时刻关注企业软件开发领域的变化与创新

架构师

12月 ARCHITECT

Luke Gelea
谈Ruby和Erlang

处理.NET中的
内存泄漏

Web开发必知
的八种隔离级别

模型驱动开发的
误解和挑战

数据库应用中的影
响性能的反模式

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语 | Editor's letter

时间总是不够用

The Matrix 中经典台词很多，然而最让我印象深刻的，却是 Morpheus 的一句并不起眼的台词："Time is always again us"，中文大意是"时间总是不够用"。是的，不仅仅是时间，人力、设备、预算，经常通通都不够用。资源总是宝贵的，资源也总是紧缺的。

开发人员都会遇到类似的问题，团队成员人力不足，或者项目进度太紧，又或者再多加两台服务器才能够满足性能要求，但是已经没有那么多预算了。资源在项目开发中，似乎总是不够用。开发者们都是追求完美的生物，他们都在力图保持着事物最接近完美的趋势。因此，资源问题往往让开发者们头疼不已。再多几个人，这个项目一定能按时发布；再多三个月，这个项目的 BUG 一定会少很多；服务器的配置再好一些，这个系统一定不会像现在这么慢。类似的抱怨，相信大家一定不会陌生。

资源受限是开发过程中的常态。我们在追求完美的同时，身上也负着沉重的枷锁。不管有多少资源，可能最终还是不够用。这对于开发人员来说，是一个难题，但更是一个有趣的挑战。既然不能"开源"，那我们还可以"节流"。我们可以让团队更加敏捷，从而消除开发过程中的浪费，提高团队生产力 。还可以通过更高效的设计和更精心的优化， 提升硬件利用率。珍惜每一分可以节约的资源，充分地利用资源，这是一门功夫，更是一门艺术。与其抱怨，不如改变。有心者，依然可以戴着镣铐翩翩起舞。

时间总是不够用，但是 Morpheus 还是成功了。相信成功之时，资源问题便不再是一个抱怨，而是一种炫耀的资本了。我很高兴看到那一天的到来，仿佛勋章一样，记述着当年开发者们和资源斗争的传奇。

本期主编： 李明

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

目　录

[新品推荐]

[封面植物]

人物专访 | Interview

Luke Galea谈Ruby和Erlang

本篇采访录制于 FutureRuby，在这次采访里，Luke Galea 谈到了他通过 Ruby 和 Merb 建站以及用 Erlang 在消息层整合它们的经验。

Luke Galea是Avid Life Media的开发总监，这是一家拥有多个大型约会网站的多伦多公司。他是Rails的Hyperactive Resource插件的作者，活跃于Ruby和Erlang社区。他最近的项目是http://www.cougarlife.com。

InfoQ：这里是 FutureRuby 2009，多伦多。坐在我旁边的是 Luke Galea。何不先来个自我介绍？

Luke：我是一名 Ruby 开发者，在多伦多的 Avid Life Media 公司工作。我们拥有很多约会网站，最著名的是 www.establishmen.com。我们处理来自世界各地的大量用户的请求。我们克服了很多技术难题，我们用 Merb 创建了一个非常棒的平台，以 Erlang 作为后端，用这个平台驱动了我们最新的网站的开发，比如说 www.establishmen.com，还有许多网站在做。这真是个有趣的经历，从我们的老网站上吸取经验，比如说 www.hotornot.com 和 www.ashleymadison.com，并且融入使用最新技术创建的网站里。Ruby、Merb、Erlang 使我们可以在极短时间里把一些很棒的东西融到一起，这些东西无论从开发的角度还是从用户的角度都有着令人惊异的性能。真让人兴奋。

InfoQ：在把不同的网站从其它平台转到 Ruby 的过程中，你也把这些不同的网站统一到一个通用平台上，对吗？

Luke：没错。在这一点上，我们更关注使平台的功能完善和稳定。我们不打算换掉所有网站的 PHP，但最终我们希望有一个平台可以处理所有东西。按照我们的开发进度，通过我们所有现代工具获得的开发速度，我预期我们很快就可以实现目标，所以很兴奋。然而，在这一点上，我们正在开发一系列网站，一个比一个更困难、更复杂，这迫使我们不断地往这个平台里添加功能，以便证明这个平台适用于非常艰难的情况以及允许我们逐渐地往里面添加功能。这的确有很多乐趣，挑战也很大。我们努力使所有东西保持抽象和通用，同时也能适应每个网站的特殊性。

InfoQ：你在前端（也就是网站）使用 Merb 和 Rails，你也把它用于其它部分的逻辑吗？

Luke：我们使用这种逐渐被认为是标准的模式：jQuery 在前端，Haml 和 SASS 处理所有显示，然后我们用 Merb 和 DataMapper 做网站本身，但是，为了使事情来得更好，我们在后端结合 RabbitMQ 使用了 Erlang。基本上，所有可以异步处理的东西我们都交给 Erlang 处理，而我们也正是这样做的。从用户的角度来看，他们很快就能看到他们的页面，但在后台，事情还在处理。当某人登录进来，我们马上在后台开始工作，为的是预先帮他们准备好需要的东西，或者找出它们接下来会看到谁，或者预先生成一组可能的搜索结果诸如此类。当然，后台还会碰到很多需求，但 Merb 处理了大部分逻辑。

InfoQ：你是怎么接触 Erlang 的？

Luke：几年前我遇到 Erlang，那时我正在为大学的帮助网络中心工作。一开始我把它看作一门 ETL（提取、转换和装载）语言，从各个不同的医疗系统获取数据，然后对它进行转换并把它装载到数据仓库或者之类的东西里。现在，所有东西要么通过自定义的专有语言完成，这些语言是函数式语言，要么人们通过他们恰好懂得的语言把它整理到一起。我认为 Erlang 是负责这项工作的极佳语言，但是，我逐渐发现，它缺少了很多很多 Ruby 爱好者喜欢的东西。

没有变形库（inflection libraries）帮你把字符串或之类的东西复数化或者单数化，所以我要到 Ruby 那里拿到 Erlang 这里用，我认为那些东西非常酷，因为大部分 Erlang 开发者都是做电信的。所以他们从不写复数库或者变形库，但是，如果 Erlang 想得到更多的支持，它就该有这些使 Ruby 变得很棒的东西。尽管如此，从专业的角度来说，我们实际上只用 Erlang 做一些消息通信和后台处理之类的东西。我希望他们继续增强类库和语言的功能，使之最终成为一个可以用来轻松开发整个网络应用程序的地方。

很高兴看到 Ruby 社区的人逐渐对 Erlang 产生兴趣。在去年的 RubyFringe，很多人开始讨论我们各自是如何使用 Erlang 的，很多人都觉得对于 Ruby 爱好者来说它将会是下一个焦点。所以我们创建了多伦多 Erlang 小组 T.Erl，Ruby 爱好者聚集到这里讨论 Erlang，我们还有少数人每天领取工资从事 Erlang 开发的，我认为随着时间的推移，我们将会逐渐看到更多人领取工资从事 Erlang 开发，越来越多 Ruby 爱好者会发现 Erlang 对其工作的帮助。

InfoQ：你是怎样与 Erlang 进行互操作的？Erlang 是以某种网络格式获取消息的吗，Jabber 还是其它方式？

Luke：我们使用的是 RabbitMQ，这是一个消息队列。基本上，我们从 Ruby 往队列发送消息，然后它们会在后台处理，这是一种做法。另一种做法是按照网站，更确切地说，按照使用情

况通过 XMPP 进行交互，这是一个 Jabber 协议。如果我们要做通知机制，或者试图获得接近实时的效果，那么我们会把它看作聊天协议。我们有一些自定义的聊天数据包可以发送到 ejabberd，这是 Erlang Jabber 后台程序，然后把它分发给所有需要知道礼物已经发出或者新的私人消息已经收到的人。

InfoQ：你提到使用 Merb，它快要被淘汰了。你会不会对此感到高兴？你打算使用 Rails 吗？你是否为此使用 Rails 的子集？你有什么计划？

Luke：这绝对是个好问题。过去，我只用 Rails，但直到你用了 Merb 才意识到你错过了什么。事后发现 Rails 很不一致。它的 API 不像 Merb 那样深思熟虑的简洁，我明白这是因为 Merb 从 Rails 那吸取了教训，并且无需顾虑向后兼容等问题。在这一点上，很难想像它们之间是如何协调的，因为在 Merb 开发者和 Rails 开发者的角度来看，这些 API 如此不同，以至于我不能简单地来回切换。

如果我写了一整天 Merb 代码然后回家，我无法轻易用回它，因为它快要关闭了，我猜这是因为 Rails 有一个很大的用户群，所以 Rails 3.0 将会偏向 Rails 而不是 Merb。从我们的角度来看，我们要么在将来某个时刻饮弹自尽，要么把所有东西都转到 Rails 3.0，或者，如果我们找到足够的支持使 Merb 存活下来，那么我们会坚持下去，但我预料大部分使用 Merb 的人都会转到 Rails 3.0。

InfoQ：据说有人计划提供了一些工具，帮助从 Merb 转到 Rails。你见过这些工具吗？

Luke：没有，我还没试过这些工具。我怀疑有些东西用过就忘了。我不认为你会在它里面运行某些东西。甚至曾经有些工具可以帮助你从 Rails 1.0 转到 Rails 2.0，它会找出代码里的参数、添加参数或者诸如此类的问题。API 的改变只需进行文本搜索，找出你在哪里使用了它们，然后你自己去那里修正。我相信这些东西是有用的，但我们的工作量仍然很大。即使 Rails 的一次小改动的升级也是一个大问题。

过去，在我工作过的 MDL（Medical Decision Logic），我们的原则是，如果我们想用新版 Rails 的一个功能，这将会是我们升级的合理理由，但我们不会为了升级而升级，因为在很多情况下，网站工作良好，所有人都感到很满意。我们不会被任何版本不一致性拖累，要证明花费的时间是合理的真的很难。Active Resource 就曾经导致我们这样做。我们在 MDL 有个应用程序用于临床癌症研究，它大约有 55,000-60,000 行 Ruby 代码，这是一个大型 Rails 应用程序。我们决定升级到新版的 Rails，是 Active Resource 使我们踏出这一步的，因为我们需要连接不同的系统，而这似乎是最佳的途径。

InfoQ：你创建了新版的 ActiveResource？

Luke：是的，我们详细检查了把这个庞大的应用程序升级到新版的 Rails 将会遇到的所有问题，然后我们开始使用 ActiveResource，我们发现所有需要注意的问题都出现了，比如说，我们启动它并查看所有文档，但注意事项不正确。它的 API 应该和活动记录（active record）的一样，但它只有很小的一部分，甚至在某些情况下，相同的方法实现了，但这个实现包含了不同的语义，行为也不一样。它们很接近，以至于你发现你期望能做的事情做不到，以及你通常能做的事情突然在后面出错了。

我们当时的做法是创建了 HyperactiveResource 并让它扩展 Active Resource，然后修改它的行为，还添加了新的行为，以便支持诸如关联之类的东西，并使一个活动资源（active resource）能从属于另一个活动资源或者一个模型，还有的就是使活动记录和活动资源共存并相互引用。我想说的是，如果你想使用 ActiveResource 来创建映射，你真的需要看看 Hyperactive Resource，否则等着你的将会是一个痛苦的世界。

InfoQ：你提到你曾在医疗领域工作过。你那时做的是什么？

Luke：开始时我为多伦多以及一个医疗网络的众多医院工作，我们的应用程序很多都是内部的，它们为医院里的多种不同功能提供支持。我第一次接触 Ruby 时就遇到 Rails 了，那时离 1.0 版的发布还有很久，我们看到一个试用它的机会，我们有 5、6 个用户，他们都是内部的，他们对停机时间以及我们期望遇到的奇怪问题都很宽容。

因此，我们引入 Rails，用于图表跟踪以及许多其它需求，像健康记录。随着时间的推移，它逐渐成长，因为它使我们很容易就可以添加新特性。越来越多功能在我们的 Rails 应用程序里完成，比如说，在后台跟踪病人的图表去哪了或者两个医生之间的来往信件等等。最终，我从那转到从事医疗研究软件，我们协助医生进行大量临床试验，你有上千个病人试用新型抗癌药物，你需要跟踪他们的进展、实验结果以及所有这类东西。

InfoQ：这基本上是数据库应用程序？

Luke：粗略地说，是的。我想，临床研究管理系统的一大部分是时间表以及核对某些事实，比如说，你有一个药不能在特定实验结果出来的两天内服用，因为它会扭曲这个结果或者这个药会产生冲突。当你面对一个要试用不同实验提供的 20 到 30 个药的病人时，找出哪些天应该服用哪些药以及实验将在何时结束就会成为真正的问题。

我们允许他们输入这项研究的元数据，并使用它们从 Ruby 里调用 Prolog，Prolog 善于约束求解（constraint solving），于是 Ruby 会定义所有这些约束，然后 Prolog 会核对所有事情发生的理想时间并把结果转换到病人的日历里。他们可以说：“好了，我知道周一或者周二我要过来试用这个药，如果因为某些原因拖延了，就会和其它东西冲突。”

InfoQ：这是否意味着你从 Ruby 或者 Prolog 输入数据创建 Prolog 代码？

Luke：是的，我们创建了一个 Prolog 输入数据。最终也没有那么多 Prolog 代码。这是因为这个语言非常适用于这个问题，它很简洁。在 RubyFringe，有很多讨论是关于使用多种语言的，我个人认为这就是生活之道。我们在需要时使用 Erlang，我们使用 Ruby，过去我们也用 PHP，Prolog，在那种情况里，Rails 应用程序生成 Prolog 输入，接着发送到一个正在运行的 Prolog 进程，然后它会取回结果并进行解析。

观看完整视频：

http://www.infoq.com/cn/interviews/galea-ruby-cn

相关内容：

- 如何进行平台型网站架构设计？
- Cells：将组件开发带入Rails 2.3
- Dhanji Prasanna谈Google Sitebricks Web Framework
- Apache Wicket 1.4 发布了
- CodePlex站点的Wiki引擎现已开源

热点新闻 | News

SOA参考架构基础的草案已经提交，正接受公众评阅中！

作者 Boris Lublinsky 译者 马国耀

OASIS的SOA参考架构模型技术委员会最近审批通过了SOA参考架构基础 1.0 版 （SOA-RAF）规范作为公众评阅的委员会草案。：

SOA-RAF基于SOA参考模型（SOA-RM）并定义了一些独立于SOA实施过程中所使用的具体技术、协议以及产品的抽象架构元素。

> 该参考架构基础并非实施 SOA 系统的全景蓝图，也不是指明了实现 SOA 系统时所需的所有技术的技术路线图。然而，它定义了很多关键的概念和组件，它们在任何精心设计的 SOA 系统中都应该出现。为了在实践中使用，构造和管理 SOA 系统，还要做许多其他的设计决定以及技术选择。

SOA-RAF 定义了 SOA 的若干抽象实现，它关注那些能够让 SOA 系统使用，实现并管理起来的元素以及它们之间的关系。其背后的重要假设是，SOA 系统应该包含：

> 分布在不同所有者边界的资源；
>
> 人员和系统互相交互，它们亦是跨越所有者边界的；
>
> 跨越所有者边界的安全，管理和治理；
>
> 人员和系统之间的交互主要是可靠的（适合于计划中的使用及目的）信息交换。

所以，SOA-RAF 不是把 SOA 看成一个独立又复杂的机器，而把它看成一个生态系统：一个由人，机器和服务共同栖息的空间，他们既在实现各自的目标，又在实现整个大社区的目标。

定义 SOA 生态系统的主要原则有如下几条：

> SOA 是行为独立的参与者进行价值交换的媒介；
>
> 参与者（及其利益主体）有理由取得 SOA 中可用资源的所有权；
>
> 参与者的行为及性能受到约定的（从一系列策略和契约中捕获的）规则的制约。

SOA-RAF 分三个视角，正好符合三大主要观点，并反映了对关注点的划分。

> 生态系统视角的观点关注的是人们如何使用 SOA 系统开展他们的业务；SOA 参考架构实现视角的观点关注的是构建 SOA 的最重要方面；而 SOA 拥有视角的观点关心的是那些与 SOA 所有权，管理及控制相关的方面。

InfoQ 有幸邀请到 OASIS SOA 参考模型技术委员会的秘书 Francis McCabe 和主席 Ken Laskey 共同探讨 SOA-RAF。

InfoQ：SOA-RAF 是什么？

FM：它是 SOA 泛型的架构描述。即是对那些让 SOA 生态系统运转起来的关键的概念及其它们之间的关系的描述。

InfoQ：SOA-RAF 的目的是什么？

KL：RM 谈到了一些概念和它们之间的关系，RAF 标明了那些将出现在 SOA 解决方案中的架构元素。但是具体架构仍需要架构的设计，因为你不能从基于 RAF 建立解决方案，但是有了 RAF 的指导，RAF 元素将出现在具体架构中。

InfoQ：SOA-RF 和 Web 服务，REST 服务，SOAML 的关系是怎样的呢？

FM：与具体的技术（如 Web 服务，REST 服务）相比，SOA-RAF 在目的和范围上站在一个更高的层次。我们明确表明要避开具体任何具体的技术。

然而，如果你想知道如何应用 Web 服务，那么 RAF 可以提供重要的指导意见。它表明了使用和混合服务、安全、治理等领域的关键需求以及一个实际的 SOA 生态系统中需要表示的社会结构。

SOA-RAF 没有涉及到的一个方面是，用于构建解决特定问题的 SOA 生态系统时的具体指导。为此，SOAML 是描述实际系统中的具体问题的优秀工具的代表。

InfoQ：SOA-RAF 包含哪些方面？

FM：SOA-RAF 包括三个主要部分或视角。第一个部分的重点是从参与人员以及他们与驱动 SOA 生态系统发展的技术之间的关系的角度，阐述了什么是 SOA 生态系统。第二部分强调了构建 SOA 生态系统的一些关键元素；包括描述的重要性，交互以及策略的重要性角色等。第三部分关心的是如果 SOA 生态系统的拥有问题，这部分强调了 SOA 生态系统的治理、SOA 系统的管理、在一个永不重启的大型系统中测试意味着什么，以及 SOA 生态系统中关键的安全方面。

InfoQ：什么是联合动作（Joint Action）？

FM：为了交付服务，服务消费者和提供者之间的交互是必要的。在 SOA 生态系统中参与者间的交互是被自动仲裁的。这些参与者可能位于不同的所有域中，为了参与者之间的交互，他们必须同时即要单独行动又要行动一致。单独行动指的是交互信息的发送和接收，而行动一致指的是二者都是交互的一部分。

联合动作是任何需要两个或多个参与者参与的动作。一个简单的例子是信息通讯：每一次通讯都需要一个演讲者和一个听众。（尽管任何一方都可能不止一个）。没有双方的参与，就没有通讯：它在本质上本是一个联合。

事实上，在服务提供者和消费者双方的交互中，在很多层面上都应用了联合的概念。在最底层，消息发送和接受自然是一个联合。上一个层次，通讯过程中仲裁的动作（开户，广播紧急信息，购买及销售等）在本质上也是联合的。再上一层，动作经常带入了参与者的社会状态。签定契约修改了参与者的状态：做出了相应的承诺，策略也因此形成。这些社会动作在本质上也是联合的。

InfoQ：该草案和前一个有何区别？

FM：该草案文档中进行了很多优化，特别是生态系统视角做了最多的优化工作。然而，我们也加入了很多重要的部分，如治理和测试。另外，规范的名字本身也有所改变，其增加的“基础”反映了在 SOA 系统的世界中肯定了我们的工作及他人的工作之间的关系。

InfoQ：下一步工作是什么？

FM & KL：由于 RAF 已经非常完善，所以需要更进一步的领域有限。尤其是治理，已经特别完善了。但是我们需要确定如何优化管理的部分，并决定是否将它整合到治理中去，或像现在的草案一样让它独立存在。自从上一个草案开始，在生态系统视角上已经看到了很多工作，并且还有一些重要的讨论要完成。然而，委员会相信目前的工作设计的已经足够充分了，所以这绝对是值得读者提出自己的见解的好机会。

公众评阅从 2009 年 11 月 14 开始，到 2010 年 1 月 13 日结束。这篇草案旨在进一步加强对SOA架构原则的理解并提供一个SOA的实施蓝图。

原文链接： http://www.infoq.com/cn/news/2009/11/RAFSOA

相关内容：

- 围绕递增式SOA的价值的辩论
- Oracle和BEA完成产品集成，融合中间件 11g发布
- 面向服务的虚拟网格简介
- SOA在互联网系统中的应用
- 书摘和访谈：Open Source SOA

热点新闻 | News

JDK 7 出人意料将增加“简单”闭包，发布时间推迟至明年底

作者 Dionysios G. Synodinos 译者 张龙

近日Mark Reinhold在Devoxx的演讲中宣布JDK 7 将增加闭包特性。由于添加了这个饱受 争议的特性，JDK 7 的发布时间将推迟至明年 9 月左右。

Project Coin（旨在对Java 7 进行小幅度的语言改进）的首席工程师Joseph D. Darcy已经确认语言的下一版本将增加某些“轻量级”的闭包：

> ...JDK 7 将增加闭包特性，增加的闭包要比 BGGA（即 Closures for the Java Programming Language，译者注）所提出的闭包更小巧，因此 JDK 7 的发布时间将推迟至明年 9 月左右：

Alex Miller也参加了此次大会，他指出尽管过去社区已经给出了 3 个不同的提案，但Sun对于闭包的态度还是很消极：

> 我真的是无语了。这几年 Sun 总强调大家并没有就闭包这个问题达成共识，也延误了成立闭包 JSR 或专家组的时机，但实际情况却是社区已经给出了 3 个提案，每个提案都有相应的原型。

就在Mark Reinhold宣布这个消息不久，Neal Gafter（已有的 3 个闭包提案中BGGA提案的发起人之一，BGGA其实就是 4 位发起者名字的首字母缩写，他们是Bracha、Gafter、Gosling和von der Ahé）就发布了一个“简化的提案”：

> 该提案有如下变化之处：
>
> - 将控制调用（control invocation）语法移到另一个单独的规范中。

- 将术语闭包字面量（closure literal）替换为 lambda 表达式（lambda expression）。
- 我们检查了 lambda 表达式的语法，从 Clang 借鉴了一些精华。现在有两种形式的 lambda 表达式：expression lambda 拥有控制表达式，而 statement lambda 拥有控制语句。
- 将术语闭包对象（closure object）替换为函数对象（function object）。
- 将术语闭包变换（closure conversion）替换为 lambda 变换（lambda conversion），而控制调用语法将拥有一个单独的块变换（block conversion）。
- 为 return 语句增加新的语义：现在可以从 statement lambda 中返回了。
- 将 java.lang.Unreachable 改为 java.lang.Nothing，这一点借鉴了 Scala。
- 移除对类型名 null 的支持。之前的版本在没有异常抛出时将 null 作为一个异常类型的占位符。现在，类型 Nothing 可以满足这个要求。
- 受限制检查。受限制（restricted）与不限制（unrestricted）的函数类型与闭包概念已经被移除了。现在所有的 lambda 表达式都是受限制的。我们可以通过控制调用语法将此前规范中的不限制闭包传递给方法。
- 增加方法引用支持：我们通过一个新引入的符号#将对方法的引用当作函数看待。该语法来源于 javadoc 中的交叉引用和 FCM 提案。

JDK 7 这种突然间的变化令Fabrizio Giudici等很多人对决策的过程表示怀疑：

> 我不想谈论这么做好不好（你知道在我听说该提案既不是 BGGA，也不是 CICE，而是一个新提案时我就觉得这不是一件好事）。我只是惊诧于仅仅几周后 Java 7 就与 Project Coin（知名的最终 5 提案）达成了一致，某些人几乎是瞬间就改变了主意。这到底是怎样的一个决策过程呢？
>
> 啊哈，我知道了——他们一定是投硬币决定的，现在我终于知道 Project Coin 项目名字的来历了。我担心 Java 7 会变成最混乱的一个 Java 版本——如果你想干掉 Java 的话这么做倒是不错（因为现在大家的争论点还不算太多，比如 Oracle 的收购或是关于 Jigsaw/OSGi 的争论）。

与此类似，Geertjan Wielenga也觉得增加闭包这个决定太出人意料了：

> 如果没人想问决策的过程是怎样的，为何得出这个结论的话，那这绝对是个好消息，或许还是最好的消息呢。首先，我们拥有一堆提案，但根本没人看。其次，突然间我们就拥有了“简单闭包”（我想知道是否现有的所有提案都可以叫做“复杂闭包”呢。难道简单就是闭包的全部么？）。好吧，闭包看起来会很简单，没有非局部的 return（non-local return）、没有控制语句、无法访问非 final 变量。我还是想问一下，到底是如何得出这个决定的？

Cay Horstmann针对这个新的BGGA提案给出了几个用例，与FCM提案何其相似：

> 我们真的不知道 Sun 到底想搞什么。之前我粗略地分析了一下 BGGA 0.6a 提案，总体上来说与现在的 BGGA 差不多。没人反对 BGGA 与 FCM 的相似性。没有非局部的 return、用于 lambda 的#。由于需要在捕获前用@Shared 注解突变变量（mutated variable），人们在编写之前可能需要考虑再三。

```
foreach(@Shared String v : values)
    funcs.add( #() => v);
```

Alex Miller提到JDK发布的延期可能会让其他特性有机会加到最终的发布中，比如ParallelArray程序库，它为映射、过滤以及Java对象数组的裁剪提供了一套函数风格的API：

> 我在 QCon 上与 Bob Lee 聊了一会，他觉得 JDK 7 的延期有可能会将 ParallelArray 纳入进来并使用新的闭包支持。

大家可以在InfoQ上找到关于闭包与JDK 7的更多信息！

原文链接：http://www.infoq.com/cn/news/2009/11/jdk7-simple-closures

相关内容：

- VB 10 中集合与数组的初始值设定项
- JVM上的Python现状
- Scheme语言即将被一分为二
- Spec#与Boogie发布于CodePlex
- .NET反应性框架为事件实现了LINQ

热点新闻 | News

Google Chrome OS细节披露

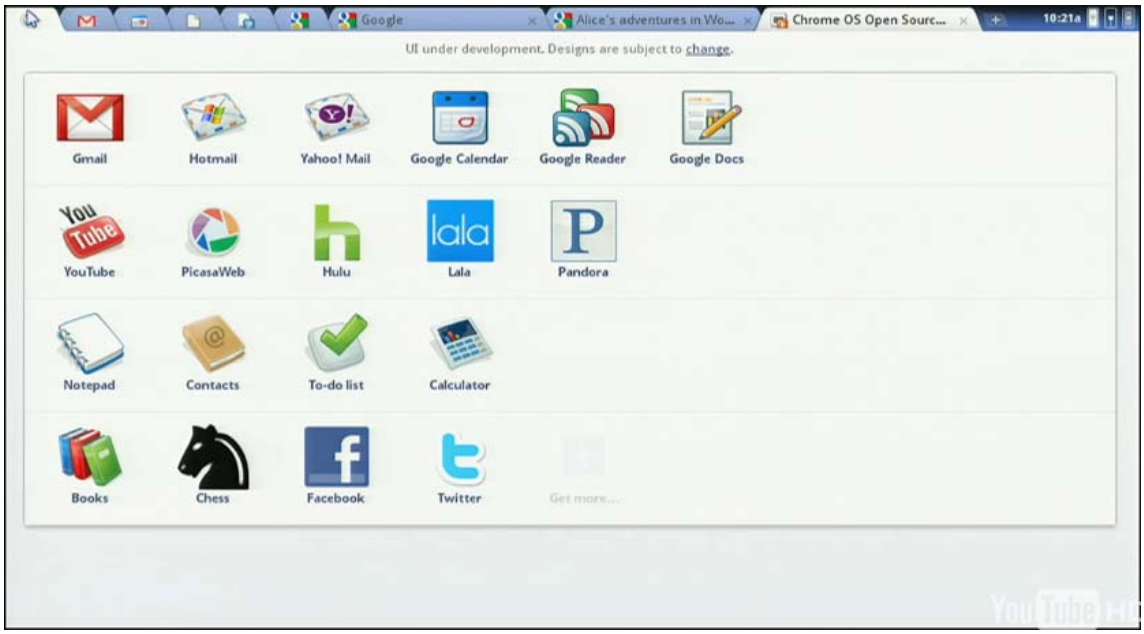
作者 Abel Avram 译者 张龙

Google计划于明年冬季发布Chrome OS，而现在已经将其开源了。目前Google正与制造商合作来制定新的参考硬件标准以支持其速度和安全上的需求，而这正是其新操作系统最关键的特性。

鉴于每个人都知道如何使用浏览器，因此无论是感官还是体验上，Google 都想将 Chrome OS 打造成为浏览器的风格。实际上，用户看到的将是构建在定制版 Linux 操作系统之上的 Chrome 浏览器，而为了简单 Google 向用户完全隐藏了这一切。所有数据都位于云中，并不会永久存储在本地，而由于速度原因，用户所使用的数据副本则保存在本地缓存中，这就意味着编辑一份文档实际上会更新云中的数据。

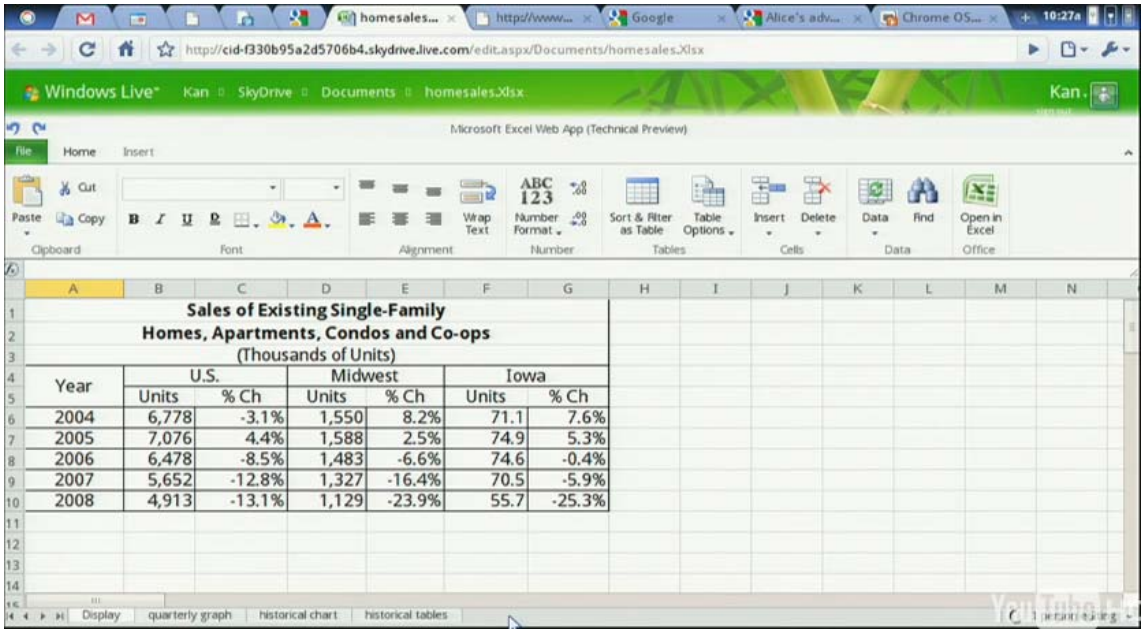
Google 计划于明年正式向市场发布 Chrome OS。当前所用的 UI 到时候可能会发生变化，想法就是将现在的浏览器标签变成应用标签，只要标签被钉上就不会移动，这样用户就可以访问 Email、Calendar、Docs 及其他应用了。他们是最左面的 5 个标签，如下所示：

通过点击左上角的 Chrome logo，用户就可以访问应用菜单了（如下所示），这是一个网页，显示了用户所能访问的所有应用，还可以进行添加：



该页面的 UI 未来可能会发生变化。启动的应用是以轻量级的窗口（叫做面板）形式展现的，即便用户选择了另一个标签，这些应用还可以位于屏幕的最上方，同时用户还能够最小化或是将其关闭。

Chrome OS 是个完全开源、非私有的操作系统。举个例子，要想访问某个存储设备（SD 卡）上的 MS Excel 文件，Chrome OS 将使用 MS Windows Live 来打开它（假如之前已经配置好用该在线应用打开这类文件）。



Chrome OS 上网本并没有硬盘驱动器（HDD），而是采用了基于闪存的内存解决方案或是固态硬盘（SSD）。这么做极大地加快了启动速度，目前的启动速度是 7 秒，而 Google 说还能更快。固件会对优化的内核签名进行验证，如果没问题则加载内核，而后者会加载浏览器。假如签名不正确（比如遭受了病毒侵袭），固件则会下载一个默认的安全内核与 OS，而整个过程全部是自动化的。这种恢复并不会导致系统或应用数据的丢失，也不会丢失设置，甚至连缓存都不会丢失。如果可能的话，Chrome OS 会自动更新到新版本上。

Chrome OS 的安全措施会将运行在标签中的 Web 应用彼此隔离开来，同时也与底层系统进行隔离。处于保护的目的，OS 的文件系统是只读的。用户数据存储在不同的分区上并通过随机数字进行加密。所有的用户数据都与云中数据保持同步，本地存储仅仅用做缓存而已。因此，一旦 OS 受到病毒侵袭或是计算机丢失，用户都可以通过其证书获得另一份数据，可以在极短的时间内加载整个 OS、应用和设置。

目前 Google 正与制造商合作来制定新的上网本硬件标准以支持其速度和安全上的需求。同时 Google 还希望上网本再大一点以支持标准的全尺寸键盘和更好用的触摸板。参考的硬件标准将包括屏幕分辨率信息，用户可以设定更大的分辨率而不像现在的上网本那样。

目前已经支持几个媒体和文档编辑应用了。视频播放、图像查看以及文档编辑软件已经可用了，任何人都可以使用大量的在线应用。不管怎样 Chrome OS 都不会变成私有。

代码是开源可下载的，在Chromium OS名下，Google也和每个人一样使用着同样的代码主干，不过其产品叫做Chrome OS。Google还开放了其设计文档，披露了未来的计划，也欢迎每个有志之士参与进来。

参考资源：Chrome OS介绍视频、源代码、UI体验及设计文档。

原文链接：http://www.infoq.com/cn/news/2009/11/Google-Chrome-OS-Details

相关内容：

- Helios使用卫星内核处理异构环境
- Windows 7 RTM已经可以下载
- HyperSpace：一个精巧的浏览环境
- 微软的浏览器操作系统
- MINIX 3 承诺比Windows或Linux更安全

热点新闻 | News

Google正制订一项新协议，旨在替换掉HTTP

作者 Abel Avram 译者 张龙

近日Google提出了SPDY——运行在SSL上的一个应用层协议，该协议旨在替换掉HTTP，而后者被认为会产生延迟。Google已经使用Web服务器与增强的Chrome浏览器开发了一个原型，结果是页面加载速度比以前快了 2 倍。

前一阵Google发出的倡议“让Web变得更快”旨在提高Internet的速度。该倡议涵盖了几个领域，从构建更快的Web服务器到更快的浏览器。例如，Page Speed就是个用来提高网页下载速度的工具。Google在开源大量工具的同时又发布了相关的教程以帮助全世界的开发者加快其Web站点的速度。

然而 Google 并没有裹足不前，他们开发了一个名为 SPDY(发音为 SPeeDY)的新的应用协议，一旦成功并得到广泛应用的话，SPDY 就会替换掉 HTTP 并彻底颠覆整个 Internet。SPDY 白皮书说要向协议栈下面渗透并替换掉传输层协议（TCP），但 Google 也认识到这样的话部署起来会 相当困难，因此他们打算对应用层协议 HTTP 进行改进。

SPDY 白皮书提到了 HTTP 协议中的很多限制，尤其是页面传输时的延迟：

- 一个请求一个连接。由于 HTTP 一次只能获取一个资源（HTTP 管道很有用，但也只是强制形成一个 FIFO 队列），因此 500 毫秒的服务器端延迟导致无法 重用 TCP 通道来处理其他请求，而浏览器则通过发送多个连接来解决这一问题。自从去年以来，浏览器已经将每个域的连接由 2 个调整为了 6 个。
- 只有客户端才能发送请求。在 HTTP 中，只有客户端才可以发送请求。即便服务器端知道客户端需要某个资源，那它也没有办法通知客户端而只能等待其发出对该资源的请求。
- 未压缩的请求与响应头。如今的请求头的大小差别很大，从 200 bytes 到 2KB 的都有。随着应用越来越多地使用 cookie 和 user agents 扩展特性 ,700-800 bytes 的头大小已经变

得很常见了。对于 modem 或是 ADSL 连接来说，上行带宽都很低，这种延迟就变得很可观了。降低头中的数据可以改进发送请求的 序列化延迟。

- 冗余的头。此外，有几个头会跨越请求在同一个通道上重复发送。像 User-Agent、Host 和 Accept*这样的头基本都是不变的，没必要重复发送。
- 可选的数据压缩。HTTP 对数据进行可选的压缩编码。内容总是应该以压缩格式发送。

SPDY 的一个目标就让页面加载时间降低 50%，同时将浏览速度提升一倍。对一个用户来说，几百毫秒不算什么 ,但每一毫秒都会对未来高度互联的 Web 应用产生积极的影响。当前 Web 上的内容并不会受到该协议的影响 ,只有 Web 服务器和客户端需要增强以充分利用该协议。

Google 究竟想用 SPDY 做什么呢？

- 在单个 TCP 会话上执行多个并发的 HTTP 请求。
- 通过压缩头以及减少不必要的头来降低当前 HTTP 所占据的带宽。
- 定义一个易于实现且提升服务器效率的协议。我们希望通过砍掉一些边缘情况来降低 HTTP 的复杂度并定义易于解析的消息格式。
- 将 SSL 作为底层传输协议以达到更好的安全性且兼容于现有的网络基础设施。尽管 SSL 引入了延迟，但我们相信从长远来看，Web 还是要依赖于安全的网络连接的。此外，要想保证跨越现有代理的通信不被破坏，SSL 也是必须的。
- 让服务器可以主动与客户端通信并向客户端发送数据。

为了达成这些目标 ,Google 在 SSL 之上增加了一个会话层来实现 SPDY ,这考虑到了“单个 TCP 连接之上会有多个并发、交错的流”。HTTP GET 和 POST 消息格式保持不变，但 SPDY 提出了一个新的“帧格式用于在线路上编码和传输数据”。流是双向的，客户端与服务器端都能开启流。

到目前为止，Google构建了一个既能处理HTTP协议，也能处理SPDY协议的内存服务器，代码将在不久后开源。还有一个修改的Chrome版本（内部的暂定名为flip，也是开源的）运行在HTTP和SPDY之上。他们还开发了一套测试工具集来保证使用SPDY后页面仍能正确下载。这些工具也将于不久之后发布。

原型表明目前的结果还是很不错的：

> 我们通过模拟的家庭网络连接下载了百强网站的 25 个，只有 1%的包丢失。每个站点都被下载了 10 次并计算平均的页面加载时间，然后计算所有站点的平均时间。结果表明普通 TCP（没有 SSL）上的页面加载时间要比 HTTP 提升了 27% - 60%，而 SSL 要提升 39% - 55%。

纵然提出了最好的协议，但显然 Google 无法凭借一己之力取得成功，还需要依靠社区的推动与努力来创建一个全新的应用层协议。其他公司对此有何反应呢，让我们拭目以待。

参考资源：SPDY协议草案规范，可以使用SPDY的Chrome。

原文链接：http://www.infoq.com/cn/news/2009/11/Google-SPDY-Replace-HTTP

相关内容：

- 网络安全：采访Intel安全与加密经理David Durham
- MIME给REST的采用带来了问题？
- JOSH：企业软件组合的新提议”
- 使用Rack::Cache进行平滑的HTTP缓存
- Ruby的Net::HTTP怎么了?

热点新闻 | News

InfoWorld评 2009 年十大新兴企业级技术

作者 霍泰稳

InfoWorld最近推出了2009 年十大新兴企业级技术排名 跨平台移动应用开发、NoSQL数据库、重复数据删除以及桌面虚拟化位列其中，而分布式处理编程框架MapReduce位列第一位。

MapReduce 是 Google 提出的一个软件架构，主要用于大规模数据集的并行运算，它通过把对数据集的大规模操作，将其分发给网路上的每个节点实现 可靠性。在 Google 内部，MapReduce 得到广泛的应用，比如分布排序、Web 连接图反转和 Web 访问日志分析等。提到为什么将 MapReduce 放在第一位，InfoWorld 解释说：

> 某种程度上来说，这是考虑到 MapReduce 的独特创新，它使得从前只能在大型商业硬件上所做的事情，在普通的 PC 机上即可操作——处理千兆级别的数据。在亚马逊的 Amazon Elastic MapReduce 产品中，以 Web 服务的方式很好地应用了 MapReduce 的实现——Apache Hadoop。而且，MapReduce 还被集成进一些来自 IBM、Oracle 等公司的主流解决方案，现在它们云计算所用的服务器中可能就跑着 MapReduce。

近几年来，移动设备上的企业应用一直没有得到很好地普及，主要有两个原因，一个是因为开发者需要耗费大量的时间去学习如何为智能设备编程，另外是因为不同 设备间的应用移植性不好。这也是为什么 InfoWorld 认为“跨平台移动应用开发”是 2009 年新兴企业技术的主要原因：

> 跨平台移动应用开发环境，比如 Rhomobile 的 Rhodes 框架，可以让开发者写一次应用，但能在多个不同的设备上运行，如 iPhone、Windows Mobile 和 BlackBerry 等，提供了很多很炫的功能。移动企业级开发的时代快要来临了！

在数据处理方面，十大新兴技术中包含了两个，一个是NoSQL数据库， 一个是重复数据删除。对于NoSQL数据库这个概念，在刚提出时就遇到了很大的挑战，因为目前几乎所有的大型应用采用的都是关系型数据库，或者说SQL数 据库。NoSQL的推出，无异于一场革命。InfoWorld给出的解释是，NoSQL数据库对于那些如安全日志或者系统日志等结构化不强的数

据而言，使 用起来很顺手。另外，因为缺少对数据的控制，NoSQL数据库处理数据的速度也很快。而对于重复数据删除这个问题，虽然目前还没有很好的应对方案，但是未 来它会在数据存储领域扮演越来越重要的角色。

其他新兴技术还包括桌面虚拟化、I/O 虚拟化、固态硬盘（Solid-state Drive ）、多核芯片、硬件电力管理、白名单（Whitelisting）等。

原文链接：http://www.infoq.com/cn/news/2009/11/infoworld-top10-emerging

相关内容：

- 支持云应用程序服务的PHP API
- 别删除数据
- SQL Server的未来之路
- 连贯NHibernate正式发布 1.0 候选版
- 解密C#-SQLite是如何移植的

热点新闻 | News

欧盟委员会发表正式声明——拒绝Oracle对Sun的收购计划

作者Charles Humble译者 张龙

Sun安全与交易委员会的一份文件称本月 9 日欧盟委员会发表了一项正式声明——拒绝Oracle对Sun的收购计划。如大家所想的一样，该项声明仅限于Oracle对MySQL的收购。文件称：

> 此项声明代表了委员会的初步评定情况，只限于 Sun 的开源数据库产品 MySQL 与 Oracle 的企业数据库产品的合并及其给数据库产品市场的竞争所带来的负面影响。虽然如此，相关人等可以就欧盟委员会的此项声明提起诉讼。这项声明只是一个初步评定，并不代表欧盟委员会的最终决定。Oracle 依旧可以对欧盟委员会的最终决定向欧洲一审法庭提起诉讼。

欧盟委员会拒绝此项收购并不令人感到意外。上月委员会就曾对 Oracle 发出了警告，委员会发言人 Neelie Kroes 说到：

> 尽管一再请求，但委员会对 Oracle 还是深感失望，因为 Oracle 无法提供有力的证据表明此间并不存在竞争问题，也没有对委员会所提出的竞争问题给出解决办法。

欧盟委员会进一步提到假如其对 MySQL 的担忧得以解决，那他们还是很愿意批准此项收购的。

目前Oracle并没有表现出悔过的迹象。其措辞严厉地向欧盟委员会发难：MySQL创建者Monty Widenius和自由软件基金会创建者Richard Stallman都在欧盟委员会任职，但这帮人却不懂什么是开源软件：

> 对开源软件有深刻理解的人都明白这一点，因为 MySQL 是开源的，任何人都无法独占它。这就是开源的全部。

Oracle 继续说到数据库市场“竞争非常激烈，至少有 8 个重量级选手，这包括 IBM、微软、Sybase 以及 3 个开源供应商”。

尽管遭遇了欧盟委员会的拒绝，但 Oracle 的态度依旧乐观：

> 由于对竞争的害处缺乏让人信服的理论和证据，我们相信最终一定能如愿以偿地达成这笔交易。

这个情况导致了大西洋两岸在反托拉法案上截然不同的处理结果：美国司法部已经在今年 8 月批准了此项收购。近日司法部表明了自己的立场：此项收购并没有违背反托拉斯法。反托拉斯法部门的助理律师Molly Boast说到：

> 对用户造成损害是无从谈起的，因为他们依旧可以从各种数据库产品中选择自己所需的。

原文链接：http://www.infoq.com/cn/news/2009/11/eu_formal_objections

相关内容：

- Ellison爆料欧洲委员会的调查将使Sun每个月损失 1 亿美元
- Atlassian收购GreenHopper，在JIRA中加入敏捷项目管理特性
- Oracle如何处理Sun的开源资产
- SpringSource收购Hyperic
- Stallman致信欧盟要求Oracle放弃MySQL

热点新闻 | News

处理.NET中的内存泄露

作者Abel Avram 译者 王瑜珩

Fabrice Marguerie是一位软件架构师和咨询师，他在MSDN发表了如何检测和避免.NET程序内存与资源泄漏的文章。此文章描述了编写.NET程序时可能发生的内存与资源泄漏，以及如何避免这些泄漏。

C#这样的编程语言使用垃圾收集器来清理内存，对于程序完全不会再访问的内存，本应是没有内存泄漏的。Fabrice 称，内存泄漏发生在一块内存不 再被使用，但却依然被程序所引用时。当一块内存无法被程序访问到时，垃圾收集器将会重新分配这块内存，但是如果程序仍然保持对内存的引用却不使用这块内存 时，就会造成内存泄漏。

Fabrice 还列举了一些可能泄漏的系统资源：

- 用于窗口管理的用户对象，包括快捷键表、符号、光标、钩子、图标、菜单和窗口。
- 用于图形的 GDI 对象：位图、画刷、设备环境（DC)、字体、内存 DC、元文件、调色板、画笔、区域等。
- 用于内存管理、进程执行和进程间通信（IPC)的 Kernel 对象：文件、进程、线程、信号、定时器、访问令牌、套接字等。

这些资源都是有限制的，注册表中的GDIProcessHandleQuota和 USERProcessHandleQuota键保存了单个进程可用的最大GDI对象和用户对象数量，默认值是 10000。虽然这个数字对于大多数程序足 够了，但如果使用的过多则可能会达到另一个限制，一个Windows session最多只能有 65536 个句柄。Fabrice说这个限制很容易就会达到。他的结论是，要小心使用和释放系统资源。

Fabrice 列举了一些内存泄漏的根本原因，以及是如何造成泄漏的：

- 使用静态引用

- 未退订的事件 - 作者认为这是最常见的内存泄漏原因
- 未退订的静态事件
- 未调用 Dispose 方法
- 使用不彻底的 Dispose 方法
- 在 Windows Forms 中对 BindingSource 的误用
- 未在 WorkItem/CAB 上调用 Remove

作者在文章中还提供了一些避免内存泄漏的建议：

- 对象的创建者或拥有者负责销毁对象，而不是使用者
- 当不再需要一个事件订阅者时退订此事件，为确保安全可以在 Dispose 方法中退订
- 当对象不再触发事件时，应该将对象设为 null 来移除所有的事件订阅者
- 当模型和视图引用同一个对象时，推荐给视图传递一个此对象的克隆，以防止无法追踪谁在使用哪个对象
- 对系统资源的访问应该包装在 using 块中，这将在代码执行后强制执行 Dispose

Fabrice最后介绍了一些工具来对付泄漏：GDILeaks（EXE）、dotTrace、.NET Memory Profiler、SOS.dll和WinDbg。

原文链接：http://www.infoq.com/cn/news/2009/11/Memory-Leaks-.NET

相关内容：

- 严重内存泄漏困扰WPF
- JProbe 8.0：Java代码、内存及覆盖率分析王者回归
- 使用BleakHouse发现Rails应用的内存泄漏
- .NET内存泄漏
- 垃圾回收器在CLR 4 中的改进

热点新闻 | News

雅虎将Traffic Server捐献给Apache基金会

作者Abel Avram

雅虎云计算资深副总裁Shelton Shugar ,在云计算会议所作的主题演讲上 ,宣布将旗下的HTTP缓存服务器Traffic Server(TS)捐献给Apache基金会。

在 2003 年与Inktomi一同被买下的Traffic Server ,一直被雅虎当作HTTP/1.1 代理服务器用于生产中，每天服务接近 300 亿的web对象与超过 400TB的数据。TS在单个 8 核心的机器上能达到 35000 的RPS。历经几个月作为Apache孵化提案，TS目前成为了孵化项目，目标是成功孵化为Apache项目，如同Hadoop所经历的那样。Hadoop是由Google的MapReduce启发而来的项目，雅虎对这一项目作出了相当大的贡献。

除了作为缓存服务器之外，TS还提供了会话与配置管理，负载均衡，权限验证与路由等等功能。它有一个外部API ,并可以通过插件来扩展。TS将会抓住那些云供应商的眼球 ,按照Shugar的说法：

> 会对它产生兴趣的将会是那些构建云和使用这些服务的人们。他们将会理解 Traffic Server 的价值所在以及他们可以用它达到的效果。从伸缩性的角度来讲，能与之相提并论的确实很少。

Cindy Borovick，Datacenter Networks 的研究副总裁，这样评论 TS 的重要性：

> 这都是围绕着构建一个吸引开发者的雅虎云平台。通过将 traffic server 开源，雅虎表明他们对于网络基础设施非常认真，并且他们的平台有网络，能够创造更好的用户体验，其结果就是提升用户的信心，紧密关系和利润。这并不是败着，而是真正的价值增加。

雅虎开源的 TS 的初衷是加入 64 位支持，将其移植到其它的类 Unix 环境上(TS 目前只能运行在 Linux 上)，加入新的特性比如 CARP，HTCP，ESI，原生 IPv6，并进一步提高其性能。

开发TS的原始团队是雅虎的员工，但其它的开发者早已对参与这一项目表达了意向。源代码除了像BDB，OpenSSL，TCL，STL，glibc和expat等等的标准库以外没有其它任何的外部依赖。

代码还带了一份管理员指南和SDK 编程指南。

原文链接：http://www.infoq.com/cn/news/2009/11/Yahoo-Traffic-Server-Apache

相关内容：

- XMemcached——一个新的开源Java memcached客户端
- 《OSGi原理与最佳实践》书评
- Apache Lucene 2.9 的改进
- 社区反应：IntelliJ开源，亡羊补牢？
- Google与某些Android开发者之间产生了摩擦

热点新闻 | News

微软将向Eclipse开发者提供大量工具

作者Abel Avram译者 张龙

近日微软宣布将向Eclipse开发者提供大量工具，包括Windows Azure Tools for Eclipse、Windows Azure SDK for Java、Eclipse Tools for Silverlight和一个类似于Windows 7 的Eclipse UI。

微软已经与Tasktop Technologies（Eclipse Mylyn的创建者）合作改进Eclipse以充分利用Windows 7 的新特性：使用任务栏进度条和跳转列表、 搜索部件集成、新的部件颜色和样式等以赋予Eclipse Windoes 7 的感官。这些增强将基于Eclipse Public License，最初的预览版将于明年一季度发布而最终版则将于明年 6 月Eclipse Helios发布时呈现给大家。微软贡献了自己的技术以为运行在Windows 7 上的Eclipse创建全新的界面。对该项目进展感兴趣的读者可以查看一下Eclipse Bug 293226。

为Windows Azure编写应用的PHP开发者从Windows Azure Tools for Eclipse（WindowsAzure4e）中获益最大，这是基于PHP Development Toolkit（PDT）的一套Eclipse插件，提供了如下功能：

- **项目创建与迁移**:全新的项目向导提供了一个用于开发 Windows Azure 的 PHP Web 应用。可以通过迁移工具将现有的 PHP 项目转换为 Windows Azure 项目（反之亦然）。
- **Azure 项目结构与管理**:windowsazure4e 插件可以创建 Windows Azure 所需的项目文件，包括一个 Windows Azure Service 项目和 Web-role 项目,同时还有 Windows Azure 配置与定义文件。可以通过 Eclipse 的属性窗口查看项目与 Windows Azure 的设定情况。
- **存储浏览器**：作 为插件的一部分，Windows Azure Storage Explorer 也出现在了 Eclipse 环境中。Storage Explorer 可以轻松管理 Windows Azure Storage 帐号。除此之外，它还提供了友好的用户界面以对 Blobs、Queues 及 Tables 执行创建、读取、更新及删除（CRUD）操作。
- **Azure 项目部署**：一旦在 Windows Azure Development Fabric 本地开发完 Windows Azure PHP 项目并测试过后就可以将其打包并部署到 Windows Azure 上了，方式很简单，就是

在 Eclipse 的项目上点下右键就搞定了。

存储浏览器是用Windows Azure SDK for Java开发的，其中位于法国的Soyatec公司（该公司开发了大量基于Eclipse的工具）也帮了不少忙。另一个项目WindowsAzure4j向Java开发者提供了与Windows Azure进行交互的必要工具。其主要特性列举如下：

- 用于操纵 Windows Azure Blobs、 Tables 及 Queues 的 Java 类（主要用于 CRUD 操作）
- 用于 HTTP 传输、认证/授权、 REST 及错误管理的辅助类
- 管理、 Instrumentation 及日志支持
- 支持在 Azure Table Storage 中存储 Java Session

与Soyatec合作开发的另一个项目是Eclipse Tools for Silverlight（eclipse4SL），该项目早在一年前就宣布了，直到今天才发布。这套工具提供了如下功能：

- **增强的交互性**:开发者可以在 Eclipse 中构建 Silverlight 应用，应用可以通过 REST、SOAP、JSON 及其他标准与 Java Web Services 协同工作。
- **Silverlight 项目系统与 Silverlight 编译器**：Eclipse 将提供用于创建 Silverlight 应用和媒体体验的高级项目系统以及用于打包 Silverlight 应用以进行部署的编译器。
- **具备代码提示与代码完成功能的 XAML 编辑器和预览器**：Eclipse 将提供高级的、兼容于标准的 XAML 编辑器，编辑器具备代码提示和代码完成功能，这有助于检测并纠正编码错误。
- **完全兼容于微软的开发和设计工具**:Microsoft Visual Studio 和 Microsoft Expression Studio 工具将完全支持由 Eclipse 创建的 XAML 和 Silverlight 项目。

eclipse4SL 1.0 提供了如下特性：

- Silverlight 2.0 支持
- 具备语法着色、关键字和模板代码完成功能的 C#代码编辑器
- 自动运行和构建
- 可配置的 Web 应用启动设施
- Silverlight 项目系统与 Silverlight 编译器 :用于创建 Silverlight 应用和媒体体验的高级项目系统

- XAML 编辑器和预览器:兼容于标准的 XAML 编辑器，编辑器具备代码提示和代码完成功能，这有助于检测并纠正编码错误。
- 移动与重命名
- 高级的媒体特性
- 跨平台能力（Mac 版本）
- 完整的用户文档与规范的指南
- 缺陷与衰退测试
- 开发者可用性测试

eclipse4SL 2.0 的路线图已经出来了 ,该版本将于明天春季发布 ,包括如下功能 :支持 Silverlight 3.0、支持浏览器外体验、对 Mac 平台支持的改进以及支持多项目开发。

原文链接：http://www.infoq.com/cn/news/2009/11/Eclipse-Tools-Windows-Azure

相关内容：

- Eclipse社区调查结果发布
- 伦敦Eclipse DemoCamp活动上的新技术展示
- Eclipse Galileo发布啦
- Google发布Eclipse插件
- IBM developerWorks十周年：精彩内容回顾

热点新闻 | News

Oracle宣布对MySQL、GlassFish、NetBeans的未来计划

作者 霍泰稳

在Oracle官方的一个关于收购Sun的常见问题文档里，罗列了其收购Sun之后，对一些产品的未来计划，包括MySQL、GlassFish、NetBeans等。虽然，这一计划并没有太多的合同约束，但是还是多少让人看到Oracle对这些产品的支持。

在上周InfoQ的“Stallman致信欧盟要求Oracle放弃MySQL”的报道中，自由软件代言人Stallman提到“如果允许Oracle收购MySQL，它肯定会限制MySQL软件平台功能和性能上的发展，从而给使用MySQL软件的人们带来巨大的伤害。”。Oracle则表示会比Sun还要支持MySQL来让用户放心：

> Oracle 计划比 Sun 投入更多的资金用来开发 MySQL。在收购结束后，Oracle 希望继续开发和提供开源的 MySQL 数据库。Oracle 计划将 MySQL 加入到自己现有的数据库产品线，该产品线目前已经包括开源数据库 Berkeley DB。另外，Oracle 现在还提供了开源的事务存储引擎 InnoDB，这也是 MySQL 里面最重要和应用最广泛的事务引擎。现在 Oracle 也已经将 MySQL 作为我们企业级 Linux 的一部分发行了。

考虑到 GlassFish 是 Java EE 很好的参考实现，不出意外，Oracle 也决定继续维护 GlassFish：

> Oracle 计划继续维护 GlassFish Enterprise Server 将其作为 Java EE 规范的开源参考实现，也会努力地支持 GlassFish 社区。另外，Oracle 计划在整合 Oracle WebLogic Server 和 GlassFish Enterprise Server 的通用基础组件和创新方面加大投入，以更好地服务于两者的用户。

关于Oracle对NetBeans的声明更值得耐人寻味，你会发现在对NetBeans的说明中没有像对MySQL那样所谓的“比Sun投入更多的资金 ”字样。另外因为此前Oracle已经有JDeveloper和基于Eclipse的开发包，所以NetBeans何去何从，也一直是Java开发者所比 较关注的。在此前专业咨询人员、讲师、软件架构师Adam Bien在他的博客中就曾列举了8 个Oracle应该继续支持NetBeans的理由，比如NetBeans体积小、采用率高、支持Java FX、可视化设计和报表出色等。

在文档中，Oracle提到：

> 同样，我们希望 NetBeans 也能像现在 Oracle 已经提供的两款免费企业级 Java 开发工具（Oracle JDeveloper 和 Oracle Enterprise Pack for Eclipse）一样，成为又一个开源的选择。对 Oracle JDeveloper 来说，它依然是开发 Oracle Fusion 中间件产品以及下一代企业级应用的官方指定工具，但是开发人员如果想开发纯 Java 和 Java EE 系统的话，那么就可以选择他们自己趁手的免费工具了：JDeveloper、Enterprise Pack for Eclipse，或者 NetBeans。

值得一提的是，虽然这个计划里多是一些美好的愿景，但是在文档的末尾，Oracle 也提到这只是他们产品的一个大概方向，只是为了提供更多的信息，没有任何合同约束，也不能作为采购决策的依据等等。也就是说，将来这些“愿景”也可能会落空。

原文链接：http://www.infoq.com/cn/news/2009/11/plan-of-mysql-glassfish-netbeans

相关内容：

- WebSphere 2009 年发展方向
- JSF在GlassFish管理控制台中的应用
- Spring的IoC容器
- OSGi原理与最佳实践（精选版）
- Oracle宣布对MySQL、GlassFish、NetBeans的未来计划

InfoQ中文站

www.infoq.com/cn

我们的**使命**：成为关注软件开发领域变化和创新的专业网站

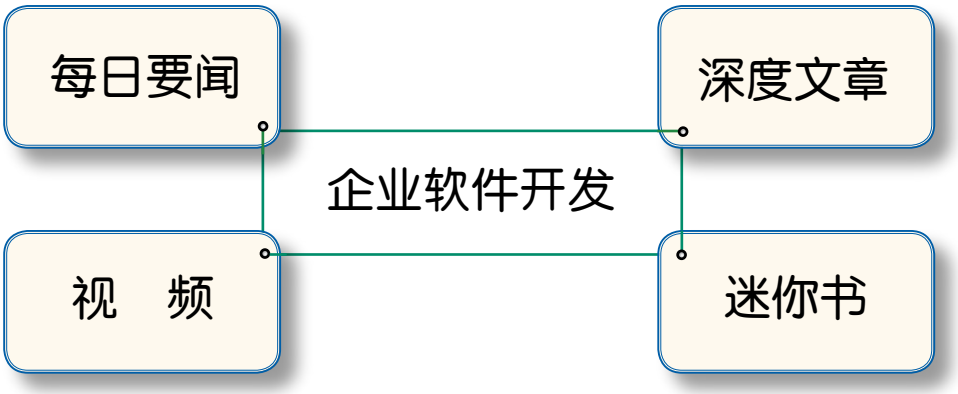
我们的**定位**：关注高级决策人员和大中型企业

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

我们的**特质**：个性化RSS定制、国际化内容同步更新

我们的**团队**：超过30位领域专家担当社区编辑

……



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

推荐文章 | Articles

Web开发必知的八种隔离级别

作者 James Leigh 译者 曹如进

ACID 性质是数据库理论中的奠基石，它定义了一个理论上可靠数据库所必须具备的四个性质：原子性，一致性，隔离性和持久性。虽然这四个性质都很重要，但 是隔离性最为灵活。大部分数据库都提供了一些可供选择的隔离级别 ,且现在许多库都增加了附加层来创建颗粒度更细的隔离。隔离级别应用范围如此之广主要是因 为放宽隔离约束往往会使得可扩展性和性能提高几个数量级。

串行一致性是可用的最古老最高的隔离级别之一 ,它之所以倍受青睐是因为其提供的简单编程模型 ,即每次仅能有一个事务对给定的资源进行操作 ,这就避免了很多 潜在的资源问题。尽管如此，大部分应用程序（尤其是 Web 应用程序）都不采用这种级别非常高的隔离，因为从终端用户的角度来看这是不切实际的-任何一个拥 有大量用户群的应用程序在访问共享资源时都将会有几分钟的延迟 ,而这会使得用户量迅速减少。弱一致性和最终一致性在大规模分布式数据源中，例如 Web 中， 随处可见。好几个成功的大型 Web 应用（例如，eBay 和 Amazon）都显示出乐观的（optimistic）弱一致性要比传统悲观的 （pessimistic）机制在扩展性方面好得多。本文将一窥八种不同的隔离级别。学会适当的放宽数据一致性的约束，你可以在自己的应用程序中使用这八 种隔离级别来获得更好的性能和可扩展性。

并发控制的主要目标是为了确保事务被隔离且不会影响到其他事务。要达到高级别的隔离需以牺牲性能为代价。并发控制可以用悲观或者乐观的机制来实现。大部分 关系型数据库都使用了悲观机制来实现写入优化。悲观机制采用了锁 ,通过使用锁它可以阻塞一些操作或者进行某些形式的冲突检测。当一个表格，页面或是行被修 改后，悲观机制中的锁可以用来阻塞其他潜在的访问修改资源的事务。然而，乐观机制并不采用任何锁，它仅仅依赖于冲突检测来维护事务隔离。乐观机制采用的冲 突检测可以允许所有的读操作，并在事务结束时检验其一致性。如果检测到冲突，那么事务会进行回滚或重做。大部分 web 服务器都是读入优化，因此使用了乐观 机制。通过允许所有的读入操作，乐观机制既可以保证很高的读写吞吐量，也可以在资源不是一直改变的情况下保证数据的一致性。

下面列出的隔离级别是用来帮助 Web 开发人员更好的理解他们编程模型中放置的约束，帮助系统架构师和开发人员共同讨论如何在保持必要的数据完整性的同时选择最有效的隔离级别。它们按照最少隔离（未提交读）到最多隔离（串行化）的顺序列出。

未提交读（Read Uncommitted）

未提交读隔离级别需要事务间很少的隔离。每一个读操作都能看到事务中等待的写操作（脏读）。然而已经提交的写操作必须要有一个串行顺序来防止脏写。悲观机 制会阻塞有冲突的写操作直到其他写操作已经被提交或已经回滚。乐观机制不会锁住这些操作，它会允许所有的操作都通过。如果一个连接进行了回滚，那么接下来 修改同一块数据的其他操作也会被回滚。在这种级别中，共享缓冲可以不加验证的进行使用。这种隔离级别最好在不需要事务（比如只读的数据集），或者事务只在 独占数据库时才修改的情况下使用。

例子：一个只在离线情况下更新的档案数据库，或者不在事务中使用的审核/登陆（audit/logging）表。

已提交读（Read Committed）

已提交读可以读取系统中任何已经提交的状态，并且可以不加验证（混合状态）的进行缓冲，只需当前连接中发生的改变能够反映到结果中即可。悲观机制将其实现 为单调视图。乐观事务则隔离存储所有的改动，使得它们直到提交后才可用。读已提交使用一个非常乐观的机制，它推迟写入所有的变化直到事务被提交为止。这种 形式的乐观隔离可以在不阻塞读操作的情况下实现复杂的写入操作，并且它没有验证模式。共享缓冲只能在已提交的状态中使用。这种隔离级别最好在结果可以使用 旧值，且事务只能用于写入操作的情况下使用。

例子：一个不必显示当前最新帖子的在线论坛，且它的帖子间数据不相冲突。

单调视图（Monotonic View）

单调视图是对读已提交的一个扩展，它其中的事务在执行时会观察数据库中一个单调上升的状态。在这种级别中，如果有明显的写入事务，那么悲观事务会在读入操 作中被阻塞。乐观事务会像在读已提交中一样操作，隔离保存所有的改动，并且会验证它们的缓冲以确保其仍然合法。这种级别可以定期地同步数据库副本，且最好 在不需要事务或者仅存在写操作事务的情况下使用。

例子：一个仅能由一个人来修改的用户偏好表。

快照读取（Snapshot Reads）

快照读取扩展了单调视图，它可以保证查询结果都能反映到数据库一致的快照中。悲观机制会在读操作时阻碍其他影响结果的写入操作。乐观机制则允许其他的写入 操作，并通知读取事务某部分已经发生改变并进行回滚。想要实现一个乐观机制，必须在读操作结束之前验证是否有什么并行的写入操作修改了结果，如果有的话， 那么结果可能会重做或回滚。这个检验过程可能只是简单的检查同一张表中是否出现了写入操作，或者只是检查改动的查询结果。乐观隔离级别可以很轻松地检测出 冲突，并且在允许并发读入操作的过程中，支持写入操作。这种级别只要能够读取到快照，便可以定期地同步数据库副本。最好在写入操作很少，不想与读入操作冲 突，且查询结果需要一致性的时候使用这种隔离级别。

例子：一个查询比修改频繁，且只保留最新值的货币换位表或者查询表。

游标稳定性（Cursor Stability）

游标稳定性隔离扩展了读已提交，并且是许多关系型数据默认的隔离级别。在这种隔离级别中，悲观事务如果在一个单独的语句中执行的话，必须得指定它将修改的 记录。这通常可以在"SELECT"查询后附加“FOR UPDATE”关键字来实现。在这种情况下，其他冲突的读写悲观事务都将被阻塞直到该事务结束为止。乐观事务会跟踪提交时被验证的所有修改记录/实体的版 本号。这是一种很流行的乐观隔离级别，因此被所有的主流对象关系映射库支持。在 Java 持久性 API 中，可以使用 FLUSH_ON_COMMIT（尽管查 询可能不影响本地改动）来接近达到这种级别，且如果检测到冲突的话，可以抛出 OptimisticLockException 异常。这种隔离也同样可以用在 HTTP 头域的 If-Match 或者 If-Unmodified-Since 中，它可以用来在更新前对比上一个资源的版本或者时间戳。这种级别最好在实体由外部信息（不从数据库中读取）更改，或者改动不会彼此覆盖的情况下使用。

例子：一个共享的公司目录或者一个 wiki。

可重复读取（Repeatable Read）

可重复读取级别扩展了游标稳定性，它保证事务内的任何数据在事务过程中都不会被修改或者移除。悲观事务需要读取所有记录上的锁，并阻塞其他服务来修改这些 记录。乐观事务则会跟踪所有的记录或者实体，并检查它们是否在提交时被修改过。这种级别最好在实体状态能够影响其他实体，或者事务由读写操作构成的情况下使用。

例子：一个订单跟踪数据库，它从一个实体中读取值并用它来计算其他的实体值。

快照隔离（Snapshot Isolation）

快照隔离扩展了快照读取和可重复读取，它保证事务中所有进行的读操作都能看到数据库中一致的快照。事务执行的的任何读操作都会有相同的结果，而不管它们在 事务中执行的早晚。这和可重复读取不同，因为快照隔离能够防止幻读（查询结果不断变化）。许多关系型数据库采用多版本并发控制（也可以叫做 SERIALIZABLE）来支持这种级别，实现方法是通过锁和冲突检测的组合。在这种级别中，考虑到它可能与悲观机制或者乐观机制相冲突，因此事务一定 要做好回滚的准备。悲观机制会通过锁住资源来尝试减少冲突的机会，但是必须在事务提交后将这些改动合并。乐观机制也会使用多版本并发控制，但是它不会阻塞 其他可能产生潜在冲突操作的事务，反而是将冲突的事务进行回滚。这种级别的隔离最好在事务可以读取和修改多个记录的情况下使用。

例子：一个基于系统状态规则的工作流系统。

可串行性（Serializability）

串行性是快照隔离的扩展，它要求所有的事务都必须一个接着一个的出现，就好比它们被串行化过一样。悲观机制需要锁住所有评估过的查询，以防止写入操作影响 这些结果。而乐观机制则跟踪所有评估过的查询，并在事务结束时使用一个后向验证或前向验证的模式来检查是否有并行写入操作影响了并行读入操作，如果有的 话，它会将冲突事务外的所有事务进行回滚。在这种隔离级别中，任何提交事务都不会改变系统的表征状态。最好在需要完整数据一致性的情况下使用这个级别的隔离。

例子：一个进行范围查询来计算新值的账目系统。

总结

下面是本文提到的隔离级别的汇总表，它可以帮助你找到最适合你应用程序的级别。

事务在不同隔离级别中可能的冲突类型：

	脏写	脏读	混合状态	不一致读	覆写	不可重复	幻读	不一致性
未提交读	不可以	可以	可以	可以	可以	可以	可以	可以
已提交读	不可以	不可以	可以	可以	可以	可以	可以	可以
单调视图	不可以	不可以	不可以	可以	可以	可以	可以	可以
快照读取	不可以	不可以	不可以	不可以	可以	可以	可以	可以
游标稳定性	不可以	不可以	可以	可以	不可以	可以	可以	可以
可重复读取	不可以	不可以	可以	可以	不可以	不可以	可以	可以
快照隔离	不可以	不可以	不可以	不可以	不可以	不可以	不可以	可以
可串行性	不可以	不可以	不可以	不可以	不可以	不可以	不可以	不可以

不同隔离级别的最佳前提：

	缓冲	数据同步	乐观冲突模式	建议操作	例子
未提交读	允许缓冲	间歇的	检测脏写	不能并发读写	档案
已提交读	允许缓冲	间歇的	没有冲突检测	单调的读/写	Web 论坛
单调视图	必须被验证	周期的	没有冲突检测	组合读入	用户偏好
快照读取	必须被验证	周期的	对比读入与修改内容	一致性读入	查询表
游标稳定性	允许缓冲	间歇的	对比修改的实体版本	CRUD 服务	目录
可重复读取	允许缓冲	间歇的	对比读入的实体版本	读/写实体	订单跟踪
快照隔离	必须被验证	周期的	对比读入的实体版本	同步实体	工作流
可串行性	必须被验证	完整同步	对比查询与修改内容	完善数据一致性	账目

数据一致性在数据库应用程序中至关重要-它允许开发者在分布式环境下使用数据。尽管强一致性级别如可串行性提供了一个简单的编程模型，但是它们会导 致开销 过大，操作阻塞或者事务回滚，这对于很多应用程序来说都是不必要的。如果有其他问题的话，可以使用更加适当的隔离级别来帮助开发人员和系统架构师，让他们 在保持性能和开销平衡的前提下更好的理解数据一致性的需求。

原文链接：http://www.infoq.com/cn/articles/eight-isolation-levels

相关内容：

- .NET 4 Beta 1 支持用软件实现的内存事务
- REST和分布式事务
- 软件性事务：一种编程语言的视角
- 开发者应该书写他们自己的事务协调逻辑吗？
- 文件系统事务——仍然是一个棘手的领域吗？

推荐文章 | Articles

RESTful HTTP的实践

作者 Gregor Roth 译者 马国耀

本文对 RESTful HTTP 的基础原理做了一个概览，探讨了开发者在设计 RESTful HTTP 应用时所面临的典型问题，展示了如何在实践中应用 REST 架构风格，描述了常用的 URI 命名方法，讨论了如何使用统一接口进行资源交互，何时使用 PUT 或 POST 以及如何支持非 CURD 操作等。

REST 是一种风格，而不是标准。因为既没有 REST RFC，也没有 REST 协议规范或者类似的规定。REST 架构是 Roy Fielding（他也是 HTTP 和 URI 规范的主要作者之一）在一篇论文中描述的。像 REST 这样的架构风格通常会定义一组高层决定让应用程序去实现。 所有实现了某种特定架构风格的应用程序，都使用相同的模式，也用相同的方式使用别的架构元素，如缓存，分布式策略等。Roy Fielding 把 REST 定义成一种架构风格，其目标是“使延迟和网络交互最小化，同时使组件实现的独立性和扩展性最大化”。

虽然 REST 受 Web 技术的影响很深，但是理论上 REST 架构风格并非绑定在 HTTP 上。然而，HTTP 是唯一与 REST 相关的实例。基于该原因，本文描述了通过 HTTP 实现的 REST，通常它也被称为 RESTful HTTP。

REST 并没有创造新的技术，组件或服务，隐藏在 RESTful HTTP 背后的理念是使用 Web 的现有特征和能力。RESTful HTTP 定义了如何更好地使用现有 Web 标准中的一些准则和约束。

资源

资源是 REST 中最关键的抽象概念，它们是能够被远程访问的应用程序对象。一个资源就是一个标识单位，任何可以被访问或被远程操纵的东西都可能是一 个资源。资源可以是静态的，也就是该资源的状态永远不会改变。相反，某些资源的状态可能随着时间推移呈现很大的可变性。这两种类型的资源都是有效的。

图 1 中所显示的这些类都能被很容易地映射成资源。面向对象设计者不容易理解把实体类

（如 Hotel 或者 Room）映射成资源，同样他们也不太理解从控制类（如 coordination，事务和某个类的控制类）到资源的映射。

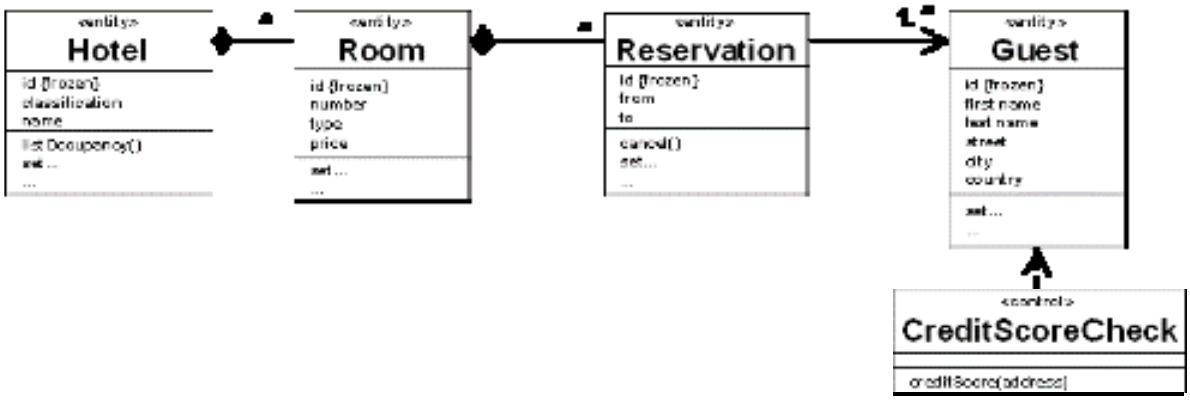


图 1：分析模型的例子

分析模型是识别资源的一个非常好的“切入点”。然而，并非只能进行 1 对 1 映射，比如，<Hotel>.listOccupancy()操作也可以被建模成资源。此外，也有可能某些资源只表示实体的某一部分。资源设计的主要驱动力是网络因素而不是对象模型。

任何重要的资源都应该能够通过一个唯一的标识被访问。RESTful HTTP 使用 URI 来识别资源。URI 提供了 Web 通用的识别机制，它包含了客户端直接与被引用的资源进行交互时需要的所有信息。

如何命名资源标识？

虽然 RESTful HTTP 并没有明确指出如何构造一个 URI 路径，但实际上经常被使用的是一些特定的命名模式。URI 命名模式有助于应用程序调试和跟踪，通常一个 URI 包 含资源类型及其后面用于定位特定资源的标识。这样的 URI 不包括指定业务操作的动词（verb），而只用于定位资源。图中 a1 给出了 Hotel 资源的一个示例，同一个 Hotel 资源也可以通过 URI（a2）访问。同一资源可以被多个 URI 引用。

```
(a1) http://localhost/hotel/656bcee2-28d2-404b-891b
(a2) http://127.0.0.1/hotel/656bcee2-28d2-404b-891b
(b) http://localhost/hotel/656bcee2-28d2-404b-891b/Room/4
(c) http://localhost/hotel/656bcee2-28d2-404b-891b/Reservation/15
(d)
http://localhost/hotel/656bcee2-28d2-404b-891b/Room/4/Reservation
/15
```

```
(e)
http://localhost/hotel/656bcee2-28d2-404b-891b/Room/4/Reservation
/15v7
(f) http://localhost/hotel/656bcee2-28d2-404b-891bv12
```

图 2：资源寻址的例子

URI 也可以被资源用来在资源表示（representation）之间建立关联。例如，Hotel 表示通过 URI 去引用已分配的 Room 资源，而 不是使用普通的 RoomID。使用普通的 ID 会强制调用者通过对资源的访问去构造 URI，而调用者如果没有主机名和基础 URI 路径等上下文信息是无法访问 到该资源的。

超链接常被客户端用于资源导航。RESTful API 是超文本驱动的，这表示客户端通过获得一个 Hotel 表示，就能够导航到已分配的 Room 表示和 Reservation 表示。

在实践中，图 1 所示的这些类经常被映射成某种业务对象，这意味着在业务对象的整个生命周期中 URI 将保持不变。如果要创建一个新资源，则要为之分配 一个新的 URI。而一旦这个新资源被删除，相应的 URI 则跟着失效。如图 2 中的（a），（b），（c）和（d）就是这种标识的例子。另一方面，URI 也可 以用来引用资源快照，比如（e）和（f）就是对这类快照的引用，其 URI 中包含了一个版本标识。

URI 还可以定位子资源，如示例中的（b），（c），（d）和（e）。通常，被聚集的对象会被映射成子资源，如 Room 是被 Hotel 聚集的。被聚集的对象通常没有自己的生命周期，如果它的父对象被删除，所有的被聚集对象也跟着被删除。

然而，如果一个子资源可以从一个父资源移动到另一个父对象， 那么在它的 URI 中就不应该包含其父资源的标识。比如图 1 中的 Reservation 资源，它就可以被分配给另一个 Room 资源。如果一个 Reservation 资源的 URI 包含了其父资源 Room 的标识，如（d）所示，则当 Room 实例标识改变时，如果该 Reservation 资源又被另 一个资源引用的话，这就会出问题。为了避免无效的 URI，Reservation 应该通过（c）这样的方式进行寻址。

通常，资源的 URI 是由服务器控制的。客户端访问资源时并不需要理解资源的 URI 命名空间结构。比如，使用（c）和（d）两个 URI 结构对客户端而言具有效果相同。

统一资源接口

为了简化整体系统架构，REST 架构风格包含了统一接口的概念。统一接口包含一组受限的良定义的操作，由它们进行资源的访问和操作。不论什么资源，都使用 相同的接口。客户端与 Hotel，Room 或 CreditScore 等资源交互时使用的接口是一样的。统一接口独立于资源

的 URI，并且也不需要类似 IDL 的文件去描述可用的操作。

RESTful HTTP 的接口非常流行且广为使用。它包含标准的 HTTP 方法如 GET，PUT 和 POST（浏览器使用它发出请求并提取页面）。不幸的是，很多开发者认为 实现 RESTful 应用就是用一种直接使用 HTTP 的方式，这种理解是错误的。举个例子，HTTP 方法的实现必须要遵循 HTTP 规范的，而通过 GET 方法 创建或修改对象是不遵守 HTTP 规范的。

应用统一接口

关于何时以及如何使用不同的 HTTP 动词（verb），在 Fielding 的论文中没有任何表格、列表或其他方式的描述。对于大部分方法，如 GET 或 DELETE，通过阅读 HTTP 规范就能清楚其含义，而对于 POST 和部分更新，就不那么容易了。在实践中，对资源进行部分更新有好几种方法，下文将有详 细介绍。

表 1 列出了大部分重要的方法 GET，DELETE，PUT 和 POST 的典型用法：

重要方法	典型用法	典型状态码	安全？	幂等？
GET	- 获取表示 - 变更时获取表示（缓存）	200（OK） - 表示已在响应中发出 204（无内容） - 资源有空表示 301（Moved Permanently） - 资源的 URI 已被更新 303（See Other） - 其他（如，负载均衡） 304（not modified）- 资源未更改（缓存） 400 （bad request）- 指代坏请求（如，参数错误） 404 （not found）- 资源不存在 406 （not acceptable）- 服务端不支持所需表示 500 （internal server error）- 通用错误响应 503 （Service Unavailable）- 服务端当前无	是	是

		法处理请求		
DELETE	- 删除资源	200 （OK）- 资源已被删除 301 （Moved Permanently）- 资源的 URI 已更改 303 （See Other）- 其他，如负载均衡 400 （bad request）- 指代坏请求 t 404 （not found）- 资源不存在 409 （conflict）- 通用冲突 500 （internal server error）- 通用错误响应 503 （Service Unavailable）- 服务端当前无法处理请求	否	是
PUT	- 用客户端管理的实例号创建一个资源 - 通过替换的方式更新资源 - 如果未被修改，则更新资源（乐观锁）	200 （OK）- 如果已存在资源被更改 201 （created）- 如果新资源被创建 301（Moved Permanently）- 资源的 URI 已更改 303 （See Other）- 其他（如，负载均衡） 400 （bad request）- 指代坏请求 404 （not found）- 资源不存在 406 （not acceptable）- 服务端不支持所需表示 409 （conflict）- 通用冲突 412 （Precondition Failed）- 前置条件失败（如执行条件更新时的冲突） 415 （unsupported media type）- 接受到的表示不受支持 500 （internal server error）- 通用错误响应 503 （Service Unavailable）- 服务当前无法	否	是

		处理请求		
POST	- 使用服务端管理的（自动产生）的实例号创建资源 - 创建子资源 - 部分更新资源 - 如果没有被修改，则不过更新资源（乐观锁）	200（OK）- 如果现有资源已被更改 201（created）- 如果新资源被创建 202（accepted）- 已接受处理请求但尚未完成（异步处理） 301（Moved Permanently）- 资源的 URI 被更新 303（See Other）- 其他（如，负载均衡） 400（bad request）- 指代坏请求 404 （not found）- 资源不存在 406 （not acceptable）- 服务端不支持所需表示 409 （conflict）- 通用冲突 412 （Precondition Failed）- 前置条件失败（如执行条件更新时的冲突） 415 （unsupported media type）- 接受到的表示不受支持 500 （internal server error）- 通用错误响应 503 （Service Unavailable）- 服务当前无法处理请求	否	否

表 1：统一接口示例

表示

对资源的操纵永远是通过其表示实现的。资源可能永远不会在网络中传输，相反，传输的是资源的表示。资源的表示包括数据和描述数据的元数据，例如，HTTP 头“Content-Type”就是这样一个元数据属性。

图 3 展示了如何使用 Java 获取表示。该例程使用了 Java HTTP 库 xLightweb 中的 HttpClient 类，这个库由作者本人维护。

```
HttpClient httpClient = new HttpClient();
```

```
IHttpRequest request = new GetRequest(centralHotelURI);
IHttpResponse response = httpClient.call(request);
```

图 3：获取表示的 Java 例程

通过调用 HTTP 客户端的 call 方法，一个访问 Hotel 资源表示的 HTTP 请求就被发送出去。返回的表示如图 4 所示，它也包含了用于指示实体主体的多媒体类型的 Content-Type 头。

```
REQUEST:

GET /hotel/656bcee2-28d2-404b-891b HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6

RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Content-Length: 277
Content-Type: application/x-www-form-urlencoded

classification=Comfort&name=Central&RoomURI=http%3A%2F%2Flocalhos
t%2Fhotel%2F
656bcee2-28d2-404b-891b%2FRoom%2F2&RoomURI=http%3A%2F%2Flocalhost
%2Fhotel%2F6
56bcee2-28d2-404b-891b%2FRoom%2F1
```

图 4：RESTful HTTP 交互

如何支持特定表示？

为了避免传输很大的数据集，有时应该接收表示属性一个子集。在实现时，用于指定部分属性的一种方式就是支持对指定属性的寻址，如图 5 所示。

```
REQUEST:

GET /hotel/656bcee2-28d2-404b-891b/classification HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Accept: application/x-www-form-urlencoded

RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
```

```
Content-Length: 26
Content-Type: application/x-www-form-urlencoded; charset=utf-8

classification=Comfort
```

图 5：属性过滤

图 5 中所示的 GET 调用只请求了一个属性（classification），如果要请求多个属性，所请求的属性要用逗号隔开，如图 6 所示。

```
REQUEST:

GET /hotel/656bcee2-28d2-404b-891b/classification,name HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6

Accept: application/x-www-form-urlencoded


RESPONSE:

HTTP/1.1 200 OK

Server: xLightweb/2.6

Content-Length: 43

Content-Type: application/x-www-form-urlencoded; charset=utf-8


classification=Comfort&name=Central
```

图 6：多属性过滤

确定所需属性的另一种方法是使用查询参数，通过它列出所请求的属性，如图 7 所示。查询参数将用于定义查询条件以及更复杂的过滤或查询准则。

```
REQUEST:

GET
/hotel/656bcee2-28d2-404b-891b?reqAttr=classification&reqAttr=nam
e HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6
```

```
Accept: application/x-www-form-urlencoded


RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Content-Length: 43
Content-Type: application/x-www-form-urlencoded; charset=utf-8


classification=Comfort&name=Central
```

图 7：查询字符串

在上述例子中，服务器总是返回以 application/x-www-form-urlencoded 编码的媒体类型的 表示。该媒体类型将实体编码成键值对列表。键值方法理解起来很容易，但缺点是，它不适用与更加复杂的数据结构。此外，这种媒体类型不支持标量数据类型的绑 定，如 Integer，Boolean，Date 等。基于这个原因，通常使用 XML，JSON 或 Atom 来表征资源（JSON 也没有定义 Data 类型的绑定）。

```
HttpClient httpClient = new HttpClient();


IHttpRequest request = new GetRequest(centralHotelURI);
request.setHeader("Accept", "application/json");


IHttpResponse response = httpClient.call(request);


String jsonString = response.getBlockingBody().readString();
JSONObject jsonObject = (JSONObject)
JSONSerializer.toJSON(jsonString);
HotelHotel= (Hotel) JSONObject.toBean(jsonObject, Hotel.class);
```

图 8：请求 JSON 表示

通过设置“Accept”请求头，客户端就可以请求指定的表示编码。图 8 展示了如何对

application/json 类型的表示的请求。JSONlib 将把图 9 中显示的返回响应消息映射成 Hotel bean。

```
REQUEST:

GET /hotel/656bcee2-28d2-404b-891b HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6

Accept: application/json


RESPONSE:

HTTP/1.1 200 OK

Server: xLightweb/2.6

Content-Length: 263

Content-Type: application/json; charset=utf-8


{"classification":"Comfort",

"name":"Central",

"RoomURI":["http://localhost/hotel/656bcee2-28d2-404b-891b/Room/1
",
      "http://localhost/hotel/656bcee2-28d2-404b-891b/Room/2"]}
```

图 9：JSON 表示

如何报告错误？

当服务器不支持所请求的表示时怎么办？图 10 展示了一个请求 XML 表示资源的 HTTP 交互，若服务器不支持这种表示，它将返回一个 HTTP 406 响应，表示拒绝处理该请求。

```
REQUEST:

GET /hotel/656bcee2-28d2-404b-891b HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6
```

```
Accept: text/xml


RESPONSE:
HTTP/1.1 406 No match for accept header
Server: xLightweb/2.6
Content-Length: 1468
Content-Type: text/html; charset=iso-8859-1


<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1"/>
      <title>Error 406 No match for accept header</title>
   </head>
   <body>
       <h2>HTTP ERROR: 406</h2><pre>No match for accept header</pre>
         ...
   </body>
</html>
```

图 10：不支持的表示

RESTful HTTP 服务端程序必须根据 HTTP 规范返回状态码。状态码的第一个数字标识返回类型，1xx 表示临时响应，2xx 表示成功响应 ，3xx 代表转发，4xx 表示客户端错误，5xx 代表服务端错误。使用错误的响应码，或者总返回 200 响应，并在消息主体中包含特定应用程序的响应，这两 种做法都是不好的实践。

客户代理和中介也要分析返回码。例如，xLightweb HttpClient 默认会把持久的 HTTP 连接保存在连接池中，当一个 HTTP 交互完成时，持久化 HTTP 连接就应返回到内部连接池已备重用。而只有完 好的连接才能被放回连接池，比如，若返回码是 5xx，那该连接就不会重回连接池了。

有时某些特定的客户端要求更简洁的返回码。一种方法是增加一个 HTTP 头“X-Header”，用它来详细描述 HTTP 状态码。

```
REQUEST:

POST /Guest/ HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6

Content-Length: 94

Content-Type: application/x-www-form-urlencoded


zip=30314&lastName=Gump&street=42+Plantation+Street&firstName=For
est&country=US&

city=Baytown&state=LA


RESPONSE:

HTTP/1.1 400 Bad Request

Server: xLightweb/2.6

Content-Length: 55

Content-Type: text/plain; charset=utf-8

X-Enhanced-Status: BAD_ADDR_ZIP


AddressException: bad zip code 99566
```

图 11：附加状态码

通常只有在进行编程问题诊断时才需要详细的错误码。尽管比起详细的错误码，HTTP 状态码的描述性总是要差很多，但是在大多数情况下，它们对于客户端正确处理问题已经足够了。另一种方法是在响应主体中包含详细的错误码。

PUT还是POST？

较之流行的 RPC 方式，HTTP 方法不仅仅在方法名上有所不同，而且 HTTP 方法中的某些属

性（如幂等性，安全性等）也扮演着重要的角色。不同的 HTTP 方法的幂等性和安全性属性也是不同的。

```
HttpClient httpClient = new HttpClient();

String[] params = new String[] { "firstName=Forest",
                        "lastName=Gump",
                        "street=42 Plantation Street",
                        "zip=30314",
                        "city=Baytown",
                        "state=LA",
                        "country=US"};
IHttpRequest request = new PutRequest(gumpURI, params);
IHttpResponse response = httpClient.call(request);
```

图 12：使用 PUT 方法

如图 12 和 13 所示，使用 PUT 操作来创建一个新的 Guest 资源。PUT 方法将封装好的资源存放在 Request-URI 之下。该 URI 是由客户 端决定的，当 Request-URI 指向某现存资源时，该资源将被新资源替换。基于该原因，PUT 方法一般用于创建新资源或更新现有资源。然而，通过使用 PUT，资源的整个状态都会被改变，若一个请求只需要修改 zip 域，它不得不包含该资源的其他域，如 firstName，city 等。

```
REQUEST:
PUT Hotel/guest/bc45-9aa3-3f22d HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Content-Length: 94
Content-Type: application/x-www-form-urlencoded

zip=30314&lastName=Gump&street=42+Plantation+Street&firstName=For
est&country=US&
city=Baytown&state=LA
```

```
RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Content-Length: 36
Content-Type: text/plain; charset=utf-8
Location: http://localhost/guest/bc45-9aa3-3f22d

The guest resource has been updated.
```

图 13：HTTP PUT 交互

PUT 方法是幂等的，幂等性意味着对于一个成功执行的请求，不管其执行多少次，其结果都是一致的。也就是说，只要你愿意，你可以用 PUT 方法对 Hotel 资源进行任意次更新，其结果都一样。如果两个 PUT 方法同时发生，那么只有其中之一会赢得最后的胜利并决定资源的最终状态。删除操作也是幂等 的，如果一个 PUT 方法和 DELETE 方法同时发生，那么资源或者被更新，或者被删除，而不可能停留在某个中间状态。

如果你不确定是 PUT 还是 DELETE 被成功执行，并且没有得到状态码 409 (Conflict)或者 417 (Expectation Failed)的话 ,那么就重新执行一遍。而不需要附加的可靠性协议来避免重复请求，因为通常重复的请求不会有任何影响。

上述描述对于 POST 方法就不适用了 ,因为 POST 方法不是幂等的 ,若要两次执行同一个 POST 请求那就要注意了。POST 方法所缺失的幂等性就解 释了为什么当你每次重新发送 POST 请求时浏览器总是弹出警告。POST 方法用于创建资源，而不需要由客户端指定实例 id，图 14 展示了通过 POST 方法 创建一个 Hotel 资源的 HTTP 交互过程。通常，客户端使用只包含基路径和资源类型名的 URI 来发送 POST 请求。

```
REQUEST:
POST /HotelHTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Content-Length: 35
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

```
Accept: text/plain

classification=Comfort&name=Central

RESPONSE:
HTTP/1.1 201 Created
Server: xLightweb/2.6
Content-Length: 40
Content-Type: text/plain; charset=utf-8
Location: http://localhost/hotel/656bcee2-28d2-404b-891b

the Hotelresource has been created
```

图 14：HTTP POST 交互（创建）

POST 方法也经常用于更新资源的部分内容，比如，如果我们要通过发送仅包含 classification 属性的 PUT 请求去更新 Hotel 资源的 话，这就是违反 HTTP 的，但是用 POST 方法则没有问题。POST 方法既不是幂等的，也不是安全的。图 15 展示了一个执行部分更新的 POST 方法。

```
REQUEST:
POST /hotel/0ae526f0-9c3d HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Content-Length: 19
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Accept: text/plain

classification=First+Class

RESPONSE:
HTTP/1.1 200 OK
```

```
Server: xLightweb/2.6
Content-Length: 52
Content-Type: text/plain; charset=utf-8

the Hotelresource has been updated (classification)
```

图 15： HTTP POST 交互 （更新）

还可以使用 PATCH 方法来进行部分更新，PATCH 方法是对资源进行部分更新的一个特殊方法。一个 PATCH 请求包含一个补丁文档，它将应用于由 Request-URI 所指定的资源。然而 PATCH 的 RFC 规范还在草稿中。

使用HTTP缓存

为提高扩展性并降低服务端负载， RESTful 的 HTTP 应用程序可以利用 WEB 基础设施的缓存机制。HTTP 已经意识到缓存是 WEB 基础设施必不可少的一部分，比如，HTTP 协议定义 了专门的消息头来支持缓存。如果服务端设置了这个头，客户端（如 HTTP 客户端或 Web 缓存代理）就能够有效地支持缓存策略。

```
HttpClient httpClient = new HttpClient();
httpClient.setCacheMaxSizeKB(500000);

IHttpRequest request = new GetRequest(centralHotelURI +
"/classification");
request.setHeader("Accept", "text/plain");

IHttpResponse response = httpClient.call(request);
String classification = response.getBlockingBody.readString();

// ... sometime later re-execute the request
response = httpClient.call(request);
classification = response.getBlockingBody.readString();
```

图 16：客户端缓存交互

图 16 显示了一个重复的 GET 调用。通过设置最大缓存大小的值>0 激活了 HttpClient 的缓存功能。如果响应消息中包含了刷新头，比如 Expires 或 Cache-Control: max-age，该响应就会被 HttpClient 缓存。这些头指明了关联的表示可以保鲜的时间为多久。如果在一段时间内发出了相同的请求，那么 HttpClient 就会使用缓存为这些请求提供服务，而不需要重复进行网络调用。在网络上总共只有一次 HTTP 交互，如图 17 所示。诸如 WEB 代理之类 的缓存中介也实现了相同的功能，而且该缓存还可以在不同客户端之间共享。

```
REQUEST:
GET /hotel/656bcee2-28d2-404b-891b/classification HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Accept: text/plain

RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Cache-Control: public, max-age=60
Content-Length: 26
Content-Type: text/plain; charset=utf-8

comfort
```

图 17：包含过期头的 HTTP 响应

过期模型在静态资源上很好用，可是，对于动态资源（资源状态经常改变且无法预测）则不尽相同。HTTP 通过验证头，如 Last-Modified 以 及 ETag 来支持动态资源的缓存。与过期模型相比，验证模型没有节省网络调用。但是，当执行带条件的 GET 方法时它会对昂贵的操作节约网络传输，图 18（2.request）显示了带条件的 GET 操作，它带有一个额外的 Last-Modified 头，这个头包含了缓存对象最后修改日期。如果该资源未 被更改，服务端将会返回一个 304 (Not Modified) 响应。

```
1. REQUEST:
```

```
GET /hotel/656bcee2-28d2-404b-891b/Reservation/1 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Accept: application/x-www-form-urlencoded

1. RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Content-Length: 252
Content-Type: application/x-www-form-urlencoded
Last-Modified: Mon, 01 Jun 2009 08:56:18 GMT

from=2009-06-01T09%3A49%3A09.718&to=2009-06-05T09%3A49%3A09.718&g
uestURI=
http%3A%2F%2Flocalhost%2Fguest%2Fbc45-9aa3-3f22d&RoomURI=http%3A%
2F%2F
localhost%2Fhotel%2F656bcee2-28d2-404b-891b%2FRoom%2F1

2. REQUEST:
GET /hotel/0ae526f0-9c3d/Reservation/1 HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Accept: application/x-www-form-urlencoded
If-Modified-Since: Mon, 01 Jun 2009 08:56:18 GMT

2. RESPONSE:
HTTP/1.1 304 Not Modified
Server: xLightweb/2.6
Last-Modified: Mon, 01 Jun 2009 08:56:18 GMT
```

图 18：基于验证的缓存

不要在服务端存储应用状态

RESTful HTTP 的交互必须是无状态的 这表明每一次请求要包含处理该请求所需的一切信息。客户端负责维护应用状态。RESTful 服务端不需要在请求间保留应用 状态，服务端负责维护资源状态而不是应用状态。服务端和中介能够理解独立的请求和响应。Web 缓存代理拥有一切正确处理请求所需的信息并管理它的缓存。

这种无状态的方法是实现高扩展和高可用应用的基本原则。通常无状态使得每一个客户请求可以由不同的服务器来响应，当流量增加时，新的服务器可以加进来，而如果某个服务器失败，它也可以从集群中移除。若要了解关于负载均衡以及故障恢复方面的更详细信息，请参考这篇文章服务器负载均衡架构 。

对non-CRED操作的支持

开发者经常想了解如何将 non-CRUD（Create-Read-Update-Delete）操作映射到资源。显然，Create、Read、 Update 和 Delete 等操作能够很容易地映射到资源的方法。然而，RESTful HTTP 还不仅限于面向 CRUD 的应用。

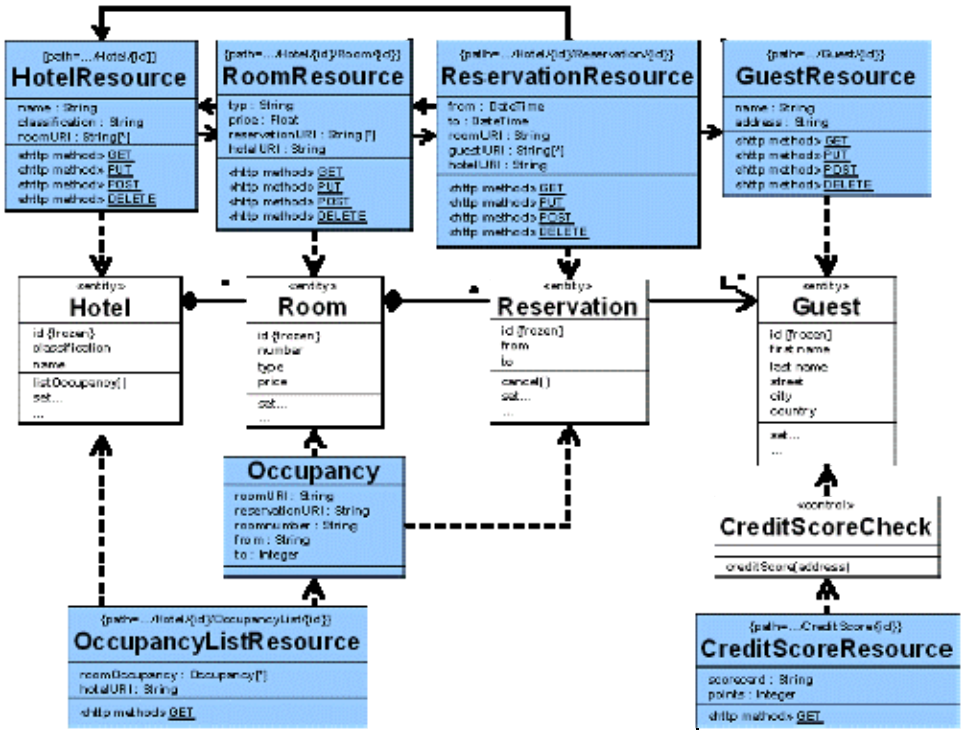


图 19：RESTful HTTP 资源

就如图 19 所示的 creditScoreCheck 而言，它提供了一个 non-CRUD 操作 creditScore(...)，该操作接受一个 address，计算出 score 并返回。这样的操作可以通过 CreditScoreResource 实现，该资源代表着计算的返回。图 20 展示了一个 GET 方法，它传入 address，然后提取 CreditScoreResource 表示，查询参数被用来指定 CreditScoreResource。 GET 方法是安全的，并且可缓存，所提它很适用于 CreditScore Check 的 creditScore(...)方法的非功能性行为。计算的结果可以缓存一段时间，如图 20 所示，响应包含了一个缓存头，它通知客户端和中 介执行响应缓存。

```
REQUEST:

GET
/CreditScore/?zip=30314&lastName=Gump&street=42+Plantation+Street
&
            firstName=Forest&country=US&city=Baytown&state=LA
HTTP/1.1

Host: localhost

User-Agent: xLightweb/2.6

Accept: application/x-www-form-urlencoded


RESPONSE:

HTTP/1.1 200 OK

Server: xLightweb/2.6

Content-Length: 31

Content-Type: application/x-www-form-urlencoded

Cache-Control: public, no-transform, max-age=300


scorecard=Excellent&points=92
```

图 20：Non-CRUD HTTP GET 交互

上述例子还显示了 GET 方法的局限性。尽管 HTTP 规范并没有指定 URL 的最大长度，但是实际上客户端，中介以及服务端对 URL 的长度都有限制。基于此，通过 GET 的查询参数发送

一个很大的实体可能会因为中介和服务器对 URL 长度的限制而失败。

另一解决方法是使用 POST 方法，如果作了设置，它也是可缓存的。如图 21 所示，第一个 POST 请求的结果是创建了一个虚拟资源 CreditScoreResource。输入的 address 数据用 text/card 这个 mime 类型进行编码，在服务端计算得到 score 之后，它发回一个 201（created）响应，该响应包含着所创建的 CreditScoreResource 资源的 URI。示例中还展示了如果进行了设定，POST 响应也可以被缓存。通过一个 GET 请求就能够取到计算结果。GET 响应也包含一个缓存控制头，如果客户端紧接着重新执行这两次请求，那么它们都可由缓存进行响应。

```
1. REQUEST:
POST /CreditScore/ HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6
Content-Length: 198
Content-Type: text/x-vcard
Accept: application/x-www-form-urlencoded

BEGIN:VCARD
VERSION:2.1
N:Gump;Forest;;;;
FN:Forest Gump
ADR;HOME:;;42 Plantation St.;Baytown;LA;30314;US
LABEL;HOME;ENCODING=QUOTED-PRINTABLE:42 Plantation St.=0D=0A30314
Baytown=0D=0ALA US
END:VCARD

1. RESPONSE:
HTTP/1.1 201 Created
Server: xLightweb/2.6
Cache-Control: public, no-transform, max-age=300
Content-Length: 40
```

```
Content-Type: text/plain; charset=utf-8
Location: http://localhost/CreditScore/l00000001-l0000005c

the credit score resource has been created

2. REQUEST:
GET /CreditScore/l00000001-l0000005c HTTP/1.1
Host: localhost
User-Agent: xLightweb/2.6

2. RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.6
Content-Length: 31
Content-Type: application/x-www-form-urlencoded
Cache-Control: public, no-transform, max-age=300

scorecard=Excellent&points=92
```

图 21： Non-CRUD HTTP POST 交互

还有其他不同的实现方式。比如不返回 201 响应，而返回 301(Moved Permanently)转发响应。该响应缺省是可缓存的。其他避免二次请求的方法是在 201 响应中增加一个新创建的 CreditScoreResource 资源的表示。

总结

大多数 SOA 架构（如 SOAP 或 CORBA）都试图映射如图 1 所示的类模型，或多或少是一对一的远程访问。通常，这些 SOA 架构的比较多地关注在编程语言对象的透明映射上，这种映射很容易理解，且易于跟踪。可是，它们把对分布性和扩展性等方面的关注排在第二位。相反，REST 架构风格的最主要驱动是分布性和扩展性。RESTful HTTP 接口的设计是由网络因素

而非编程语言的绑定驱动的。 RESTful HTTP 也没有试图去封装很那些难隐藏的因素，如网络延迟，网络健壮性以及网络带宽等。

RESTful HTTP 应用用一种直接的方式使用 HTTP 协议，而不需任何抽象层，也不存在 REST 指定的数据域，如错误域，安全令牌域等。RESTful HTTP 应用只使用 WEB 的固有能力。设计 RESTful HTTP 的接口意味着远程结构的设计者必须在 HTTP 协议上进行思考。这通常增加了开发周期中额外步骤。然而，RESTful HTTP 使得应用程序实现具有高扩展性，更健壮。特别是为很大用户群提供 Web 应用的公司，如 WebMailing 或 SocialNetworking 的应用就能从 REST 架构风格中获益。通常，这些应用要更快更高地扩展，而且，这些公司通常在一 些低预算的基础设施（基于广泛使用的标准组件和软件之上）上运行应用。

关于作者

Gregor Roth，xLightweb HTTP 库的作者。在 United Internet 组织担任软件架构师，该组织是最重要的欧洲因特网服务提供商，其产品有 GMX, 1&1, and Web.de 等。他感兴趣的领域包括软件和系统架构、企业架构管理、面向对象设计、分布式计算和开发方法论等。

原文链接： http://www.infoq.com/cn/articles/designing-restful-http-apps-roth

相关内容：

- 使用绑定实现灵活通信
- Dan Diephouse谈Atom、AtomPub、REST和Web服务
- 纯GET的REST集成模式——是同步，还是集成？
- 异步REST操作的处理
- REST的业务用例

推荐文章 | Articles

虚拟座谈：Bug追踪器的演变

作者 Dionysios G. Synodinos 译者 刘申

Bug追踪系统（Bug Tracking Systems）已经成为任何一个开发团队的必备工具之一，在过去几年中发生了很大的变化。InfoQ组织了一场虚拟座谈，邀请了几位来自JIRA、FogBugz、Basecamp以及MantisBT的嘉宾，共同探讨了Bug追踪器的演变及其未来的发展。

参加此次虚拟座谈的嘉宾有：

- Jon Silvers 和 Ken Olofsen（来自 Atlassian）
- Daniel Wilson（来自 Fog Creek Softwre）
- David Heinemeier Hansson（来自 37signals）
- Victor Boctor（来自 MantisBT）

InfoQ：能描述一下过去 3 年你们产品的演变过程吗？其中最重要的里程碑是什么？

Jon 和 Ken：过去几年中，Bug 追踪器已经从追踪简单的任务变成更加复杂的软件（比如它可以管理项目的生命周期、创建用户工作流以及为未来的改进工作提供报告）。随着项目管理功能的增强，Bug 追踪器已经不仅限于开发者所涉及的领域。特别是过去 3 年，JIRA 已经演变成高强度开发团队的核心。JIRA 最初只是一个简单却很实用的追踪软件缺陷的系统。如今，它已经能追踪管理各种各样的事 务——从初始开发任务的 bug 到团队的招聘活动。JIRA 支持所有 Scrum 功能——从计划板（Planning Boards）到燃尽图（Burn-down Charts）。

在 JIRA 演变中的几个关键点包括它的插件架构（它允许 JIRA 能够更方便的进行扩展和自定义）、与 IDE 的连通（它能够与团队的开发平台界面集成在一起）与 Atlassian 主要的协作开发工具整合（从持续集成服务器到结对代码复查，再到源代码的浏览，开发中需要的每一件事，都可以在这一整套工具中完成）。

而且从 JIRA 4.0 开始，能够看到一些重大的改变，包括一个全新动态的用户界面、信息面板、Gadget（它们都是基于 OpenSocial，JIRA 是一个 OpenSocial 的容器和发布商）以及社交功能，比如 ActivityStreams（它把用户活动聚合成非常容易订阅的 feed，就像 FaceBook 一样）。总而言之，JIRA 以及其他的 bug 追踪器，已经演变成拥有复杂功能的多面手。

Daniel：wow，互联网时间中的三年！回头看 5.0 版的 FogBugz，大概是 06 年 8 月份推出的，就好像在审视遥远地质时代的化石。对于菜鸟来说，显然 FogBugz 7.0 看起来要更加友好。不过，事实的确是 FogBugz 正演变成一个完整的项目管理系统，缺陷追踪是它的核心组件。如果你仔细想一下，如果使用一个工具管理你的项目计划，再用另外一个工具去追踪开发过程中的 bug，的确是件非常麻烦的事。其实，在这两个过程中，你都做了同样的事情：描述任务、评估工作量、在时间表上跟踪项目进度。

所以，我们把 FogBugz 7.0 设计成可跟踪整个项目，从计划到最后的发布。我们的 Subcases 功能允许你把任务细分，并且 Agile/Scrum 团队可以从任务列表中管理产品 backlog。你会按时发布，因为有我们的基于证据调度（Evidence-Based Scheduling）算法的帮忙，它会借鉴开发者过往的估算记录。不用再发邮件通知文档规范不同版本之间的变化了，因为 FogBugz 还包括有一个全功能的 wiki，可帮助你撰写、查看项目文档。

我们是在自己核心的 bug 追踪功能上构建了以上这些功能，使其更加如鱼得水。我们用 AJAX 强化的任务列表功能，使输入 bug 就像在文本编辑器上做记录一样简单。在客户支持方面，贝叶斯过滤（Bayesian Filtering）算法会阻击所有 spam，让 email 能够准确投递到相应的部门。最后，我们还引入了插件架构，它能够使你自定义以及扩展几乎 FogBugz 的每一个功能。第一批的 Fog Creek 插件让你可以修改项目的工作流，使其与团队的工作方式相匹配，以及增加自定义项目域等等。这个平台对全部有志于开发出强大插件并使其整合到 FogBugz 的开发者都是开放的。

David：过去 3 年我们做的最重要的一件事是不做太多的改变。很容易对本来受欢迎的产品不断的添加功能，最终导致过度演变，使其变得越来越复杂，很难使用。

这并不是说我们不对产品进行改进了，恰恰相反，比如说，最近我们就通过 Flash Widget 改进了上传进度提示以及多文件的选取功能。我们还添加了导出成 HTML 的功能，所以你可以把整个项目变成一个独立的本地站点。

但可能是最重要的，我们一直在不断的精炼人们每天要面对的那些 UI。像增加了在信息面板上快速浏览里程碑的功能，这样你就无需点击并放大附加的图片了。

Victor：过去 3 年发生了很多变化，下面这些是比较重要的：插件的支持、针对 CVS、SVN 以及 GIT 的更简单/多功能的源代码控制集成、 Twitter 的整合、标签、改进的 LDAP/AD 支持、完全可自定义的缺陷域、XML 导入/导出、便捷的 SOAP 的 API、更新的 Docbook 手册以 及 46 种语言的本地化，等等。

在我们开发上的变化是，我们转移到了 Git 上，把它作为我们的源代码控制工具。这大大提高了开发者的工作效率，为社区提供了一个更好的框架，可使大家自定义分支，并把新的改进提交给我们，并最终集成到核心的产品上。

InfoQ：对于 bug 追踪器与主流 IDE 的集成，你怎么看？你认为最终会发展成什么样子？

Jon 和 Ken：IDE 在过去几年中也发生了很大的改变，开发者也越来越希望在 IDE 中就能完成所需要做的事情。我们 现在通过免费的连接器对 JetBrains IntelliJIDEA 和 Eclipse 都支持对缺陷的跟踪、创建代码复查、查看编译错误以及远端代码库中浏览。就好比你可以在 iPhone 上使用 Goolge Reader 或者 Gmail 一样，不用改变任何设置，我们相信开发者每天都会越来越依赖 IDE，回到 Web 界面上进行配置并实现更复杂的功能。

Daniel：大多数用户习惯在工作的时候在浏览器中打开任务列表，但是一些开发者或许想要在 IDE 中管理这些任务。所以，我们为 Visual Studio(2005 和 2008)以及 Eclipse 都提供了相应的插件，可以在相应的面板中管理任务。至于 IDE 插件是否会在将来像我们的 Web 界面 那样受到大家的欢迎，我不感确定，但是我相信它们会作为缺陷追踪工具的分支，一直存在的。

David：Basecamp 没有提供任何与 IDE 集成的组件，但是我们提供了一个功能强大的 API，大家可以引用它。我并不了解这点对广大开发者来说到底有多重要，从我们来讲，并没有感觉有什么不便，我们团队内部一直在使用 Basecamp 进行 bug 的追踪。

Victor：MantisBT 提供了 MantisConnect 与 Eclipse IDE 进行整合。在我们看来，bug 追踪功能应该集成到每个人所使用的工具当中。项目经历可能需要 Excel 的集成，开发者则使用 IDE 的集成，而客户则倾向使用 Web 界面，等等。提供 API 是集成战略中的一部分，它能够让社区创建自己所需要的连接器。

InfoQ：你认为什么时候适合把 bug 追踪器当作一种服务以及一个组织什么时候需要在自己的架构上安装这种软件？你认为云计算会对上面两种方式有什么帮助吗？

Jon 和 Ken：由于我们大部分的客户都位于防火墙后，所以可以看到我们的在线产品（JIRA Hosted 和 JIRA Studio）越来越受到欢迎——特别是对小团队而言，他们并不需要特定的设施。对于那些想在自己架构上搭建的客户来说，主要有两个原因：a）他们有一个本地的企业级软件需要集成；b）他们对 IP 的安全性非常敏感，并且不想承担任何风险。

我认为未来所有的软件开发以及部署都会在云端。而且，应用程序运行在网络上需要使用很多外部服务（比如通过 Goolge Visualizations 提供复杂的图表），但同时也降低了总体成本。

Daniel：如果你的组织并没有控制整个系统的需求，那么，bug 追踪提供商会非常了解如何更安全的托管你的数据，所以，把 bug 追踪当作服务来使用是一种非常不错的选择。FogBugz on Demand 越来越受到大家的欢迎，因为它非常安全（128 位加密，世界级的服务器设施）以及为客户免去了很多管理上的负担。对于那些想自己掌管一切的公司，他们还是会选择 FogBugz 的安装版。

至于云计算会对哪种方式有所帮助，这要看客户的具体需求。我的意思是，FogBugz On Demand 从定义上是一种云计算服务。但是我们都非常担心安全和带宽的问题，所以我们通常不允许在线插件（它会为云端的外部服务请求提供响应）。但是如果你是自己维护 FogBugz 实例的话，你可以安装与云端（或者自己的防火墙内）的任何服务进行交互的插件。

David：把软件作为服务后，一切就会简单的多。在我印象中，现在已经很少有关于是否应当自己架设、维护系统的争论了。但是，除非你要做的是最机密的 CIA 项目，那就会是一个例外。

Victor：没有通用的解决方案。有些组织倾向于自己架设 bug 追踪器、实施系统集成以及自定义。但是其他一些组织，就会倾向于这种拿来直接用的服务。这要取决于公司的策略和规模。关键的一点是，无论哪种选项都要提供给客户，MantisBT 两种方式都会提供给大家。

InfoQ：你认为 Bug 追踪器如何更好的适应组织的内部流程？如何适应那些软件开发流程框架，比如 Scrum 和 Lean？

Jon 和 Ken：通常情况下，bug 追踪器现在都已经可以很容易的为每个团队自定义工作流。

因为每个公司都有各自的开发模式（瀑布式或者敏捷），应用程序必须拥有足够的灵活性以满足不同的团队。

以敏捷为例，我们提供了对 JIRA 的扩展——GreenHopper，它可直接的支持敏捷实践，比如通过把问题描述成一个虚拟的索引卡片，使得迭代计划和任务的执行更加方便。但是我们同样也允许客户把 JIRA 连接到其他的敏捷项目管理工具上。如果你运行在一个封闭的环境内，是不可能称之为一个成功的 bug 追踪器的，所以，我们尽可能的把 JIRA 开放出去。

Daniel：我们最初创建 FogBugz 是为了满足我们在 FogCreek 自己的工作方式。我们喜欢简单的任务工作流，它可以自动把完成的任务赋给创建者使其审核，然后让任务域的数量最小化。插件最令人称道的是他们能够使 FogBugz 适用于任何组织。我们的自定义工作流插件（Custom Workflow Plugin）让客户创建自定义的任务目录、状态以及任务赋予规则。自定义域的插件让客户能够很轻松的创建一个或者两个他们所必须的额外域。并且，因为 FogBugz 是一个完整的项目管理系统，它提供了对各种软件流程框架的良好支持。比如，我们在 FogBugz 7.0 支持 Agile/Scrum 方法，它引入了燃尽图、backlog 管理插件以及更简单的管理短周期项目里程碑的方式。

David：在 Basecamp 里，我们试图保持所有事情尽可能的简单，所以我们没有强迫使用任何特有的方法。其中我们主要追踪 bug 的方式就是把他们发布到优先级 todo 列表中，把这些 todo 项链到相应的说明文本上。

我认为很容易使你的软件过度专业化，变成某一种特定的风格，然后便会很快地发现他们有多么的麻烦。

Victor：MantisBT 允许用户自定义项目状态、工作流、默认问题域、自定义域、报告等等。实际上，社区已经为 MantisBT 开发出了一个 Scrum 插件。我们还了解到，有人也在使用 MantisBT 管理非软件领域的缺陷问题。

InfoQ：bug 追踪软件的可用性在过去几年中呈上升趋势。你认为这其中还有可改进的地方吗？你认为某些技术（比如 HTML 5）会对其产生多大的影响？

Jon 和 Ken：Atlassian 已经把 bug 追踪软件引入到主流当中。7 年前，我们开创了全新的、多功能基于 Web 的界面，那时候，大多数商业软件只提供 windows 单机版。在过去的 3 年中，我们看到了很多关于富互联网应用的创新，未来，我们会看到更多。我们开始看到基于 Web 的应用程序（Gmail/Google Reader）已经好

于同类的桌面应用。我们也将会不断的提高产品中的可用性与生产率。至于HTML 5 会带来巨大的改变吗？很显然，对离线以及更多控制组件的支持，将会让它更容易开发出相应的功能，最终会带来生产率的提升。不过，规范中的大部分已经通过 浏览器插件以别的方式实现了。HTML 5 会让它变得更标准。

Daniel：当然，我们仍能不断改进它的可用性。我们已经引领了这种风潮，把bug追踪器从“Web 1.0 基于数据库的前端界面”转变成“基于Ajax的富Web应用”。我们会聆听客户的声音，不断改进，把最顶尖的开发精英投入到最困难的可用性问题上 （顺带提一下，如果任何人有对FogBugz改进的建议，请发邮件到 support@fogcreek.com ）。我们都很关注HTML 5，看它的哪些功能能够改善用户体验。举两个例子，通过在后台线程中运行“Web Worker”（译者注：这是HTML 5 引入的一个新概念，详情请移步这里）能够在某些情况下提高性能；“Canvas”元素对于图像显示来说，或许是Flash一个很好的替代品。

David：好的可用性不会依赖于新奇的技术。一件非常重要的事情，大多数开发者都应当把它做好，而且应当投入更多的时间在那上面——应用程序中的文案（copywriting）。文案即是设计，大多数应用程序都有糟糕的文案。HTML5 拥有一些很令人振奋的特性，比如从一个窗口拖拽到另外一个。我确定马上就会看到相应的产品。

Victor：bug 追踪器很重要的一方面是减少汇报 bug 中的阻力。所以，新技术的应用都是为了实现这一点。比如，提供离线支持、富用户体验等等。但是，对于那些使用过时浏览器的用户，我们仍有为其提供支持，并且持续为他们提供多种连接与工作方式。

InfoQ：你认为 bug 追踪器在不远的将来会发展成什么样子？你希望看到哪些新功能？

Jon 和 Ken：Atlassian 会在以下两个领域持续创新：

- IDE 和界面的整合：前面我也提到了，把丰富的用户体验混入到不同类型的客户端界面当中去，使其成为一个 IDE、好用的 Web 应用或者可添加到其他 Web 应用程序（例如 Gmail）的组件（比如 Gadget）——为用户提供更大的自由度、更顺畅的体验以及更少的场景转换。OpenSocial 是这其中的关 键一环。

- 改进社交功能以及用户关联度：我们相信我们是这条路上的领航者，但是社交功能应当是任何协作工具都应当与生具来的功能。扩展现有的社交功能，比

如 ActivityStreams、人际网络、高级用户档案等等是这部分的关键之处。

Daniel：作为一名软件开发者，我们最喜欢的新功能就是那些使 bug 追踪的过程更简单、更愉快，这样我们才能专注做出好的软件。FogBugz 的未来版本会专注于使终端用户更加愉快，因为缺陷跟踪只有在人们愿意参与进来的时候，才会真正起作用。

David：我们不断的令它保持简洁。这 5 年多来，我们一直都是在用我们的产品改进自己的产品。我不认为 bug 跟踪与其他的任务跟踪有什么特别的不同。就像我所说的，bug 并没有什么特别的。他们只是待做事情的一部分，与功能以及其他方面的改进列在一起，只是排了一下优先级而已。

Victor：下面是我对 bug 追踪器在不久将来发展的一些看法：

应用于小团队/甚至是单人团队 - 现在已经比你当初自己购买、安装、托管 bug 追踪器的成本减少太多了。现在，你可以免费在 SourceForege 上使用，在托管商那边也只需要 1 键安装。

移动接入 - 随着越来越多的人使用移动电话来完成工作并接入互联网，提供一个富移动体验将是非常关键的。

离线接入 - 随着 HTML 5 兼容的浏览器逐渐获取市场份额，提供离线接入会大大提高 Web 应用程序（包括 bug 追踪器）的生产率。

互用性 - 越来越多的工作需要在 bug 追踪器之间协同完成，从而管理相关的上游和下游 bug。理想的目标是构建一种标准的方式用于不同 bug 追踪起之间协同工作。

原文链接：http://www.infoq.com/cn/articles/bug-trackers

相关内容：

- 设计者-开发者工作流中的迭代模式
- 作为思考工具的敏捷
- .NET和Mono项目版Git——Git#
- GitHub发布问题跟踪系统和新API
- 借助信息化工作空间实现高效的团队自我管理

推荐文章 | Articles

模型驱动开发的误解和挑战

作者 Bertrand Portier, Lee Ackerman 译者 张兵

多年以来，采用模型驱动开发（MDD）的水平似乎仍没预期的那么好。阻碍、限制 MDD 使用的因素有很多，例如对实际的 MDD 成功案例缺乏认知、不确定如何在平常使用 MDD、缺少预先投资的拨款模式、或是没有战略举措的重点。

如果你过去尝试过 MDD，那你很可能遇到了一些挫折，导致你现在不再用它。也或许你正在尝试采用 MDD，而又面临着一些挑战和阻碍。无论你遇到上述哪种情况，本文都对你有所帮助。我们会在本文中看一看与采用 MDD 相关的挑战和误解。

建模早已证明了它在改善沟通、促进业务编排、提升质量、提高生产率上的价值。它的使用范围很广，分析、设计和开发都会有所涉及。考虑到这一点，我们就来看看有关 MDD 的诸多误解和挑战，我们又该怎样利用现代方法和相关工具集解决这些问题。

挑战：方法不当且不可用

过去 ,MDD 的一个关键抑制因素是人们实施活动的时候没有现成的 MDD 最佳实践。比如说，人们在阅读有关如何执行特定任务（诸如设计高可用的解决方 案）的过程文档时，文档里并没有任何 MDD 的内容。为了得到 MDD 实践，人们不得不到论文或书本里去找，然后再应用到现有的非 MDD 文档上。

如今，MDD 从业者在进行日常工作时，可用的 MDD 指南已经越来越多，而且那些信息嵌在他们每天使用的工具中。我们先看看开发过程，它包括利用 MDD 原则的“工具向导”最佳实践，这些“工具向导”隶属于整个方法和过程。

特定任务的指南（例如需求评审、设计用户接口或设计高可用的解决方案）现在都包含指向 MDD 内容的链接。比如推荐设计模式、提供设计中应用模式的指南、利用现成工具中的模式实现。

以前还有另一个阻碍因素，就是 MDD 与特定开发方法过度掺杂，人们无法提取 MDD 最佳实践，并将其应用到不同的场景中。一个典型的例子就是面向对象 分析和设计（OOAD）中

存在大量工具，你要么采用全部的 OOAD 内容，将其作为从 MDD 受益的一部分内容，要么就完全抛弃 OOAD。MDD 的最佳实践曾 是 OOAD 框架的一部分，但人们并不知道如何在框架之外利用这些最佳实践。抽取出 MDD 的内容并将其应用到不同的场景中是不可能的。

这些因素再加上其他一些原因导致企业很难在它们的环境里采用最佳实践（包括 MDD 最佳实践）。公司已经有了合适的过程和方法，而给这些方法添加 MDD 方面的内容却很困难。

为了在组织和特定类型的项目中采用 MDD，业界在裁剪特定开发过程方面已经做得越来越好。比如有的研讨会旨在指导团队完成定制的开发过程，这通常被 称作“方法采用研讨会”。研讨会的目的是针对特定项目裁剪现有的方法内容，它通常会涉及以下人员 过程工程师（管理组织开发过程的人）、首席架构师、开发 人员组长和项目经理。

支持定制后，方法工具浮出水面，比如 Rational Method Composer 和 Eclipse Process Framework Composer，它们包含定制的最佳实践库。这些工具的思想是整理、打包最佳实践，用工具为组织裁剪并采用这些最佳实践。在工具中，你选择想要采用的某 些方法元素，对它们进行修改、编辑，并将其组织成希望关注的过程。然后将该过程以可读格式（例如 HTML）在组织内发布，供从业者在日常工作中遵循。

尽管使用上述工具和方法的可用指南有很多，但仍然要求用户找到、理解并遵照指南。跨越这一障碍的措施是，除了在工具里提供指南之外，还要将方案的全 面自动化。举例来说，你能在使用基于 Eclipse 的产品时利用备忘单（Cheat Sheets）。备忘单提供完成任务的步骤指南，并能自动化工作流里的步骤。

下一节我们会讨论关于模式实现的机制。不管选用什么方法，要点都是获取越来越多的指南，并将其落实到工具中，从而更好地指导用户充分利用模型。

正如软件解决方案可能会过度工程化一样，创建指南也会发生同样的问题。克服这个挑战的最后一点是要务实、主动。计算出构建方案所需步骤的全部细节， 无论从价值来说还是从时间来说都没有什么意义。那关键的步骤是什么呢？他们如何与团队的技术相结合？在文档化步骤上投入时间的意义何在？相对于自动化，在 创建静态文档上又该投入多少呢？

过程，尤其是 MDD，都不是放之四海而皆准的，这将在“4-误解”中讨论。

挑战：基础设施和工具不能从MDD获益

近几年，我们看到建模工具已不局限于对特定图形符号（比如 UML）的支持了，经过发展，它已然能帮助从业者完成工作。这些工具不仅支持图形建模符号，也内置了 MDD 特性，这些特性有利于：

- 业务编排：业务编排是 SOA 等成功方法的关键方面。通过使用 MDD 模型、自动化、以及与之关联的“追踪”，你能记录决策的原 因，跟踪满足业务需求的所有方式。另外，我们可以研究利用 MDD 的特定版本，比如业务驱动开发（ BDD ）。顾名思义，BDD 关注的是业务建模。你可以在这 种情况下进行建模，也可以在某些情况下模拟组成业务的流程。
- 高质量：由于实践内建在工具中，并进行了自动化处理，因此出错的几率非常小，甚至不会出错。
- 增强的一致性和治理：由于工具支持指南和最佳实践的自动化，所以提高了解决方案中元素的一致性。另外，工具也能确保构建的元素是固定的，并与需求和最佳实践保持一致。
- 提高的生产率：重复、耗时的工作现在都自动化了。从业者可以“复用”，把时间花在最紧要的事情上（比如业务逻辑 ）。依赖自动构建，用户群的复杂度和自动化要么可以在当前项目中实现，要么可以分散在多个项目中实现。
- 改善的沟通：使用模型、工具和自动化，从业者（ 例如架构师 ）能针对不同的受众创建不同的视图。
- 影响分析：MDD 的可追踪性能让你分析出需求变更对解决方案造成的影响，反之亦然。

让我们以设计模式为例，来说明工具如何给 MDD 带来了活力。假设某本书中描述了设计模式，我们将其称为模式规范。该规范非常有用。它描述了模式的使 用时机、模式的特征，以及使用模式的好处和意义。模式规范能帮助人们理解模式并做出恰当的选择。但模式规范并不能确保设计的高质量和生产率的提高。为了从 中受益，你必须将这些模式“自动化”。我们将其称为模式实现，也就是模式规范在工具中的可复用编辑。使用模式实现，设计者可以将模式快速应用到他们的设计 中，也能确保这些应用准确无误。

领域独立的工具不太可能内置领域所需的所有 MDD 工件。工具除了提供开箱即用的 MDD 工件外（比如一组设计模式实现），也允许你扩展现有的工件、创 建自己的工件。现在的工具包含“扩展框架”，以及最佳实践、模板和 API。Rational Software Architect 之类的工具还允许你构建适用于领域的 MDD 工件（ 例如模式实现、规则、约束等 ）。

既然你能构建这些 MDD 工件，那么基于资产的开发（ ABD ）就能让你与他人共享工件、提升复用的实践和基础设施。换句话说，ABD 最佳实践和基础设 施的改进支撑了 MDD 的采用进程。诸如 Rational Asset Manager 的可复用资产库能让你管理可复用的软件工件，让开发社区共享和复用工件。试想一个为领域创建的模式实现 ,你现在可以把它提交到资产库中，

该 模式经过评审和认可，社区中的其他从业者就可以复用它了。作为这个生态系统的一部分，你可以监控资产被复用的时机和方式，收集反馈信息并确保整个团队在使 用合适资产的正确版本。

我们重新回到务实和实用上来。在对用户和用户所在组织有意义的情况下，ABD 及复用倡议才需要被采用。你需要识别出你的成熟度级别，并采用支持该级别的工具和过程。通过事先思考和计划，你可以随需确定、推广 ABD 计划，避免不必要的开销和成本。

误解：MDD==UML？

有一个误解是 MDD 意味着你必须使用统一建模语言（UML）——现状如此，完全是因为来自对象管理组织（OMG）的 UML 规范进行了这样的描述。消除这一误解的方法有很多。

打消这种念头的第一种方法是用 MDD 的方法工作，这只需要你在执行任务时把模型作为关键的工件使用，并使用利用这些模型的自动化机制。在这种情况 下，模型是用语言简化了的现实，而这些语言具有定义良好的语法和语义。因此，可以在 MDD 中使用的语言有很多，而不仅仅是 UML。

在大多数情况下，我们确实要为手头上的任务选择合适的工具。如果我们的 MDD 需要标准化、为人熟知、被广泛支持的语言，那 UML 就是一个不错的选 择。UML 也是可扩展的。严格来说，它能通过配置（提供定制的元素、属性和约束）进行定制。这能让 UML 对所作的工作来说变得具体，也能让语言更加易学易 用。增强建模语言可用性或针对性的另一种方式是创建你自己的领域特定语言（DSL）。

要记住的是，我们受益于使用的语言和创建的模型。为了指导投资，我们要权衡以下问题：

- 是否能有效地设计和理解解空间？
- 能否轻松地和其他人沟通？
- 是否能基于已经创建好的模型生成方案的其他部分？
- 能否有效利用开发生命期之外的结果？
- 是否能从实现追溯到设计？甚至需求？

误解：MDD放之四海而皆准

根据前面的误解可以看出，MDD 显然不是放之四海而皆准的解决方法，任何非预设的生产

线工具集都可以用来构建产品。MDD 就是用模型为特定情况增加价值，它适用于特定领域，跟你所开发的软件类型也是配套的。因此，我们能在自己的场景中看到很多使用 MDD、有意义的方式。

其中一个例子是，我们可以和传统的面向对象分析和设计（OOAD）一起使用 MDD。在审视 OOAD 和 MDD 的时候，往往会发现使用了很多模型，比如 用例、分析、设计和实现。有很多现成的例子和文档演示了如何使用这些模型来完成方案。但这并不意味着你必须用上所有的模型。关键是我们要有效地利用抽象。 抽象的层次取决于所处的场景、使用的语言、相关的约束、规则和假设，以及可能实施的自动化。

除了在模型数量（和相关的抽象层次）上务实之外，也要切合实际地选择模型中用于交流的图表。比如说，如果使用 UML 作为建模语言，就没必要使用所有 可用的图表类型（类图、交互图……）。语言中有一系列图表可供选择，一般用途的建模语言能为各种需求提供服务就可以了。在特定情形下，对于你试图完成的工 作来说，只选择那些能为其增加价值、有助于沟通的图表才是有意义的。至于完整性，我们可以进行进一步的讨论，要注意的是，即使在一个图表中，你也不必使用所有可用的模型元素。

误解：图表就是模型

MDD 中关键的一点是要认识到我们在创建模型——正如前面所讨论的，模型是用语言简化现实，该语言要具有良好定义的语法和语义。我们在模型中可以发现大量 模型元素和一组图表。每种图表都提供了模型元素之上的一个视图。每个模型元素属于零或多个图表。我们要关注模型元素——它们是什么？有哪些关系？有什么属 性？我们通常使用图表来帮助我们理清这些问题。此外，我们还将图表作为和其他人沟通的方式。但模型的关键信息存在于模型元素中——因为这能让我们生成所需 的视图、创建所需的图表，从模型生成其他元素。如果 MDD 只是图表，那工具能画出漂亮的图片就能满足我们的需求了。这并不是说图表（和支持图表的工具）不 重要。创建模型和图表的工具需要进行调整，以适合目标受众。

结合有选择性地使用图表，我们还能利用视角让模型更加可用、更加利于沟通。视角是组织模型的一种方式，以便模型的某些方面可以提供额外的图表，使模型面向 各种各样的受众。透视图通常只包含图表，而没有额外的语义元素。严格来说，在你更新语义元素时，透视图会自动保持同步。使用透视图可以让你与其他角色和小 组有效沟通，从而为 MDD 增加价值。每个小组都想理解方案中的一部分内容，也就是与他们的需求相关的那部分。在不打乱模型、不构建独立模型、或是不在维护 同一元素的多个版本上浪费时间的情况下，透视图可以支持这些需求。请记住，我的意思并不是让你支持所有不同的小组，并创建庞大的一组

透视图。再次强调一 下，关键是要务实，要创建有意义、能带来价值的图表与视角。

误解：代码就是模型，模型就是代码

以前对MDD的误解之一就是它只能应用于代码。MDD基本上被局限在一个较低的抽象层次，因此它的影响也很有限。很多人只用MDD工具“可视化”代码（也 就是将代码图形化的逆向转换）。这样是有好处的，比如说，更好地理解大段代码，以及组件或类之间的关系。但撇开这些来说，代码可视化并不能获得先前讨论的 那些MDD优势（比如业务编排、改善质量、提高生产率或影响分析），因为它所作的一切也就是让你以图形化的方式查看代码而已。这是基本、初级的图形使用方 法，和预期的一样，它的投资低，收益也低。

再复杂一点儿，在代码可视化之后，让可视化结果和预期的设计保持一致。例如设计师或架构师想评审开发团队开发的代码，代码的可视化视图就能让他们对代码和 设计进行比较，因为可视化结果和设计使用了相同的可视化技术（比如UML类）。不过，尽管可视化结果和设计用相同的语言表示，但两者之间仍然有很大差距， 因为它们所处的抽象层次不同。MDD工具凭借可视化、可追踪性、分析和发现功能、重构支持能帮助设计师完成工作。一旦标注出设计和代码之间有分歧的地方， 人工干预就必不可少了，设计师就要和开发团队进行沟通。这能提升价值，但仍然无法完全拥有MDD的优势。为了支持分析和沟通，需要增加时间和精力，而且每 个项目都需要投入多次。

MDD应该适用于任何层次的抽象，并有助于不同层次之间的连通。你应该在较高层次的抽象上进行建模。以分析模型为例（像系统的用例模型），它是设计模型的 输入。分析模型中的有些元素可以在设计中予以利用。比如说，功能域信息分类（包）和用例可以用来创建设计模型中交互图的基础模板，用例会在设计模型中实 现。利用工具及其扩展性，你可以修改“分析到设计”的转换过程，接着让组织内的成员在质量和生产率上获益。

MDD适用于所有层次的抽象，而抽象的层次是无穷尽的。要为领域和组织选择有意义的抽象层次。例如，在SOA中，可以在开发方案时采用以下的抽象层次：

- 业务：该层次对业务策划师、业务分析师或产品所有者来说是有意义的。在这个层次上，模型元素是业务目标、关键性能指标、业务方针和功能域之类的内容。
- 分析：分析和设计通常要一起看，分析模型的元素有时会演进为设计元素。在SOA里，考虑分析是很重要的，因为分析是支持业务元素的技术模型元素的起点。
- 设计：SOA方案中，大多数在架构上重要的元素都是在这个层次建模的。设计时要用文档记录架构的关键元素，以及它们的实现方法。

- 实现：实现是“代码”层次的抽象。在该层中，你可以用 MDD 基于设计生成代码存根，并在需要的时候让代码和设计保持一致。

另一方面会出现这样的情况：人们热衷于模型和 MDD，甚至仅仅为了建模而建模，却忘了把模型转换成可操作或可执行的内容。架构师可以和利益相关者、设计者 和开发人员沟通，但你仍然不能完全受益于 MDD。在你策划 MDD 的策略和方法时 ,要思考一下如何利用模型。譬如，部署方案最终用什么平台？如何提高代码质 量和开发人员的生产率？是否能将模型转换成代码存根？

另外 ,模型所包含的有用设计信息要多于生成代码所需的信息 ,所以我们还要看看其它方式，来利用这些已捕获的重要而有价值的信息。这包括文档的生成、测试用例、部署脚本等，这样就能显著提高项目的整体生产率。众所周知 ,实际的代码编写只是整个项目的一部分工作而已。

没有什么银弹。所需的代码并非都能自动生成（除非你的领域非常小）。最后，你必须处理模型和代码，MDD 则会指导你利用模型、保持代码与模型之间的同步。

不过双向工程怎么样呢？如何利用自动化保持不同抽象层次之间的模型同步呢？这也是一般方案中较为棘手的问题。例如，从较高层次的模型向较低层次的模型转换 时，许多元素会展开——一个元素会在较低层次上演化出多个元素。一旦创建了较高层次的模型，用户就可以更新、移除、添加较低层次上的模型元素。那又该如何 映射回较高层次的模型去呢？若干组详细的元素又怎样转换/映射到少量的高层次元素呢？面临这样的挑战，就很有必要想清楚，追求的这种方法到底是不是开发方法的一部分。

由于修改极可能在代码级别发生 ,所以若没有保持模型和代码一致的方法 ,模型很快就只剩文档了。最近，Rational Software Architect 之类的工具在“保持一致”方面有了很大的改进，提供了可视化代码、比较和合并的功能。请注意 ,用于协调这些变化的方法比工具化的能力更 为重要，这和治理也是相关的。举例来说，架构师看到了代码和模型之间的差异，怎么办呢？去和开发人员讨论？让开发人员修改代码？还是架构师修改模型？正如 你所看到的，这些都不是完全自动化的方法。

已经取得巨大成功的另一个方式是预先在工具化上投资（要么购买要么定制），通过约束、规则和假设减小问题空间。对问题空间所能做的限制越多 ,生成高比例解决方案、减少抽象层次、消除双向工程需求的可能性就越大。在这种情况下，今后的关注点只需放在工程上。

挑战：平台无关性面临挑战

虽然不确定平台无关性发生的时间或原因，但是在高层次上进行建模、然后生成解决方案的想法已经引起了广泛关注。或许平台无关性来自于 MDA 的平台无关模 型，也或许来自其它地方。不管来源如何，都要认识到很难从很高层次的内容进行延展，也很难将一种表示定位到许多不同类型的实现上去。已经有一些解决方案能 让用户利用模型生成全部的结果代码了。但在那些情况下，也正如前面小节中所讨论的，工具化对领域来说很有针对性，而且利用了一组约束、规则和假设才使转变 成为可能。解决方案空间比较狭小，这样才为生成高层次的内容提供了可能性。随着解决方案空间的扩展，生成会变得越来越困难。

就连迁移到 DSL 上也会提出这样的问题：使用相同的模型作为输入，生成不同的底层实现有多容易。在利用 DSL 的时候，关键应该是具体的领域和当前的项目。 正如从许多敏捷过程中（以及自己的经验）学到的，过度工程化、计算每种可能性都要付出代价。这同样适用于建模和使用的语言。针对具体领域并不一定就是什么 坏事，事实上它反而是最佳利益。不过，创建一个领域特定的解决方案，再大范围地加以应用是不切实际的。

挑战：保持编码人员的创造力

在我们转向 MDD，期望简化设计表达、改善沟通、生成部分解决方案的时候，我们还需要认识到这会对团队产生影响。有些团队成员可能喜欢在较低的抽象层次工 作；他们也许会在场景建模时觉得拘束，反而在努力实现解决方案的时候感到自如。这些担心并非都是合理的，但还是要听出“弦外之音”。我们需要保证每个团队成员都能发挥最大的作用。

即使在处理模型的时候，我们也需要底层实现的相关专业知识。应该使用什么框架？这些框架如何整合？下面以模式为例进行说明。构建模式实现的关键输入是参考 解决方案，也就是样例，它用来决定模式实现应该做什么以及怎么做。如果我们要构建自己的模式实现，那谁来构建样例？谁来判定该样例是不是解决问题的最佳方 式？既然期望能简化建模体验，那又由谁来给出规则、假设和约束呢？又该怎样把它们编辑到人人都要用的工具中呢？这些问题都强调，有很多地方都需要专业知 识、创造性、以及解决问题的技巧。MDD 策划、启动时有一点非常重要，那就是与团队成员沟通这些挑战，并确保每个成员都能以有建设性、有效率的方式为项目 效力。反思一下过去的项目，真正创新的工作花费了多少时间？而机械、乏味、重复的任务又占用了多少时间？

挑战：没有可利用的内容

和其它相对比较新的方法一样，在最佳实践被充分理解和基础设施就位之前，产出的内容都很有限。现在 MDD 在软件行业越来越成熟，有了越来越多的推进力，可以看到，高质量的 MDD 内容和资产也越来越多。让这些内容从一开始就可用，对采用 MDD 来说是至关重要的。

面对有挑战性的业务问题，仅有工具和基础设施还不足以交付解决这些问题的软件。最终解决问题的往往都不是工具，而是使用工具的人。如果希望大家使用工具，那么最初就有工具的话，情况就会有很大不同。你是否曾经面对过一块白板、一张纸或 IDE 工作空间？如果你一开始就有参考或模板，或者有内容指导、组织你的方法，岂不是更容易一些？

这里讨论的 MDD 内容不仅仅是设计模式或 UML 项目模板。我们所说的内容是指行业或方案的参考架构（比如呼叫中心参考架构或银行参考架构）、作为可执行模 型的行业标准集（比如保险业的 ACORD 或电信行业的 SID）或实现存根的模板大全。这类资产的一个成功案例就是 WebSphere Business Services Fabric（WBSF）的行业内容包（ICPs）。WBSF 框架由运行时和相关工具组成。ICPs 为特定行业（领域）提供了可定制的内容，从而成为框架 的有益补充。这些内容包括不同抽象层次（比如业务、设计和实现）上的模型和模板，它们遵循行业标准，由组织加以裁剪和采用。

这些资产的核心价值在于提供了更多的业务价值，而且更接近组织的战略。换句话说，业务能看到它们会影响损益底线。如果我们比较可复用设计模式的价值和行业 框架的价值，毫无疑问，行业框架能创造更高的价值。但行业架构的适用性是很有限的。譬如说，如果是保险业的行业架构，那就无法在电信行业中使用。与此相 反，设计模式的应用与行业无关，但设计模式提供的价值却有限，而且离损益底线更远。跟基于资产的开发（ABD）社区所认可的一样，让内容可定制（技术术语 是“可变点”）有助于扩大其适用性。

要注意的是，此类内容并不局限在高层次的抽象上（比如业务模型）。由于运营资产都是可执行的，所以它们会影响损益底线。例如安全领域的资产，能复用、改变的细粒度访问控制策略。可以确定的是，这些会对损益底线有所影响，人们也能从这里建立到高层次业务安全策略的联系。

误解：MDD仅用于开发

构建软件解决方案的时候，使用模型来指定架构、关联的服务和组件具有很大的价值，从解决方案的其它方面来说也是如此。但这仅仅是 MDD 给组织带来的一部分价值。要想利用模型并从中获益，就没必要把使用范围限定得这么窄。

我们之前曾将业务驱动开发（BDD）作为 MDD 的特例进行了讨论。那种情况下的焦点是业务建模——业务里的过程是什么？它们如何工作？如何对它们进行优化？如果在这部分没有做好，那就会遭遇“无用的信息输入和输出（Garbage In，Garbage Out）”。

此外，我们还能利用模型来支持规范一致性。模型能提供易于理解的表示，详细说明结果方案如何支持规范要求。比如说，要表明组织是如何对细分客户群、业务范 围（LOB）或渠道持续应用某规则的，就能用模型来实现这一规范需求。只提供代码到文档的一致性并不足以成为一个最佳的方案。

如果要增强已有解决方案的功能，又怎么样呢？如果需求‘A’变化了，这对系统又会有什么影响呢？你如何确定 IT 布局中的哪些部分应该进行验证和修改呢？如果不能跟踪从需求到实现的过程，这个问题就很难回答，回答的代价也很高。

在企业里利用 MDD 的例子还包括对企业架构和运作建模的支持，但也不局限于此。虽然目标千差万别，但我们仍期望能够沟通、利用抽象、保证治理、支持一致性、提高生产率。

总结

MDD 带来了很多好处，它能促进沟通、改进业务编排、提升质量、提高生产率。如果你以前关注过 MDD，那现在应该换个眼光来看待 MDD。如果你从没关注过 MDD，那现在可是关注的好时机，因为工具支持已经很成熟了。

MDD 在工具集里有点儿与众不同——就像你不会只使用一种语言，或是某种语言的单个库，为了达到目的，你需要选择合适的 MDD 方法和角度。如果想在项目中利用 MDD，为了找到适合你的方式，你需要认真考虑下面的问题：

- 处于怎样的情境？
- 对建模工具有什么需求？
- 对建模语言有哪些需求？
- 需要哪几个抽象的层次？
- 如何简化并自动化构建好的方案？
- 需要哪些类型的图？需要多少个图？
- 和谁进行沟通？他们要了解些什么？
- 如何确定 MDD 方法和工具能被整个团队采用？

- 如何发挥整个团队的优势，并让每个人都参与进来？
- 问题空间里是否有现成的可用内容？
- 如何利用 MDD 来支持业务？如何利用 MDD 支持 IT？如何利用 MDD 提供业务和 IT 编排？
- 有哪些可用内容？这些内容如何针对你的情况进行定制？

致谢

感谢 Brian Byrne、Greg Hodgkinson 和 Alessandro Di Bari 分享他们的洞见，提出了提高本文质量的宝贵建议。

资源

- InfoQ采访——结合MDD和SOA：
http://www.infoq.com/articles/bertrand-portier-on-mdd-soma
- 使用模型驱动开发和基于模式的工程在devWorks上设计SOA（系列文章）
- 用基于资产的开发实现战略复用：http://www.redbooks.ibm.com/abstracts/sg247529.html
- 使用Rational SDP构建SOA解决方案：
http://www.redbooks.ibm.com/abstracts/sg247356.html?Open
- 揭秘SOA中架构和服务的基本原则——第一部分：利用架构和抽象层次创建更好的SOA：
http://www.ibm.com/developerworks/library/ar-archserv1/
- EclipseCon 2008 大会：模式领悟！基于模式的自动化技术实践：
http://www.eclipsecon.org/2008/?page=sub/&id=432
- 模型驱动的开发和相关方法的探讨：在模型驱动的体系结构中应用领域特定建模：
http://www.ibm.com/developerworks/library/ar-mdd4/
- Eclipse建模框架：http://www.eclipse.org/modeling/emf/
- Eclipse流程框架：http://www.eclipse.org/epf/

- 利用SOA、BPM和EA进行战略业务和IT编排：

http://www.ibm.com/developerworks/websphere/bpmjournal/0812_jensen/0812_jensen.html

- 模式：使用IBM Rational Software Architect进行模型驱动开发：

http://www.redbooks.ibm.com/abstracts/sg247105.html?Open

原文链接：http://www.infoq.com/cn/articles/mdd-misperceptions-challenges

相关内容：

- DeMarco反思 40 年软件工程发展之路
- 语言约束和责任感，我们应该信赖谁？
- 跨平台开发——Banshee/Mono启示录
- Wolfram|Alpha，菱形六十面体背后的细节
- 当SOA遇到形式化方法

推荐文章 Articles

Alan Cooper会怎么做？

作者 Naysawn Naderi 译者 曹如进

无论使用的是桌面的，Web的或是手机上的应用程序，用户界面都在其中起着一个重要的作用。Alan Cooper，用户界面设计的思想领袖，在他的《About Face》一书中给出了一些有趣和有用的建议来帮助应用程序创建UI。下面是该书中的部分重要思想。

为中级用户（Intermediates Users）设计

Cooper 认为，不管什么软件产品，它的绝大多数用户都可以归类为中级用户——即那些基本了解如何使用产品和一般使用产品重复进行同样操作的用户。他估计，这样的用户约占使用产品总人数的 80%，而剩下的 20%平均分布在初学者和高级用户当中。

尽管如此，他饶有兴趣地指出，虽然中级用户是产品的主要使用群体，但是他们的需求却往往在 UI 设计中被忽视。他指出，由于管理层经常与初学者打交道，因此 都倾向于满足初学者大部分的需求。而开发人员，应该定义为专家用户，他们在思考时都倾向于将终端用户也看作为专家用户。Cooper 认为，虽然中级用户是 产品的主要使用群体，但是他们在这样的混乱情况下都显得很迷惑。他解释到，三种类型用户的需求由他们熟悉产品程度的不同而不同。专家用户想要快速地干活并且可以定制应用程序，而初学者更多关注的是基本工具的概况。中级用户则倾向于使用工具来重复地做同样的事情，并且不希望界面妨碍他们的工作流程。

使用工具来帮助初学者成为中级用户

虽然只有 10%的产品用户是初学者，但是所有的产品用户都要经历初学者的阶段。因此要特别注意去设计一些界面来帮助新手用户学会应用程序的基础。虽然 UI 设计的目标是帮助初学者转移到中级用户，但是 Coppe 很快指出，一旦一名用户从最初状态过渡到了下一个状态，那么那些帮助都不再有太大用处。事实上，它可能会妨碍用户的工作流程。

大量产品可以帮助培养初学者，但同时它们也干扰着该产品的中级用户。例如在 VS2008 中，

我们在生成的单元测试代码里加入注释来帮助初学者理解一个单元 测试的各个组成部分。虽然它们很好地达到了目的，但是一旦组成部分被理解后，它们就会妨碍用户。因此我们在VS2010 中将它们移除了。想要帮助用户成为中级用户，Cooper 建议，可以在 UI 上添加一些概要信息，来帮助用户如何使用产品。他指出，当用户想要关闭这些信息时，它们应当可以被移除。这可以采用经典的“入门指南”视频的形式来概述产品的功能。Cooper 认为，在线帮助和工具提示实际上更好地满足了中级用户而并非初学者，因为他们的需求略有不同。他提到，中级用户会在帮助文档中查找参考资料，而初学者往往对文档的概要信息更为感兴趣。

少即是多

我喜欢一句话叫做：“不管你的界面有多酷，少一些会更好”。有时在 UI 设计中，我觉得我们忘记了用户是将产品当做一个工具在完成特定的目标。有时我们也被很酷的导航工具所左右，将注意力过多地放在它上面。一个在黑暗中可以闪闪发光的紫色锤子，如果不能够有效的把钉子钉进木头的话，它仍然是毫无用处的。Cooper 是一个简约设计的主张者：在简约设计中每一个选项都应该有目的并且是直接的。Cooper 主张减少用户界面中过多的直接选项。他认为，在一个精心设计的 UI 中，用户界面对用户几乎是透明的，因为它自然地符合了用户的思维模式。

为最可能的操作进行设计，为可能的操作提供方法

Cooper 主张在设计产品和通过应用程序对主线流程进行优化时，应当按照用户的思维模式来做。虽然这常常意味着要在窗口中去掉很多程序员看来很舒服的选项，但是这并不意味着没有方法来做那些复杂精细的事情，而是说访问它们要困难一点。

Cooper 认为，不管用户在应用程序工作流中的哪一个步骤，总会执行一两种活动类型。因此，界面应当被优化来帮助用户找到这些活动，并且通过工作流来指 导他们，而并非每次仅仅列出所有的选项。此外对于用户而言，也应当有方法来让他们访问到非主流的活动，但是 UI 不应当为此而进行优化。我认为 Office Ribbon 在这方面做得很出色，它为最可能的操作进行了优化并且还允许其他可能进行的操作。在 Ribbon 中，用户最喜爱使用的条目只需要 1 次点击，使 用较少的条目需要 2 次点击，而那些不太可能用到的条目需要 3 次或更多的点击。可以很清楚地看到，他们为最可能的操作优化了 UI。

移除错误提示和确认对话框

按照遵循用户的思维模式和移除所有妨碍中级用户工作流程的原则，Cooper 主张移除掉产

品中所有的错误和确认对话框。他认为从统计学角度来看 ,应用程序应该有着很好的正确率，它要做的是为用户提供一个撤销他们操作的选项。Cooper 说，用户都想看到对他们可用的选项，然后进行操作，并希望执行完后可以得到一个确认来告知刚刚的操作已经成功执行。如果一个操作结束后如果没 有确认的话，用户会想知道他们的操作是否已经真正地执行。为了能够与用户交流操作的结果，相比较使用一个弹出的提示框，Cooper 更提倡使用内嵌的状态 提示和积极的听觉反馈。虽然我同意移除所有环境下的确认对话框很理想化，但是要将他它们全部移除显得太过于鲁莽，这不是我的风格。想要实现 Cooper 的建议，开发者需要让用户 执行的每一个操作可逆，这样做不仅代价高，而且在我来看也不是很必要。我喜欢尽全力地移除确认对话框和让操作可逆，但是万一碰到暴力的（ drastic） 操作，例如磁盘格式化和删除文件，弹出一个确认对话框看起来会更好，实际上这也是用户所期待的。另一方面，我同意 Cooper 关于移除所有错误对话框的观点。尽管错误对话框对程序员来说很有意义，但是它们对终端用户的意义却很小。我看到很多用户在他 们的工作流被错误对话框破坏后变得非常泄气。Cooper 提到，虽然它们可以用来标志代码的某个部分发生了错误，但是用户往往都将其理解为“我做错了什么 事情”。当用户被反复告知出错的话，他们会开始讨厌你的产品。我最近在和 Mac UI 打交道，并且很惊讶地看到它们似乎不断地吃掉了自己的异常。显然，出现了问题是不应该呈现给用户看的。

关于作者

Naysawn Naderi是微软Visual Studio测试小组的程序经理（ Program Manager ）。目前他正专注于创建正确的用户体验 ,来让手工测试人员可以更好地进行测试以及更有效地与软件小组人员进行合作。他拥有麦吉尔大学（ McGill University）的电气工程学士学位。他的blog是 Testmundo。

原文链接： http://www.infoq.com/cn/articles/UI-Principles-Naysawn-Naderi

相关内容：

- JRuby GUI MVC框架Monkeybars 1.0 发布
- 比较PureMVC和Cairngorm的GUI架构
- 书摘和访谈：《FXRuby：用Ruby创建精简的GUI》
- 颜色与UI
- Pivot：重新发明Java Applet?

推荐文章 Articles

数据库驱动应用程序中影响性能的反模式

作者 Alois Reitbauer 译者 张晓庆

几乎所有现代应用程序都要通过数据库实现数据持久化。数据库访问层经常要对严重的性能问题负责。一旦遇到数据库的问题，大多数人开始研究数据库本身。正确 的索引和数据库结构对提高性能非常关键。然而，很多时候糟糕的性能或可伸缩性问题的罪魁祸首却是应用程序层，而不是数据库。

应用程序层控制并驱动数据库的访问。这一层的问题不能从数据库上得到补偿。所以要想得到高性能和扩展性，数据访问逻辑的设计非常关键。虽然数据库驱 动的应用程序中使用情况各不相同，但所有问题能够归结到几个反模式上。分析你的应用程序中是否使用了下列的反模式，并且解决他们，能够以最小的代价简单让你的软件更快、更稳定。

对象/关系映射的误用

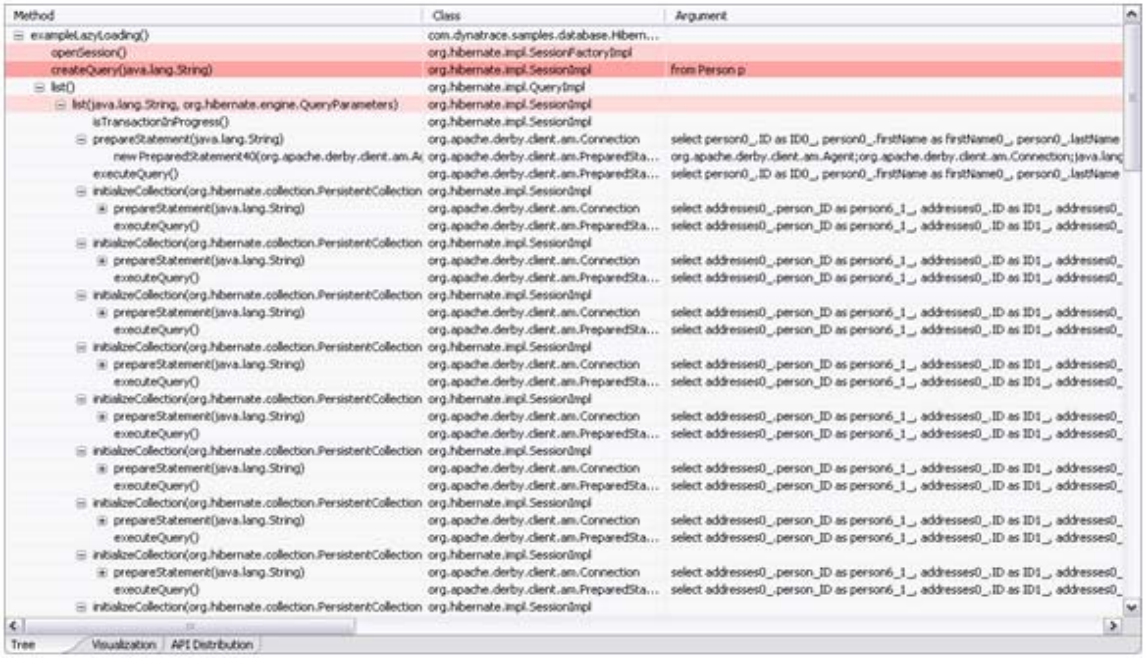
对象/关系映射已经成为现代数据库应用程序的中心部分。对象/关系映射让人从面向对象软件中翻译和访问关系型数据的重担中解脱出来。它们向应用程序人员隐藏了数据访问大部分的复杂逻辑。由于开发人员更专注于实际的业务逻辑，而不是基础架构细节，会使得生产效率更高。对象关系层不需要看到细节就可以轻 松操作复杂的对象图。这经常让人产生错误的印象，认为这些框架让人从设计数据访问逻辑的重担中解脱了出来。

开发人员经常认为数据访问框架很容易就把一切搞定了；然而，不理解内部工作机制就使用对象/关系映射框架，很多时候会导致程序性能低下。主要有两个误解引起了这些问题——加载的行为和加载的时间。

对象/关系映射基于每个对象加载数据。这意味着只有当一个对象被请求或者访问时，需要的 SQL 语句才会被创建并执行。这个原则非常普遍，乍一看多数情况下没问题。但同时它也常常是性能和扩展性问题的原因所在。

让我们看一个简单的例子。在一个存储地址信息的数据库中，我们有一张表存储人和一张表

存储地址。如果我们想得到每个人的名字及其居住的城市，我们不 得不遍历人那张表，然后访问地址信息。下图显示了使用直接（out-of-the box）查询机制的结果。可以看出，这个简单的例子就导致了大量的数据库查询。



这直接引起了对象/关系映射中第二个重要的细节——加载时间。对象/关系映射-如果没有事先告知-会尽量晚地加载数据。这一行为就是延迟加载。延迟 加载保证了数据尽可能晚地加载，目的是执行尽量少的数据库查询，同时避免创建不必要的对象。虽然这个方法通常情况下是可行的，但当它访问那些没有加载的数 据，而数据连接已经不存在时，就可能导致严重的性能问题，以及所谓的 LazyLoadingExceptions。

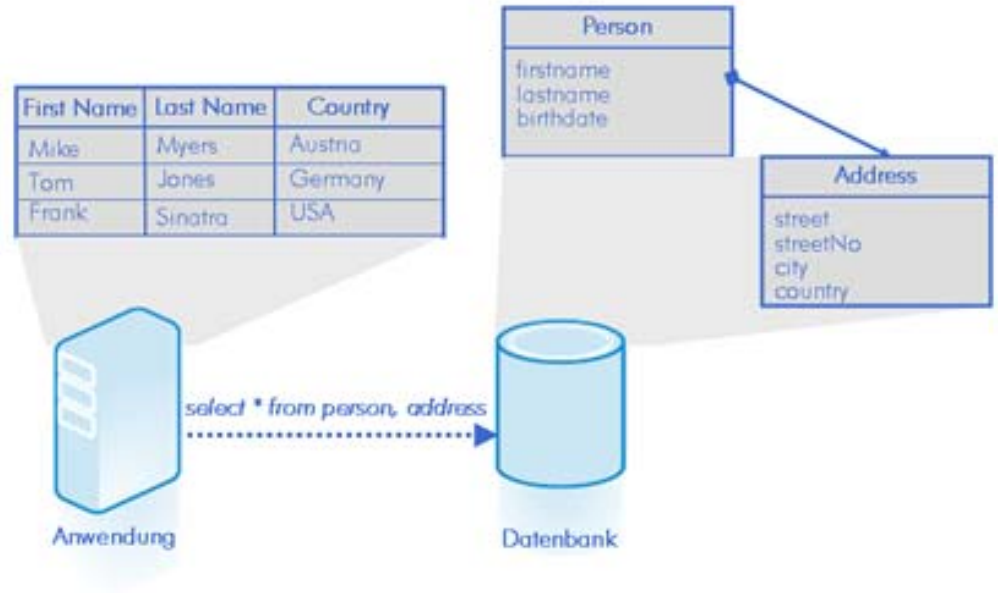
在如上所述的情况下，使用专门的数据查询能够显著提高性能。

因此，虽然对象/关系映射在数据访问的开发方面作用很大，设计合适的数据访问逻辑的重担仍然需要我们挑起。像 dynaTrace 这样带有工具的动态架构验证，能够帮助有效地识别程序中性能的弱点，并能主动解决。

加载了太多数据，实际不需要这么多

数据库访问中经常出现的另外一个反模式是加载了太多的数据，而实际上不需要这么多。导致这样的原因很多。快速应用程序开发工具提供了简单的方式，能 把数据结构和用户接口控制连接起来。由于数据层由领域对象构成，通常它们包含的数据要比实际显示的多得多。

再次使用地址薄作为例子。这一次需要显示人的名 字及其居住城市。两个对象——地址和人——都被加载了，而不是只加载这 3 个字段。这导致了数据库、网络和应用程序层的大量开销。使用专门的查询能够大大减 少查询的数据量。然而这种性能的提升需要额外的工作去维护。表中新增一列可能需要对数据访问层修改多处。



设计的服务接口不合理也经常引起这种反模式。服务接口通常要设计的很通用，以支持大量的用例。其好处是各种各样的用例中都可以使用服务。另外，用例 要比后台服务实现变化的快得多。这会导致服务接口在某些场景下不适合。开发人员然后不得不使用一些补救方法，这可能导致数据访问逻辑效率低下。这个问题在 数据驱动的 Web Services 上经常出现。

为了克服这些问题，开发过程中需要不断地分析数据访问模式。如果是敏捷开发方法，每个用户故事完成后都应该检查数据访问逻辑。除此之外，应该跨应用程序用例分析数据访问模式，以理解数据访问逻辑，这样能够在开发中相应地优化数据访问逻辑。

未充分利用资源

数据库是应用程序中资源的瓶颈，所以使用越少越好。通常情况下大家对数据库连接的使用关注甚少。像任何共享的资源一样，数据库连接会严重影响整个系 统的性能。尤其是 web 应用和使用对象/关系映射框架并用了延迟初始化的程序，会让数据库保持连接的时间比需要的更长。处理开始时获得连接，直到页面生成完成或者再也没有数据访问了才断开。在使用对象/关系映射的应用程序中，连接经常保持着以避免可恶的延迟初始化的问题。通过重

新设计数据访问逻辑，把它从后处理（比如页面生成）中分离出来，应用程序的性能和扩展性能得到极大的提高。

下图展示了 10 个并发数据处理线程的反应时间。第一个使用了 1 个数据库连接，第二个使用了 2 个连接，第三个使用了 2 个连接，但是有 2/3 的处理是在释放连接之后执行的。第三个场景数据访问经过更好的设计，仅用了 1/10 的资源就获得了几乎同样高的性能。

一刀切

一刀切是一种反模式，开发过程中经常见到，敏捷团队中则更多。这种反模式的特征是开发了主要功能之后，所有的数据访问就同样对待，好像它们没有任何区别。然而，区别对待不同类型的数据和查询可以显著提高应用程序的性能和扩展性。

应该对数据进行分析，考虑其生命周期的特性。它是否经常变化，它是可修改的还是只读的呢？数据的访问频率和访问模式，就隐含了一些潜在的代码，比如 可以做缓存。访问频率也暗含了一些线索，比如在哪里做优化更有意义。这可以避免过早进行优化以及不必要的优化，保证了性能调优效果最好。

对数据的使用模式进行分析也有助于调整数据访问层。理解真正使用了哪些数据有助于优化加载策略。比如，理解用户怎样浏览搜索结果对优化 fetch size 很有用。知道了用户是否查看订单详细信息可以给订单选择延迟还是立即加载。

除数据之外，查询也应该被分析并分类。重要的因素包括查询时间、执行频率、是否用于交互用户的上下文或者批量处理的场景中。事务特性有助于更好地调整查询的隔离级别。

比如，在同一个连接中运行用户短暂的交互查询和时间很长的报表查询，很容易导致终端用户的体验很糟糕。报表查询花费的时间很长，会占用大量的数据库 连接，让终端用户的查

询无法拿到连接。通过给不同的查询类型使用不同的数据库连接池，会使终端用户的性能更可预测。降低数据库查询中不需要的隔离级别，也 能引起性能和扩展性的显著提高。

糟糕的测试

最后，缺少测试或者测试不正确是数据库访问应用程序性能和稳定性问题的一个主要原因。最近我曾就这一主题作了一个演讲，并询问听众是否把数据库访问 看作应用程序中一个性能问题。虽然他们都赞成，但没人有这样的测试流程，来测试数据访问的性能。所以虽然这个话题看上去是很重要，大家似乎都没有花时间去 做。然而，即使有测试流程，这也不一定说明测试就是正确的。虽然代码完成后能够立刻发现数据访问逻辑中的很多问题，但通常很晚之后才执行测试，比如负载测 试的时候。由于在生命周期的晚期才改动，可能需要修改架构，从而引起额外的开发和测试工作，这带来了很高的不必要的代价。

而且，必须设计一些测试用例，来测试真实世界的数据访问场景。测试数据访问必须在并发模式下进行，并且使用不同的访问类型。只有结合使用读/写访问才可能识别死锁和并发的问题。除此之外，输入的数据应该多种多样，以避免数据库访问时经常命中缓存，这是不切合实际的。

很多时候人们对预期的负载知之甚少，也不知道去测试哪些负载。很不幸的是，根据我的经验这种情况比比皆是。然而，不能把这当作借口，不定义负载和性能标准。要知道，定义一些标准比一点也不定义要好得多。

如果你对性能数据真的毫无头绪，最好是使用负载渐增测试法，逐步增加负载，直到达到了应用程序的最大值。这样你就知道了应用程序的负载峰值。如果负 载峰值既合理又现实，那就说明你做的不错。否则你得知道在哪方面提高性能。大多数情况下初始的测试表明，应用程序能够处理的负载要比期望的少得多。

结论

数据库访问是影响现代应用程序性能和可伸缩性的一个关键点。虽然框架支持构建数据访问逻辑，仍然需要对数据访问逻辑投入相当的精力，以避免种种陷阱和问题。问题之关键是要理解应用程序数据访问层的动态和特性的一切细节。

作者简介

Alois Reitbauer 是 dynaTrace 软件公司的一名高级性能架构师。在研发部门任职期间，他参与制定了 dynaTrace 的产品策略，并与大客户紧密 合作，为应用程序的整个生命周期实现了性能管理解决方案。Alois Reitbauer 在 Java 和.NET 领域有 10 年的开发和架构经验。

译者注

本文“未充分利用资源”一节中举了这样一个例子：第一个场景使用了 1 个数据库连接，第二个场景使用了 2 个，第三个场景使用了 2 个，但是后处理（比如 页面生成）是在释放数据库连接之后进行。从图表看出，第一个场景的反应时间最长，第三个场景的反应时间与第二个差不多，但是只用了 1/10 的资源。

根据上下文，实际上应该是这样的：第一个场景使用了 1 个连接，第二个使用了 10 个，第三个使用了 1 个，但是后处理（比如页面生成）是在释放数据库连接之后进行。只有这样，文中的结论才合理。

事实上，InfoQ 总站该文后面 Ray Davis 有一个跟帖，就该问题提出了疑问，认为第二个场景的 2 个连接应该为 10 个。除了 Ray Davis 的疑问，译者认为，第三个场景的 2 个连接应该是 1 个，这样上下文不矛盾了。

原文链接：http://www.infoq.com/cn/articles/Anti-Patterns-Alois-Reitbauer

相关内容：

- Qizmt：MySpace的开源MapReduce框架
- 别删除数据
- 支持云应用程序服务的PHP API
- Goat Rodeo：面向Web应用的统一数据模型
- Entity Framework窍门

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

新品推荐 | New Products

Dojo 1.4RC1 发布：性能提升、新的Editor插件

作者 张凯峰

著名开源 Javascript 工具库 Dojo 在两次发布 1.4 beta 的一个月后 ,终于在近日释出 1.4 RC1，同时也期望这是唯一一次的 RC 版本。

原文链接：http://www.infoq.com/cn/news/2009/11/dojo1.4rc1

RubyMine 2.0：动态开发的指路灯

作者 Robert Bazinet 译者 杨晨

市场上首批 Ruby IDE 中，JetBrains 集中精力开发的 Ruby IDE RubyMine 曾有一席之地。而今，自从 1.0 版本发布后 6 个月，RubyMine 2.0 发布了。

原文链接： http://www.infoq.com/cn/news/2009/11/rubymine-20

Flash Player 10.1 及AIR 2.0 Beta版发布，支持多点触摸

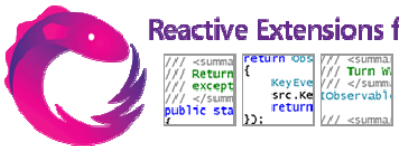
作者 Jon Rose 译者 张龙

近日 Adobe Systems 发布了 Flash Player 10.1 及 AIR 2.0 Beta 版，大家可以从 Adobe Labs 站点上下载。

原文链接：http://www.infoq.com/cn/news/2009/11/flash-10.1-and-air-2.0

微软发布Reactive框架，简化异步及事件驱动编程

作者 赵劼



微软近期于 DevLabs 发布了 Reactive 框架，目的是简化异步及事件驱动程序的构建，尤其适合如 Silverlight 或基于云服务的应用程序。 同时，Channel 9 对其主要设计者 Erik Meijer 进行了采访，Erik 谈论了 Reactive 框架的设计原因，思路及可用场景。

原文链接：http://www.infoq.com/cn/news/2009/11/Rx-Preview

Google试验新语言——Go

作者 Abel Avram**译者** 侯伯薇

Go 是 Google 推出的一种试验性的开源新语言，有点像 C 但加入了反射、垃圾收集、动态类型、并发和并行等特性。

原文链接：http://www.infoq.com/cn/news/2009/10/VS-2010-Beta-2

JIRA 4.0 发布：功能改进，价格更低

作者 Craig Wickesser 译者 张龙

近日 Atlassian 发布了 JIRA 4.0——使用广泛的一款缺陷跟踪、敏捷项目管理与工作流产品。InfoQ 有幸采访到了 Atlassian 团队就此次发布与未来的路线图进行了探讨。

原文链接： http://www.infoq.com/cn/news/2009/11/jira_4

JRuby 1.4 正式发布，修正大量Bug

作者 丁雪丰

2009 年 11 月 2 日，在经历了 3 个 RC 版本之后，JRuby 社区终于迎来了 JRuby 1.4 的正式版本。除了修正大量 1.3.1 依赖的 Bug 以外，新版本将 Ruby 1.8.7 作为最低 Ruby 版本，且改善了 1.9 的支持，对 Windows Native Launcher、YAML、Java 集成等多方面做了改进。

原文链接：http://www.infoq.com/cn/news/2009/11/jruby-release-new-version

Google发布Android SDK 2.0

作者 张龙

近日 Google 发布了 Android SDK 2.0（代号为 Eclair），此次发布为开发者与最终用户提供了大量激动人心的新特性，然而对于已有和即将上市的 Android 手机的升级问题，Google 并没有给出明确的回答。

原文链接： http://www.infoq.com/cn/news/2009/11/google-release-android2

推荐编辑 | Ruby 社区编辑 丁雪丰

大概两年前，我发现了一个叫 InfoQ 的网站，当时觉得这个网站内容还不错，后来认识了一些 InfoQ 的朋友，也许因为都热衷于技术，大家相聚甚欢。我偶尔也会给 InfoQ 提点建议，但直到今年 9 月参加敏捷中国大会时，我都没有想过自己有一天也会成为这个团队中的一员。

人生中总是充满了各种各样的巧合，所以如果要问我为什么加入 InfoQ，我会回答这是一个巧合。在一个恰当的时刻，一个恰当的人向我伸出了橄榄枝，恰巧我对此很感兴趣，于是就答应了。

这里有很多老朋友，更有不少新朋友，能在这么一个充满激情的团队中与众多高手共事，我深感荣幸。虽然和很多编辑都素未谋面，但共同的目标把我们聚集到一起，相信这个团队中的每一位成员都希望能为推动国内技术社区的发展贡献一份微薄之力。

每天都有那么多人访问 InfoQ，从这里获取信息，因此每条新闻、每篇文章，都要力争精准可靠；IT 领域知识更新很快，这要求每位编辑都要关注技术的发 展，并将所见所得分享给所有网站的读者……有挑战才会有乐趣 ,在 InfoQ ,我们竭尽全力为读者带来更多更好的内容，虽然这份工作充满了艰辛，但每一位参 与其中的人体会更多的则是成长的喜悦，所以很多编辑都说自己是在和 InfoQ 中文站、和中文站的读者们一同成长。

InfoQ 正肩负着自己的使命大步向前，这期间离不开大家的支持和帮助，如果你不想只做一个看客，那请不要犹豫，这里绝对有你发挥的舞台。

封面植物

蝽实

蝽实，稀有种。落叶灌木，高 150 至 300 厘米，分布于山西、河南、陕西、甘肃、湖北和安徽等省局部地区，零星生长在海拔 350 至 1900 米的阳坡或半阳坡地上。耐寒、耐旱、喜光。花期 5 至 6 月，果期 9 至 10 月。

蝽实为我国特有单种属，因不合理开垦和过度放牧被严重破坏。蝽实花密色色艳,开花期正值初夏百花凋谢之时,帮更感可贵,夏秋全树挂满形如刺猬的小果,甚为 别致.在园林中可于草坪、角坪、角隅、山石旁、园路交叉口、亭廊附近列植或丛植，北京地区园林中多丛植于草坪、路边及假山旁，也可盆栽或做切花用。

蝽实是秦岭至大别山区的古老残遗成分，由于形态特殊，在忍冬科中处于孤立地位，它对于研究植物区系、古地理和忍冬科系统发育有一定的科学价值。猬实花序紧簇，花色艳丽，是一种具有较高观赏价值的花木。

在湖北神农架已建立自然保护区，正在筹建的河南济源猕猴保护区，已将猬实作为重点物之一。中国科学院北京植物园及欧美植物园有引种栽培。可在分布区内的适生地区繁殖栽培。


1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 12 月刊

每月 8 日出版

本期主编：李明

总编辑：霍泰稳

总编助理：刘申

编辑：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：李明，InfoQ 中文站 Ruby 社区首席编辑

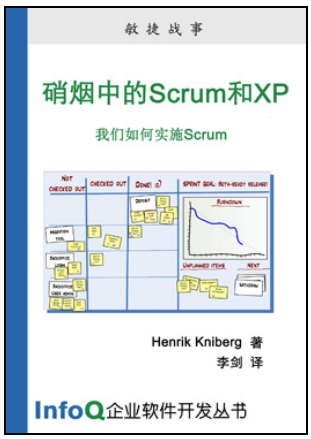
李明（ nasi ），InfoQ中文站Ruby社区首席编辑，毕业于东北大学，曾供职于百度网页搜索部，从事分布式网络爬虫及其国际化的研发工作。目前在某通信公司任系统架构师，进行高性能大规模分布式系统的设计与开发。擅长搜索引擎技术，关注开源社区发展，注重敏捷开发实践。参与翻译《Website Optimization》和《Algorithms in a Nutshell》等多部技术图书，《Google API 大全》的合著者之一。同时他还是一位吉他手，兴趣广泛，乐于分享。可以在 http://twitter.com/nasiless找到他。。

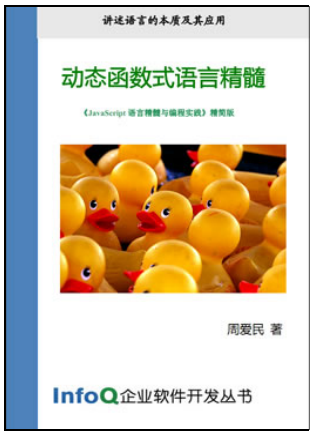
InfoQ中文站
www.infoq.com/cn

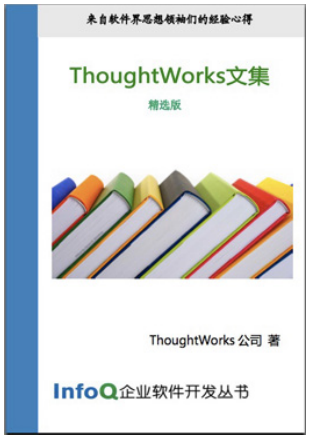
《架构师》月刊由InfoQ中文站出品。

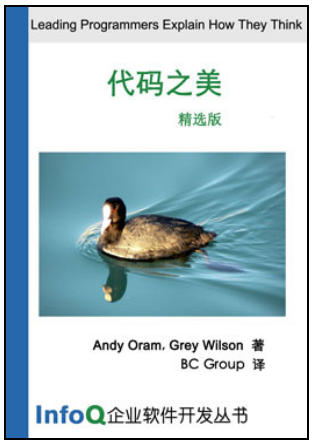
InfoQ企业软件开发丛书

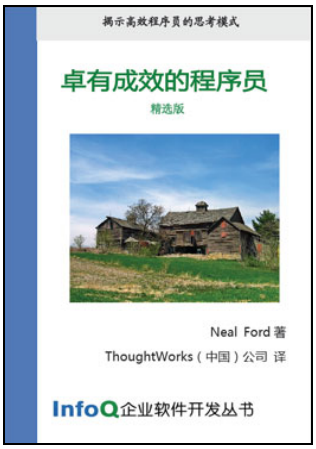
欢迎免费下载

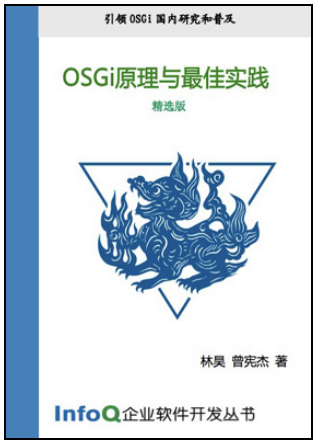












商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com