时刻关注企业软件开发领域的变化与创新

架构师

5月 ARCHITECT

功能小组模型
的过程与质量控制

Apache Pivot初探
基于Java的RIA框架

采用OSGi框架
开发项目的十个问题

新兴行业SOA最佳实践

Azure云计算网格计算

特别专题

RIA技术选型与案例解析

构建前端UI组件的新思路

基于Flash的软件开发

Silverlight在中国人寿的应用

.....

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语 | Editor's letter

不积跬步无以至千里

不久前，我参加了一次企业开发年会。在年会的五个分会场中，架构师与团队管理分会场人头攒动，场面格外壮观。这给我一种错觉，大有“架构师多如狗，项目经理满街走”的态势。然而，当我在演讲到架构与通用设计原则时，分明看到下面听众茫然的眼光。我心底一惊，有种不妙的感觉浮上心头。为了印证这是否错觉，我随意询问了听众几个问题。听众的回答让我的心拔凉拔凉的。原来，距离架构师的百花齐放，我们还有很长的路要走。

对于这个行业，我们最大的问题并非不知其不足而盲目地自大，而是我们的软件从业人员内心已经浮躁了，这其中自然也包括我。每个纯粹走技术道路的开发人员都向往着架构师的头衔，却忽略了若要成为一名合格的架构师，需要经历的沉淀与历练。拿破仑的一句名言“不想当元帅的士兵不是好士兵”，激励着我们向着远大目标迈进。可是不要忘了，我们中国还有一句古话：“一屋不扫，何以扫天下？”

恰好在前几日，我应西南大学之邀，为计算机学院的学生们开展了一次讲座。讲座的题目为“软件从业人员的五项能力修炼”。我偷窃了 Peter M. Senge 所谓“第五项修炼”的概念，不过内容上却是大相径庭。据我过往的经验，我认为软件从业人员必须具备如下五项能力：

- 实现域的逻辑思维能力
- 问题域的分析与抽象能力
- 业务域的建模与设计能力
- 未知域的学习研究能力
- 组织域的交流协作能力

这或许是我的一孔之见，却也是我沉思所得。这里所指的“软件从业人员”其实是片面的，我心底里的思想还是在向架构师靠拢，而且是面向技术设计的架构师。这五项能力正是架构师所必备。如果没有好的逻辑思维能力，就无法提供好的算法与实现，不能写出优美的代码；不具备分析与抽象能力，就很难解决纷繁复杂的需求问题，无法建立一个可以经得起考验的架构蓝图；架构师需要建模，无论是数据模型还是领域模型，无论是计划式设计还是演进式

设计，模型都可以帮助架构师理解需求，进而帮助开发人员理解设计。

人类社会的发展总是充满未知，科学如璀璨的星空指引我们探秘与求索。软件开发同样是一门科学，它的善变是永恒的。站在高端的架构师，如果不具备良好的学习研究能力，因循守旧，故步自封，最后就会湮没在变化的浪涛中。软件开发需要团队，脱离交流与协作的架构师，就如匹马单枪的堂吉诃德挑战风车，结果是头破血流，败得一塌糊涂。交流的重要性不言而喻，一个再好的设计，如果没有合适的方式表现出来，并能够被人理解，它就是无用的。

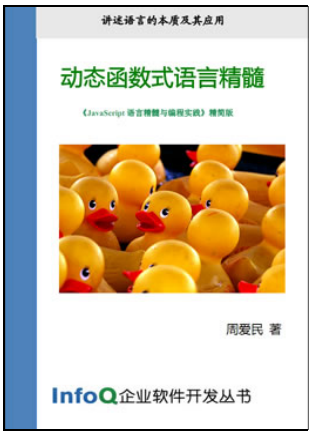
架构师的成长漫长而充满艰辛。是否能够成功，除了需要远大的目标，还需要脚踏实地。最近，我阅读了两本好书。一本是 Kent Beck 所著的《实现模式》，另一本是 Robert C. Martin 的《代码整洁之道》。他们是举世公认的设计大师，但在这两本书里，他们谈的不是架构，而是代码，是实现。荀子曰："不积跬步，无以至千里；不积小流，无以成江海！"架构师站得高，所以望得远，但如果根基不稳，就极容易跌下来，摔得屁滚尿流。

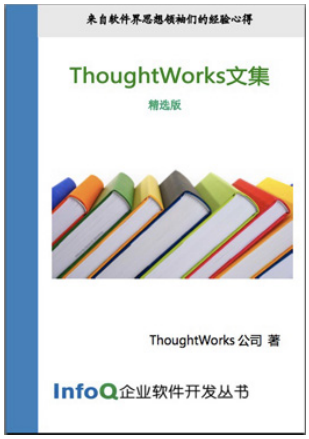
窃以为， 不浮躁而保持虚心，不畏难而坚定执著，不偏狭而开放进取，此乃架构师成功之道。不知读者诸君，意下如何？

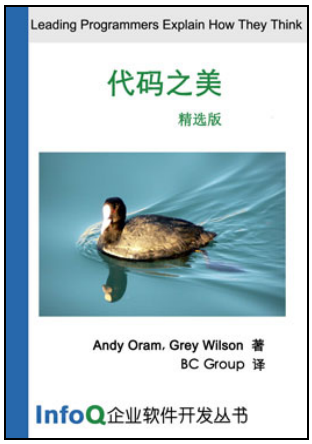
本期主编：

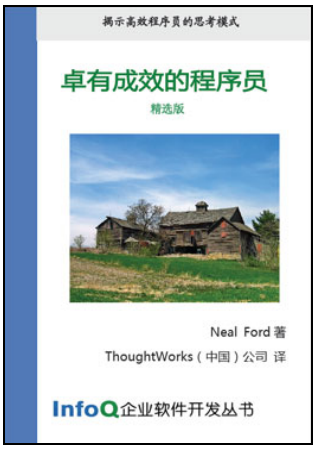
InfoQ企业软件开发丛书

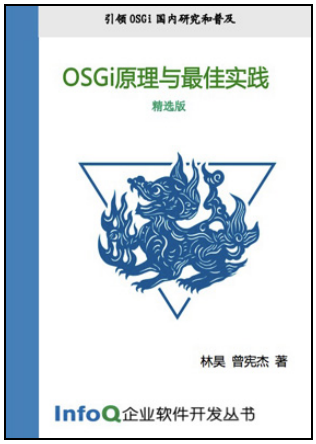
欢迎免费下载











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]

不积跬步无以至千里 I

[人物专访]

功能小组模型的过程与质量控制 1

[热点新闻]

ANGLE为在WINDOWS下进行WEBGL渲染提供了便利 8
APACHE PIVOT初探——基于JAVA的RIA框架 10
SCALING OUT最流行的社交游戏，FARMVILLE 12
ORACLE宣布GLASSFISH路线图 15
DIGG与REDDIT加入NOSQL阵营 17
CLOUDCAT支持在AMAZON EC2 和GOGRID的云上部署WEB应用 20
JAVA EE 6 BEAN VALIDATION提供了实体验证元数据模型与API 23
采用OSGI框架开发项目的十个问题 27
ASP.NET MVC　2特性汇总 29
JRUBY近况 31
新兴的行业SOA的最佳实践 33
注释还是不注释，这是个问题 36

[推荐文章]

基于AZURE云计算平台的网格计算（第三部分） 40

在不改变语言的前提下如何推进JAVA的不断演进.....59

面向资源的架构：REST的另一面.....64

使用APACHE SHINDIG为客户应用程序加入社区内容.....74

书评：《代码之道》.....88

[新品推荐]

微软发布ODATA SDK和“DALLAS” CTP2.....92

ECLIPSE GIT插件EGIT发布了.....92

WINDOWS PHONE 7 SERIES将不支持本地代码.....92

SPRINGSOURCE发布TC SERVER SPRING EDITION.....93

WINDOWBUILDER PRO V7.4 发布：更棒的性能、更快的解析.....93

U-PROVE:在保护隐私的情况下提供安全性.....93

OPENCREDO宣布为SPRING INTEGRATION增加AMQP支持.....94

[特别专题：RIA 技术选型与案例解析]

导言.....95

构建前端UI组件的新思路.....96

基于FLASH的软件开发.....102

案例分析：SILVERLIGHT在中国人寿的应用.....107

漫谈B端的沙箱技术.....114

评论：前端的HTML5 时代来临.....119

[封面植物]

瑞丽山龙眼.....122

人物专访 | Interview

功能小组模型的过程与质量控制

InfoQ 中文站特邀编辑滕振宇最近采访了微软亚太研发集团服务器与开发工具事业部的部门经理 Ramesh Rajagopal。在采访中，Ramesh 从过程控制、架构与设计的控制以及测试组织等方面分享了他所带领Visual Studio软件生命周期管理工具团队使用敏捷方式组织管理大规模软件团队方面的经验，并就如何在组织中引入 Scrum 提出了自己的建议。

Ramesh Rajagopal 是微软亚太研发集团服务器与开发工具事业部的部门经理，在上海带领 Visual Studio 软件生命周期管理工具团队，帮助 C/C++开发人员在 Windows 平台上创建世界级的应用程序。Ramesh 多次受邀与独立软件开发商分享 微软敏捷开发的经验。他的大部分职业生涯致力于包括 Visual Studio 在内的软件开发工具研发。Ramesh 拥有北卡罗来纳州立大学的计算机科学硕士学位。

InfoQ :在Visual Studio开发团队中 ,团队一般都监控哪些过程改进方面的数据？Visual Studio 2010 支持对哪些数据的监控？

VS 2010 Ultimate 版本本身自带了一些图表和报表。 我在下面列出其中的一些：

- Team and Individual capacity charts（团队和成员生产力图）
- Burn down charts（燃尽图）
- User Story progress（用户故事进展）
- Task progress（任务进展）
- Builds status（构建状态）
- Code Coverage（代码覆盖）
- Code Churn（代码变化）
- Issue Trends（问题趋势）

- Bug Trends（缺陷趋势）

在这里着重介绍的是一个 Code Churn（代码变化）度量。这是一个能够说明系统某一部分代码的变化程度的度量。敏捷开发鼓励通过重构，演进式架构的方式来开发系统，因此不可避免的系 统会有些改动。但是我们还是可以通过这个度量去观察我们系统的稳定性。帮助我们发现问题。

VS 2010 Ultimate 版本中另一个很棒的功能就是仪表板（dashboard）。这些仪表板被组织在一起，它们在一张视图上展示了一组相关的度量数据 （metrics）。这帮助功能小组成员交流各自负责功能的重要信息。例如：燃尽图仪表板包括任务燃尽图、任务进展、用户故事进展、7 天问题趋势以及当前 待处理问题。质量仪表板包括测试计划进展、构建状态、代码覆盖、代码变化等。

敏捷的一项重要原则就是项目进展和问题交流的透明性，只有这样才能保证对问题做出及时调整。功能小组可以根据需要采用不同度量数据。提醒一点：决定 采用哪些度量数据以符合团队和产品相关人员的要求，需要一段时间去体验。因此，如果你对敏捷不是很了解，或者你正在带领一个新团队，建议你选取所有的或者 大部分敏捷度量数据，然后逐步去除那些对团队不必要的信息。

例如，在我们的项目中，某些功能团队在一个 Sprint 中使用燃尽图来跟踪进展情况。有些则不使用燃尽图，因为他们觉得每日站立会议上所提供的信 息，足以用于跟踪项目进展和进行调整了。同样的，有些功能团队可能需要使用团队生产力图辅助他们来进行估算下一 Sprint 能够完成多少任务，而有些团队 则不需要这种方式。

当然，对于整个部门来说，还使用到一些其它的度量来跟踪产品（对于我们来说就是 Visual Studio）的整体进展，例如：主干分支上产品的构建状态、价值主张进展、整体缺陷趋势等。由于产生这些报告所需的数据可以很容易地从 TFS 数据库挖掘 出来，所以创建定制报表相当简单。

InfoQ：Visual Studio 开发团队的 Scrum 完成标准（Definition of Done）有哪些？标准是由谁定义的？又是如何演进的？

很多独立功能小组分别负责多个主题，这些主题集合在一起就构成了价值主张，而这些价值主张帮助达到特定产品发布所要实现的商业目标。管理好复杂的项 目，如 Visual Studio，需要面临很多挑战，包括管理和跟踪不同功能团队所开发功能及之间的依赖关系、每天成千上万行代码的变化、确保签入代码达到期望的质量等等。

因此，在代码从功能分支集成到主干分支上之前，整个部门的项目管理团队定义了一套质量

标准，只有到达这些质量标准才能将代码签入到主分支。这套质量 标准包括下述内容：

1. 完整的功能说明（主要用于与依赖方交流产品设计）
2. 完整的开发设计文档（以设计出产品的各个复杂部分）
3. 完整的测试计划
4. 完整的安全计划和分析
5. 完成代码分析和修复错误
6. 自动化测试达到 70%代码覆盖率（防止回归错误）
7. 完成性能测试并修复性能回归问题
8. 完成本地化测试并修复缺陷
9. 完成 API 检查
10. 完成所有测试用例，包括集成测试，修复所有优先级为 0 和 1 的缺陷

以整个部门的完成标准为基本标准，每一个功能小组还会会有一个自己的一个完成标准，小组通过自己的反省 Retrospective 会议，不断地演进 自己完整标准。对于功能小组，我们为实现用户故事定义一套完成标准。

1. 编写用户故事，并要经过复查
2. 开发人员为产品复杂部分编写设计文档并经过复查
3. 编写测试计划，包括接受性测试、功能测试、性能测试、压力测试以及集成测试，并经过审查
4. 所有签入的产品代码要有单元测试，并都要执行通过
5. 对签入的代码都要运行代码分析工具，并修复报告的所有错误
6. 编写验收测试用例并且要全部通过
7. 功能实现能够满足性能要求
8. 完成探索性测试（Exploratory Testing）并完成缺陷登记
9. 功能、集成、性能、压力和安全测试都要被执行并且通过
10. 修复所有优先级为 0 和 1 的缺陷。

11. 完成风险模型 Threat Model（数据流图，Data Flow Diagram）

InfoQ：Visual Studio 2010 的代码及架构分析功能能够对代码进行哪些方面的分析？从您的经验来看哪几类分析对提高系统的内部质量最有价值？

Visual Studio 2010 提供了多种工具帮助对代码的实现，系统的设计等等进行分析和扫描。对于托管（managed）项目，你可以将代码分析规则组织成规则集。规则集允许你将一条或多条代码分析规则任意组合以方便管理。Visual Studio 2010 自带了不少规则集：

- Microsoft Security Rules（微软安全规则）
- Microsoft Basic Correctness Rules（微软基本正确性规则）
- Microsoft Extended Correctness Rules Microsoft Globalization Rules（微软扩展正确性规则）
- Microsoft Globalization Rules（微软全球化规则）
    - Microsoft Basic Design Guidelines Rules（微软基本设计准则规则）
    - Maintainability Warnings（微软维护性准则）
    - 等等

这些规则集帮助你确定在安全、性能、可靠性、全球化、互操作性（与 COM 对象交互）、可维护性等特定领域中的问题。开发者还可以灵活地创建自定义规 则集。比如，如果你的应用程序没有与任何 COM 对象进行交互，那么你就可以关闭那些规则。

根据我们的经验，打开所有的规则和警告是十分有用的。很显然，有些规则集（如：安全）对于确保开发人员不会在无意中引入任何可被利用的安全漏洞是至 关重要的。运行其他规则集，如性能、设计准则等，也是有价值的。因为这些规则是在设计语言和框架专家的最佳实践基础上制定的，可以帮助开发人员（尤其是初 级开发人员）在开发阶段尽早确定和提高代码质量，也可以帮助他们学习最佳实践，成为优秀的开发人员。例如，考虑下述警告“CA1502：Avoid excessive complexity（避免过度的复杂性）”这条原则会检测方法中的圈复杂度（cyclomatic complexity）。过高的圈复杂度意味着方法里面嵌套了太多的逻辑判断（if, switch）或者循环（while, for），也就是一个方法包含了太多的逻辑，违反了单一职责（Single Responsibility），这会造成代码很难读，系统很难维护。

另外 Visual Studio 2010 还提供了一种层次图（Layer Diagram，详细内容参见MSDN “如何创建层次图”，“如何使用层次图审查代码”）。在Visual Studio中可以方便的对项目或者是Assmebly进行扫描，生成层次图。层次图提供了对系统的一种多层次的逻辑抽象，可以帮助

开发人员对系统的整体 以及局部有一个更好的把握。

在 Visual Studio 2010 中，你可以让代码分析作为构建的一部分运行（我们也正是这么做的），甚至可以将其作为代码签入政策（check-in policy）的一部分。质量控制所要求在将代码整合到主干分支前必须运行代码分析，由于我们从开发周期的一开始就决定采用这种做法，这样就避免了处理成 千上万的警告了。

在一个迭代中，测试的种类有那些？哪些是手动的？功能类的自动化测试有哪些类型（比如基于 UI 的自动化，UI 层以下的自动化，集成测试，单元测试等 等）？如何在功能小组内组织这些测试？

通常在每一个迭代中，我们要编写单元测试、验收测试、功能测试、以及一些压力和性能测试。除了这些之外，我们也会编写覆盖完整用户场景的集成测试， 集成测试一般会跨越产品的多个功能。

单元测试是由开发人员设计并实现为自动化测试，针对系统中加入的单个低层 API。单元测试会与产品代码一起签入到 TFS 版本控制系统。

验收测试和功能测试由测试人员根据项目经理的功能说明书来编写的。

验收测试用例在所有测试用例中优先级最高，通常为 1 级（P1），它们覆盖了功能说明书中描述的最基本/核心的用户场景。验收测试用例保证了产品实现 的功能达到了用户故事描述的设计。通常情况下，功能说明中每个用户故事对应一个或两个验收测试用例。

除了最基本的用户场景之外，产品功能的其它扩展或临界情况则由功能测试用例来覆盖。测试人员编写的测试计划包含了功能测试用例。这些测试用例要根据 所覆盖用户场景的重要性以及用户可能使用产品的方式进行优先级划分。根据以往的经验，一部分功能测试用例会被自动化，其余的用例则作为手动测试。手动测试 的原因有很多，例如：自动化测试技术的局限、时间和资源限制等。

在项目的周期中，测试人员的手工测试以及早期试用者会发现新的产品缺陷。经常有缺陷会揭示出现有测试套件的测试漏洞。对于其中严重的缺陷，如：功能 无法正常工作、产品崩溃或者没有响应，测试人员要为这些缺陷在测试计划中添加新的功能测试用例，并自动化它们以防止再次发生。

通常，验收测试和功能测试是在用户界面（UI）层实现的。在我们的测试团队中，我们利用不同 UI 自动化技术来实现测试用例的自动化， 如：Visual Studio Extensibility Model（Visual Studio 扩展模型，DTE）、Microsoft Active Accessibility（MSAA）、WPF 的 UIA（User Interface

Automation）、以及 Visual Studio 2010 种的 Coded UI Test Framework(Coded UI 测试框架)等。

在完成标准中有些特殊的要求比如性能测试、压力测试等等。这些需要一些特殊的工具或者技巧。为了确保我们还是确保每个团队独立完成每一个用户故事， 减少了对外的依赖。每个功能小组都有相应的人员能够完成这样的任务。为了保证工具的标准以及知识共享，我们会建立有一个虚拟团队，比如所有做性能测试的测 试人员的团队，这个团队会负责发展性能测试的工具，框架以及知识分享。这样我们就能达到一个平衡。

InfoQ：作为从来没有接触过 Scrum 的团队，如果我们想借助 Visual Studio 的 Scrum 模板引入 Scrum，有哪些建议？

在选择何种战略有利于团队采用 Scrum 时，有几个因素需要考虑：团队和管理层对改变其流程的意愿、团队的规模、项目性质等。

根据我的经验，有几种行之有效的办法。第一种是单纯从项目管理角度采用 Scrum。由于 Scrum 是一种围绕项目管理的方法，并不要求技术实践，这 可以使新的流程更容易被接受。当然，团队应继续延用当前的技术实践，因为这些都是项目成功所必须的。这种变化可以逐步推行，以帮助团队解决他们当前问题。 例如，假设团队已在某一次迭代中已无法如期完成功能。项目经理（或 Scrum Master）可以开始使用 TFS 工作项记录迭代（里程碑、Sprint 或其他何种术语）中开发人员的任务，使用燃尽图追踪进度。这些燃尽图可为团队及早 发现和纠正行为提供帮助。又如，如果团队在迭代进行到一半时发现他们已经落后于目标进度了，就可以减少那些最不重要功能相关的任务，以帮助他们赶上进度。 一旦团队认识到了燃尽图的价值，那么在下一次迭代中，Scrum Master 就可以引入另一个新的概念了——比如，制定 Sprint 计划以确保团队在开始时不至于承诺过多的任务。Sprint 计划中包含的用户故事可以 作为工作项录入到 TFS 中，团队可以随后创建“任务”工作项来实现这些故事，并和从前一样开始跟踪进展。现在，假设在下一个 Sprint 中，团队开始发现 已经实现的故事中出现产品质量回归，Scrum Master 就可以引入持续集成（Continuous Integration ）的概念（这是一个技术实践方法，可以极大地帮助团队尽早发现产品缺陷)。这可以通过创建一个 Team Foundation Build 的构造定义来实现。Team Foundation Build 中可以配置构造，使其不仅能编译团队的代码，还可以运行测试。我们可以将构造配置为 Continuous Integration（持续集成，每次签入代码后都会开始一次构造）、 Rolling Build（滚动构造，在构造期间发生的签入都将积累，直到构造完成进入下一次构造过程）、Gated Check-in（门控签入,构造并运行测试，只有在所有测试都通过的情况下才提交变更)等等。这种在 Visual Studio 2010 中逐步引入 Scrum 实践已解决团队当前正面临问题的做法已被证明是有效的。

另一种方法是建立一个小型的功能小组（由愿意尝试变革的团队成员组成），使用 Visual

Studio 来实践 Scrum。当其他团队看到该功能小组是如何确定问题、提出修正、并在每次迭代结束时向客户交付产品后，再让他们也采纳这些最佳实践就 变得容易得多了。

原文链接：

http://www.infoq.com/cn/articles/vs-agile-practice-part2

相关内容：

- Malcolm Groves访谈：工具云以及软件工具的管理与部署
- 潘正磊谈微软研发团队管理之道
- .NET 4 的新特性：图表、SEO及可扩展的输出缓存
- Visual Studio与SQL Server Reporting Services依旧不兼容
- .NET和Mono项目版Git——Git#

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

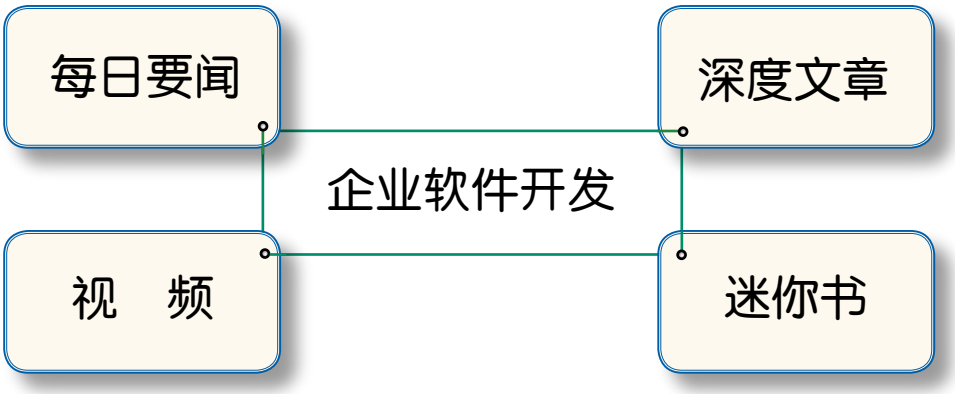
我们的**定位**：关注高级决策人员和大中型企业

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

ANGLE为在Windows下进行WebGL渲染提供了便利

作者 Abel Avram 译者 侯伯薇

为了渲染本地Chrome中的 3D图形，Google使用的是WebGL。 这样做的问题在于WebGL依赖于OpenGL 2.0，而并非所有的Windows系统都安装了它的驱动。ANGLE （近似本地图形层引擎 Almost Native Graphics Layer Engine）项目试图在WebGL和DirectX之间建立简单的层，用来支持Chrome在任意Windows系统中都可以处理 3D图形。

WebGL（Web图形库）是：

> 一种基于OpenGL ES 2.0、跨平台、免版税的针对低级 3D图形API的web标准，它通过HTML5 的Canvas元素暴露出来，作为文档对象模型（Document Object Model）的接口。熟悉OpenGL ES 2.0 API的开发者会发现，WebGL使用的是GLSL基于着色的API，它在语义上与底层的OpenGL ES 2.0 API拥有类似的结构。它与OpenGL ES 2.0 的规格非常相近，只是其中有一些为期望在内存托管语言，像JavaScript，之外使用它所做的特别处理。

实际上，WebGL 让浏览器具备了在本地处理 3D 图形的能力，而所有主要的浏览器厂商——Apple、Google、Mozilla 以及 Opera 等等——都是 WebGL 工作组的成员，Khronos 组（一个非盈利的协会）的成员中，一部分是推动者：AMD、Apple、ARM、Intel、 Motorola、Nokia、NVIDIA、Sun、德州机器，还有很多贡献者：创新、戴尔、EA、Google、IBM、Opera、 Mozilla，以及许多其它与图形领域有关系的公司。

WebGL是在Gecko 1.9.3 中引入的图形特性，它将会在Firefox 3.7 中发布。Khronos指出在Max OS X 10.6 的WebKit每夜构建中会支持WebGL。另外，它也被包含在所有的Chromium版本和

Chrome开发版本中。当前在Opera 中还没有为WebGL提供支持，但据来自Khronos的消息，那是指日可待的。

但是存在这样的问题，当前对 WebGL 的实现是基于 OpenGL 2.0 运行的，而并非所有 Windows 的机器上都会安装它的驱动，或者对于某些图形卡来说根本就不存在。这使得在这样的系统中我们无法通过 WebGL 进 行 3D 渲染。而这也正是 Google 使用 ANGLE 所要解决的问题，它会在 WebGL 和 DirectX 之间创建简单的层，用来在 Windows 中不依赖于 OpenGL 驱动来支持 Chrome 进行 3D 渲染。

ANGLE还会帮助开发者在Windows中创建原型，并且也将手机和嵌入式系统作为将来的目标，因为OpenGL ES本来就是专门针对这样的系统所设计的。 ANGLE已经基于新的BSD许可公开了源码。

原文链接：http://www.infoq.com/cn/news/2010/03/WebGL-ANGLE

相关内容：

- Flash SEO Tool简介
- Chrome 4 现已支持HTML 5 Web SQL Database API
- Google将不再支持老式浏览器
- Chrome中 5 大安全增强
- HTML 5 通过sandbox属性提升iFrame的安全性

热点新闻 | News

Apache Pivot初探——基于Java的RIA框架

作者 Gilad Manor 译者 张龙

Apache Pivot是个开源项目，旨 在通过Java创建一个现代化、富客户端的开发平台。Pivot起初是VMWare在 2007 年的一个研发项目，后来在 2008 年基于Apache 2.0 协议开源了；去年Pivot加入到了Apache Incubator中并于 12 月成为Apache顶级 项目。目前的Pivot完全是由软件开发社区推进的。

Pivot对自己的定位是现代化的RIA工具集，构建在Java2D之上并使用了MVC模式。可以通过Java、JavaScript以及XML（WTKX）编写 Pivot应用。

Pivot带有一个GUI库和一个框架以简化GUI应用的创建，如：

- 声明式UI
- 数据绑定
- 特效与转换
- Web Services集成

Pivot应用既可以Applet的形式也可以独立的桌面应用的形式运行，只要操作系统安装了Java Runtime Environment（JRE）6+（推荐使用Java 6 update 14+） 即可。

根据JVM的安全策略，Pivot应用是受限的。要想使用扩展特性如剪贴板和文件系 统，Applet或独立应用必须要签名并且得到信任。

Pivot与JavaFX的不同之处在于 可以使用Java编写应用，而不是私有的JavaFX脚本语言。Pivot本身是个应用创建工具，可以使用动画和特效增强应 用的用户体验。

接下来的 3 个版本（1.5、2.0 和 2.1）将增强很多GUI组件，稳定性也将有所提 升。主要包括：

- 用于简化 Web 查询中文件传输的文件序列化类（1.5）

- 在Pivot发布包中增加WTKX的DTD以及在examples/documentation中增加DTD引用（2.0）
- 增加对javax.accessibility API的Pivot回调（2.0）
- 增加“HTML provider”API，包装Flying Saucer XHTML渲染库（2.1）

原文链接：http://www.infoq.com/cn/news/2010/03/apache-pivot

相关内容：

- JavaFX的未来在哪里？
- Sun发布JavaFX早期体验版RAD
- 二月份的WPF Toolkit带来 3 个新控件
- Flex开发者需要知道的 10 件事
- David Wadhwani访谈：开放屏幕计划与富媒体平台

热点新闻 | News

Scaling Out最流行的社交游戏，FarmVille

作者 Abel Avram 译者 黄璜

拥有每月 8735 万活跃用户的Farmville是Facebook上最流行的游戏，也是因特网上最流行的web game之一。为了支持scale out，这一应用被部署在云中，广泛地使用了缓存，有能力在顶峰时期关闭一些功能，并很好的运用了性能监测和管理。

按照来自开发该游戏的Zynga公司的开发者Luke Rajlich的说法，FarmVille于 2009 年 6 月上线，4 天之后就达到了 100 万的用户，60 天内就达到了 1000 万。拥有每月 8000 万的活跃用户，FarmVille影响到了20%的Facebook用户以及超过总数 1%的世界人口。要在这样短的时间内在这样的规模下scaling out，需要一定的硬件和软件解决方案。

InfoQ 采访了 Luke Rajlich 以了解更多的架构信息。首先，这一应用运行在云中的虚拟 Linux 服务器上，所以它可以方便的请求和接受额外的计算资源。该应用运行在基本 的 LAMP 栈上，这里的 P 指的是 PHP。该应用广泛的使用了缓存：

> 基于上来说我们的应用是面向对象的，MVC 应用，有一个定制的 DB/Cache 接口。我们对于缓存相当依赖，特别是 memcache，用以支持我们 的负载。同时，我们也有水平分区(shard)的数据库。

为了应付峰值的流量，这一应用依赖于在短时间内增加额外的容量：

> 从架构上来说，我们能够快速的增加容量，因为应用的负载可以被划分到任意的层次（负载均衡，web 服务器，memcache，数据库）。同时，我们 对于在任意层次增加容量有专门的形式化的程序。因此，增加容量的操作很容易管理并且能快速的执行。同时我们运行于虚拟的环境之上，因此我们可以快速的增加 容量而不用预备额外的硬件资源，这一点非常明显的减少了从我们决定增加容量到我们实际增加所必需

> 的硬件资源之间的时间。我们逐步的采用了配置工具，比如 puppet，以此减少了增加额外硬件所需要的开销。剩下的困难的部分首先在于如何知道和发现应用的哪一部分会破坏性能。为了顾及这一考虑，我们对于前面 所提的服务降级投入了许多时间，同时还花费了大量的精力投入到应用性能的监控上面。

这一游戏有好些个部件，当性能成为瓶颈的时候"我们可以有效的关闭在平台上使用的不太重要的功能"，以此缓解对应用的需求：

> 还有好些个其它的组件[在游戏本身以外]，比如朋友阶梯，送礼请求，等等。我们可以将这些元素从游戏本身剥去，因为游戏本身的基本部分不会受到这些 组件性能的影响。这是至关重要的，因为我们的游戏主要是一个基于时间的游戏，用户在特定的时间回到游戏执行特定的操作。如果我们存在宕机时间这些特定的操 作将会产生巨大的用户体验的影响，因此我们想要避免产生这种情况。

这一应用的读写比例高达 3:1，他们广泛地使用缓存来处理它，在跟Todd Hoff的采访中Rajlich揭示了其中的奥秘：

> 用户的状态包含了大量的数据，这些数据具有微妙而复杂的关系。举例来说，在一个农场中，物体相互之间不能冲突，所以如果一个用户在农场上建了一所房 子，在后台需要检查在用户的农场不会有其它的物体占据了重叠的空间。与其它的大部分的主要是读操作的主流网站，比如 Google 或 Facebook 不同的 是，FarmVille 有着相当重的写操作负载。数据读写的比例是 3:1，这是一个高得惊人的写比重。FarmVille 后端最主要的请求之一就是一些更 改玩游戏的用户状态的请求，为了增强伸缩性，我们致力于使我们的应用主要与缓存部件交互。

FarmVille 与 Facebook 平台之间的顶峰流量达到了 3GB/s，所以客户端应用程序需要关闭一些对平台的请求来避免阻塞通信链接：

> FarmVille 与 Facebook 平台之间的流量是巨大的：在顶峰时刻，两者之间的流量基本达到 3GB/s，同时在缓存集群和应用程序之间又有 1.5GB/s 的流量。此外，因为性能可以差异较大，应用程序具有动态的关闭任何到后台调用的能力。我们可以拨号进行微调，以关闭迅速增加的更多的对于后 端平台的调用。此外我们还使得所有到后台的调用都避免阻塞应用本身的加载。这里的思想是，就算所有其它的都失效了，玩家至少仍可以玩游戏。

对于性能监控和管理"我们使用了 nagios 来警报，munin 来监控，以及 puppet 来做配置。我们极大地利用了系统内部状态来追踪程序所使用 的服务的性能，比如 Facebook，DB，以及

Memcache。此外，当我们发现性能降低的时候，我们基于样本来调节一个请求的 IO 事件。”

附注一下，根据Inside Social Games分析师Justin Smith的说法，FarmVille背后的公司Zynga，去年获得了 4.9 亿美元的收入，今年有望超过 8.35 亿。

原文链接：http://www.infoq.com/cn/news/2010/03/FarmVille

相关内容：

- 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
- Ruby VM近况 :1.9.2 增加DTrace支持、Rubinius 1.0 RC2 添加安装程序、IronRuby IDE
- 为网站和智能手机构建FlightCaster前台应用
- 在多核GPU和CPU上使用Accelerator V2 执行并行编程
- Eclipse Awards获奖者名单发布了

热点新闻 | News

Oracle宣布GlassFish路线图

作者Charles Humble 译者 张龙

自从今年一月 Oracle 正式收购 Sun Microsystems 以来，GlassFish 用户始终被阴云所笼罩着。接下来，Oracle 将 GlassFish 应用服务器定位为“部门内部”使用 的服务器，而 WebLogic 依然还是面向那些需要高性能和可伸缩性的企业用户，这么一来问题就变复杂了。其实 Oracle 的这种策略与 IBM 对 WebSphere Application Server 的做法如出一辙，因为 WebSphere Application Server 就有一个基于 Apache Geronimo 的社区版和一个 WebSphere Application Server。很多人怀疑 Oracle 这么做会从 GlassFish 中剥离出很多“企业级”特性，如集群支持等等。或许是为了回应这种质疑，James Gosling（现在是 Oracle 客户端软件小组的 CTO）在不久前于拉斯维加斯举办的 TheServerSide Java 研讨会上指出 GlassFish 应用服务器是首个 EE 6 实现，它可以用在数据中心，其一个月的下载量达到了 100 万次：

> 人们在 GlassFish 上运行了很多大规模的站点。千万别把它当成玩具，它也绝非是玩具。

昨天，Oracle宣布了GlassFish v3 的路线图（pdf文档），结果还是不错的。将于今年发布的 GlassFish v3.1 将提供集中管理、集群和Coherence支持。

展望未来，Oracle 计划在明年发布 GlassFish v3.2，提供改进的集群支持、更好地集成 Oracle 身份管理、虚拟化支持、JavaEE 6 更新以及有限的 Java EE 7 early access API 等等。定于 2012 年发布的 GlassFish v4 将支持 Java EE 7 并与 WebLogic 共享通用的服务器平台。

另一个让人担心的事情就是协议问题。Oracle 已经说过将继续以开源软件（很可能采取现在的 GPL/CDDL 协议）的形式开发 GlassFish，名字叫做 GlassFish Server Open Source Edition。此外，Oracle 还将提供一个商业版（叫做 Oracle GlassFish Server），该版本包含了闭源的插件并可获得 Oracle 的全方位支持，这与收购前 Sun 所采取的方式是一样的。

原文链接：http://www.infoq.com/cn/news/2010/03/glassfish_roadmap

相关内容：

- IBM和Eclipse加大对OSGi的支持
- Eclipse Virgo项目获得批准
- IBM WebSphere 7 添加XPath 2.0，XSLT2.0 和XQuery 1.0 支持
- 微软TechEd 2009 大会发布七项企业级新产品
- Spring 3.0 呼之欲出

热点新闻 | News

Digg与Reddit加入NoSQL阵营

作者Abel Avram 译者 黄璜

Digg和Reddit在这个月都宣布了将转向Cassandra ，因为MySQL对他们来说伸缩性不够了。一些人认为MySQL+memchche不再是事实上的伸缩解决方案了。

Digg在去年九月宣布了他们转向 Cassandra的计划，而这一过程在这个月完成了。仔细比对了其它项目——HBase，Hypertable，Tokyo Cabinet/Tyrant，Voldemort，以及Dynomite——，他们最终选择了Cassandra，出于以下的考虑：

> 每个系统都有它自己的强项和弱项，但是 Cassandra 综合了各个方面。它提供了面向列的数据存储，因此这相比纯粹的 key/value 存储又更 结构化一些。它通过分布式高可用的点对点集群来运作。尽管目前它的核心功能还有一些欠缺，但相较于其它方案而言，它更接近我们想要的。

Digg 几乎重建了整个基础设施，跳出了 LAMP 栈。主要的起因是 MySQL，如同任何其它的 SQL 数据库一样，它是为读取而作的优化，而不能很好的处理写入：

> 我们不再使用用 MySQL 的主要动机，是因为要在一个快速增长看不到头的数据集上，构建一个高性能密集写入的应用越来越困难了。这种增长迫使我们使 用水平和垂直的分区策略，这就消除了关系型数据所带来的大部分价值了，同时仍然会产生各种开销...
>
> 随着我们系统的扩张，跨越多个数据中心的冗余，网络性能，以及增加容量和无宕机时间的更换失效节点对我们来说就尤为重要。我们计划继续使用廉价硬 件，并假设失效是平常的。这些都为 MySQL 增加了难度。

另一个网站，Reddit，曾经出现过memcacheDB相关的问题，一开始他们通过增加更多的RAM来处理，但他们需要一个长久的解决方案是再明显不过的。他们在十天之内完成了向Cassandra的转换，在Cassandra开发者和社区的帮助以及EC2 在测试和部署Cassandra的实例

方面的支持下，只用了一名开发者就做到了。

因为许多重要的网站，比如Facebook或Twitter，都已经在使用或者计划迁移到Cassandra，一些人声称MySQL+memcached 作为事实的伸缩性解决方案的时代结束了。Todd Hoff并不认为MySQL将会在即将来到的某个时间消失，但它不会再代表首位的解决方案：

> 从一个小的角度来看，很清楚 MySQL+memcached 的时代已经过去了。它会坚持一段时间。旧的技术很少完全的消失。一些人仍在骑马。一些人 还在使用 CD。而因特网也不会完全取代电视这种古老的电磁广播技术，但大多数将会奔向一个新的时代...
>
> 我们很清楚 MySQL+memcached 背后的许多思想都是说到了点子上的，这些都在新的系统里得到了保留，只是它的实现显得有点笨重。开发者参 与进来，铺平了缺口，磨掉了棱角，创建了一个新的强健的平台，其本身成为了新生态系统和新时代的根基。

对于Hoff所说的"很清楚MySQL+memcached的时代已经过去了"，Mark Atwood表示不同的意见，他认为memcached 仍将会长期得到应用：

> memcached 作为实现伸缩速度的前沿技术这一时代或许"过去"了，但这并不是因为 memcached 失败，而是因为另外的技术(并不是替代， 而是补充)正在涌现...
>
> 但这不会是 memcached 的终结。可以说，这一高性能的 key-value 存储仍将是一个有用的构建模块，不管是对于其自身，还是作为其它的技 术组件的子组件。
>
> 我肯定 memcache 将会继续演化。将会有更多的实现，更多的局限将会被移除，将会有更多的管理工具，还有更多的系统加上 memcached 的网 络协议，而一些 ORMs 框架在构建时就会假设 memcached 可用，将会有协议方面的特性以及在共享托管和云环境方面的实现。

Hoff 后来在他文章的评论中又说到："我并不是想说缓存将会消失或者 MySQL 将会消失。我坚定的相信内存将是新的硬盘这一概念...所过去了的 是 MySQL 与 memcached 相辅相成的，作为开发可伸缩系统的默认平台。"

尽管 MySQL 与 memcache 仍将会是伸缩性问题的良好解决方案，但已经有新的 non-SQL 方案产生了，而它们看起来能为超大规模的系统带来 更好的结果。

原文链接：http://www.infoq.com/cn/news/2010/03/Digg-Reddit-NoSQL-Cassandra

相关内容：

- Facebook架构师QCon北京 2010 演讲：扩展Memcached实战

- ConcurrentDictionary：.NET 4.0 中新的线程安全的哈希表
- Json.NET性能改进，宣称超越其他.NET序列化机制
- 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
- 在多核GPU和CPU上使用Accelerator V2 执行并行编程

热点新闻 | News

MuleSoft公司的CloudCat支持在Amazon EC2 和GoGrid的云上部署Web应用

作者 Srini Penchikala 译者 侯伯薇

CloudCat是 一种作为Apache Tomcat的servlet容器的云服务产品，它提供了虚拟镜像，允许开发者和QA团队在云环境中构建和测试web应用程序。MuleSoft， 也就是创建了Mule ESB的公司，最近发布了CloudCat产品，它可以被用做在物理内部服务器上托管Tomcat的一种选择。MuleSoft还宣布，与云基础架构托管提供商GoGrid达成合作伙伴关系，从而以云服务的形式来提供CloudCat。他们 之间的组合为开发人员提供了一种方式，可以同时提供云计算和开源软件的好处。

当还没有CloudCat的时候，在云中使用Apache Tomcat除了要安装其它必要的软件之外，还需要对Tomcat进行手动的安装和配置。通过在CloudCat中使用预配置的Apache Tomcat镜像，开发者和操作团队就可以在云中部署和测试他们的web应用程序，而不需要投资购买并存放物理服务器。目前CloudCat已经在Amazon Elastic Compute Cloud (Amazon EC2 AMI) 和GoGrid（GoGrid GSI)的云环境中以云服务的形式提供。它包括了运行在Linux(在GoGrid使用Redhat，EC2 上使用Ubuntu ）和MySQL上 的Apache Tomcat 6 服务器。

Cloudcat 服务器的主要特性包括：

- 为开发和测试 Tomcat 应用程序提供了 Cloudcat。
- 使用MuleSoft的Tcat服务控制台提供了Cloudcat运行时的诊断工具
- 与Apache Maven集成，在开发和测试环境之间 提供持续集成
- 与 Tcat 服务器的 REST API 集成，提供管理和控制
- 针对 Tomcat 应用程序的请求式运行时能力以及远程重启的能力。

InfoQ 对 Mulesoft 的产品管理主管 Sateesh Narahari 进行采访，向其询问了关于 Cloudcat 服务器的发布以及新的与 GoGrid 之间和合作伙伴关系的问题。

InfoQ：发布 CloudCat 的主要动机是什么呢？

Sateesh：在 Mulesoft，我们拥有唯一的主要动机，它推动了所有一切工作，那就是创建出在企业中和在云中都易于使用的中间件。这次，我们专注于 Apache Tomcat，那是我们最喜欢的 Web 应用程序服务器。在开发中 Tomcat 很可靠，并且应用广泛，但是对于 IT 管理员来说，却很难在生产环境中来管理， 因为缺少好的操作工具和商业化的技术支持。当我们想在各种公有云中找到干净的、即时的对 Tomcat 的支持镜像时，发现根本没有。而 CloudCat 正是我们填补该项市场空白的初次尝试。通过使用我们的最佳实践和 Tomcat 的技术秘诀，还有我们在 Amazon EC2 和 GoGrid 提供工具以及为 Tcat 服务器管理服务提供附加价值的经验，我们相信 CloudCat 会得到试图寻找在云上部署应用的企业的青睐。

InfoQ：CloudCat 能够被用于在生产环境中托管 web 应用程序吗，还是仅限于在云中在开发/QA 环境中测试应用程序？

Sateesh：CloudCat 能够用于开发/测试环境，也可以用于生产环境。CloudCat 包含了已经验证过的初始化脚本，可以为 IT 操作员提供可靠和合适的重启以 及服务器控制。

InfoQ：CloudCat 服务器环境也支持负载平衡吗？在 CloudCat 中故障排除是怎么做的呢？

Sateesh：CloudCat可以与已经由云提供商所提供的负载平衡解决方案协同工作。我们在CloudCat自身中不提供负载平衡的能力，但是可以与基础架构提供的 能力协同工作。例如，用户可以在EC2 中使用Elastic负载平衡。

InfoQ：开发者和 QA 团队成员能使用新的 CloudCat 服务器来做性能测试吗？

Sateesh：可以。因为我们将 CloudCat 集成到任何其他云基础架构的提供过程中，这样就节省了提供新的 CloudCat 实例的时间，并且能够满足开发/QA 团队 成员使用 CloudCat 实例做性能测试或者模拟高负载场景的需要。当与 Amazon EC2 协同工作的时候，CloudCat 实例还能够通过可选的 EC 2 插件从 Tcat 服务控制台直接创建。

InfoQ：对于开发、单元和集成测试、调试、应用程序概要分析等等，我们为想要在 CloudCat 环境部署应用程序的开发者提供了 什么样的工具作为支持呢？

Sateesh：在 CloudCat 中可以直接使用 Tcat 服务器，而没有任何附加费用。Tcat 服务器为运行在 Tomcat 实例上的 web 应用程序提供了深层次的诊断和 调试能力。我们为 CloudCat 实例提供了这些能力，同时也在 Tcat 服务控制台提供了同样的能力。此外，Tcat 服务器还提供

了部署的能力，它使得将 应用程序从开发环境迁移到测试环境最终到生产环境变得非常容易。

InfoQ：在新的 CloudCat 服务器上提供了什么样的监控工具呢？

Sateesh：Cloudcat 实例可以从默认的云监控工具中监控，或者还可以从 Tcat 服务控制台监控。当前，Cloudcat 不提供任何警告的能力，但是任何行业领 先的支持云的监控工具都能够监控 Cloudcat 服务实例。

InfoQ：关于新特性，Cloudcat 服务器产品将来的路线图是怎样的呢？

Sateesh：Cloudcat 会始终是 MuleSoft 的战略投资所在。我们期望拓展 Amazon EC2 和 GoGrid 之外的云提供商。有了这个版本的 Cloudcat，我们已经获得了大量用户反馈，那会对产品的路线图产生影响。我们还在寻找更易于在 私有云中使用 Cloudcat 的方法。我们将会在未来和合作伙伴一起在这个领域发布更激动人心的产品。

原文链接：http://www.infoq.com/cn/news/2010/03/mulesoft-cloudcat

相关内容：

- Terracotta和Eucalyptus合作以提供云中的数据管理和弹性服务
- Windows Azure:迫近的成功还是最终的利基？
- Windows Azure现已普遍提供，从免费转为收费
- 云计算呼唤基于事件的API
- 实时Web与云管理标准

热点新闻 | News

Java EE 6 Bean Validation提供了实体验证元数据模型与API

作者 Srini Penchikala 译者 张龙

在此前发布的Java Enterprise Edition（JEE）6系列文章中，我们介绍了Java API for RESTful Web Services（JAX-RS）、 Contexts and Dependency Injection（CDI）、Web 端（Servlet 3、JSF 2）以及EJB 3.1。在本系列文章行将结束之际，我们再来谈谈Bean Validation（JSR 303）——Java EE 6 的一个核心特性，它为实体验证定义了一个元数据模型和API。其默认的元数据源是注解，但开发者可以通过XML描述符对其进行扩展。 Validation API并不依赖特定的应用层或是编程模型，这样同一套验证可由应用的所有层共享。它还提供了通过扩展Validation API来增加客户化验证约束的机制以及查询约束元数据仓库的手段。

在JEE6 的Bean Validation出现之前，开发者不得不在表示层框架、业务层以及持久层中编写验证规则以保证这些规则的同步性，但这么做非常浪费时间而且极易出错。 Bean Validation是通过约束实现的，这些约束以注解的形式出现，注解可以放在JavaBean（如backing bean）的属性、方法或是类上面。 约束既可以是内建的注解（位于javax.validation.constraints包下面），也可以由用户定义。一些常用的内建注解列举如下：

- **Min**：被@Min所注解的元素必须是个数字，其值要大于或等于给定的最小值。
- **Max**：被@Max所注解的元素必须是个数字，其值要小于或等于给定的最大值。
- **Size**：@Size表示被注解的元素必须位于给定的最小值和最大值之间。支持Size验证的数据类型有String、Collection（计算集合的大小）、Map以及数组。
- **NotNull**：@NotNull确保被注解的元素不能为null。
- **Null**：@Null确保被注解的元素一定为null。
- **Pattern**：@Pattern确保被注解的元素（String）一定会匹配给定的Java正则表达式。

下面的示例来自于Java EE 6系列文章，代码中通过Bean Validation注解声明了一些约束：

```
public class Address {
    @NotNull @Size(max=30)
    private String addressline1;

    @Size(max=30)
    private String addressline2;

    public String getAddressline1() {
       return addressline1;
    }

    public void setAddressline1(String addressline1) {
       this.addressline1 = addressline1;
    }
}
```

@NotNull 指定被注解的元素 addressline1 不能为 null；@Size 指定被注解的元素 addressline1 和 addressline2 不能超过给定的最大值，即 30 个字符。

在验证 Address 对象时，addressline1 的值被传递到针对@NotNull 约束的验证类以及针对@Size 约束的验证类中，而 addressline2 的值被传递到针对@Size 约束的验证类中，由相关的验证类进行验证。

如下代码自定义了一个名为 ZipCode 的约束：

```
@Size(min=5, max=5)
@ConstraintValidator(ZipcodeValidator.class)
@Documented
@Target({ANNOTATION_TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface ZipCode {
    String message() default "Wrong zipcode";
    String[] groups() default {};
```

```
}
```

可以将@ZipCode 用在类、属性或是方法上，就像其他约束一样。

```
public class Address {
    @ZipCode
    private String zipCode;

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }
}
```

Validation API

开发者可以借助于 Validation API 以编程的方式验证 JavaBean。Bean Validation API 的默认包是 javax.validation。下面对该包中的一些类进行说明：

- **ConstraintValidator**：这是一个接口，具体的约束验证类需要实现该接口。该接口定义了相关的逻辑以验证给定对象类型中的约束。
- **Validator**：Valida http://java.sun.com/javaee/6/docs/api/index.html?javax/validation/Validator.html tor 接口持有对象验证图的契约。该接口的实现必须是线程安全的。
- **ConstraintViolation**：ConstraintViolation接口表示给定bean上的约束验证失败，它公开了约束违背上下文以及描述该违背情况的信息。
- **ValidationException**：如果在验证过程中出现了某些不可恢复的错误就会抛出ValidationException异 常。某些情况下可以指定该异常，如不合法的分组（group）定义、不合法的约束定义以及不合法的约束声明等等。

约束元数据请求API

Bean Validation 规范提供了查询约束仓库的手段。该 API 主要用于工具支持和与其他框架、库以及 JSR 的集成。Bean Validation 规范旨在为对象约束提供一个验证引擎和元数据仓库。需要进行约束定义、验证和元数据的框架（Java EE 或 Java SE）可以利用 Bean Validation 规范完成这些功能，从应用或是基础设施的角度来看，这么做可以避免不必要的重复工作。

Bean Validation已经集成到了JSF 2.0和JPA 2.0中。在JSF中可以将表单输入 域与域对象的属性绑定起来。JSF 2 和Bean Validation可以判断出绑定的是哪个属性并执行与之相关的验证，还会将约束违背的信息显示给用户。

Hibernate Validator 4是Bean Validation规范的参考实现框架，其最新版增加了不少新特性，如分组验证、与JPA 2 和JSF 2 的自然集成以及扩展的注解集等等。

原文链接：http://www.infoq.com/cn/news/2010/03/javaee6-validation

相关内容：

- Apache Beehive正式退役，迁移到Apache Attic上
- Caucho将在Resin 4.0 中支持Java EE 6 Web Profile
- Java EE 6：EJB 3.1 的变化引人注目
- Java EE 6 Web Services：JAX-RS 1.1 提供了基于注解的REST支持
- Oracle看重Java疏远Sun Cloud

热点新闻 | News

采用OSGi框架开发项目的十个问题

作者 崔康

近期，InfoQ 针对 Java 模块化（基于 OSGi）这一热点技术问题连续做了四篇深度报道：

- 模块化Java简介
- 模块化Java：静态模块化
- 模块化Java：动态模块化
- 模块化Java：声明式模块化

其中对 OSGi 的基本概念和现状以及模块化技术细节做了详细描述：

> OSGi是Java领域里无可辩驳的最成熟的模块系 统，它与Java几乎是如影相随，最早出现于JSR 8， 但是最新规范是JSR 291。 OSGi在JAR的MANIFEST.MF文件中定义了额外的元数据，用来指明每个包所要求的依赖。这就让模块能够（在运行时）检查其依赖是否满足要求， 另外，可以让每个模块有自己的私有 classpath（因为每个模块都有一个ClassLoader）。这可以让dependency hell尽早被发现，但是不能完全避免。和JDBC一样，OSGi也是规范（目前是4.2 版），有多个开源（及商业）实现。因 为模块不需要依赖任何OSGi的特定代码，许多开源类库现在都将其元信息嵌入到manifest中，以便OSGi运行时使用。有些程序包没有这么做，也可 以用bnd这样的工具，它可以处理一个已有的JAR文件并为 其产生合适的默认元信息。自 2004 年Eclipse 3.0 从专有plugin系统切换到OSGi之后，许多其他专有内核系统（JBoss、WebSphere、Weblogic）也都随之将其 运行时转向基于OSGi内核。
> .....

不过，Java社区领袖Adam Bien最近在其博客中认为，从技术角度讲，OSGi的确是实现模块化的可行办法，但OSGi的主要挑战不是技术，而是模块和bundle的管理。他建议在决定采用OSGi框架开发项目之前，考虑以下问题：

1. 针对模块（bundle），采取何种版本 控制方案？大、小版本如何定义？
2. 采用何种软件配置管理策略？允许开放和维 护模块所有版本的分支吗？预计要维护多少个分支？通过 SVN 吗？
3. 在生产环境中，同时存在多少不同版本的模 块？
4. 针对模块和模块组合，如何进行测试？每一 个版本都会显著增加复杂度。
5. 采用何种发布管理策略？提供客户专属的模 块组合吗？缺陷修补/补丁策略是什么？
6. 需要在系统运行中替换模块吗？如何处理正 在进行的事务？
7. 对于 Eclipse RCP 应用，是否应该开放插件给最终用户？
8. 采用何种软件分发系统？很多公司已经有了 一套软件分发系统。应用和 JVM 经常打包到一个二进制文件中整体安装。增量更新几乎是不可能的。
9. 模块之间如何交互？只通过 Java 接口 吗？如果是 那么 JPA 实体的直接关联如何处理？
10. 是否采用 Maven 描述模块和 OSGi？Maven 模块版本会在 OSGi bundle 版本中得到体现吗？

Adam Bien 认为，在深入思考这十个问题之后，OSGi 可能就不再是项目的最佳选择。

读者朋友是否同意 Adam Bien 的观点，针对他的十个问题，有何看法？在实际开发中又是如何解决的？欢迎踊跃发表意见。

原文链接：http://www.infoq.com/cn/news/2010/03/ogsi_introduce_question

相关内容：

- IBM和Eclipse加大对OSGi的支持
- Bundle.update：NetBeans与OSGi
- 模块化的一年
- Java EE中的OSGi、JSR 294 被叫停
- OSGi现状

热点新闻 | News

ASP.NET MVC 2特性汇总

作者 Jonathan Allen 译者 王瑜珩

微软的开源MVC框架ASP.NET MVC 2已经发布了,它同时兼容.NET 3.5 和即将发布的.NET 4。在此版本中，新增了很多新特性：

Templated Helpers可以根据类型选择渲染模板，自动生成用于显示和编辑的 HTML。只要使用Html.DisplayFor(expression)，渲染引擎就会寻找与之相匹配的自定义模板。

区域使你可以将项目分成多个更小的、更容易管理的部分。每个区域都有独立的model、view和controller。

ASP.NET 线程池的大小有一定限制，这是ASP.NET可伸缩性方面的一个已知问题。在CPU繁忙的请求不构成问题的情况下，I/O繁忙的请求也可能会让服务器陷于 无线程可用的境地。性能计数器会告诉你服务器其实并不忙，但是却无法处理更多的请求。通过使用异步Controller，开发人员就可以避免这种情况的发生。

Action 方法的参数可以支持默认值。在支持可选参数的语言中（VB，C# 4），可以通过语言本身实现。而在 C# 3 这样的语言中，则需要使用 DefaultValueAttribute。

DataAnnotations属性可以被用来执行基本的数据验证，这 需要和Model-Validator Providers以及具有插件机制的客户端验证控件一同使用才能完成。

对于 REST 运动的一个窘况是，大多数浏览器实际上并不支持 REST 所需要的 HTTP verbs。通过使用 HttpMethodOverride 方法，框架会生成一个特殊的隐藏字段，以便在 AJAX 请求中模拟 REST verbs。虽然真正的请求还是一个 HTTP POST，但是框架会自动将请求映射到正确的 PUT 或 DELETE action 上去。

原文链接：http://www.infoq.com/cn/news/2010/03/ASP-MVC-2

相关内容：

- ASP.NET MVC 2 中的不兼容修改
- Spring.NET 1.3：VS.NET Solution Templates、MSTest支持及Spring Integration.NET
- ASP.NET的未来：简化开发，HTML 5 及性能提升
- ASP.NET MVC 2 Preview 1 发布
- 23 个.NET开源项目

热点新闻 | News

JRuby近况：JRuby-Prof帮助快速进行性能剖析，JRuby::Synchronized提供自动同步

作者 Werner Schuster 译者 丁雪丰

JRuby-Prof是由Daniel Lucraft开发的一款新的特定于JRuby的性能剖析器。虽然有很多成熟的针对Java的剖析器，但JRuby的混合执行模式让它们的输出难以阅读； 某些JRuby代码会被解释执行，某些JRuby热点则会被编译成字节码。如果使用Java的剖析器，JRuby的内部方法会出现在剖析结果中，占据输出 的前几位，而用户往往只是想知道特定的Ruby方法的调用情况。

JRuby-Prof 通过 JRuby 的钩子方法来获取类似方法调用之类的事件信息，这是一种实现剖析器的有效途径，至少和 set_trace_func 方法相比会更有效一些，set_trace_func 是一个回调方法，在一行代码执行前会被调用。

可以通过 Gem 的方式来获取 JRuby-Prof：

```
jruby -S gem install jruby-prof
```

在GitHub上能找到JRuby- Prof的源代码。

JRuby::Synchronized 模块是最近加入JRuby的新特性。Charles Nutter在JRuby用户列表中解释了JRuby::Synchronized背后的思想：

在 IRC 上和 MenTaLguY 讨论了要给 JRuby 增加一些并发集合后，我们增加了一个新特性：

```
require 'jruby/synchronized'
class MyClass include JRuby::Synchronized ...
```

或

```
obj.extend JRuby::Synchronized
```

include 或 extend 该模块的效果是让对这个类的对象（或单例对象）的所有方法调用都像包围在 Java 的 synchronized 块中一样。不会 存在两个线程同时调用该对象的方法的情况（显然是以对象为基础的）。

这是为 JRuby 提供并发工具的一个尝试——您还希望看到什么新特性呢？

原文链接：http://www.infoq.com/cn/news/2010/03/jruby-jrubyprof-synchronized

相关内容：

- Charles Nutter：为何选择JRuby？
- Rails 3 首个Beta版发布
- Ruby 1.9.2 计划 8 月份发布最终版
- 标准化对Ruby意味着什么
- Rails 3 首个Beta版发布

热点新闻 | News

新兴的行业SOA的最佳实践

作者Boris Lublinsky 译者 马国耀

根据Larry Pizette，Salim Semy，Geoffrey Raines和Steve Foote等合著的MITRE新白皮书，以下几个方面是成功的SOA实施的最佳实践：

- **判断 SOA 是否是最好的选择**，根据白皮书：

  虽然 SOA 能够带来重用、敏捷及松耦合等好处，然而这些好处却不一定总是软件架构师首选的考虑因素。

  例如，实时系统或者网络限制的分布式系统，SOA 可能就不是合适的架构选型。

- **从解决业务及运维中的挑战来开展 SOA 活动**，为了使企业能够完全收获 SOA 实施带来的在时间及成本上的节约，它关注的应该是解决业 务问题而非只关注技术。

  如果企业的 SOA 采用是由解决业务问题而驱动的，并与之对齐，那么企业将会看到共享的服务，业务敏捷，和简化的整合带来的好处。

  作者们给出了一组 SOA 实施的工业案例，在这些案例中，都对业务目标进行了很好的定义。

- **将服务用于支持关键的业务流程的步骤**，SOA 主要的价值主张之一就是通过提供服务并实现能被多个业务流程公用的关键能力来降低 IT 冗余度。

  通过关注跨企业使用的业务流程步骤，服务的重用就自然变成架构的产出之一……将服务定位在合适的业务流程级别是非常必要的（如， 服务所 包含的能力范围），这样它们就能被轻易地映射到业务流程步骤中。

  作者们认为，服务的一个重要特征是松耦合，它能在对服务进行修改时带来灵活性。白皮书在最后还谈到了服务契约规范化以及服务可视化，这通常是通过服务注册 实现的。

- **检查你的数据**， 这意味着 SOA 不会解决你的数据问题，却会把它们暴露出来。SOA 带

来灵活性的一种途径是实现业务和数据间的解耦。这通常会导致创建数据服务，由它们提供 跨企业的通过服务对数据的访问。但是，作者们提醒到，

> 实施 SOA 本身不会解决数据问题。在 SOA 让数据更易于访问的同时，因为数据质量、可用性以及不同社区在词汇上的差异，它依然可能是不可用的…… 缺乏对数据的重视可能会损害 SOA 实施所带来的价值。

作者们认为，在 SOA 实施中成功地融入数据需要：定义数据语义，用于保证在服务消费者和服务提供者之间数据交换的含义一致性；数据质量，用于消除冗 余，不一致性或失效的数据及数据可用性……提供能够支持异构的，或可能是不可预知服务消费者集合的的跨网络的分布式的数据服务。

- **开始于一小点，学习并发展**，白皮书的作者们认为，SOA 项目失败的主要原因之一是不合理的范围。

> ……架构师们往往希望把他们最初的 SOA 项目变成是整个企业范围的工作，即便从某个业务场景来看这并不合理……采纳 SOA 时，最 好从一小点开始，学习并发展。递增的方法允许我们在实施大范围的 SOA 项目之前收集经验和教训；它可以最小化业务风险并能递增地收获回报。然而，在界定 SOA 工作范围时，至关重要的几个方面是：解决有意义的业务问题、关注架构的检验、清晰地描述期望的产出以及对成功的评判尺度，还要吸取经验教训，以备在将来的工作中使用。

白皮书的作者们基于对行业的研究，建议从试验性 SOA 开始，解决具体的业务问题并定义清晰的业务和架构成功的准则。他们还建议要吸取经验和教训，并在开展 下一个 SOA 项目之前把学到的经验教训传授给企业的其他人员。

- **要有远见**，即便从很小的 SOA 开始，追求长期的 SOA 远景也是非常重要的。

> 今天构建的 SOA 的结果可能是面向某个特定服务消费者的耦合服务，可能是系统间两两互联的一些通道。此外，缺乏对将来或者不可知 的使用的计划，SOA 可能无法满足将来的服务需求。因此，在 SOA 实施中，设计时应该考虑到将来可能的需求发展。SOA 的架设应该能支持扩容和扩展，既要 包含 SOA 部署的范围的扩展也要包含需求的扩展。

作者们认为，确保远见的主要步骤包含：服务管理（通过监控），由它提供管理目标是否被很好地实现的信息，它还辅助定位问题及解决问题；确保服务实 现的延伸性……包含基础设施，使能服务交互，可扩展部署及测试活动；创建合适的安全基础设施等。

- **SOA 的关键组件之一是治理的保障.**

  对企业范围内的 SOA 采用中所必需的变更的成功实施需要政策和流程的参与，这主要都是受业务需求驱动的。企业对治理的保障对 SOA 项目的成功是至关重要的。

白皮书中描述的治理的主要方面包括：架构——设立一个最低限度的约束集合来确保服务实施的一致性；基础设施——建立政策来确保基础设施平台（包 括消息传输，安全和其他工具服务）在所有项目中都是标准的；信息——定义数据所有权、建立依附于企业所选的数据标准的政策及方针、设立政策来保障 数据质量度量尺度的遵守等。服务编目——建立政策保证服务声明周期的一致性、策划业务及工具服务、确保开发服务时恰当地利用了现有的企业应用。

- **设定投资回报预期**， 基于行业研究，作者们指出：

  ……尽管节约开支是采用 SOA 的主要动机之一……然而 SOA 采用的一个根本的挑战是，考虑到过高的学习曲线、技术技能和熟悉程度 的缺失、成熟行 业标准的缺失、以及用户企业管理能力的缺失（比如，治理）等情况，如何决定投资 SOA 来制定整体的 SOA 实施。一个企业要施行 SOA 方法，就应该对需要进 行多少投资以及投资回报有理性的预期。

他们强调，与传统的基于应用实施相比，最初的 SOA 实施几乎总要更昂贵一些，而且成本的节约并不会在企业的每个部门发生。在他们看来，大部分成本节 约都是长期的，并且会反映在 SOA 提供的组织敏捷以及共享信息的能力上。此外，服务重用将带来维护成本的降低。

虽然白皮书相当高层，但是，它的确强调了大部分重要的 SOA 关注点。对于正在考虑进行 SOA 实施的人们，它可以的是一个很好的指引，对与 SOA 执行 者 的，它可以作为一个提醒——SOA 不仅关心很酷的技术，还关心架构，业务和投资回报等。

原文链接：http://www.infoq.com/cn/news/2010/03/SOABest

相关内容：

- SOA反原则?
- 像初创公司一样运行企业SOA项目
- SOA概述
- SOA执行者应该首先定义标准

热点新闻 | News

注释还是不注释，这是个问题

作者Abel Avram 译者 张龙

每个开发者都曾在代码中写过注释，有些人的注释很多，为的就是更好地说明代码的意图。本文搜集了关于编写代码注释的一些实践以飨各位读者。

最近，Seattle Area Alt.Net小组的成员们就编写代码注释的必要性和实践进行了激烈的讨论。Kelly Leahy坚持少写注释并编写能够自我说明的代码，因为他相信注释"只会向系统中引入不和谐的声音；一旦修改代码，那些注释就入土为安了"：

> 编写注释对于大多数人来说是个人的事情，但我本人非常反对写注释，因为一旦修改代码，那些注释就入土为安了——这种情况我见得太 多了：注释还在那儿，但代码早就被删除了或是描述的行为被删掉了、代码并没有紧跟注释之后（因为有人在注释和原来的代码之间插入了新的代码）、注释和代码 不一致，因为代码被修改了而注释却没有变化。
>
> 我觉得这种注释比不写注释还恶心，我憎恨注释。

Leahy 并没有完全否定注释的作用，但他只对 10,000 行代码写了一行注释：

> 有时注释还是很有用的，比如设计上有不可避免的约束（性能变化等）时就可以加上一些注释。我尽量不写注释，我们大概每 10,000+行代码会写一行注释 （除了无聊的 xmldoc 注释，我个人觉得这种注释只对于 public API 有用，别的地方一无是处）。

Justin Rudd 说他得在目前所从事的项目中写大量注释，因为里面的 API 实在是"太糟糕了"：

> 目前我正为 Visual Studio 编写一个源代码控制包。API 实在是太糟糕，没办法只能多写注释了，这样我就能想起来为什么在一个地方需要传递 Guid.Empty 而在另一 个地方（看起来很相似的地方）需要传递一个特殊的 GUID，否则一切就都乱套了。
>
> 我对实现的 4 个解决方案事件接口加了注释，这样后面的人即便是看到 7 个方法中有

> 6 个似乎没用也不会把他们删掉了。
>
> 我对方法为何要返回一个特定的 HRESULT 加了注释，因为如果返回 S_OK 的话 Visual Studio 会崩溃（Connect 中是这么说的）。
>
> 虽然文档说可以传递 null，但你不能这么做，我在这个地方加了注释。
>
> 在这个项目中，代码与注释的比例为 2：1。

Chris Tavares 使用注释来标明 bug 被修复了：

> 注释"// doing this because it fixes bug #####"并不是一种坏味道——事实上这是必要的。

然而 Brandon Molina 认为在代码中通过注释来标明修复的 bug 并不好，使用版本控制系统才是上策：

> 如果修复了 10 个 bug 该怎么办呢？代码将充斥着大量无用的信息。在这种情况下应该使用版本控制。注释加上 diff 才是明智之举。

Brad Wilson 补充了一条原则以避免差劲的注释：

> "Why"注释 == good
>
> "How"注释== suspect

Timo Hilden写了一篇关于代码注释的文章，他坚持认为好的注释是非常有必要的：

> 不理会代码注释很容易，但还是让我们面对它吧。作为程序员，我们不指望获得普利策奖（普利策奖是由美国颁发的一个奖项，主要用于奖励那些在新闻报道、文学 创作以及音乐创作方面有杰出贡献的人，该奖项由美籍匈牙利人约瑟夫 普利策创建，现在由位于纽约的哥伦比亚大学负责——译者注），因此写不写注释与表达能力强不强没有什么关系，这仅仅事关懒惰与否。
>
> 坦白地说，更糟糕的是在代码需要注释的时候却没有编写。首先，这间接地说明了程序员缺少对同伴的尊重。每个人都知道查看大量的类是多么不爽的一件事，尤其 是其中包含很多含义模糊、自我描述性差的函数，更有甚至，之前编写代码的人度假去

了、离职了或是由于其他原因找不到了。程序员应该是个艺术家而同伴则是其 观众。作为艺术家，应该尊重我们的观众。

其次，不写注释表明程序员太过于自负了。当然了，在写代码的时候我们很清楚自己在干什么，编程的时候脑子里会有很多想法，但下一次再看自己所编写的代码时 将很可能无法找回当初的想法。以上这些都是常事，这需要我们在适当的地方编写具有描述力的注释。

Hilden并不赞同取消文档（几年前由Scott Swigart和Jeff Atwood提出的观点）的做法，看看下面这个过度注释的代码片段：

```
// Declare category id for products
const int prodCategoryId = 1024;

// Create an iterator over products
vector<Product>::iterator iter = products_.begin();

// Iterate through all products
for ( ; iter != products_.end(); ++iter )
{
    // Assign categody id to each product
    iter->AssignCategoryId( prodCategoryId );
}
```

他建议将上面这些注释合并成一行用于说明代码的总体意图而非每行的意思：

```
// Assign categody id to each product
const int prodCategoryId = 1024;
vector<Product>::iterator iter = products_.begin();

for ( ; iter != products_.end(); ++iter )
{
    iter->AssignCategoryId( prodCategoryId );
}
```

各位读者，你采取何种手段编写注释呢？这些方法是公司的硬性规定还是自己选择的呢？你

支持“编写尽可能少的注释”这种观点么？是否相信后继者能理解 你所编写的代码么？你认为程序员是否需要花时间对代码中那些不太好理解的地方加注释么？

原文链接：http://www.infoq.com/cn/news/2010/03/To-Comment-or-Not-to-Comment

相关内容：

- 语言约束和责任感，我们应该信赖谁？
- 有效进行软件重用的小提示
- 与Tony Hoare爵士一席谈
- “完成”宣言
- 反对if行动

时刻关注企业软件开发领域的变化与创新

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

Java社区：企业Java社区的变化与创新

.NET社区：.NET和微软的其它企业软件开发解决方案

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

SOA社区：关于大中型企业内面向服务架构的一切

Agile社区：敏捷软件开发和项目经理

Architecture社区：设计、技术趋势及架构师所感兴趣的话题

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

推荐文章 | Articles

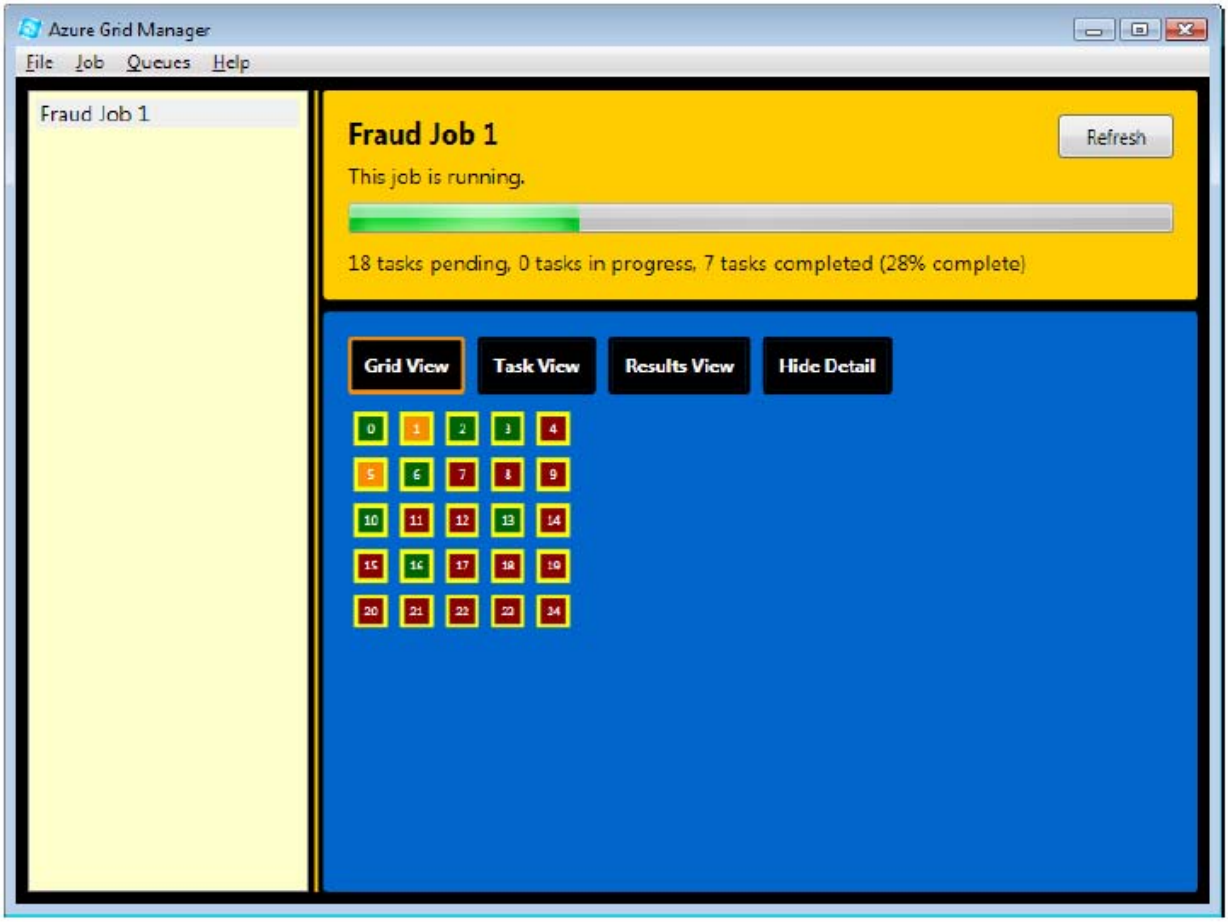
基于Azure云计算平台的网格计算（第三部分）

作者David Pallmann 译者 朱永光

在本系列的第 1 部分， 我们介绍了在Azure上进行网格计算开发的设计模式，并在第 2 部分， 我们编写了一个欺诈评分（fraud scoring）网格计算应用程序，命名为Fraud Check。在这里，第 3 部分中，我们将来运行这个应用程序，首先在本地运行，接着移到云上运行。

我们使用的框架是Azure Grid，是Neudesic Grid Computing Framework的社区版。

在运行我们的网格应用程序之前，有几件事情需要处理——比如配置跟踪数据库和设置配置信息——不过，我们先来一睹应用程序在云中运行起来的效果图：



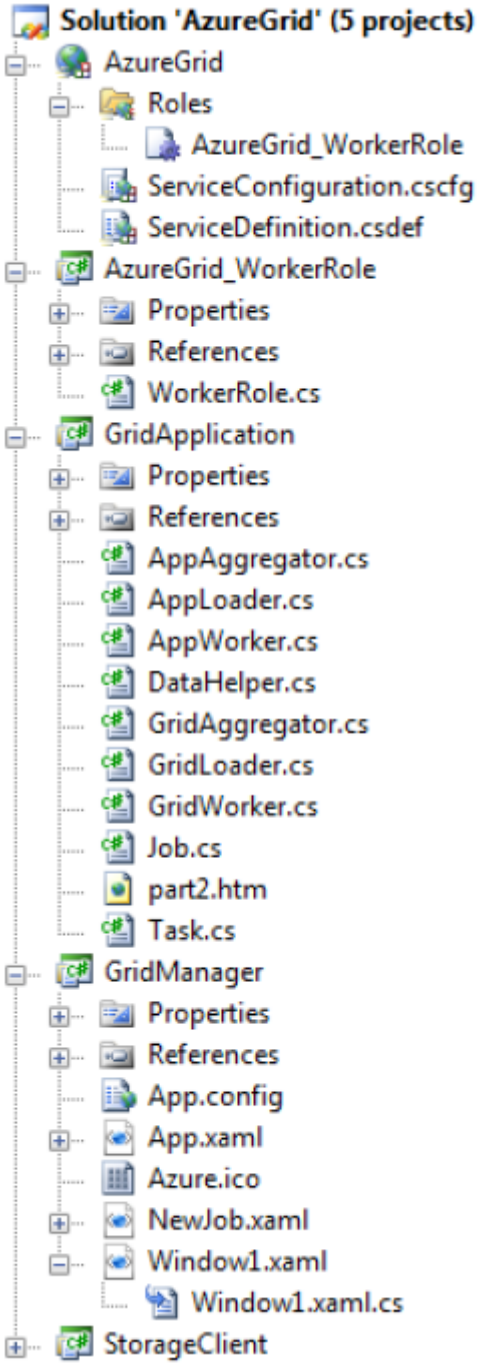
那么让我们来完成如下步骤的设置，以便我们能运行它。

网格应用程序解决方案结构

Azure Grid 为你提供了一个基础的解决方案模板 ,在其中你可以添加特定于应用程序的代码。上次，我们为 Fraud Check 网格应用程序添加了代码，不过并未讲解解决方案结构本身。现在我们就来说一说。

我们的网格计算解决方案具有云中的部分，也有企业内的部分。Azure Grid 框架把所有这些都包含在一个单独的解决方案中，如下所示。那么，在解决方案中都有些什么？

- GridApplication 项目包含了我们在本系列文章第二部分编写的特定于应用程序的代码。回忆一下，我们编写了 3 块代码：任务代 码、加载器和聚合器。解决方案的其他项目含有应用程序的代码：一些代码将在云中执行，一些将在企业系统中执行。
- 在 Azure 中的网格执行器代码位于 AzureGrid 和 AzureGrid_WorkerRole 项目中。这是一个带有执行器角色的标 准 Azure 托管应用程序，可通过 Azure 门户部署到 Azure 托管项目中。你的任务应用程序代码就将在这里面执行。
- 企业端的代码位于 GridManager 项目中。这是一个用于启动和监测网格工作的桌面应用程序。其也在后台运行你的加载器代码和聚合器 代码。
- StorageClient 项目是一个访问云存储的函数库，其继承自 Azure SDK StorageClient 示例。云中的网格执行器和企业系统中的网格管理软件都用到了这个函数库。



要运行这个解决方案里云中和基础系统中的部分都很简单：

- 要运行网格管理器，右键点击 GridManager 项目，并选择 Debug > Start。
- 要在本地机器上用本地开发环境运行网格执行器，右键点击 AzureGrid 项目，选择 Debug > Start。
- 要在 Azure 上运行网格执行器，利用 Azure 门户按照通常的步骤把 AzureGrid 项目发布到 Azure 中，以形成一个托管应用程序。

为了在本地测试所有功能，把 AzureGrid 和 GridManager 都设置为启动项目会很有用，那样只按一次 F5 就可以启动这两个应用程序。

设置本地数据库

Azure Grid 在本地 SQL Server 或 SQL Server Express 数据库（2005 或 2008）中跟踪工作运行、任务、参数和结果。从 CodePlex 下载的项目源码中还包含了创建这个数据库的 SQL 脚 本。

配置解决方案

解决方案里云中和基础系统中的部分都需要少许配置，最重要的地方是和云存储相关的。

云中的网格执行器配置在 AzureGrid 项目的 ServiceConfiguration.cscfg 文件中设定，如下：

- ProjectName：你的应用程序的名称
- TaskQueueName：在云存储中用于接收来自网格执行器结果的队列名称
- QueueTimeout：一个任务在重新被其他执行器覆盖之前可以执行多长时间（以秒计）
- SleepInterval：网格执行器检查要执行的新任务之前休眠的间隔时间（以秒计）
- QueueStorageEndpoint：用于云存储项目或本地开发存储的队列存储端点
- AccountName：云存储项目的名称
- AccountSharedKey：云存储项目的密钥

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="AzureGrid"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfigurati
on">
  <Role name="WorkerRole">
    <Instances count="10"/>
    <ConfigurationSettings>
      <Setting name="ProjectName" value="FraudCheck"/>
      <Setting name="TaskQueueName" value="grid-tasks"/>
      <Setting name="ResultsQueueName" value="grid-results"/>
      <Setting name="QueueTimeout" value="60"/>
      <Setting name="SleepInterval" value="10"/>
      <Setting name="QueueStorageEndpoint"
value="http://queue.core.windows.net"/>
      <Setting name="AccountName" value="mystorage"/>
      <Setting name="AccountSharedKey"
value="Z86+YKIJwKqIwnnS2uVw3mvlkVKMjfQcXawiN1g83JTRycaRwwSSKwhnaNsAw3W9zNW7Lx
GHy2MCJ1qQMX+J4g==">
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

用于网格管理器的配置是在 GridManager 项目的 app.config 文件中设定。大部分设置都具有和云中配置文件一样的名称和含义，且需要 设置为相同的值。多出的一个本地 SQL Server 数据库的连接字符串设置，用于网格应用程序跟踪记录。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="ProjectName" value="FraudCheck"/>
    <add key="TaskQueueName" value="grid-tasks"/>
    <add key="ResultsQueueName" value="grid-results"/>
    <add key="QueueTimeout" value="60"/>
    <add key="SleepInterval" value="10"/>
    <add key="QueueStorageEndpoint" value="http://queue.core.windows.net"/>
    <add key="AccountName" value="mystorage"/>
    <add key="AccountSharedKey"
value="Z86+YKIJwKqIwnnS2uVw3mvlkVKMjfQcXawiN1g83JTRycaRwwSSKwhnaNsAw3W9zNW7Lx
GHy2MCJ1qQMX+J4g==">
    <add key="GridDatabaseConnectionString" value="Data
Source=.\SQLEXPRESS;Initial Catalog=AzureGrid;Integrated Security=SSPI"/>
  </appSettings>
</configuration>
```

跨环境对网格应用程序进行测试和逐级推进

由于网格应用程序能被极大地进行伸缩，理所当然你应该仔细地测试它们。在基于 Azure 的网格计算场景中，我建议按照如下顺序对这个新的网格应用程 序进行测试：

Environment	Compute	Storage	Scale
Development	Local dev fabric	Local dev storage	1 machine, small # tasks
QA	Local dev fabric	Azure storage project	Multiple machines, more tasks
Staging	Azure hosting - Staging	Azure storage project	Cloud-hosted, small # tasks
Production	Azure hosting - Production	Azure storage project	Cloud-hosted, full workload

1. 开发人员测试

用少量的任务在本地开发机器上测试网格应用程序。在此，你要关注的是，验证应用程序是否正确运行，正确的数据是否在网格应用程序和基础存储之间进行移动。

- 对于云运算部分，网格应用程序会在本地开发环境中执行。
- 对于云存储部分，也使用本地开发环境替代。

Grid Manager 当然（也一直会）在本地机器上执行。

2. QA 测试

还是在本地环境 ,在多个执行器机器上用大量的任务测试应用程序。这个测试的目标是验证，同样的工作在单机环境中和在利用云存储的多机环境中都能执 行。

- 对于云运算部分，网格应用程序会在本地开发环境中执行，不过是在多台本地机器。
- 对于云存储部分，使用 AzureStorage 项目。

注意，由于这种方式依旧主要是一个本地测试，只是用到了 Azure 的云存储。在我们无法协调跨多台机器的工作的时候，这样的方式是必须的。

3. 登台（Staging）环境

把网格应用程序部署到 Azure 的登台环境中，用少量的任务来运行它。在这个阶段，你可以验证本地正常运行的工作是否能在云中运行。

- 对于云运算，网格应用程序部署在 Azure 托管项目中——Staging。
- 对于云存储，用到了 Azure Storage 项目。

4. 生产环境

现在，你已经准备好把应用程序推进到 Azure 托管生产环境的级别，用饱和的负荷来运行它。

- 对于云运算，网格应用程序托管在 Azure 托管项目中——Production。
- 对于云存储，用到了 Azure Storage 项目。

你可以使用 Azure Manager 来监测网格应用程序是否正常运行。

本地测试运行

现在我们来准备在我们的开发机器上进行本地测试。根据我们的测试计划，最初是在本地开发机器上用少量负荷进行测试。

输入数据显示在下面的这个 CSV 工作表，它用 Excel 创建并保存为 CSV 文件。Fraud Check 应用程序需要这个输入文件的打开账号和启动 GridManager.exe 的账号一致。在这个测试中，我们将使用 25 条记录。

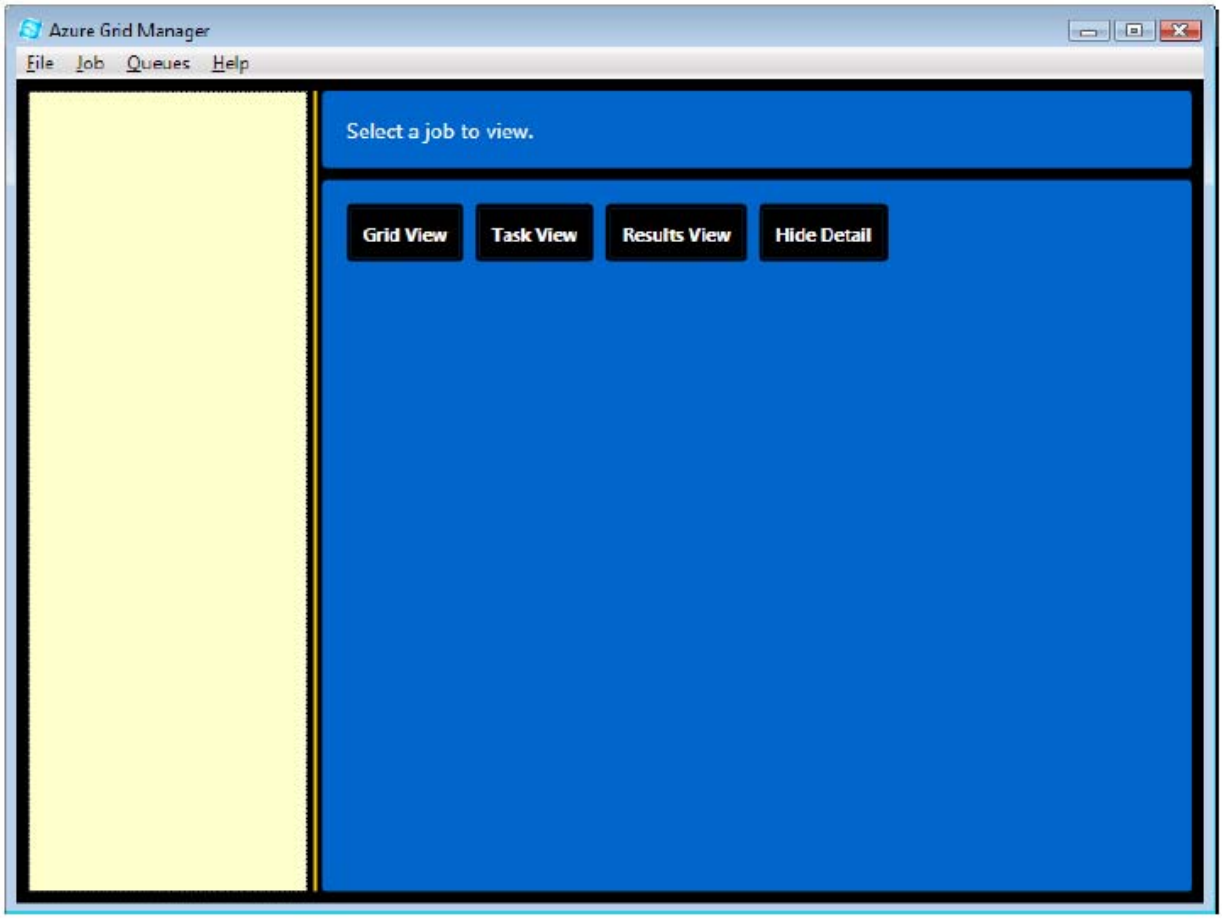
	A	B	C	D	E	F	G	H
1	LastName	FirstName	State	Country	Age	SSN	Relation	MonthsEmployed
2	Beethoven	Ludwig	CA	USA	112	111-11-1111	Applicant	48
3	Bach	J.S.	NY	USA	27	156-77-8935	Applicant	0
4	Mozart	Wolfgang	Tirol	Austria	23		Co-Applicant	3
5	Strauss	Johann	GA	USA	48	095-65-7844	Co-Applicant	150
6	Tchaikovsky	Pyotr	TX	USA	17	110-44-15435	Applicant	6
7	Chopin	Fryderyk	NY	USA	90	110-44-15435	Applicant	1
8	Gershwin	George	NY	USA	81	110-44-15435	Applicant	14
9	Haydn	Joseph	NY	USA	37	110-44-15435	Co-Applicant	7
10	Sibelius	Jean	NY	USA	32	110-44-15435	Applicant	4
11	Brahms	Johannes	NY	USA	55	110-44-15435	Applicant	8
12	Dvorak	Antonin	NY	USA	40	110-44-15435	Co-Applicant	6
13	Mahler	Gustav	NY	USA	76	110-44-15435	Applicant	6
14	Pachelbel	Johann	CA	USA	25	110-44-15435	Applicant	18
15	Handel	George	CA	USA	48	110-44-15435	Co-Applicant	9
16	Vivaldi	Antonio	CA	USA	62	110-44-15435	Applicant	10
17	Bizet	Georges	CA	USA	45	110-44-15435	Applicant	11
18	Debussy	Claude	CA	USA	29	110-44-15435	Co-Applicant	32
19	Grieg	Edvard	CA	USA	32	110-44-15435	Applicant	13
20	Liszt	Franz	NY	USA	31	095-65-7844	Applicant	102
21	Puccini	Giacomo	NY	USA	55	095-65-7844	Co-Applicant	15
22	Mendelssohn	Felix	NY	USA	68	095-65-7844	Applicant	16
23	Rachmaninoff	Segei	NY	USA	73	095-65-7844	Applicant	24
24	Schubert	Franz	NY	USA	85	095-65-7844	Co-Applicant	18
25	Smetana	Bedrich	NY	USA	22	095-65-7844	Applicant	19
26	Verdi	Giuseppe	NY	USA	105	095-65-7844	Applicant	11

为了确保数据库正确设置，云中和企业中的配置都要反映到本地开发存储里。

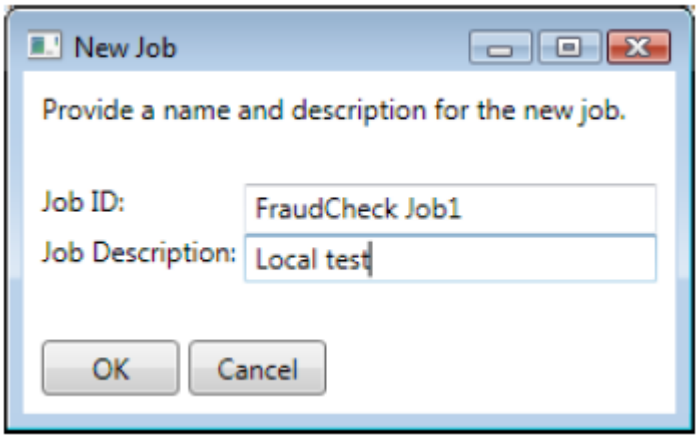
编译应用程序，启动 AzureGrid 和 GridManager 两个项目。

AzureGrid 项目不会显示任何可见的东西，不过如果你想的话可以通过开发环境 UI（Developer Fabric UI）来检查它的运行情况。

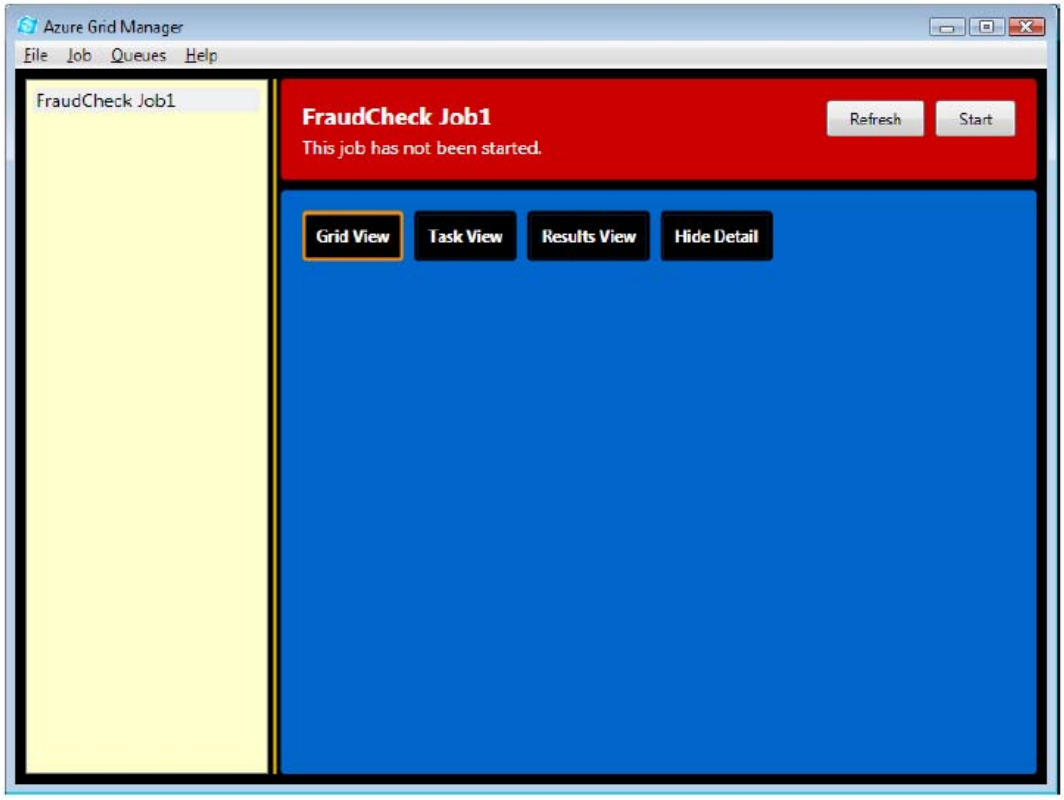
GridManager 应用程序是一个基于 WPF 的控制台，启动后如下面所示：



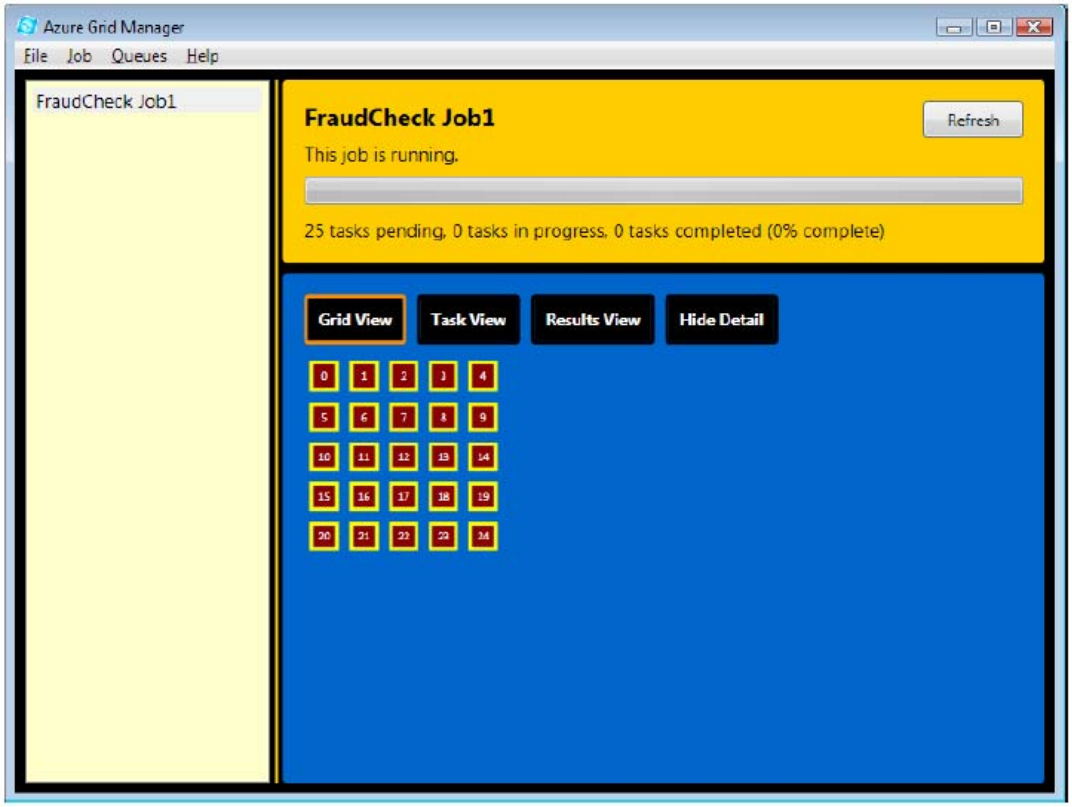
要在应用程序中启动一个工作（Job），从菜单选择 Job > New Job。在显示出的对话框中，填写名称和描述。



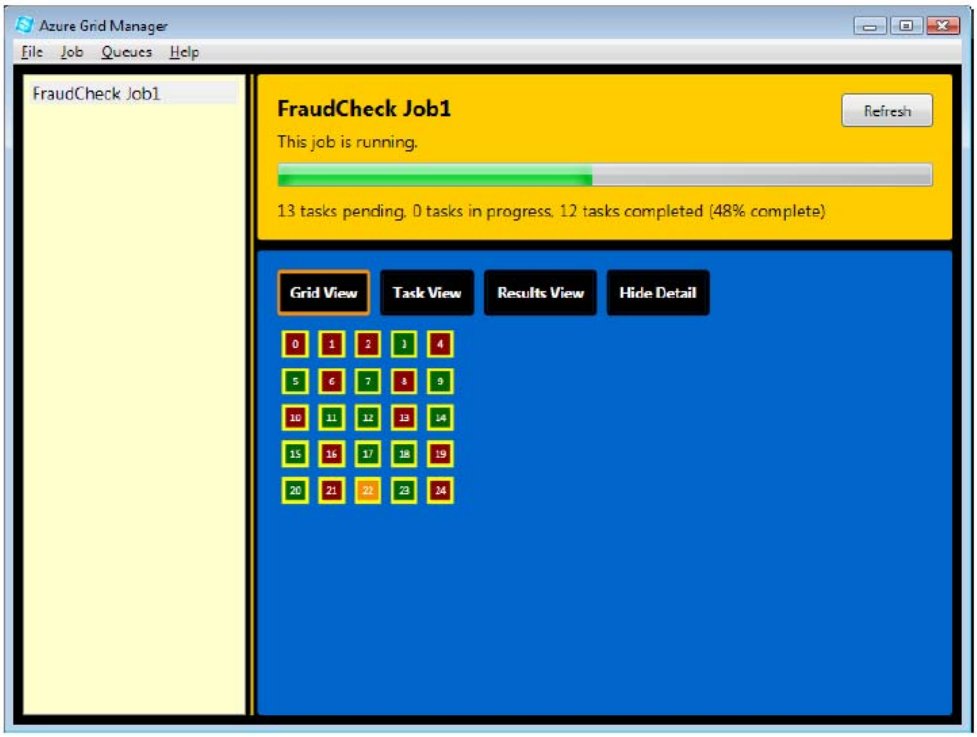
点击 OK 按钮，你将看到工作被创建不过还未运行。工作面板显示为红色以表示工作尚未开始。



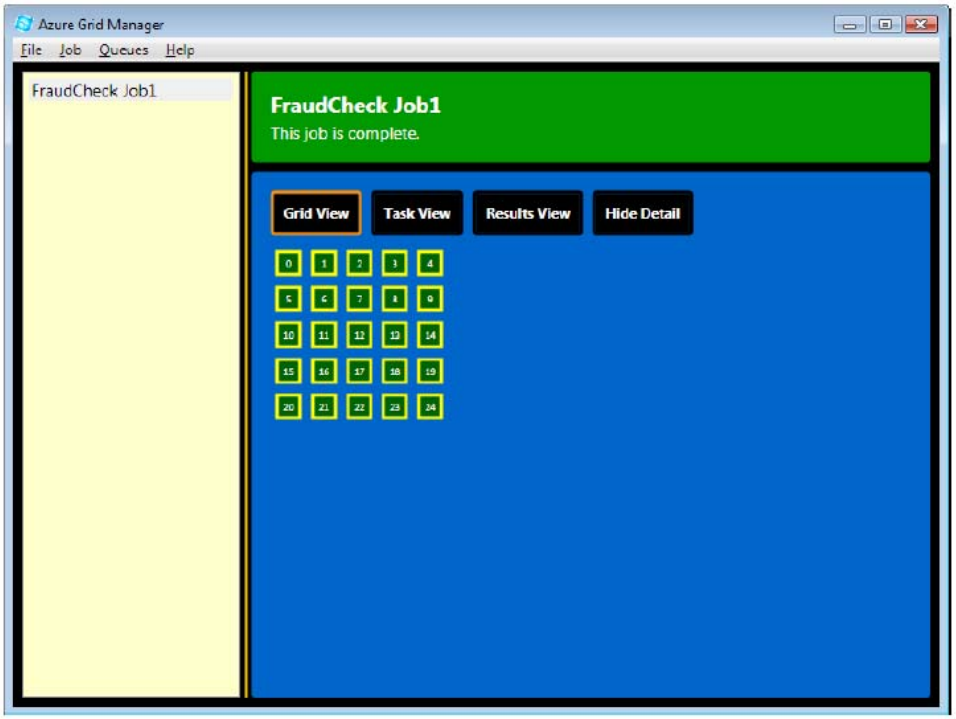
现在，我们已经准备好启动网格应用程序了。点击 Start 按钮，很快你就会看到工作面板变为黄色，任务的网格也显示出来。



上述显示结果将每 10 秒钟更新一次。不久，你将看到一些任务显示为红色，一些黄色，而一些是绿色。任务在等待的时候显示红色，被执行的时候显示黄 色，当执行完成就显示绿色。



当所有任务都完成后，工作面板就显示为绿色。



在网格应用程序执行完成后，我们来检查一下结果。Fraud Check 被编写为从输入电子数据

表中读取输入参数，再把结果写入到输出电子数据表中。打开这个新创建的输出电子数据表，我们能看到网格应用程序为每条输 入记录创建了一个评分、说明文本和批准决定。

	A	B	C	D	E
1	Last Name	First Name	Score	Accepted	Notes
2	Strauss	Johann	1000	1	
3	Chopin	Fryderyk	700	1	Employed less than 1 year.
4	Haydn	Joseph	700	1	Employed less than 1 year.
5	Brahms	Johannes	700	1	Employed less than 1 year.
6	Pachelbel	Johann	850	1	Employed less than 2 years.
7	Vivaldi	Antonio	700	1	Employed less than 1 year.
8	Grieg	Edvard	850	1	Employed less than 2 years.
9	Mendelssohn	Felix	850	1	Employed less than 2 years.
10	Smetana	Bedrich	850	1	Employed less than 2 years.
11	Mahler	Gustav	700	1	Employed less than 1 year.
12	Bizet	Georges	700	1	Employed less than 1 year.
13	Liszt	Franz	1000	1	
14	Schubert	Franz	850	1	Employed less than 2 years.
15	Beethoven	Ludwig	0	0	Age out of range, automatic rejection. SSN suspect.
16	Bach	J.S.	700	1	Employed less than 1 year.
17	Mozart	Wolfgang	0	0	Country not in covered area, automatic rejection. SSN missing. Employed less than 1 year.
18	Tchaikovsky	Pyotr	0	0	Age out of range, automatic rejection. Employed less than 1 year.
19	Gershwin	George	850	1	Employed less than 2 years.
20	Sibelius	Jean	700	1	Employed less than 1 year.
21	Dvorak	Antonin	700	1	Employed less than 1 year.
22	Handel	George	700	1	Employed less than 1 year.
23	Debussy	Claude	1000	1	
24	Puccini	Giacomo	850	1	Employed less than 2 years.
25	Rachmaninoff	Segei	1000	1	
26	Verdi	Giuseppe	0	0	Age out of range, automatic rejection. Employed less than 1 year.
27					

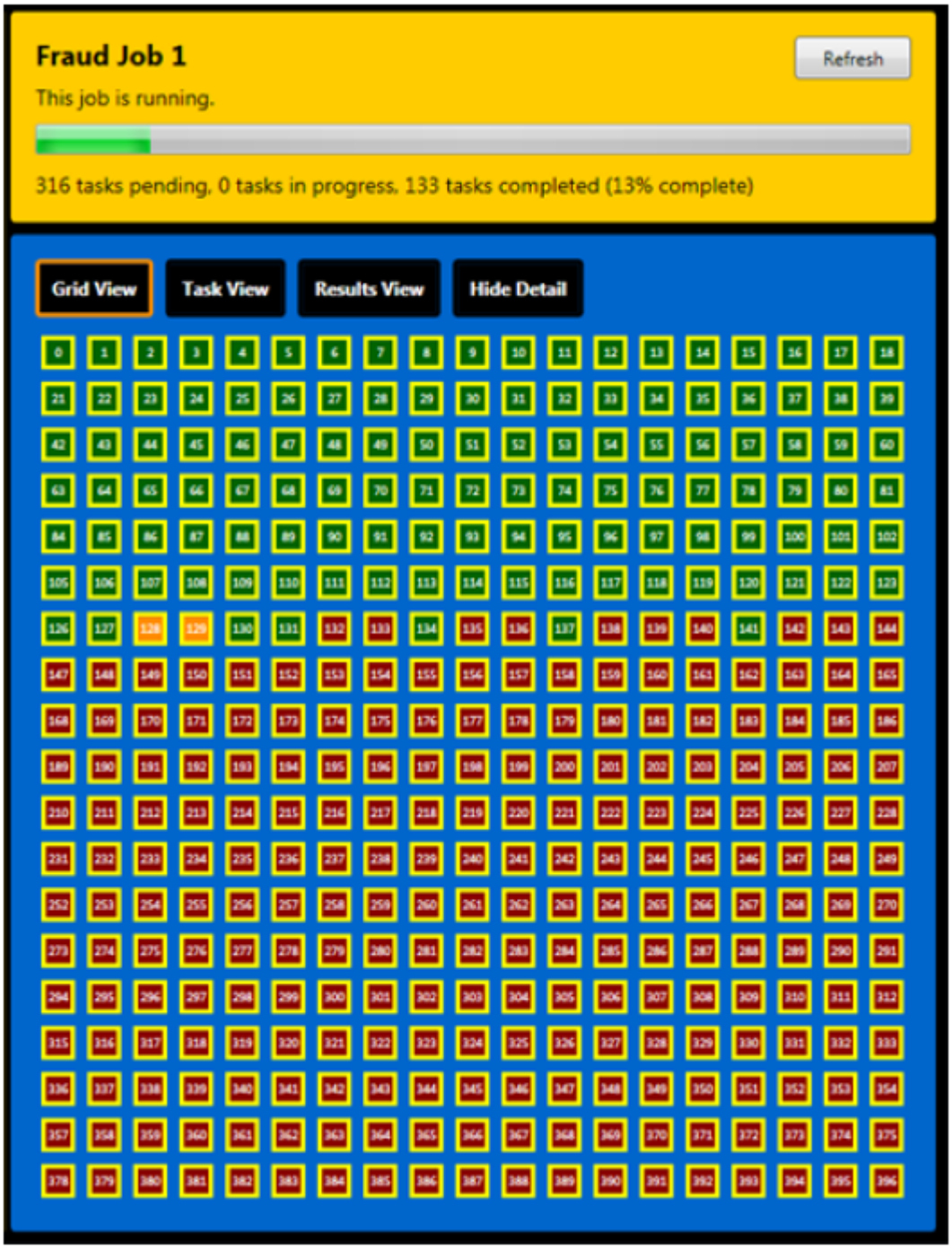
查看网格

在这个我们刚刚完成的测试运行中，我们看到一个 Grid View，藉此我们能了解到网格应用程序执行的总体情况。每个小框代表一个网格任务，框中显示任务的 ID。在下面的图中，你能看到一个显示了大量任务的 Grid View。

红色的任务是等待中的，意味着它们还没有被执行。这些任务保存在云存储上的任务队列中（加载器把任务放到这个队列中）。也就是说，这些任 务还未被网格执行器来提取。

黄色的任务是执行中的。你通常会看到和执行器角色实例数目一样的黄色小框。

绿色的任务是执行完的。这些任务已经被执行，它们的结果已经被放到聚合器队列上，以等待聚合器提取。

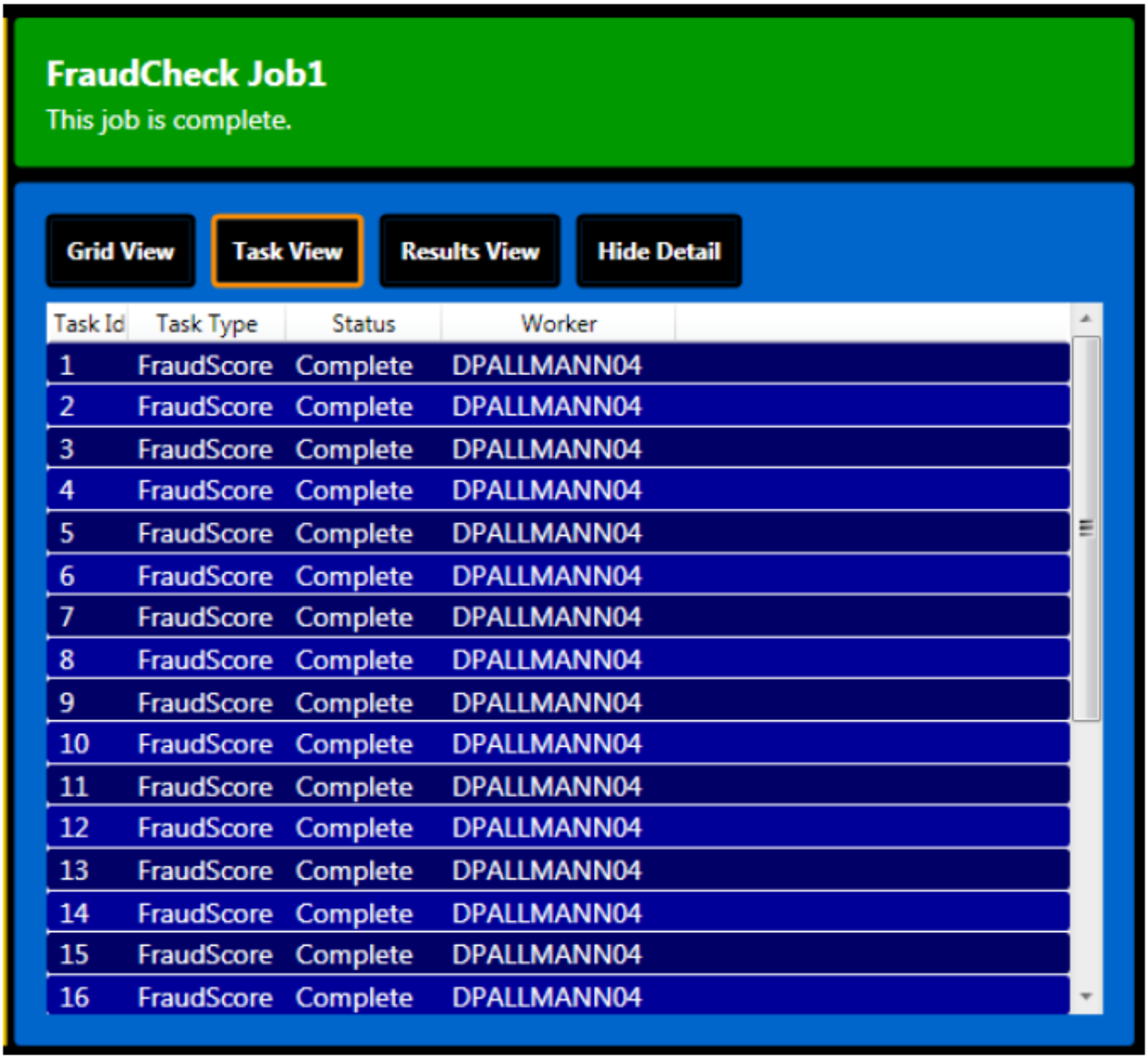


除了 Grid View 外，你还能看到任务和结果。Grid View、Task View 和 Results View 按钮让你可以看到网格应用程序的不同视图。

查看任务

在工作运行之中或之后，你可以点击 Task View 按钮来查看任务的相关信息。Task View 为每个任务显示一行数据，包括 Task Id、Task Type 和 Worker 机器名称。在本地运行的时候，列出的总是你自己的本地机器名称，而在云中运行的时候，你会看到负责处理每个任务的特定

机器名称。



查看结果

Result View 类似于 Task View——每个任务一行数据——不过以 XML 的形式为每个任务显示了输入参数和结果。你也可以拉宽这个窗体来查看完整的信息。

FraudCheck Job1
This job is complete.

Grid View | Task View | Results View | Hide Detail

Task Id	Task Type	Status	Parameters	
1	FraudScore	Complete	<Parameters> <Parameter name="LastName" value="Beethoven" /> <Parameter name="FirstName" value="Ludwig" /> <Parameter name="State" value="CA" /> <Parameter name="Country" value="USA" /> <Parameter name="Age" value="112" /> <Parameter name="SSN" value="111-11-1111" /> <Parameter name="Relation" value="Applicant" /> <Parameter name="MonthsEmployed" value="48" /> </Parameters>	<Results> <Result name="Score" value="0" /> <Result name="Notes" value="Age out of ranc <Result name="Accepted" value="0" /> </Results>
2	FraudScore	Complete	<Parameters> <Parameter name="LastName" value="Bach" /> <Parameter name="FirstName" value="J.S." /> <Parameter name="State" value="NY" /> <Parameter name="Country" value="USA" /> <Parameter name="Age" value="27" /> <Parameter name="SSN" value="156-77-8935" /> <Parameter name="Relation" value="Applicant" /> <Parameter name="MonthsEmployed" value="0" /> </Parameters>	<Results> <Result name="Score" value="700" /> <Result name="Notes" value="Employed less t <Result name="Accepted" value="1" /> </Results>
3	FraudScore	Complete	<Parameters> <Parameter name="LastName" value="Mozart" /> <Parameter name="FirstName" value="Wolfgang" /> <Parameter name="State" value="Tirol" /> <Parameter name="Country" value="Austria" /> <Parameter name="Age" value="23" /> <Parameter name="SSN" value="" /> <Parameter name="Relation" value="Co-Applicant" /> <Parameter name="MonthsEmployed" value="3" /> </Parameters>	<Results> <Result name="Score" value="0" /> <Result name="Notes" value="Country not in <Result name="Accepted" value="0" /> </Results>
4	FraudScore	Complete	<Parameters> <Parameter name="LastName" value="Strauss" /> <Parameter name="FirstName" value="Johann" /> <Parameter name="State" value="GA" /> <Parameter name="Country" value="USA" /> <Parameter name="Age" value="48" /> <Parameter name="SSN" value="095-65-7844" /> <Parameter name="Relation" value="Co-Applicant" /> <Parameter name="MonthsEmployed" value="150" /> </Parameters>	<Results> <Result name="Score" value="1000" /> <Result name="Notes" value="" /> <Result name="Accepted" value="1" /> </Results>

在云中运行网格应用程序

在看了本地机器运行的网格应用程序之后，现在让我们来在云中运行它。这要求我们修改一下配置信息以利用云存储项目而非本地开发存储；也要把 AzureGrid 项目部署到云托管项目中。

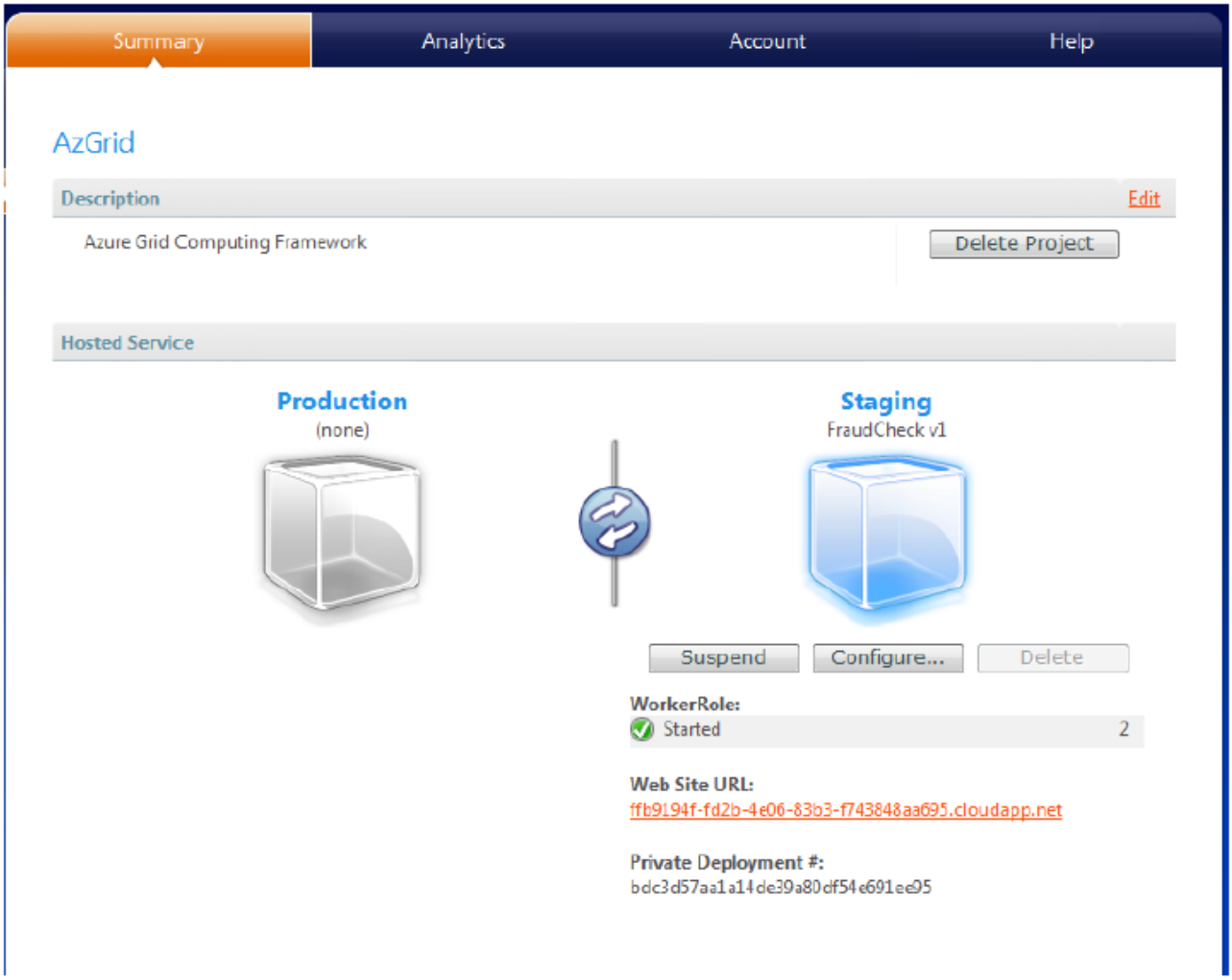
部署到云中的步骤和任何托管 Azure 应用程序类似：

1，右键点击 AzureGrid 项目，选择 Publish。

2，在 Azure 门户上，访问托管 Azure 项目，并点击 Staging 下面的 Deploy 按钮，来上传你的

应用程序包和配置文件。

3，点击 Run 来运行你的实例。

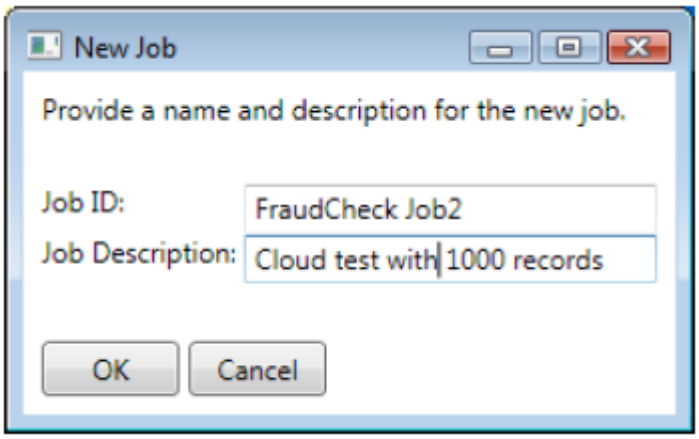


下一步就是准备你的输入数据。在我们的本地测试环境中，输入到应用程序的数据是一个包含 25 行待处理的申请信息的电子数据表。这次，我们将使用一个 具有 1000 行的大数据。

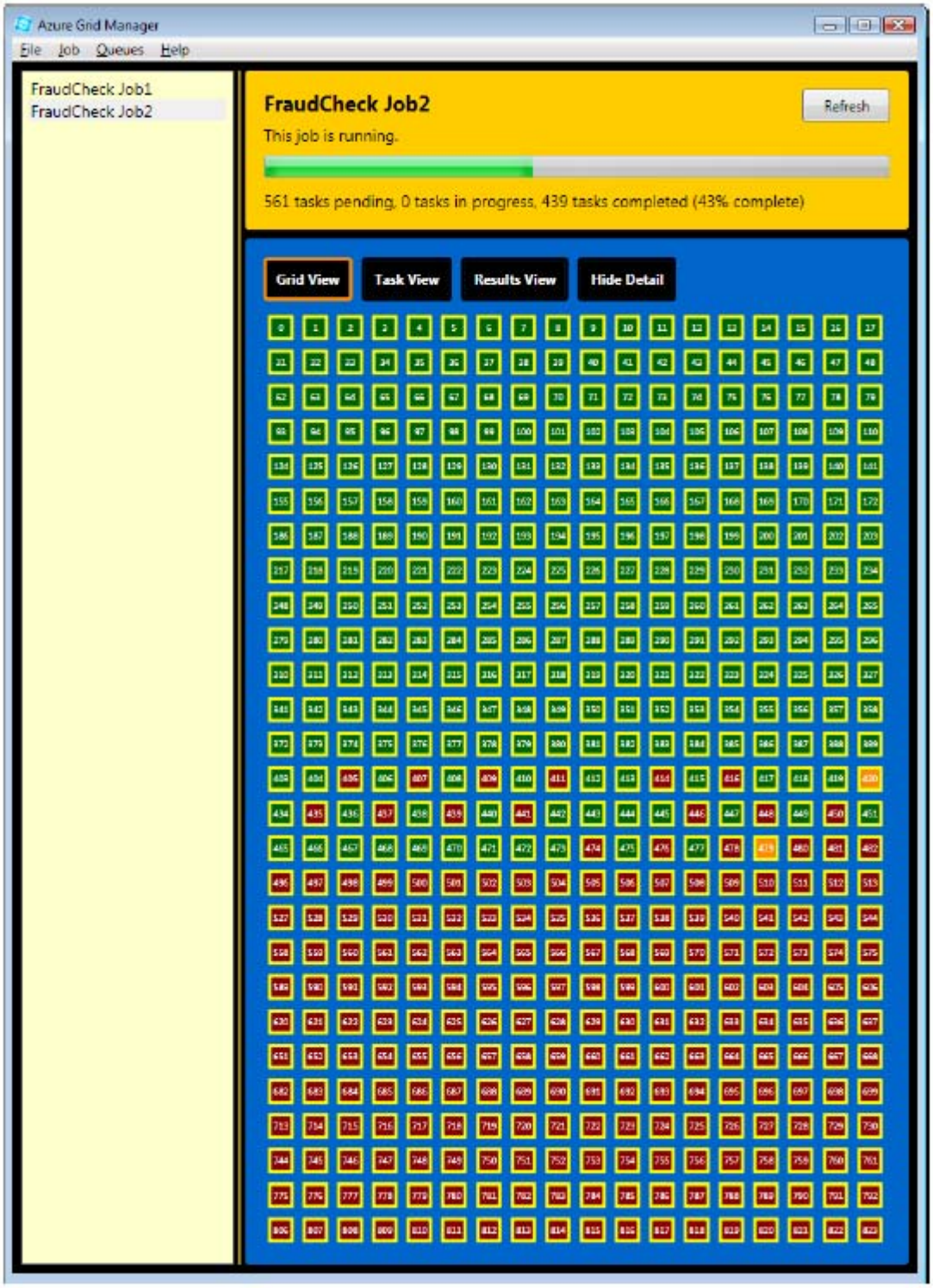
	A	B	C	D	E	F	G	H
1	LastName	FirstName	State	Country	Age	SSN	Relation	MonthsEmployed
2	Davis	Bette	CA	USA	112	111-11-1111	Applicant	48
3	Smith	John	NY	USA	27	156-77-8935	Applicant	0
4	Strauss	Hans	Tirol	Austria	27		Co-Applicant	3
5	Brennan	Michael	GA	USA	48	095-65-7844	Co-Applicant	150
999	Carruthers	Rebecca	MO	USA	64	156-77-8935	Applicant	6
1000	Carruthers	Daniel	Okinawa	Japan	74	156-77-8935	Applicant	5
1001	Carruthers	Debe	Okinawa	Japan	84	156-77-8935	Co-Applicant	4

由于云中的那部分程序已经在 Azure 平台里运行起来了，所以我们只需要在本地启动 Grid Manager 应用程序。

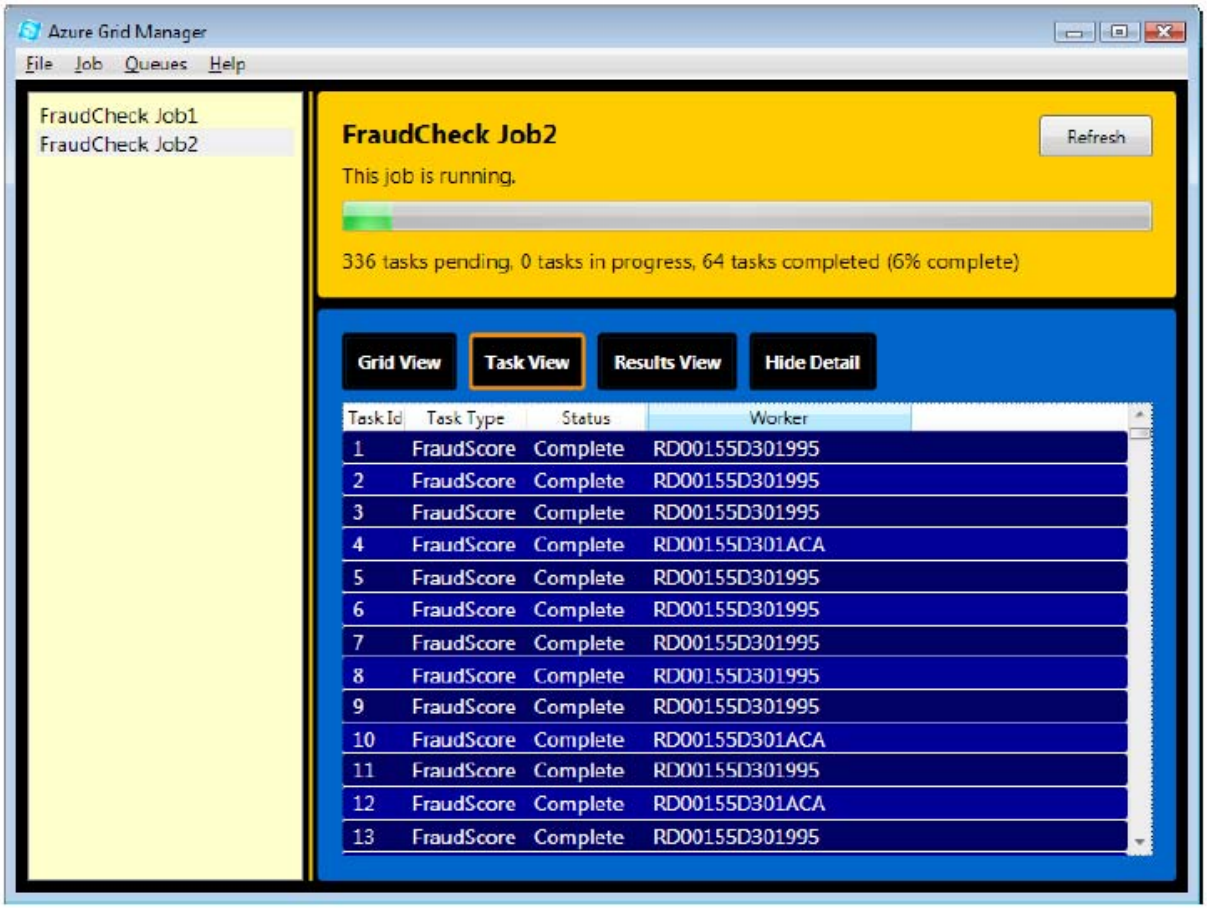
我们再次通过 Azure Manager 菜单的 New Job 菜单项，选择一个工作来启动。



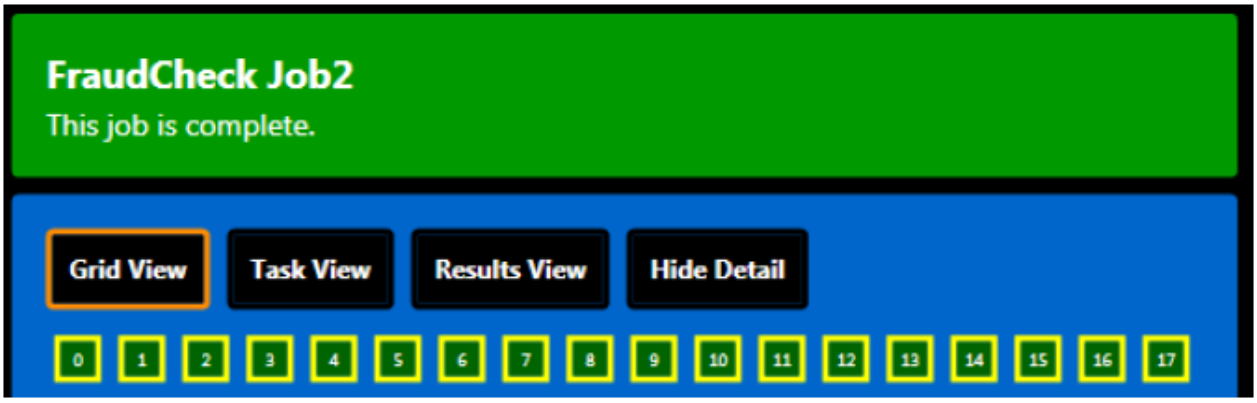
和前面一样，我们点击 OK 完成对话框的录入，接着点击 Start 来开始工作的运行。你可以在 Grid View 中看到加载器生成了 1000 个任务。



切换到 Task View，你能看到执行每个任务的云执行器的机器名称。



即使在网格应用程序完成计算之前，你就可以通过 Results View 开始检查结果了（之前所显示的那样）。



在工作完成之后，云存储已经被清空。由于已经没有工作可做，我们可以暂停部署到云中的应用程序，这样就避免了额外的计算花费。

一旦应用程序完成运行，期望的结果（在 Fraud Check 这个例子中的输出结果是一个 CSV 文件）也就准备妥当，行的数目正如我们所预料的（和输入数据的数目一致），每个任务都有了计算结果。

	A	B	C	D	E
1	Last Name	First Name	Score	Accepted	Notes
2	Brennan	Michael	1000	1	
3	Jansen	Bernhardt	700	1	Employed less than 1 year.
4	Cooper	James	700	1	Employed less than 1 year
999	Pallmann	Frank	850	1	Employed less than 2 years.
1,000	Pallmann	David	850	1	Employed less than 2 years.
1,001	Pallmann	Susan	850	1	Employed less than 2 years.

我们能看到 1000 行数据已经被处理了，不过记录没有保持最初的顺序。这是很正常的：我们无法控制任务执行的顺序，而且我们所关心的只是每个任务保 证其的原子性。把输入参数复制到输出结果中的一个原因是，可以把输入信息和结果信息关联起来。

性能和调优

当前，很难估计在Azure中的网格计算应用程序的性能，由于我们还只能使用Azure的预览版【译者注：本文翻译的时候Azure已经正式商用， 不过尚未在中国大陆推出，详情见此处。】，它提供给开 发人员用于测试的实例数量只是每个项目 2 个。

不过有一种让你的网格计算应用程序的实例超过 2 个的方法，就是把网格执行器托管到多个地方。你能使用多个托管账户，可以和其他 Azure 账户拥有者 进行合作。你也能在本地运行一些执行器。这样做可行，是因为云存储中的队列是网格执行器和企业加载器/聚合器的通信机制。只要你使用一个通用的存储项目， 你就能把网格执行放置到各种各样的地方。

我们也希望网格计算应用程序业已成真的地方，能在 Azure 中也能成真：例如，计算密集的应用程序能通过网格计算这种方式能获得巨大的回报。

随着 Azure 正式发布的临近，我们应该能收集到很多关于网格计算性能和如何调优的大量有意义数据。

原文链接：http://www.infoq.com/cn/articles/Grid-Azure-3-David-Pallmann

相关内容：

- 基于Azure云计算平台的网格计算，第 2 部分：开发网格应用
- 基于Azure云计算平台的网格计算，第 1 部分
- OOPSLA辩论：被高估的云计算
- Joyent：别样的云计算平台
- 基于云的跨企业信息系统会取代企业外部网吗？

推荐文章 | Articles

在不改变语言的前提下如何推进Java的不断演进

作者Charles Humble 译者 张龙

James Gosling在“The Feel of Java”中说到：

> Java 是一种蓝领语言，它并非博士的论文素材而是用于完成工作的语言。各式各样的程序员都非常熟悉 Java，因为在设计 Java 之初我就坚持这样一种观点：选择久经考验的东西而非仅仅是听起来很美。

Java 所获得的巨大成功证明了这种设计方式是正确的，但如果这依然是当今 Java 的重要目标的话，那么其结果就是语言的演进将变得非常缓慢。除此 以 外，Java 是一门成熟、使用广泛的语言这个事实也将导致其演进过程充满了困难。一方面，添加到语言中的每个特性都可能在一定程度上造成不可预料的结果， 这么做会疏远那些已经使用了该语言的开发者。另一方面，本身很完美的特性可能在同语言中的其他特性进行交互时产生不可预料的结果。更糟的是，一旦增加了某 个语言特性，几乎就不可能再将其移除了，即便是该特性会对整个语言产生不利影响也没办法。为了证明某个新特性是正确的，语言设计者必须确信从长远来看，该 特性会给语言带来好处，而不是短期效益或是针对某个问题的快速解决方案，之后就变得可有可无了。为了降低风险，语言设计者通常都会创建单独的一种语言或是 分支来进行试验，比如 Pizza 语言就是在实现前用来测试 Java 泛型的。这种方式的问题在于试验的参与者非常小众并且都是自己想参与进来的；显然他们对 语言特性很感兴趣，很多人都是学者或研究员。但是，在普通的程序员开始使用这些特性时，那些学者或是研究员认为很棒的特性可能会变得很糟。

为了直观感受一下这种情况，请考虑关于Java 7 闭包特性的激烈争论。一段时间以来，有人在提案中给出了闭包的实现，但最终却还是没有达成共识。随后，Sun决定不打算在JDK 7 中

添加完整的闭包支持。这时争论的焦点转向为Java是否变得越来越复杂了，在Java 5 中添加泛型（尤其是通配符语法）时就已经出现了这种争论；在Java已经通过匿名内部类部分实现该功能的情况下，完整的闭包支持是否是正确的呢。需要完 整闭包支持的两个重要场景是简化fork/join API(添加到了JDK 7 中以改进多核编程)的使用以及辅助资源的清理。Josh Bloch的ARM block提案（期望通过Project Coin加入到JDK 7 中）就第二个问题给出了另一种解决方案。Cliff Click博士在面向Java的可扩展、非阻塞编程风格的研究中给出了关于fork/join的另一种方案，随着处理器核心数的不断增加，这种方案看起来更合理。如果这一切都成为可能的话，那么Java中使用闭包的地方将变得非常少了，语言根本没必要提供这个特性。

话虽如此，但对于编程语言来说，持续不断地平稳发展还是非常重要的。因此本文探究了如下 3 种技术以向 Java 中增加新的语言特性而又不改变语言本 身，他们 是客户化领域特定语言（DSL）、Java 6 的注解处理器（通过库来增加可选的语言特性）以及将语法糖从语言迁移到 IDE 中。每项技术都可以让众多的主流开发者以非侵入的方式体验这些新特性，最棒 的想法则可以融入到语言核心当中。

客户化DSL

在这 3 项技术中，人们谈论最多的还是DSL。该术语的确切含义至今尚未统一，但出于讨论的目的，我们在这里简单地 把它看作是用于解决特定问题、应用范围很窄 的一种语言而非用于解决所有计算问题的通用语言。这样，DSL就并非是图灵完备（non-Turing complete）的。当然了，还是会有一些边际情况存在的，比如说Postscript是一种图灵完备的语言， 但根据我们方才的定义，它也是一种DSL。

如上所述，DSL并非新概念。其他类似的DSL还有正则表达式、XSLT、Ant以及JSP等等，所有这些都需要某种客户化的解析器对其进行处理。Martin Fowler还说fluent interfaces/API也可以看作是另一种DSL，称之为内部DSL。他说内部DSL是直接在宿主语言中开发出来的。这对于Lisp和 Smalltalk开发者来说很容易理解，而最近Ruby社区也开始对内部DSL情有独钟了。

虽然很多知名的DSL都是由商业公司开发和维护的，但一些企业开发团队也已经使用该技术来创建能够快速解决其问题的语言了，但毕竟还是小众，这可能 是 DSL领域门槛比较高的缘故吧。负责DSL的团队必须要设计语言、构建解析器和其他工具来支持开发团队，还要对每个新员工进行培训，让其了解DSL的工作 机理。这时，涌现出了能够支援DSL开发的工具，这极大地改变了当前的状况。Intentional Software所开发的Intentional Domain Workbench比Java还要久远，它首度实现了该工具的功能。该项目创建于微软研究院，Charles Simonyi博士在 1995 年所发表的论文“The Death of Computer Languages, the Birth of Intentional

Programming”描绘了其愿景。2002 年，Simonyi创建了Intentional Software以继续实现其想法，大家可以看看介绍该系统的视频，极具震撼力。目前该产品的版本是 1.0，但只有极少数的合作者能够访问。

其他一些软件公司也在研究这个概念，其中就包括以IntelliJ IDEA Java IDE而扬名天下的JetBrains，JetBrains最近发布了Meta Programming System（MPS）1.0 版。MPS并没有使用解析器而是直接使用了Abstract Syntax Tree（AST）。它提供了一个类似于文本的投影编辑器（projectional editor）以便程序员能够操纵AST，同时该编辑器也可用于编写语言和程序。当程序员使用投影时就会为树上的每个节点创建一个文本投影，这样变换就会 反映到节点当中。开发者可以通过这种方式以任意组合（通常称之为语言组合）扩展和嵌入语言。JetBrains正在内部使用该产品，最近发布的bug追踪 产品YouTrack就是使用该系统开发的。

Java 6 注解处理器

相对于Ruby、Smalltalk和Lisp来说，DSL在很多主流语言（如Java）中的流行程度就稍逊一筹了，但最近Java语言的一些变化 （尤其 是Java 6 中新增的注解处理器）为开发者提供了新的机遇以在其中使用DSL。对于Java EE 6 中的JPA 2.0来说，其某些API本身就是 DSL。注解处理器会为应用中的每个持久化类建立一个元模型类型（metamodel type）。虽然开发者可以手工处理Java中的元模型，但这实在太无聊而且极易出错。注解处理器的出现改变了这一切，因为它内建于Java 6，因此无需特殊的IDE支持——IDE会代理编译器所触发的注解处理器，之后会自动生成元数据模型。

程序库也可以通过注解处理器来提供新的语言特性。比如说，Bruce Chapman的原型“no closures”提案就凭借该技术将方法转换为 Single Abstract Method（SAM）类型，然后在Java 6上编译。在与其交谈过程中，Chapman指出SAM类型还支持自由变量（free variable），这是闭包的一个关键技术：

> “除了 Single Abstract Method 所需的参数外，方法体还可以通过@As.Additional 注解声明额外的参数。在获得 SAM 类型的实例时，这些参数可以带有绑定值，然后在每次调用时传递给方法。

Chapman还创建了Rapt项目以探索该技术的其他使用场景，同时为语言的两个变化提供了自己的实现——多行字符串（Multiline Strings）与XML字面值（XML literals）——这两个特性是为JDK 7 准备的，但却不会包含到最终的发布中。Java甚至也可以使用这种方法实现闭包，Chapman对此说到：

> 我们刚刚使用该技术完成了一个 Swing 项目，在这个过程中发现了泛型的一些小 bug，其中一个 bug 还没有修复，除此之外一切 都很棒，没人再想使用传统的匿名内部类了。

另一个探究注解处理器的项目是Lombok，它将该技术又向前推进了一大步。Lombok将注解作为回调 以运行Java agent，后者会根据注解重写各种javac内核。由于操纵的是内部类，因此它不太适合于产品使用（JVM各个小版本中的内部类也可能不一样），但该项 目对于注解处理器到底能做什么这个问题上还是颇具启发意义的，包括：

- 通过@Getter 和@Setter 注解定义各种访问级别的属性，如 @Setter(AccessLevel.PROTECTED) private String name;
- @EqualsAndHashCode 注解会根据对象中的属性实现 hashCode()和 equals()方法
- @ToString 注解会实现 toString()方法
- @data 方法相当于 @ToString、@EqualsAndHashCode、所有属性的@Getter 以及所有非 final 属性的@Setter 的集合，可以使用 @data 方法和构造方法初始化 final 属性

还可以通过这种方式进行其他的语言试验，比如移除 Java 中的非运行时异常等。

虽然注解处理器技术为语言试验开辟了一条新航线，但还是要注意生成代码的可读性，保证开发者能读懂生成的代码。Chapman 给出了很多建议：

> 要生成源代码而不是字节码，注意生成代码的格式（尤其是缩进）。编译器不在乎生成的代码是不是都在同一行，但用户在乎。我甚至还 使用注解处理器在恰当的地方增加了一些注释和 javadoc。

如果这项技术逐渐流行起来，用户将可以通过 IDE 查看编译期所生成的代码。

IDE中的语法糖

Bruce Chapman还提到了第三项技术——将语法糖从语言迁移到IDE中——他在博客中对该问题做过深入阐述。对于Java IDE来说，生成部分样板代码已成为不可或缺的功能了，比如类的getters和setters，但IDE开发者刚刚开始深挖该这个概念。 JetBrains的IntelliJ 9为内部类提供了一个类似于闭包的简洁的代码块语法，开 发者也可以自己加。就像代码折叠一样，该语法会扩展到编译器能够处理的完整的匿名内部类——这样坚持使用标准的匿名内部类语法的开发者很容易就适应了。 Eclipse也有一个类似的插件。关键在于这种语法仅仅是实际代码的另一种展示方式而已，编译器和任何源代码管理工具都能够像以前那样处 理他们。

开发者可以在两种方式间自由切换（就像代码的展开与收起一样），无权访问该语法糖定义的人仅仅会看到正常的Java代码。Chapman说到：

> 要想实现易于使用的目标还有很多工作要做，但从长远来看，我发现开发者会很轻松地实现加糖/脱糖（sugaring /desugaring）的转换（可以参考jackpot来了解实现原理）、不断尝试、不断演进并与同事和社区分享好的点子。这么做的好处与语言的演进别无二致。最好的东西会流行起来并形成实际语言演进的基础，如果必要的话还可以随时去除该方法无法实现的“噪音”。

由于语法糖还要兼顾（非常多）其他的语言特性，因此无法提供完整的闭包支持；比如说BGGA闭包就有一些特性无法匹配匿名内部类，因此不能通过这种方式实现。话虽如此，这种想 法却展示了通过各种新语法来表示匿名内部类的可行性，类似于BGGA语法或是FCM语法，开发者也可以选 择自己喜欢的语法。其他的语言特性（如null-safe Elvis operator）可以通过这种方式实现。要想进一步验证该想法，可以体验一下这个NetBeans module（由 Chapman开发），这正是他所说的用于 Properties的原型。

结论

在语言的发展过程中总是需要考虑稳定与发展之间的平衡。上面介绍的技术所带来的好处是他们不会影响平台或是语言本身。这么做允许我们犯错误，也有益 于进行 快速激进的试验。由于开发者能够自由地进行试验，我们看到越来越多的人开始解决常见的样板代码“噪音”问题，如匿名内部类语法等，同时将这些想法以合理的 方式组织起来以获得最大的价值。我们将欣喜地看到开发者使用这些方法将 Java 平台推向新的远方。

原文链接：http://www.infoq.com/cn/articles/evolving-java-no-lang-change

相关内容：

- 类加载器特技：OSGi代码生成
- 模块化Java：声明式模块化
- 模块化Java：动态模块化
- 使用jBpm支持高级用户交互模式

推荐文章 | Articles

面向资源的架构：REST的另一面

作者 Brian Sletten 译者 黄璜

系列前言

作一个假设，假如把你送回万维网出现以前。想象一下你如何去跟当时那些毫无准备的人们解释那即将发生的重大变化。结果很可能是他们根本无法深入地理 解万维网的出现将会对他们生活的每个方面产生的影响。在回顾中，那种感觉就像一场突如其来的海啸，永远地改变了我们身边的景象。而现实可就朴实得多了，我 们现在所经历的这些结果都是由一系列深思熟虑的技术选择及它们之间的相互作用所造成的。

而现在，稍作停顿，想一想把你自己放在那些曾经对 Web 保有怀疑态度的人们所处的位置，正要被别人开化。除非你保持足够的警觉，不然你又会再一次毫 无准备地面对新一轮经济上、社会化、技术化、以及组织形式上的冲击！尽管这种改变令人感觉发生在一夜之间，实际的情况是它们都经历了经年的积蓄，而现在才 产生了有形的结果。这种新的浪潮就是 Web 的发展已经逐渐超越了文档，而成为了数据的网络，包括个人的和私有的。我们不再关注信息的载体，而转向关注信息 本身以及它们如何联系。

这一浪潮同样历经了多年的努力，并且同样是由认真思索后所选择的特定技术来驱动的。通过这些选择的组合，可以解决因不断发生的技术变化、业务膨胀、 新的及变更的数据源、以及那些普遍存在的既昂贵而又易出故障的花了大价钱却又不能交付相应价值的项目所带来的问题。Web 服务与面向服务的架构原本被看做 是解决问题的一个答案，然后他们那优雅的愿景却永久地因为拙劣的技术解决方案而染上了污点。

可喜的是我们并非是从零开始。我们是建立在原有的数据 Web 的技术的发展之上的。我们可以将数据库、软件包、服务以及其它的内容源打包成新的抽象集 合，从而帮助我们摆脱过去那种单调乏味的工作。我们将公共的数据 Web 与我们自己的私有数据集成起来。不断渐进地采用这些技术将会收获新的能力，而这些能 力又能释放出更多的能力。

本文是这系列文章中的第一篇，该系列突出了面向信息系统的演化、告诉我们位于什么位置，并给出了我们发展的路线图。不管从表面上看是怎样，这些选择 不是随机或神秘地做出的决定，而是基于学术与应用工程的长期传统而作出的根本抉择。

我们重新回到表象化状态转移(REST)架构风格。它经常被引用，可更多的是被误解，这一构建网络软件系统的方式允许我们将文档，数据和面向信息的 服务融合成一个丰富的具名资源的逻辑性生态系统。从这里开始，我们将引入语义网的愿景，并领略由灵活与扩展兼备的数据模型和查询能力所代表的语义网的核心 技术。我们将看到如何将关系型数据、来自文档的内容、电子表格、RSS feed 等等包含进一个由可重用内容组织的富 Web。

在展示了这些基础之后，我们将会去领略各种构建于这些技术之上的成功项目，并回来重新恢复 Web 服务技术所支持的曾对我们许诺过愿景。我们将会描述 一个流程，我们可以以此达成一个信息系统统一理论；它不仅能处理，并且将拥抱那些过去痛苦而又棘手的技术性和社会性变更。

关于语义网的炒作已经太多了，但同时也有一股稳定的分支在默默的通向成功。这一系列文章对新的以及已熟知的领域都会作一个指引。但我们将会从你未曾 看到过的更深的层次将这些技术联接起来。我们将会强调企业，政府组织以及标准实体所发生的事件和行动，以表明我们是正处于这样的进程中，并且这将改变一 切。我们将展现通过微妙的视角转换和施行促进变更的标准将会对你的系统实现带来多么大的变化。

第一步，是为基础设施的所有方面采用公共的命名模式。一个只关注服务的架构往往忽略了流经系统的数据流。到头来，我们的组织第一个关注的是信息。 REST 和 Web 架构将这一优先级摆在了前面，并为我们后面的讨论做好了铺垫。

Rest的其他方面

在与复杂性的斗争中，人们讨论表象化状态转移(REST)已成为了一种时尚。这场战斗的敌人，据某些人的观点，是 SOAP 以及围绕它的 Web 服务技 术栈。这种敌我的修辞给论点带来了激情，但却很难产出有意义的对话，所以人们对于其蕴含的消息以及其重要性仍然感到困惑。REST 的目标不是代替 SOAP；而是构建更好的系统。

REST 甚至不是 SOAP 的直接替代。它并非什么方便的技术，或者所谓的通过 URL 调用 Web 服务的简单解决方案。对于信息资源的管理与调用随意性 的行为本就不是同一回事，这种混淆使得人们构建出的所谓"RESTful"解决方案，既非 RESTful，也不是好的解决方案。

REST的益处既得自于它的限制又得自于它的灵活性。如果你读过Roy Fielding博士的论文（你

值得一读），你会明白其意图是描述特定的架构性约束是如何组合起来而产生了一系列我们在网络软件系统里所期待的属性。采用统一的接口，赫赫有名的统一资源定位器（URL） 对REST的定义有贡献，但仍不足以定义它。同样的，仅仅是通过URL来暴露随意性的服务接口，也无法产出我们所看到的暴露Web同样成功的益处。它要花 费更丰富的一系列交互与系统划分才能得到完整的结果。

大多数人的理解是，REST 牵涉到使用几个动词通过 URL 来请求和提供信息资源的应用信息。你对 URL 发起 GET 请求来抽取信息，你通过 POST 和 PUT 来创建和更新，而 DELETE 请求用于移除信息。

这一总结并没有错，但它只是其中很小的一部分。不幸地是，它省略掉的部分所产生的自由度往往使得人们作出错误的决定。在这种鸿沟中，人们脱离动词来 创建URL，这就消除了给事物命名所带来的好处。 他们认为REST仅仅关心CRUD操作。他们会创建一些神奇的不相关的URL，你必须事先知道如何解析，因此丧失了超文本引擎的可发现能力。而最不可原谅的，就是创建的这些URL 单独与特定的数据格式关联起来，过早地对客户端的信息状况作了决定。

理解 REST 整体的含义有助于你避免这些问题；它能帮助你开发强劲的、灵活的、可伸缩的系统。但这同时也是对于信息以及它是如何被使用的一种全新理 解的一个起点。在这样的 Web 架构基础之上，应用语义网的其它技术就会产生出前所未有的力量，而对我们作为个人，作为政府，组织等等如何相互进行交互产生 影响。这也解释了我们为什么一来就深入到 REST 的那些不怎么为人所知和被人谈及的部分。这些部分的含义包括以下主题：

- URL 作为标识符
- 自由的形式
- 逻辑连接的延迟绑定系统
- 超文本作为状态转移的引擎（HATEOS）

URL作为标识符

我们已经建立了这样一种前提，大多数人都知道URL和REST。比较明显的一点是人们都知道把URL用作调用服务，但人们对于把URL作为信息的命 名这种宏观的概念还不是很熟知。名字是我们用来标识人员、地点、物体和概念的。如果我们没有这种标识的能力，我们就没有表示的能力。每天想想Abbott 和Costello的臭名昭著的诙谐剧“誰在一壘”。有了命名我们 就

可以消除歧义，并在一定的上下文范围内明确地标识我们所关心的东西。有了命名和公共的上下文我们就能在这一范围内明确的对命名的事物进行参照。

统一资源标识符（URI）是父级模式。它是用于编码其它模式的一种方法，取决于你是否想要包含进它们的解析信息。图书管理员和其它的长久数据管理员都喜爱不会改变的名字。统一资源名称（URN）是不包含定位信息的 URI；它只包含名字。好处是这些名字不会失效。而坏处是这些名字没有解析过程。一个URN的例子是书籍的ISBN号：

urn:isbn:0307346617

为了查找关于这本书的更多信息，你需要找到一个可以根据 ISBN 号来查找信息的服务。

如果我们的系统和信息的上下文不会有改变的话，我们也许会始终把解析信息包含在资源名称里以便解析。但只要是遇到过失效的连接的人都能理解对于一些 重要的事物我们会想要长期有效的名字。回顾我们使用 URL 的历史，我们做了不少错事，比如：

http://someserver.com/cgi-bin/foo/bar.pl

http://someserver.com/ActionServlet?blah=blah

http://someserver.com/foo/bar.php

这些 URL 的问题在于，用于产生结果的技术与信息的消费者毫无关系。这样创建 URL 没有任何站得住脚的理由。应该关注的是信息，而不是技术。实现技 术总是在不断更新。如果你摒弃了他们，举个例子，任何一个链接到基于 Perl，Servlet 或 PHP 的 URL 就会失效。接下来的文章中我们会讨论到处理 这一问题的一些基础设施，而现在，我们将只关注对于命名我们的信息资源如何作出明智的选择。

尽管脆弱，但 URL 模式确实为我们提供了在全局的上下文里消除信息引用歧义的能力。

http://company1.com/customer/123456

就能很好的区别于

http://company2.com/customer/123456

而像“123456”这样的非上下文的标识符则不能。

将这一理念根植于更广泛的信息系统框架，你可以将 URL 想象成不特定于任何具体数据库的主键。我们可以在各种不同的数据库，文档，应用中通过 URL 来引用一个条目，并且可以确定我们引用的是同一个事物，因为在整个全局的上下文中它有一个唯一确定的名字。在后续的讨论中我们将使用这一特性来描述 RESTful 系统并将其与其它内容和元数据连接起来。

URL 另一个值得讨论的方面就是它们的统一适用性。我们有着一个公共的命名模式可以去识别：

- 文档（报告，博客，公告）
- 数据（成果，实例信息，元数据）
- 服务（REST!）
- 概念（人员，组织，领域特定术语）

我们不需要创造一种另外的机制来参照每个不同类别的物件。某些特定准则的严谨的应用允许我们模糊这些物件的区别，这就把我们领到了关于 URL 的最后 一点。这些名字不仅对于参照我们所关心的信息是有用的，而接收这些引用的系统也可以轻易的请求它们。URL 中的“L”（定位器）给予了我们就算不了解它的 其它任何情况，也能解析它的能力。对于我们命名的物件，通常我们都可以对其调用同样的基础操作。对于以 URL 的形式所表示的文档，数据，或者是产生该数据 的服务，甚至是一个抽象的，不可通过网络访问的概念，发起一个 GET 请求工作原理从根本上来说是相同的。对于我们被授权操作的这些事物，我们也可以以同样 的方式来创建，修改或者是删除它们。

自由的形式

我们对于 Web 的经验就信息的形态来说仍显被动。当我们点击链接时，我们期望内容以某种特定的形式返回，通常是 HTML。对于许多类型的信息来说这 是很好的，但架构对于更多的会话风格提供了丰富的支持，使得客户端可以用它们更青睐的形式请求信息。

要理解它的作用所在，可以考虑一个公司的销售报表。可以想象得到对于管理层、销售人员、其它的员工、客户和投资者而言这将是衡量公司业务表现的一个 有效的指标。这种报告的名字可以在 URL 中包含进年份和季度：

```
http://company1.com/report/sales/2009/qtr/3
```

我们可以将它与 3 月份的销售报表作个对比：

```
http://company1.com/report/sales/2009/month/3
```

两者都是很好的有逻辑内涵的名字，而且不会因时间而失效。人们可以轻易的在浏览器输入这样一个 URL 然后就能以 HTML 的形式得到他们寻找的信息， 这种愿景非常诱人。这个报表可以被添加到书签，通过电邮发送，被链接到，我们所钟爱的 Web 的方方面面。

题在于，信息锁定在表现形式中（除非我们在本系列稍后引入 GRDDL 和 RDFa 这样的技术后！）。我们习惯于从页面获取内容，但却放弃了而只获取摘 要。随着页面的排列改变了，我们的脚本也就失效了。

如果你曾是公司的一名编程人员，而你想要直接获取信息的话，你可能会想要通过 XML 去请求它。你可以得到原始的结构化数据，你可通过模式来验证。 HTTP 和 REST 让这变得易如反掌，只要服务器知道该如何响应。通过在请求中传递一个“Accept:application/xml”的头部信息，你 可以表明你请求的是（或你的需求是）XML。成功的话，你将会得到一个字节流的 MIME 类型，以表明你的请求被接受了。如果失败，服务器将通过一个 406 错误表示无法帮助你。在这种情况下，你也许会想要联系负责该信息的部门并要求他们加入你所需要的支持；他们既能做到而又不会影响现有的客户。 如果你是业务分析师，你会觉得 XML 过于粗糙而有点棘手，你可能想要返回电子表格，这一格式将非常适宜于融入你现有的工作流，工具和流程。

关键点在于该报表的逻辑名字在它被请求的时候可以轻易的转换成多种不同的格式。而运行能够接受这些各种各样修改后的格式的系统也是同样的容易。客户 不能看到信息是如何被存储的真实情况，他们只需要知道它能工作就行。而这样的自由度竟未被构建 RESTful 的人员加以完全的利用。当他们搭建一个服务并 决定仅仅返回 XML 的时候，他们忽略了 REST 原本可以为组织提供的潜在价值。

许多开发者出于不了解内容协商，或者觉得很难在浏览器里对它进行测试，他们就对不同的格式定义了不同的 URL：

```
http://company1.com/report/sales/2009/qtr/3/report.html
http://company1.com/report/sales/2009/qtr/3/report.xml
http://company1.com/report/sales/2009/qtr/3/report.xls
```

一旦你越过的特定用途的约束，这样的开发者捷径反而会变成一种局限。本质上讲，我们现在有三种信息资源，没有一个可以以其它的形式来展示。这不仅在 全局上下文中创建了身份标识的分支，它同时还过早的向客户端承诺了特定的形式。如果你将一个对 URL 的引用作为工作流或者是编配的一部分，那你就掠夺了上 游客户自由选择数据格式的权益。

有几种不使用浏览器测试正确的 RESTful 服务的方法，例如：

```
curl -H "Accept: application/xml" -O
http://company1.com/report/sales/2009/qtr/3
```

使用流行的 curl 程序，任意一个合理的 HTTP 客户端都会提供类似的功能。

支持可协商数据格式的富生态系统所带到的益处也许并不是立竿见影的，但一旦你投入进去，你就会发现它是迈向一个生命长久而灵活的系统的关键，它有益 于客户，而不是开发者。

逻辑上连接的延迟绑定系统

一旦你为信息资源作出了良好的逻辑性的命名，你就会发现因这样的决定而带来的附加的好处。命名的引用可以安全而高效地作为结果传递回去，而不用返回 实际的数据。这对于大量而敏感的数据有着极大的意义，但同时这也意味着可能出现技术和架构的迁移。

出于指针在 C 和 C++语言的作用同样的原因，URL 作为数据的引用而传递给信息的消息者是一种更为紧凑和高效的方式。诸如金融交易，卫星照片这样的 大数据集可以在工作流中引用而不必要求所有的参与者都去承担处理大容量内容的压力。

何接触到实际数据的编配都必须考虑到将其传递给其它系统可能带来的安全性影响。很快就会发现要提供哪些人在流程的每个步骤被允许做什么的精确信息是 难以维持的。如果一个引用一步一步地被传递，那就该是信息源本身来保障访问权限。有些步骤不要求访问敏感的信息，因此在它们解析引用的时候就可以排除它们 接收信息。

这意味着延迟绑定解析可以在整个请求的上下文中发挥作用。使用某一应用访问一个资源的特定用户也许有出于业务需要查看敏感数据的需求。而同一个人使 用另一应用却不一定有业务上的正当理由来访问同样的数据。一个 RESTful 服务可以检查会话标志并且可以声明式的保证这一访问策略。这一层次上的明确性 是需要的，以此才能防范内部欺骗，它通常是处理敏感内容系统的最大的威胁。这样一个系统的细节应当是特定实现的，并且与命名流程和解析逻辑命名的内容很大 程度上是正交的。

依赖于逻辑关系能够保护客户不受实现变更的影响。当流行的网站从一种技术迁移到另一种技术，它们都能够成功地向用户隐藏这些变更。RESTful 也 是这么做的。这让我们可以自由地将遗留系统打包成逻辑接口并得以保留，直到因业务发展而投入新的实现。当出现这种情况的时候，用户也可以受到庇护而不受影 响。

为了调停技术的变更，RESTful系统允许你使用Postel法则的一个变形：保守的给予，自由的接受。你可以对于你接受和返回的内容保持严格的验证。然而，如果你有一个现有的客户群向你提供某种给定形式的内容，你可以允许其 它的客户提供不同的形式，不同的模式，而不影响现有的客户。与终端保持紧密关联合约的系统不可能提供这样的自由度，这将使他们更脆弱并且会很快瓦解。

超文本作为状态转移的引擎（HATEOS）

当系统遇到信息资源的引用时，很多人认为应当需要某种描述语言来指示什么是的可能或者应该做的。实际的情况是一个经过严密思考后的 RESTful 系 统通常不需要这一概念。这对于 SOAP 开发者而言难以接受，但这是与架构风格的约束相关的。因为我们将信息资源看作是通过统一接口(URL)来操作的物 件，并且将我们的行为局限着一个很小的动词集合里，因此确实没必要对服务进行描述。

如果你在这一点上有所疑惑，很可能是在架构风格的品位上你仍在操作资源与调用随意性行为之间产生了犹豫。REST 动作对应用一个信息资源提供了全套 的操作。当然，你需要知道返回的是什么信息你才能知道如何处理它，但这是 MIME 所关心的问题。尽管重用已知的类型 （application/xml，image/png，等等）更为人所接受，但很多开发者并不明白如果需要的话他们可以创建自己的应用特定的数据类型。

在本系列文章的轨迹中，我们将讨论使用富元数据查找和绑定随意性资源的问题。现在，我们只需要简单的记住 Roy 所强调的“超文本作为状态转移引擎” 的重要性（又被 REST 粉丝简称为“HATEOS”）。这恐怕是论文中引起最多误解的部分，要理解它的全部内涵，我们得重新过一遍 Web 是如何工作的。

你在浏览器中输入 URL，它会向该资源发起一个 HTTP GET 请求。不变的是，服务器将响应一个字节流，一个响应代码（通常成功返回 200），以及一个 MIME 类型以表示响应的是 HTML。浏览器将决定它知道 如何处理这一类型，并将结果解析成某种类型的文档模型。在这一模型中，可以找到对其它资源的引用：链接、图片、脚本、样式表等等。它会区别对待每一项，但 都是在解析原始资源的过程中去发现它们的。这里不存在服务描述，浏览器作为客户端，直接就知道如何分析结果。

RESTful 服务也应当使用同样的机制。URL 本身不应当是“魔力的”。客户端不应当被要求了解如何去分析一个 URL 或是对于一个层次相对另一层 次而言意味着什么有详细的认识。RESTful 客户端只需要取回资源，调查返回的 MIME 类型并分析结果。这样一来，客户端就知道如何分析返回的类型了。

举例来说，一个客户端可能会接收一个我们上面所描述的用于报表服务的主 RESTful 服务的引用：

```
http://company1.com/report/
```

如果是通过浏览器发出的请求，可能会返回一个包含如下引用的 HTML 文档：

```
http://company1.com/report/sales
```

用户可以点击进入并找到可浏览的年份列表。要说明的是浏览器对于 URL 的结构并没有特别的认知，但它知道如何分析结果并以用户可以浏览的结果返回内 容。

对于其它的 MIME 类型道理也是一样，比如以 XML 的格式请求 2009 年的季度报表：

```
http://company1.com/reports/sales/2009/qtr
```

可能得到：

```
<reports>
    <description>2009 Quarterly Reports</description>
    <report name="First Quarter"
src="http://company1.com/reports/sales/2009/qtr/1"/>
    <report name="Second Quarter"
src="http://company1.com/reports/sales/2009/qtr/2"/>
    <report name="Third Quarter"
src="http://company1.com/reports/sales/2009/qtr/3"/>
</reports>
```

你可以将 URL 想成是贯穿信息空间的向量。每一个层次都进一步的将你指向最终的资源。不同的路径可能产生同样的结果。客户端需要知道如何分析这些结 果，但通过对响应给出可识别的类型，我们可以触发合适的分析器。这一结构可通过爬虫降序的从引用来抓取，或者以某种接口展现给用户浏览。一个 RESTful 接口成为了客户端通过基于已知来请求信息的方式。它们以一个已知或者已发现的点作为开始，就像你浏览 web 一样来浏览信息。

这就是 HATEOS 所指代的。应用的状态在超文本响应中被转移和发现。就像浏览器需要知道 HTML、图片、声音文件等等一样，一个 RESTful 客 户端也需要知道如何分析解析资源引用的结果。然而，整个过程是简单、受约束、可伸缩并且灵活的——这正是我们所期望的网络软件系统的属性。

许多人搭建的"RESTful"系统都要求客户端事先知道 URL 的每一层次的意义。如果服务端将信息重组了，这些系统的客户端就会崩溃。真正体现了 HATEOS 的客户端与它们所通讯的服务器之间才更加能做到松耦合。

展望

我们每天挣扎于解决由快速变更的领域、技术、客户需求以及行动知识等等方面引出的问题。我们花费了太多时间来编写软件，将我们所学到的与我们所已知 的联系起来。对象与数据

库没能够跟上我们所经历的变革。我们需要找到新的方式来看待我们所产生和消费的信息，比过去的解决方案更为扩展并且更加健壮。我们 需要技术来帮助我们达成一致。我们不需要先达成了公共模型形式的一致才来使用我们的技术。

在这篇文章中，我们作了该系列的开篇介绍，并且开始探讨了 REST 与 Web 技术如何作为新的面向信息架构的基础。我们建立起了能够让我们统一引用各 种内容，服务与文档的命名模式。客户端可以利用这样的自由度以他们想要的形式来协商信息。当他们解析引用的时候，他们可以发现通过新的关系联系起来的新内 容。

这一架构风格和技术与相关的语义网技术结合起来，可以漂亮地创建强大的、可伸缩的和灵活的软件系统。它们创建数据 Web 的能力对我们的生活将产生的 影响就像 Web 之前所产生的那样。这将会是一场信息系统的革命，我们的许多认知都将有很大程度上的改变。它不仅能降低数据集成的成本，还会使得我们只能进 行想象的新型业务能力成为可能。

我们正在进入一个信息的新世界，信息可以被连接起来并使用，而不管它是被包含在文档内，数据库里，或者作为 RESTful 服务的结果而返回。我们将 能够发现内容并将其连接到我们所已知的部分。我们可以将现在隐藏在数据库，电子表格，报表或者其它仓筒的数据暴露出来。我们不仅能获得访问这些信息的权 限，还能够以我们想要的方式消费这些数据。

这是语义网的一个主要的，小小的目标。达到这个目标，而我们现在也有能力做到，将会是一切改变的开始。

原文链接： http://www.infoq.com/cn/articles/roa-rest-of-rest

相关内容：

- 使用Flash Builder 4 beta进行以数据为中心的开发
- 如何获取（GET）一杯咖啡——星巴克REST案例分析
- Web请求异步处理降低应用依赖风险
- RESTful服务的版本控制策略
- RESTful HTTP的实践

推荐文章 | Articles

使用Apache Shindig为客户应用程序加入社区内容

作者 Boris Lublinsky 译者 杨娟

社区网络是目前网络上最热门的事情之一，所以毫不奇怪，许多Web应用程序试图加上社区网络内容来吸引更多的访问者。尽管API正在标准化[1]， 但大多数现有的社区网站依然提供专有API，或者对OpenSocial的不同版本提供支持,而这正是许多开发人员所面临的挑战。结果是目前 OpenSocial有多版本的的API和大量的开源代码支持他们[2]。在这篇文章中我将讨论Apache Shindig[3]如 何实现OpenSocial，如何减少其中的一些问题。

我首先讨论OpenSocial的标准和Shingig架构，然后说明它们如何为一个应用程序添加社区网络内容。最后，将以OpenSocial Gypsii社区网络[4]为 例，讲解启用Shindig的方法。

OpenSocial APIs

OpenSocial 所定义的规范[5]：

“OpenSocial 是一组用于构建在网络上运行的社交应用程序的 API。OpenSocial 的目标是通过提供一个可在很多不同的环境中使用的通用 API ，让更多的应用程序被更多的用户访问。开发人员可以使用标准的 javascript 和 html 创建应用程序 ,运行在那些已经应用 OpenSocial 的社交网站中。这些网站作为 OpenSocial 的容器 ,允许开发者访问他们的社区信息 ;反过来，又为他们的用户收到了大量的应用程序。

OpenSocail API 暴露了在一个[社区网络]容器中获取用户以及他们的好友信息的方式。”

社区应用程序围绕人、活动和关系展开。人是社区网络软件和 OpenSocial API 的基本组成部

分。Person 对象提供对用户信息的访问接口，它包含了用户的基本信息或其他扩展的信息，如“最喜爱的电视节目”、“我所期望的工 作”以及“最喜欢的地方”。OpenSocial API 提供了用于查看和更改个人资料的支持。另一个重要的用户信息是他们在社交图中的连接。为了能够共享信息和体验朋友动态变化，OpenSocial 支 持用户的朋友概念，并提供检索用户朋友信息的 API。当应用程序想进行交互或显示“朋友的朋友”的数据，OpenSocial 允许增加一个你所做操作的可 选过滤器来支持扩展查询，如用户可以通过朋友的距离来排序。容器可以有选择地支持“朋友的朋友”查询，“朋友的朋友的朋友”的查询，等等。 OpenSocial 还允许朋友分享他们最近一直在做的信息-这些被称为活动。OpenSocial 的公开活动流是一个用户所采取的行动的集合。活动模板 允许应用程序开发人员在消息中为应用程序或用户数据块定义占位符。这种数据层和显示层的分离允许多个活动合并成活动包：活动综合包，让用户知道他们朋友消 息，而不必辛苦地搜集大量消息。社区互动的另一个重要方面是能够读取，张贴和删除用户之间在网络中的消息。 OpenSocial 定义的消息类型，包括公共消息（如文件的评论）和私人消息（仅限于某些个人和团体）。消息归纳，其中包括一个唯一的 ID，名称和信息 数量大小。

目前的规范定义了两种版本的API——JavaScript[6]和REST[7]。

OpenSocial的JavaScript规范是提供一个小工具Gadget[8]设 计来支持OpenSocial的功能。架构如图 1：

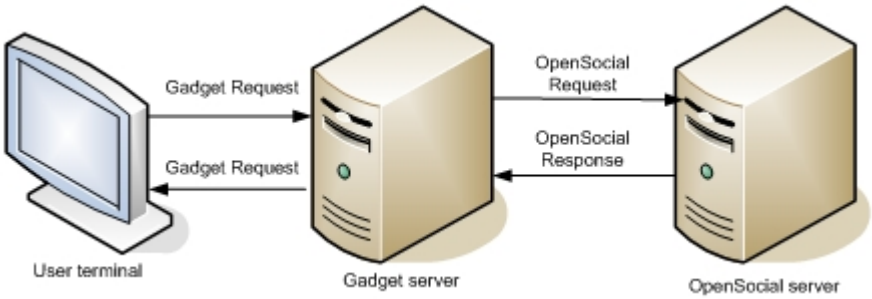


图 1 使用 JavaScript OpenSocial API 典型架构

在这种情况下，一个用户应用程序被作为一个 Gadgets 的集合，Gadget 自己用 OpenSoical API 可以获取社交信息。因此，JavaScript API 与 Gadget API 的紧密结合，并非总是适合不是基于 Gadget 的实现。

相反的，OpenSocail REST API 不直接与 Gadgets 关联，允许任何客户端应用程序来查看和发布用户的 OpenSocial 数据。我们将用这些 API 来实现 OpenSocial 应用。

Apache Shindig

Apache Shindig 是 Gadget API 和 OpenSocial API 的开源实现。下面将简单回顾一下什么是 Apache Shindig；Apache Shindig 是一个 OpenSocial 的容器，帮助您通过提供代码托管、代理请求和处理 REST 和 RPC 请求来快速启用 OpenSocial。

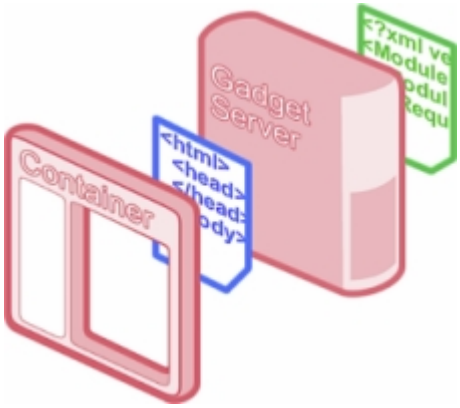


图 2 Apache Shindig[9]

对于OpenSocial，Shindig同时实现JavaScript和REST的API。 Shindig OpenSocial实现[10]总 体结构如图 3：

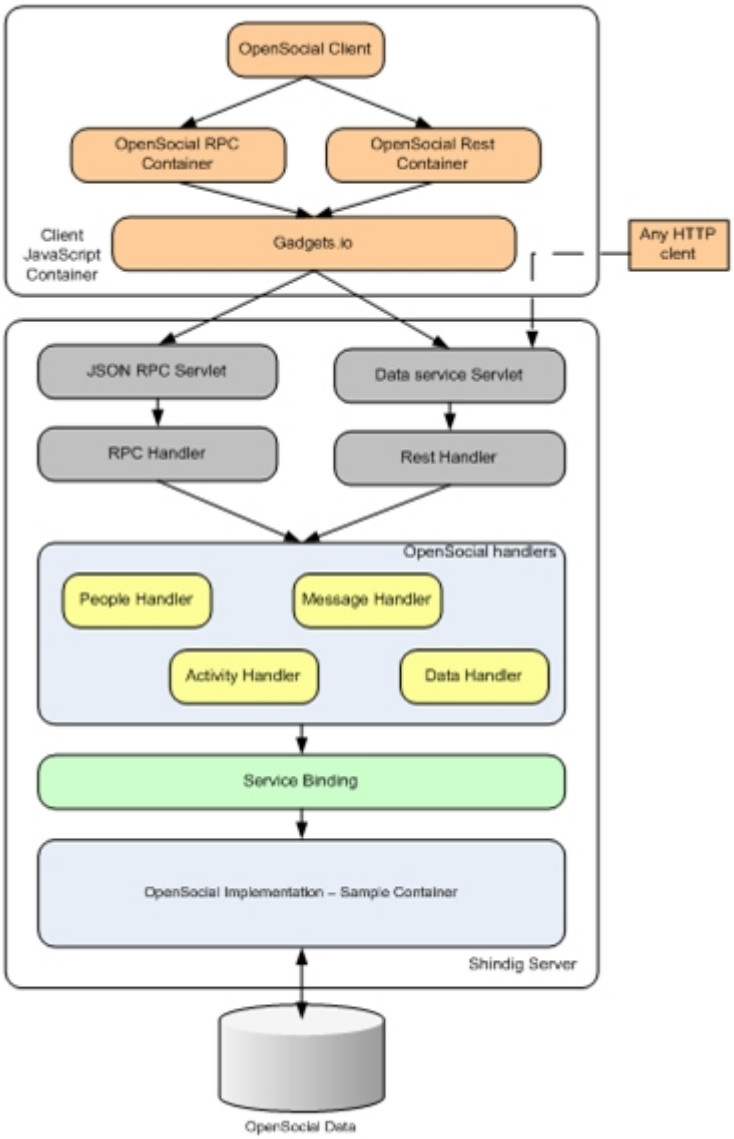


图 3 Shindig 实现 Open Social 的整体结构图

从图 3 中我们可以看到 Shindig 实现包括两个主要部分：客户端的 JavaScript 容器和 Shindig 服务器。 服务器是一个 WAR 文件，它可以运行在任何 Servlet 容器中-我们使用 Tomcat。正如上面所定义的，因为我们正在直接用 REST OpenSocial API，所以在这里我们不会讨论客户端 JavaScript 容器。

服务器执行包含两个监听器：JSON RPC Servlet 和数据服务（REST）Servlet。 每个 Servlet 被它自己的处理程序支持。 因此有两个处理器：RPC 和 REST 的处理器。这两种处理器共享同一基础类库，包括日期转换器（这里没有显示）和一系列 OpenSocial 的处理器：

- 人处理器—一种支持用户管理操作的处理器，如检索有关用户信息和他们的关系信息。
- 活动处理器—一种支持检索和存储一个用户活动信息的处理器。

- 数据处理器——种支持存储和检索应用数据的处理器。应用数据是一项信息，如一个偏好设置，由社区网络对于某个给定的应用根据一个用户的行 为来存储。
- 信息处理器——种用于社会网络中发送信息和检索信息的处理器。

Shindig服务实现基于依赖注入框架Guice[11]。 它使用的是绑定类（在web.xml文件中配置），所定义的服务被每一个处理器调用：即每个处理程序使用相应的服务。这种架构极其易于使用自定义实现覆盖 Shindig发布服务一部分的实现，如实现与现有社区网络应用程序的集成。

Gypsii

Gypsii[12]是 一个位置感知的社区网络。为实现共同的社区网络,除了传统的用户归属，它还允许其参与者存储和共享地点与位置信息，包括地址，地理坐标，可选的图片和说 明。GyPSii中一个特殊的地点是当前用户的位置。它也支持POI（兴趣点）和广告。

GyPSii提供了一套API：GyPSii OenExperience API（OEx）[13]、XML-RPC API，还提供了访问GyPSii所有功能的方法。 GyPSii XML-RPC的变种是由XML - RPC规范[14]创 造的，但已适应GyPSii的需要。该协议定义使用XML双方沟通时一个客户端服务器的通信模式。 Gypsii为开发人员提供了XML-RPC协议和应用了该协议的JAVA客户端实现的文档。

实现

整体实现架构

实现的整体架构如图 4。实现一个 Gypsii 的适配器，通过自定义 OpenSocial 容器替换 Shindig 提供的默认 OpenSocial 容器 的方式，使得现有的 Shindig 支持 OpenSocial REST。 请注意，这个自定义提供者可以连接到任何其他社区网络供应商，如 Facebook，MySpace 或 Ning。

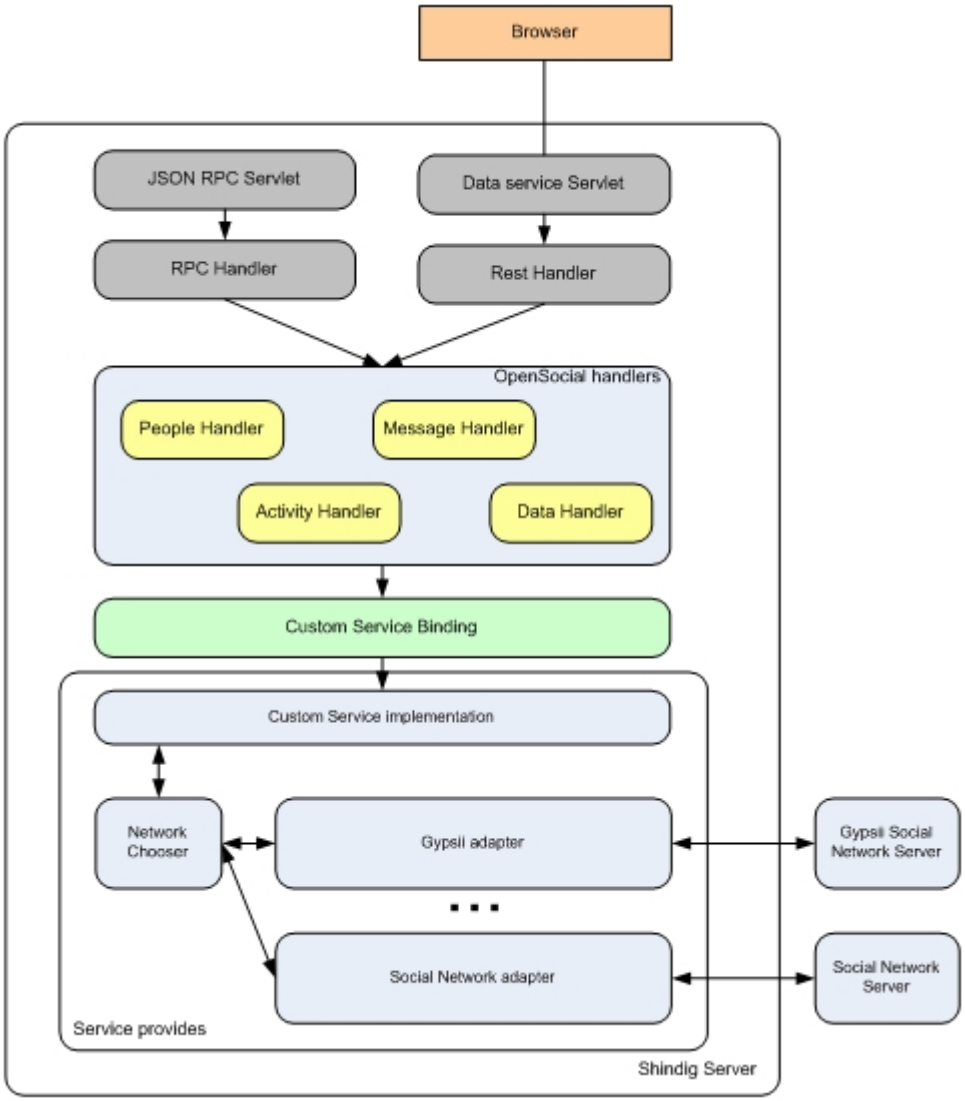


图 4 整体解决方案框架

这种方法的好处是有效地将我们的应用程序与特定的社区网络的实现/集成方法屏蔽开来。客户端 API 由 Shindig 提供和内部控制。因此，我们的应 用程序不依赖于具体的技术和社区网络供应商公开的 API。这些细节被封装在一个特定的网络适配器中。

安全

与多个社区网络供应商集成的复杂问题之一是安全/证书支持，范围涉及从一个登录用户名称/密码到OpenID[15]， 再到OAuth[16]。 我们的实现是基于一个简单的身份转换，如图 5 所示。

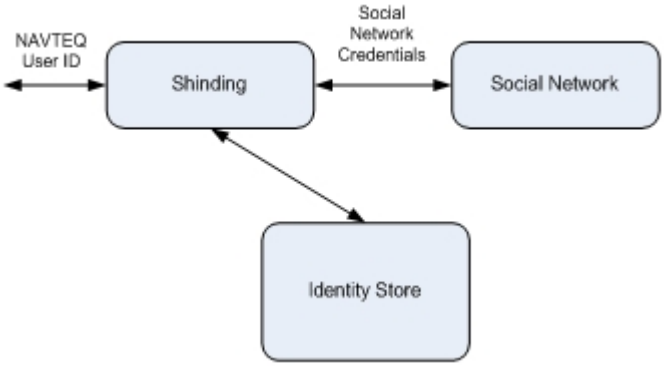


图 5 身份转换

例如，用户 ID 123468@gypsii 意味着 NAVTEQ 用户以 ID123468 访问 Gypsii。

用户请求具有 NAVTEQ's 用户 ID 的 Shindig：令牌。网络要求一种特定的社会网络适配器将令牌转换成证书。基于用户存储的转换包含转换所 需的信息：用户名/密码，OpenID，特定的键或其他任何一个特定的转换要求。这允许我们使用 Navteq 公司的用户 ID 作为 OpenSocial REST API 的标准化参数。它还允许多个供应商之间的社区网络的请求，基于以下命名约定简单实现：

```
OpenSocialUID = NAVTEQUID@Network
```

例如用户 ID 123468 @ GyPSii 意思是 ID 123468 的 NAVTEQ 用户访问 Gypsii。这允许构建一个非常简单的网络适配器（图 4），社区网络适配器根据所提供的用户名选择线路。

替换默认的社区服务

在依赖注入支持下，替换默认的社区网络服务就相当简单，主要包括以下主要步骤：

- **实现自定义 OpenSocial 服务。**自定义服务要么作为一个单独的类（清单 1）实现，要么为一组类共同实现 Shindig 所定义的 API。

```
public class CustomOpenSocialService implements ActivityService,
      AppDataService, PersonService, MessageService {
……..................................
}
```

清单 1 客户自定义的 OpenSocial 服务

- **创建绑定类。**一旦自定义的 OpenSocial 服务到位（我们假定一个：清单 1）,一个绑定类必须实现（清单 2）并告诉 Shindig 调用它处理请求。

```
public class NAVTEQBind extends SocialApiGuiceModule {
    ……
    bind(ActivityService.class).to(CustomOpenSocialService.class);
    bind(AppDataService.class).to(CustomOpenSocialService.class);
    bind(PersonService.class).to(CustomOpenSocialService.class);
    bind(MessageService.class).to(CustomOpenSocialService.class);
    ……
}
```

清单 2 绑定自定义 OpenSocial 服务

- **用绑定类配置 Shindig。**最后，这个绑定类必须知道 Shindig 的实现。通过一个配置文件 Web.xml 改变来处理 （清单 3）.这个配置文件是通过 org.apache.shindig.common.servlet.GuiceServletContextListener 类 调用 Shindig 应用程序和设置正确的绑定。

```
<context-param>
    <param-name>guice-modules</param-name>
   <param-value>
org.apache.shindig.common.PropertiesModule:com.navteq.opensocial.
bind.NAVTEQBind
   </param-value>
 </context-param>
```

清单 3 Web.config 变化

从浏览器中访问OpenSocial REST API

我们需要从浏览器中访问OpenSocial REST APIs。这里的问题是现在浏览器都具有同源策略限制[17]：

“同源策略是指阻止代码获得或者更改从另一个域名下获得的文件或者信息。也就是说我们的请求 URL 域必须和当前网站的域相同。这基本上意味 着，Web 浏览器将不同域的内容隔离以阻止它们彼此对终端用户的操作。”

这里有几种克服限制的常用方法：

- 具有 GET/POST 数据的网页，通过服务器端发送请求，服务器充当一个到达第三方服务

器代理。虽然使用广泛，但是这个方法却不够灵活， 无法伸缩，并增加了处理请求的延迟（负载大时会更明显）。

- 在页面中使用框架元素创建新区域来访问任何第三方内容。没有返回数据的 POSTs 方式运行的很好，框架间由于有限的访问很少应用 GET 方 式。但是试图下载文件时也会受到同源政策限制。
- 实现 GET 返回 JSONP（JSON with Padding）：在一个函数调用中包装的 JSON 数据，数据在一个方法调用中返回。当脚本加载时执行。因为同源策略不会阻止动态插入的代码，不会测试脚 本是否与 Web 页面来自相同的域。

由 Shindig 提供的 OpenSocial API 是利用 GET 从社区网络中获取信息和 POST 更新信息。为了支持 GET 方式，我们充分利用 JSONP，而为了支持 POST 方式，我们使用框架技术。

为了实现服务器上JSONP支持，我们使用了开源JSONP过滤器[18]， 不要求对Shindig的方案变动支持。在Shindig的Web.xml中定义过滤器用法（清单 4）：

```
<filter>
    <display-name>jsonp</display-name>
    <filter-name>jsonp</filter-name>
    <filter-class>org.jsonp.JsonpFilter</filter-class>
    <init-param>
      <param-name>jsonp</param-name>
      <param-value>jsonpCallback</param-value>
    </init-param>
    <init-param>
      <param-name>json-mime-types</param-name>
      <param-value>application/json</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>jsonp</filter-name>
    <url-pattern>*</url-pattern>
  </filter-mapping>
```

清单 4 配置 JSONP 过滤器

我们还必须稍微修改 Shindig 代码使之能够支持 Post 方式。Shindig 发布版不允许内容类型为 application/x-www-form-urlencoded， 因为它与 OAuth body signing 冲突。因为我们没有使用 OAuth 和 application/x-www-form-urlencoded 的 内容类型的 POST 浏览器，我们必须修改类 org.apache.shindig.protocol.ContentTypes（清 单 5）和 org.apache.shindig.protocol.DataServiceServlet（清单 6）允许 application/x-www-form-urlencoded：

```
……................
public static void checkContentTypes(Set<String>
allowedContentTypes,
     String contentType, boolean disallowUnknownContentTypes) throws
InvalidContentTypeException {


   if (StringUtils.isEmpty(contentType)) {
      if (disallowUnknownContentTypes) {
       throw new InvalidContentTypeException(
          "No Content-Type specified. One of "
             + StringUtils.join(allowedContentTypes, ", ") + " is
required");
      } else {
        // No content type specified, we can fail in other ways later.
        return;
      }
   }
   contentType = ContentTypes.extractMimePart(contentType);
//BL.  comented out to support
 //    if
(ContentTypes.FORBIDDEN_CONTENT_TYPES.contains(contentType)) {
 //      throw new InvalidContentTypeException(
 //          "Cannot use disallowed Content-Type " + contentType);
 //    }
.............................................
```

清单 5 修改 ContentTypes 类

```
……............

public static final Set<String> ALLOWED_CONTENT_TYPES =

     new
ImmutableSet.Builder<String>().addAll(ContentTypes.ALLOWED_JSON_C
ONTENT_TYPES)
```

```
        .addAll(ContentTypes.ALLOWED_XML_CONTENT_TYPES)
        .addAll(ContentTypes.FORBIDDEN_CONTENT_TYPES)
        .addAll(ContentTypes.ALLOWED_ATOM_CONTENT_TYPES).build();
```

……

清单 6 修改 DataServiceServlet 类

另一个问题是，浏览器 POST 方式提交名称/值。虽然，Shindig 支持 POST 名称/值，它也支持由 POST body 直接处理的数据。作为一个执行增加了一个额外的名称/值参数的主体内容。问题是，在浏览器提交的情况下，Body 是空的而其参数的内容被覆盖。org.apache.shindig.protocol.DefaultHandlerRegistry 类 （清单 7）更改修复了这个问题。

……..

```
public Future<?> execute(Map<String, String[]> parameters, Reader
body,
                        SecurityToken token, BeanConverter
converter) {
     try {
       // bind the body contents if available
       if (body != null) {
         String bString = IOUtils.toString(body);
         if(bString.length() > 0)
         parameters.put(operation.bodyParam(), new
String[]{bString});
       }
       RequestItem item = methodCaller.getRestRequestItem(parameters,
token, converter,
           beanJsonConverter);
       listener.executing(item);
       return methodCaller.call(handlerProvider.get(), item);
```

……..

清单 7 禁止覆盖空消息体

最后，当 POST 方式不会返回任何数据，Shindig 实现将空数据转换成一个空的 JSON 对象，返回的类型为 “application/json”。这将导致浏览器尝试主动与用户会话。org.apache.shindig.protocol.DataServiceServlet（清 单 7）解决了这一问题：

```
org.apache.shindig.protocol.DataServiceServlet (Listing 7) solves
this problem:

   servletResponse.setContentType(converter.getContentType());

   if (responseItem.getErrorCode() >= 200 &&
responseItem.getErrorCode() < 400) {

     Object response = responseItem.getResponse();

     // TODO: ugliness resulting from not using RestfulItem

     if (!(response instanceof DataCollection) && !(response
instanceof RestfulCollection)) {

        //BL - modified to not return fake responce

       if(response instanceof Map){

          if(((Map)response).isEmpty()){

             servletResponse.setContentType("text/plain");

             return;

          }

       }

       response = ImmutableMap.of("entry", response);

     }
```

清单 8 禁用空响应

示例应用程序

在这篇文章中，我们描述 OpenSococial 的实现，用于构造 GeoSpatial OpenSocial mashup，允许用户和他的朋友把他们最喜欢的地方放在地图上（图 6）

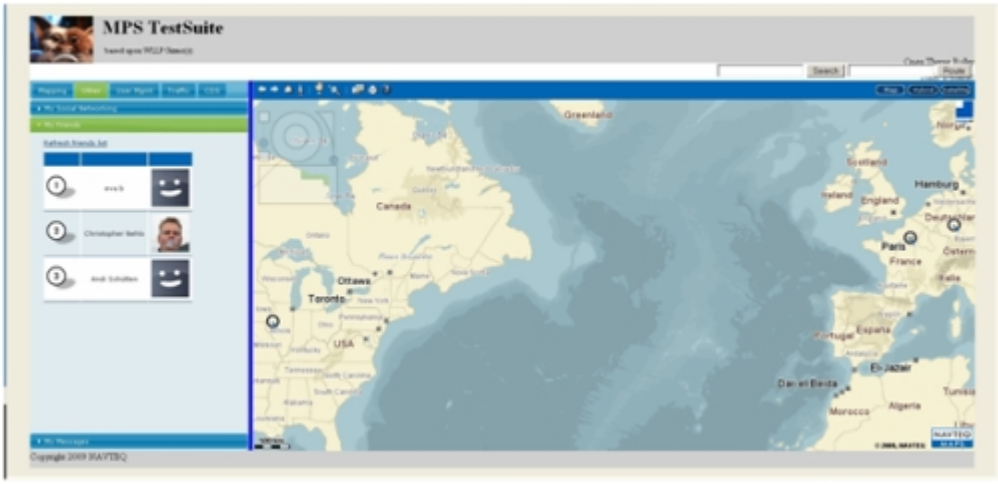


图 6 地图上用户和他的朋友

它还允许将用户信息和他最爱的地方以及社区网络信息（图 7）发送给另一个朋友。

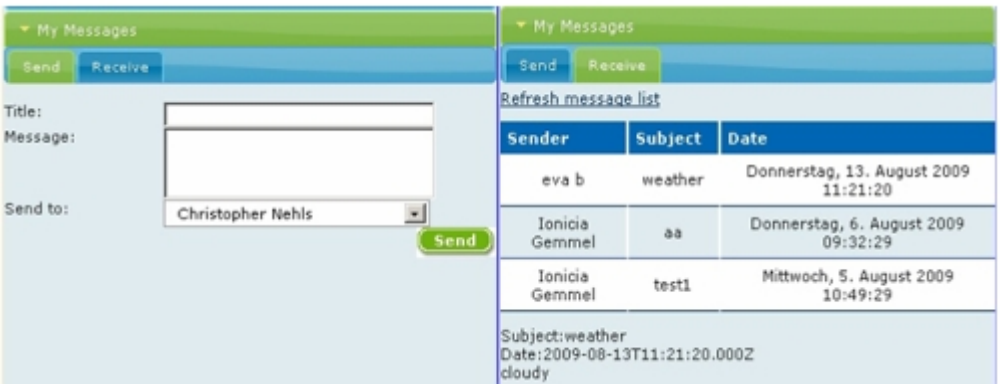


图 7 社区网络信息

这种简单的实现只是社区网络和地理空间数据之间许多潜在协同效应之一。基于以上所描述的结合两者的方法，能实现许多其他有趣的混搭网站。

结论

开源社区 API 的扩展使它很难开发出应用程序支持多种现有的社区网络。在本文中提出的一个解决方案，通过第三方提供的标准 API“隐藏”前端实现差 异来克服这些问题。 Apache Shindig，提供 API 和“插件”架构，大大简化了整个实现。

---

[1] http://www.opensocial.org/

[2] http://www.thegoodnamesweregone.com/post/83-open-source-web-api-frameworks.aspx

[3] http://incubator.apache.org/shindig/

[4] http://www.gypsii.com/

[5] http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/OpenSocial-Specification.html

[6] http://wiki.opensocial.org/index.php?title=JavaScript_API_Reference

[7] http://wiki.opensocial.org/index.php?title=OpenSocial_REST_Developer%27s_Guide

[8] http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html

[9] Source : http://incubator.apache.org/shindig/

[10] Rajdeep Dua. Architectural Overview of Shindig , an OpenSocial Reference Implementation. http://chrisschalk.com/shindig_docs/rajdeep/shindig-overview/onjava-shindig-overview-tidy.html and Rajdeep Dua. Overview of REST Implementation in Shindig - Java Version. http://sites.google.com/site/opensocialarticles/Home/shindig-rest-java

[11] http://code.google.com/p/google-guice/

[12] http://www.gypsii.com/home.cgi

[13] http://developer.gypsii.com/docs.cgi?op=view&id=api.html

[14] http://www.xmlrpc.com/

[15] http://openid.net/

[16] http://oauth.net/

[17] Seda Özses, Salih Ergül Cross-domain communications with JSONP, Part 1: Combine JSONP and jQuery to quickly build powerful mashups. http://www.ibm.com/developerworks/library/wa-aj-jsonp1/

[18] http://code.google.com/p/jsonp-java/

原文链接：http://www.infoq.com/cn/articles/social-content-apache-shindig

相关内容：

- OpenSocial案例专访：TheBroth.com和KlickNation.com
- OpenSocial规范、实现现状与展望
- OpenSocial的分析与实现
- OpenSocial只是Google的一个公关骗局？
- QClub北京：OpenSocial的规范、实践与展望

书评：《代码之道》

作者 崔康

乍一看书名——《代码之道》——呵呵，好大的口气！何为道？道家 创始人老子认为“道”是宇宙的本原和普遍规律。这样一本仅仅 200 页的薄书能担得起“道”之名吗？笔者带着一丝怀疑翻开这本备受瞩目的书，细细的阅读，不 知不觉中已是豁然开朗——书的内容言简意赅，每个话题占用仅仅数页，却清晰明白的讲述了若干个道理，笔者不由得感慨：在我们身心疲惫的或是斗志昂扬的追逐 于各种丛出不穷的新技术、新概念的时候，我们能否暂停脚步，想一想走过的路，看一看类似《代码之道》这样的非技术书呢？

《代码之道》英文原书名为《I.M.Wright's "Hard Code"》，属于微软核心技术丛书之一，作者 Eric Brechner，现任微软公司部门总监，在软件行业拥有超过 20 年的工作经验，从 2001 年起在微软内部刊物上采用 I.M.Wright 的笔名撰写 “Hard Code”栏目，其中讨论软件开发过程和职业生涯的种种问题，这本书就是对这个栏目的总结，话题包括：

- 项目管理
- 过程改进
- 软件质量
- 软件设计
- 提高效率
- 职业发展
- 自我完善
- ……

针对每一个话题，I.M.Wright 都根据丰富的工作经验提出了自己的观点，并介绍了来龙去脉，令人信服。笔者认为，大多数程序员，过于关注新技 术和新概念，往往忽略这种对工作的

思考和总结——现在，技术更新换代的速度令人咋舌，虽然提倡终身学习，但我们的有限精力应该放在“核心竞争力”上。不论 技术如何发展，有一些能力是可以应用于各种环境的：

- 解决问题的能力
- 时间的安排
- 事务优先级的处理
- 工作与生活的平衡原则

《代码之道》对“核心竞争力”的各个方面也提出了自己的观点。对这本 200 页的精炼小书来说，笔者很难像写普通的书评那样对这 49 个主题再加以概括 总结，因此，笔者挑选了几个有关“核心竞争力”的主题，结合自己的实际体会，谈一谈《代码之道》提供的经验。

1. 如何集中精力、避免打扰？

在日常工作中，电子邮件、即时通讯工具、口头交流等等都会在你埋头苦干的时候打扰你，如何应对呢？《代码之道》引用了两个研究成果（p131）：

- 沉重的情境切换与抽大麻相比，会导致更多的智商下降。
- 员工在刚好的两个项目同时工作时最富有生产力。

I.M.Wright 提倡的方法是：在工作时，把邮件通知提示等都尽量关闭，然后安心做自己的工作；定期的（比如，每隔半小时）浏览收到的邮件，并 且在查 看邮件时尽力把它完整处理掉，避免情境的切换。你可能会认为这样会导致某些低优先级的邮件浪费时间，但 I.M.Wright 却认为你在过后再次切换到这封 邮件所消耗的精力和时间要比即时处理还要多。

笔者也一直受到工作时被打断的困扰。特别是，一看到新邮件的提示，总是放下手中的活，冲动的打开新邮件先看大体内容，殊不知，这就导致了一次情境切 换，有时候暂时不想处理，于是关掉邮件，继续手中的工作，又一次情境切换。就这样白白浪费了自己的精力和时间。笔者猜想，有相当数量的程序员在日常工作中 被各种各样的事情打断，每天的有效工作时间逐渐压缩，这是否也是导致程序员经常加班的原因之一呢？

I.M.Wright 提出要想避免被打扰，还可以“让自己消失”，去一个没人能找到你的地方，使用笔记本电脑和远程桌面安心工作，或者选择一个其他人不在的时候工作，比如早上 早一点来公司，晚上回家用远程桌面工作。在这样一个安静的地方和时刻，你就可以抛开不必要的困扰，集中精力做一些工作。

笔者对这个观点非常赞同，也是自己之前亲身实践过的。一段时间以来，笔者坚持早上八点半到公司，此时偌大的办公区基本空无一人，然后带上笔记本电 脑，找一个小会议室，一直工作到上午 10 点钟才回到自己的办公桌。这段时间非常的安静，不论上班的同事如何来来往往，笔者都不受干扰，集中处理一些事情， 效率很高， 即使在平常工作时间，也会尽量多呆在安静的地方。同时，笔者发现一个有趣的现象：当我呆在办公桌的时候，时不时的就会有同事询问各种各样的问题，小到一个 域名对应的 IP 地址，大到项目计划的制定。但是，一旦我呆在会议室工作时，这些口头的问题就不翼而飞了，小的问题同事们可以独立解决，大的问题会通过正式 的邮件和会议进行交流，这样大家的工作效率都会有所提高——因为极大减少了彼此的情境切换次数。

2. 平衡工作和生活

这个话题对程序员来说有些沉重和苦涩，软件开发似乎是个工作和生活严重失衡的职业。不过，I.M.Wright 认为这不是命运注定，他在应聘微软的 时候，就向未来的上司提出，他非常顾家，需要保证正常的工作时间，早上送孩子上学，晚上回家用餐，而老板完全同意了他的要求，并且说话算话。 I.M.Wright 提出了一个平衡工作和生活的五步原则：

1. 了解并接受你选择的生活方式。
2. 跟你的管理者一起设置一些基本原则。
3. 不要默默妥协。
4. 必要的时候使用“远程访问服务”和远程桌面访问。
5. 抛开分离造成的精神分裂错觉。

具体的说，你需要了解你自己，好好思考一下你的工作目的是什么，事业和生活的优先级是什么，你的忍耐程度是多大，一旦把这些问题想明白了，你就会在 平衡问题上做出正确的选择；把自己制定的原则告诉你的上司，不要担心你的工作和前途，在第一步时你已经把这些问题和限度考虑清楚。I.M.Wright 发 现这种交流反倒会使你赢得上司的尊重，因为你具有强烈的自信和清晰的价值观；在工作中，偶尔的破例是可以容忍的，但是如果经常出现超出原则的情况，你就需 要找机会重申你的态度，因为一旦你很容易就妥协了，上司会认为你并不在意你自己指定的原则（如果连你自己都不在乎，又能指望谁来在乎呢？ ）。他可能会提出 越来越多的要求，直到你的原则一退再退，你原先的努力就白费了；如果你热爱自己的工作而且的确很重要，可以在家里通过远程访问等方式访问；不要试图把工作 和生活分开，保持一个统一的心态，避免情景切换。

笔者在平衡方面也是个菜鸟，很少思考自己的原则问题，平常也是迁就工作，压缩生活空间，《代码之道》这方面给了很多启示，我们需要严肃的、正式的考 虑一下平衡问题，有类似需求的程序员朋友们可以参考 I.M.Wright 提出的五步原则。

《代码之道》谈论的主题很多，不是一篇短短的书评可以全部覆盖到的，笔者觉得这本书很适合程序员们阅读，不是丰富我们的技术水平，而是为了让我们工 作和生活更有效率、更有意义。

附：华章授权InfoQ中文站独家为大家提供额外的样章进行试读：欢迎下载第 3 章、第 9 章。

原文链接：http://www.infoq.com/cn/articles/bookreview-hard-code

相关内容：

- 书摘与采访：依赖注入
- 书摘与采访：Rails for .NET Developers
- 《服务设计的SOA原则》的设计实践摘录
- 《Ladder to SOE》作者访谈
- 程序员需要知道的 97 件事

新品推荐 | New Products

微软发布OData SDK和“Dallas” CTP2

作者 Abel Avram 译者 朱永光

微软发布了用于.NET、Java、PHP、Objective-C（用于 iPhone 和 Mac）和 JavaScript 的 OData SDK，协助开发人员创建相应的客户端来使用基于 OData 的信息；也发布了“Dallas”的 CTP 2，一个用于销售和购买某些数据的网络商店。

原文链接：http://www.infoq.com/cn/news/2010/03/OData-SDK-Dallas-CTP-2

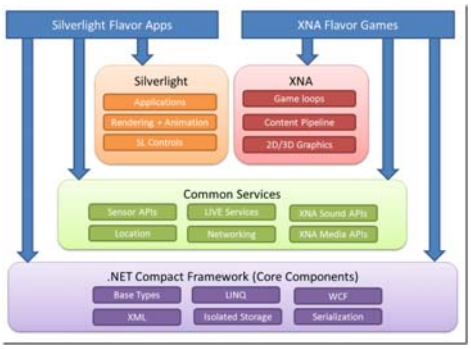
Eclipse Git插件EGit发布了

作者 Alex Blewitt 译者 张龙

org.eclipse EGit v0.7.1 的首个公开版于 EclipseCon 发布了。EGit 构建在纯 Java 实现 JGit 之上，这意味着它并不依赖于外部程序库和本地代码，而这些内容阻碍了 Eclipse 对 Subversion 的支持。

原文链接：http://www.infoq.com/cn/news/2010/03/egit-released

Windows Phone 7 Series将不支持本地代码



作者 Jonathan Allen 译者 张龙

微软开发 Windows Phone 7 Series 的事实终于大白于天下。正如人们所怀疑的那样，Windows Mobile 7 的开发将主要基于 Silverlight、XNA 和 Flash。事实上，该平台只支持托管代码。

原文链接：http://www.infoq.com/cn/news/2010/03/Mobile7-NETCF

SpringSource发布TC Server Spring Edition

作者Josh Long**译者** 张龙

近日 SpringSource 发布了一个更新版的 TC Server Spring Edition。TC Server 是由 SpringSource 出品的基于 Tomcat 的服务器 ,提供了增强的监控与云功能，目标是提升开发者的生产率。

原文链接：http://www.infoq.com/cn/news/2010/03/springsource_tc_server_spring

WindowBuilder Pro v7.4 发布：更棒的性能、更快的解析

作者 张龙

近日 Instantiations, Inc 发布了 WindowBuilder Pro v7.4、SWT Designer v7.4、Swing Designer v7.4 以及 GWT Designer v7.4。这些发布对处于业界领导地位的 Java GUI builders for SWT、Swing、RCP 及 GWT 提供了很多增强。

原文链接：http://www.infoq.com/cn/news/2010/03/WindowBuilderPro

U-Prove:在保护隐私的情况下提供安全性

作者 Abel Avram 译者侯伯薇

微软最近公开了 U-Prove CTP 版本的源码，它是一种用来实施认证的加密的解决方案，并且不会透露关于用户的个人信息。CTP 版本包括 U-Prove 加密规范 V1.0、对该规范的 C#和 Java 实现、对 WIF、AD FS2 和 CardSpace 2 的扩展，此外还有大量用来说明该项技术的白皮书。

原文链接：http://www.infoq.com/cn/news/2010/03/U-Prove-Security-Privacy

OpenCredo宣布为Spring Integration增加AMQP支持

作者 Josh Long 译者 张龙

近日 OpenCredo Ltd 宣布支持 Spring Integration（轻量级类似于 ESB 的消息框架）与基于 Advanced Message Queuing Protocol（AMQP）的消息服务器的通信，这有助于那些不支持 JMS 的 MOM 厂商争取 Spring Integration 用户。虽然一些 Message Queues(如 RabbitMQ）可伸缩性非常棒，但却不支持 JMS。

原文链接：http://www.infoq.com/cn/news/2010/03/amqp_support_spring_integration

策划：InfoQ 中文站

执行：霍泰稳 刘申

导言

RIA（富互联网应用，Rich Internet Application）是近几年随着互联网环境的改善而异军突起的一个热点词汇，前有 Adobe、微软等公司的推波助澜，后有开源技术 Laszlo 的参与，让这一技术在各大网站得到很好的应用，比如在线购物、视频网站等，而 Google 等厂商大力支持的 HTML5 的加入，更是将其推向高潮。随着 Sun 的被收购，JavaFX 已经少人问津，Adobe 和微软在加紧推出自己的新品，Ajax 和 JavaScript 的普及，HTML5 因其标准的公开性也逐渐为更多人接受。基于此，我们邀请了 RIA 各个技术片区的专家，结合自身的实践，来分析和分享其研究领域的案例，使得读者通过此专题能够大体了解该技术的长处和不足，并为自己在做相关技术选型时做参考。

构建前端UI组件的新思路

前端 UI 组件，目前流行的实现方式大多源自传统客户端的 UI 设计体系。无论是早期的 Bindows，还是近几年兴盛的 ExtJS，其 UI 组件都在模仿客户端软件，代码实现建立在复杂的继承体系上。好处是可以构建出和客户端体验一致的一整套 UI 组件，但弊端也很明显：组件长得都差不多，代码则继承太深，牵三挂四，不够轻便。

如何才能让前端 UI 组件轻便灵活起来呢？首先得意识到 Web UI 设计有自己的独特性。Web 页面可分为两种：一种是以展现信息为主的 Web 页面（web page），另一种是以操作信息为主的 Web 应用（web app）。对复杂的 Web 应用来说，可以采用 ExtJS 等类库来构建类客户端体验。但是，越来越多的 Web 应用已逐步脱离模仿客户端的阶段，开始从 Web 的独特性出发，将传统 UI 组件的功能融入到 Web 页面中。

举一个例子：国内的 Web 邮箱，无论是 163 还是 QQ 邮箱，都会让人联想起 Outlook 或 Foxmail 等客户端软件。这种模仿成本很高，然而带来的效果个人觉得却不是很好。反观 Gmail，整体 UI 设计，简明轻快，但功能一点也不逊色。在 Gmail 里，看不到 Tree，看不到 DataGrid，脱离了这些传统 UI 组件的框框，将功能融入到 Web 元素里，反而让用户更自然、更高效。

前端 UI 组件的 Web 特性，需要我们打破传统思维，换一个角度，重新去思考 UI 组件的基本组成要素。换什么角度去思考？什么角度才是合适的？这离不开具体场景。下面以一个实例来说明。

在淘宝页面中，以下是几个常见的 UI 组件：

传统思路里，我们会构建三个组件来实现：

- Slide（轮播）组件
- Tabs（标签页）组件
- Carousel（旋转木马）组件

Tabs 组件，我相信任何一个前端开发工程师最多半天都能搞定，而且还能把延迟触发、动态加载等特性也给实现了。Slide 组件也不难。Carousel 组件看起来稍微复杂一些，但熬上一个通宵研究研究循环的实现，也不是什么大难题。

上面是传统思路。新思路是：我们真的需要分三个组件来实现吗？

仔细思索，我们可以提取出这三个组件的共同点：

- 都有一组触点（Triggers）。 Slide 的触点是数字，Tabs 的触点是标签头，Carousel 的触点是小图。
- 都有一组面板（Panels）。就是内容区域。
- 通过触点可以让面板切换（Switch）。

上面的几条里有一个很重要的概念：切换。所有这些组件的共同点是可切换的。如果我们实现了一个可切换（Switchable）组件，上面三种组件就都是特例了。

根据上面的抽象，很容易实现 switchable.js：

```
function Switchable(container, config) {
   …
}
S.mix(Switchable.prototype, {
    init: function() { }
    switchTo: function() { }
    next: function() { }
    prev: function() { }
});
```

上面仅实现了基本的切换功能，我们可以进一步通过插件来实现更多功能：

- plugin-autoplay.js – 提供自动播放功能。
- plugin-effect.js – 提供切换时的各种特效。
- plugin-circular.js – 提供循环切换功能，比如 Slide 自动播放到第 5 张时，能无缝循环播放到第 1 张。
- plugin-lazyload.js – 提供数据的延迟加载功能。

插件的实现在 JavaScript 这种动态语言里是小菜一碟。至少有两种思路，一种是埋好钩子（hooks），插件根据钩子进行扩展：

```
S.mix(Switchable.prototype, {
    init: function() {
        …
        S.each(Switchable.Plugins, function(plugin) {
            if(S.isFunction(plugin.init)) {
                plugin.init();
            }
       });
    },
    switchTo: function(index) {
        …
        this.fire('beforeSwitch', {toIndex: index});
        …
    }
});
```

在插件的代码里，定义好 init 方法，以及监听相关事件（事件可以看成是一类 hooks）即可实现需要增加的功能。

插件的第二种实现方法是动态修改基础对象，可以重写某些方法，也可以利用 AOP 特性，将增加的功能织入到基础对象中：

```
S.weave(function() {
      // plugin code
      });
}, 'after', Switchable.prototype, 'init');
```

上面的代码表示在 Switchable 的 init 方法执行完成后，再紧接着执行 plugin code。

通过这种方式，我们无需用到任何继承概念，没有 super，没有 extend，利用 JavaScript 的原生动态语言特性，就比较完美地解决了问题。

从 Switchable 的角度看，上面三个组件可以描述为：

- Tabs 是普通的 Switchable 组件。
- Slide 是可自动切换且切换时有特效的 Switchable 组件。
- Carousel 是可自动切换、切换有特效、可循环切换的 Switchable 组件。

来看下 Slide 的实现，变得非常简单：

```
var defaultConfig = {
    autoplay: true,
    circular: true
};
Function Slide(container, config) {
    config = S.merge(defaultConfig, config);
    Slide.superclass.constructor.call(this, container, config);
}
S.extend(Slide, S.Switchable);
```

这里用到了类似 YUI 的 extend 方法，实现了继承，同时较好的保持了 JavaScript 的原汁原味。

可以看出，Tabs、Slide 和 Carousel 组件，彼此之间没有本质差别。封装出这三个类，仅仅是为了让开发者能方便快捷的调用（这些是高级 API）。对于资深开发者来说，在实例化 Switchable 时，通过传入不同参数即可实现所需效果（Switchable 是中级 API）。

更有意思的事情是，换一种思路实现代码后，也能帮助我们换一种思路看世界：

这个组件，可以看成是触点为图片的 Tabs 组件。左右两个翻页，无非是调用 next/prev 方法。进一步：

- Tabs 组件可以看成是仅有基本切换功能的 Slide。
- Slide 可以看成是触点悬浮在图片上面的 Tabs。
- 等等等等

最后会发现，这三个组件，本身就是相通的。原本同一物，何必分开实现呢？我们可以得到一个结论：

> 只要符合 switchable 可切换特性的 UI 组件，原则上都可以通过 Switchable 实现。唯一限制的是您的想象力！

比如，在 Switchable 的基础上，我们可以进一步实现 Album（画报），实现 CoverFlow（仿 iTunes 的封面切换效果）等等。

上面对 UI 组件的思维角度是可切换（Switchable），这是一个形容词。进一步思考，我们还可以得到以下形容词：

- Draggable – 可拖曳的
- Positionable – 可定位的
- Selectable – 可选择的
- Sortable – 可排序的
- Stackable – 可堆叠的

- Clickable – 可点击的
- Ajaxable – 可 ajax 的
- …

在这种新思路下，前端 UI 组件的基本组成要素已不是 Panel、Memu 和 Button 等传统 UI 基础单元，而是上面这些形容词。假设我们要实现一个可拖曳的可动态加载数据的可排序的表格时，或许像下面这样写一行代码就实现了：

```
S.Widget('tableId').draggable().ajaxable().sortable();
```

> 这是一个梦想。但有梦，去追，说不定就能实现。

备注

上面的代码里使用了KISSY UI类库，详细请参考：
http://kissy.googlecode.com/

Switchable的详尽代码实现请参见：
http://kissy.googlecode.com/svn/trunk/src/switchable/

作者简介：王保平，前端架构师，网名射雕，花名玉伯。崇尚简洁而不简单，相信付出才有收获。就职于淘宝网UED部，忙并快乐着。个人博客。

基于Flash的软件开发

这里所提及的 Flash 技术，包括 Adobe 的 Flash Player 以及 AIR。基于 Flash 的软件开发，涉及的人比较少——更多的人或许认为 Flash 更适合做在线产品，比如 Media Player、Data Show 等等。其实做软件开发也是一样，只要 Flash 适合你的产品，为何不抛开传统的开发理念而采用 Flash 呢？

本文根据自己开发维护DoSWF(详细信息请查看 :http://www.doswf.org)的经验 ,对基于Flash的软件开发，做一个全面的解剖。

为什么要基于Flash做软件开发

- **强大 UI 处理、交互设计功能**：借助于 Flash IDE 以及 Flex Builder，可以让你的软件的 UI 展现、用户体验完成得非常出色。
- **跨平台**：Adobe 已经为跨平台做了很多工作了，包括 Flash Player 以及 AIR 的部署。这让你的软件更便捷的做跨平台分发。
- **Online 和 Client 完美结合**：传统的软件开发模式，更多的是面向客户端，如果要和在线端结合的话，还得重新做在线版（浏览器插件之类）。而基于 Flash 开发则完全可以将 Online 版本发布为 Flash Player 平台，而 Client 版本发布为 AIR 平台，将二者完美结合。
- **丰富的第三方产品支持**：很多时候你会发现，Flash 几乎能完成你的软件所有的功能，但是就是有一个关键的功能实现不了，比如调节系统音量功能——AIR 也不能实现！这个时候，你就得善用 Flash 的第三方产品。最常见的就是 Flash 打包工具了。很多打包工具提供了很全面的系统级别接口供 flash 来调用。比如 ZINC，Flash Packer，都是非常不错的打包工具。借助于这些优秀的第三方产品，你可以将 Flash 技术应用到一个更高的境界。

Flash能做什么

和传统的软件开发技术不一样的是，相对来说 Flash 能做的实在是太少了。Flash 从一开始就

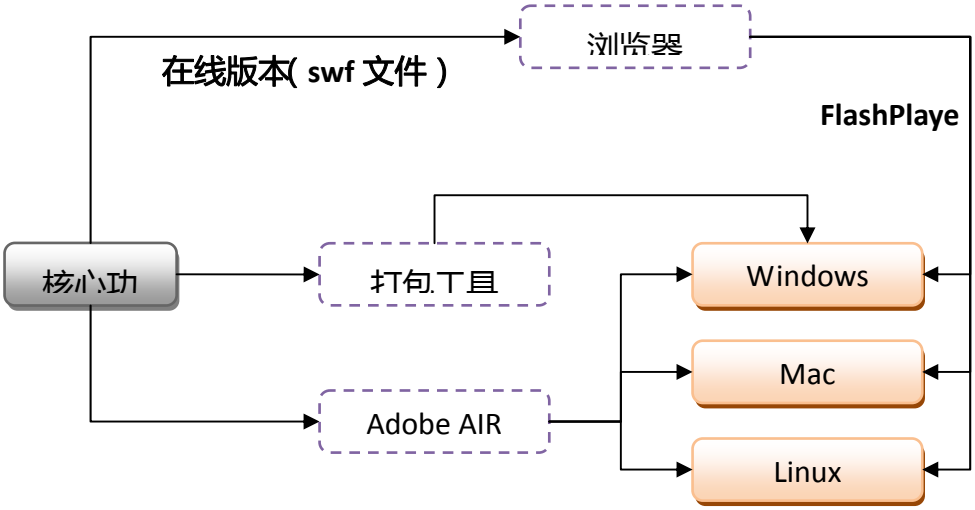
依附于浏览器，就注定它要在安全方面如履薄冰。从最简单的 Actionscript 1.0，到 Actionscript 2.0，再到目前相对比较强大的 Actionscript 3.0，虽然说目前用 Flash 我们能做越来越多的事了，但因为安全问题，Flash 仍然有很多事不能做。就算是 Adobe 后来推出的 AIR 技术，也无法完全跨越安全的限制。

所以在你确定要使用 Flash 技术进行软件开发前，你必须要知道 Flash 什么不能做，什么能做。这样你才能知道 Flash 技术是否适合在你的产品上使用。比如你要开发一个杀毒软件，Flash 能做吗？显然不能。但是如果你要开发一个 RSS 订阅器，这个完全则可以实现。所以，如果有必要，请事先咨询一些 Flash 开发人员，他们更清楚 Flash 的优势和劣势，更清楚你的产品是否适合使用 Flash 技术。

怎样基于Flash开发软件

产品架构

既然我们使用 Flash 来开发我们的软件，那么我们在架构产品时就应该更多的让 Flash 来发挥它的优势：跨平台性，以及 Online Application 和 Client 完美结合。这里看一下 DoSWF 设计初的产品架构：



产品分三条主线分发：

- 借助于浏览器，通过 Flash Player 的跨平台特性以及其高安装率分发软件的在线版本。
- 通过打包工具，将软件打包成客户端程序。选择不同的打包工具会有不同的目标平台。

这里仅打包 Windows 平台可执行程序。

- 借助于 Adobe AIR 的跨平台性，将软件发布 AIR 版本，实现客户端的跨平台。

不难看出，基于 Flash 的软件开发，在跨平台上的分发上，比传统的软件开发方式就存在很大优势。更特别的是借助于浏览器，以及 Flash Player 的高安装率，可以通过发布在线版本（如果功能上可以的话，比如你的软件是一个聊天软件），让你的产品更快更便捷地呈现给用户，服务于用户。

功能开发

Adobe Flash 系列产品，包括 Flash IDE, Flex, Flex Builder, Flash Player, AIR。借助于这些优秀的软件，你可以很方便很快捷地进行功能开发。

1. 环境编译

很多时候你的软件要发布成不同的版本，比如一个用于在线发布，一个用于打包成 exe 发布，而另一个用于发布成 AIR 程序。每个版本所对应的接口是完全不一样的。你不可能每个版本都开发一次吧？ 抑或是将软件模型提取出来，接口调用再各写一份？这样能解决问题，但是在后期的版本维护中，你会发现版本非常混乱。

最好的解决方案就是使用环境编译（详细信息请查看官方帮助）。它容许你在编译时有条件的选择相应的代码编译。比如：

```
//==============================================
//更新检查
//==============================================
var updater:IUpdater;

CONFIG::air {
updater = new AIRUpdater();
}
CONFIG::exe {
updater = new FlexUpdater();
}
CONFIG::online {
updater = new OnlineUpdater();
}
```

2. 换肤以及多语言

Flash 生来就是做 UI 的，这是 C、Java 等传统软件开发模式所不能比拟的。特别是借助于 Flex Framework，里面内置了很丰富的组件，使开发人员完全可以从 UI 处理中脱离出来，进而更关注软件的核心实现。通过 Flex Framework，可以很方便的完成换肤以及多语言支持。

在我看来，开发 Web 游戏，或在线应用，是否使用 Flex Framework 确实需要很好的斟酌一下。但是如果是进行软件开发，完全可以尝试一下——你不需要考虑它的体积（几百 KB 而已），也不需要考虑它的效率（比起你程序核心的消耗，它将会是很低的）。当然了，如果你的程序有发布在线版本的计划，如果在浏览器中运行效果不是很尽如人意，可以考虑将在线版本发布成一个 Demo 版本。

3. 自动升级

这个是软件开发必须面对的一个问题。Adobe AIR 目前的版本已经集成了自动升级模块。如果你的程序发布成 AIR 版本的话，可以很方便地完成自动升级的开发。但是发布成另外的版本，比如 Windows 的 exe 版本，就得自己去架构自动升级模块了。需要说明的是，最好是复用 AIR 升级模块的 xml 文件格式，不然维护这个升级文件会很复杂。

商业应用

这个是所有商业软件需要面对的一个问题。最最基本的模式就是出售授权来获取商业利益；如果再高级一点，就是提供商业性质的服务——比如顾问、软件定制等等。这些都不是重点，而且我确实没有深入研究。但是当你的软件发展到一定阶段的时候，我个人认为有一个东西就应该考虑了：产品系列化。

像 DoSWF，之前只有一个 Professional 版本，慢慢我发现很多人只用里面最最核心的几个功能，其他功能他们根本用不着，而且也不知道怎么用——并不是每个人都很熟悉你软件所涉及的领域。于是后来我把 DoSWF 里面最核心的功能提取出来，发布了 DoSWF MINI 版本，这样用户完全从 DoSWF Professional“混乱”的功能中脱离出来，而且费用更低。再到后来，我发现 DoSWF 不能满足用户针对项目的需求，于是又有了 FPE（Flash Project Encrypter)。这就是产品的系列化。

在软件一开始你可能会对整个产品用户有一个很准确地把控。但是实践会告诉你，你的用户远远超出你的想象。所以，产品的系列化是产品发展中一个很重要的环节，同时也意味着你要去做数据统计、分析，然后重新审视你的用户群体。这些就不只是针对基于Flash软件开发而言了。补充一点 如果你的程序发布成AIR版本后 ,可以将软件发布到Adobe的AIR Market。

总结

做软件即做产品，是一个很深很广的领域，这里只是对基于 Flash 的软件开发做一个抛砖引玉。很多产品，换一种思维，换一种方式，突破传统的束缚，你会有一个更新更广的天地。

个人简介：陈了然，网名laan，2004 年接触Flash，入门较晚，倍感压力。08 年开始研究Flash加密与混淆，并自主开发一款Flash加密软件。曾就职于sina事业产品部，以及sohu研发中心。现就职在百度Flash开发组。个人技术博客：http://www.laaan.cn/。

案例分析：Silverlight在中国人寿的应用

笔者自 2003 年首次听到 Macromedia 公司提起 RIA(富互联网应用)一词到现在整整 7 年了。一度被认为是互联网应用趋势的 RIA 经历了 7 年之痒，但仍然没有在互联网上得到大规模普及，特别是企业应用就更加少见。做个不恰当的比喻，传统基于 Html 的应用就像互联网应用中的绿叶一样，而 RIA 技术由于酷炫的用户体验效果就像是美丽的花朵。现在开心网和腾讯 QQ 等商业应用中已经运用了 RIA 技术在其社交网站中得到应用，但这毕竟还是少数，大多数互联网应用特别是企业级应用仍然选择传统高稳定性与高响应能力的 Html 应用路线，那未来的互联网应用，特别是企业应用能够迎来 RIA 的春天，绽放出漫山遍野的花朵呢？

（图片摄于笔者家乡--成都）

还记得 2000 年刚接触Flash开发，发现ActionScript可以通过面向对象的方式实现MIS系统应用，但Macromedia公司对Flash的定位仍然是面向设计人员，在面向开发方面的进展不得不用缓慢来形容，直到 2004 年在被Adobe公司收购的头一年才真正意义上推出了面向开发者的Flex 1.0 版本。笔者虽然对当时FLEX Store应用所振奋，但很快发现其第一个版本在大数据

量下低迷的表现。而Flash+AS和FLEX这两条路线，使得开发者很难去抉择。当我还在一边看着LUAR编写的《Flash MX 2004 ActionScript 2.0 与RIA应用程序开发》，LUAR本人却在博客上发表了《Flash未来未有来》的文章，作为中国Flash应用开发的领军人物，主动得出“Flash没有帮助开发者开发RIA”的结论。当然现在的Flash经过 10 年的苦心经营，应用已经非常广泛，特别是在一些Web页面的应用上面。

RIA技术选型

2008 年，中国人寿养老保险公司提出开发面向企业客户的养老金精算咨询系统（PACS），但由于养老金咨询中的方案数据采集与精算过程非常复杂，使用传统 B/S 架构很难实现。由于多年来的 RIA 开发经历，作为项目经理的我提出在此项目中尝试使用 RIA 技术进行开发，以到达客户友好性与易用性的目的。微软发布一年的 Silverlight 引起了项目组的注意，由于企业 B/S 开发的两大阵营，一个是 Java，另一个是.NET 平台。针对主流 RIA 技术笔者进行了对比，当时 Java 平台的 JavaFX 仍然处于 beta 阶段，而在此前笔者已经在一些项目中运用了 ExtJS 等通过 JS 实现的类 RIA 应用，但实际开发效率较低，而且脚本出错概率较高，JS 调试相对困难，企业架构要求的稳定性得不到很好的保障。因此，我们将目光集中在了.NET 框架下的 Silverlight 上，经过与微软中国的多次沟通交流，并进行了必要的性能测试。以下是通过实际测试当中的性能对比（对比基准为 Flash）：

测试一：后台运算性能测试，使用相同的素数查找算法

测试项目	Silverlight	Flash
查找 100 万以内	单线程:0 .423 秒 双线程: 0.267 秒	1.335 秒
查找 1000 万以内	单线程 9.833 秒 双线程: 6.144 秒	35.62 秒

在线测试及源代码下载地址：http://www.itwriting.com/primetest

测试二：UI 性能测试，使用小球碰撞模拟

测试项目	Silverlight2	Silverlight 3（cached）	Flash	Flash（cached）
16 个小球	238FPS	116FPS	64FPS	64FPS
32 个小球	133FPS	112FPS	54FPS	64FPS

64 个小球	62FPS	107FPS	41FPS	64FPS
128 个小球	35FPS	99FPS	18FPS	57FPS

在线测试及源代码下载地址：http://bubblemark.com

注：在最新公布的 Silverlight 4.0 中，官方指出 UI 线程性能提升了 5.5 倍，Worker 线程处理性能提高 870 倍。

这里要特别说明一个问题，Silverlight 与 Flash 本身的运行时都在客户端，以前我也在 Silverlight 发布会中提出 Silverlight 后台语言支持能力更强，但被很多人误认为是服务器端语言，其实 Silverlight 与 Flash 一样可以通过多种网络协议连接不同技术架构的服务端程序，但就客户端的运行时来说，Silverlight 是基于.NET 的 JIT Compiler 的，而且有 CLR 基础类库支持，所以与 Flash 前端运行时有很大差别。

Silverlight 实现RIA端到端的优势

谈到微软专门为 Silverlight 包装的 Core CLR 运行时环境 就不得不说到整个.NET 框架的优势，很多人将.NET CLR 与 JRE 做比较，两者都具有相当强大的基础类库支持，可以说为企业开发提供了基础，而 Silverlight 享受到了这样的基础类库，但为了减小运行时文件大小，提高用户下载安装体验，微软去除了无用的服务器端基类，保留了主要的客户端类库，下面我将结合 Core CLR 来说明 Silverlight 在企业级开发中的优劣势。

下表是笔者总结的在企业级应用开发方面 Silverlight 的优势：

企业应用特性	Silverlight	优势
数据访问能力	✓	✓ **Core CLR精简类库支持** ✓ 支持TCP\UDP\Socket通讯协议，支持WCF RIA services的直通访问方式，支持Web service、HttpHandler方式 ✓ 支持传输任意传输对象
性能	✓	✓ CoreCLR支持，运行编译的本地代码 ✓ **多线程支持、客户端缓存** ✓ 支持硬件加速
安全性	✓	✓ 通讯安全：支持数据与通讯的加密 ✓ 访问安全：基于透明安全模型，使得程序和数据都在“沙箱”中运行，实现高安全性
跨平台	✓	✓ 浏览器：IE, Safari, Opera, Firefox... ✓ OS：Windows, MAC, Linux，Windows Phone, Symbian
运行时特性	✓	✓ 无需部署.Net平台就可以执行.Net代码 ✓ Silverlight直接解释XAML语言供搜索引擎索引
设计时特性	✓	✓ **前后端语言的无缝集成** ✓ **丰富的控件与类库支持** ✓ 设计与编码分离 ✓ **前后端一体化调试** ✓ 原生快速原型工具

其中最为重要的一点就是前端Core CLR的支持，大家可以参见MSDN来获得Silverlight 4 基础类库的全部资料。

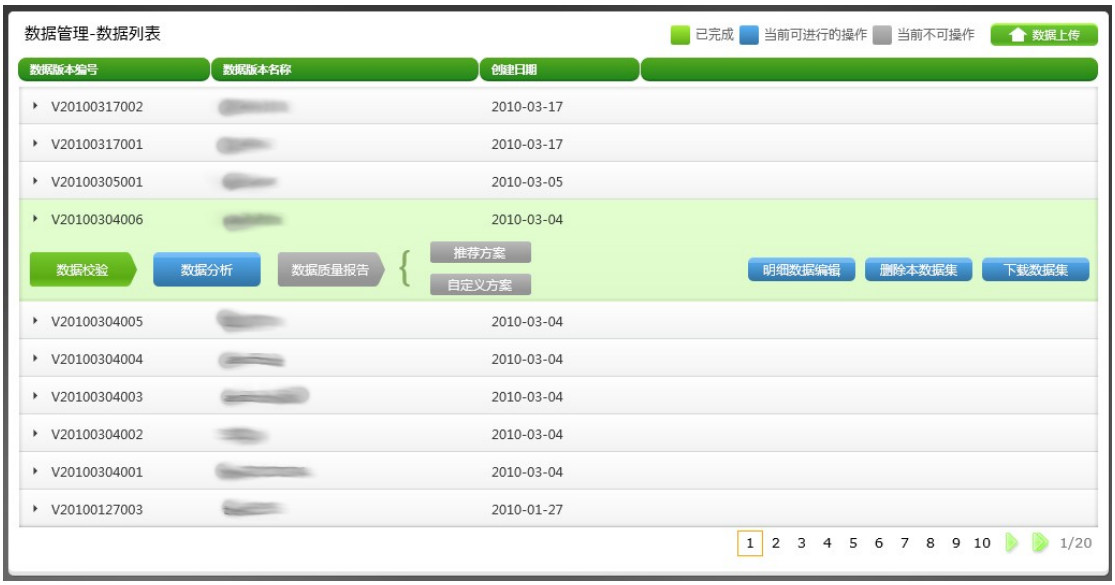
本文不会对 CoreCLR 做过多介绍，结合客户端架构将 Silverlight 主要基础类库总结如下：

层次	基类库支持程度
XAML 表现层	丰富的原生控件（System.Windows.Controls 超过 100 种控件类）与自定义控件库（ System.Windows.Controls. UserControl），支持全局样式修改与资源管理(System.Resources)，支持动画(System.Windows.Media.Animation)，效果（System.Windows.Media.Effects）
客户端控制逻辑层	.Net Framework 多种基本类库支持，包括多线程类库（System.Threading），安全类库（System.Security），反射类库（System.Reflection），Linq 类库（System.Linq），数据流操作类库（System.IO），XML 操作类库
通讯层	包括 ADO.Net 交互类库（System.Data.Services），WCF 交互类库（System.ServiceModel），网络类库（System.Net ）等等

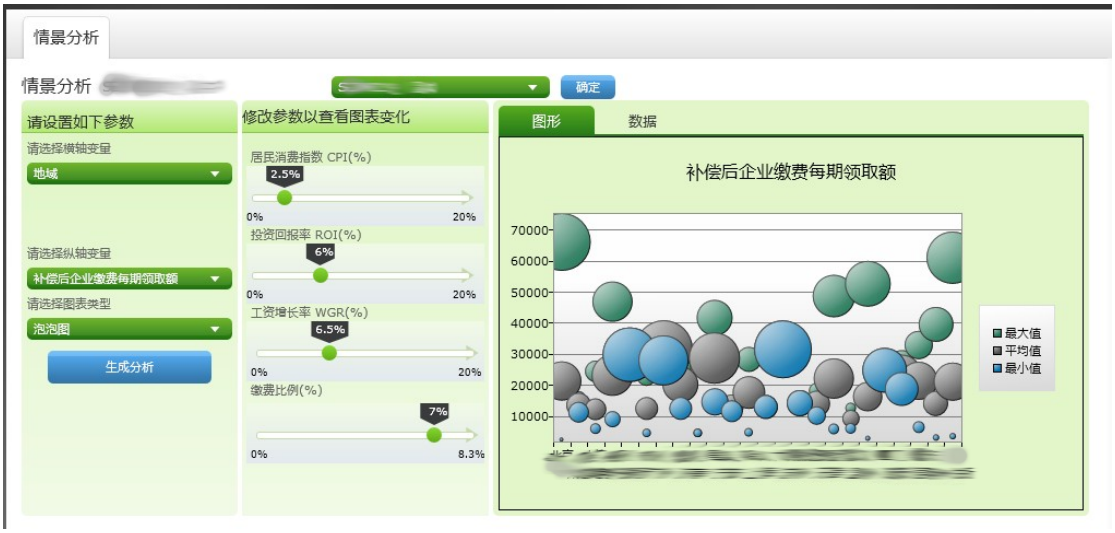
而使我们选择 Silverlight 的另外一点是，结合.NET 服务器端技术，可以在一个技术体系下构建端到端的 RIA 应用，在开发上由于语言的一致性可以共享类库，而同样的 IDE 环境能够提供方便的前后端一体化调试功能等功能。

通过Silverlight如何加强数据展现能力

要构建企业级应用，就必须要使得数据的展现和编辑更加方便直观，传统基于 HTML 标签语言的 web 应用，可以通过 input 和 Table 等来实现数据编辑与展现，但无法超出其能力之外，很多第三方的 Web 应用控件都是利用 JS 脚本来实现排序，拖拽等效果，但脚本语言毕竟是低效的，并且具有许多外部原因（如浏览器环境等）使得其稳定性不高，而 Silverlight 原生的控件在数据展现与编辑上面具有良好的表现。以 PACS 系统中的数据列表为例，使用 Silverlight 的 DataGrid 控件，原生支持列排序，列拖拽，也可以通过代码实现过滤，分页。由于我们在一些应用场景中需要对每一条记录都有对应的处理流程和功能按钮，因此使用 DataGrid 控件中定义行模板(DataGrid.RowDetailsTemplate)就可以实现自定义行样式的效果。而在<DataTemplate>< DataTemplate />中你可以加入任何 XAML 元素，使得 Silverlight 应用可以简单快速的实现以前难于实现的效果。



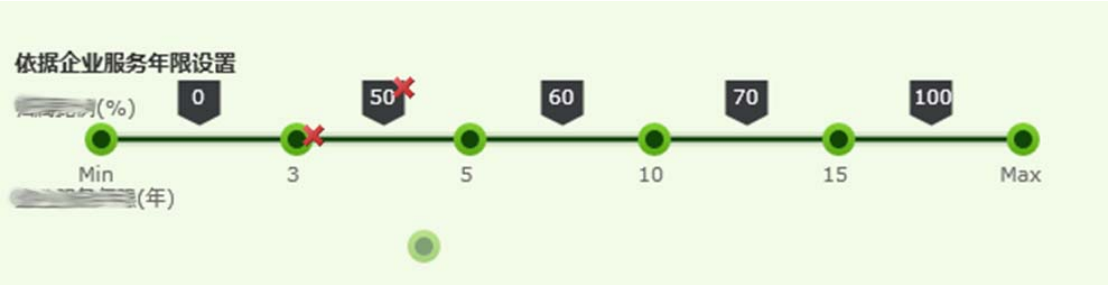
当然除了数据列表的展现方式，Silverlight 提供更多的图形化展现方式，比如原生的各种统计图表，以下是 Silverlight 的 Chart 控件应用，在拖动各种参数时，图表将以动画形式变换，而不需要写任何代码。



Silverlight Do It in Naturally

除数据展示以外，在传统 WEB 应用中，可能会遇到更多的数据录入上的限制，比如一种分段设置参数的应用场景，当我们要输入任意年龄段的对应参数时，传统 Web 应用没有好的解决方案，可能只会通过下面的方式来进行数据录入：

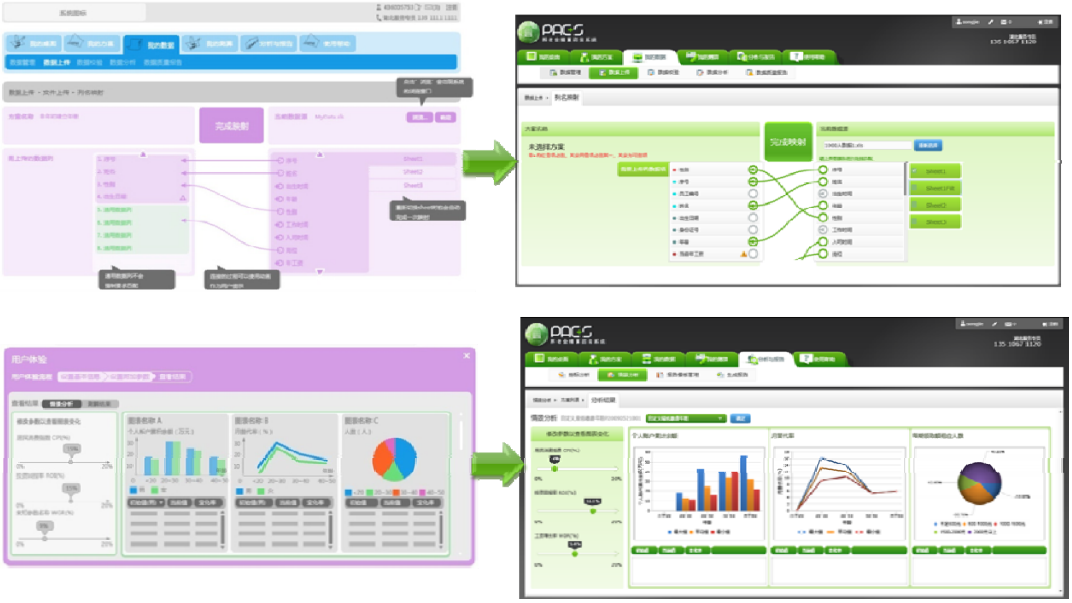
但如果使用 Silverlight 自定义控件就可以实现如下更加人性化的设置：



通过点击就可以简单实现在分段的中间插入分段，拖拽就能删除分段，这样的设计符合“do it naturally”用户体验设计原则。而且 Silverlight“一切皆为用户控件”的思路，使得开发者可以很简单的开发一个用户控件，而且这些用户控件可以被整个项目充分复用。

“体验先行”的开发流程

Silverlight 基于“体验先行”的设计，颠覆了传统业务需求模式，改变传统开发流程，是开发团队可以选择以界面而非功能为起点的开发模式。在 PACS 项目需求分析中，我们没有采用需求规格说明书面向功能的阐述方法，因为业务与 IT 的认识隔阂往往会导致项目失败。在 Silverlight 设计工具 Blend 中包含了原型开发工具 SketchFlow，通过简单的原型控件拖拽就可以快速开发系统原型，使得业务部门迅速理解界面与逻辑关系，避免了业务与 IT 在最终交付物认识上的差异。以下是 PACS 在需求分析过程中的原型与最终系统截图对比：



Silverlight 在 PACS 开发中带来了开发过程的革命，经过最终评估，采用快速原型的方式大大减少了需求变更，确保了业务满意度。

展望未来

展望 Silverlight 技术未来发展，在刚刚结束 MIX10 大会上，微软已经将 Silverlight 定位为未来 Windows Mobile 的主要开发技术，加之以前在跨浏览器，跨平台的策略，实际上已经实现“手机、PC、电视”三屏合一，可以很清晰地看到微软有使用 Silverlight 实现客户端应用大统一，从而全面收复 Web 应用失地的趋势。因此，未来微软肯定还会不断加大此项技术的认知度，而 Adobe 公司不会放弃通过占有率全面反击的机会。这也是我们作为企业开发者希望看到的竞争态势，那么 RIA 在经历 7 年之痒后，春天是否真的要来了，我们拭目以待！

（免责申明：本文仅代表作者个人观点）

作者简介：吴磊，就职于中国人寿，曾获得系统分析师、PMP、MCSE 等认证。2003 年毕业之后就职于中国人寿，一直从事开发工作，担任过软件开发人员、软件设计师、项目经理、软件架构师。

漫谈B端的沙箱技术

从语言学的角度上来说，允许代码无节制地使用全局变量，是最错误的选择之一。而更可怕的，就是一个变量“可能”成为全局的（在未知的时间与地点）。但是这两项，却伴随 JavaScript 这门语言成功地走到了现在。

也许是限于浏览器应用的规模，所以这一切还迟迟没有酿成灾难。在此之前，出现了两种解决方案。一种是 ECMA 在新的规范（Edition 5）中对此做出了限制，其中最重要的一条便是 eval()的使用变得不再随意和无度。而另一种方案，则是相对没有那么官僚与学术的，尽管也拥有一个同样学术的名字：沙箱。

沙箱（Sandbox）并不是一个新东西，即使对于 JavaScript 来说，也已经存在了相当长的时间。在 SpiderMonkey JS 的源代码中，就明确地将一个闭包描述为一个沙箱。这包含着许多潜在的信息：它有一个初始环境，可以被重置，可以被复制，以及最重要的，在它内部的所有操作，不会影响到外部。

当然事实上远非如此。JavaScript 里的闭包只是一个“貌似沙箱”的东西——仍然是出于 JavaScript 早期的语言规范的问题，闭包不得不允许那些“合法泄漏”给外部的东西。而对于这一切无法忍受的前端工程师们，开始寻求另外的解决之道，这其中相对较早的尝试，是基于 IFRAME 的实践。例如 dean.edwards 在 2006 年提出过的方案（注 1）：

```
a_frames.document.write(
  "<script>"+
  "var MSIE/*@cc_on =1@*/;"+ // sniff
  "parent.sandbox=MSIE?this:{eval:function(s){return eval(s)}}"+
  "<\/script>"
);
```

显然，由于在不同的 IFRAME 中运行着各自的 JavaScript 引擎实例，所以上述的方案也意味着沙箱是“引擎”这个级别的：在任何一个沙箱中的崩溃，将导致该引擎以及对应 IFRAME 崩溃。但——理论上说——不会影响整个浏览器。

问题是，这并不那么理想。往往的，引擎会导致整个浏览器锁在那里，例如用 alert()弹出一个对话框而又因为某种意外失去了焦点。又或者单个的 IFRAME 会导致全局的 CPU 被耗光，例如一个死循环。于是更加复杂的方案——在 JavaScritp 中包含一个完整的执行器——出现了。最有名的则是 Narrative JavaScript，它内建了一个执行器，用于逐行地解释执行 JavaScript 代码，这使得它可以控制所有的代码执行序列，或者随时重置整个执行引擎——如同一个沙箱所要做的那样。

这一切或者太过依赖于环境，又或者太过复杂，但都不乏追随者。例如 jsFiddle 这个项目（注 2）在“嵌入或装载”这样的路子上就已经有了不俗的成绩。但是，YUI 在新版本中却给出了它自己的选择：以更加明确的编程约定，来实现应用级别的沙箱。这包括一个非常简单的、新的 YUI 语法：

```
YUI().use('dom-base', function(Y) {
  // Y是一个新的沙箱
});
```

在'dom-base'位置上，可以是 1 到 N 个字符串，表明一个需要在沙箱中装载的模块列表。这可以是沙箱的初始列表，或者后续的 callback 函数(亦即是用户代码)所需依赖的模块列表。在这种实现方案中，YUI 为每个沙箱维护各自的装载模块列表和上下文环境中的变量、成员。但是出于 JavaScript 语言自己的局限，这个沙箱依然是相当脆弱的。例如下一示例中沙箱内的代码就会污染到全局：

```
YUI().use('', function(Y) {
  abc = 1234;  //<--这里可能导致一个全局变量'abc'被隐式地声明
});
```

同样，在上述的沙箱里也可以使用类似 window、document 等全局变量、修改它们的成员或无限制地调用方法（例如使用 setTimeout()来创建时钟）。所以 YUI 的沙箱事实上是靠“规约”来约束的，而不是真正意义上的沙箱。当然，这也意味着，如果用户能按照规约来处理沙箱内的代码，那么也就能自由地享用它带来的便利：安全、移植和有效的隔离副作用。

而我们再穷究其根底，YUI 沙箱的实质不过是一行:

```
// code from yui.js
//  - mod.fn(this, name)
mod.entryFunc(sandbox, modName);
```

其实际含义是：

- mod ：沙箱当前装载的模块；
- entryFunc：上述模块的入口函数；
- sandbox ：当前的沙箱的实例，即 YUI()返回值；
- modName：模块名

除了依赖关系（以及可能需要的异步加载）之外，YUI 沙箱环境仅是用下面的代码来简单地调用 callback 函数：

```
callback(Y, response);
```

然而这些需求的实现并不那么复杂。首先，我们设定数据结构 mod 为一个对象：

```
{ name:modName, fn: entryFunc, req: [], use: [] }
```

则一个环境对象 env，将包括多个 mod(将它们处理成对象而非数组，主要是便于使用名字来索引模块)和以及对它们进行管理操作的方法：

```
{ mods:{}, used:{}, add:..., use:...}
```

最后，所谓一个沙箱 sandbox，就是上述环境对象的一个实例，并在初始时 sandbox.mods 与 sandbox.used 为空。由此简单的实现为：

```
/**
 * tiny sandbox framework
 * mirror from YUI3 by aimingoo.
**/
function Sandbox() {
  if (!(this instanceof arguments.callee)) return new arguments.callee();
  this.mods = this.mods || {};
  this.used = {};
}

Sandbox.prototype = {
  add: function(modName, entryFunc, reqArr, useArr) {
    this.mods[modName] = { fn: entryFunc, req: reqArr, use: useArr }
  },

  use: function() {
    var mods = [].slice.call(arguments, 0); // 0..length-2 is modNames
    var callback = mods.pop();        // length-1 is callback
    var recursive_load = function(name, mod) {
      if (!this.used[name] && (mod=this.mods[name])) {
        mod.req.forEach(recursive_load, this);
```

```
        mod.fn(this, name);
        mod.use.forEach(recursive_load, this);
        this.used[name] = true;
      }
    }
    mods.forEach(recursive_load, this);
    callback(this);
  }
}
```

现在我们来尝试一个与 YUI 类似的语法风格：

```
Sandbox().use('', function(){
   alert('user code.');
});
```

或者，先向整个 Sandbox 环境注册一些模块（在真实的框架实现中，这一步可能是通过框架的装载器来初始化）：

```
// for test, entry of mods
f1 = function() { alert('f1') };
f2 = function() { alert('f2') };
f3 = function() { alert('f3') };

// mods for global/common env.
Sandbox.prototype.mods = {
  'core': { fn: f1, req: [], use: [] },
  'oo':   { fn: f2, req: ['core'], use: ['xml'] },
  'xml':  { fn: f3, req: [], use: [] }
}
```

然后再尝试在一个沙箱实例中运行代码：

```
// f1 -> f2 -> f3 -> user code
Sandbox().use('oo', function(){
   alert('user code.');
});
```

其实即便是上述代码中用于处理模块依赖的逻辑，也并不是什么“神奇的”代码或者技巧。除开这些，这样的沙箱隔离泄露的能力还抵不过一个嵌入式 DSL 语言。而后者所应用的技巧很简单，看不出什么花招（注 3）：

```
with (YUI()) this.eval("... mod_context ... ");
```

这样一来，在 mod_context 里的代码就只会在 YUI()的一个实例中造成污染了。当然，仍然

是源于 JavaScript 的限制，我们还是无法避免一个变量泄露到全局——除非，我们回到 js in js 这个项目（注 4），真的在环境中重新初始化一个 js 引擎。

从这一意义上来说，引擎级别的沙箱与操作系统的进程一样，带来的是终级的解决方案，所以 Chrome、IE 等等主流浏览器纷纷有了“独立进程”模式。而在这样的背景之下，试图用“框架内置沙箱”来改善 ECMAScript ed3 中一些设计疏失的种种努力，不过是一张张空头的支票罢了。

甚至，用这本支票签完单也未必有人会收的。

备注

注1：http://dean.edwards.name/weblog/2006/11/sandbox/
注2：http://jsfiddle.net/
注3：http://blog.csdn.net/aimingoo/archive/2009/09/08/4532496.aspx
注4：http://mxr.mozilla.org/mozilla/source/js/narcissus/

作者简介：周爱民，国内软件开发界资深软件工程师，架构师。有十余年的软件开发、项目管理、团队建设的经验，历任部门经理、区域总经理、高级软件工程师、平台架构师等职。现任支付宝（中国）公司业务架构师。

评论：前端的HTML5 时代来临

今年 MIX2010 大会上微软的工程师在介绍 IE9 时，从前端技术的角度把互联网的发展分为三个阶段：第一阶段是 Web 1.0 的以内容为主的网络，前端主流技术是 HTML 和 CSS，第二阶段是 Web 2.0 的 Ajax 应用，热门技术是 Javascript/DOM/异步数据请求，第三阶段是即将迎来的 HTML5 时代，亮点是富图形和富媒体内容（Graphically-Rich and Media-Rich）。我也认同今年前端技术将进入一个崭新的时代，至少已经开启了这扇门。

Web 2.0 时代把大量服务从桌面转移到网络上，B/S架框的应用迅速流行起来，这个时期Ajax和Javascript框架得到广泛的应用。网站成为提供Web服务的主要形式，但随着智能手机和 3G 网络的普及，手机应用和手机上的Web应用将会成为新的产品形态。原有的互联网产品需要对各种形式的终端提供友好的展现形式。比如豆瓣电台(douban.fm)目前就有网页版、iPhone 版、Android版、不久还会看到HTML 5 版和桌面版。网络逐渐变成一个传输和存储数据的透明仓库，向不同形式的产品终端输送数据。而且这个数据内容将会非常丰富，这就是微软眼中的富图形和富媒体内容，所以，IE9 必须、也不得不支持SVG和HTML5 的Video和Audio。

苹果推出 iPad 无疑又是一场革命，在拥有更大的多点触摸屏幕的终端上跑智能手机系统，把硬件成本降到一个足够低的程度，我想 iPad 和类似的平板电脑将会流行起来。同时也意味着，产品前端又增加一种形态。

PC 上的 Web 应用受浏览器历史原因的束缚太大，在国内 IE6 等低端浏览器目前仍占据绝大多数的市场份额。因此这些不称职的平台需要靠第三方插件来弥补先天的不足，像 Flash/Flex、Silverlight、JavaFx 等，包括 Google Chrome，本质上都是为了弥补浏览器的缺陷，当浏览器足够强了也就不需要这些插件了。智能手机出现的比较晚，所以完全没有历史包袱。现在的智能手机系统 iPhone OS 和 Android 加起来就占了 90%以上的市场份额，默认的浏览器都是基于 Webkit 内核的 HTML 浏览器，对 HTML 5 和 CSS3 的支持比较好。这让 HTML5 和 CSS3 有了最佳实践的机会。产品形态向不同终端转变的成本因此大大降低了。这是 HTML5 技术升温的原因。

比如iPhone上的Mobile Safari浏览器支持HTML5 的Canvas/Canvas2D、Video、Audio、Geolocation、Local Storage/Session Storage、Application Cache、Web SQL Database等，CSS3

的 3D/2D动画、RGBA、圆角、阴影和渐变等主要功能都支持。用你的手机浏览器访问 http://html5test.com，可以测试一下都具体支持哪些。利用这些特性加上传统的 Javascript/HTML/CSS就可以打造一个很强悍的Web应用。Paul Graham在 2005 年Web2.0 时代来临时曾说“Javascript now works”，那现在可以说“HTML5 now works”了。HTML5 发挥作用同时，也意味着HTML5&CSS3 将成为前端开发者的必修技能。

HTML5 和 Flash 之间的讨论前段时间很热。插件技术本身是在原生技术满足不了的情况下才会使用。在学习路线上，HTML/CSS/Javascript 到 HTML5/CSS3 将是前端开发者的必修课，插件技术永远是选修课。Flash CS5 将支持导出 HTML5 的 Canvas，可见 Adobe 毕竟是一家软件公司，它会想办法实现不同技术之间转换和包容，有人说也许几年后 Flash 会成为 Canvas 开发工具，我们只需坐享其成。像从 Flash 到 Flex 再到 AIR 的发展过程中可见一斑。做为前端开发者来说，一定要身兼多技，但务必要先学好必修课。很多公司会设置一些专一性技术的职位，比如 Flash 开发工程师，这会误导一些人只学 Flash，对日后的职业发展很不利。

套句老话“形势喜人，形势逼人”，前端技术越来越丰富，前端开发者的担子越重，越多东西要学。总之，玩起来吧！

作者介绍：张克军，前雅虎中国资深前端工程师，现豆瓣前端工程师，主要负责产品的前端架构和开发。热衷于前端领域的技术推广，是国内最活跃的前端布道者之一。个人网站：http://hikejun.com。

推荐编辑 | 架构社区编辑 侯伯薇

大家好，我是侯伯薇，很高兴能够在这里与大家见面。看到我的相片可能有人会有些失望和意外：本以为是位娇小的 MM，却没想到看到的是个东北大汉。

蓦然发现，自己离开大学校园已经是十年前的事儿了。在这十年之中，我做过培训，当过老师，但更多的时间还是在比较安分地做程序员，以编写各种代码为乐。既经历过国内项目的灵活多变，也曾在对日项目中用严格的规范和严谨仔细的作风来锤炼自己。值得庆幸的是，这么多年来自己一直在提高，在进步。

第一次访问 InfoQ 是因为看到其中一本非常不错的迷你书，之后就开始逐渐开始关注这个与众不同的技术站点。当泰稳主编发出征集令的时候，我就想自己是否也能够加入其中，为我们这些热爱代码的人做些事情呢？很荣幸，也很幸运的是，我的这个想法成为了现实，我成功加入了这个大家庭，成为架构社区的一名编辑。

回顾一下在这个充满激情和智慧的团队中的日子，感慨良多。当初刚进入的时候，我对自己的翻译水平还颇为自负，然而，在泰稳、郑柯、晓刚等各位前辈的指点下，却发现自己在这方面还存在很多的不足，尽管在各位的帮助下，已有少许进步，但还是存在很大的提升空间；而在技术知识方面，尤其是在敏捷、架构方面，经过与大家的思想的激情碰撞，也获得了很多新的认识，从大家那里学习到了很多，弥补了自己在很多方面知识的空白。

除了知识方面的进步，在做事方面，加入 InfoQ 之后我也有了不少的改变。之前虽然一直喜欢思考，但是很少在网络上与大家交流和分享。加入这个团队之后，让我更真切地感受到网络这个世界的力量。一方面网站上的内容给了我很多思考的题材，让我的思维更加活跃；另一方面，我也越来越愿意把自己的一些想法拿出来，放在自己的博客上与大家分享和交流，在这个过程中受益颇丰。此外，各位编辑严谨的态度也让我做事更加仔细，因为他们让我知道，做为大家服务的工作，就是应该在质量上精益求精，而不能敷衍了事。

总之，感谢 InfoQ，感谢各位编辑，感谢广大读者，是你们的帮助，你们的理解，你们的批评和鼓励让我的努力有了更有效的成果。在以后的日子里，我会一如既往地走下去，也希望能够与大家更多地交流和分享。

看了我的经历，有激情、有智慧的你还在等什么呢？快来加入到 InfoQ 中吧，这将是你不悔的选择！

封面植物

瑞丽山龙眼

形态特征：常绿乔木，高 5—10 米；小枝几无毛，仅在叶着生处被锈色绒毛；芽被锈色绒毛。叶薄革质，无毛，倒卵状长圆形或长圆形，稀倒披针形，长 6—13 厘米，宽 2.5 一 5 厘米，先端急尖，或圆钝、具短骤渐尖头，全缘或具浅锯齿，齿端具腺体，侧脉 6—8 对；叶柄长 2—5(一 10)毫米。总状花序多生于小枝落叶腋部，长 7—12 厘米；苞片显著，叶状，披针形至长圆形，长 5—10(一 20)毫米，宽 1.5—3(一 5)毫米，微锐尖；小苞片长 1—2 毫米；宽约 0．5 毫米；花黄色，长约 13 毫米。果序上常有 2—3(一 5)枚叶状苞片。果倒卵圆形或近球形，顶端具细尖。

特性：瑞丽山龙眼分布区主要位于西部南亚热带季风常绿阔叶地带西端，小山与峡谷相间的地方。年平均温 14.9—20℃，极端最低温可达 0℃左右，年降水量 1 250—2150 毫米，分配不均匀，多集中在 4—9 月，干湿季交迭分明。立地的土壤为在花岗岩、砂页岩风化而成的山地红壤和黄壤。生长在常绿阔叶林中，同生的优势种有元江锥 Castanopsis orthacantha Franch 、小果锥 C. fieuryi Hich et Camus 、滇青冈 Cyclobalanopsis glaucoides Schott．等。生于山地密林或疏林中。花期 8--9 月，果期 10--12 月。

保护价值：瑞丽山龙眼是我国特有种。其苞片大而显著，在属内为独特类型，有一定的科研价值。

1kg.org 多背一公斤

爱自然 | 更爱孩子

架构师 5 月刊

每月 8 日出版

本期主编：张逸

总编：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：张逸，InfoQ 中文站.NET 社区编辑

张逸，软件架构师，敏捷教练，微软最有价值专家，InfoQ .NET社区常务编辑，msup资深讲师。主要专注于软件架构、设计模式、领域驱动设计和敏捷开发，以及基于.NET平台的企业级系统应用开发和分布式开发。曾先后为可口可乐、摩托罗拉、HP、京东方、尼桑、摩洛哥电信、盛大、第九城市、CA、携程等企业提供设计、开发、咨询与培训服务。他的著译作包括《软件设计精要与模式》、《WCF服务编程》。可以通过http://www.agiledon.com访问他的个人网站。

InfoQ中文站
www.infoq.com/cn

《架构师》月刊由InfoQ中文站出品。

所有内容版权均属C4Media Inc.所有，未经许可不得转载。

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ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