere拥抱REST
- RESTfulie：一个创建超媒体感知服务与客户端的Gem
- 纯GET的REST集成模式——是同步，还是集成？

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 | Articles

虚拟研讨会：软件架构文档

作者Srini Penchikala 译者 晁晓娟

软件架构文档是企业应用开发过程中的重要一环，理解一个项目中的架构文档的关键是理解它在项目生命周期中所扮演的角色。一个项目产生架构文档的根本 原因是为了交流、分析、记录和保存（比如，跟踪决策过程使之不会随着时间而流失）。一个项目产生的架构文档的数量和类型应当反映该项目为了创造产品所需的 交流及分析。

架构文档在项目中用于在上至其管理团队，从架构师或首席设计师到开发者，以及随着时间的流逝，未来的维护者和开发者之间进行沟通所用。仅这个简单的 声明就引出了三个问题，它们的答案有助于决定文档的数量和类型。项目中出现了多少疏漏？ 有多少开发人员在开发产品，他们的水平如何？以及该产品将使用多久？疏漏越多，说明需要更多的文档用来交流管理。开发者越多和他们的开发水平越低，就意味 着开发者需要更多的指导。而产品使用时间越长，则需要越多用于和未来该产品的开发者们交流的文档。

架构文档的交流部分包括与管理层，开发者们沟通和在软件生命周期中的交流。分析需求可能来自为了决定产品的质量（包括性能，安全，可靠性等）的内部原因或来自如兼容某些规则或标准的需求。

在这个虚拟研讨会中，InfoQ 希望能从顶级的软件架构专家们那里找到软件架构文档的重要性和如何记录架构，特别是在敏捷软件开发环境中。

回答我们问题的小组成员有：

- **Len Bass**，SEI的高级技术成员，《Software Architecture in Practice》和即将面世的《Documenting Software Architectures: Views and Beyond》的第二版 的作者之一。
- **Grady Booch**，IBM荣誉科学家，《Handbook of Software Architecture》的作者。
- **Paulo Merson**，SEI的高级技术成员，《Documenting Software Architectures: Views and Beyond》的作者之一。

- Eoin Woods， Barclays Global Investors应用程序架构组的领导人，《Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives》作者之一。

InfoQ：在使用敏捷和精益开发方法（如 SCRUM，极限编程，看板等）的开发团队中，软件架构文档的作用是什么？

Len：架构文档的作用并不因为开发方法的不同而变化。 文档并不是唯一的交流方法，在敏捷中如 SCRUM 或 XP 面对面交流等方法取代了一部分文档的需求。但并没有替代对于设计决策的详细交流的文档需求。面对面 的会议很难正确表达它们。敏捷方法并不能处理跨时间的交流或向管理层的交流。 在敏捷方法中，所减少的文档是面对面交流能够解决的那部分沟通需求，但不会有更多减少。

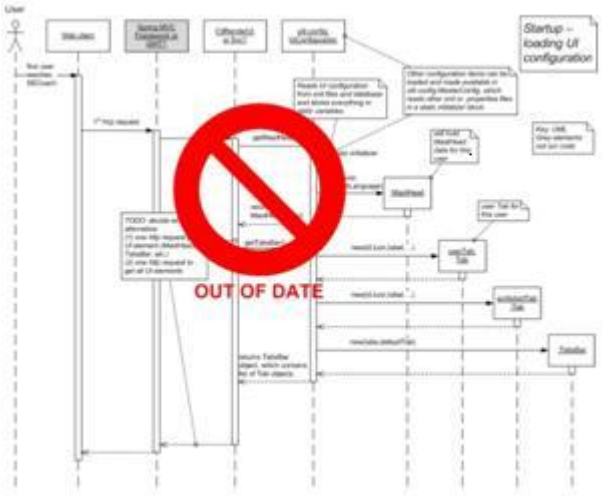
Grady：对于那些使用更形式化的方法的团队来说其角色也一样。简单来说，记录一个软件密集型系统的架构有如下几个目的：让团队推 敲重要决定，去记录组织的重要事件，对于每一次迭代的专门管理提供一些基本资料。对于这些轻量级的方法，架构文档更多地用于反映实际建造的系统，而不是将 要设计的系统。

Paulo：软件架构的角色通常来说是为了在不精确且动态变化的需求与可运行的代码实现之间搭起桥梁。 架构文档捕捉设计决策，它是个指导实现（最主要）的部件，能够在早期用于评估所设想的解决方案的方向是否正确，还有助于任务分配和跟踪。但是对于敏捷呢， 如果说敏捷开发是架构文档的对立面显然不对。敏捷并不是为了避免设计，而是避免预先大量设计（BDUF）。更深远广泛的架构策略是预先制定的，但许多其他 的决策会推迟直到需要时才做。比如说，数据模型经常会被充分的预先讨论以避免之后的大变更。另一方面，在对于那些在下一个版本发布中的要开发的特性，它们 的序列图和接口定义文档可以在实现前几天来完成。

敏捷开发者应该是有设计能力的，而敏捷架构师应该是有编写代码的技能的。 因此设计决策的交流就会更短而且更容易聚焦。实践来看是什么样的呢？

- 多一些图表，少一些散文
- 设计只要足够编程即可
- 图表中多使参考已有的架构（或设计模式的应用方式）。具有相似特性或流程的设计可以用很少的文档，因为解决方案能够从参考的设计那里推断出来
- 使用概图而不是正式的全面的图
- 设计图被划掉而不是更新，这样好过保留过时的设计。如下图是我经常使用的一个可视

化标记



Eoin：可能听起来很明显，但是这个问题的回答实际上取决于团队，项目和相关的背景。 一个架构描述，如同任何可交付的产品，是需要为了某个目标和受众而创建的，否则只是浪费时间。有时候读者就是创建者本人（为了帮助分析或只是一个备忘 录），但是有时候通常会是更广泛的人群，他们对任何重大的交付都有兴趣。

我个人来说，我从来没有发现在敏捷开发和软件架构中有什么巨大的矛盾。即使我知道这并不是每个人的经验……也许只是我比较幸运。我的经验很清楚，不管是敏捷还是其他，简明的架构描述，并随着项目的进度进行增量开发能给开发团队提供一个有用的背景环境。

一个好的项目描述捕获设计的那些从代码看来并不明显的方面（比如部署设计，外部接口或系统监视方法）并且解释这个系统是如何满足它的非功能性需求的（典型的包括硬件，安全和性能的需求）。如果没有人对这些感兴趣，那么就没有必要存在架构描述，但是我发现这很少见。

还有一点值得提到的是架构描述会有很多种形式，在敏捷团队中厚厚的文档常常是有效性最低的那个。重要的是在创建架构描述时你不断收集的信息及进行的 分析而不是你所用的文档模版。一个架构描述必须是一个有效沟通的媒介，包括 wiki，模型，数据库，电子表格以及一组短小且重点明确的文档都可能是比重量 级大型文档更适合捕捉和交流架构设计信息的方式。

InfoQ：当基于 DSL（领域特定语言）来记录应用程序的架构文档时有没有特殊的模式可遵从？

Eoin：我还没有在某个生产系统上用过 DSL，但是我不觉得实现技术的不同会对我记录架构文档的方式有很大影响。你仍然需要去定义 系统环境、功能结构、部署环境、信息结构及

运营环境等，而且还要解释非功能性需求是如何达到的。你需要在每个部分（或视图）描述你的设计的基本元素。可能 因为实用技术不同而有所不同，但你仍然需要描述一些相同的内容。

Len：如果扩展 DSL 语言，概要文件和原型，额外的建模语言以及元模型等定义了符号（有的有，有的没有），那么他们会有助于使文档更加简明。如果读者和作者都理解这些额外符号的话就能够减少成本。

Grady：不是的，任何描述软件架构文档的方式的本质是为了让相关者明确他们关注内容的视图。确认视图上的内容取决于领域和开发文化。

Paulo：架构文档包括多个体现系统结构的视图。 用 DSL 并不影响这些视图，比如 部署视图或运行时视图（又叫组件和连接器或进程）。使用 DSL 使得架构中的模块(或者叫代码)视图更明显，这个视图显示了实现单元，它们如何在模块和子模 块中组织的，以及它们之间不同的关联和依赖。在任何一个视图中，标识出描述各种元素的类型很重要。DSL 规定了良好定义的的一组元素类型和关系。特别是当 DSL 和其他语言混用在一个完整解决方案中的情况，在架构文档中标识出元素类型非常重要。在实际中的大部分情况下，概览图就是你需要的全部。因此你使用不 同的形状或颜色标识不同的元素类型的时候也标明了在文档中一个长方形和一个椭圆形分别代表不同的意思。而且别忘了也要区分线条和箭头。

DSL 经常和自动化工具，类库或代码生成器一同出现。因此明确实现的范围及哪些是由工具实现的也很重要。上下文结构图会有所帮助。当使用代码生成器 的时候，要确保生成了哪一类的运行元素以及它们的属性。找出是否单进程/线程，使用何种通信协议，使用什么数据存储机制以及访问质量相关方面 ,比如安全， 性能和互操作性等。我曾经参加用 Microsoft Visual Studio 定义 DSL 模型的演示 ，很难忘。但即使你用 DSL 你也可以用一个通用的建模语言如 UML 来记录设计。UML 概要文件经常用于定义从 UML 符号衍生来的定义领域相关的符号。

InfoQ：新的动态 Java 模块规范(比如 OSGi)的哪些方面符合软件架构文档的工作？

Paulo：软件架构师们通常关注在架构文档中的模块视图（也叫代码视图）。他们花时间解释在程序包和子程序包中实现单元是如何组织 的，模块间如何相互依赖，它们如何按水平或按层次组织的，以及特殊事务的消息流程。创建架构的这个代码视图固然重要，但架构师们常常忽略运行时视图和部署 视图。记述这些视图需要对运行环境、平台结构、组建模型很好的理解。运行时视图体现了实现单元如何被转换为线程，进程，Servlet，DLL 等其它组 件，还体现了数据存储、基础结构服务、外部服务、通信机制（如 JMS，SOAP，http，本地或远

程方法调用）等方面。运行时视图允许你来推算运行时特性，如性能、安全及可用性等难以看着程序包和类图就能估计出的 部分。部署视图不仅体现了硬件基础架构，而且解释了部件如何打包到一起并部署的。这些视图有助于能够帮助对服务能力，性能，安全性和其他属性的估计。

现在让我来解释清楚你的问题。如果你在架构一个 OSGi 应用程序，你应该理解在 OSGi 运行环境和它们的生命周期中都有哪些种组件。我的唯一 OSGi 经验是开发一个 Eclipse 插件，它有点隐藏 OSGi 框架的意思，你只需要创建声明（manifest）文件。 但让我问 OSGi 应用程序架构师们一些问题：

1. 你的架构文档体现了每个部署包里都有什么 java 包/类吗？
2. 在运行时，对于每一个包都有对应的线程吗？或者每一个包的服务都有一个独立的线程吗？
3. 对于以上任何一种方式，你的文档都列明了这些线程吗？
4. 每一次服务调用都创建一个新的线程实例吗，类似于每一个 http 请求同一个 servlet 的多线程实例？
5. 如果我想要部署一个被多个包调用的 jar 文件，文档里能知道我应该把它放哪吗？
6. OSGi 的发布-订阅机制发送的事件是同步的还是异步的？
7. 你的运行视图区分了调用的类型和使用的协议（比如 http）吗？
8. 文档体现了服务注册的交互吗？这肯定会影响可用性，性能和安全。

第一部分的答案是创建架构的运行和部署视图来反映运行环境的组件类型，通信机制和基础服务。

第二部分的答案是关于易变性。我们的结构文档模版中有专门关于它的一节。在变更指导部分我们描述了让系统变得可配置的解决方案的机制。以下是一些变化机制例子：

- 替换提供同样接口的组件
- 组件复制
- 组件可选的包含（插件和附件）
- 构建时组装配置

类似于 OSGi 的框架给了你这些开箱即用的变化机制。架构的运行时情况可能会随组件的不同而变化。 架构文档应该在变更指导中记录什么组件会变化，有什么可选项或配置及其影响，什么条件下可能会触发特殊的选项，和每个选项的绑定时间(对于 OSGi 可能是 运行时，但也可能是构建或部署时)。

Eoin：任何让一个系统的组件结构在运行时（就设计时而言）更加可见是一件很好的事情，现在当我们从设计进行到代码时，我们丢弃了 很多这样的信息（因此需要一个架构描述--参见前述答案！)我最近在学习 OSGi 并且看起来它解决了这个高度关注的问题，帮助系统功能结构更容易被理解。 同样的，像 OSGi 这样的模块技术只是帮助展示了架构结构的一种，而在实践中，还需要考虑其他种类。因此即便它们可能提供了很多其他技术上的好处来证明他 们自己，它们依然不允许我们放弃我们的架构描述。

Len：一个标准的视图软件架构是模块分解视图。 每一个视图中的模块都代表了一个封装范围。OSGi 是指定封装范围的一种方式，它很适合模块分解视图。

Grady：OSGi 主要关注 Kruchten 所说的实现视图，也就是与组件的物理打包及在随后为部署视图做准备相关的事情。它们处理了这些组件生存的物理拓扑结构。这样，从一个架构的角度来，OSGi 只是一个特殊的封装打包技术。

InfoQ：UML 的特性如"配置文件" 和 "模版" 是如何帮助记录软件架构的？

Grady：这些是主要的 UML 可扩展性机制……但是以我的经验来看，基本的 UML 语言已经足够记录大多数软件集约型系统的架构核心部分了。

Paulo：UML 最初是为了给面向对象系统建模而形成的，现在被看作通用的建模语言。 意味着同样的符号能够被用来代表很多不同的事物。比如，一个叫 Dispatcher 的 UML 组件可以很好地代表 Java EE 中一个 Servlet 或在 OS 内核的一个线程。 你可以定义模版来使 UML 符号专门化，如果你知道你的一些组件将会是 servlet，就把它们模版化为<<servlet>>。

如果使用正确的话，模版会自动的把你的 UML 图表达得更清楚。UML 自带一些标准模版，但我们也可随意创建自己的模版。一个 UML 配置文件是一组有 机组合的 UML 模版。UML 中有一个 Java EE 的配置文件，一个.NET 的配置文件，一个 SOA 的，一个系统工程的以及更多其他的配置文件。 重用或重新创建模版及配置文件都是个很好的主意。

Eoin：一句话，我认为他们是救生圈。 UML2 的扩展特性是很突出的特点之一，并且该特点也让 UML 可用于很多不同的领域。 很好利用它的人很少，但是现代 UML 工具已经最终支持了语言的扩展能力并且能用模版和配置文件来调整 UML 使之适合更手头的工作所使用特定的语言。只需要 很少工作，你就能定制一个定制化 UML 版本专门用于表达你的架构中的

那些结构。其结果是有了一个基于 UML 的架构描述语言和现成的工具支持，因此能克服多数基于研究的 ADL 语言（如 xADL，Acme 或 Darwin）引进（采用）时的主要障碍之一。

InfoQ：有一些新规范描述了在 SOA 中进行服务设计时的 UML 概要文件和元模型，比如面向服务架构的建模语言（SoaML）。在使用 SOA 和云计算架构模型的应用程序中架构文档应该如何来做呢？

Eoin：我的观点是 SOA 和云计算是相对独立的技术（即使一个能让另一个更容易）。SOA 是一种架构风格，它指导你将系统设计成一 组松耦合的通信服务；云计算是一个部署模型，它通过运行时平台的虚拟化来简化系统所驻存服务器的提供，运维和管理。他们都是与位置独立相关的技术，但是从 不同角度解决该问题。 比如 DSL，我并不认为这些技术从根本上改变了你需要从架构描述中捕获的内容或者最多需要捕获多少。然而，就像我们上面所讨论的，定制型的建模语言（如 SoaML）肯定有助于让记录某些架构类型的过程更加直截了当。

Paulo：SOA 是一个架构风格，能够用很多技术来实现，比如 SOAP Web 服务，REST，消息系统，BPEL 和 ESB。我认为在实际中当你创建一个基于 SOA 的架构时候，很快你就要明确你将要使用的技术。

比如，我认为如果在 SOA 设计中一个服务提供商组件被模板化为<<dll>>或 者<<servlet>>而不是通常的< <participant>>，对于读者来说这就更加一针见血。 想一想，软件开发者们不得不不断跟进新的编程语言、脚本语言、技术、开发框架、设计模式、IDE 和插件、建模工具以及开发技术的增长速度。

作为一个软件开发者，如果 UML 概要文件本身很简单，且包含了我已经熟悉的解决方案的元素模板，那我就愿意去学习它。 我认为一个技术无关的 UML 概要文件（比如 SoaML）受到广大架构师的欢迎是很难的。如果你用的建模工具支持它的话，熟悉这个配置文件当然非常重要。不 管有没有 UML 概要文件，要正确记录采用 SOA、云计算、OSGi、P2P 或其他架构方式的应用程序，关键是要包含一个用于明确表达该方式的特定的元素类 型及关系类型的架构视图。

Grady：没错，它关系到哪些是重要的设计决策的决定。在选择使用服务的系统中，消息本身的设计（不只是点对点的服务）是非常重要的（但也经常被忘记）。在云计算的场合（顺便提一下，这个词汇在当下更迎合市场而不是技术），更多关注的是部署视图的元素。

InfoQ：哪些是软件架构师们在记录他们的软件架构时需要牢记的最佳实践和“陷阱”（gotchas）？

Len：

- 软件架构师们需要牢记生成文档的目的，过度或不足都是对文档目的错误判断的表现。
- 项目并不是使用文档作为经年沟通的最佳决定因素，因为随着时间的消逝，大部分项目组成员将不再担任维护和升级的工作，文档是否存在对于他们没有什么利害关系。
- Wiki 被证明为维护架构文档的一种有效的媒介。

Grady：我想到三件事情：不要过度文档，对每一个迭代保证文档的新鲜度，一定要注意那些不易从代码本身抽取出来的视图及横切关注点。

Paulo：

- 用多个视图来记录架构。
- 总是记得在你的图表中做记号。
- 如果你使用了一个设计或架构的模式，请让你的读者知道。
- 遵循模版。
- 只有当某些信息对相关的人有用的时候才花时间记录。
- 记录重要的设计决策的理由。
- 记得检查你所设计的内容。

Eoin：我想起的第一件事就是 SEI 的《记录软件架构》（Documenting Software Architectures）一书中有一个很好的列表，七个“优秀文档规则”，目前也是他们在网站上公开的技术报告之一（SEI-2000-SR4）。

我认为对这个问题的全面回答可能需要一本书，但我认为记录软件架构时必须牢记的一些更重要的以下几点：

- 记录时有明确的目的。
- 对明确的受众来创建架构描述。
- 描述精确，即使你希望抽象。
- 谨慎定义你使用的符号，这样其他人能够清楚地明白他们的意思。
- 使用一组软件架构观点定义作为辅助备忘录来指导描述的内容。
- 不要让文档成为关注的焦点，你所从中获得的过程，内容和交流才是最重要的事情。

- 迭代地描述架构，经常检查并确保它的正确性及有效性。

InfoQ：在当前的经济和市场情况下，企业中的软件架构文档如何能对该公司有所帮助？

Grady：这是一个把架构像工艺品一样看待的精髓之处，在 IEEE 软件的架构部分，在我的专栏里我刚写了一篇关于这个主题的好几页的文章。

Paulo：Steve McConnell、Barry Boehm 和其他一些人已经发现源自需求或架构的一些缺陷很难修复。 换句话说，一个稳健的架构是有回报的。你的软件架构师有很多经验并且对模式、目前的技术、建模工具和语言非常了解并不重要，因为如果这个家伙创建了一个非 常独特巧妙地设计但是开发者们并不理解它，这个项目并不会从这个好架构中受益。软件文档用于交流想法时很容易失败。一个原因是软件工程师通常没有受过写技 术文档培训。好的架构文档可以推动：

- 架构评估
- 软件重用
- 成功的采购计划或征求书
- 项目管理任务（估计，分配作业，跟踪）

总之，详细程度恰到好处的好文档会提高软件项目成功的几率。

Eoin：如果能被很好的开发及使用，架构描述和一直进行的架构检查能提供给组织更多的在建系统（以及那些已经存在的系统）的可见性来解答以下类似的疑问：

- 我们的系统有正确的功能吗（个别地或共同地）？
- 我们的系统之间有重复吗？
- 我们的系统是持续一致的方式来开发的吗？
- 我们有没有过度的复杂、灵活、协作、安全或其他潜在的昂贵系统的特性？
- 我们是不是建立了一个不够灵活的系统，它会导致我们接下来会花费很多来改进它？
- 我们有没有犯成本方面的错误？

Len：软件架构文档是为以上的目的服务的。他们帮助一个企业实现它所期望的对项目监管、强化开发者们交流，强化经年年的交流以及对设计的分析等目标。

InfoQ：你认为软件架构文档的未来会是什么样？

Eoin：我希望未来我们需要较少的软件架构文档因为我们将能够在代码和运行时系统中看到架构！今天我们需要很多架构文档的原因之一 是我们手头的技术还不能直接地表示架构结构。我希望能看到我们的架构结构体作为一流的实现结构，而且架构文档能够发展并用于捕捉决策，逻辑和分析，而不只 是捕获结构。在达到这一理想的过程中，我希望在 DSL 和 ADL（架构描述语言）领域的工作能更直接指引前进的方向，当我们改善描述语言的时候，我们也在致 力于寻找如何把信息准确嵌入运行时系统中的方法。

Paulo：软件架构原则还是个很新的概念，当一个架构师创建出架构文档后，它能很容易地被一个从来没有和这个架构师合作过的一个开 发者所理解，实现这一目标还有很长的路要走。达到这个目标的方法是让新的架构师在学校学习软件架构而不是在实战中不断的尝试和出错。这种教育包括软件架构 的正确表现形式以让其他人更好地消化和理解。通向良好的软件架构的教育这个方向的重要工作包括： Grady Booch 在软件架构手册方面的工作以及 SEI 开发的课程和出版物等。

Grady：当下在架构框架相关方面：TOGAF、NEA、DoDAF、MoDAF、FSAM、Zachman等已经花了很多精力。 好消息是有一个这些框架和方法相关的充满活力的对话在进行中—但是我希望随着时间他们之间会有洗牌或变得更简化。

Len：理想的开发环境是文档只需要按一下按钮就可得到。这需要一个集成的开发、需求管理及项目管理的环境。尽管离它的到来还需很长时间，它依然是一个值得我们努力的目标。

原文链接：http://infoq.com/cn/articles/virtual-panel-arch-documentation

相关内容：

- Mingle将与Google Wave集成
- Tasktop Pro 1.6 支持C/C++项目的任务管理与自动化的时间跟踪
- JIRA 4.0 发布：功能改进，价格更低
- 开源自动测试框架Tellurium
- Java程序员学习Flex和BlazeDS的十三个理由

推荐文章 | Articles

SQL Server报表服务以及使用重叠数据

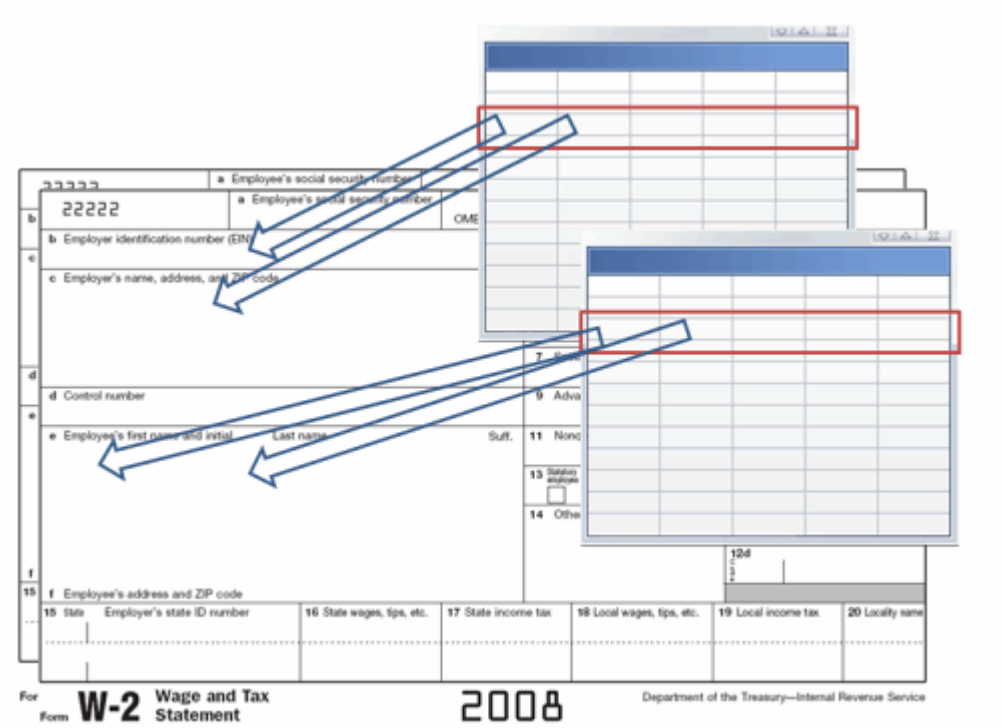
作者Grzegorz Gogolowicz and Trent Swanson 译者 李高任

代码可从MSDN代码库下载。

导引

作为业务的一部分 ,许多公司都需要由扫描图像或各种官方支持的不同格式来生成报表和表格。对于这些文档，通常都有着严格的格式要求，甚至连对文本 框、标签大小和字体样式的丝毫变动都不允许。而且，重新创建与原始表格完全一致的报表，是一项代价可能很大、也非常容易出错的任务。填充表格的数据通常存 储在关系型数据库中，在输出的表格中，必须将它们显示在准确的位置上。这种情况有时被称为固定布局的报表，有时被叫做像素级完美报表。

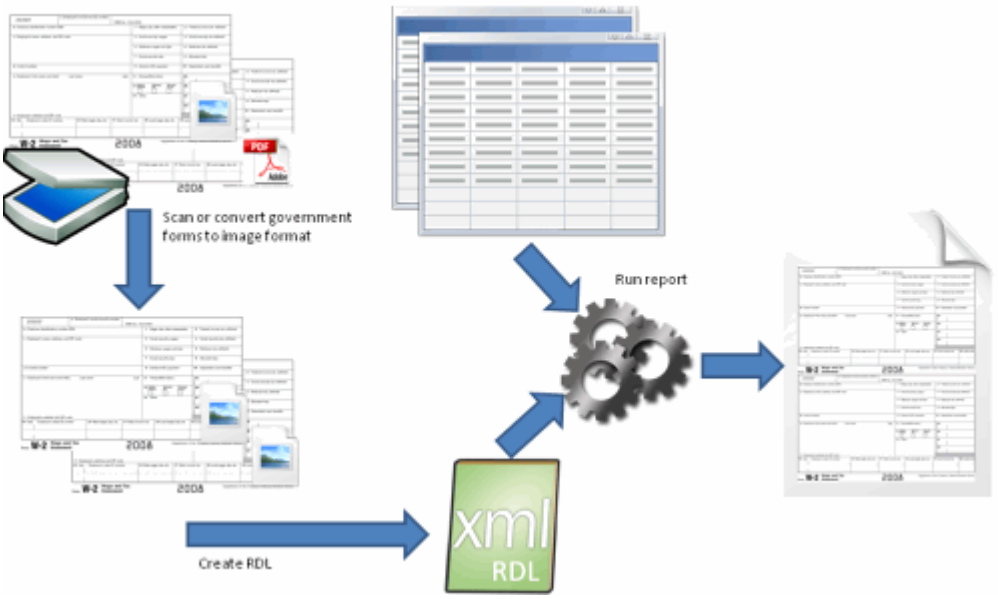
本文基于 SQL Server 2008 的报表服务(SSRS)讨论了针对这一问题的解决方案。在本方案中，我们假设有一个表格以由政府或其他有关当局指定的 PDF 或图像格式的方式输 入。该表格的构成是：固定元素（各种标题、标签和说明）和可变文本或图像元素，这些变化的部分依赖于存储在企业数据库中的数据。另外，这个表格可能会是单 页或多页的。



场景

一家企业扫描或下载了一张官方表格，以生成员工信息表。它可能是图片，也可能是PDF格式，其实任何以PDF格式提供的表格都可以很容易地转换为图像。在此例中，我们使用“关于工伤/病的雇主报告 — 纽约州 — 劳工赔偿局”。

由于 PDF 格式的文件很容易转换为图像格式，我们将转换出来的图像作为新 SSRS 报告的背景。该图片中有源表格中的所有固定元素项。在 SSRS 的报 表设计器中，我们在所有需要显示数据库中信息的位置上添加文本框和复选标签 ,并根据其背景图像上的线条来设定它们的位置。因此，该图片将被数据库中的数据 以准确的位置加以覆盖。该报告可在所有 Web 浏览器的页面中显示正确且不失真，并能够在不损失质量的前提下进行打印。用户还可以输出报告到 TIFF，PDF 格式，或者 SQL Server Reporting Services 支持的其他格式。



解决方案概述

最核心的需求是在所有典型场景下对报表应用支持的灵活性。这意味着，报表要能在所有浏览器的上无失真的正确显示，以及能够无质量损失的进行打印输出，当然，还应该有一个导出报表不同格式的选项，如 TIFF 或 PDF 格式。

挑战

本解决方案中出现了一些不同的挑战，主要是由于报告要按照我们的要求在网络浏览器显示和打印。浏览器通常不能理解或支持图像编码中"点每英寸"（DPI）的计量单位，而总是以 96 dpi 分辨率来显示图像。 96 dpi 的精度在打印报告和表格时通常也不能被接受，而是更高的分辨率，一般 300 dpi 或以上，才能够达到可接受的印刷质量。

A：HTML 渲染时使用的 96 dpi 分辨率光栅被 ReportViewer 组件用于浏览屏幕上的图像。与此同时，在有打印要求的其它场景下，需要较高的分辨率（300 dpi 或更高）才能保证可接受的文档打印质量。

B：当使用高清晰度的图像时，打印预览将消耗大量资源，在某些情况下甚至会因为性能太低和内存溢出而无法使用。

C：HTML 渲染不支持元素的叠加显示。这意味着，如果在报表中包含有放置在图片图像元素（如背景或表单中的图像）之上的文本框（表单中的某个字段域），那么它在浏览器中将不会被显示在图片在上面，而是在最右边。而其他一些内置的渲染器是支持元素叠加显示的。

权衡和决定

各种业务都需要固定布局的报表，它们往往会有许多不同的需求，而这些需求都有着不同的优先级。我们将各种不同的需求和多样的选择列出来，一起讨论不 同方案的利弊。我们将不会触及这些方案的细节，而是讨论如何设计和实现一种能满足既要让报表在浏览器中显示、又可以进行高分辨率打印需求的方案。

所有场景都使用一个 96 dpi 分辨率的背景图片

利：这种方法意味着解决问题的最简单方法。

弊：以 96 dpi 分辨率的图片为背景的报表在 ReportViewer 里应该能正确显示，但在打印或输出为 PDF 时，质量应该会很差。

所有场景都使用一个高分辨率的背景图片

利：报表基于高分辨率图像组件，可以获得高品质印刷效果。

弊：报表在浏览器中的效果不会像打印它时一样。预览进程可能会因消耗大量资源而表现不佳，并很可能最终导致内存溢出的异常。

在一个报表中混合使用低分辨率和高分辨率的图像

利：将一个报表内同时放入低分辨率和高分辨率图像，能够涵盖最典型的应用场景。

弊：这种方案在设计每张报表和在进行可复用的开发时，都需要额外的代价。这种方案的另一种实现方法是针对网页和/或 Windows 应用实现一个定制的报表查看器。

高低双分辨率解决方法

此解决方案在一份报表中使用两种不同分辨率的图像，一张 96 dpi 的图片和一张本例中为 300dpi 的高分辨率图片。ReportViewer 默认就定制有此功能，可以在不同场景中选择正确处理的图像，如在浏览时和打印时。

在低分辨率时对打印质量低下的挑战（见上文挑战 A），通过在每页中使用两个不同版本的图像，并执行一个特殊的逻辑在浏览和打印时来切换这些图像得到了解决。

第二个挑战是在高分辨率图片时如何在屏幕上正确显示报表（见上文挑战 B），这个只要不在浏览和打印预览时使用高分辨率图像就解决掉了。我们在 WinForms 报表查看器中指定预览时使用低分辨率的图像，在网页激活打印时使用 PDF 格式预览。这使我们能够显著减少对性能的影响和内存的压力。

第三个挑战是 HTML 渲染不支持元素重叠显示（见上文挑战 C），只要不使用 HTML 渲染器，

而使用内建的 JPEG 渲染器就可以解决这个问题，一次只将报表的一个单页渲染成一张 JPEG 图像，并建立一个自定义报表查看器来浏览报表中的每个单独页面。

至此，我们已经创建了两个自定义 ReportViewers：一个是针对 Windows 应用，基于 WinForms 的 ReportViewer 控件；另一个是针对 Web 场景，基于自定义 aspx 页通过 web 请求充分利用 SSRS 的功能，同时内部调用 WebForm ReportViewer 控件。报表每一次在屏幕上显示时都会使用 96 dpi 的低像素图片，而在打印或导出报表时就会使用高分辨率的图片。由于报表的代码本身不能确定是在什么环境中被使用，我们就在报表中增加了一个名为 ForPrint 的布尔型参数。我们的自定义代码会在报表是在被显示在屏幕上时将该参数设置为 False，而当报表是在被打印或输出时，将其设置为 True。

我们的解决方案见下图所示：图 1 所示的是针对 Windows 应用，图 2 是针对各种 Web 应用场景。

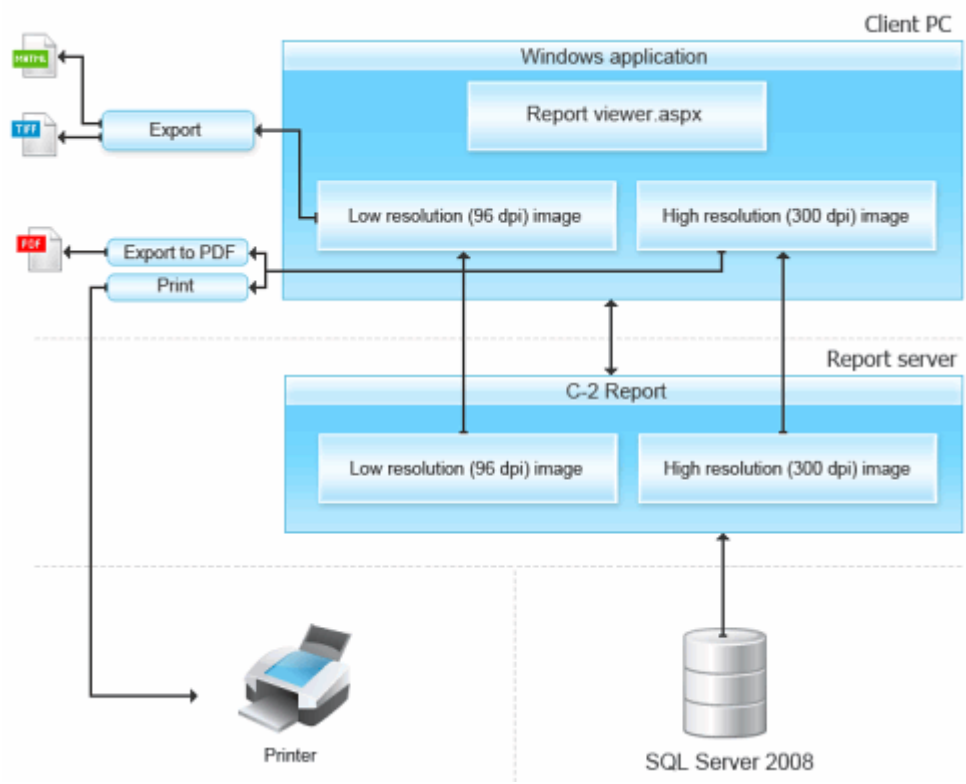


图 1 Windows 应用程序解决方案概览

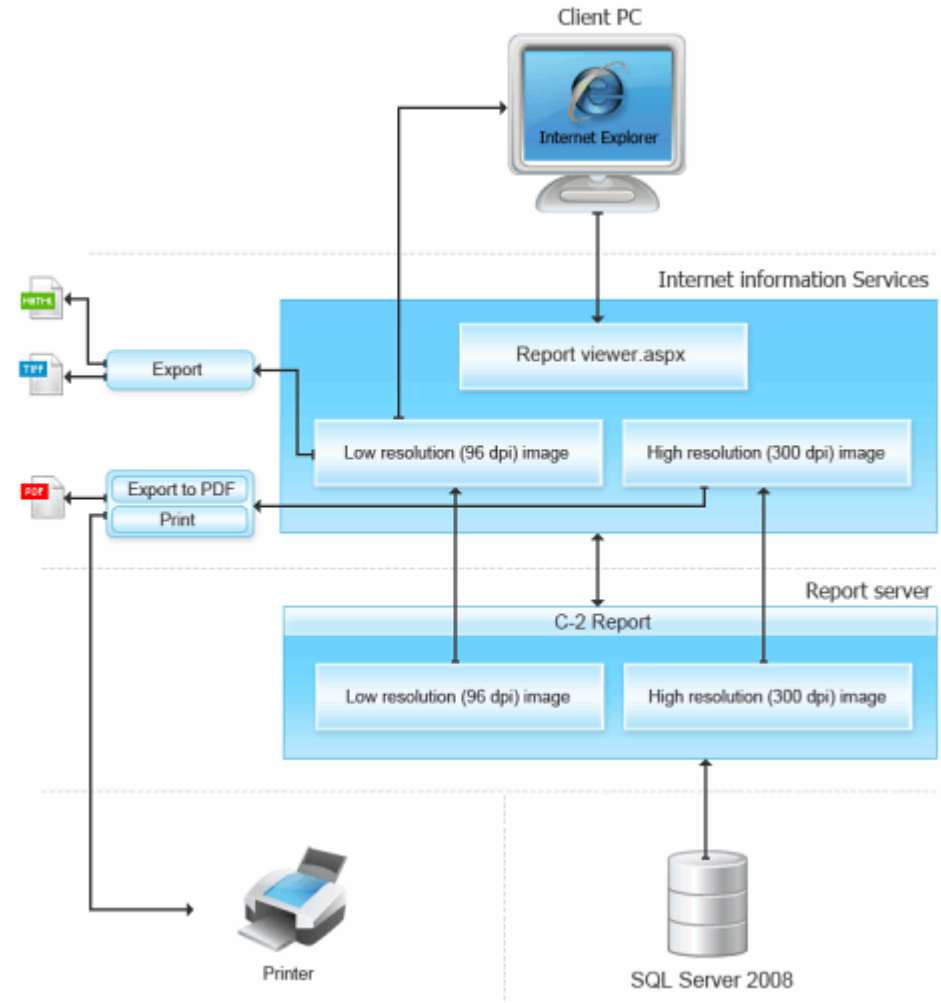


图 2 Web 应用场景解决方案概览

实现说明

如前所述，我们解决的挑战之一（上文提到的挑战 C）是与网络浏览器的图像失真有关：默认 SSRS 的 HTML 渲染器会让文本框和复选框控件、与设计好 的相对于背景图片的位置有轻微偏移。产生这一结果的原因是，HTML 标签生成的报表包含有一个 HTML 表格/网格和叠加于网格不同单元格中的控件（如编辑 框）。这些单元格的坐标在不同的浏览器会以不同的方式计算（例如，Internet Explorer 7 似乎是四舍五入到下一个整数），而且偏差会随着行和列的增加替增。不同的浏览器都会让最终的图像显示有不同的失真，但却没有一个浏览器能让报表实现像素 级完美的呈现。

一种可能的解决办法是开发自定义的 HTML 渲染器（查看 MSDN 的文章可以获得更多关于自定义渲染器的资料）来渲染那些经常用于固定布局的报表组 件，例如文本框，图像，Tablix 等。使用自定义渲染器已经超出了我们的示例范围，我们决定使用更简单的解决方案：用一个自定义的 aspx 页面代替默认 的 ReportViewer.aspx 来生成固定布局的报表。浏览报表时，

这个自定义页面通过定制相应的网址发起 Web 请求来调用 SSRS 得到报表的各 页，它们是 JPEG 格式的 96 dpi 图像。在报表各页之间浏览并打印或者导出时，我们在页面顶部添加自定义按钮。打印的实际上是后台导出到 PDF，因此实际上是后台生成的 PDF 文档被 发送到打印机，而非网页显示的内容。

我们的方案关注于解决基本问题：通过在表格图像上叠加数据来生成报表，我们已经说明了其可行性。一些个别的报表可能需要额外的自定义代码。例如某些 表单的字段，跨越了几行，每行的长度也可能不尽相同，这使得标准的报表组件很难处理。类似的还有，一些表单有些固定字段，如 SSN 就是由多个独立的单元格 组成。对于这些额外要求，可以通过定制报表组件的报表设计时行为和运行时行为来实现 :该设计时行为允许精确定义每条件的尺寸和字符串“切分之后”填充到单 元格的每个字符和准确位置；而运行时行为则会考虑在设计时就已经定义好的尺寸和“间距”。更多关于自定义报表项的信息你也可以在这里找到，但是我们认为实 现它们已经超出了本文的范围。

实施细节

我们的解决方案包括三个部分，每个都由一个 Visual Studio 2008 的项目代表：

1. 图像上重叠数据的多页固定格式报表示例
2. 在 Windows 程序中浏览，打印，导出各种报表的通用解决方案示例
3. 基于 Web 的浏览，打印，导出各种报表的通用解决方案示例

环境要求

要打开并运行它需要安装 Visual Studio 2008，SQL Server 2008 并将这两个产品正确地集成，最简单的集成方法就是将两种产品都安装在本地。

对于 PDF 格式页面的浏览，并从网上打印网页，您将需要安装 Adobe Reader 6 或以上版本。

为有图表叠加内容的 C - 2 表格创建一个 SSRS 的报表

以下是如何使用SQL Server 2008 Reporting Services创建一个多页固定布局的图表叠加报表的步骤说明。在此例中，我们使用一个命名为“关于工伤/病的雇主报告 — 纽约州 — 劳工赔偿局”的多页表格 -以下文简称C2 表格。创建一个单页报表可以按照如下步骤进行。

示例报表是完整可用的，可以通过在 Visual Studio 2008 中打开“Fixed Layout Image Overlay Report.sln”工程来查看。报表需要使用的数据结构和样例数据，可以在 SQL Server

Management Studio 中执行“SQLScripts”目录下的“C2SampleData.sql”脚本来生成。

第 1 步 — 创建表单图像

每个表单页面都创建两个图像：即每页都有一个 96 dpi 的版本（下文称之为 c2_Page__96，用于屏幕浏览）和一个 300 dpi 的版本（c2_Page_，用于打印）。

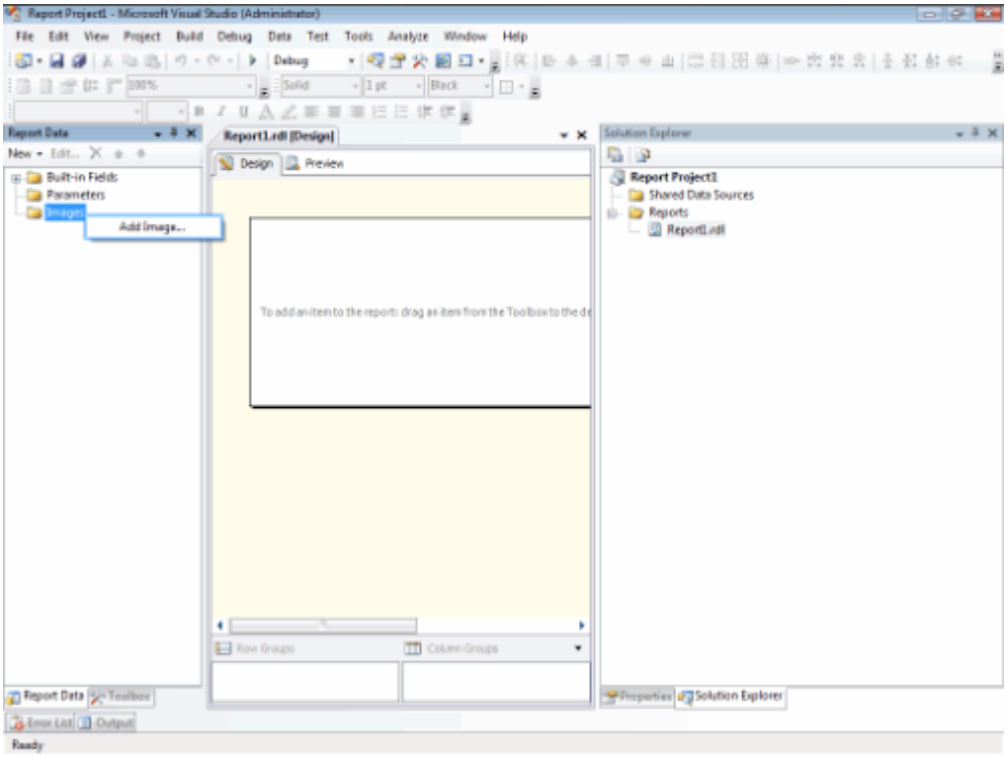
我们还创造了一个用于检查的小图像（“Check.jpg”）。

第 2 步 — 创建项目

打开 Visual Studio 2008 并创建一个报表服务器项目。添加一个新的报告，不要使用向导（右键单击解决方案资源管理器，选择“Add” ->“New Item”。在“Add New Item - Report Project 1”窗口中选择“Report”，然后点击“Add”按钮）。

第 3 步 — 将图像添加到项目

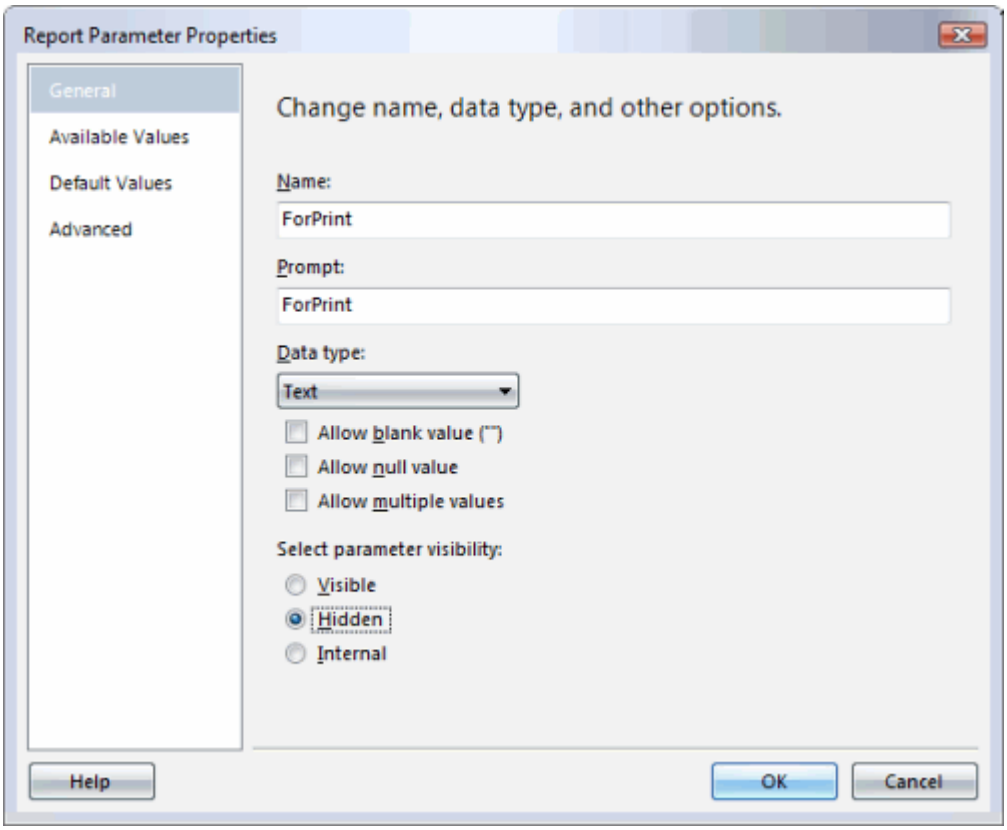
将所有在步骤 1 中创建的图像添加到报表项目中。在此演示项目中，我们将使用最简单的方法在项目嵌入图像，但需要注意的是，当图像文件总的大小超过报表服务器对 rdl 文件尺寸的限制时，它将无法工作。这种情况下，应该使用外部图像。



第 4 步 — 添加模式参数

添加一个隐藏的名为 ForPrint 报表参数名，布尔类型，默认值为“false”。这将用于以编程方

式切换低、高分辨率的背景图像。要添加该参数，在“Report Data”窗口中的右键点击“Parameters”选择“Add Parameter”即可。



第 5 步 — 添加报表主 Tablix

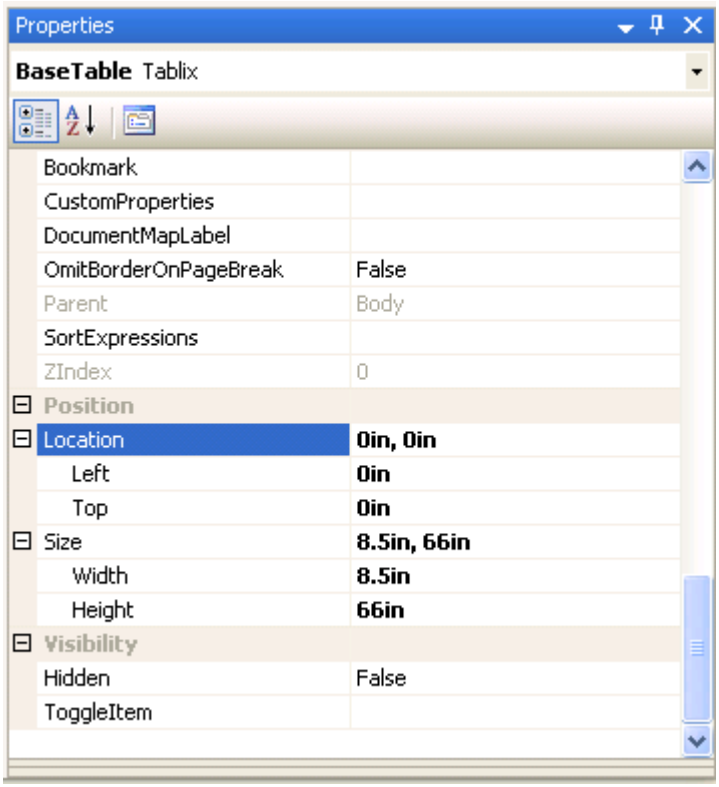
从工具箱中将“List”报表组件添加到“Body”区域。在“Data Source Properties”窗口中设定数据源信息(它会在拖进"List"控件之后自动弹出) 然后在 Data Source 栏中编写查询来提供数据。重命名 Tablix 为 BaseTable，从工具箱把拖动矩形控件到 BaseTable 区域并重命名为 BaseRectangle。

第 6 步 — 设置页面大小和分页符

在“Report Properties”对话框中选择纸张尺寸（8.5in * 11in），并设置边界距都为 0 的。

调整区域大小，使之为一个查询记录的所有页面的大小之和（例如，8.5in * 11in 是一张信纸的大小，8.5in * 22in 是两张信纸的大小，8.5in *（11in * N）是 N 张信纸的大小）

将 BaseTable 的位置设置为 (0in; 0in)，并将其尺寸调整到与 Body 大小相同。设置 BaseRectangle 的 KeepTogether 属性置为 False，以允许软分页方式。



第 7 步 — 添加页面

为每个报表页面的 BaseRectangle 容器添加一个矩形（检查该矩形的 Parent 属性，其值必须是“BaseRectangle”，而不是“Body”！），将其命名为 Page_。

矩形的大小是基于原报告页面的大小，位置为 x = 0，y<=(之前所有页面高度之总和)，例如：（0in; 0in），（0in; 11in），（0in; 22in）。

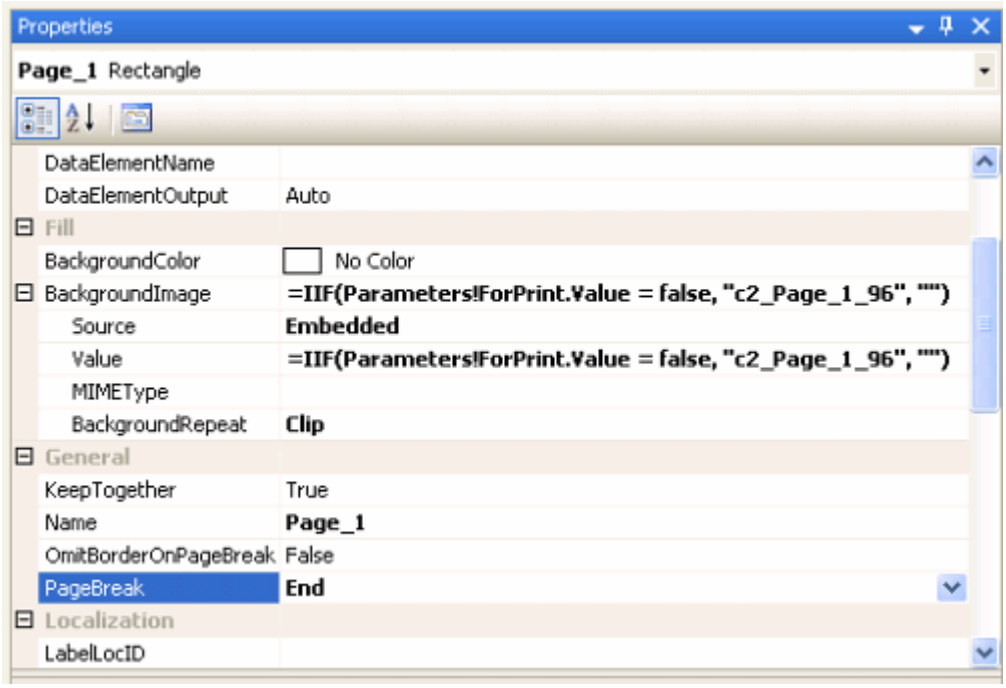
第 8 步 — 添加低分辨率的背景图像

低分辨率的图像将只用于报表在屏幕的显示——因为如果用它来打印，质量会很差。

我们将使用的页面矩形的 BackgroundImage 的属性，因为所有的渲染器都支持它。按照下面的说明更改所有 Page_矩形的属性：

- Source = "Embedded";
- Value = "=IIF(Parameters!ForPrint.Value = false, "c2_Page__96", "")" ,n 是指定矩形所在的实际页码
- BackgroundRepeat = "Clip"

这样，我们只有当页面是在屏幕上显示时使用低分辨率的背景图像和并在打印时隐藏它（根据 ForPrint 参数值进行判断）。



第 9 步 — 添加高分辨率的背景图像

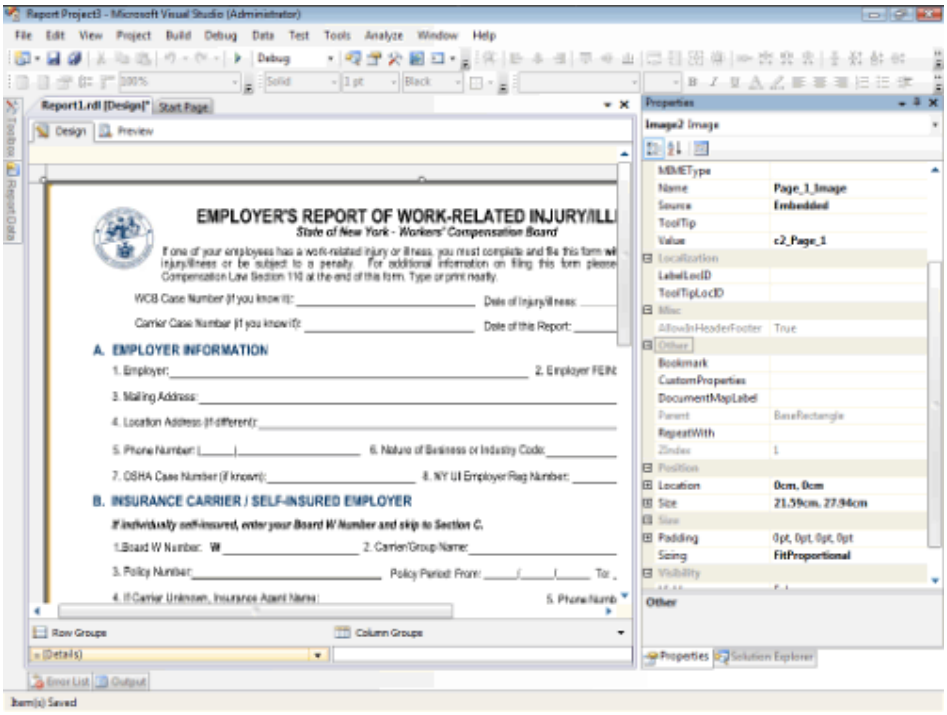
对于打印操作，我们将使用高分辨率背景图像，并以 Image 报表控件作为每个页面的矩形的背景（矩形的 BackgroundImage 属性不支持图 像拉伸，所以我们需要为每个报表页添加额外的 Image 控件）。HTML 渲染器不支持这种高分辨率图像的背景，而且它将消耗大量系统资源，所以我们将只用 在打印和 PDF 导出操作上。

对于每个报告页面添加一个 Image 控件到 BaseRectangle，将其命名为：Page__Image）。将其大小定义为原始报表页的 大小，位置为 x = 0，y=<(之前所有页面高度之总和)，例如：（0in; 0in），(0in; 11in)，(0in; 22in)，因此，它的尺寸和位置与在同页上的矩形大小完全一致。大小和位置数值的单位是英寸或者毫米，而不是点。

更改 Page__Image 的属性如下：

- Source = "Embedded"
- Value = "c2_Page_", < n> is appropriate page number
- Sizing = "FitProportional"
- Hidden = "= IIF(Parameters!ForPrint.Value = true, false, true)"
- 在每一个图片上应用工具栏上的“Send To Back”按钮（右键单击图片->Layout->Send To Back，或打开 Layout 面板：View->Toolbars->Layout）

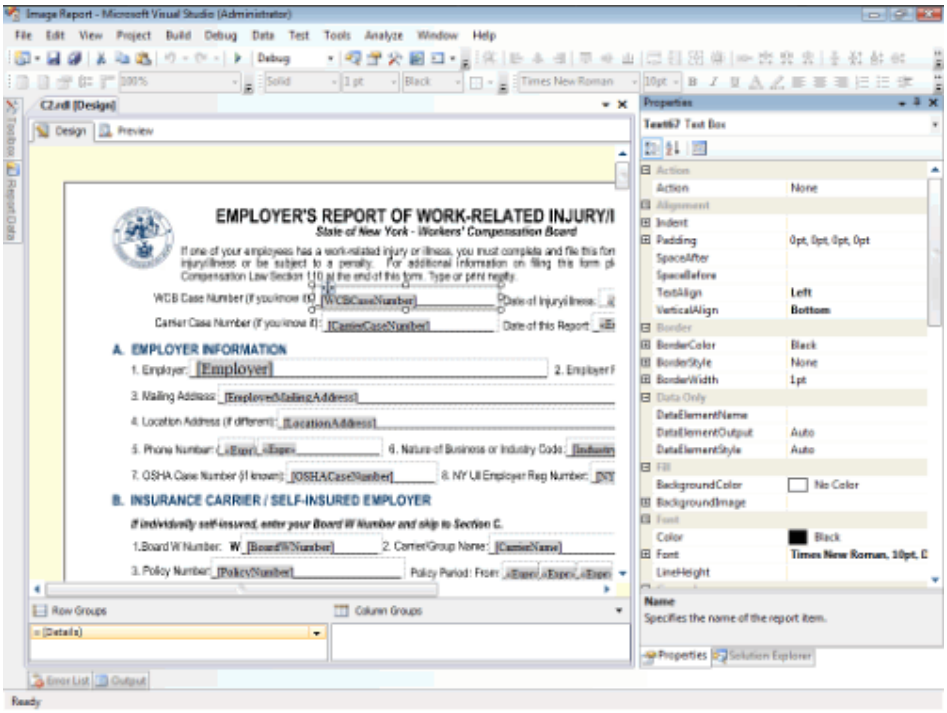
经过这样的设置，Image 控件只在当 ForPrint 参数值为 True 时可见。



第 10 步 — 添加文本框

各种报表数据都可以很容易地通过 Text 报表组件来展现。从“Report Data”标签中将“字段”拖到选定的 Page_rectanglle 中（不要忘记检查它的 Parent 属性！）。你可以指定文本框的位置，大小和字体， 以符合原始报告的外观。对每个数据字段重复此操作。

请务必将 CanGrow 属性设置为 false（默认为 true），并确保 CanShrink 属性值为 false。

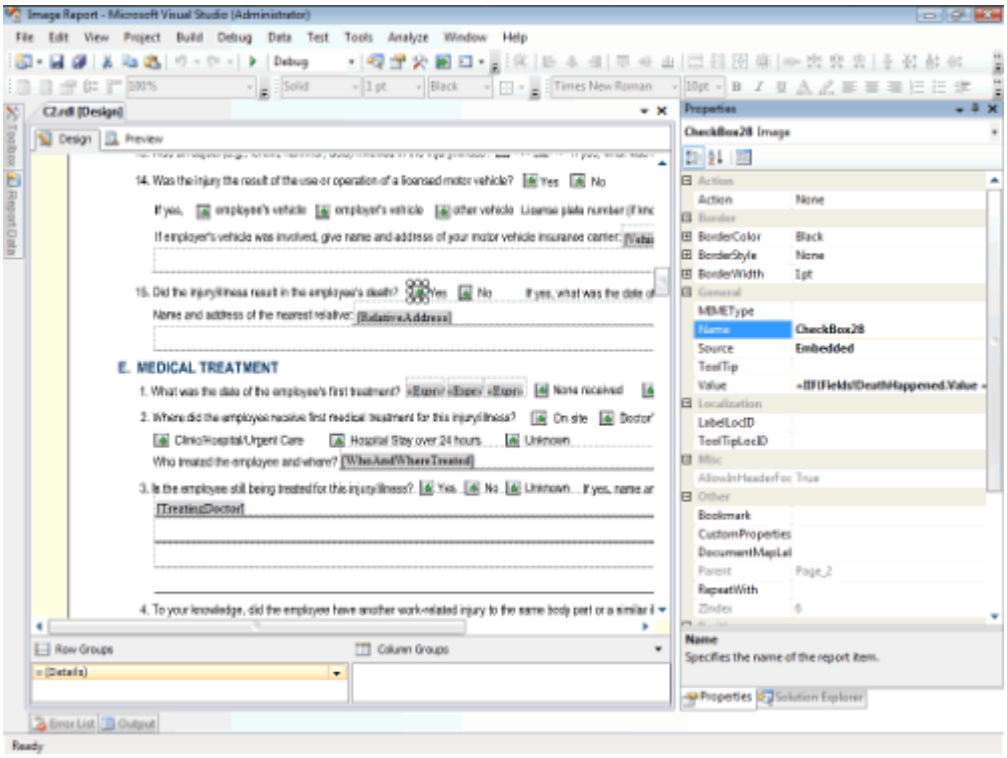


第 11 步 — 添加复选框项

为了达到原始 PDF 表格中的复选框的效果，基于该字段的数据逻辑（如：性别=“男”），我们将设置一个小的图片的 visible 属性（置为“checked”）使其可见。

要创建一个复选框添加一个 Image 控件，设置好它的位置，并调整其大小。更改其属性为：

- Source = "Embedded"
- Value = "= IIF(= true, "Check", "")，为这个域使用一些逻辑
- Sizing = "FitProportional"

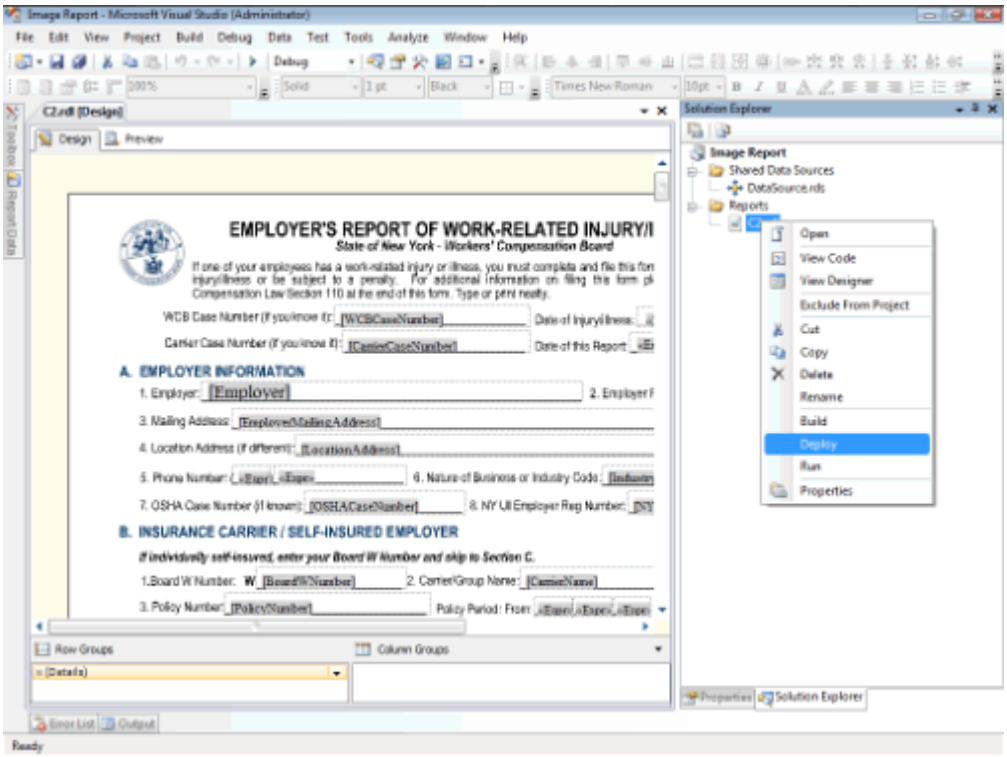


第 12 步 — 部署

到此，该报表已经可以用于在报表服务器上部署。设置“Project Deployment > properties”：

- TargetDataSourceFolder = "Data Sources"
- TargetReportFolder = "FixedLayoutReport"
- TargetServerURL = http://localhost/ReportServer（或者你自己的报表服务器地址）

之后，就可以部署它：



在接下来的部分中，我们将展示如何在不同场景中使用已布署好的报表。

如果您还没有从零开始创建报表，而是希望直接使用示例 ImageReportC2 中的 C - 2 报表，那么，您需要在 Visual Studio 2008 中打开此报表，执行第 12 步将其部署到报表服务器。

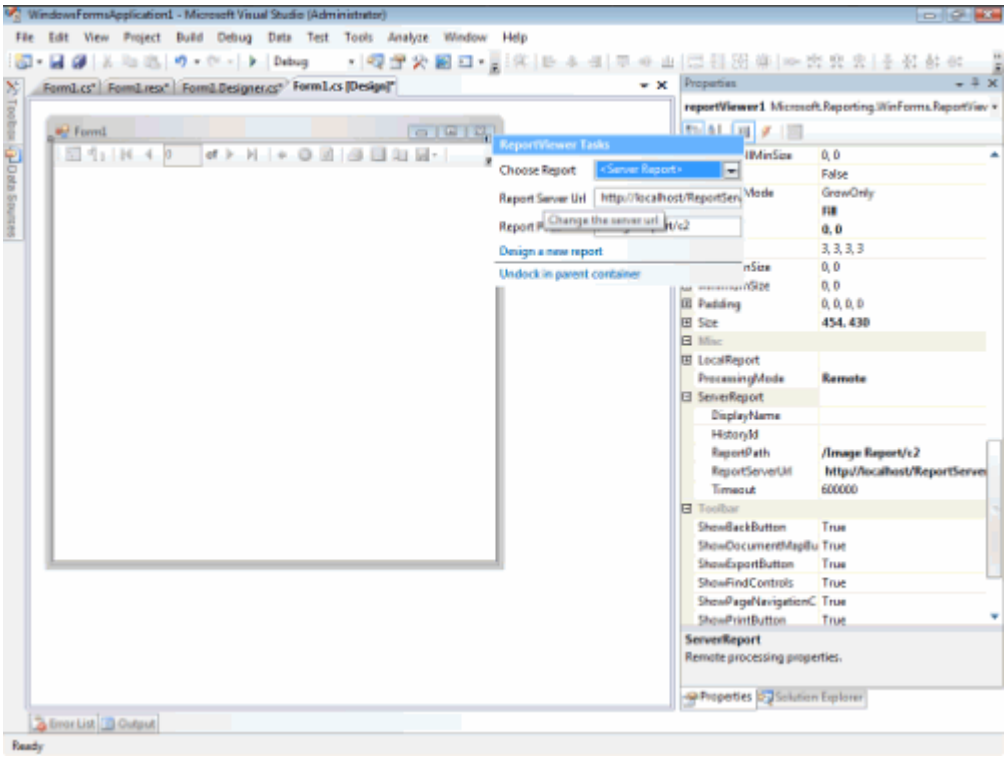
在使用此报表运行下一个示例之前，请确保该报表的数据源正确的链接到了已经安装有示例数据库的 SQL Server 2008 实例上。确认这一点，只要查看 DataSource.rds 的连接属性，该数据库的名称应该是 C2，而且，与 SQL Server 的连接认证测试必须是成功的。

请注意，当报表需要由许多数据行（超过 10,000 行以上）生成时，报表服务器上的资源消耗将大幅增长。推荐使用 64 位系统制作这类报告。

创建一个可以在 Windows 应用程序中浏览、打印和导出固定布局报表的通用解决方案

下一步，是创建一个通用应用，允许 Windows 应用中程序中浏览、打印和导出固定布局的报表。在本例中，我们将使用 WinForms ReportViewer 控件和报表服务器远程处理，以及可自动处理 ForPrint 参数的自定义代码。

要实现这个应用，你可以创建一个新的 WinForms 项目，将工具箱中的 ReportViewer 控件到窗体上，按照下图设置 ReportServerURL 和 ReportPath 属性：



然后，你需要实现几个事件处理程序来正确管理 ForPrint 参数。在 ReportExport 和 Print 事件处理器中，该参数必须设置为 true，见下例：

```
private void rvMain_Print(object sender, CancelEventArgs e)
{
    //Reset report to high-resolution mode for printing
    rvMain.ServerReport.SetParameters(new ReportParameter[]
   { new ReportParameter("ForPrint", "True") });
}
```

在 RenderingBegin 和 FormActived 事件处理器中，该参数需要被置回到 False，见下例：

```
private void rvMain_RenderingBegin(object sender, CancelEventArgs e)
{
    //Reset report to low-resolution mode for screen rendering
    if (rvMain.ServerReport.GetParameters()[0].Values[0] == "True")
    {
   rvMain.ServerReport.SetParameters(new ReportParameter[]
{ new ReportParameter("ForPrint", "False") });
    }
}
```

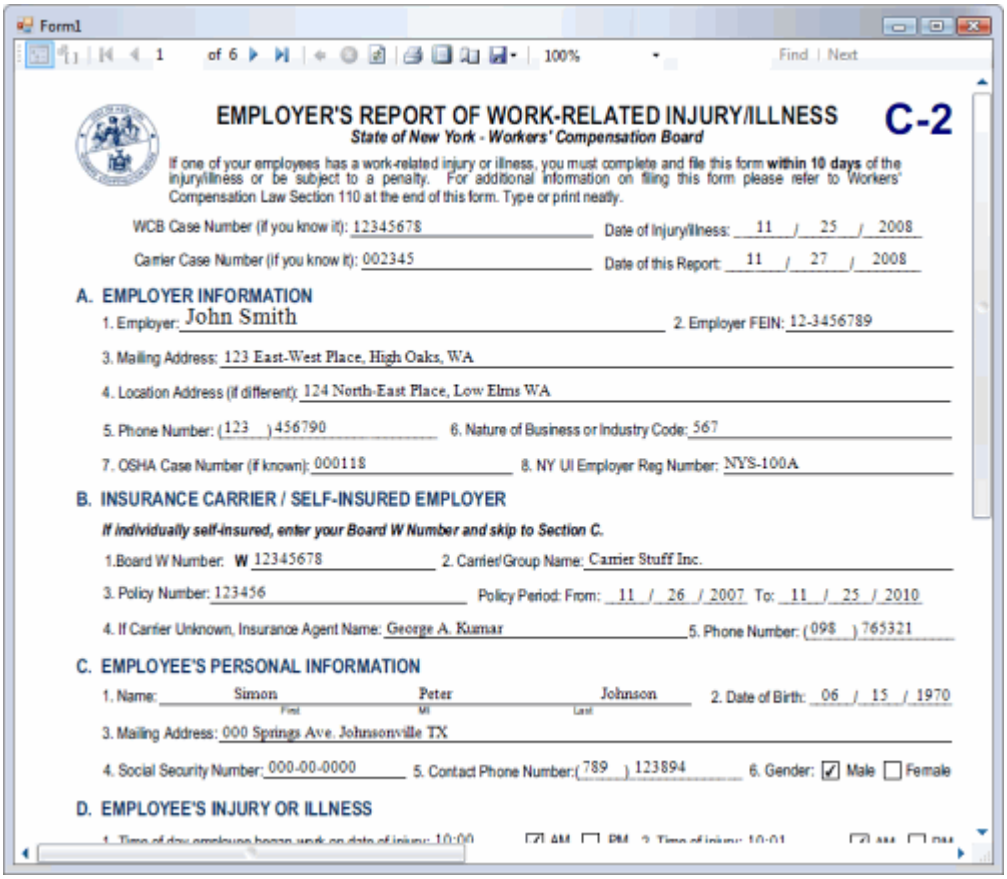
要查看示例应用程序，在 Visual Studio 2008 中 打开到 WinFormsViewer 目录下的 FixedLayoutReportWinFormsViewer.sln 解决方案。由于该解决方案引用了示 例 C - 2 报告，它（或其它被引用的报表）必须是已经被部署到服务器上的（上文第 12 步说明了如何进行部署 ）。

在运行示例应用程序之前，在 app.config 文件中指定报表的实际位置。你可以编辑 ReportWinForms.Properties.Settings 成员框内的两个设置：

- “ReportServerURL”一这里填写部署了 C-2 报表的报表服务器地址
- “ReportPath”一报表名称。本例中，它的值是/FixedLayoutReports/c2

如果你不能确定报表服务器的 URL ,可以在报表服务配置管理器中的“Web Services URL”页中找到它。

图 xx 显示了一个由示例应用基于 C2 表格图片生成的图表数据叠加 C-2 表单



创建一个基于网页的用于浏览，打印和导出固定布局报表的通用解决方案

在实现说明环节中已经提到，基于 Web 解决方案的关键部分是使用一个自定义 aspx 页面来替代缺省的 ReportViewer.aspx 页面执行其角色，但它被专门设计用于解决默认的

ReportViewer 处理固定布局报表时所出现的问题。我们的自定义页面包括：

- 一个单独的标准 Image WebControl
- 几个用来在报表页面间导航的控件（第一页，上一页，下一页和最后一页按钮，当前页和最后一页文本框）
- 下拉列表控件，它包含了可用的导出格式：MHTML，PDF 和 TIFF。这里我们并没有考虑 Word 和 Excel 格式因为它们在固定格式导出应用中并不普遍。
- 导出按钮，用来执行导出到指定格式。
- 打印按钮，把报表页导出到 PDF 格式，然后调用 Adobe PDF 进行打印预览。

“Web Viewer”目录包含了我们基于 Web 的解决方案。要查看请在 Visual Studio 2008 中打开“FixedLayoutReportWebViewer.sln”。在运行示例程序之前，请打开 Web.config 文件，在 configuration/appSettings 配置区指定报表服务器的实际 URL。

报表名和路径应该作为参数附加到 URL 串中。示例解决方案中报表名应该是“/FixedLayoutReports/C2”。要设置该项，打开 Property Pages>Start options>Specific Page，把已对斜线等字符进行过编码的报表 URL 填写到报表路径中：

```
FixedLayoutReportViewer.aspx?%2f FixedLayoutReports%2fC2
```

或者，你可以在浏览器地址栏中指定该地址。要注意的，本解决方案依赖于报表服务器 URL 访问机制，尤其是当命令和格式都是在 URL 参数中指定时更要注意。例如，要在以 JPEG 格式显示一个指定的页面，我们使用下面的 URL 参数：

```
(rs:Command=Render&rs:Format=IMAGE&rc:OutputFormat=JPEG&rc:StartPage=).
```

如前所述，不使用 HTML 渲染器，而是通过将整个报表中的每一页都以一个单独的 jpg 图片的方式来呈现，这解决了失真问题（上文提到的挑战 C）。导 出功能是通过在报表 URL 中包含导出命令（"rs:Command=Export"）和格式参数（e.g. "rs:Format=MHTML"）实现的。对于 PDF 输出高分辨率的图像我们通过设置报表 URL 的 ForPrint 参数值为 True 实现。要打印文件，我们执行交互式 PDF 导出，一段 JavaScript 脚本将调用 PDF 对象的 PrintWithDialog()方法实现打印。

结论

读过本文，你知道了如何使用 SQL Server 2008 Reporting Service 来创建符合官方标准格式的

精确报表。本方案通过使用符合正确用途的图片作为背景图像来创建固定格式的表格，最大限度地减少了所需的工作量。 我们还演示了哪些报表可以在基于 Windows 应用程序和基于 Web 应用程序的解决方案中进行处理。

原文链接：http://www.infoq.com/cn/articles/Gogolowicz-Swanson-SSRS

相关内容：

- 微软发布工具帮助用户从SQL Server快速迁移到SQL Azure
- SQL Server的未来之路
- 微软进入主数据管理市场
- 新版SQL Enlight T-SQL分析器发布
- 关系型的云呼之欲出

推荐文章 | Articles

使用jBpm支持高级用户交互模式

作者 Boris Lublinsky 译者 胡键

许多通用业务流程都包含人类参与者。人类活动，从简单场景（如人工批准）到复杂场景（涉及复杂的数据输入），在流程实现中引入了新的方面，如人类交互模式。人类交互模式的一个典型集合[1]包括：

- **四眼原则（The 4-eyes principle）**，通常又被称为“职责分离”，它是决策由多人彼此独立作出时的一个常见场景。在很多情况下，很容易就能得到另一个观点/签名。
- **任命（Nomination）**是指上级根据团队成员的任务安排、工作负荷或经验人工地将任务分配给他的情形。
- 任务通常被建模来表达一种预期：它们将在确定时间段内完成。如果任务没有按预期地进展，就需要一种**上报（escalation）**机制。两种典型的上报实现是：重新分配任务，并常常附带一个上报已经发生的通知；或任务未按时完成的通知（通常发给经理）。
- **链状执行（Chained execution）**是一个流程（片断），其中的一系列步骤是由同一个人来完成。

在本文中，我将讨论如何使用 JBoss jBPM 来实现这些高级的交互模式。

jBPM中的任务管理

jBPM的一个核心功能[2]是为人类管理任务和任务列表。jBPM允许将任务和任务节点作为整个流程设计的一部分使用。

任务一般在jBPM中定义成任务节点。单个任务节点可以包含一个或多个任务。包含任务节点的jBPM流程的一个公共行为就是等待任务节点中的全部任务完成，然后继续执行。某个任务可被分配[3]给个人、用户组或泳道：

- 假如任务被分配给某个特定用户，那么就只有这个使用者可以执行它。
- 假如任务被分配给某个用户组，那么这个组内的任何参与者都能执行这个任务。jBPM使用的是参与者池（pooled actors）符号（它可以包含组名、组名列表和参与者个人列表等），而不是组ID。如果用户开始执行在他们组任务列表中的任务，最终可能会引起冲突[4]——可能有多人开始执行相同的任务。为了避免这种情况，在开始执行任务之前，用户应该将任务从组任务列表移动到他们自己的任务列表中。
- 泳道代表一个流程角色，它通常被分配给一个用户组。它是一种指定流程中的多个任务要由同一参与者完成的机制[5]。因此，在第一个任务被分配给某个泳道之后，流程就会记住所有在相同泳道内的后续任务都将由同一参与者完成。

jBPM提供了两种定义任务分配的基本方法：作为流程定义的一部分或通过编程实现。如果是作为流程定义的一部分，分配可以通过指定具体用户、用户组 或泳道 完成。此外，可以使用表达式根据流程变量动态确定某个具体用户。完整的编程实现是基于分配处理器（assignment handler）的[6]，它允许任务根据任意的计算规则去查找用户ID。

流程定义描述流程实例的方式类似任务描述任务实例的方式。当流程执行时，一个流程实例——流程的运行时表示——就会被创建。类似，一个任务实例——任务的运行时表示——就会被创建。根据任务定义，任务实例被分配给一个参与者/参与者组。

任务实例的一个作用就是支持用户交互——把数据显示给用户并从用户那里收集数据。一个jBPM任务实例拥有访问流程（令牌）变量[7]的全部权限，而且还可以有自己的变量。任务能够拥有自己的变量对于以下场景非常有用：

- 在任务实例中创建流程变量的副本，这样对任务实例变量的即时更新只有在该任务完成且这些副本被提交给流程变量时才会影响流程变量。
- 创建更好支持用户活动的“派生（计算）”变量。

任务自己的变量在jBPM中是通过任务控制器处理器（task controller handler）支持的[8]，它可以在任务实例创建时生成任务实例数据（从流程数据），并在任务实例完成时将任务实例数据提交给流程变量。

实现四眼原则

我们上面已经说过，实现四眼原则意味着要允许多人同时干一个活。它的实现有以下几种可能方法：

- 在任务外解决：需要大量时间的任务并行循环（parallel looping）[9] 。
- 使用动作处理器（Action handler）：附加到任务节点的进入事件（enter event），基于流程实例变量创建多个节点实例[10]。
- 在任务内解决：引入“任务接受量（task take）”（类似 jPDL 4）并允许某个任务实例可被接受多次。

根据jBPM最佳实践[11] ——“扩展jBPM API而不是去搞复杂的流程建模”[12] ，我决定采用任务内解决的方法。这就要求修改jBPM提供的任务和任务实例类。

扩展 Task 类

jBPM 任务的定义被包含在 org.jbpm.taskmgmt.def.Task 类中。为了支持四眼原则，我们需要给类增加以下的字段/方法（清单 1）：

```
protected int numSignatures = 1;
public int getNumSignatures(){
   return numSignatures;
}
public void setNumSignatures(int numSignatures){
   this.numSignatures = numSignatures;
}
```

清单 1 给 Task 类增加字段和方法

这个新的参数允许指定任务完成所需的任务处理人数量。缺省值为 1，这意味着，只有 1 个用户应该/可以处理这个任务。

jBPM 使用 Hibernate 来向数据库保存和读取数据。为了让我们新加的变量持久化，我们需要更新 Task 类的 Hibernate 配置文件（Task.hbm.xml），它在 org.jbpm.taskmgmt.def 文件夹中，增加代码如下（清单 2）

```
<property name="numSignatures" column="NUMSIGNATURES_" />
```

清单 2 在 Task 映射文件中指定新增域

为了让我们新加的属性能被流程定义和数据库正确读取，我们需要修改 org.jbpm.jpdl.xml.JpdlXmlReader 类以正确地读取我们的新属性（清单 3）

```
String numSignatureText =
taskElement.attributeValue("numSignatures");
```

```
if (numSignatureText != null) {
   try{
      task.setNumSignatures(Integer.parseInt(numSignatureText));
   }
   catch(Exception e){}
}
```

清单 3 读取 numSignature 属性

最后，因为 JpdlXmlReader 根据模式来验证 XML，因此我们需要在 jpdl-3.2.xsd 中增加一个属性定义（清单 4）：

```
  <xs:element name="task">
    ……..
      <xs:attribute name="numSignatures" type="xs:string" />
```

清单 4 在 jpdl-3.2.xsd 中增加 numSignatures 属性

当完成这些工作，任务定义就被扩展可以使用 numSignatures 属性（清单 5）：

```
 <task name="task2" numSignatures = "2">
   <assignment pooled-actors="Peter, John"></assignment>
 </task>
```

清单 5 给任务定义增加 numSignatures 属性

扩展 TaskInstance 类

在扩展完任务类后，我们还需要创建一个自定义的任务实例类来跟踪分配给该任务实例[13]的参与者，并确保所有被分配的参与者完成类执行（清单 6）。

```
package com.navteq.jbpm.extensions;

import java.util.Date;
import java.util.LinkedList;
import java.util.List;

import org.jbpm.JbpmException;
import org.jbpm.taskmgmt.exe.TaskInstance;
```

```
public class AssignableTaskInstance extends TaskInstance {

   private static final long serialVersionUID = 1L;

   private List<Assignee> assignees = new LinkedList<Assignee>();

   private String getAssigneeIDs(){
      StringBuffer sb = new StringBuffer();
      boolean first = true;
      for(Assignee a : assignees){
         if(!first)
            sb.append(" ");
         else
            first = false;
         sb.append(a.getUserID());
      }
      return sb.toString();
   }

   public List<Assignee> getAssignees() {
      return assignees;
   }

   public void reserve(String userID) throws JbpmException{

      if(task == null)
         throw new JbpmException("can't reserve instance with no
task");

      // Duplicate assignment is ok
      for(Assignee a : assignees){
```

```
            if(userID.equals(a.getUserID()))
               return;
         }

         // Can we add one more guy?

         if(task.getNumSignatures() > assignees.size()){
            assignees.add(new Assignee(userID));
            return;
         }

         throw new JbpmException("task is already reserved by " +
getAssigneeIDs());

      }

      public void unreserve(String userID){

         for(Assignee a : assignees){
            if(userID.equals(a.getUserID())){
               assignees.remove(a);
               return;
            }
         }
      }

      private void completeTask(Assignee assignee, String transition){

         assignee.setEndDate(new Date());

         // Calculate completed assignments
```

```
      int completed = 0;
      for(Assignee a : assignees){
         if(a.getEndDate() != null)
            completed ++;
      }
      if(completed < task.getNumSignatures())
         return;
      if(transition == null)
         end();
      else
         end(transition);
   }

   public void complete(String userID, String transition) throws
JbpmException{

      if(task == null)
          throw new JbpmException("can't complete instance with no
task");

      // make sure it was reserved
      for(Assignee a : assignees){
         if(userID.equals(a.getUserID())){
            completeTask(a, transition);
            return;
         }
      }
       throw new JbpmException("task was not reserved by " + userID);
   }

   public boolean isCompleted(){
```

```
      return (end != null);

   }
}
```

清单 6 扩展 TaskInstance 类

这个实现扩展了 jBPM 提供的 TaskInstance 类，并跟踪完成该实例所需的参与者个数。它引入了几个新方法，允许参与者预留（reserve）/退还（unreserve）任务实例，以及让指定参与者完成任务执行。

清单 6 的实现依赖一个支持类 Assignee（清单 7）

```
package com.navteq.jbpm.extensions;

import java.io.Serializable;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Assignee implements Serializable{

   private static final long serialVersionUID = 1L;
   private static final DateFormat dateFormat = new
SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

   long id = 0;
   protected String startDate = null;
   protected String userID = null;
   protected String endDate = null;

   public Assignee(){}

   public Assignee(String uID){
```

```
        userID = uID;
         startDate = dateFormat.format(new Date());
    }

//////////////Setters and Getters ////////////////////

    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getStartDate() {
        return startDate;
    }
    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }
    public String getUserID() {
        return userID;
    }
    public void setUserID(String id) {
        userID = id;
    }
    public String getEndDate() {
        return endDate;
    }
    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
```

```
   public void setEndDate(Date endDate) {
      this.endDate = dateFormat.format(endDate);
   }

   public void setEndDate() {
      this.endDate = dateFormat.format(new Date());
   }

   public String toString(){

      StringBuffer bf = new StringBuffer();
      bf.append(" Assigned to ");
      bf.append(userID);
      bf.append(" at ");
      bf.append(startDate);
      bf.append(" completed at ");
      bf.append(endDate);
      return bf.toString();
   }
}
```

清单 7 Assignee 类

自定义的TaskInstance类和Assignee类都必须保存到数据库中。这意味着需要给这两个类实现Hibernate映射[14] （清单 8，9）：

```
<?xml version="1.0"?>
 <!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

  <hibernate-mapping auto-import="false" default-access="field">

    <subclass
namename="com.navteq.jbpm.extensions.AssignableTaskInstance"
        extends="org.jbpm.taskmgmt.exe.TaskInstance"
        discriminator-value="A">
    <list name="assignees" cascade="all" >
```

```
      <key column="TASKINSTANCE_" />
      <index column="TASKINSTANCEINDEX_"/>
      <one-to-many class="com.navteq.jbpm.extensions.Assignee" />
    </list>

   </subclass>
 </hibernate-mapping>
```

清单 8 自定义任务实例的 Hibernate 映射文件

```
<?xml version="1.0"?>
 <!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

  <hibernate-mapping auto-import="false" default-access="field">
   <class name="com.navteq.jbpm.extensions.Assignee"
         table="JBPM_ASSIGNEE">
    <cache usage="nonstrict-read-write"/>
    <id name="id" column="ID_"><generator class="native" /></id>
    <!-- Content -->
    <property name="startDate"     column="STARTDATE_" />
    <property name="userID"     column="USERID_" />
    <property name="endDate"       column="ENDDATE_" />
   </class>
 </hibernate-mapping>
```

清单 9 Assignee 类的 Hibernate 映射文件

要让 jBPM 能够使用我们的自定义任务实例实现，我们还需要提供一个自定义的任务实例工厂（清单 10）。

```
package com.navteq.jbpm.extensions;


import org.jbpm.graph.exe.ExecutionContext;

import org.jbpm.taskmgmt.TaskInstanceFactory;

import org.jbpm.taskmgmt.exe.TaskInstance;


public class AssignableTaskInstanceFactory implements
TaskInstanceFactory {


   private static final long serialVersionUID = 1L;


   @Override
```

```
   public TaskInstance createTaskInstance(ExecutionContext
executionContext) {


      return new AssignableTaskInstance();
   }


}
```

清单 10 自定义的任务实例工厂

最后，为了让 jBPM 运行时使用正确的任务实例工厂（清单 10），还必须创建一个新的 jBPM 配置（清单 11）。

```
<jbpm-configuration>

    <bean   name="jbpm.task.instance.factory"
class="com.navteq.jbpm.extensions.AssignableTaskInstanceFactory"
singleton="true"
/>

</jbpm-configuration>
```

清单 11 jBPM 配置

完成所有这些变更之后（清单 1-11），一个典型的任务处理显示如下：

```
List<String> actorIds = new LinkedList<String>();
actorIds.add("Peter");
List<TaskInstance> cTasks = jbpmContext.getGroupTaskList(actorIds)
TaskInstance cTask = cTasks.get(0);
AssignableTaskInstance aTask = (AssignableTaskInstance)cTask;
try{
   aTask.reserve("Peter");
   // Save
   jbpmContext.close();
}
catch(Exception e){
   System.out.println("Task " + cTask.getName() + " is already
reserved");
   e.printStackTrace();
```

```
}
```

清单 12 处理可分配任务实例

这里，在得到某个用户的任务实例并将其转变成可分配任务实例之后，我们将试着预留它[15]。一旦预留成功，我们将关闭jBPM运行时以提交事务。

实现任命

JBoss jBPM 可以非常轻易的实现手动将任务分配给特定用户。根据 jBPM 提供的简单 API，可以完成将任务实例从一个任务列表移动到另一个任务列表，因此给某个用户分配任务相当直接（清单 13）

```
List<String> actorIds = new LinkedList<String>();
actorIds.add("admins");
String actorID = "admin";
List<TaskInstance> cTasks = jbpmContext.getGroupTaskList(actorIds);
TaskInstance cTask = cTasks.get(0);
cTask.setPooledActors((Set)null);
cTask.setActorId(actorID);
```

清单 13 将任务重新分配给指定用户

jBPM 提供了 2 类不同的 API 来设置参与者池：一类接收字符串 id 数组，另一类则接收 id 集合。如果要清空一个池，就要使用那个接收集合的 API（传入一个 null 集合）。

实现上报

前面已经说过，上报一般被实现为任务的重新分配，并常常附带一个上报已发生的通知；或是实现成一个任务未及时完成的通知。

实现为重新分配的上报

尽管 jBPM 不直接支持上报，但它提供了 2 个基本的机制：超时和重新分配（参见上节）。粗一看，实现上报只需将这二者结合即可，但是仔细一想还是存在一些困难：

- jBPM实现中的关系并不总是双向的。如，从一个任务节点我们可以找到所有这个节点定义的任务，但是从一个任务，并没有API可以完成找到包含它的任务节点的工作[16]；由某

个任务实例，你可以得到一个任务，但是没有由某个任务得到所有实例的API，诸如此类。

- 超时不是发生在任务自身，而是发生在任务节点上。由于某个节点可以关联多个任务，并且 jBPM 关系实现并不是双向的（见上），因此要跟踪当前任务实例就需要其他的支持手段。

以重新分配实现的上报的整个实现[17]涉及 3 个处理器：

- 负责给任务分配参与者的分配处理器。这个处理器跟踪它是一个首次任务调用还是一个上报任务调用。清单 14 给出了一个分配处理器的例子。

```
package com.sample.action;


import org.jbpm.graph.def.Node;
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.taskmgmt.def.AssignmentHandler;
import org.jbpm.taskmgmt.exe.Assignable;


public class EscalationAssignmentHandler implements
AssignmentHandler {


   private static final long serialVersionUID = 1L;


   @Override
   public void assign(Assignable assignable, ExecutionContext
context)
   throws Exception {


      Node task = context.getToken().getNode();
      if(task != null){
         String tName = task.getName();


         String vName = tName + "escLevel";
         Long escLevel = (Long)context.getVariable(vName);
```

```
            if(escLevel == null){
               // First time through
               assignable.setActorId("admin");
            }
            else{
               // Escalate
               assignable.setActorId("bob");
            }
         }
      }
}
```

清单 14 分配处理器示例

这里我们尝试得到一个包含了给定任务上报次数的流程变量。如果变量未定义，则就分配“admin”为任务拥有者，否则任务就被分配给“bob”。在这个处理器中可以使用任何其他的分配策略。

- 任务实例创建动作处理器（清单 15），它保存流程实例上下文的任务实例 id

```
package com.sample.action;
import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.def.Node;
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.taskmgmt.exe.TaskInstance;

public class TaskCreationActionHandler implements ActionHandler {
   private static final long serialVersionUID = 1L;
   @Override
   public void execute(ExecutionContext context) throws Exception {
      Node task = context.getToken().getNode();
      TaskInstance current = context.getTaskInstance();
      if((task == null) || (current == null))
         return;
      String tName = task.getName();
```

```
        String iName = tName + "instance";
        context.setVariable(iName, new Long(current.getId()));
    }
}
```

清单 15 任务实例创建处理器

- 任务节点计时器触发调用的超时处理器（清单 16）。

```
package com.sample.action;

import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.def.GraphElement;
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.taskmgmt.exe.TaskInstance;

public class EscalationActionHandler implements ActionHandler {
    private static final long serialVersionUID = 1L;
    private String escalation;
    @Override
    public void execute(ExecutionContext context) throws Exception {
        GraphElement task = context.getTimer().getGraphElement();
        if(task == null)
            return;
        String tName = task.getName();
        String vName = tName + "escLevel";
        long escLevel = (long)context.getVariable(vName);
        if(escLevel == null)
            escLevel = new long(1);
        else
            escLevel += 1;
        context.setVariable(vName, escLevel);
        String iName = tName + "instance";
```

```
      long taskInstanceId = (long)context.getVariable(iName);
      TaskInstance current =
context.getJbpmContext().getTaskInstance(taskInstanceId);
      if(current != null){
         current.end(escalation);
      }
   }
}
```

清单 16 超时处理器

这个处理器首先记录上报计数器，接着完成此节点关联的任务实例。任务实例的完成伴随有一个变迁（一般是回到任务节点）。

使用以上描述的处理器实现上报的简单流程例子显示在清单 17 中。

```
<?xml version="1.0" encoding="UTF-8"?>
 <process-definition
   xmlns="urn:jbpm.org:jpdl-3.2"
   name="escalationHumanTaskTest">
    <start-state name="start">
      <transition to="customTask"></transition>
   </start-state>
    <task-node name="customTask">
      <task name="task2">
         <assignment
class="com.sample.action.EscalationAssignmentHandler"><
/assignment>
      </task>
      <event type="task-create">
         <action name="Instance Tracking"
class="com.sample.action.TaskCreationActionHandler"></action>
      </event>
      <timer duedate="10 second" name="Escalation timeout">
         <action
class="com.sample.action.EscalationActionHandler">
            <escalation>
               escalation
            </escalation>
         </action>
      </timer>

      <transition to="end" name="to end"></transition>
      <transition to="customTask" name="escalation"></transition>
   </task-node>
```

```
  <end-state name="end"></end-state>
</process-definition>
```

清单 17 简单流程的上报

实现成通知的上报

jBPM为邮件传递提供了强大支持[18]，这使得实现成通知的上报变得极其简单。邮件传递可由给节点附加定时器，然后触发，它使用已经写好的邮件动作来完成通知传递。

实现链状执行

链状执行直接由 jBPM 泳道支持，并不需要额外的开发。

总结

不管我们在自动化方面投入多少努力 ,面对复杂的业务流程 ,总免不了要有人工介入的可能。在这篇文章中，我给出了一系列已建立的高级人工交互模式，并展示了用 jBPM 完成它是多么轻而易举。

[1] WS-BPEL Extension for People - BPEL4People

[2] 本文依据jBPM 3 完成。jBPM 4 对任务支持引入了一些扩展。

[3] 任务定义规定了分配定义。实际的分配发生在任务实例创建的时候。参见以下的任务实例。

[4] 这一问题在jBPM 4 中有所缓解，它引入了任务完成所需的任务接受API。

[5] 参见上面的链状执行模式。

[6] jBPM任务节点开发

[7] 使用JBoss / JBPM编排长时间运行的活动

[8]不象流程变量，jBPM任务控制器处理器变量是保存在内存中的。因此，如果必须允许多次访问一个支持保存中间执行结果的任务实例 ,使用任务控制器处理器可能是个错误的解决方案。

[9]参见InfoQ文章 for implementation of parallel loop.

[10]这要求给任务节点增加create-tasks="false"属性。

[11] JBPM最佳实践

[12]严格说来，使用任务处理器方法将可能实现我的需求且代码更少，但是这样它就与上报实现（见下）不兼容。此外，我还想展示如何修改缺省的任务和任务实例实现。

[13]类似jPDL 4 实现。

[14]在给自定义任务实例创建Hibernate映射时，将映射实现成缺省类接口的子类要简单得多。现有jBPM Hibernate映射严重依赖任务实例映射，因此从标准实现派生将使引入新任务实例的改动最小。

[15]冲突条件仍然可能存在，但是已经最小了。

[16]这可以使用Hibernate Queries完成，但没有可供直接使用的API。

[17]这个实现假定某个任务节点只包含一个任务，该任务只创建一个任务实例。

[18] Jbpm邮件传递

原文链接：http://infoq.com/cn/articles/jBPM-user-interaction-patterns

相关内容：

- TorqueBox：JVM上的Rails企业级解决方案
- Gavin King谈JSR-299 和Weld 1.0 对Java EE与JBoss的影响
- 使用JBoss/jBPM编排长运行活动
- Bill Burke 谈 REST-*、SOA/ROA 和 REST
- REST-*组织

推荐文章 | Articles

AMQP和RabbitMQ入门

作者 Joern Barthel 译者 杨晨

准备开始

高级消息队列协议（AMQP[1]）是一个异步消息传递所使用的应用层协议规范。作为线路层协议，而不是API（例如JMS[2]），AMQP客户端能够无视消息的来源任意发送和接受信息。现在，已经有相当一部分不同平台的服务器[3]和客户端可以投入使用[4]。

AMQP 的原始用途只是为金融界提供一个可以彼此协作的消息协议，而现在的目标则是为通用消息队列架构提供通用构建工具。因此，面向消息的中间件 （MOM）系统，例如发布/订阅队列，没有作为基本元素实现。反而通过发送简化的 AMQ 实体，用户被赋予了构建例如这些实体的能力。这些实体也是规范的一 部分，形成了在线路层协议顶端的一个层级：AMQP 模型。这个模型统一了消息模式，诸如之前提到的发布/订阅，队列，事务以及流数据，并且添加了额外的特 性，例如更易于扩展，基于内容的路由。

本文中区别发布/订阅是为了将生产者和消费者拆分开来：生产者无需知道消费者按照什么标准接受消息。队列是一个先入先出的数据结构。路由封装了消息队列中的消息的相关信息，这些信息决定了消息在异步消息系统中的最终展现形式。

在这里，我尝试解释一下这个模型的一些概念，Aman Gupta使用Ruby[5]实现了AMQP模型[6]。它使用的是一种事件驱动架构（基于EventMachine[7]），在阅读和使用的时候都会让人觉得有些不太熟悉。但是API的设计表明了在AMQ模型实体之间通信的方式是非常简单的，因此，即便开发者对Ruby并不熟悉，他同样也可以得到收获。

应该注意到，至少有三个或者更多的Ruby客户端[8, 9, 10]可供选择。其中的一个客户端Carrot很明显使用了非事件驱动的同步Ruby架构，因此，这个客户端在使用事件驱动的Ruby API的时候，风格非常简洁。

本文中的AMQP服务器是使用Erlang[11]编写的RabbitMQ。它实现了AMQP规范 0-8 版的内容，并且将在近期实现 0-9-1 版的内容[12]。

在开始之前再交代一些东西：异步消息是一个非常普通并且广泛使用的技术，从例如 Skype 或者 XMPP/Jabber 这样各种各样的即时消息协议到古老的 email。但是，这些服务都有如下特征：

- 它们会在传输消息的时候或多或少加入一些随意的内容（例如一封 email 可能会包含一个文本和关于办公室笑话的 PPT）和一些比较正式的路由信息（例如 email 地址）。
- 它们都是异步的，也就是说它们将生产者和消费者区分开来，因此可能将消息加入队列（例如某人发给你一条消息，但是你不在线或者你的邮箱会收到一封 email）。
- 生产者和消费者是具有不同知识的不同角色。我不需要知道你的 IMAP 用户名和密码就能够给你发送 email。事实上，我甚至不需要知道你的 email 地址，是否是一个马甲或者“真实”地址。这个特性意味着生产者不能控制什么内容被阅读或者订阅了 - 就像我的 email 客户端会舍弃掉大多数主动发送给我的医药广告。

AMQP 是一个抽象的协议（也就是说它不负责处理具体的数据），这个事实并不会将事情变得更复杂。反而，Internet 使得消息无处不在。人们通常使用它们和异步消息简单灵活地解决很多问题。而且构建 AMQ 中的异步消息架构模型最困难的地方在于上手的时候，一旦这些困难被克服，那么构建过程将变得简单。

你可能需要安装一些软件来自己动手实现这些例子。如果你已经在系统上安装了Ruby，那么只需要不到十分钟的设置时间。RabbitMQ网站也有许多信息[13]帮助你尽快开始。你只需做这些准备工作：

- Erlang/OTP包。下载地址是 http://erlang.org/download.html，安装说明在 http://www.erlang.org/doc/installation_guide/part_frame.html。
- RabbitMQ。下载地址是http://www.rabbitmq.com/download.html，安装说明在 http://www.rabbitmq.com/install.html。
- 一个Ruby虚拟机。如果在你的系统平台上没有可供选择的Ruby解释器，你可能需要下载 Ruby MRI VM。在http://www.ruby-lang.org/en/downloads/可以找到下载地址和安装说明。
- 两个 Ruby “gem”（已打包的库）。gem 工具应该会随着你的 Ruby 安装包一起分发。
    - 如果你需要全新安装或者不确定它是不是当前版本，那么你可以选择升级 gem 工具。

输入 gem update --system。在 BSD/UNIX 系统中，你可能需要有超级用户的权限才能运行此命令（以及后续指令）。

- 告诉gem在GitHub搜索包：gem sources -a http://gems.github.com。
- 安装 AMQPgem：gem install tmm1-amqp。这也会安装 event-machine gem。

现在你需要做的就是启动RabbitMQ服务器[14]。

AMQ模型

在 AMQ 规范中描述了一些实体。一个用来分辨这些实体的方法是检查它们是否由服务器管理员配置或者由客户端在运行的时候声明。

可配置的实体有：

- 消息协商器（Message Broker），它在 TCP/IP 等端口监听 AMQ 消息。
- 将消息协商数据划分到多个不同集合的虚拟主机，它很像 webserver 中的虚拟主机，例如 Apache 的 http 守护进程。
- 使用安全凭据连接到虚拟主机的用户。

```
1 require 'rubygems'
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 # connect to the rabbitmq demonstration broker server
(http://www.rabbitmq.com/examples.html#demoserver)
13
14 AMQP.start :host => 'dev.rabbitmq.com', :port => 5672, :user =>
```

```
'guest', :password => 'guest', :vhost
=> 'localhost'
15
16 event_loop.join
```

值得注意的是，规范中仅仅授予用户访问虚拟主机的权限，并没有采纳其他比这高级的访问控制措施，因此RabbitMQ并不支持这些高级访问控制措施。一个由厂商开发的解决方法15期望会加入到下个主要版本中。但是，这个功能16可以通过使用Mercurial代码库的默认branch17来实现，而且已经有一些RabbitMQ用户在使用了。

为了和协商器交流，一个客户端需要建立一个或者多个连接。这些连接只是限于连接用户和虚拟主机。客户端默认使用 guest/guest 访问权限和访问虚拟主机的根目录，这些默认实现也是 RabbitMQ 的默认安装选项。

在一个连接中，客户端声明了一个通道。这个通道是消息协商器的网络连接中的一个逻辑连接。这种多工机制是必要的，因为协议中的某些操作是需要这样的 通道。因此，通过单一连接到协商器的并发控制需要建立一个可靠的模型，这里可以使用通道池和串行访问或者例如线程本地通道这样的线程并发模型。在这个例子 中，Ruby API 对用户隐藏了通道管理这样的细节。

如果需要在一个通道上进行操作，那么客户端需要声明 AMQ 组件。声明组件是断言特定的组件存在于协商器中——如果不存在的话，那么在运行时创建。

这些组件包括：

- 交换器（Exchange），它是发送消息的实体。
- 队列（Queue），这是接收消息的实体。
- 绑定器（Bind），将交换器和队列连接起来，并且封装消息的路由信息。

所有这些组件的属性各不相同，但是只有交换器和队列同样被命名。客户端可以通过交换器的名字来发送消息，也可以通过队列的名字收取信息。因为AMQ 协议没有一个通用的标准方法来获得所有组件的名称，所以客户端对队列和交换器的访问被限制在仅能使用熟知的或者只有自己知道的名字（参见[18]了解这种访问控制的信息）。

绑定器没有名字，它们的生命期依赖于所紧密连接的交换器和队列。如果这两者任意一个被删除掉，那么绑定器便失效了。这就说明，若要知道交换器和队列的名字，还需要设置消息路由。

消息是一个不透明的数据包，这些包有如下性质：

- 元数据，例如内容的编码或者表明来源的字段。
- 标志位，标记消息投递时候的一些保障机制。
- 一个特殊的字段叫做 routing key。

2.1 接受和发送消息：交换器类型

发送消息是一个非常简单的过程。客户端声明一个它想要发送消息的目的交换器，然后将消息传递给交换器。

接受消息的最简单办法是设置一个订阅。客户端需要声明一个队列，并且使用一个绑定器将之前的交换器和队列绑定起来，这样的话，订阅就设置完毕。

```
1 require 'rubygems'
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue
13
14   exchange = MQ.fanout('my-fanout-exchange')
15   queue = MQ.queue('my-fanout-queue')
16
17   queue.bind(exchange).subscribe do |header, body|
18     yield header, body
19   end
20
```

```
21 end
22
23 def send_to_exchange(message)
24
25   exchange = MQ.fanout('my-fanout-exchange')
26   exchange.publish message
27
28 end
29
30 subscribe_to_queue do |header, body|
31   p "I received a message: #{body}"
32 end
33
34 send_to_exchange 'Hello'
35 send_to_exchange 'World'
36
37 event_loop.join
```

三个标准决定了一条消息是否真的被投递到了队列中：

1. 交换器的类型。在这个例子中类型是 fanout。
2. 消息的属性。在这个例子中，消息没有任何属性，只是有内容（首先是 Hello，然后 World）。
3. 给定的绑定器的唯一可选属性：键值。在这个例子中绑定器没有任何键值。

交换器的类型决定了它如何解释这个连接。我们的例子中，fanout 交换器不会解释任何东西：它只是将消息投递到所有绑定到它的队列中。

没有绑定器，哪怕是最简单的消息，交换器也不能将其投递到队列中，只能抛弃它。通过订阅一个队列，消费者能够从队列中获取消息，然后在使用过后将其从队列中删除。

下列交换器类型都在规范中被提及。随后我会由浅入深地介绍它们。

- direct 交换器将消息根据其 routing-key 属性投递到包含对应 key 属性的绑定器上。

```
1 require 'rubygems'
2 require 'mq'
```

```
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue(key)
13
14   exchange = MQ.direct('my-direct-exchange')
15   queue = MQ.queue('my-direct-queue')
16
17   queue.bind(exchange, :key => key).subscribe do |header, body|
18     yield header, body
19   end
20
21 end
22
23 def send_to_exchange(message, key)
24
25   exchange = MQ.direct('my-direct-exchange')
26   exchange.publish message, :routing_key => key
27
28 end
29
30 subscribe_to_queue('hello_world') do |header, body|
31   p "I received a message: #{body}"
32 end
33
```

```
34 send_to_exchange 'Hello', 'hello_world'
35 send_to_exchange 'Cruel', 'ignored'
36 send_to_exchange 'World', 'hello_world'
37
38 event_loop.join
```

- topic 交换器用过模式匹配分析消息的 routing-key 属性。它将 routing-key 和 binding-key 的字符串切分成单词。这些单词 之间用点隔开。它同样也会识别两个通配符：#匹配 0 个或者多个单词，*匹配一个单词。例如，binding key *.stock.#匹配 routing key usd.stcok 和 eur.stock.db，但是不匹配 stock.nasdaq。

```
1 require 'rubygems'
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue(key)
13
14   exchange = MQ.topic('my-topic-exchange')
15   queue = MQ.queue('my-topic-queue')
16
17   queue.bind(exchange, :key => key).subscribe do |header, body|
18     yield header, body
19   end
20
21 end
22
```

```
23 def send_to_exchange(message, key)
24
25   exchange = MQ.topic('my-topic-exchange')
26   exchange.publish message, :routing_key => key
27
28 end
29
30 subscribe_to_queue('hello.*.message.#') do |header, body|
31   p ”I received a message: #{body}”
32 end
33
34 send_to_exchange 'Hello', 'hello.world.message.example.in.ruby'
35 send_to_exchange 'Cruel', 'cruel.world.message'
36 send_to_exchange 'World', 'hello.world.message'
37
38 event_loop.join
```

- 在规范中还有其他的交换器被提及，例如 header 交换器（它根据应用程序消息的特定属性进行匹配，这些消息可能在 binding key 中标记为可选或者必选），failover 和 system 交换器。但是这些交换器现在在当前 RabbitMQ 版本中均未实现。

不同于队列的是，交换器有相应的类型，表明它们的投递方式（通常是在和绑定器协作的时候）。因为交换器是命名实体，所以声明一个已经存在的交换器，但是试图赋予不同类型是会导致错误。客户端需要删除这个已经存在的交换器，然后重新声明并且赋予新的类型。

交换器也有一些性质：

- 持久性：如果启用，交换器将会在协商器重启前都有效。
- 自动删除 :如果启用 ,那么交换器将会在其绑定的队列都被删除掉之后自动删除掉自身。
- 惰性：如果没有声明交换器，那么在执行到使用的时候会导致异常，并不会主动声明。

2.2 默认交换器和绑定器

AMQP 协商器都会对其支持的每种交换器类型（为每一个虚拟主机）声明一个实例。这些交换器的命名规则是 amq.前缀加上类型名。例如 amq.fanout。空的交换器名称等于 amq.direct。

对这个默认的 direct 交换器（也仅仅是对这个交换器），协商器将会声明一个绑定了系 统中所有队列的绑定器。

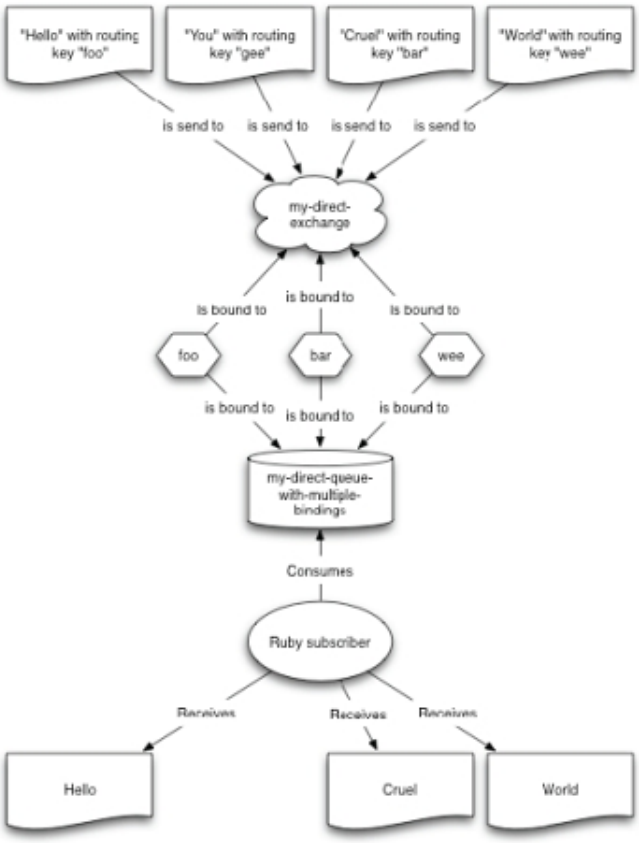
这个特点告诉我们，在系统中，任意队列都可以和默认的 direct 交换器绑定在一起，只要其 routing-key 等于队列名字。

2.3 队列属性和多绑定器

默认绑定器的行为揭示了多绑定器的存在 - 将一个或者多个队列和一个或者多个交换器绑定起来。这使得可以将发送到不同交换器的具有不同 routing key（或者其他属性）的消息发送到同一个队列中。

```
1 require 'rubygems'
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue(*keys)
13
14   exchange = MQ.direct('my-direct-exchange')
15   queue = MQ.queue('my-direct-queue-with-multiple-bindings')
16
17   bindings = keys.map do |key|
18     queue.bind(exchange, :key => key)
19   end
20
21   bindings.last.subscribe do |header, body|
22     yield header, body
```

```
23   end
24
25 end
26
27 def send_to_exchange(message, key)
28
29   exchange = MQ.direct('my-direct-exchange')
30   exchange.publish message, :routing_key => key
31
32 end
33
34 subscribe_to_queue('foo', 'bar', 'wee') do |header, body|
35   p "I received a message: #{body}"
36 end
```




```
37
```

```
38 send_to_exchange 'Hello', 'foo'
39 send_to_exchange 'You', 'gee'
40 send_to_exchange 'Cruel', 'bar'
41 send_to_exchange 'World', 'wee'
42
43 event_loop.join
```

虽然不能被命名，但是队列也有以下属性，这些属性和交换器所具有的属性类似。

- 持久性：如果启用，队列将会在协商器重启前都有效。
- 自动删除：如果启用，那么队列将会在所有的消费者停止使用之后自动删除掉自身。
- 惰性：如果没有声明队列，那么在执行到使用的时候会导致异常，并不会主动声明。
- 排他性：如果启用，队列只能被声明它的消费者使用。

这些性质可以用来创建例如排他和自删除的 transient 或者私有队列。这种队列将会在所有链接到它的客户端断开连接之后被自动删除掉 - 它们只是短暂地连接到协商器，但是可以用于实现例如 RPC 或者在 AMQ 上的对等通信。

AMQP上的RPC是这样的：RPC客户端声明一个回复队列，唯一命名（例如用UUID[19]），并且是自删除和排他的。然后它发送请求给一些交换器，在消息的reply-to字段中包含了之前声明的回复队列的名字。RPC服务器将会回答这些请求，使 用消息的reply-to作为routing key（之前提到过默认绑定器会绑定所有的队列到默认交换器）发送到默认交换器。注意仅仅是惯例而已。根据和RPC服务器的约定，它可以解释消息的任何属 性（甚至数据体）来决定回复给谁。

队列也可以是持久的，可共享，非自动删除以及非排他的。使用同一个队列的多个用户接收到的并不是发送到这个队列的消息的一份拷贝，而是这些用户共享这队列中的一份数据，然后在使用完之后删除掉。

2.4 消息投递的保障机制

消费者会显式或者隐式地通知消息的使用完毕。当隐式地通知的时候，消息被认为在投递之后便被消耗掉。否则客户端需要显式地发送一个验证信息。只有这个验证信息收到之后，消息才会被认为已经收到并且从队列中删除。如果没有收到，那么协商器会在通道[20]关闭之前尝试着重新投递消息。

```
1 require 'rubygems'
```

```
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(1) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue
13
14   exchange = MQ.fanout('my-fanout-exchange-with-acks')
15   queue = MQ.queue('my-fanout-queue-with-acks')
16
17   queue.bind(exchange).subscribe(:ack => true) do |header, body|
18     yield header, body
19     header.ack unless body == 'Cruel'
20   end
21
22 end
23
24 def send_to_exchange(message)
25
26   exchange = MQ.fanout('my-fanout-exchange-with-acks')
27   exchange.publish message
28
29 end
30
31 subscribe_to_queue do |header, body|
32   p "I received a message: #{body}"
```

```
33 end
34
35 send_to_exchange 'Hello'
36 send_to_exchange 'Cruel'
37 send_to_exchange 'World'
38
39 event_loop.join
40
41 __END__
42
43 First run:
44
45 "I received a message: Hello"
46 "I received a message: Cruel"
47 "I received a message: World"
48
49 Second run:
50
51 "I received a message: Cruel"
52 "I received a message: Hello"
53 "I received a message: Cruel"
54 "I received a message: World"
55
56 ... and so forth
```

消息生产者可以选择是否在消息被发送到交换器并且还未投递到队列（没有绑定器存在）和/或没有消费者能够立即处理的时候得到通知。通过设置消息的mandatory和/或immediate属性为真，这些投递保障机制的能力得到了强化。现在在本文例子中使用的Ruby AMQP API还不完全支持这些标志位。但是，在GitHub上已经有两个patch[21, 22]展示了完全支持之后的情况。

此外，一个生产者可以设置消息的 persistent 属性为真。这样一来，协商器将会尝试将这些消息存储在一个稳定的位置，直到协商器崩溃。当然，这些消息肯定不会被投递到非持久的队列中。

2.5 拥塞控制

在给出的例子中，对消息的使用永远看做是一个订阅。那么考虑到了拥塞控制吗？规范制定了QoS[23]特性，限制了通过一个通道发送到一个消费者的消息总量。很不幸的是，这个特性在当前RabbitMQ的版本中还不支持（计划在 1.6），但是在原则上是应该被AMQP API支持的。作为一个替代方案，客户端可以选择从队列中取出消息而不是通过订阅。当使用这种方法的时候，拥塞控制可以手动地实现。

```
1 require 'rubygems'
2 require 'mq'
3
4 event_loop = Thread.new do
5   EM.run do
6     EM.add_timer(5) do
7       EM.stop
8     end
9   end
10 end
11
12 def subscribe_to_queue
13
14   exchange = MQ.fanout('my-fanout-exchange')
15   queue = MQ.queue('my-fanout-queue')
16
17   queue.bind(exchange).pop do |header, body|
18     yield header, body
19   end
20
21   EM.add_periodic_timer(0.25) do
22     queue.pop
23   end
24
25 end
```

```
26
27 def send_to_exchange(message)
28
29   exchange = MQ.fanout('my-fanout-exchange')
30   exchange.publish message
31
32 end
33
34 received = 0
35
36 subscribe_to_queue do |header, body|
37   p "I received a message: #{body}"
38 end
39
40 send_to_exchange 'Hello'
41 send_to_exchange 'World'
42
43 event_loop.join
```

一个模型样例

想像一下你想创建一个普通的聊天应用，那么应该有以下几个基本特性：

- 聊天 - 两个用户应该可以相互发送消息。
- 一个好友系统 - 用户能够控制谁给他发送消息。

我们假设在协商器上有两种消费者：好友服务器和聊天客户端。

3.1 成为好友

为了成为Bob的好友，Alice首先得发送一个消息给fanout交换器iends，我们假设这个交换器是访问受限[24]的：普通用户不能够将队列绑定到它。在这个消息中，Alice表示想和Bob成为朋友。在协商器上有大量的聊天服务器，从绑定到friends交换器的一个单一持久队列中持续地取出消息。这个队列的名字是例如 friends.298F2DBC6865-4225-8A73-8FF6175D396D这样

的，这难以猜测的名字能够阻止聊天客户端直接取出信息 - 记住：不知道队列的名字，就不能设置订阅。

当一个聊天服务器收到 Alice 的消息（只有一个会得到这个消息，虽然它们都是从同一个队列中获取），决定这个请求是否有效，然后将其（也许是做过 一些调整或者参数化）发送到默认交换器（可以是直接的或者持久的）。它使用另外一个只有 Bob 知道的 routing key 来投递。当 Bob 上线的时候（或者一个服务器做了这件事），他会声明一个队列，这个队列的名字就是之前的 routing key（记住在虚拟主机上的默认绑定器是将所有的队列和默认交换器绑定在一起）。Bob 的聊天客户端现在询问 Bob 是否想和 Alice 成为朋友。在她的请求消息中，有一个特殊的属性叫做 reply-to - 这个属性包括了一个持久和排他的好友队列的名字，这个队列是 Alice 声明将用于和 Bob 的未来聊天用。如果 Bob 想和 Alice 成为朋友，他会使用这个 队列的名字作为 routing key，发送一个消息到默认交换器。他也会需要声明一个持久和排他的好友队列，将其名字设为 reply-to 的值。

例如：Alice 和 Bob 的好友队列的名字是 B5725C4A-6621463E-AAF1-8222AA3AD601。Bob 发送给 Alice 的消息的 routing-key 的值便是这个名字，也是 Alice 发送给 Bob 的消息中 reply-to 的值。因为好友队列是持久的，因此发送到消息在用户离线的时候也不会丢失。当用户上线之后，所有的在好友队列的消息将会发送到用户，然后才去获取新的消息。

当 Bob 不再想和 Alice 成为好友，他可以简单地删除掉为 Alice 声明的好友队列。在她使用 mandatory 标志位发送消息的时候，Alice 也会注意到 Bob 已经不再想是她的好友。因为交换器会将她的消息认为不可投递而返回。

仍未提及的事情

仍然有很多本文没有介绍的东西，例如事务语义，关于信息的重路由，header 交换器的规范以及不同 AMQP 规范之间的差异 - 尤其是在 1.0 版本之前的模型改变。为了简介起见，一个聊天的模型同样也被略过了。

这里也没有介绍了整个系统的管理，因为还不清楚 AMQP 和 RabbitMQ 将会走向何方。现在有一个课题，关于在保留的 amq 命名空间中可用的交换 器，它能获取协商器所有的日志信息。但是，能够列出现在已经声明的组件和已连接的用户的工具是用 rabbitmqctl 命令行接口而不是 AMQ 实体来实现 的。

```
1 require 'rubygems'
2 require 'mq'
```

```
3
4 PATH_TO_RABBITMQCTL = '/usr/local/sbin/rabbitmqctl'
5
6 event_loop = Thread.new { EM.run }
7
8 def subscribe_to_logger
9
10   random_name = (0...50).map{ ('a'..'z').to_a[rand(26)] }.join
11
12   exchange = MQ.topic('amq.rabbitmq.log')
13   queue = MQ.queue(random_name, :autodelete => true, :exclusive =>
true)
14   binding = queue.bind(exchange, :key => '#')
15
16   binding.subscribe do |header, body|
17     body.split("\n").each do |message|
18       yield header, message
19     end
20   end
21
22 end
23
24 def exchange_info(vhost = '/')
25   info :exchange, vhost, %w(name type durable auto_delete
arguments)
26 end
27
28 def queue_info(vhost = '/')
29   info :queue, vhost, %w(name durable auto_delete arguments node
messages_ready messages_unacknowledged
                          messages_uncommitted messages
acks_uncommitted consumers transactions memory)
30 end
```

```
31
32 def binding_info(vhost = '/')
33   info :binding, vhost
34 end
35
36 def connection_info
37   info :exchange, nil, %w(node address port peer_address peer_port
state channels user vhost timeout
                          frame_max recv_oct recv_cnt send_oct
send_cnt send_pend)
38 end
39
40 def info(about, vhost = nil, items = [])
41
42   column_length = 20
43
44   puts "#{about} info\n"
45
46   cmd = "#{PATH_TO_RABBITMQCTL} list_#{about}s"
47   cmd << " -p #{vhost}" if vhost
48   cmd << " #{items.join(' ')} 2>&1"
49
50   pipe = IO.popen(cmd)
51
52   pipe.readlines.map { |line| line.chomp.split("\t").map { |item|
item.ljust(column_length)[0,
                          column_length] } }.slice(1..-2).each do
|exchange|
53     print exchange.join(' ') + "\n"
54 end
55
56 end
57
```

```
58 subscribe_to_logger do |message|
59   p "RabbitMQ logger: #{message}"
60 end
61
62 %w(connection exchange queue binding).each do |method|
63   self.send "#{method}_info".to_sym
64 end
65
66 event_loop.join
```

必须提及的是，已经有一些使用AMQP（或者RabbitMQ）的分布式架构。这些架构（例如Nanite[25]或者Lizzy[26]）在AMQP的顶部引入了一些抽象层，这样简化了一些操作，例如cluster中在Ruby客户端之间工作的分配。

4.1 下一步该做什么？

要想使用本地的协商器来玩玩好友和邮件列表，第一步应该是学习有关AMQP和RabbitMQ的知识。不仅仅应该在RabbitMQ网站上阅读幻灯片和文章，还应该通过在IRC的#rabbitmq通道上和社区成员交流或者阅读关于RabbitMQ和/或AMQP的网志[27, 28]，例如LShift的博客。在Twitter上也可以通过#rabbitmq或#amqp这两个标签找到很多关于AMQP或者RabbitsMQ的内容[29, 30, 31, 32, 33, 34, 35]。

欢迎进入异步信息的世界，祝您玩得愉快！

原文链接： http://infoq.com/cn/articles/AMQP-RabbitMQ

相关内容：

- 使用Java构建高伸缩性组件
- 书摘和访谈：Open Source SOA
- Blaze Data Services还是LiveCycle Data Services？
- Anissa和Judy谈Glassfish的开发与测试
- OSGi原理与最佳实践（精选版）

推荐文章 | Articles

SOA治理的仙境

作者 Michael Poulin 译者 马国耀

引言——掉进兔子洞

Harry W.曾在论坛里说到“爱丽丝并非掉进兔子洞。而是，一只小白兔引起了她的兴趣，于是她愿意跟随兔子深入兔子洞”。没错，“仙境”的秘密就在这些文字的含义之中：Harry坚持认为爱丽丝“**愿意跟随**”兔子深入兔子洞，而爱丽丝则显然“发现自己已经掉入洞中”。SOA治理的“仙境”中也发生着相同的故事——它就像兔子洞一样，人们心想的是治理而谈论的却是管理或其他完全相反的事情……[译注：本文的爱丽丝，兔子洞源自经典童话《爱丽丝漫游奇境》]

我们通常认为管理包括治理。该理解没错：牛津字典和Merriam韦氏字典以及英语同义词字典都把“治理” 放在“管理”的近义词中。然而，反过来则不正确，治理不是管理。治理关心的是政策和控制规则，而管理的含义则是政策执行和控制。管理需要治理，否则就无从知道如何运行，执行以及保护。

我了解到在过去两年间，至少有 3 本书为SOA治理而著，infoQ上至少发表了 8 篇关于这个主题的文章。然而，我发现在绝大多数出版物中，SOA治理都直接关联政策加强或管理。迄今为止，只有OASIS SOA 参考架构（Reference Architecture），公众评阅草案 1中认识到“兔子洞”的影响，并将SOA环境中的治理和管理清晰地分离开。

你也许会问，为什么把治理从管理中分开那么重要？我的回答是：分离就能让我们更好地理解它们分别是什么，为什么需要那些工作，必须由谁来做并对治理 及其交 付件负责。再者，分离还有助于我们组织治理控制、合规报告以及通过治理政策对企业战略计划的敏捷性进行监督。图 1 描述了治理和管理的权力分离在 SOA 项目 和普通项目中都需要实行。因此，政策和规则的制定者不一定是其执行者——这是简单的权力平衡。换言之，管理本身必须要符合治理政策，然后它又能够在治理的 主题上加强政策的执行。

另一个问题是我们为什么需要治理 SOA 以及为什么我们以前没有谈过其他技术的治理，如 CORBA，和面向对象（OO）设计。很好，问题的后半部分并 不完全 正确——我记得有些人尝试过为 OO 应用开发建立政策和最佳实践，但这些话题从来没有走出过 IT 部门。而面向服务的架构为什么这么特别需要我们做出以前没有 做过的工作呢？

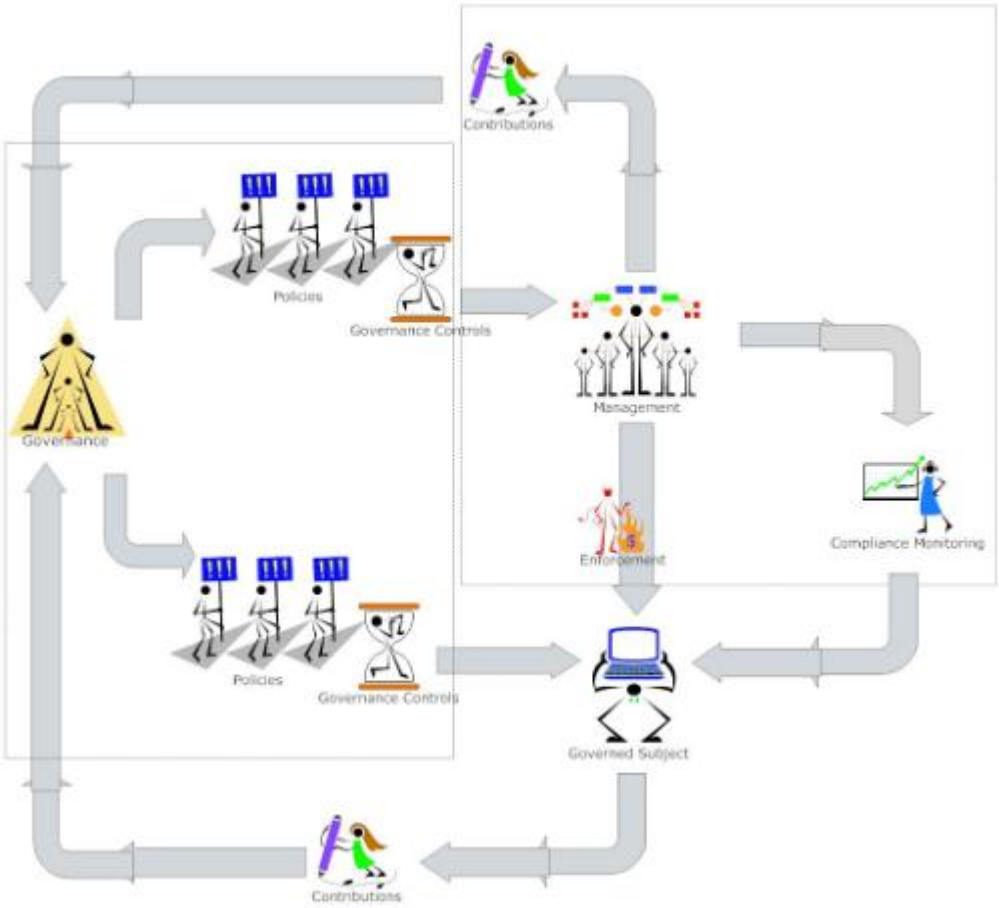


图 1

本文中，我将详细介绍治理和管理分离的结果，并尽力解释是哪些面向服务的具体方面让治理在 SOA 环境中受到如此重视。在展开细节之前，我希望每个人 都清楚 SOA 治理并不是关于政策开发和服务执行的。相反，它关注的是通过以下几个方面的最佳实践形成战略战术的业务方向及目标：a) 关系，b) 架构，c) 设计，d) 实施，e) 复合应用开发，i)敏捷的企业业务 j) 敏捷应对市场需求。所有的这些都是为实现企业的根本业务目标所需的创造性精神。

SOA治理定义漫游

在写这篇文章之时，至少已有两家标准化组织发布了他们对SOA治理的定义——OASIS和国际标准化组织（The Open Group）。后者（对SOA治理）有两个定义，其一在SOA治理框架标准

中 ，其二在TOGAF 9 中。OMG SOA SIG吸收了来自行业分析家，媒体成员和用户的 18 条SOA治理的定义。 我认为绝大多数定义都认同SOA治理的目的是通过政策和合规控制的手段尽可能地减少违背面向服务的概念和破坏业务目标的风险。不幸地是 ,由于治理和管理的混杂，我们的SOA治理弄得一团糟。

例如，标准化组织的SOA治理框架标准声称：“通常，治理指的是建立并加强人员和解决方案如何合作才能实现企业目标、强调实现控制、并区别治理和日常管理活动的差异。” 。我很想知道作者是怎么思考的，治理的加强不在**日常管理活动**中，难道是在假期活动中？

标准化组织的TOGAF 9中是这样描述治理的： “它要 较少地公然控制和严格参照规则，而更多地是引导资源的有效和公平的使用，从而确保企业战略目标的可持续性” 。所以，看起来标准化组织还没有拿定主意——到底治理是“实行控制”还是“较少地公然的控制和严格参照规则”？在“较少地……严格参照规则”的上下文中 ‘加强’的含义何解？

OASIS SOAR A 定义：“SOA 系统的治理需要决策者的能力……去设定参与者，服务以及他们之间的关系。他需要确保有效地描述和执行政策的能力。它还需要有效方法以保证 服务及参与者的过去和当前的效率”。这种说法要求确保政策加强的能力却没有描述 SOA 治理中执行本身的含义。

谈论到 SOA 治理和管理的关系时，OASIS SOAR A 做了如下概述：

在治理和管理周围总存在一些疑团。……治理关心的是决策。而管理关注的则是执行。换种说法 ,治理描述了领导者所希望的世界 ;而管理则执行活动去实现 管理者 想要的世界。[补全——通过加强治理政策的遵行]。治理所确定是，谁具有权力并负责下决定以及为要下的决定制定向导。管理是决定，实施和度量这些决定的实 际流程。

SOA 治理的责任是建立政策和规则 ,治理(包括管理)主体的职责和责任是在此之下定义的。这包括了“在必须透明的地方要透明、不偏不倚的信任、治理的统一施行以及确保 SOA 生态系统中的可靠且有力的活动。”[OASIS SOAR A]。

没有管理则治理一文不值 (除非受管人员极为规矩) ; 没有治理的管理是不可控和无方向性的。如图 1 所示，治理提供了不同类型的政策并定义了执行和控制 的过程，而管理则实现政策 (实施和加强政策) 并提供政策合规的监控机制。政策被应用到多种事物上 ,包括服务、服务开发、人员、流程和过程。治理要立足于本 地法 律，行业制度和来自企业管理层以及受管对象的反馈。

治理在企业架构中的定位

在给SOA治理定位之前，我必须要先解释这里所指的SOA和企业架构。我理解的SOA作为一个面向服务的架构，它是对具体技术和业务实现不敏感的。在我谈到SOA时，我将依照OASIS SOA RM标准和OASIS SOAR A。而企业架构指的是跨业务-IT边界，包含了企业的业务架构和技术架构在内的企业范围的架构组织。如果你的企业期望最大限度地利用SOA却没有这样的结构，那我建议创建一个这样的架构。

尽管SOA是企业架构的一个非常重要的组成部分，但是它不能代替企业架构。同时，在架构组织中，不一定所有的事物都是或都应该是面向服务的。SOA 为真实 业务世界建模，在这里业务服务是最重要的实体，但并非唯一的。作为一个架构方法论，SOA影响业务和技术方法，但并不具体指定技术。在从面向流程的组织结构（典型地，价值链模型）到面向服务的组织结构（典型地，价值网络模型）的转型过程中，SOA扩展了企业架构。在转型过程中转入的或在SOA下创建的那些组织的部分，如产品，功能和服务等，必须是端到端面向服务的，上至用户接口，下至数据存储访问层。这是SOA成功实现的前提条件。

SOA 治理并非孤立存在，它是企业治理的一部分。一方面，在逐步增加基于 SOA 的解决方案时，我们一个接一个地解决业务任务（如 SOA 最佳实践推荐 的那 样）。但是，SOA 治理不能也这样一个项目到另一个项目，这样它将不起作用。具体来说，如果我们在一个业务部门（BU）的项目中实现服务 ，SOA 治理必须位于部门级，而不是项目级。如果多个部门实施 SOA，则 SOA 治理一定要位于业务线（LOB），部门或跨部门级。这样才能让治理政策均衡 地应用在所有的项目和服务中。

所以，SOA 治理一定要在建设和实施 SOA 人员之上的一个级别。解释这个需求很简单——SOA 治理的权力必须要大于受管对象的权力。这里问题的是不 仅仅在权力的加强（这是管理的特性之一）上，还要在服务组合和聚合的观念上。SOA 治理设定了调控服务组合和相关服务协作的规则。比如，某 SOA 政策可能 规定了 组合本身容许实现的业务规则的最大复杂度，而更复杂的业务规则应该转移至专门的“规则引擎”。基于以上解释，我们可以描绘一个 SOA 治理的结构图，如图 2 所示。

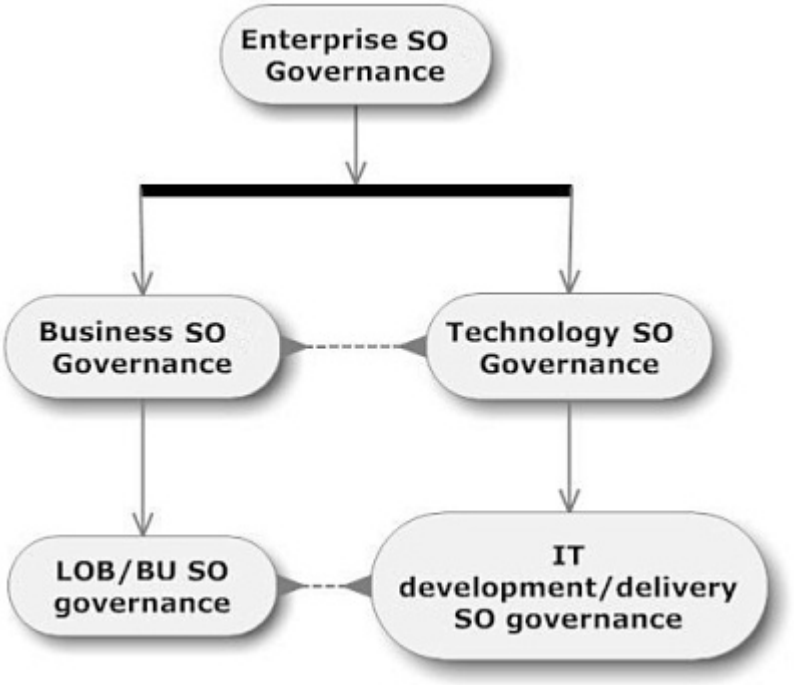


图 2

应该特别注意 SOA 政策职权级别与政策范围之间的关系。我们假设有如下四条政策：

政策 1：

‘所有的技术业务服务必须部署在 IBM 的 WebSphere 应用服务器上，依据 2009 技术路线图的规定确定版本（如 6.1 或以上）。这项政策无需应用于使用动态服务调用的组合服务’。

政策 2：

‘所有使用动态服务调用的技术业务服务必须部署在 IBM 的 WebSphere Business Service Fabric 上，参照 2009 技术路线图规定的版本’。

政策 3：

‘所有技术服务的发布测试必须包含被测服务所依赖的所有服务的端到端的测试。’

政策 4：

‘对所有技术服务的发布测试所遵循的技术和业务执行的政策必须要和这些服务应用在生产环境时一样。’

取决于组织的 SOA 实施深度（有时也称为 SOA 成熟度模型），如 1，2 这样的治理政策可能应用在部门级、LOB 级或企业级。然而，如果不同的 BU/LOB 使用不同的技术来实现 SOA，

前面提到的政策 1 和 2 就不应该应用在企业级或 LOB 级，因为某些 BU 可能使用的是其他技术而不是 IBM 的 WebSphere 产品。与此同时，政策 3 与 4 与实现技术无关，所以它们可能要坚决地应用在企业级。

SOA 治理可能在各个级别影响企业架构。如我们前面提到的，取决于公司内 SOA 实施深度，一些高级别的政策可能暂时不会受到影响，直到 SOA 发展到 企业 级。然而，强烈推荐的开展 SOA 治理的方式是自顶向下的方式，从企业级架构到项目设计。这样，起初 SOA 治理可能只占业务和 IT 治理的一小部分，但却是端 到端的——从业务功能和用户接口直至业务员数据模型和数据存储访问层。当 SOA 采纳扩展到更多的业务和技术领域时，就要对起初建立的基本的 SOA 治理进行扩展，使之兼备所覆盖的新业务域的具体情况。

我希望我正在讨论的东西是相对容易理解的，而且我也曾期望TOGAF 9 在讨论治理和SOA时也采用相似直接的方式。然而，我发现事与愿违。

TOGAF 9 对治理有很好的词藻：

“治理是一种能力，它规定参与行为并支持所有参与者并鼓励他们对确保实现企业的利益和目标而要付出的努力有兴趣并负责任”，而在另一个地方它是这么说的： “架构治理是企业架构及其他架构在企业级范围内被管理和掌控的实践和方向。它包括：实现一个控制系统，用于控制所有架构元素并监控这些活动，以此实现组织 的架构有效地引入、实施和发展……”

每个句子都是正确的而且我也赞同它们。但是，描述了开发企业架构并成为TOGAF核心的架构开发模型（ADM）中的SOA和治理又是怎样的呢？他们是向兔子洞里嬉皮笑脸的猫那样在空中做了一个怪异的表情后消失了吗？

如果 TOGAF 的 ADM 仅有实现治理阶段 那么由谁又如何来治理企业架构的其他方面呢？SOA 治理究竟藏在哪里呢？

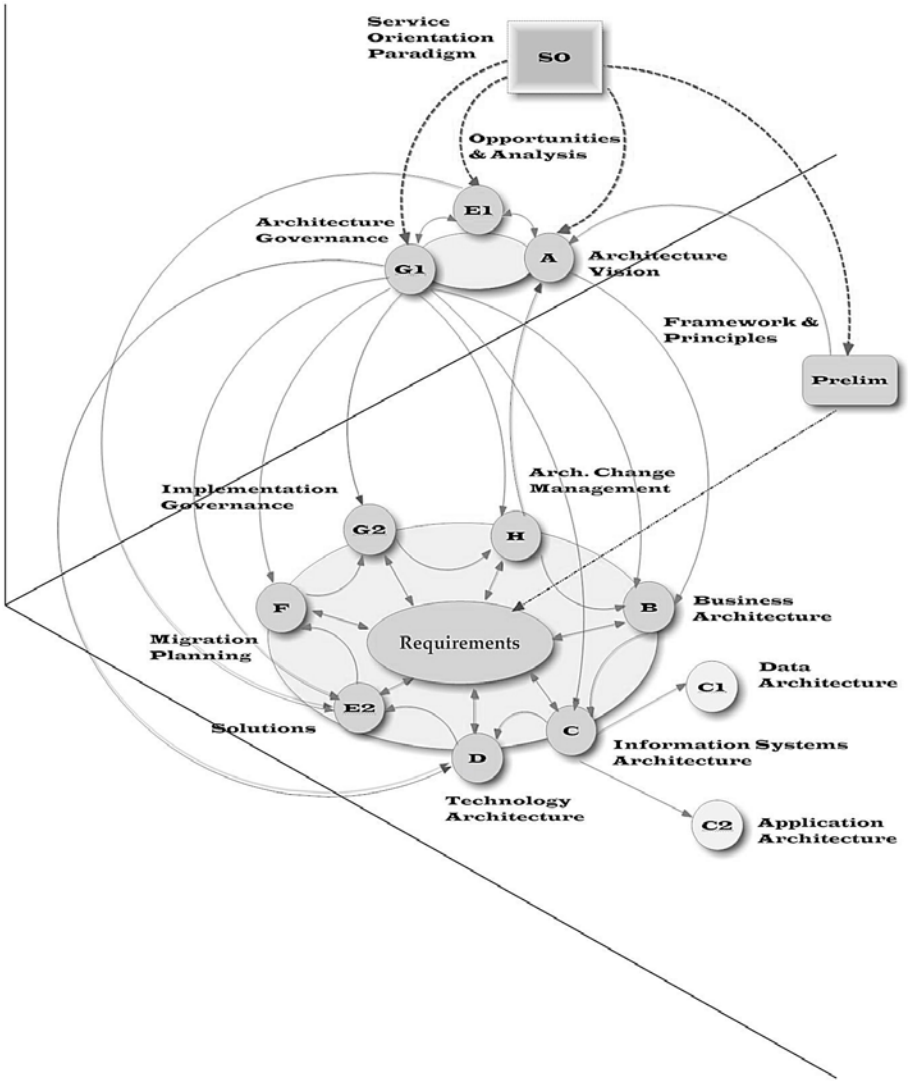


图 3.

我曾参与过一个项目并从 SOA 的角度对 ADM 进行了重定义，如图 3 所示的。我认为图中的架构愿景（Architecture Vision），架构治理（Architecture Governance）和企业级的分析和机会（Analysis and Opportunities）要先于任何其他的架构开发阶段。这三个功能是面向服务的范型（Service Orientation Paradigm）的直接提供者。一个承载面向服务的概念的架构治理必须要高于其他架构建设功能，否则它们将背离制定的方向。

架构愿景（Architecture Vision）必须要与组织的业务战略平行。其意思是任何企业架构中的业务更新都不会看似“从空中掉下来的”；它们必须先在架构先见阶段给审视。而且，在 更新通过了 ADM 周期的业务，信息和技术架构之后，再执行架构机会是不是有点晚了呢？这时侯，架构解决方案应该已经准备好， 在此之后只需要通过迁移计划（Migration Planning）优化一下即可。

架构愿景（Architecture Vision）和分析及机会（Analysis-and-Opportunities）不一定需要在每

次 ADM 循环中建立或更新。这就容许我将它们移到其 他阶段之上且与架构治理（Architecture Governance）放在一块儿。面向服务的范型（ Service Orientation Paradigm）对它们三个都有影响。你会发现实施治理（Implementation Governance）的位置与修改之前一样，还是在 G 处，但是它受到架构治理的控制；该结构更符合 SOA 治理的层次关系。

治理的责任和所有者

已经 有很多出版物从不同的方面 探讨了SOA治理的实施细节。一方面这些出版物包含非常有价值的信息，可是他们并没有强调一些SOA治理的细节，如SOA治理必须能跨所有者边界划分：

- 资源
- 人员和系统安全地交互
- 和管理的方面

即使在小企业的环境中，“事实上，不同的组织和部门的行为常常显示出部门之间存在所有者的界限，这反映了组织实践以及真实的动机和经营这些组织的人的意愿”[OASIS SOAR A]。为了更好地理解所有者边界如何影响 SOA，有必要走近企业环境中的社会结构去看看。

社会环境定义了其中发生的任何行为（包括参与者通过服务的交互）的意义。社会环境的形式化表达就是社会结构。社会结构代表着参与者之间关系的某些文 化方面的特征。“社会结构允许任意多的参与者，并且特定参与者又可能是多个社会结构的成员。因此，在社会结构之间存在频繁地交互，有时当社会结构的目标不 一致时就会产生分歧。” [OASIS SOAR A]

每个社会结构都有一个目的，有时称之为目标。例如，一个实现了价值链业务模型的企业，其社会结构就有助于提高单个业务活动的业务价值。如果这个社会结构中存在的任何关系对某特定活动的业务价值产生了潜在的风险，那么该关系就可能会被停止。

社会结构定义了一些参与者需遵循但不一定明文规定的规则。若试图建立一个不符合社会结构的“精神”的规则，其结果只能是强烈的反抗或无视其存在。这 解释了为什么要把不合适的规则放在合适的权力级别。如果一个企业架构已经建立了评审所有 IT 项目并检查其是否符合企业技术政策的机构，并拥有停止任何合规 项目的机构，那么这种不符合的规则将导致社会结构的修正。

实践证明在有适当的机构存在的情况下，不必加强政策。因为对于要遵循政策人们来说，他们足以认识到不遵守政策将产生的不良后果。然而，这种非加强的实践还不能成为取消这种

权力的理由。

特定的社会结构会影响所有者对某资源或能力拥有的权力，这不仅关系到权力本身，还关系到所有者的责任。换言之，所有者隐含了责任和权力；后者用于设 定政策，而政策往往与责任一起应用于相应的资源或能力。如果所有者不拥有资源和能力，这就要求使用资源和能力的权力最终必须回归到所有者那里。对于基于价 值链模型的社会结构而言，对所用的资源不具有所有权的效果构成了利用该资源交付业务价值的风险。这就解释了为什么价值链模型总是设定足够紧密的治理政策来 保证所有权，或者，至少直接控制所有相关的资源和能力。对于 IT 而言，这导致了竖井的形成并极大地孤立了单个所有权下的应用。

如我们所知，SOA 不适用于独立应用，或者整合的目的不是改变所有者模型而是保护它的场景。由于所有者总是与所有者边界相关联，在所有者边界内的参 与者对属于不同所有者的资源和能力的使用权力是受限制的。这解释了为什么基于价值链模型的企业通常在实施 SOA 时存在严重问题。它不仅仅是关于限制在所有 者边界内的服务的重用，而且还关于为别的所有者边界创建服务的管理上的制约。这还导致了选择服务的实现技术时不会考虑来自其他所有者边界的潜在服务消费 者。这些竖井和孤立就是 SOA 解决方案要斗争的对象。

我要说的是价值网络的业务模型几乎摆脱了上文所描述的价值链模型所面临的问题。我使用“几乎”的原因是任何模型都不能改变人的本性和所有权的诱惑。然而，价值网络为业务所有者和相关管理制定了不同的所有者内容与职责。

如前文提及的，有效的治理采用自顶向下的权力，它可以被看成是某种特定的权力和责任向下一级的代理，从而达成最初级别上一致的一部分。例如，一个 LOB 可能包含多个 BU，而每个 BU 管理这它本职范围内的事务，这些事务与该 BU 本身的任务关联的同时还与 LOB 的目标一致。另一个 LOB 也可能拥有其自 身的企业级代理给它的治理权力。合在一起就创建了符合整合企业组织的治理链的结构。如果某企业实现了价值网络，它就会收获更多的业务价值，它们来自不同参 与者的活动和合作。经实践证明，该价值超越了类似场合中由价值链模型带来的个体业务价值总和。相反，如果某企业实现了价值链模型，那么它将始终面临不同治 理链之间的利益冲突的风险，那么企业执行官们将忙于将价值链同企业战略计划对齐，而非提高公司在市场上的地位。

那么，SOA 治理委员会在价值网络模型中的目标是赢得个体和工作单元的集合效应（如果不是这样，那应让其这样）。该事实符合服务组合的 SOA 观点， 服务组合最大限度地实现了 SOA 的业务价值。协同工作的需求首先是通过将 SOA 治理功能委托给企业内跨部门和跨功能的业务组织实现的，逐步分至功能单元。 跨功能域的业务组织的治理权力拥有对多个治理域的权威仲裁并协助避免和解决所有权冲突。

SOA 治理在价值网络模型中再分发会导致个体业务团队及其经理的权力和义务的变化。在这种情况下，治理政策要求团队为业务服务的所有消费者担当义其务，而其权力和所有权却缩小到只与该团队/服务本身相关的活动。

如果某人认为这种治理幻境只存在于兔子洞中，那么他们应该去学习那些最成功的企业是如何管理他们的内部机构的。在电信行业，金融行业和零售行业里就可以找到这样的成功企业。

SOA治理细节

SOA 治理细节并不在某些特殊的治理方法中，而是在整个企业中从业务到技术的受管对象中。这样的分布要求特别关注治理政策一致性。例如，如果某人为 用户接受测试（UAT）制定了政策而没有为测试环境设定政策要求它必须包含所有支撑性服务，那么就存在 UAT 测试结果的不一致的风险，并且在将来会为此付 出代价。

SOA 治理应用于以下列出的 SOA 环境的主要方面：

- 服务结构——包含服务以及它在服务交互域中的关系最小元素集。相关的政策关注开发，整合和部署。
- SOA 基础设施——“提供支持和促进服务使用的工具和产品”[OASIS SOAR A]，这里我们解决部署和运行时政策。
- 服务目录——“要求运行服务在基础设施内可通过手动或自动地公开接口进行访问”[OASIS SOAR A]，它关系到服务的管理政策。
- 参与者交互——希望所有参与者遵守的一致约束 [OASIS SOAR A]。该领域的政策关注服务的可达性和运行时行为。

以下列出的是 SOA 治理中常被错误的忽略的一些方面：

- 服务设计——这里的政策一定要求定义服务范围，边界以及标识服务描述和服务契约的必备元素和可选元素。
- 服务开发——此处的政策提供了服务开发流程的定义、最佳实践规则、推荐的工具和服务创建的项目管理实践细节和服务测试和发布的方向等。
- 服务组装——在这里，政策定义了聚合协作或符合服务的规则以及组合更改的规则（流程、过程及所有者关系等）。
- 服务资源访问——相关的政策描述了哪些内部共享的资源（如数据存储，遗留系统等）

是否被使用及如何使用。

- 服务生命周期管理政策包含服务的版本机制和过程。
- 服务监控和审计。
- 企业的各个级别的服务管理角色以及服务契约协商过程。
- 交互渠道限制包含自动和手动的 UI 接口。

该列表并不意味着 SOA 治理不关心也不包含没有列出的其他方面。

虽然看起来上述列表管理了 SOA 环境中的所有方面，以至于没有创新发展的空间，然而这并非事实。SOA 政策是用于引导 SOA 及其实现，而不是决定怎么实现。而且，有些政策仅仅是暂时的，因为一旦有了足够的 SOA 实践，这些政策就将退役。

这里是我个人的一个经历。几年前，我工作的公司在 IT 部门有一些特殊的卓越中心（Center of Excellence, CoE），其中一个负责 SOA，而另一负责开源软件的使用。有一次，开源 CoE 运行使用 Apex Web 服务 V1.1 ,而另一个 CoE 必须要提出一个临时政策来限制在 WebSphere 5.1 的 Web 服务开发工具上使用 Web 服务。 后来，Apex V1.1 Web 服务又在另一个 WebSphere 版本上通通允许使用，所以相应的策略也就消除了。

优秀的SOA治理细节不仅能应用于服务的技术实现，还能用于它们的业务领域。这表明了上文列表中的名词“服务”所代表的不仅仅是自动服务，还包含半自动服务甚至人工服务。这类服务的描述可以在《通向SOE的梯子》一书中找到。

SOA 治理负责定义跨所有域的控制以及跨所有域的交互规则。如果所有的服务交互参与者都认同单一的治理机构，这才可能实现。这并不是多所有权场景的 最可能方式。相反，所有参与者的治理实体不得不设定双方协议和政策映射机制。这类似于跨多个独立实体的企业安全控制——存在很小并集中的安全策略标准集和 许多横切的协议以及对特殊安全策略的映射。建立如此架构需要特定的治理流程和合适的权限。因此，“SOA 治理的主要区别是，对企业合作本身的正确评价和为 维持高效合作，它需要依赖于长远的共同目标”[OASIS SOAR A]

SOA治理手段

SOA治理在IT面来看仍然被视为奇怪的活动范围，而在业务面看它是夹在治理和管理中间的一段模糊地带。当执行者进入这一领域时，很多人会问“我们 该怎么做呢？”或“我们该用什么技术和工具呢？”。作为回答，提供商提供了一大堆的工具，它们可被视为对“如何进行治

理”这一问题的回答。很多工具都打出 了能够“记录服务”的标语，而这其实与治理并不相干。这非常类似于爱丽丝在奇境中玩槌球游戏时所见到的，她说“他们都在嘈杂地相互争吵，每个人都听不到别人说的话，并且似乎不存在任何的规则。”那么，SOA治理应该遵循什么样的规则呢？

第一条规则，我的理解是，要理解SOA治理可以缓解或解决哪些我们在使用服务时已有的问题或将出现的问题。只有这样我们才能知道要治理什么。对此，Dave Linthicum曾提到 ：“关注SOA治理的方法和实践，而不是 技术。首要问题是创建SOA治理战略并保证它在每个环节中起作用”。

在弄清了必须要管理的事情之后，第二条规则是在服务和服务组合中找到确切的需要加强政策的点。加强本身并非治理的责任，而是实现管理的职责。

第三条规则是鼓励政策合规准则和校验方法的定义。

规则四要求你思考先行，理解为什么以及你在做什么。因为，治理是一个双刃剑，它能为企业交付业务价值带来很大帮助，也能破坏它。

只有此时才是回答如何进行治理的时机。Dave Linthicum 解释到“只有在当你对问题域的语义层、服务层和流程层有了全面的理解之后，如果实在需要治理技术，才应购买 SOA 治理技术。而不要在此 之前”。遵照上述规则，你就能定义治理工具所需的功能。适当的 SOA 治理始于你需要什么而不是你拥有什么。

很不幸，在实践中情况却是相反的，就像这一领域中常常出现的一样——我们购买了承诺支持 SOA 治理的工具，而治理却没有发生。SOA 治理的工作是应该由我们来完成，不管有没有工具。如果我们清楚我们在做什么，那么工具会有所帮助，而如果我们不清楚，工具反而会让我们困惑。

我们在前面讨论过治理流程包含一组手动和自动的活动，从创建政策和注册开始，然后是政策应用，合规检验和报告等。IBM 的 Muhammed Yaseen Muriankara 认为，治理实现或治理框架必须要自动化才能提供治理流程的功能。

他说：

一个启动项目要包含的不可或缺的自动能力如下：

- 用于发现和发布服务相关构建和元数据的服务注册存储库，用于：
  - 查找适当的已授权服务

- 避免重复工作
- 促进重用
- 标识服务在 SOA 生命周期中的当前状态
- 为服务订阅者提供可视性
- 了解关联服务以及服务变更的影响
- 互通服务变更完成的信息

- 理想情况下注册库应该可在运行时优化，这样，存储其中的元数据可在运行时用于通过动态路由提供信息扩增的能力……

没错，集中的服务注册很重要，但是其重要性出现的前提是所有权益人都同意为当前和将来的服务使用它。例如，注册库对于“避免重复工作”的必要性就不 清晰；可以提出应用于避免重复劳动的特殊购买（即购买服务注册库）申请的主体是治理政策。所有这些意味着第一个治理手段是治理政策注册库，而然后是服务注 册库。

服务政策元数据是另一个暗礁。为了让其发挥作用，应该要求所有潜在参与者共享政策本体和语义。我们已经谈到，达到如此的共享并非易事。

服务注册和存储方面的引领工具是HP Systinet /S2，SAG Centrasite，SOA Software，TIBCO，ActiveMatrix和Oracle-BEA。然而，服务注册可以从普通的Excel表格开始。具体选择哪个工具合适 取决于个体公司的情况和需求。这些工具有助于“对服务元数据进行收集和编目”，但我不同意一些提供商 所宣称的编 目工具应该用于“组织相互依赖，它们[服务]互相依赖，按照SOA政策文档的描述定义了如何通过服务组合实现业务需求”。我认为这些提供商跨越了能力边界 ——服务之间如何相互依赖和“如何通过服务组合实现业务需求”属于业务需求与其实现的特权，（而非编目工具的）。不过，政策编目工具非常有助于在一个集中 的地方维护SOA治理的政策以实现更好地观察，维护和管理。

SOA治理中反映了SOA的一个非常特殊的方面——服务执行过程中所遵循的政策必须要外部化，而不是插入到服务实现之中。该细节的结果是，某些已经 嵌入其执行政策的遗留系统不管使用何种接口都不适合担任“服务”的角色。事实上，这样的系统应该由实际的服务应用进行包装，只用作服务资源。该逻辑导致了 创建插入政策的接口需求（每种政策一个）。SOA治理必须要考虑这种接口并指明支持这种政策实现方式的合适工具。插入政策的接口允许在不修改服务的情况下 新增或修改需执行的服务政策。

最后，我想对“务实SOA治理”运动发表一下我的观点。正如该运动的名字所言，它并非关于SOA治理本身，而是为人们已经做的事情冠以“SOA治 理”的名号。举例来说，如果IT开发人员先创建了域对象，然后思考这些对象如何被转换成业务服务，“务实SOA治理”只能“祝福”它，而尽管这种做法可能 会导致很多跨服务边界的问题，这正是SOA实施中的一个主要谬误之一。MuleSource的CTO和合伙创立人Ross Mason支持这种“实用”，他说“在当今世界，传统自顶向下的SOA明显已经过时，今天的企业要看到真正的价值需要一种新的实用方法” 这对我来说又是一个兔子洞中的测试，因为由OMG签署的“SOA导航协定”，The Open Group和OASIS都明确地说明面向服务的方法论是自顶向下的，而SOA的实现可以从任何方向开始。

Mason 曾质问过自顶向下的 SOA（包括 SOA 治理）的效力问题，该问题源自“只有 20%的企业从 SOA 获益”这一事实。对我来说，这听起来就像抓住兔子的“耳朵和胡须”显示自己是绅士。该数字只反应一个简单的事情：只有 20%的企业拥有正确的 SOA 和 SOA 治理。

Mason 认为 ,为了使用这些 SOA 治理工具 ,SOA 治理需要用来避免人们的“重大的行为改变”。进一步说，“任何变更的引入，都要在尊重受影响的 角色和程序的方式下完成。对于 SOA 治理政策，架构师的冗长的需求文档几乎被会大多数开发人员忽略。相反，那些需求可以也应该实现成自动政策 ……只有当日常 IT 从业人员看到新方法对于开发带来的益处时，企业才能看到整体价值”。

一方面，把不受欢迎的变更放进自动政策的想法是个很好的点子；另一方面，隐藏不受欢迎的变更并不能改变人们的行为，如果没有合理执行变更的话，总会 人找到方法绕开它。例如，Parasoft 用于编写 java 代码的 JTest 工具拥有数以百计的编写风格政策。如果代码违反了风格政策，则无法通过编译， 即开发人员就无法交付工作。我亲眼见过开发人员关闭这些政策控制，因为在不同的代码开发阶段需要不同的风格政策，而工具本身无法足够聪明地适应真实的开发 流程。

面向“日常IT从业者”的问题是：正确的SOA和SOA治理的确要求对开发流程周期的理解上 进行重大改变。然而，这并不是新鲜事物或者首次登台容易被打断的演员。从CS结构到多层结构模型的转化一样需要来自开发人员的不同理解 ,多层模型被广泛采 纳也经历了一段时间。我只能说，对于必要的改变，SOA治理政策将不会为了那些不具备面向服务概念的人的防范而作出修改。也就是说，如果某开发人员尝试 用，比方说，面向对象的方针进行服务开发，那么他只有两条出路，不是他改变自己的思维，向面向服务的方式改变并领导其公司走向成功；就是破坏SOA项目并 承担相应的后果。

我对 SOA 治理的定位如此自信的原因是面向服务关心的是真真切切的业务产品和过程。它为如何做事增添了更强的控制。任何不适合保护 SOA 业务价值的事物迟早都将撤退。因此，

真正意义的务实 SOA 治理应该保护并提高业务面和技术面的业务目标。

总结

尽管有 SOA 治理标准的多方努力，但是仍然需要一些概念方面的改进。因此，我在文中提出了一些观点供大家参考和讨论。

最基本的观点是 SOA 治理和隶属于管理功能的治理执行二者之间需要分别对待。

第二个观点提出了 SOA 治理必须要在企业范围内引导所有的 SOA 相关项目，因此，要放在它们之上。这就产生了我对 TOGAF ADM 的修改并对相应的架构开发阶段的重新定位。

第三个观点讨论的是 SOA 治理跨越部门所有者边界和业务面和技术面之间的人为边界的重要性。如果正确实现了 SOA 治理，就会对企业的社会结构，包括人员，资源和功能关系产生影响。

第四个观点建议 SOA 治理应该始于以下方面：为什么需要它、它是什么、应该在哪些领域进行以及谁提供它。在此之后再讨论支持工具和手段。昂贵的服务 注册/存储系统甚至免费的开源产品，在人们意识到需要它们之前，是不会有任何帮助的。SOA 治理较之把 Web 服务接口注册在某地要复杂很多；最小的 SOA 治理也需要一个存放 SOA 政策的存储库以及相应的维护和管理机制。

最后，我希望大家打破技术的“框框”去理解 SOA 治理；尝试思考它并循序渐进且准确地应用在底层业务流程和相应的业务活动中。这才是 SOA 治理前进的方向。

原文链接：http://infoq.com/cn/articles/poulin-wonderland-soa-governance

相关内容：

- 有效的SOA治理一定需要注册与存储吗？
- SOA的管理策略
- 十年SOA：当前的位置和未来的方向
- SOA与服务识别
- 程立谈大规模SOA系统

新品推荐 | New Products

YouTube发布HTML 5 视频，但并不支持FireFox 3.6

作者 Alex Blewitt **译者** 张龙

近日 YouTube 发布了 HTML 5 video Beta，无需使用 Flash 插件即可实现回放功能。视频格式仅限于 H.264，这在某些硬件设备上会提升性能，但却将 FireFox 3.6 排除在外，因为它只支持 Ogg 视频格式。

原文链接：http://www.infoq.com/cn/news/2010/01/youtube-html5

Gremlin，一门操作图表的语言

作者 Abel Avram **译者** 马国耀

Gremlin 是操作图表的一个非常有用的图灵完备的编程语言。它是一种 Java DSL 语言，对图表进行查询、分析和操作时大量使用了 XPath。

原文链接：http://infoq.com/cn/news/2010/01/Gremlin

Windows Workflow 4：旧瓶装新酒

作者 Jonathan Allen 译者 张龙

Windows Workflow 4 基本上对原有的程序库进行了重写。虽然目标一样，都是为长时间运行的任务提供一种模型语言，但重写之后的程序库还是有很多与众不同的地方。

原文链接：http://infoq.com/cn/news/2010/01/WF-4

VS2010 和.NET 4.0 将延期发布

作者 Jon Arild Tørresdal译者 朱永光

在开发部门的市场与沟通经理 Rob Caron 的一个简短公告中，Visual Studio 2010 和.NET Framework 4.0 将于 2010 年 4 月 12 日官方发布，而非之前宣布的 3 月 22 日。然而，RTM 的发布时间依旧未知。

原文链接：http://infoq.com/cn/news/2010/01/vs2010-launch-postponed

jQuery 1.4 发布：性能改进、焕然一新的API文档与支持论坛

作者 Dionysios G. Synodinos译者 张龙

在 jQuery 四周岁生日来临之际，jQuery 团队发布了万众期待的 jQuery 1.4。该版本最显著的特点就是最常用的 jQuery 方法的性能得到了显著的提升。

原文链接：http://infoq.com/cn/news/2010/01/jquery-1.4

Google Collection 1.0 增强了对Java集合框架的支持

作者 Josh Long 译者 晁晓娟

Google Collections 库，1.0 最终版，2009 年 12 月发布了。你可以在 http://code.google.com/p/google- collections/下载它。这个类库是 Google 工程师 Kevin Bourrillion 和 Jared Levy 的智慧结晶。最近几年由于 Google 工程师如 Doug Lea, Josh Bloch 和 Bob Lee 以及开源社区的贡献，它发展地很快。

原文链接： http://www.infoq.com/cn/news/2010/01/google_collections_10

Scala 2.8 Beta 1 发布

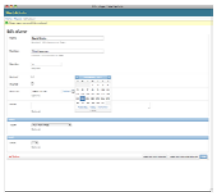
作者 Mirko Stocker 译者 张龙

今天，万众期待的 Scala 2.8 Beta1 终于发布了。该版本包含很多新特性，比如重新设计的集合库、具名（Named）与默认参数以及改进的 Eclipse IDE 等。

原文链接：http://infoq.com/cn/news/2010/01/scala-beta1

MerbAdmin：Merb数据管理好帮手

作者 丁雪丰



Merb 中要加入类似 Django 的 Admin 功能早有传闻，如今在 Erik Michaels-Ober 等人的努力下，期待已久的 MerbAdmin 终于基本成型。

原文链接：http://www.infoq.com/cn/news/2010/01/merb-admin

封面植物

多室八角金盘

稀有种。多室八角金盘是我国台湾省特有的珍贵植物。目前仅在台湾中央山脉一带星散分布，由于森林植被不断遭到破坏，本种分布范围逐渐狭窄，数量也日益减少。

形态特征：常绿小乔木或灌木，幼枝被褐色绒毛，后变无毛。叶丛生枝顶，圆形，掌状 5 一 7 深裂，宽 15 - 30 厘米，裂片之间呈圆凹形，裂片长椭圆形，先端渐尖或长尾状 渐尖，边缘有疏锯齿，上面绿色，下面淡绿色，幼时两面有棕色绒毛，后变无毛；叶柄圆柱形，基部粗肥，有纤毛，较叶片略长或稍短。伞形花序排成圆锥状，顶 生，长 30 - 40 厘米，基部分枝长 14 厘米 ,密生黄色绒毛 ;伞形花序有花约 20 朵 ,花小；萼筒短，钟形，先端截形，有 10 棱；花瓣 5，长椭圆形，长约 4 毫 米，先端短尖；雄蕊 5，较花瓣略长；子房 8 一 10 室，每室具 1 悬垂胚珠；花柱 8 一 10，分离，长约 0.5 毫米。浆果球形，直径约 4 毫米。

特性：多室八角金盘分布区的气候温凉湿润，年平均温 10.7℃，最高月平均温 13.8℃，最低月平均温 5.8℃，年降水量为 4246 毫米，云雾多，湿度大。土壤 为发育良好的棕壤或灰棕壤，表层疏松，含较多的腐殖质，pH 值 5.8 - 6.0。多室八角金盘为耐阴的灌木或小乔木，通常生于阔叶林林下荫湿地，根系为小中 径的斜出根，深度可达 60 厘米，细根较多，主根多与地表面平走。在通气良好的肥沃土壤生长良好。

保护价值：八角金盘属共 2 种，1 种特产于我国台湾，另 1 种分布于日本。本属和兰屿加属 Boerlagiodendron 中的单叶类型有较密切的亲缘关系，对于研究五加科的系统发育以及中国、日本植物区系，特别是台湾、日本的植物区系有一定的科学意义。


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：霍泰稳

总编助理：刘申

编辑：李明 胡键 宋玮 郑柯 朱永光 郭晓刚

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

本期主编：霍泰稳，InfoQ 中文站总编辑

霍泰稳，InfoQ 中文站总编辑，有多年的软件开发经验和媒体从业经历，以技术传播为己任，关注企业软件开发领域的变化与创新。曾先后参与《程序员》杂志、《MSDN 开发精选》杂志、《开源大本营》图书和《开源技术选型手册》2008 版图书的策划编辑工作。。

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中文站
www.infoq.com/cn

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