时刻关注企业软件开发领域的变化与创新

# 架构师

9月 ARCHITECT

采访Lisp之父
John Mccarthy

SQL SERVER的
未来之路

软件开发7大浪费

TechLead三重人格

并发与不可变性

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语 | Editor's letter

# 不要生活在真空中

很多时候，我常常听到那些有志在技术上有所成就的工程师表示“宁愿和机器打交道也不愿意和人打交道”。而发出这种论调的，尤以刚刚踏入工作岗位的工程师居多。的确，和人相比，机器的可控性要高得多，而且只要你掌握了足够的技能，它基本上就会任你摆布，唯命是从。此时，对机器而言，你就是上帝。对此，Brooks 在《人月神话》中这样写道：“我想这种快乐是上帝创造世界的折射，一种呈现在每片独特、崭新的树叶和雪花上的喜悦”。相反，与人打交道就没有这么轻松了。这其中不可避免的要有说服、妥协、耍心机等等一系列的活动，用一个词来形容就是“累”。

可是，有一点被这些工程师们忽略了，那就是如何判断你在技术上有所成就了？或者更通俗的说，你怎么知道自己已经是牛人了？人是一种存在感很强的动物，这种存在感在和他人进行比较或者得到他人的认可时会大大加强。说到底，你必须和他人接触才能得到问题的答案。在这一过程中，和人打交道自然无法避免。结果，“只和机器打交道”只不过是自己的一厢情愿罢了。从另一方面来讲，技术人员执着于技术，无非是想通过自己最擅长的手段去体现个人的价值。但是，仔细想想便会明白，所谓价值，它是不会凭空体现的，它必须要有个载体，这个载体就是需求。没有需求，自然就没有价值。而需求本身则是非常个性化的东西，为了能展现出自己的价值，你就得千方百计的去了解需求，并最终将其解决。到头来，你会发现你已经和不少人打了交道。

既然不可能与世隔绝，为何不干脆直面现实？既然横竖要做，为何不开开心心的做？天堂还是地狱，只在一念之间。

以上文字不过是笔者毕业以来的一些个人感悟罢了。而我之所以啰啰嗦嗦写出这些，只不过是想说明，要想成为一名优秀的架构师，沟通是你必须学会的技能。而且它也是你最重要的技能之一。为什么这么说？因为在我看来，你的架构合不合理，有无价值都得由需求说了算。只有在搞清楚需求的前提之下，才有可能去构建出优秀的架构；否则，要是需求弄错，即便你的架构在你看来是完美无缺，但在客户那里，它也是一文不值。除了和客户沟通，将你的设计意图很好地向你的团队传达同样也很重要。换句话说，你得和你的团队进行沟通。软件

行业发展到今天这个程度，单凭一己之力去构建一个大规模系统，几乎已经成了不可能的任务。在每个成功软件的背后，都有一个成功的团队。“默契”是这些团队的共同特征，而这其中，架构师和团队成员之间的通畅沟通发挥着至关重要的作用。缺乏沟通，就好比拥有优秀的大脑和强健的体魄，但却没有发达的神经网络，到时指东打西，南辕北辙也不是什么新鲜事。

平心而论，但凡一个人，只要他还有理智，基本上不存在沟通的问题。而大多数令技术人员不愉快的沟通，也是因为技术人员只注重个人技能的修炼，而不费心沟通技巧的结果。再加上和人打交道的时间本来就少，久而久之，沟通技巧便越来越差。一旦面对他人，其效果可想而知。然而，作为社会性的动物，人不是生活在真空中的。要想达到自己的目的，实现自己的理想和抱负，你必须学会和他人合作，而所有这些的前提都是有效的沟通。

作为本期杂志的开篇语，罗列沟通技巧显然不是本文的目的，况且 InfoQ 中文站上的敏捷社区中已有不少文章对此已有专门论述，笔者在此就不再作学舌之语。最后，在各位阅读本期杂志的正文之前，容笔者再唠叨一句：“功夫在诗外”。

本期主编：胡 键

# 目　录

## [新品发布]



## [架构师大家谈]



## [封面植物]

人物专访 | Interview

# Guy Steele采访Lisp之父John McCarthy

在OOPSLA 2008 会议上，Guy Steele 在观众面前对 John McCarthy 作了一次电话采访。Guy Steele 跟他天南海北的阔论起来，试图发现这一语言如何在五十年代形成与其后续演变的一些细节。

**InfoQ：我是 Guy Steele 同时也代表 Alan Kay，您已经有一张问题列表了，P.Gabriel 转给你的而且我相信你也浏览过了。我打算把其中一些问题过一遍，当然不一定要按照原来的顺序，让它自由连贯会更好。同时我还准备了一些你没有看到的问题，比较简短，我想可以把它留到这次谈话的最后来进行。那不如让我们从这个列表上第 4 个问题开始：在 Lisp 的早期概念与早期开发中，谁对你的帮助最大？**

John：我也愿意谈一谈这个问题因为我可能会遗忘一些人。用 Lisp 结构来表达数据和表达程序这一思想我是从 Newell 和 Simon 那里借鉴的。他们并没有意识到这会对我有帮助，但无论如何，这是从他们那里得来的。而函数式编程这一思想我又是从 John Backus 的 FORTRAN 那里得来。...作为函数这一概念来自于 Herbert Gelernter 和 Carl Gerberich。当时我正在指导他们工作于一项 FORTRAN-Lisp 处理语言，我当时提倡用它来实现 Minsky 的几何机器。 然后 Steve Russell 提出了使用我的 eval 作为解释器，于是初始 AI 项目的 6 个数学研究生和两个程序员，Russell 和...完成了最初的开发工作。 Phyllis Fox 编写了第一版手册，而来自 IBM 的 Nathaniel Rochester 也作出了一部分最初的编程工作。这些就是最初参与的人员，也许还有一些我忘记了。

**InfoQ：现在让我们来看列表上的第 3 个问题。如果时光能够倒转，你会为 Lisp 加入什么？你将会删去哪些部分，而哪些部分又需要修改呢？**

John：人们已经加入了一些英语化的成分，至少从语法上讲我认为这并不符合 Lisp 的精神。我对这一内容并不存在任何反对，实际上整体而言我对加入到 Lisp 的东西都不存在任何反对，因为你完全可以不用它们。就目前而言，我并没有什么特别的追求将某些特定的东西加入到 Lisp。我想要加入一些直接逻辑，但我找不到很好的实现方式。

**InfoQ：经年以来又不少的项目试图将 Lisp 与 Prolog 以某种方法组合起来，以加入逻辑。你**

**有没有注意过它们呢？**

John：我从未注意过它们。我不反对这一想法。

**InfoQ：能否举一例说明一下 Lisp 里面什么样的英语化特性是你所不愿使用的呢？**

John：在我看来一些关于对象的成分是以一种英语化的方式加入的-我已经忘了。你肯定应当记得。

**InfoQ：也许你想到的是 LOOP。我只是想知道你用“英语化”来指代什么。**

John：那可能就是刚才我想到的吧。

**InfoQ :现在我们上到第 2 号问题 :一个好的编程语言与一个不太好的编程语言区分在哪点？**

John：对此我没有一个一般性的答案。我有一个主张，现在来想的话，Lisp 部分具有这一属性 ,而且我也许应当主张将它显式地加入到 Lisp 以及其它的编程语言当中。而这一主张就是，一个语言应该能够使用其自己的抽象语法。抽象语法是我在 1962 年的一篇论文中提出来的一个概念。现在要过多的讲它就太费力了，但它的基本思想是，在抽象语法看来，无论“加”是被表达成 a+b 还是(+ a b)，就像 Lisp 所做的那样，还是像哥德尔数所做的那样，哥德尔数 a（其值为 19） x 哥德尔数 b（其值为 19），没有任何区别。你仍然需要辨别辨一个表达式是不是求和，获得 2 个加数并组成一个加法表达式的功能。这就是抽象语法，我相信一个编程语言应当有其自身的抽象语法函数作为组成部分。

**InfoQ：当然，你在 Lisp 中加入抽象语法的一个重要目的之一就是让它可以编写自己的分析器。与此同时，你还有其它的目的吗？**

John：是的，任何对同一语言的程序进行操作的程序，不管是分析器或是编译器，还是出于某种目的判断一个程序是否合格，都会牵涉到该语言的抽象语法。如果你看看普通的编程语言如 Java 或其同类，如果你要写一个 Java 程序来处理 Java 程序，它必须要扫描并判断一个事物是否为“加”，它必须要寻找“+”记号并判断它是否运用了其先例的规则。

**InfoQ：让我们继续，看看列表上第一个问题：编程在 AI 研究中扮演一个什么样的角色，它应该是一个什么样的角色？**

John：这里有两件事，一是程序员们所编写的做 AI 的程序，另一个是他们编写这些程序所使用的程序。目前我并没有什么特别的东西可讲，后者十分的重要，而这也正是 Lisp 显得方便的地方。考虑抽象语法与 Lisp，Lisp 虽说并没有显示的抽象语法，但至少，从 Lisp 的语法很容易容易得到抽象语法。而判断出一个事物是否是“加”也并不困难。因此可以说它几乎

拥有了自己的抽象语法。

**InfoQ：我见过其它编程语言所定义的抽象语法的数据结构，比如 Ada。看起来还是少用它们或者少学它们为妙，因为它牵涉了许许多多非常特殊数据结构，每一个对应一种抽象语法，而不是一般性的使用一个表达式来表示所有的抽象语法。你对使用统一的表达式表示的利与弊有什么评价吗？**

John：我没有研究过他们是如何使用的，但可以肯定的是，有不同的方式来给语言定义抽象语法。首先，对抽象语法的使用-在我的论文大力提倡它以后，被用于定义了 PL 1 语法的抽象，这是由 IBM Vienna 小组所做出的，但我认为他们的工作并未被这个大公司的其它成员所认可。

**InfoQ：这是指的维也纳定义语言(Vienna Definition Language)，不是吗？与之相关的一些书都是 30 年前写就的了？**

John：比这更早，1962 年的事儿了。

**InfoQ：那个时候我还太小。我准备暂时抛开你面前的这个问题列表，问问大家提出来的一些相对简短的问题。其中有些人在 1958 年曾经参与过 Lisp 的一些相关的早期活动，希望你能让他们满意。第一个问题是，特性表(property list)这一想法是从何而来的？**

John：这一想法，某种程度上讲是对象这一概念的实现，实际上早于 Lisp 就在我脑海里形成了。我在 programs with common sense 这篇论文中就提到了，你能给对象加上某种指代并作为其结构的一部分。我记得当时以 1776 这个数字为例，除了其算术结构之外，你能记住它还因为它是美国独立革命的日期。这就是这些对象，并且我给出了好几个例子，在 Lisp 当中，我想到了使用这个特性表。它们的使用已完全超出了我的想像。

**InfoQ：你原本是怎么想的呢，又发生了怎样的出入呢？**

John：首先，我的想像——因为它并没有和任何提出的实现联系起来，某种程度上讲是比较模糊的，我也记不得了。现在，你们也试试假设五十年之后，人们问你："你当时是怎么想的呢？" 我想对于这个问题恐怕我没有什么更好的答案了。

**InfoQ：你的评述引起我了两个问题。一个是，如 John 所说，人们会在 50 年之后问这个房间里的人："你当时在想什么呢？"。我希望你们能够从历史学家那里学到，把东西记下来，或者你可以保存你的电子邮件，甚至是将它打印出来以保存得更长久。实际上这就引出了一个列表上的问题：John，你在 standord 时期的笔记和手写材料还在吗？你仍然还保留得有 MIT 时期的笔记吗？在你致力于 Lisp 的时候你是否有记录某种形式的实验笔记的习惯？**

John：它们大多是陈旧的文件，一部分得到了保存。当 Herbert Stoyan 来美国的时候，他对 Lisp 的历史产生了浓厚的兴趣，他去了 MIT 和 Dartmouth，翻遍了属于我的文件柜，这些柜子还在那里。我不是很清楚了，也许它们都被扔掉了。我想从现在到未来的 50 年间，情况会变得好很多了，因为所有的东西都可以是电脑文件，有希望被保存下来，你只需要编写正确的程序就能把信息挖掘出来。

**InfoQ：我希望 Brewster Kahle 的因特网档案库能够有助于填补这个空缺，但如果你真想把它们保存下来的话，我仍然建议人们打印出纸质文件。我们将会见证这是否能有成效。回到特性表这个问题，我注意到特性表使用键值对来记录信息，这似乎是一个重要的编程指南。关联表(association list)同样提供了一个有着键值的对应。这多用于变量查找与环境。十分有趣的是，特性表与关联表，尽管都是服务于同一个相同的目的，但在控制台来说却有着两个截然不同的结构。这是 Lisp 当中的一个历史巧合，还是重复发明的轮子呢？发生了什么事？**

John：我想这是个历史巧合。

**InfoQ：两个不同的程序员，又相互都比较粗枝大叶——似乎说不通。**

John：我想它们两个都是我发明的，不管那个在前哪个在后，我在发明时并没有考虑到我之前所做的。

**InfoQ：这是一个很小的点，但却引起我的好奇，你在 1958 年之前有什么样的实际编程经验，在你处理 Lisp 之前？**

John：我编写过一些 FORTRAN 程序。特别地，我为我将要编写的国际象棋程序编写了合法行棋路线程序，后来由[Alan] Kotok 和[Elwyn] Berlekamp 等人接手过去，成为了 MIT 国际象棋程序的基础。我实际上并未实现策略部分，因为我想先有一个宏观的计划随后再去实现。我没有完成，所以他们最后实现了一个非常直接的程序，但却实现了一个非常好的政治目的：Hubert Dreyfus 声称没有国际象棋程序能打败他，实际上 Dreyfus 作为国际象棋选手也比较拙劣啦，因此一个非常原始的程序就击败了他。

**InfoQ：当你开始 Lisp 的工作时，特别是从 1958 年的 9 月到 11 月这段时间，你每天投入多少时间在 Lisp 上呢？你是 100%的投入呢，还是说在进行其它项目的同时投入部分的时间在它上面？**

John：我不能这么说，但实际上，当那个夏天的末尾我在 IBM 的时候，我就开始这项工作了，我决定为代数表达式的微分写一个程序。我所注意到的是，我可以编写条件表达式，在我为国际象棋程序编写的合法行棋线路程序中就曾经用过，它们能够以一种非常直接的形式包含微分文本所给定的规则。代数表达式的微分是启发 Lisp 程序函数式形式的首要例子。

**InfoQ：在 50 年代末期，你也参与了 ALGOL 的设计工作。我们想知道你是否考虑过以 ALGOL 58 作为基础来设计 Lisp 而不是使用 FORTRAN，或者 Lisp 是否因为你对 ALGOL 58 的参与而带来的间接影响？**

John：我并未参加最初的 ALGOL 58 会议，所以对于基本结构的建立没有包含我的任何贡献。我也没想过要对其产生重要的影响，因为他们更倾向于创造一个能够作为数字运算的标准语言。我很赞同应该有这样的语言的想法。到 ALGOL 60 的时候我是委员会的成员之一，而整个委员会更加雄心勃勃，我提交了条件表达式和布尔表达式，尽管它们的实现可是糟透了。在 ALGOL 60 中的 a AND b 运算，你必须先对 a 和 b 都进行求值然后才能运算结果，而这是不正确的，如果写成条件表达式，当 a 的求值为假时，就根本不需要再对 b 求值了。

**InfoQ：有时候这是处于正确性的原因。实际上，我们现在对它们的使用已经是习惯了，或者从更早开始，1965 年吧。以这种条件的方式对逻辑连接求值对于程序的正确性来说是十分重要的，有鉴于此我们就习惯这种惯用的方式。这样的话，如果 a 不通过而去对 b 求值的话，就可能不正确了。**

John：是的。

**InfoQ：你第一次接受递归风格的编程是什么时候？**

John：一方面，我从 Newell 和 Simon 那里学习而来，另一方面，来自于递归的函数风格，那是在 1958 年的夏天，我做微分程序的时候。而 Newell 与 Simon 在 1956 年来 Dartmouth 参观夏季 AI 项目的时候曾经说到过它。那时他们已经完成了 IPL 或者第一版的 IPL。

**InfoQ：这一将计算机程序以递归的方式组织起来的想法也是随着 Lisp 本身逐步进行的吗？**

John：是的。但是我的意思是，IPL 某些方面做得很不好。在当时看来似乎很糟糕，所以我并未想过照搬它们。

**InfoQ：你所不想照搬的不好的方面指的是什么呢？**

John：它不是可组合的，也就是它并非基于可以组合的功能。它是基于子程序的名字，比如 J45，别人告诉我说，这是依据老式的 JOHNNIAC 装载程序来的。

**InfoQ：我看过一些 IPL-V 的代码，看起来它确实是非常声明式和命令式的语言，而命令又夹杂着十分神秘的名字和数字，这让我想起 Lisp 机器上的汇编语言。你现在相信 S 表达式或者以 S 表达式编程不是一个好主意。对此你有改变看法吗，如果是的话，是什么时候？**

John :我一开始的想法是把它做得像 Fortran 一样，这样你可以使用 M 表达式编程。接下来，

当 Steve Russell 吸取我的 eval 之后，他告诉我可以直接使用 S 表达式编程。这是一种创新，而作为一个保守派，我当时是倾向于抵制创新的，所以我用了好长时间才慢慢适应了它。

**InfoQ：听你这样说很有趣，因为你作为 Lisp 的发明者，你在很多方面都被认为是一个出色的创新者，所以你将自己描述为保守派确实令我们有一点吃惊。**

John：每个人对它当时所做的事情都是保守的。

**InfoQ：我现在准备回到你面前的问题列表。从第 5 个问题开始，然后向前进行。编程语言应当为理论家的推理而设计呢，还是为实践者构建系统而设计呢？或者对于这两种目标各有语言呢？不同的语言设计者的技术集是不同的吗？**

John：我的观点是一个语言应当从一个抽象语法的层面上来进行设计，并且它也许应当有好几种具体的语法：一种是易于编写并且非常缩略的；另一种是非常好看很迷人的，但最终，计算机将会产生它；另一种是使得计算机易于操控的。还有其它的情况，让计算机证明事物应当是简单的，而且它们应当基于同样的抽象语法。对于你的问题上，抽象语法就是理论者将要使用的，而一个或多个具体语法则是实践者要使用的。

**InfoQ：这实际上让我想起了 ALGOL 项目早期的机器可读的语法与用于发布中的参考语法之间的区别，这一区别是从你这里来的吗？**

John：没有。但他们没能够走远。我没有在 ALGOL 委员会上提出抽象语法这一问题——我应该这样做，但是我没有。

**InfoQ：我所提到的 ALGOL 格式中没有做到至少像 S 表达式那样符合抽象语法？**

John：你认为这些不同类型的程序表现方式是否会对设计者提出不同的技术要求呢？你是否建议作为一个设计专家，在核心抽象语法之上，可以上加不同的修饰呢，比如说“口红”——“口红”（Lipstick）在当前的美国特定的政治时令中是非常流行的。我对此表示怀疑。我认为人们有足够的适应能力去针对不同的规范进行设计。

**InfoQ：你在回答前面的问题的时候，你指出人编写程序是重要的，同样，能编写程序的程序也是重要的。你认为同样的编程语言可以很好的服务于这两个目的吗，或者它们具有同样的抽象语法，而人类也许想使用不同的具体语法？**

John：我认为同样的抽象语法就是派得上用场的，除非没有这样的抽象语法，因此至少我并没有直接地见到真正很好的例子。也许其它的人有——那就是用于编写 Lisp 程序的 Lisp 程序。

**InfoQ：让我们来看列表上的第 6 个问题。Lisp 有着很奇怪的历史，有着许多的变种(方言)，许多的实现，和许多的争议。你认为 Lisp 所尝试的这些思想的平台，对于它本身以及编程语言整体来说，是有益的吗？**

John：是的，我认为是这样，我没法详尽的说明。

**InfoQ：实验效果好吗？你认为 Lisp 的设计有什么特别的地方可以促进了实验或者说让实验更容易？**

John：当然，不存在什么老大。我从未想过当大哥。如果你看看 Fortran 或者 IPL，它们都有所谓的老大——APL 也有老大。

**InfoQ：现在 Python 有它的老大，Guido van Rossum，C#也有它的老大——Heilsberg 在那儿呢——还有 James Gosling——他不再想扮演 Java 的老大了，但他仍被看作是老大一样的角色。**

John：如果 Lisp 有什么老大的话，就是你了！参考手册可是你写的。

**InfoQ：我一直以来都拒绝扮演这种角色，也许是本能的参照你的例子又或者只是我不想当这个老大——我也不知道。有意思的事情是在过去的 10 到 15 年以来，编程语言都有一种关于领导的文化，而如你所说，这对语言的发展来讲也许不是个好主意。让我们来看看第 7 个问题。你认为 Lisp 所作出的最重要的贡献是什么？**

John：它为使用 Lisp 结构编程提供了一个方便的方式——比 IPL 方便多了，所以它有着实践的重要性。同样，Lisp 程序或 Lisp 数据也有着实践的重要性。

**InfoQ：当然，你使用 Lisp 结构作为程序的展现方式，因此它们是相关的？**

John：是的。

**InfoQ：我想知道的是最重要的一个贡献。我会列出 4 到 5 个其它我能想到的点并且征求你的意见。与 Lisp 程序以及 Lisp 数据密切相关的是它支持编码，应用与求值(eval)，就好像作为这一语言的普遍实现一样。求值(eval)的思想是一个通用的图灵完备函数，这也许是一个重要的思想。它还包括了面向表达的语言，对于递归函数的支持，影响到其它所有语言的麦卡锡条件表达式——甚至是 C，虽然有着怪异的语法，但也支持了它——，以及垃圾回收。我有没有漏掉什么重要的概念？也许有。我准备问问观众：我的列表上遗漏了什么吗？高阶函数，符号，动态编程。我想请问你，John，你对于这些概念有什么评论吗，比如它们是如何适应于 Lisp 这一宏图的，而你当时想要完成的目标是什么呢？**

John 就说垃圾回收吧——当我编写这个微分程序的时候，是以一种非常直白的方式来写的，使用 cons 来组合导数的部分，而要编写一个擦除器，即擦除数据的工具，却十分的笨拙。IPL 需要显式的擦除数据，它有一个责任位的符号，因此，当两个 Lisp 结构合并的时候，其中一个将负责它们的公共结构，它需要被编程。因此我深入地思考，“有没有什么方式我们可以消除显示的擦除器，这样就可以按照数学家描述微分函数的方式来编写它们了？”他们并没有描述，他们也没有在微分学教材中提到关于擦除的事儿，因此我不断的努力，最后得到了垃圾回收。 也可以通过使用引用计数的方式来达到这一目的，但当时的 IBM 704 的字没有足够的位来容纳引用计数。使用 CDC 机器的可以做到，但我认为垃圾回收不管怎样也是需要的。这个例子体现了将事物按照合适的数学的抽象标记来实现所带来非常重要的实际好处。当然，有的人告诉我说：“你并不是第一个想到垃圾回收的人。在此之前在这样或那样的程序里就有出现了。”我并不讨厌这么看，不过我想这可能是真的。

**InfoQ：回首往事，在你职业生涯中最值得你骄傲的是什么？而你想收回想法的又是什么？还有没有什么思想是你提出来的而没有被注意，但你想让它产生更多影响的？**

John：我想那就是如果 IBM 能采用我的时分共享的思想的话，具体的讲就是 Gene Amadhl 能在做 360 项目的时候采用我的时分共享的思想，那在 60 年代事情就会变得好很多了。因为他所做的不过是对 Bell 实验室设计的磁带对磁带操作系统的一种复制而已。如果说要产生更加实际的影响，那么这就是了。

**InfoQ：我听到观众都在说：“阿门”。这里有一个稍微奇怪一点的问题，列表上的第 9 个问题。斯坦福的 AI 实验室是一个独特的地方，远离校园，而且由一组不拘一格的组成，包括了搞软件的人，搞硬件的人，还有音乐家，每天都有排球比赛，而且有自己的计算设施。这些因素对于做这种有深远影响的研究有所帮助吗，或者这些因素其实没什么相关？**

John：我想可能没太大关系吧。Doug Lenat 曾经说如果我不牵涉斯坦福 AI 实验室的话也许更好，坐在一个办公室里致力于将数理逻辑中的常理形式化。可以说 Lenat 也许是对的，如果我能够在形式化数理逻辑的常理方面取得成功的话，但我不敢打保证我能成为的那个人。也许氛围也起了一定的作用吧，但这种自由氛围其中的一部分原因却是来自于我没办法将 DARPA 给我的所有资源都用于对我最重要的工作上，也就是形式化常理，因为我没法想出一个项目来做这件事情。因此，每个人都可以做他们想做的事情了。

**InfoQ：这也是你不想做领导，让人们做他们想做的事情的又一例子吗？**

John：我告诉你吧：如果我知道我想让他们做什么，我可能会试着当这个领导了。

**InfoQ：我认为，形式化常理是一个非常困难的问题，到现在仍是。**

John：确实如此。例如，我启发人们进行视觉研究，一部分是因为我参与了 50 年代关于视觉思想的争论，而这与歧视又有一定的关系。我创造了这一口号："描述，而不是歧视"，并且给出了一个例子，如果你想要机器人拾起什么东西，而你不能认定它，你必须描述它的形状以及它排放的样子。所以我们从事机器人研究并从 DARPA 拿到了赞助。

**InfoQ：如果没记错的话，60 年代末期，MIT 也在进行机器人的研究，特别是解决视觉的问题，他们声称"我们在这个夏天就可以解决视觉问题"因此它们有了 Summer Vision 项目，到了夏天结束时他们有了一些不错的关于摄像机的子例程，但我并不认为在这一点上解决了视觉的问题。直到 1967 年，花了一年的时间，John White 用 Lisp 为这个项目写了一些矩阵与版本的例程。**

John：那是哪一年？ (John White 回答) 第一个视觉项目是 66 年——Knight,我，Sussman，Lamport 还有其它一些人。

**InfoQ：计算机视觉仍然是一个困难的问题。现在我将再一次把这个列表放到一边，让我们的观众有机会向你直接的交流。在这个房间里我们准备了麦克风。如果有谁想向 John 直接提问的，那就来吧。(Herbert Sterns 提问)：在 Lisp 的发展中，Marvin Minsky 产生了什么影响？**

John：我不认为他有什么影响，也不认为他尝试过有什么影响。他所做过的一件事是，做符号化集成程序的 James Slagle 是他的学生。Slagle 对 Lisp 产生了一些影响。他对于自由变量的问题做了一些工作。他发现我所假设的这一问题可以解决其本身的方式存在一些不足，但我不知道他的问题是否有受到 Marvin 的直接影响。

**InfoQ：( John White 提问 ) Marvin 因为在 1960 年左右编写了将内存转储到二级存储，再取回内存的垃圾回收器而受到赞扬。这比我要早，但我读到的文献是这样说的。我想这可能是他对 Lisp 作出的主要贡献。**

John：这是加入到 Lisp 的垃圾回收器吗？

**InfoQ：我直到六年之后才参与进来，所以我不知道——不是我所工作于的那个 Lisp。**

John：不是很清楚，我记得曾经读过关于这个的一个备忘录。但不知道它是否真的被加入了生产环境的使用中。

**InfoQ：( Kent Pitman 提问 ) 我想知道，Lisp 这个名字，现在对你而言，在多大程度上表示这一语言是以列表处理为中心的？作为我个人来讲，我想关于 identity 或者 dynamic typing 对于这一语言的贡献比列表更为重要，所以我很好奇，如果我们回溯过去你是否还会叫它**

**Lisp？你认为其中心的特点是什么？**

John：这让我想起了上一个问题。我得对我刚才的回答再加几句。

**InfoQ：前一个问题是关于 Marvin Minsky 对于 Lisp 的影响。**

John：我问到："他是否加入了 Lisp 呢？"。这是一个真诚的问题，因为就算是在早期我也不是 Lisp 的领导。Lisp 会有新的东西增加而不经过任何人通知我。我并不是——所谓的——看护者。提示一下现在这个问题。

**InfoQ：这是 Kent Pitman 的问题，关于 Lisp 是名字是来自列表处理。你再回首的时候是否觉得列表处理是这一语言最中心或最突出的部分，或者说你会因为其它的一些更为重要的东西给它另起一个名字吗？**

John：我没有可以建议的替代名字，我不认为动态变量会是个好名字，你有什么名字建议吗？

**InfoQ：（Kent Pitman）我没有一个专门的名字好建议，不过我还有一个与此相关的问题。你如何评价动态类型，我认为这是这一语言的中心之一，而静态类型的缺失是一种特性呢还是一个 bug 呢？**

John：我想两者都有吧。作为我的想法来考虑它是一个 bug。实际上，我记得关于 Slagle 的问题第一次的讨论，我的态度是："欧，你们总得把这事儿搞定！"之后我们有过严肃的讨论最终的结果是他们谈到有两种不同类型的变量等等。我实际上并未参加这一讨论或决定，都是由他们来做的。所以，这不是仅仅跟 Lisp 相关，其它的编程语言也是这样，我直到很后来才能完全理解。

**InfoQ：（John White）当我在卡耐基念本科的时候,Perlis 所追求的圣杯就是用符合和代数来对数学进行形式化，但他走的是 ALGOL 的道路。而你确实提到了关于 ALGOL 的一些事以及对于它的一些想像，于是他长时间寻求公式化的 ALGOL 以成为 Slagle 工作的竞争者，直到很多年之后他个人致歉并表示他对 Lisp 的赞同。我想知道的是你是否听过"formula ALGOL"这一术语，如果是的话，那么这种工作以及它在符合代数，数学实验室，极值等等方面产生的分支，是否可以以一种类似于 ALGOL 的语法和语义来实现？**

John：当 formula ALGOL 一被提出的时候我就看过了。我认为它很不好，它很混乱，而这混乱之处来自于：它重载了代数操作符并将它们用于公司。在一定程度上，你可以这么做。但另一方面，你有着加法，减法等操作，你可以将它们用于公式而得到两个公式的和或者两个公式的差。但一般来说，你并不想这么对公式操作。至少在他们给出的例子中，他们没有包括对公式的微分等等。

**InfoQ：John，现在我回到这一问题列表。你能看一下第 14 个问题吗？你认为好的标记能促进好的思想而不好的标记会妨碍它这一说法吗？**

John：我想抽象语法是一个好的主意，而并不是标记。抽象语法可以工作于特定的标记的幕后。然后，因为人们并不使用抽象语法，所以你会问："从历史来看，标记扮演着怎样的角色？"——我想我只能说我不知道。

**InfoQ：OK。我只是想知道：抽象语法只是抽象的吗还是它是其他样子的，或者对于抽象语法是否也有良好的设计与不良的设计呢？**

John：不知道有什么其它替代的抽象语法。就"和"而言，就抽象语法而言，真的不存在任何替代。

**InfoQ：这是一个比较奇怪的问题，我想看一下我能得到怎样的结果。最后一个问题了。你对于现在的青年研究员有什么建议吗？你觉得最重要的一个研究课题是什么？**

John：形式化常理！

**InfoQ：在当时是一个重要的问题，而如今仍是如此。OK，非常感谢你，John。我们占用了你一个小时的时间，我们也非常感谢你能打进来。我们对于你发明了 Lisp 以及在逻辑方面所做的其它工作表示崇敬。我想在这个房间里几乎所有的人都可以说他或她的职业多多少少是站在了你的肩膀之上——我自己毫无疑问。我们只想表达我们的感谢。再次感谢你！**

John：谢谢！

**观看完整视频**：

http://www.infoq.com/cn/interviews/Steele-Interviews-John-McCarthy-cn

**相关内容：**

- 探索JVM上的LISP
- Scheme语言即将被一分为二
- 讨论：Java编程风格的改变
- Gestalt：使用Ruby，Python和XAML编写网页脚本
- .NET反应性框架为事件实现了LINQ

热点新闻 | News

# 大事件：VMware以 4.2 亿美元收购SpringSource

作者 Scott Delap 译者 张龙

VMware今天宣布以 3.62 亿美元的现金加上 5800 万的股权收购SpringSource。新闻如是说：

> ...SpringSource 的 CEO Rod Johnson 说到：“VMware 通过创新的虚拟化与云架构领导着现代化的数据中心基础设施，以此节省用户的费用，为其提供敏捷及更多选择，而 SpringSource 团队与社区致力于改变公司构建、运行及管理应用的方式。通过这种强强联合，我深信我们将为用户提供卓越的解决方案以简化企业 IT”。

SpringSource创建者Rod Johnson同时还在SpringSource博客上写到：

> ...这样一来，问题就变成了如何以简单、强劲而又实际的方式在数据中心以及云中利用 SpringSource 技术？这是一个改变游戏规则的好地方...通过与 VMware 的合作，我们计划为数据中心、私有云及公共云创建一个独立、集成同时又融合了构建、运行及管理于一身的解决方案。该解决方案会利用应用结构的力量，同时还会与中间件及管理组件协作以在部署期与运行期提供最优的效率及高伸缩的虚拟环境。它会发布一个 Paas（构建在你所熟知的技术之上）以降低费用及复杂度。该解决方案将构建在开放、便携的中间件技术之上，能够运行在传统的 Java EE 应用服务器上（该服务器也将处于传统的数据中心与 Amazon EC2 或是其他弹性计算环境中，当然了，还有 VMware 平台）。

Johnson还在博文当中提到他希望SpringSource能继续并增加对开源的贡献。VMware R&D部门的高级副总裁及CTO Steve Herrod也对此次收购发表了自己的看法。

InfoQ 将会持续跟进业界对此次收购的反响并为广大读者呈现第一手信息。

**业界反响**

Enomaly的Reuven Cohen推测此次VMware对SpringSource的收购是其Paas战略的部分。

Techcrunch说此次SpringSource的收购将成为Benchmark Capital意义重大的一天，对于FriendFriend也一样。

第一手的分析总是来自于受人尊敬的RedMonk...

**下面的内容来自于 SpringSource 官方网站**

请大家放心，我们对开源及协议等的承诺是不会因此次收购而改变的。我们期望能持续不断的为开源世界贡献力量，同时这些开源项目也将保持现有的发展势头。Spring 仍将保证部署环境之间的可移植性。

这两个公司的合并将为广大用户提供令人振奋的新技术，同时还将简化基于云的解决方案的开发工作。祝贺 Rod 及所有的 SpringSource 技术专家，正是由于他们的努力工作才使得这些伟大的技术成为现实，同时还开辟了成功的商业模式。

**原文链接：**http://www.infoq.com/cn/news/2009/08/vmware-springsource

**相关内容：**

- SpringSource Tool Suite 2.1.0 RC1 开始支持Spring 3.0 及OSGi开发工具
- SpringSource收购Hyperic
- VMware发布云操作系统vSphere
- 企业的虚拟化早已上路
- Jerry Cuomo谈虚拟化，云计算和WebSphere Virtual Enterprise

热点新闻 | News

# Java EE 6 的依赖注入终于达成一致

作者 Geoffrey Wiseman 译者 张龙

今年初，Google Guice和SpringSource宣布将合作提出一套标准的用于依赖注入的注解，即JSR-330。但这些注解与JSR-299却并不一致，随后引发了众多的争论，不过现在一切都已经尘埃落定：JSR-299 采用了JSR-330 的注解，两者都将成为Java EE 6 的一部分。

有不少人针对 JSR-299 与 JSR-330 的冲突谈到了自己的一些看法，列举如下：

- Gavin King：我认为引入另一套语义上与 299 相同的注解完全是个错误，而且其尝试解决的问题也与 299 大同小异。
- Bob Lee：虽然 299 对于那些小型的Java EE应用来说很适合，但其全局配置以及不直接的天性使之很难适应于数百万代码行的应用，就像Google所开发的。我们能够在Guice上轻松支持 299 风格的注解，但却无法通过 299 实现Guice的全部功能，因此没有理由放弃Guice而转向 299。就我个人来说，我认为你们在 299 上已经进行了不少的创新，但却没有完全理解用户代码是需要维护的这个事实。
- Alex Miller：向JSR 299 领域进军是个危险的信号。
- Antonio Goncalves：我希望我们不要打响一个新的战役，就像Java Module（JSR 277）和Modularity Support（JSR 294）之间那样。
- Rickard Öberg说出了反对意见：相对于泛泛的使用@Inject这样的注解，我们选择使用能代表目标对象范围的注解，因为什么都是也意味着什么都不是。

JSR-330已经通过了JSR评审的投票，但众多投票者都强调了两个规范的和谐相处：

- Sun：我们希望该 JSR 能与 JSR-299 共同努力以便为 SE 和 EE 平台达成一个一致、全面的依赖注入标准。这个标准务必先于该 JSR 的公共预览版发布前形成。

- Red Hat：我们认识到该草案是有社区支持的，因此打算在专家组发布公共草案时再发表最终意见。如果该 JSR 与 JSR-299 之间能达成某种一致（这种一致性会为依赖注入定义一种轻量级的模型），那我们会毫不犹豫地投出赞成票。Red Hat 承诺会为这种一致性贡献自己的一份绵薄之力。
- Ericsson：我们支持为标准化 Java SE 的依赖注入所付出的努力，但更想强调的是保持与 JSR 299 的一致性对于 Java SE 和 EE 都是非常重要的。
- IBM：我们也认为这样一份描述 SE 应用的依赖注入规范是很有必要的，然而所提出的注入模式却与 EE 平台中的定义有出入。SE/EE 的注入模型必须要形成一个单独可扩展的编程模型：为 SE 定义一套核心功能并通过 EE 的功能对其进行扩展。因此，要是不统一的话，IBM 是不会支持 JSR 299 或是 330 的。
- Oracle：虽然支持该 JSR，但 Oracle 严重关注该草案的完整性及其与 JSR 299 的分歧，因为这可能会导致平台的分裂。因此，我们期望在该 JSR 的公共预览版发布前能与 JSR 299 达成一致。我们相信 JSR 250 的一个修订或是维护版会比较适合发布依赖注入相关的注解。最终我们希望这种一致性的努力会让 SE 和 EE 平台的依赖注入保持一致，形成一个标准化的机制以满足各种需求。

目前这些规范之间的冲突已经得到解决。JSR-330（面向Java的依赖注入）以及JSR-299（面向Java EE平台的上下文与依赖注入）已经达成一致了，后者将采取前者的注解，两者都将成为Java EE6 的一部分。迄今为止，社区的反响还是积极的（Matt Corey、Jeremy Norris、Alex Miller、Oliver Gierki、Niklas Gustavsson）。

**原文链接：**http://www.infoq.com/cn/news/2009/08/dependency-injection-javaee6

**相关内容：**

- Java依赖注入
- Web Beans（JSR-299）：与规范领导者Gavin King的问答
- Guice和JavaConfig：使用Annotation进行反转控制的两种方式
- 想快快喝下Google果汁——Guice吗？
- Guice 2.0 发布

热点新闻 | News

# 讨论：Java编程风格的改变

作者 赵劼

最近Stephan Schmidt在博客中发表了题为《下一代Java编程风格》的文章，阐述了他眼中Java编程风格的改变，以及未来的走向：

> 许多公司和开发人员正在从 Java 转向其他编程语言：Ruby、Python、Grrovy、Erlang 或 Scala 等等。不过你可能做不到这一点。即便如此，你也可以改变你的编程风格，获取这些新语言的优势。事实上，在过去的 15 年中，Java 编程风格也已经有明显变化了。

Stephan 在文章中提出了以下几点：

1. **尽可能地标注final**：让所有东西不可变，把变量标为final可以阻止改变它的值。很多时候，重新为变量赋值会引入bug，你应该使用新的变量。除此之外，final可以提高代码的可读性。我针对这个话题还写过一篇文章：《Java中所有变量都应该是final的》
2. **没有setter**：许多Java程序员会自然而然地为类中所有的字段加上setter。思考一下，真的每个字段都需要修改吗？更好的方法是创建包含改变后状态的新对象。此外，也试着去除getter，我们应该遵循“Tell, don't ask”的思想。
3. **避免使用循环来操作List**：从函数式编程那里获得的经验，循环并不是进行集合操作最好方法。例如，我们可以使用Google Collections提供的过滤功能。

```
Predicate canDrinkBeer = new Predicate() {
public boolean apply(HasAge hasAge) {
return hasAge.isOlderThan( 16 );
}
};
```

```
List<Person> beerDrinkers = filter(persons, canDrinkBeer);
```

4. **使用单行代码**：Java 是一门繁杂（noisy）的语言，我们应该编写更精确的代码。尝试将代码写为一行。例如：

```
public int add(int a, int b) { return a + b; }
```

5. **使用大量接口**：领域驱动设计已经大行其道，一个应该拆分为多种"角色"，即实现多种接口，提高复用程度。方法应该面向"角色"，而不是面向特定的类。我在《不要在Java中使用String》一文中讨论了更多这方面的内容。

6. **使用Erlang风格的并发**：Java的并发特性（如lock和synchronized）过于低端，难以使用。Erlang风格的并发是一种更好的做法。Java平台上已经有了Akka和Actorom。此外，也可以使用java.util.concurrent中的Join/Fork和数据结构进行编程。

7. **使用 Fluent Interface**：Fluent Interface 可以使代码更短，更容易编写。Google Collections 中的 MapMaker 是个不错的示例：

```
ConcurrentMap graphs = new MapMaker()
.concurrencyLevel(32)
.softKeys()
.weakValues()
.expiration(30, TimeUnit.MINUTES)
.makeComputingMap(
new Function() {
public Graph apply(Key key) {
return createExpensiveGraph(key);
}
});
```

8. **避免在 DTO 中创建 getter 和 setter**：如果你拥有简单的 DTO（Data Transfer Object），不要耗费精力去编写 getter 和 setter，直接使用公开的字段吧。不过在你无法完全控制代码的使用情况时，还是小心为上。

这篇文章发表之后，有许多人发表了不同的看法。其中Cedric Otaku发表了文章《下一代Java与现在差不多》予以回应，其中反对了Stephan提出的大部分观点。

- **尽可能 final**：太多 final 会降低代码的可读性，它无法代码额外的好处。我已经不记得上次因为重新给变量赋值而造成错误是什么时候了。值得一提的是，在字段以外的成员上标记 final 违反了 Google 的风格指南。
- **避免setter**：看上去不错，不过这不现实。有些时候你不愿把所有的参数都通过构造函数传入。此外，如果使用对象池的时候，可变的对象会让编程更为方便。Stephan 不是第一个提出要将访问器（accessor）从OO编程中移除的人，不过这个说法很明显不可行。
- **避免循环**：Java 并不适合函数式编程风格，所以我认为使用 Predicate 的代码反而难以读懂。我估计大部分的 Java 程序员会同意我的观点，即使他们已经熟悉了闭包风格。
- **单行代码**：这要视情况而定。并引入临时变量把一个表达式拆开可以提高代码可读性，也容易为其设置断点。
- **使用接口**：不错的建议，但也不能过火。过去我也争论过类似的话题，不过引入太多接口会导致细小类型的爆炸，使你高端的类型意图变得模糊。
- **Erlang 风格并行**：重申一点，使用 Java 设计以外的编程风格是危险的做法。java.util.concurrent 中包含了非常有用的功能，我遇到过不少基于这些元素的 Java 抽象，它们要优于 Erlang 风格的 actor 架构。
- **Fluent Interface**：这个建议比较有趣，它与 Stephan 提出的另一个建议“避免 setter”相违背。Fluent Interface 只是 setter 的另一种形式，不是吗？
- **使用公有字段**：不，千万别这么做。你不会因为加了访问器而后悔，但是我能保证你会因为一时偷懒，使用了公有字段而后悔莫及。

在 Cedric 的文章之后，Stephan 又对他的说法进行了补充：

> 没有 setter 并不代表你不能修改这个对象，我只是说纯粹的 setter 不是面向对象的思维方式。例如，你觉得 stop()和 setStop(true)哪个更好一些？
>
> （针对 Predicate 代码不易读）我认为你的假设有误。循环是“程序化”的代码，而 Predicate 是经过封装的，可以重用的，易于理解的“对象”。这里并没有函数式编程，这里是纯粹的 OO - 我提起 FP 只是因为我从那里“引入”了这个方式。

还有许多人对 Stephon 和 Cedric 的文章发表了评论，例如有人支持 Stephan 的观点，认为 final 的可以更好的表示出代码的意图。甚至有人提出：

> 更简单的解决方案是使用 Scala :) - 不可变的状态、统一访问原则（字段、属性、

> 方法看上去一样）、单行代码、使用 monads 或函数来替代循环……这些特性都已经在 Scala 中优雅地体现出来了。

您的 Java 编程风格是什么样的，和过去相比有什么改变吗？

**原文链接**：http://www.infoq.com/cn/news/2009/08/java-programming-style

**相关内容**：

- 反对for行动
- 反对if行动
- “原罪”（没有原生数据类型，Java会更好吗？）
- 讨论：怎样才算“易于维护”？
- Andrej Bauer对于语言设计的观点

热点新闻 | News

# PHP在IIS 7 上雄起

作者 赵劼

WordCamp是WordPress爱好者及开发人员的大会，在上周末的WordCamp China 2009 大会上，来自微软的王超群发表了题为“熟悉的陌生人 - 微软对PHP的新支持使WordPress在IIS 7 上雄起”的主题演讲，展示了PHP在IIS 7 上运行的现状及微软在这方面做出的努力。

在演讲中，王超群首先引用了首席架构师 Ray Ozzie 的话：

> 我认为如今任何的公司及技术提供商，即使是微软，也必须在开源的用户和贡献者中寻找一个合适的平衡点。

以及 Linux 创建者、开源领袖 Linus Torvald 近期对微软所发表的的看法：

> 哦，我强烈认为“技术高于政治”……自由软件世界中有一些“极端主义者”，这也是我不再把我做的事情称作“自由软件”的主要原因。我不想和那些有排斥和憎恶心态的人建立联系

此外，王超群还展示了微软在 Redmond 的开源实验室，以及和 Novell 共同组建的互操作实验室。微软希望借此表明他们对于开源的态度，为开源社区更好地接受 IIS 7 与 PHP 平台的合作打下基础。

微软认为，在 Windows 为 PHP 提供良好支持的意义在于：

- 无需明显的修改，便可以向 Windows 上移植 PHP 应用程序
- 将 PHP 与微软技术相结合，更好地构建丰富 Web 站点
- 重用已有的 PHP 应用程序以及 Windows 基础设施
- 在无需增加成本的情况下，为客户提供更多可用的应用程序及更好的性能

王超群表示 过去约有 80%的 PHP 开发者在 Windows 上进行开发 ,但只有少数使用 Windows 和 IIS 托管 PHP 应用程序。而现在情况已经改变，IIS 7 是 PHP 应用的优秀平台，并且易于设

置和管理。

王超群谈到，微软通过和 Zend 技术合作：

- 为 PHP 在 Windows 上的运行进行了优化。
- 构建了 IIS FastCGI 这一基于开放标准的运行方式，比传统 CGI 方式相比性能有显著的提升，并为非线程安全的 PHP 应用提供更为稳定的支持。
- 已经测试，并确保了众多流行 PHP 应用的兼容性。

并且努力将 Windows 与 IIS 7 打造为优秀的 PHP 生态环境：

- 使IIS URL Rewriter与mod_rewrite兼容
- 提供Microsoft SQL Server的PHP驱动程序
- 为 PHP 提供 Expression 工具的支持
- 在CodePlex上提供大量PHP项目
- 提供 IIS 上.NET 使用指南

王超群表示，IIS 7 已经成为一个托管 PHP 应用的优秀 Web 服务器，其模块化架构提供了丰富的功能及扩展能力，它的 Server Core 模式进一步减少了系统的资源占用，并足以托管 PHP 应用程序。利用.NET 来扩展 IIS 7，还可以打造一个统一的，通用的应用程序执行环境，让 IIS 更好地为应用程序服务。

在会上，王超群还公开了之前与康盛创想合作进行的性能评估结果，证明在Windows Server 2008 + IIS上运行PHP，从平均相应时间，每秒处理的请求数，以及数据吞吐量等多方便均显著优于Linux + Apache的托管方式。

王超群在会后公布了此次演讲所用的幻灯片。除了演讲中材料和展示部分外，幻灯片中还附有在Windows中运行PHP的的最佳实践和深度认知。

**原文链接：**http://www.infoq.com/cn/news/2009/08/php-on-iis-7

**相关内容：**

- Eclipse PHP开发工具套件 2.0 发布
- Websphere CTO谈REST和Project Zero
- Adobe推出支持Zend框架的AMF

热点新闻 | News

# SQL Server的未来之路

作者 Jonathan Allen 译者 张龙

去年我们曾报道过，SQL Server 2008 R2 将支持 256 个逻辑处理器。其他特性还包括更棒的多服务器管理工具以及 Reporting Services 3 中的地理空间可视化。现在新的预览版已经对 MSDN 和 TechNet 订阅用户开放了，而普通用户则要到本月末才能下载。

R2 提供了几个专门面向独立系统供应商（ISV）的特性。ISV可以为Gemini项目预定义模板。即将成为Office 2010 一部分的Gemini是Excel的分析引擎。微软声称Gemini能够“在单独的Excel工作簿中操作数百万行的数据”。ISV还可以预定义SQL Server Reporting Services组件。目前我们就知道这些，至于这与单独分发组件有何不同则不得而知了。微软还一直鼓吹要支持基于ATOM的feed。但纵观其不断尝试将SQL Server变成一个Web服务器以及无数次爽约的行径，开发者对这个承诺要小心了。对于用户来说，该特性列表还是颇具价值的。SQL Server 将能支持 256 个逻辑处理器，这对于那些寻求纵向扩展（scale up）而非横向扩展（scale out）的公司来说是个好消息；而对于进行横向扩展的公司来说，R2 提供了更棒的管理工具以与多服务器协同工作。

对于报表来说，即将发布的Report Builder 3.0 提供了对地理空间可视化的支持。凭借该支持，我们可以从数据直接创建基于地理信息的图表。可以从SQL Server新的博客上了解到关于地理空间可视化的相关信息。

**原文链接：**http://www.infoq.com/cn/news/2009/08/SQL-Server-R2

**相关内容：**

- 微软进入主数据管理市场
- 关系型的云呼之欲出
- 微软正式发布SQL Server的MapPoint插件
- 连贯NHibernate正式发布 1.0 候选版

热点新闻 | News

# Ruby 1.9 综述：Ruby 1.9 的Ruby-debug、 Ruby Switcher和MacRuby

作者 Werner Schuster 译者 杨晨

近一段时间Yehuda Katz一直在强烈质疑：到底是什么问题使得开发者远离Ruby 1.9。其实一个很重要的原因是应该归咎于不能在Ruby 1.9.x上运行的库和工具的列表上那一长串名字。

这个列表正不断缩短。不仅如此，不断的更新中还有近期发布的Ruby 1.9.x的ruby-debug工具。在这个版本中，Ruby-debug使用原生扩展来减少调试器中运行Ruby代码的开销。

Mark Moseley最近在GitHub上发布了一些代码，专注于如何更有效地减少调试器的开销。其基本思想是：在编译代码的时候采用插入特殊指令的方法来设置Ruby代码的断点，从而中断执行。

现在ruby-debug 1.9.x的安装指南已经可以在GitHub上找到。

随着Ruby版本的增多，在不同的Ruby实现版本上测试Ruby代码变得更加重要。我们在这里提供了一些可用的备选方案，例如使用MultiRuby或者Ruby版本库站点。

近期出现了一个名为Ruby Switcher的新工具。其采用了一种非常简单的思想：使用一个简单的命令来切换Ruby版本，例如使用命令use_ruby_191 切换到Ruby 1.9.1，而use_jruby使得ruby命令以及其他的程序库切换到JRuby。同样，为了正确地切换，这个工具也会下载和安装大量Ruby的不同版本。

本文的最后，是来自MacRuby的消息：MacRuby的实验性分支已经被并入到了主干库中。这个实验性分支开发了一个全新的VM，这个VM使用了LLVM来实现JIT和预编译( Ahead of Time，简称为AOT )技术。另外一个更新是MacRuby移除了GIL。在这里我们简单介绍一下MacRuby：MacRuby是Ruby 1.9.x的一个衍生版本，它继承了标准Ruby的线程特性，当然包括GIL在内。

在下一个稳定发布版本中，观察MacRuby的线程性能开销会是如何是个不错的主意；要知道

标准的Ruby 1.9.x在近期的版本中仍然会保留GIL。

**原文链接：**http://www.infoq.com/cn/news/2009/08/ruby19-rubydebug-macruby

**相关内容：**

- JRuby综述：Ruby 1.8.7 支持、Android支持及Bcrypt-ruby
- JRuby综述：JRuby团队转投EngineYard，YAML支持的更新，OSGi的支持，Installer的讨论
- IronRuby综合报道——IronRuby 0.9.0 及其基准”
- Android开始支持脚本语言Python、Lua及Beanshell，未来还将支持Ruby
- Ruby On... SAP：借力全新Ruby VM，企业化路上又迈一步

热点新闻 | News

# BPM和SOA的最佳实践和最差实践

作者 Boris Lublinsky 译者 马国耀

Peter Woodhull在他的新作“BPM和SOA中的最佳实践和最差实践”开篇这样写道：

> 很多企业继续借助于 BPM 和 SOA 追求业务流程效率和效用的提高，但还是失败了。而促成或破坏一个项目的方法都有好几种。

Peter 讨论了一些 SOA 和 BPM 实施的最佳实践和最差实践。在他看来，以下是一些最差实践：

先买软件。Peter 认为，最坏的错误是一个 BPM/SOA 的项目从评估和购买软件开始。问题是很少有公司能够真正事先知道他们需要那类软件，把解决方案往买软件上靠的做法无异于让别人来掌管你的业务。

> ……大部分从软件购买开始的项目都是有 IT 部门负责的，并且其最终结果往往是自底向上的支持和实现的策略。这种做法和业务的战略目标脱节，因为它更偏向于以技术为中心而不是以业务流程及业务需求为中心。

不重视组织结构的变化。因为人们总是反对变化的，不论变化是否能够给他们的工作带来便利。

> 对于即将开发的新流程和系统，如果用户能够以合适的方式参与进来，并且有机会去评审、加注、验证以及做辅助决定，那么，人们将消化这些变化并接受它们。

试图“煮沸整个大海”。将一个 BPM/SOA 的解决方案的实施当作大范围的翻新并铺开的做法是几乎不可取的。

> BPM 和 SOA 的工作本身是不断发展的，最好以一种小规模、受控并且频繁发布新能力的方式迭代成长，其能力应该以一种受控的迭代方式展开。流程和服务应该分开管理和实施，从而为其用户群带来即时价值

最佳实践，Peter 也描述了以下几条：

**一切始于发现。**Peter 认为在没有对问题有清晰了解之前就提出解决方案的做法是很多失败的原因之一。

> 准确定义将要管理的流程并文档化服务合约（WSDL 文件和数据结构），这是任何实施项目最首要而且最重要的工作。一旦流程规约被准确而清晰地记入文档，并且通过客户以及合作伙伴的验证，签名和批准后，只有在这之后才能由开发团队实施开发和原型设计。

**BPM 和 SOA 应是一个复合解决方案。**很多人认为 BPM 和 SOA 是两个不相干的事物，经常由不同的部门实施，并且具有不同的优先级。

> BPM 和 SOA 实际上是……解决业务上的一些常见且普遍存在的问题的策略和技术。而且……技术平台对它们都有很好的支撑。BPM 套件是非常有效的整合工具，特别是存在将人和计算机系统集成到一个统一的解决方案的需求时，而 Web 服务和 SOA 技术是实现代码重用以及在计算机系统、平台以及组织之间实现互操作的很好的机制。

**从关键任务流程开始。**和任何新的方法一样，SOA/BPM 也需要通过验证才能赢得管理层的支持。

> 从某个关键任务的业务流程开始，而且，其价值应该可以明确并且可以量化。理想情况下，应该选择一个正好是客户关心的且没有明确的解决方案的业务流程……这样的结果将是公司的业务部门负责（业务流程的）实施，而不是由 IT 部门负责。

Peter 在文章结束时强调了 SOA 和 BPM 联合实施的复杂性及其强大能力。他还鼓励采取业务驱动而非技术驱动的方法进行 SOA 和 BPM 实施，紧随其后是“几要几不要”的建议，这些建议虽然不能确保成功，却可以降低失败的风险。

**原文链接：**http://www.infoq.com/cn/news/2009/08/BestWorst

**相关内容：**

- 开放流程用户组(OPUG)发起人辛鹏专访
- 要不要自建工作流引擎？
- 使用BPM与SOA来最大化业务价值
- 用工作流方式构建应用
- 将企业视为一个事件网络

热点新闻 | News

# CRUD不适合REST吗？

作者Boris Lublinsky 译者 马国耀

Arnon Rotem-Gal-Oz在他的新博文"CRUD不适合REST"中这样开篇：

> 表面上看起来它们很相配（无论从技术上还是架构上），然而，一旦深入了解，你就会发现它们并不相配。

今天，REST 架构风格的常见实现是基于 HTTP 协议及其相应动作（如 POST，GET， PUT 和 DELET）的。而且，这些动作经常会被实现者映射为 CRUD 术语——Create，Read，Update 和 Delete。典型的做法是 1 对 1 映射。

- GET 通常被映射为 CRUD 中的 Read，只是 GET 还提供了超越开箱即用（Out-of-the-box）的 SELECT（即 Read）映射的一些特征。
- DELETE 通常被映射为 CRUD 中的 Delete
- PUT 通常映射为 CRUD 中的 Update，只不过增加了一些限制：
  - PUT 替换的对象是整个资源，而 Update 可以替换一部分。
  - PUT 可以用于创建一个资源（当客户端设定 RUI 时）
- POST 通常被映射为 CRUD 中的 Create，但它仅支持创建子资源，可是 POST 支持对资源的部分更新。

Arnon 认为：

> HTTP 动作更加面向文档而不是数据库，至少在谈到 HTTP 动作时，执行 Update，Delete 和 Create 新资源采用的做法和 CRUD 在数据库的世界里的做法是不完全一样的。

然而，CRUD不适合于REST的最大原因是架构上的，REST的核心是使用超媒体实现的协议状态机。Arnon引用Tim Ewald的话：

> ……这是我所理解的。每一种交互协议都有一个状态机，有些很简单，有些则比较复杂。当你通过 RPC 来实现协议时，你是在创建用于修改交互状态的方法。从交互端点看来，状态是由一个黑盒子维护的。由于协议状态是隐藏的，所以很容易出错。比如，你很有可能在初始化之前就调用某个流程。很长一段时间里，人们一直通过为接口的类型信息下注解的方式来寻找避免这类问题出现的方法，但是我从没有看到任何主流的解决方法。事实上，协议的状态隐藏在方法调用的背后，在方法调用的过程中隐式地修改状态，这个事实更增了版本控制的趣味。
>
> REST 的精髓是通过 URI 来显式描述协议的状态。协议状态机的当前状态是由最近一次操作的 URI 以及该操作获取到的状态描述决定的。若要修改状态，那么就对 URI 进行相应的目标状态的操作，使资源呈现期待的新状态。状态描述包括指向其他状态的链接（状态图中的弧线），通过这些链接可以从当前状态移动到目标状态。基于浏览器的应用就是这么工作的，而且你的应用程序协议没有理由不能那样工作。（ATOM 发布协议是一个经典的状态机例子，尽管它被经常认为是关于实体的，而不是状态机的）

接着John Evdemon&rsquo的文章中所阐述的为什么CRUD服务是SOA的反模式，Arnon描述了CRUD REST的缺点：

- 限制了服务的整体概念——没有业务逻辑。
- 暴露了内部的数据库结构，或者数据接口，而不是仔细考虑后的接口。
- 鼓励直通服务和数据的做法。
- 建立的是块（Blob）服务（数据源）
- 鼓励细小的多重服务（为块服务定义多重接口），而忽略了分布式计算的若干谬误。
- 仅仅是“批着羊皮”的 C-S 结构。

Arnon 在博文结尾的时候再次强调，仅仅采用诸如 HTTP，XML，JASON 等标准（尽管他们很有用）还不能构成 REST，而只有采用了 REST 架构才算真正的 REST 这篇博文的重要性在于它提醒到：REST 和 SOA 类似，它不是一组标准和流行的 API，而是一种架构模型，这才是需要去理解和遵循的。

**原文链接：** http://www.infoq.com/cn/news/2009/08/CRUDREST

**相关内容：**

- MIME给REST的采用带来了问题？
- REST在IT/Cloud管理中的角色——API的对比
- REST – 善，恶，丑

热点新闻 | News

# 叠飞机与敏捷项目知识传递

作者Vikas Hazrati 译者郑柯

将某种情形下的知识从一个单位（可以是个人、团队、部门、组织）传递到另一个单位，这就是知识传递。很多组织用了很多时间将自己积累的知识记录成文档，希望知识传递过程能由此变得更顺利、高效。而敏捷并不鼓励文档，它强调"可工作的软件胜过全面的文档"。在在一系列有趣的试验中，Steve Bockman试图找出在敏捷项目中传递知识的最佳途径。

在试验中，Steve试图将一只不寻常的纸飞机作为产品，并将其相关的知识通过三种方式传递。他使用了下面三种策略：

- **文档**：工作者们得到写下的纸飞机制作说明（包括 22 个步骤）。
- **反向工程**：工作者们得到一个已完成的纸飞机，他们可以用之学习如何重现制作纸飞机的步骤。
- **指导**："首席设计者"按步骤制作一只纸飞机，而工作者们重复完成的每一步。

参与实验的共有 8 个人，每种方式各用 5 分钟。实验结果令人惊讶不已。

只有 **12.5%**的人能够按照文档完成任务。使用反向工程方法，有 **25%**的参与者成功做出飞机，而指导方法则可以让 **100%**的参与者全部成功做出飞机。

这毋容置疑地指出：健康的沟通和指导，是传递和分享知识的最佳方式。Steve 还认为：对于需要经常沟通和反馈的软件开发来说，这个原则更具价值。在他看来：

> 假如我是一个开发人员，我发现了一个技巧，可以将一些数据绑定到某个用户界面里的控件中，而且写出了代码实现。这个技巧构成了一种模式，与我一起开发的同事们希望了解具体做法。如果你是我的同事，有三种方法：a）我给你一个说明该技巧的相关文档；b）我告诉你代码在哪里，建议你自己弄明白；c）我跟你结对编程，通过一组新数据实现该模式；你会选哪一种？

Young Ye和Royce Fay建议使用另外一种使用不均衡结对编程（Asymmetric pair Programming）高效传递知识的方法。该方法的本质在于：它除了在开发人员之间结对之外，还可以在开发人员和领域用户之间结对。这样做的重点也在于人与人之间的沟通，而不是文档。

结对编程有一个广为人知的好处，就是快速的知识分享和传递。Alan Skorkin同意这个观点，同时指出：

> 我认为：最重要的好处在于，结对对于有机的知识传递效果非常好，尤其是大型系统中，这是关键，因为根本没有其他方式能够做好这一点。

因此，大家都同意传递知识的最好方式就是通过沟通、指导和一起工作。虽然，有些文档确实有用，但单单依赖文档能带来的好处很有限。

**原文链接：**http://www.infoq.com/cn/news/2009/08/agile-knowledge-transfer

**相关内容：**

- 学徒模式
- 结对编程 vs. 代码复查
- 成功实施结对编程
- 初心，聆听之术
- 敏捷记忆卡：学习和记忆敏捷的索引卡

热点新闻 | News

# 速览：软件开发中的 7 大浪费

作者 Mike Bria 译者 金毅

精益软件开发的基本原则是追求浪费最小化。TPS总结了制造业的 7 种主要浪费，而Poppendiecks已经把它们引入到我们软件开发的领域中。Jack Mulinsky最近在agilesoftwaredevelopment.com上发表了系列文章对这些浪费进行一一介绍。

该系列文章的第一篇讨论了部分完成工作这一浪费。软件中的这种情况相当于精益制造业中的“进行中的清单”，Mulinksy认为这是所有浪费中最极品的部分。他对“进行中的工作”这种浪费归纳如下：

- 已完成但尚未签入的代码
- 没有相关说明文档的代码
- 未测试的代码
- 没人使用的代码
- 被注释掉的代码

第二篇文章中，参考那个众所周知的统计数据：在现有的软件应用程序中，多达三分之二的功能几乎或从未被使用过，Mulinsky提出额外的功能（制造业中的“生产过剩”）这一浪费。除此之外，他还指出什么是直接浪费（开发的花费而不是人员）以及什么是间接浪费（更多的功能意味着更多的代码以及更高的维护复杂度）。

Mulinsky的第三篇文章关注的是软件再学习/返工的浪费，这对应于精益制造业理论中的“额外处理”这一浪费。他用了这样一些浪费的例子来描述了他的想法，包括：

- 糟糕的计划
- 低劣的质量

- 在不同的任务间切换
- 不足的沟通和知识积累
- 没有相关说明文档的代码

这一系列中的第四篇（写此文时最新的部分）讨论的是交接带来的浪费，软件中的交接相当于制造业中的“运输”。他提到了以下一些可能产生浪费的技术方面的例子：

- 开发人员之间的代码交接
- 开发人员和测试人员间软件的交接
- 软件从开发到部署的交接

Mulinsky 应该很快会继续其余 3 个软件浪费的话题：在不同的任务间切换，延期和缺陷。软件浪费这一概念，是由 Mary Poppendieck 和 Tom Poppendieck 在他们两本非常出色的精益软件方面的书《Lean Software Development: An Agile Toolkit for Software Development Managers》和《Implementing Lean Software Development: From Concept to Cash》中提出的。如果这是你和这一概念的第一次亲密接触，敬请期待 Mulinsky 的后续大作。

同样，对那些很熟悉这些观点的朋友们，不妨分享一下你们觉得浪费是怎么在软件开发中产生的，你们又是如何消除它们的？这会对大家都有很帮助。

**原文链接：**http://www.infoq.com/cn/news/2009/08/seven-wastes-intro

**相关内容：**

- 软件债务的累积会消耗巨大成本
- 结对编程的经济价值论
- 通过“敏捷三角形”度量敏捷成效
- 敏捷项目注资——得到高效结果的巧方妙计
- 敏捷项目中的人力资源管理

热点新闻 | News

# 百度技术大会推“框计算”概念引热议

作者 霍泰稳

在 8 月 18 日举行的 2009 百度技术创新大会上，百度董事长兼CEO李彦宏表示，百度公司将推出全新的计算平台“框计算”，这个全新的概念在技术社区引发热议。在介绍“框计算”时，李彦宏说道：

> 百度推出的“框计算”是要把所有用户的需求无论是在找信息，还是要求各种各样的应用，还是有什么其他的需求，都集成到一个框里，用一个框来满足用户的需求。
>
> “框计算”为用户提供基于互联网的一站式服务，是一种最简单可依赖的互联网需求交互模式。用户只需要在搜索框中提出需求，系统就能明确的识别到这种需求，并同时将该需求分配给最优秀的应用或内容资源提供商处理，最终返回给用户相匹配的结果。

这个全新的概念受到了诸多专家的肯定。著名经济学家、国务院发展研究中心高级研究员吴敬琏教授表示：

> 中国增长模式陈旧，不可持续，在当前新的产业革命正在孕育的形势下，我们必须急起直追，迎头赶上。寄望于“框计算”能在振兴我国产业和实现新的产业革命中崭露头角，做出贡献。

而微软亚洲搜索技术中心工程总监刘激扬则认为：

> “框计算”将为技术开发者提供更广阔的互联网技术开发舞台。

与专家们反应有所不同的是，技术社区内对该概念传出质疑的声音。有众多网友表示，该概念与当前流行的“云计算”太过接近。还有网友表示“框计算”听起来只是“产品用户界面的整合”，并无任何创新之处，甚至被部分网友戏谑为“‘诓’计算”。早在此之前，谷歌便提出“搜索一次到位”的整合搜索技术，该技术从描述上看与“框计算”并无本质不同之处。而且，该技术也并非百度原创：

> 2004 年 7 月，Autonomy 通过其位于美国旧金山的控股公司推出搜索工具 Blinkx。这一工具可以提供类似“模糊搜索”或“语义搜索”的功能。该系统经过 “学习”积累了一定“经验”后，可以满足用户类似“最便宜的笔记本电脑是什

> 么”“中国队能否赢得世界杯”这样的搜索需求。
>
> Autonomy 公司是智能搜索技术发明者，目前在商用搜索市场居绝对领先地位，占这一市场 55%以上的份额，英国警察署、美国本土安全局、国防部、航天局等政府部门，以及通用电器、通用汽车等企业，都是它的客户。但在民用搜索方面作为并不大，远不如 Google、百度这么有名。

近期，谷歌又推出了当前搜索引擎的修订版“Caffeine”，据称谷歌工程师重新编写了搜索引擎的部分基础架构，而且速度更快。对于谷歌在技术上的步步紧逼，雄霸中文搜索市场的百度公司拿出创新技术加以应对，也是理所应当。相似的是，前段时间百度强势推出的“阿拉丁”开放搜索平台，也被质疑和Google Base太过接近。虽然同为用户自定义提交结构化数据的产品，Google Base支持包括CSV、RSS和Atom在内的多种提交格式，而百度的阿拉丁平台则只支持百度自定义的XML格式，因此被指并不如名字一般“开放”。而百度方面则表示，“阿拉丁”作为“框计算”的重要部分，能够应对“暗网”的收录和检索，是“搜索引擎的未来”。“框计算”作为一种占领用户终端的手段，对于未来用户终端的发展也将起到深远影响。CNET（中国）媒体总编刘克丽在会上表示：

> 李彦宏的“框计算”听起来很朴实，相信未来它终将改变所有的终端界面。

目前业界普遍认为，浏览器是未来终端的主力技术，无论是谷歌的Chrome OS，还是微软的Gazelle，都是遵照该理念进行技术创新。尚不知百度的“框计算”是否能成功颠覆这一思路，创造出新的终端技术。另外，有媒体质疑该技术会影响互联网的生态环境，增加百度对中文互联网的垄断：

> 有了“框计算”，百度便不再是搜索引擎了，而是互联网世界的“南天门”。用户需要什么，都由百度这个“框”一手操办，要么是把百度自有的服务推给用户，要么是把商业合作伙伴推给用户。这个“框”不仅是用户进入互联网的大门，更像是一个绳索，绑架了用户，绑架了市场需求，绑架了合作伙伴。从这个角度来看，“框计算”似乎更像是垄断的代名词，打算对用户的互联网世界进行“包办”。

作为全新的概念，“框计算”究竟是中国领先的自主创新，还是山寨抄袭，请读者朋友们留下你们的一家之言吧？

**原文链接：**http://www.infoq.com/cn/news/2009/08/baidu-box-computing

**相关内容：**

- 和Google互补的搜索引擎Wolfram|Alpha
- Google App Engine转向了Jetty
- Joyent：别样的云计算平台

# InfoQ中文站

www.infoq.com/cn

我们的**使命**：成为关注软件开发领域变化和创新的专业网站

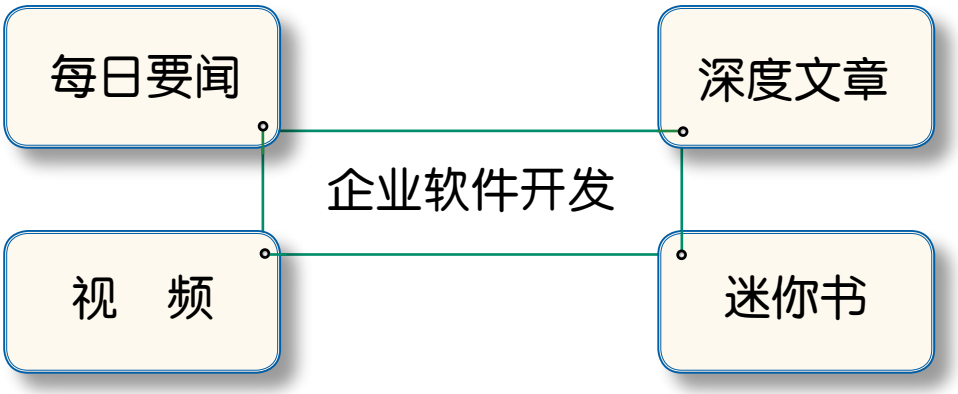
我们的**定位**：关注高级决策人员和大中型企业

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

我们的**特质**：个性化RSS定制、国际化内容同步更新

我们的**团队**：超过30位领域专家担当社区编辑

……




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

推荐文章 | Articles

# Tech Lead的三重人格

作者 熊节

很多团队都有 tech lead 这个角色的存在，但同时很多团队对这个角色都缺乏明确的定义。大多数时候，团队只是指派其中经验最丰富、技术最精熟的开发者来担当 tech lead。但除了“tech”的成分之外，这个角色还有“lead”的成分，这就决定了他不仅需要技术上的能力，还要眼观六路耳听八方，才能带领团队——至少是开发者们——取得成功。

Tech lead 需要关注的事情可谓纷繁芜杂。把这些事情分门别类，我们可以看到，这个角色大致有三方面的职责：技术决策者、流程监督人、干扰过滤器。

**技术决策者**

从技术的角度，tech lead 需要关注以下三个方面：架构设计、局部设计和关键技术点。

**参与架构设计**

系统的架构通常是在项目初期的 quickstart 阶段设计出来的。这个架构设计包含的都是高层面的内容，例如 C/S 或 B/S 的选择、开发平台、编程语言、数据迁移策略、集成点、部署结构等。

尽管架构师（architect）是架构设计的主要负责人，tech lead 还是会参与设计过程，以获得对系统全局的清晰了解，并且尽早发现架构设计中不合理或者风险较高的部分。

主要关注点：

- **选择什么平台和编程语言？**
- **需要和什么系统集成？**需要使用哪些第三方软件或工具？给自己画一张集成架构图：先用虚线圈出为用户提供服务所需的整个系统范围，再用实线圈出将用主要编程工具（例如 J2EE 或者 Ruby on Rails）开发的应用程序，两者之间的其他东西就是你需要集成的目

标。找出集成点和集成方式。留住这张图并保持更新，你会经常用到它。

- **如何部署？**如何迁移现有数据？开发团队日常部署到什么环境？UAT（用户验收测试）和性能测试的环境由谁部署？生产环境由谁部署？部署的频度如何？开发和测试使用的数据集从何而来？数据迁移是部署过程的一部分吗？除了自动化部署脚本之外还需要什么额外的环节吗？
- **团队的技能是否与项目需求匹配？**

**攻克技术难题**

作为“tech”lead，解决技术上的难题、为团队扫清前进道路，自然是题中应有之义。由于具备对系统全局架构的了解，tech lead 能够识别出项目中可能遇到的技术挑战，及时进行研究和实验，尽力使其不对开发工作造成阻碍。

主要关注点：

- **集成点是否得到妥善处理？**能成功集成吗？用于集成测试的环境到位了吗？出错的情况妥善处理了吗？
- **如果技术栈中的某一部分出现问题怎么办？**项目中用到的第三方组件有谁熟悉？如果是开源软件，相关的社群在哪里？如果是闭源软件，我们能得到其生产厂商的技术支持吗？

**确定设计方案**

敏捷项目中，大部分设计都是在项目过程中做出的，其中有些设计会对系统整体产生较大的影响，有时甚至会改变起初的架构设计。所以贯穿整个项目，tech lead 需要保持对系统整体和细部的把握，选择适当的设计方案，或是通过重构让好的设计浮现出来。

主要关注点：

- **代码中是否出现明显的 bad smell？**是否看到明显的大类、长方法或者重复代码？条件逻辑嵌套太深了吗？本应属于模型的逻辑在视图或者控制器里堆积了吗？是否应该考虑做一做对象健身操？（参见《ThoughtWorks 文集》，第 6 章）
- **局部设计是否会对系统造成明显的损害？**这个设计会严重损害性能吗？它是一个安全漏洞吗？它是难以测试的吗？它损害了系统遵循的概念一致性吗？
- **团队是否正在进行关于设计和重构的讨论？**

要在局部设计上具备发言权，tech lead 就不能脱离开发工作——不写代码的人是无权评价代码的。在讨论细部设计时，tech lead 需要把握一个微妙的平衡：既不要过于民主而耗费太多时间，也不要过于集中而使团队成员失去学习和思考的机会。一个好的实践是：每天早上把所有开发者召集起来，用 15 ~ 20 分钟浏览前一天编写的所有代码，这样每个人都有机会在看到 bad smell 时指出。

**流程监督人**

为了保障开发工作顺畅进行，tech lead 需要从以下三方面入手来保持对开发流程的关注：开发环境、持续集成和测试。

**开发环境**

Tech lead 需要保障良好的开发实践得到贯彻，尤其是当团队中有较多缺乏经验的成员时这一点就更显重要。一家公司内部很可能有一些成熟的开发套路，这也使得每个 tech lead 的责任更重大：如果团队成员在一个项目中没有养成好习惯，受损害的不仅是这一个项目，还有整个公司的习惯套路。

主要关注点：

- 如何使开发团队拥有统一的环境？
  IDE 的设置和快捷键如何同步？shell 设置和别名如何同步？数据库设置如何保持一致？需要将开发环境虚拟化吗？
- 如何实现一段代码并提交？
  开发者从哪里获取验收条件？编写代码时应该遵循哪些惯例？功能完成之后是否需要邀请 BA/QA 快速确认？提交代码之前应该如何进行本地构建和测试？提交时的注释格式有什么要求？
  ThoughtWorks 的很多 Ruby on Rails 项目都采用“rake commit”这个任务来提交代码。这个任务会从 svn 服务器更新代码、执行本地构建、运行测试、要求开发者输入符合格式的注释内容、然后提交修改内容。我们通常还会给这个命令指定一个 shell 别名“rc”，这样我们只需要敲三下键盘就可以开始一次标准的提交。
- 如何结对编程？
  哪些任务需要结对？哪些任务可以不必结对？结对的轮换合理吗？是否有谁在某一个任务上做了太长时间？是否有谁对某一部分完全不了解？结对过程中大家都全心投入了吗？是否需要用特别的结对方法来引导经验较少的团队成员？需要开发者和 QA 结对吗？

Tech lead 同样需要与团队成员结对，有时是为了理解某一部分的实现，有时是为了指导和帮助队友。Tech lead 应该始终牢记自己是开发团队的一份子，每天都应该尽量安排出时间参与结对；同时其他开发者也应该谅解 tech lead 还有其他职责，不要因此拒绝和他结对。

**持续集成**

每个人都需要对持续集成负责，有一个人需要负更多的责，这个人就是 tech lead：他要清楚持续集成中每个阶段的用意，当集成失败时他要知道这意味着什么。关于持续集成的设计，我强烈推荐 Dave Farley 的文章“一键发布”（《ThoughtWorks》文集，第 12 章）。

主要关注点：

- 持续集成环境和生产环境有多大差异？
- 持续集成覆盖哪些阶段？
  项目各方的关注点在持续集成中都有体现吗？性能测试被覆盖到了吗？打包部署呢？拿到持续集成产出的安装包，我们一定有信心将它部署到生产环境吗？
- 持续集成给开发团队快速而有用的反馈了吗？
  经常失败的是哪些阶段？随机失败的概率大吗？随机失败会掩盖真正的问题吗？从提交代码到走完所有集成阶段通常需要多长时间？需要使用更大规模的并行集成吗？（例如引入更多的 Cruise Agent。）

**测试**

测试就是开发者要满足的目标。没有良好的测试，就等于没有良好的目标。作为 tech lead，要关注不仅是单元测试，还包括功能测试和各种非功能性需求的测试；不仅是开发者编写的测试，还包括 QA 乃至客户的测试。当然，从技术的角度出发，tech lead 更关注的还是自动化的测试。

主要关注点：

- 团队如何实施 TDD？
  测试的粒度和层面合理吗？是否有适当的系统测试？是否有滥用系统测试取代单元测试的倾向？修复 bug 时先用测试来描述 bug 了吗？集成点都有测试覆盖吗？数据迁移都有测试覆盖吗？
- 功能测试/验收测试的质量和进度如何？

QA 和开发者如何借助功能测试沟通？如果功能测试由 QA 编写，能赶上开发的进度吗？测试代码的质量如何？如果由开发者编写，测试覆盖到所有验收条件了吗？QA 能否理解和维护测试代码？是否需要安排 QA 和开发者结对编写功能测试？

- 性能测试得到足够关注了吗？
  有清晰的性能需求吗？有性能测试描述这些需求吗？如何得到性能测试的结果？
  我的同事 James Bull 在“实用主义的性能测试”（《ThoughtWorks 文集》，第 14 章）一文中对性能测试的做法有精彩的论述，此处不再赘述。

**干扰过滤器**

在大部分软件项目中，开发者的工作——细部设计和编程实现——都位于关键路径上：它们未必是最有价值的工作（尽管我个人坚持这样认为），但它们一定是最耗时间的工作。换句话说，开发者的时间是否充分用于开发，将决定项目能否按时交付。所以，tech lead 的很大一部分责任就是过滤各种干扰，使开发者们全神贯注地编程。尽管项目经理和 BA 也起到这样的作用，但还是有很多编程之外的事需要“技术人员”来做。

**与客户技术团队沟通**

大多数定制开发项目都会涉及到客户方的技术团队：开发、测试、DBA、运维、支持，等等。光把系统做好还不够，你还得把做好的系统交到他们手上，项目才真算完成——“交到他们手上”这件事就得由 tech lead 来负责。

主要关注点：

- 如何与客户的开发团队交换知识？
  有哪些惯例需要遵循？有哪些既有的工具可以利用？如果双方开发者同时开发，如何协作？如何结对编程？如果只需要在开发结束后移交产品，如何做知识传递？如何确保对方保持关注？如果双方不在同一地点，是否需要安排定期的电话会议或视频会议？

- 如何与客户的支持团队协作？
  生产或 UAT 环境的服务器由谁管理？如何交付部署包？客户的测试人员如何进行测试？测试结果如何获取？
  运维团队在软件项目中常常被忽视，但他们对产品的成功上线至关重要。通过搭建和维护 UAT/性能测试环境，可以尽早地让运维团队参与到项目之中，并且了解部署和日常管理的相关信息，从而使最终的上线变得相对容易。

- 如何与客户的 DBA 协作？

数据库迁移计划经过 DBA 复审了吗？DBA 是否能获得系统运行时的数据库访问日志？如何与 DBA 讨论解决数据库相关的问题？

一个好的实践是在每次部署到 UAT 环境之前将近期的数据库改变总结出来告知 DBA，并请他持续关注 UAT 环境被使用时的数据库日志。如果 DBA 指出一些明显性能低下或者有其他问题的 SQL，应该重视他的意见。

**与 BA/QA 协作**

团队内部的“干扰源”主要是 BA 和 QA。BA 和 QA 往往缺乏技术背景 如果他们经常用一些“愚蠢”的问题去打断开发者的工作，开发者们可能会觉得他们添乱多过帮忙。这时 tech lead 就得表现出更多的耐心，先过滤掉那些没有营养的问题，从而让开发者们觉得与 BA/QA 的沟通是有帮助的。

主要关注点：

- Story 的内容和工作量估计合理吗？

  Story 涉及的功能实现起来有多困难？是否有更简单的方式来实现同样的目标？相关的风险大吗？

  理论上，工作量估算是由开发者来做的。但有两种原因使得 tech lead 需要代表开发者来做这件事：（1）找开发者做估算可能打断他们的工作节奏；（2）项目初期可能其他开发者不了解情况，甚至还没有加入项目。同样，如何尽量让所有开发者都感到自己的意见得到尊重，又不过多占用他们的时间，这也是需要平衡的。而代表开发者做估算的准确度也将直接影响 tech lead 在他们心里的地位。

- QA 的工作需要帮助吗？

  QA 发现 bug 时会如何处理？他发现的 bug 经常是“误报”吗？需要帮助他编写自动化测试吗？需要帮助他做性能测试吗？

**打杂**

其他所有需要由“技术人员”来做的事，tech lead 都应该有自己一肩挑的准备——例如查一下数据库里有哪些不符合业务规则的数据，生成一份 CSV 文件，小小改动一下界面，甚至倾听一下客户和项目经理的抱怨再给他们一点“技术性”的安慰，等等。

但这不表示你就总是应该把这些事一肩挑。Tech lead 这个角色的微妙之处就在于：他仍然是“技术人员”，和所有的开发者一样。换句话说，tech lead 能做的事，其他开发者也应该能做。所以，再一次地，你应该有一个平衡：把一部分杂事自己消化掉，不让它们干扰开发者的正常工作；另一部分杂事则分配给开发者去做，让他们感到自己除了编程之外还参与了项目的

其他方面，同时也给自己挤出一点时间做其他更重要的事。

**小结**

在我所经历过的大部分项目里，tech lead 都是个超级忙的家伙——看这篇文章就不难理解为什么。人们期望 tech lead 做的事很多，而且在项目的各个阶段还有所不同，一个人要肩负这样的期望确实不容易。

不过只要意识到“tech lead”只是一顶很多开发者都可以戴的帽子，随着项目的进展，你就可以慢慢地把更多的责任放在整个团队的肩上——只要不至于造成损害，于是自己也有了更多的时间来编程。最终你可能会得到一支这样的团队：其中每个开发者相当平均地扮演一部分 tech lead 的角色，各种任务随优先级被有效地处理。这时你就可以说，这个项目的 tech lead 确实做好了他的工作。

**原文链接：**http://www.infoq.com/cn/articles/thoughtworks-practice-part8

**相关内容：**

- 环境无关的环境
- Mock不是测试的银弹
- “持续集成”也需要重构
- 为什么我们要放弃Subversion
- 如何在敏捷开发中做好数据迁移

推荐文章 | Articles

# 并发与不可变性

作者 Dhanji Prasanna 译者韩锴

对于今天的应用程序来说，并发是一个重要的、也愈发受到关注的方面。随着交易量的增加、业务日趋复杂，对大量并发线程的需求也越来越急迫。另外，由依赖注入管理的对象在应用程序中的其角色也极为关键。 Singleton 就是典型的这种需求。

对于一个每分钟需要处理几百个请求的大型 Web 应用来说，如果 Singleton 设计得很糟糕，它会成为严重的瓶颈，以及系统的并发性能的短板，甚至在一些特定的条件下，会导致系统失去可伸缩性。

糟糕的并发行为可能比你想象的要普遍。并且，由于它们产生的影响只有在性能测试期间才会暴露出来，这使得识别和解决这些问题变得更加困难。因此，研究 Singleton 与并发的关系就变得很重要了。

“可变性”是这个问题的一个关键要素。“不可变性”的理念背后也有很多隐晦的陷阱。所以，我们首先讨论一下所谓的“不可变”到底是指什么。为了直接切入问题，我们将以一系列谜题的方式来探讨。

**不可变性谜题#1**

下面的 Book 类是不可变的么？

```
public class Book {
   private String title;
   public String getTitle() {
      return title;
   }
   public void setTitle(String title) {
      this.title = title;
```

```
    }
}
```

**答案#1**

这个问题的答案很简单：不是。只要调用 setTitle()就可以任意修改 title 域的值。所以这个类不是不可变的。如果将 title 声明为 final 的，就可以让 Book 类成为不可变的，如下所示：

```
public class ImmutableBook {
   private final String title;
   public ImmutableBook(String title) {
      this.title = title;
   }
   public String getTitle() {
      return title;
   }
}
```

一旦在构造函数里面设置了 title 的值以后，就不能再改变它了。

**不可变性谜题#2**

下面的类是不可变的么？

```
public class AddressBook {
   private final String[] names;
   public AddressBook(String[] names) {
      this.names = names;
   }
   public String[] getNames() {
      return names;
   }
}
```

**答案#2**

域 names 的值是 final 类型，只会在构造函数中设置一次。所以 AddressBook 应该是不可变的，对不对？错！事实上，容易混淆的地方在于 names 是一个数组，将它声明为 final 只会

使它的引用成为不可变的。 下面的代码完全是合法的，但它却潜在地破坏了数据。而这正是多线程程序所担心的问题：

```
public class AddressBookMutator {
  private final AddressBook book;

  @Inject
  public AddressBookMutator(AddressBook book) {
     this.book = book;
  }

  public void mutate() {
     String[] names = book.getNames();
     for (int i = 0; i < names.length; i++)
        names[i] = "Censored!"
     for (int i = 0; i < names.length; i++)
        System.out.println(book.getNames()[i]);
}
}
```

虽然 names 域是不可改变的，但是方法 mutate()会破坏性地改写数组。如果运行了这段程序，AddressBook 中的每个名字都变成了“Censored（已篡改）”。解决这个问题的真正方法是避免使用数组，或者在真正理解它们以后非常谨慎地使用。更好的办法是使用容器类库（比如 java.util），这样能够用一个不可修改的封装类来保护数组的内容。参看谜题 3，它示范了用 java.util.List 取代数组。

**不可变性谜题#3**

下面的 BetterAddressBook 类是不可变的么？

```
public class BetterAddressBook {
  private final List names;

  public BetterAddressBook(List names) {
     this.names = Collections.unmodifiableList(names);
   }
  public List getNames() {
     return names;
```

```
 }
}
```

**答案#3**

谢天谢地，没错，BetterAddressBook 是不可变的。Collections 类库中的封装类可以确保一旦设置了 names 的值，就不能对它再有任何更新。下面的代码虽然可以编译，却会在运行时导致异常：

```
BetterAddressBook book = new
BetterAddressBook(Arrays.asList("Landau", "Weinberg",
"Hawking"));

book.getNames().add(0, "Montana");
```

**不可变性谜题#4**

下面是谜题 3 的变体，仍然使用我们前面的见到的 BetterAddressBook 类。是否存在某种构造方法，使得我仍然可以在构造以后修改它？前提是不允许修改 BetterAddressBook 的代码。

答案非常简单，只是有点儿混乱：

```
List physicists = new ArrayList();
physicists.addAll(Arrays.asList("Landau", "Weinberg",
"Hawking"));
BetterAddressBook book = new BetterAddressBook(physicists);
physicists.add("Einstein");
```

现在遍历 BetterAddressBook 的 names 列表：

```
for (String name : book.getNames())

System.out.println(name);
```

恩，看来，我们必须要重新审视谜题 3 中的答案了。只有满足了 names 列表没有泄露到 BetterAddressBook 类以外的前提条件，BetterAddressBook 才是不可变。更好的方法是，我们能够重写一个完全安全的版本：在构造的时候复制一份列表：

```
@Immutable

public class BestAddressBook {

   private final List names;
   public BestAddressBook(List names) {
      this.names = Collections.unmodifiableList(new ArrayList
(names));
   }
   public List getNames() {
      return names;
   }
```

```
}
```

现在，你可以随意泄露甚至修改原来的列表了：

```
List physicists = new ArrayList();
physicists.addAll(Arrays.asList("Landau", "Weinberg",
"Hawking"));


BetterAddressBook book = new BetterAddressBook(physicists);


physicists.clear();
physicists.add("Darwin");
physicists.add("Wallace");
physicists.add("Dawkins");

for (String name : book.getNames())
   System.out.println(name);
```

...同时 BestAddressBook 不会受到任何影响：

```
Landau

Weinberg

Hawking
```

尽管你不必每次都使用这么小心的方法，但是如果你无法确保参数是否可能泄露到其他的对象中去，那么建议你使用这种方法。

**不可变性谜题#5**

下面的 Library 类是不可变的么？（调用了谜题 1 中的 Book）

```
public class Library {

   private final List books;

   public Library(List books) {
      this.books = Collections.unmodifiableList(new
ArrayList(books));
   }
   public List getBooks() {
      return books;
   }
}
```

**答案#5**

Library 依赖于一个 Book 列表，不过它非常小心地先复制一份列表，然后把它封装在一个不

可变的包装类中。当然它唯一的域也是 final 的。每件事看起来都无懈可击了？事实上，Library 其实是可变的！尽管 Book 的容器是不变的，但是 Book 对象自身却不是。回忆一下谜题 1 中的场景，Book 的 title 可以被修改：

```
Book book = new Book();
book.setTitle("Dependency Injection")
Library library = new Library(Arrays.asList(book));
library.getBooks().get(0).setTitle("The Tempest"); //mutates Library
```

不可变性和对象图的黄金规则是每一个被依赖的对象也必须是不可变的。在 BestAddressBook 中，我们很幸运，因为 Java 中的 String 已经是不可变的了。在声明一个“不可变”的对象以前，仔细地检查它依赖的每一个对象也都是安全不可变的。在谜题 4 中见到的@Immutable 标注可以帮你传达这一意图，并将它记录到文档中。

**原文链接**：http://www.infoq.com/cn/articles/dhanji-prasanna-concurrency

**相关内容**：

- 使用并发与协调运行时
- Java 6 中的线程优化真的有效么？
- Java 6 中的线程优化真的有效么？——第二部分
- Concurrent Basic——基于消息并发的声明式语言
- SAP创始人鼓吹高并发计算和面向列的数据库

推荐文章 | Articles

# 使用绑定实现灵活通信

作者 Anthony Elder 译者 马国耀

SCA 最重要的特征之一是对交互协议的广泛支持。如果你的服务要同 Web 服务、JMS、CORBA 或者 REST 交互的话，通过 SCA 和 Tuscany 就可以轻松做到。如果因为特定应用的需要，服务使用特殊的或私有的协议进行通讯的话，SCA 也没有问题。甚至，你的业务逻辑根本不需要知道交互协议是什么，（没错，你已经猜对了）协议的选择是通过对组件的配置实现的。这太酷了，不是吗？而 SCA 绑定就是让这一切成为可能的“魔力”。

在这篇文章中，我们将了解如何在服务以及引用上使用绑定；此外，如果没有配置绑定话，意味着什么；最后，我们还要去看看 SCA 的域，去了解如何在 SCA 域内和域外运用绑定。

**为服务和引用配置绑定**

你可以在服务以及引用上配置绑定。为服务设定一个绑定意味着人们可以通过由该绑定所指定的交互协议来访问这个服务。而服务的实现不需要做任何特别的改变来促成这个目标，你要做的仅仅是为服务增加一条绑定配置。

例如，如果要让 Bookings 服务以 Web 服务的方式暴露出来，那么组件的定义看起来就应该是这样的：

```
<component name="TripBooking">
  <implementation.java
class="com.tuscanyscatours.TripBooking" />
  <service name="Bookings">
    <binding.ws
uri="http://tuscanyscatours.com:8085/Bookings" />
  </service>
</component>
```

我们通过增加<binding.ws>元素告诉 SCA 运行时，Bookings 服务要以 Web 服务的形式暴露出

来，使用 SOAP/HTTP 的 方式，并且以 uri 属性中指定的值作为服务的端点（ endpoint ）。这就是要创建 Web 服务我们要做的全部事情，无需学习 JAX-WS 或者在 Bookings 服务的实现上做文章。

可以为一个服务配置多个绑定。如果既要通过 JMS 访问又要通过 Web 服务方式来访问 Bookings 服务，则只需要增加另一条绑定。请看下面的例子：

```
<component name="TripBooking">
  <implementation.java
class="com.tuscanyscatours.TripBooking" />
  <service name="Bookings">
    <binding.ws
uri="http://tuscanyscatours.com:8085/Bookings" />
    <binding.jms />
  </service>
</component>
```

这个定义让 Bookings 服务同时以 JMS 的方式暴露出来，并且使用了缺省配置。同样，我们不需要去学习 JMS 的 API 或修改服务的实现代码。

绑定还将 SCA 和 Tuscany 与更广阔的外部世界联系起来！Bookings 服务是以 SCA 实现，运行在 Tuscany 中的。由于使用绑定对这个服务进行了配置，它可以被以非 SCA 方式实现的客户端或运行在 Tuscany 中的客户端调用。对于 Bookings 的 Web 服务调用，客户端可以是任何语言编写的，运行在任何遵循 WS-I 的 Web 服务运行时上的程序；同样，对于 JMS 方式的调用，客户端可以直接使用任何 JMS 提供者所提供的 JMS API，只要该 JMS 提供者与 Tuscany 中配置的 JMS 绑定对应的 JMS 提供者兼容即可。

现在我们已经看到如何通过在 SCA 的服务上配置绑定让服务以标准的交互协议向外提供服务。同样，绑定也可以用于 SCA 引用上，让其通过标准的交互协议去调 用外部服务。这种情况下角色正好倒置：客户端由 SCA 实现，而服务端使用标准的交互协议。例如，服务提供者可能是一个支持 SOAP/HTTP 的 Web 服务 端点，也有可能是一个以 RMI-IIOP 方式交互的 EJB（ session bean ）。与 Web 服务交互时，SCA 引用使用<binding.ws>绑定；而调用 EJB 时，它使 用<binding.ejb>绑定。

图 1 显示了如何在组件的服务以及引用上配置绑定。

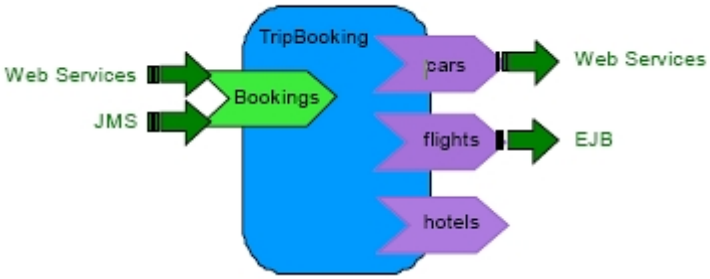



**图 1：Bookings 服务配置了 Web 服务和 JMS 的绑定，“cars”引用配置了 Web 服务绑定，“flights”引用配置的是 EJB 绑定。**

列表 1 中的示例代码显示的是图 1 中描绘的 TripBooking 组件对应的组件定义文件。

**列表 1 为组件的服务和引用配置绑定和连线**

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   targetNamespace="http://tuscanyscatours.com/"
   name="bookings">
    <component name="TripBooking">
      <implementation.java
class="com.tuscanyscatours.TripBooking" />
      <service name="Bookings">
   <binding.ws
uri="http://tuscanyscatours.com:8085/Bookings" />
   <binding.jms />
      </service>
      <reference name="cars">
   <binding.ws uri="http://tuscanycars.com:8081/Cars" />
      </reference>
      <reference name="flights">
   <binding.ejb
uri="corbaname:rir:#flight/FlightProviderHome" />
      </reference>
      <reference name="hotels" target="HotelProvider" />
    </component>
</composite>
```

你可能已经发现列表 1 中有两个引用（cars 和 flights）配置了绑定而没有 target 属性，这是

因为该绑定元素已经提供了引用的目标端点信息，同时还指定了所使用的交互协议。

至此，我们已经了解如何在服务和引用上使用绑定。在下一节中，我们要看看如果在服务和引用的配置中不指定绑定将会发生什么？

**缺省绑定**

在列表 1 中我们为“hotels”引用没有设置绑定，这意味着它有一个隐含的.sca 绑定（经常被称为缺省绑定）。缺省绑定用于连接 SCA 服务和 SCA 引用，把交互技术的选择工作交给部署服务和引用的 SCA 运行时，而其他的绑定（如 WS 绑定和 JMS 绑定）则要选择具体的交互协议或 API，从而让 SCA 服务或引用可以与非 SCA 的程序进行交互。正因为如此，非缺省的绑定通常被称为可互操作的绑定。

Tuscany 使用基于 SOAP/HTTP 的 Web 服务实现缺省绑定的远程调用 其他的 SCA 运行时可能使用不同的标准协议，如 RMI/IIOP，它们也 可以使用某种私有协议。将来，Tuscany 的缺省绑定的通讯协议也可以从 Web 服务转向其他的协议，因此，Tuscany 的应用程序不应该想当然地假设 使用缺省绑定就意味着使用 Web 服务。应用程序若要使用 Web 服务在组件之间交互的话，为保险起见，还是应该要指定使 用<binding.ws>绑定。

缺省绑定只能用于连接位于同一个 SCA 域中的服务和引用，而当连接跨越域边界时，应该要为其指定某个可互操作的绑定。域在 SCA 中是一个重要的概念，我们将在第四章详细介绍域的概念，不过在下一个节中我们先简单介绍一下什么是域以及它与绑定和连线的关系。

**域，绑定和连线（wire）**

SCA 域是一个 SCA 组件的部署和管理边界，比如，一个域可能是单个应用服务器或一个服务器集群，也可以是一组服务器或一组集群。而一个完整的域通常只运行某一家提供商的 SCA 实现。

任何 SCA 组件都是 SCA 域的一部分；同一域中的服务和引用可以通过连线进行连接；对于同一域中的服务和引用之间的连接，可以使用缺省绑定，因为 SCA 保证了缺省绑定的实现在域中的一致性。相反，对于域内到域外的交互，不能使用连线进行连接，而应该使用可互操作的绑定。

在第 2 章我们已经介绍了 TuscanySCATours 公司，下面我们将通过该公司的一个场景来描述域的使用。现在，这个公司已经壮大，并且设立了一个独 立的部门提供专门的 hotel booking（酒店预订）服务。这个服务不仅要服务于 TuscanySCATours 的 trip booking（行程预订）服务，还要直接服务于只需预订酒店的客户。公司的两个部门使用不同的域管理他们所提供的

服务：TuscanySCATours 域提供原先的 trip booking（行程预订）服务，而 TuscanySCAHotels 域提供新的 hotel booking（酒店预订）服务，如图二所示：

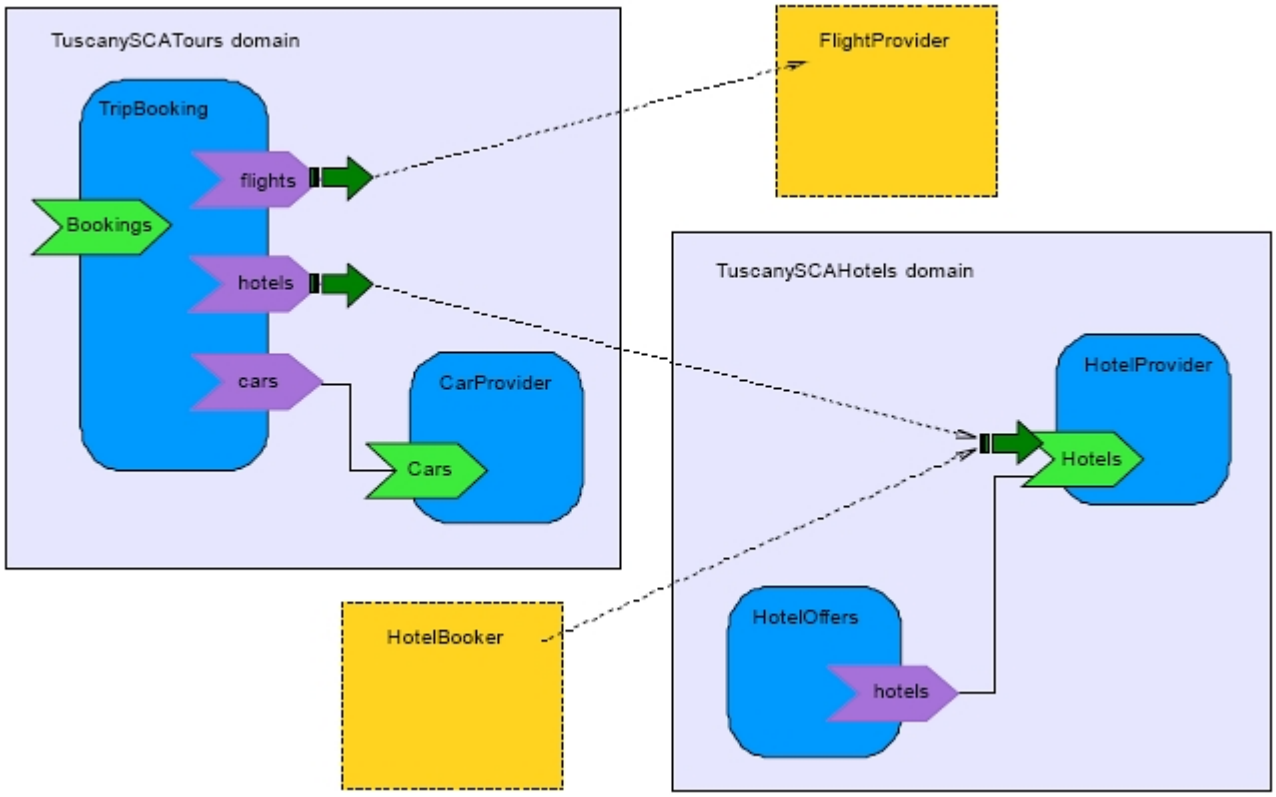



**图 2 中有两个 SCA 域及多个组件，展示了组件是如何连接的，连线（实线）用于连接同一域中的组件，而 绑定（带箭头的虚线）用于连接跨域的组件。**

图 2 中用实线表示连接组件的 SCA 连线，而用带箭头的虚线表示用可互操作的绑定配置的连接。我们先来看看 TuscanySCATours 域中 TripBooking 组件的 3 个引用。首先，flights（航班）引用连到一个实现 flights 预定的 Web 服务，因为这个服务不是 SCA 服务，所以 flights 引用通过可互操作的绑定<binding.ws>和 flights 预订服务的 URI 进 行配置。

其次，“hotels”引用指向了 TuscanySCAHotels 域中的由 HotelProvider 这个组件所提供的 Hotels 服务。由于连线不能 跨越域的界限，所以这样的服务和引用都要使用<binding.ws>和服务端点 URI 进行配置。最后，“cars”引用连到了 TuscanySCATours 域中的由 CarProvider 组件提供的 Cars 服务，因为这对引用和服务位于同一个域中，所以我们可以使用连线以及缺 省绑定连接它们。“Cars”服务没有对外暴露，也不需要和外部的非 SCA 程序交互，所以没有必要为其配置可互操作绑定。

接下来，看看 TuscanySCAHotels 域中由 HotelProvider 组件周围的连线。除了来自 TripBooking 组件的“hotels” 引用连到它之外，还有来自 HotelBooker（非 SCA 的酒店预订客户端程序）

和 TuscanySCAHotels 域内的 HotelOffers 组件 的“hotels”引用。由于“Hotels”服务配置了客户操作的绑定类型——<binding.ws>，所以 HotelBooker 客户 端软件可以使用任何 Web 服务程序调用“Hotels”服务。HotelOffers 和 HotelProvider 之间的连接使用 SCA 连线的原因是这些 组件处于同一域中。这个连线既可以使用缺省绑定也可以使用该服务提供的可互操作 binding.ws 绑定，在本例中，我们使用 binding.ws，目的 是为了说明同一域中的连线也并不一定要使用缺省绑定。

了解图 2 中所描述的场景在组件配置文件中是如何配置是很有必要的。列表 2 给出了位于 TuscanySCATours 域中的组件的定义。

**列表 2 TuscanySCATours 域中组件的组件定义文件**

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   targetNamespace="http://tuscanyscatours.com/"
   name="toursdomain">
    <component name="TripBooking">
      <implementation.java
class="com.tuscanyscatours.TripBooking" />
     <reference name="flights">
   <binding.ws
uri="http://flightbookingservice.com:8084/Flights" />
     </reference>
     <reference name="hotels">
    <binding.ws
uri="http://tuscanyscahotels.com:8083/Hotels" />
     </reference>
     <reference name="cars" target="CarProvider/Cars" />
    </component>
    <component name="CarProvider">
       <implementation.java
class="com.tuscanyscatours.CarProvider" />
    </component>
</composite>
```

列表 3 给出了 TuscanySCAHotels 域中组件的组件定义文件：

**列表 3 TuscanySCAHotels 域的组件定义文件**

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   targetNamespace="http://tuscanyscahotels.com/"
   name="hotelsdomain">
    <component name="HotelProvider">
      <implementation.java
   class="com.tuscanyscahotels.Hotels" />
      <service name="Hotels">
   <binding.ws uri="http://tuscanyscahotels.com:8083/Hotels"
/>
      </service>
    </component>
    <component name="HotelOffers">
      <implementation.java
class="com.tuscanyscahotels.HotelOffers" />
    <reference name="hotels"
   target="HotelProvider/Hotels" >
   <binding.ws/> 1
      </reference>
    </component>
</composite>
```

1 此处的引用使用 binding.ws 进行配置

为 HotelOffers 的“hotels”引用指定引用绑定是有必要的，如果没有，它就可能被设定成默认的 binding.sca 绑定，而这样会导致错误，因为它（“hotels”引用）的目标服务的配置中没有提供 binding.sca 方式的绑定。

图 2 中的例子展示了引用以及服务的大多数连接方式。不过，对于同一个域中的 SCA 服务和 SCA 引用，除了使用 SCA 连线之外，也可以使用一对匹配的绑定进行连接。很多 Tuscany 的例子就是这么做的，这样很容易展示如何为客户端和服务端配置匹配的绑定。

我们已经描述了 SCA 服务和引用连接的若干种选择，这里简要做一个总结。在同一个域中连接引用和服务，你可以：

- 使用带缺省绑定的 SCA 连线
- 使用 SCA 连线，并为其配置一对匹配的可互操作的绑定

- 为引用和服务配置匹配的绑定

连接不在同一域中的 SCA 引用和服务，你可以：

- 为引用和服务配置匹配的可互操作的绑定

将引用或服务连接到非 SCA 程序，你可以：

- 为应用或服务配置一种可互操作绑定

可互操作绑定使 SCA 程序能够与非 SCA 程序交互，也支持不同提供商的 SCA 程序跨域的交互。而缺省绑定的好处是在互操作不是必须的情况下，提供商可以提供更优质的交互能力。

若了解源代码，示例章节，作者的论坛以及其他资源，请访问http://www.manning.com/laws。

**原文链接：**http://www.infoq.com/cn/articles/elder-etal-tuscany

**相关内容：**

- SCA访谈
- “我能以后再调用你吗？”使用SCA开发异步服务
- 向服务组件架构出发
- 《Open Source ESB in Action》作者谈开源ESB
- 通过邮件服务器集成应用

推荐文章 | Articles

# 敏捷背后

作者 Mukund Srinivasan 译者 金毅

**山雨欲来**

作为一个组织，当我们一年前首次邂逅 Scrum，第一感觉就相信它是管理层所期望做到的领域之一，因为对于工程方面运作方式的改变势在必行。那时，感觉减缩开支并不是非“敏捷”不可，可能只要些不一样的做法，而且这种方式也必然经历了当时“彻底的改革之风”潮流的洗礼。

**改革之风**

一年飞逝，我们公司从经历的改变中获益匪浅，这一切要归功于我们新上任的分管工程的副总裁，是他给我们和 Scrum 牵了线，搭了桥。有了这一扎实的根基，我们可以来吹捧些好处了（看，我也能说得很草根！）：

- 能更好地对不断变化的市场和投资者们的市场要求做出回应——特别是当我们正处在金融海啸中。
- 能尽早遇到困难 ——我们对此感激不尽，因为不管你遵循什么过程都会有苦难 ，对我们每个人来说，越早经历越好。
- 有优先级、有重点的执行（一段时间内）。问任何经理那是否值得，十有八九会以“除非发生了变化，计划就是计划” 这句引言做回答。陈述本身并没有错，但把责任和职责一起交到产品所有人手里是在我们组织有过的最棒的事。
- 自我管理团队：怎样理解主动承担责任这种方式比要求团队做出承诺并坚守它好呢？

一年前，我们团队中也不是完全没有这些；只是当初很多都错过了，而现在每个团队都能做到所有这些。为了避免这篇文章落入宣扬敏捷益处的套路，我想快点言归正传——毕竟如果撇开已有的敏捷转变的结果不谈的话，已经有很多有资历的专家在谈那点了。

## 面纱背后

我想要写下这些并非由于我看到了团队中那些可见的变化，而是来自一些无形的东西。转变的面纱背后，可以这么说，人际关系中感观价值上的改变，更强调信任的必要性，解决多地点交流不够高效的迫切需求等，这些都是我想要一探究竟的。过去一年，在我点击和转发的数以千计的链接中，在那些你能想到的任何类型和格式的共享文档中，还不到 5 %（或二十分之一）能触及到这个话题，我渐渐发现它是我们转变中的一个副产品。面对信息海洋，我不记得从哪儿找到这个，但从去年我就深深地记住了这个引用——“实事求是是敏捷的基础”。一个同事后来推荐给我一个链接，能够更好地解释这一点。

这并不意味着我所有的同事和朋友们以前都不诚实，而一夜之间就改邪归正；恰恰相反，事实上，在职业生涯中我还没遇到过有足够的证据说谁故意不诚实的。其实，依我的拙见，当需要一个人完全诚实的时候，环境会造就或破坏一个人对此的反应。此前我天真地以为——不诚实，是非此即彼的；在某段时间内，你只能是道德高尚的或者道德败坏的，诚实的抑或不诚实的。有些不符合社会道德准侧的杂念也许会在我们的记忆中逐渐消失，就像我们人类倾向于仅仅记住快乐的事情一样。

言归正传，我提到的诚实和真实是抽象概念，是我们对自己以及身边人的发自内心的承诺。举个简单例子，如果在几年前我还是一名架构师的时候，有人要求我给新任务做一个粗略估算，我会毫不犹豫。然而现在，我会考虑再三。但现在除了我的工程估算能力有所改进，什么都没改变；另外层意思就是我会根据“我是否真的确定，我的承诺能代表整个团队？”来做出判断。当我看到一个团队成员在竭尽所能地兑现团队的共同承诺时，我会想要去停止一切的讨价还价，我不想做出一个他 / 她不能兑现的承诺。如果你仔细看，界线模糊了，角色被合并了——懂得协作的团队将走到最后！在敏捷转变之前，我们可能已经对周围的人做了个承诺，但是由于各种外界原因，那些最紧迫的活动的范围并没有立即去做，随后又不得不去面对由此引发的一系列反应。有了敏捷和 Scrum，由于一切困难都与我们正面遭遇，由于我们想要成为群体中的一部分（羊群心理？），我们会想要自觉地做出一个承诺 ——一个我们能够兑现的，不被周围人孤立的约定。

## 运用到日常生活中去

想象一下，如果在你的生活圈中，每个人都坦承说出自己的承诺，并对此供认不讳，老实说，这会给其他人施加一种“同辈压力”，逼迫他们也去实现自己的承诺。我有个信心十足的想法，在我们的影响范围之内，我们能够控制我们的命运——这意味着对我们自己的承诺不应该有借口。我读过的某篇文章说，鉴于我们把大量的时间用在了工作上，我们的个人生活只是我们工作的衍生。限于我使用 Google 或其他搜索引擎的能力，我没能推荐一篇关于怎样把在

工作中采用的敏捷开发方法论转化到运用到工作以外的生活中，从而让生活更有质量的文章。但可以考虑一下这个想法，如果我们在工作之外，个人生活中，采纳了相同的原则，那会是给我们孩子不错的一课吗？在当今社会，已经不能像记忆中那样单靠一件事情就能名垂青史，这与几个世纪前的战争年代，很有可能一夜成名有所不同。今天我们能留下的不再是英勇的行为或事迹，而实际上是一个人职业生涯所做的并能够持续影响未来二、三十年的那些贡献。

如果这意味着一个人职业生涯的转变能够直接给社会带来正面影响，这就很值得庆祝了。如果我们有机会在我们的生活中来做个榜样，并因此史上留名（小范围地），这得感谢背后的敏捷，对此，我很赞同——我的朋友们，关键就是背后的敏捷。

是的，敏捷有一个方面还没有在我们的日常生活中得到运用——“生活是一场马拉松，不是冲刺。”

**原文链接：**http://www.infoq.com/cn/articles/hidden-face-of-agile

**相关内容：**

- 诚实——是敏捷的价值观吗？
- Steven "Doc" List 谈开放空间会议
- 敏捷应对“团队的五重机能障碍”
- 案例分析：荷兰铁路公司的分布式Scrum开发
- 郭晓谈实效敏捷和敏捷在中国的发展

推荐文章 | Articles

# 基于Facebook和Flash平台的应用架构解析

作者 Abel Avram 译者 马国耀

Flash 平台可帮助你构建富用户体验的应用，而 Facebook 平台可帮助你构建富社会化体验的应用。将二者合而为一，你就可以构建高交互性、富于表现力，并融入了社会化功能的杀手级应用了。

本系列文章（共分三部分）将为你介绍基于 Facebook 和 Flash 平台的应用程序架构，解析你能在此平台上构建的各种应用类型，并说明这些应用如何与你的服务器、Facebook 服务器通讯。

**可构建的应用类型**

你可构建三种 Flash 与 Facebook 平台集成的应用：基于 Facebook 的嵌入式应用、对外服务的 Web 应用和桌面应用。

- **基于Facebook的嵌入式应用**，部署在你自己的服务器上，但其用户通过Facebook站点访问。 用户看到的是一个将你的应用包容其中的Facebook容器。你访问Facebook上的某个应用（通常是因为收到朋友邀请，或搜索某个应用时得到一个链接）时，Facebook服务器会将请求转发给你的应用服务器，得到一个页面（HTML和Javascript代码），最后在Facebook站点上展示出来。这种方式对于访问Facebook站点的用户而言，提供的体验是无缝的。这类应用的例子很多，比如来自Playfish和Zyng德克萨斯扑克的谁有最聪明的大脑应用。
- **提供对外服务的独立Web应用**，也部署在你自己的服务器上，但用户通过你提供的URL而不是Facebook站点来访问该应用。在外部站点中，你可以通过Facebook API和Facebook Connect来增加Facebook的特性。比如利用Facebook API实现用户登录，应用在一个新的浏览器窗口中打开Facebook登录页面，用户必须在这里登录后，才能访问你本身的应用。为了避免在Facebook站点上登录，为用户提供无缝体验，你就需要使用Facebook Connect了。比如，某用户阅读一篇博客后，可能想写点评论。那么不必让用户在你的站点上再

注册一个账号吧，用他/她在Facebook上的帐户就好了。为提升用户体验，Facebook API和Facebook Connect还允许你访问用户的全部数据，比如在评论旁边显示评论者的姓名和个人图片。类似的如购物网站，通过让用户用他们的Facebook帐户登录，你可以查看其朋友是否推荐或评论过什么商品。这类的应用如RedBull Connect和City Search，在这些网站上你可发表评论、阅读朋友的评论，或将自己的评论发表到Facebook Wall或朋友的新闻种源中。

- **桌面应用**，除了基于 Flash 平台的桌面 AIR 这个特点外，其他和对外服务的 Web 站点是大致相同的。AIR 桌面应用同样是借助 Facebook Connect 为用户提供无缝的 Facebook 登录体验。这类应用的例子有 Seesmic 和 Nomee。

在开发Facebook应用之前，必须在Facebook开发者应用上注册，获得API授权和应用的密钥。至于具体步骤，请参考构建你的第一个Facebook应用快速入门。在注册过程中，需要你指定要开发应用的一些设定，例如应用是基于Web还是桌面的，是否使用FBML或iFrame。

- 开发基于 Facebook 的嵌入式应用时，你需指定为 Web 应用类型，并选择使用 iFrame 或 FBML。
- 开发对外服务的独立 Web 应用时，你需指定 Facebook Connect 信息。
- 开发桌面应用时，你需指定为桌面应用类型。

现在，在我们深入分析基于 Flash 平台的嵌入式应用、对外 Flash 平台站点和 Flash 桌面应用的架构前，先让我们大致了解一下常规嵌入式的、非 Flash 的、基于 iFrame 和 FBML 的 Facebook 应用的架构。

**iFrame Facebook 应用**

当用户访问Facebook的某个应用（比如http://apps.facebook.com/someapp）时，Facebook对此请求的处理方式，与应用是基于iFrame还是FBML有关。如果是iFrame应用，Facebook服务器返回的页面包含一个Facebook容器，容器容纳一个iFrame，你的应用就在其中加载（如图1）。

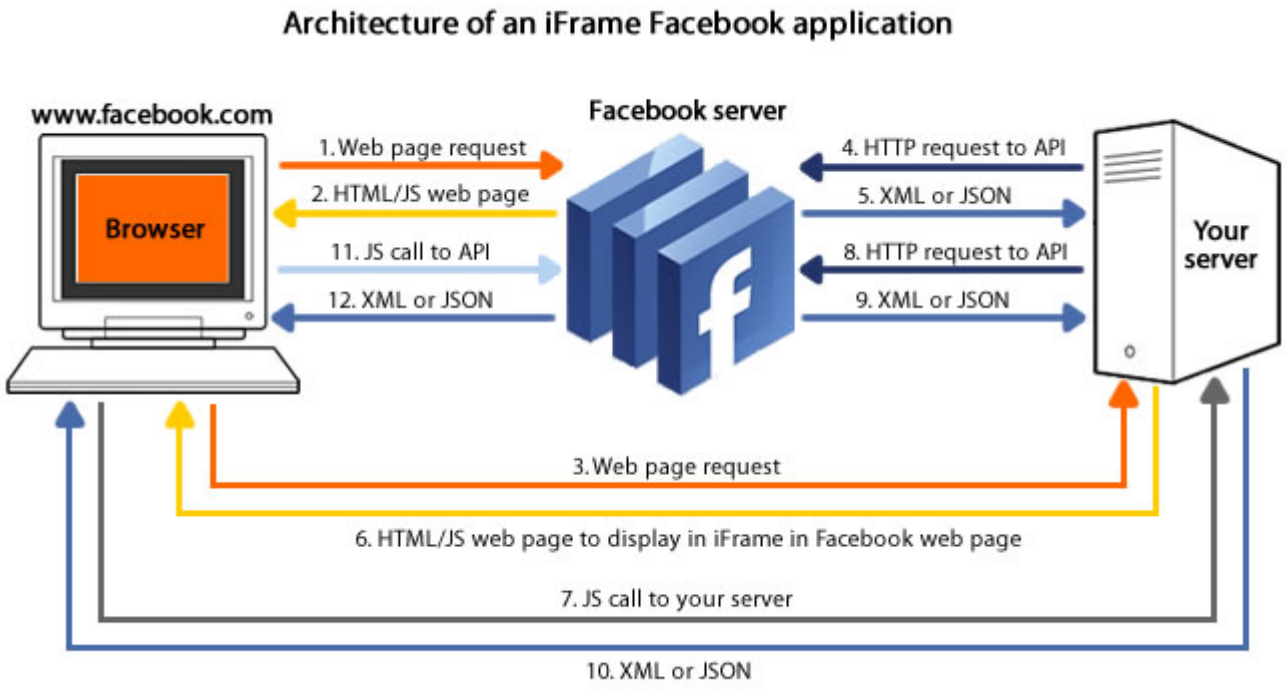



**图 1 iFrame 应用架构**

1. 当用户在 Facebook 网站上访问你的应用时，浏览器会向 Facebook 服务器发送一个 HTTP 请求。

2. Facebook 服务器返回一个 HTML/JavaScript(JS)页面，其中包含了 Facebook 站点容器和一个 iFrame HTML 标记。

3. 用户的浏览器向你的服务器请求将显示在 iFrame 中的页面( 一般是一个 PHP、ColdFusion 或者 JSP 式的服务端页面 )。Session 信息会通过 GET 请求中的 URL 参数传递，这样你的应用服务器就知道此请求是来自 Facebook，以及请求是哪个用户发出的。

4. 服务端页面执行时，可能会根据需要访问数据库或其他服务器，其中也包括通过REST API向Facebook服务器发出请求。调用REST API时，必须包含认证信息，比如在Facebook上注册应用时获得的API Key、该调用的签名（通过传给Facebook方法的参数、用户请求你的应用时指定的Session的MD5 哈希串生成）、应用的密钥和其他信息。通常来说，服务端页面会利用标准的代码库生成对Facebook的请求，并且其签名也在服务端脚本中产生。尽管Facebook官方只提供了一个服务端用户库( 支持PHP 5 )，不过其他的服务端库已由社区开发了很多。另外，Facebook官方还提供了两个客户端库，分别支持JavaScript和ActionScript 3.0。

5. Facebook 服务器将被请求的数据（XML 或 JSON 格式）返回到你的服务器。

6. 你的服务器向用户浏览器返回 HTML/JS 页面，并由客户端浏览器显示在 iFrame 中。在用户和你的应用交互时，交互行为包括：

   - 如果你的应用包含了新的服务端页面请求，将重复步骤 3-6。
   - 如果你的应用包括对你的服务器的 JavaScript 异步调用，那么步骤 7-10 将被执行。和上面有所不同的是，此时向用户浏览器返回的通常是 XML 或 JSON 数据，由页面的 JavaScript 脚本负责处理。
   - 还有一种情况，是页面中的JavaScript代码直接访问Facebook服务器，而不通过你的应用服务器中转（步骤 11-12）。Facebook官方提供了对应的JavaScript API库。利用这个库，你可将多个API调用打包，通过一个HTTP请求向Facebook服务器发出。这项技术有利有弊。好处是减少了总的HTTP访问次数，缺点是导致页面大小和复杂度的上升。

提示：你也可以通过 XFBML 技术，在你的应用中放置一些简单的 FBML 标签（具体会在下一个部分中讨论）；当然为了使用这些标签，你必须用 JavaScript 代码扫描标签的 DOM，然后也可以将一批 API 请求组织成向 Facebook 服务器的一次调用。

**FBML Facebook 应用**

现在，你已经对 iFrame 应用的工作原理有了一个大致把握，接下来我们讨论 FBML Facebook 应用（见图 2）。在前面，你的应用是存在 iFrame 中的一个独立实体，现在，它将变成 Facebook 服务器响应请求时返回的 HTML 页面的一个部分。在这种情况下，Facebook 服务器会代为处理所有对你的应用的调用。当然，除了返回 HMTL 和 JavaScript，你的服务端页面也可以返回 FBML 代码，Facebook 服务器在将它返回给用户浏览器前，会自动将其转换为 HTML 和 JavaScript 代码。

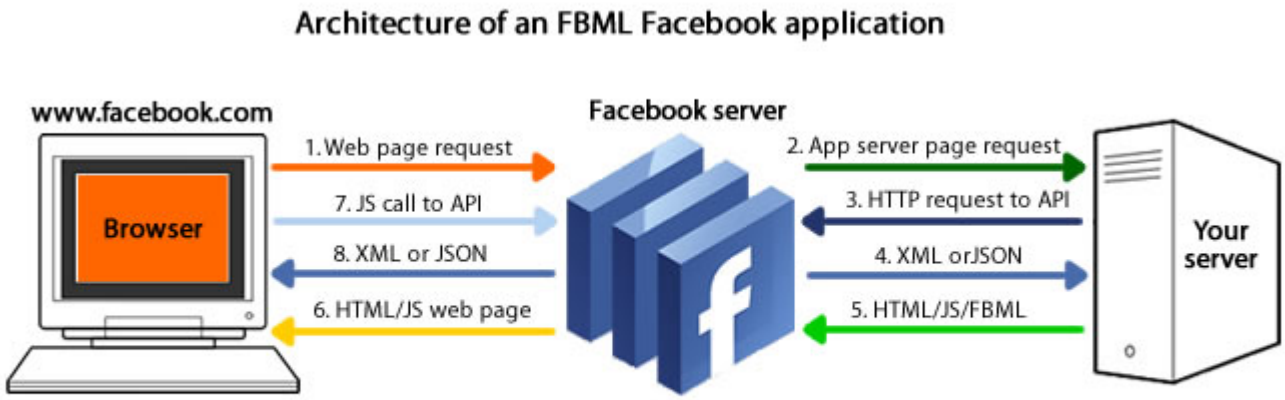



**图 2 FBML Facebook 应用架构**

1. 当用户在 Facebook 网站上访问你的应用时，浏览器会向 Facebook 服务器发送一个 HTTP 请求。

2. Facebook 服务器将请求转给你的服务器，一般来说都是请求一个服务端页面（如 PHP、ColdFusion 或 JSP）。在这种情况下，Session 信息将通过 POST 请求（而 iFrame 通常是 GET 请求）中的 URL 参数传递，这样你的应用服务器就知道此请求来自 Facebook，以及请求的发送用户是谁。

3. 服务端页面执行时，可能会根据需要访问数据库或其他服务器，其中也包括通过 REST API 向 Facebook 服务器发出请求。调用 REST API 时，必须包含认证信息，比如在 Facebook 上注册应用时获得的 API Key、该调用的签名（通过传给 Facebook 方法的参数、用户请求你的应用时指定的 Session 的 MD5 哈希串生成）、应用的密钥和其他信息。

   和iFrame应用一样，服务端页面通常都借助一个代码库生成对Facebook的请求和签名。因为所有返回都是通过Facebook代理的，你的应用请求Facebook服务器时，就没必要每次单独调用一个API。

   FBML提供了大量 标签，可用于获得用户姓名、图片、创建对话框和小组件等。对于这类要求，你只需要直接返回FBML代码，后面的工作留给Facebook服务器就可以了，它在将页面返回给用户浏览器前，会自动将FBML转换为HTML和JavaScript代码。当然，不是所有功能都有标签支持的，比如要取得朋友的生日信息，还是得从你的服务器向Facebook发送对应的API调用请求。

4. Facebook 服务器向你的服务器返回被请求的数据，格式是 XML 或 JSON。

5. 你的服务器向 Facebook 服务器返回 HTML/JS/FBML 页面。
6. Facebook 服务器将 HTML/JS 页面返回给用户浏览器。在用户和你的应用交互过程中产生的交互行为包括：
   - 如果你的应用包括新的服务端页面请求，重复步骤 1-6。
   - 不同于请求新的页面，你应用程序中的JavaScript可通过使用官方提供的JavaScript库直接向Facebook服务器发出请求（同上面iFrame讨论中的 7-10 步骤）。

提示 :虽然在 FBML 应用中 ,你也可以向你的应用服务器发出异步请求( 同图 1 的步骤 7-10 ), 但这些调用必须位于通过<fb:iFrame> 标签在 iFrame 里加载的内容中。

**Flash iFrame Facebook 应用**

至此，我们已经介绍了 iFrame 和 FBML 的情况，接下来开始讨论在应用中如何集成 Flash 内容的问题。虽然你完全可以构建一个包含了 HTML/JavaScript/ActionScript 的混合应用，但为了说明的方便，我们还是将注意力集中在基本的 Flash 应用层面——其全部视觉效果和功能都封装在 SWF 文件（Flash Player 可识别并渲染的、编译后的字节码格式）中。

在上述混合应用中，可以使用到多种 API 调用，比如服务端 API、前面已讨论过的客户端 JavaScript API，以及这里将讨论的客户端 ActionScript API 调用。

你可将 SWF 文件集成到 iFrame 或 FBML 应用中。图 3 说明了在 iFrame 应用中的简单实现方法。

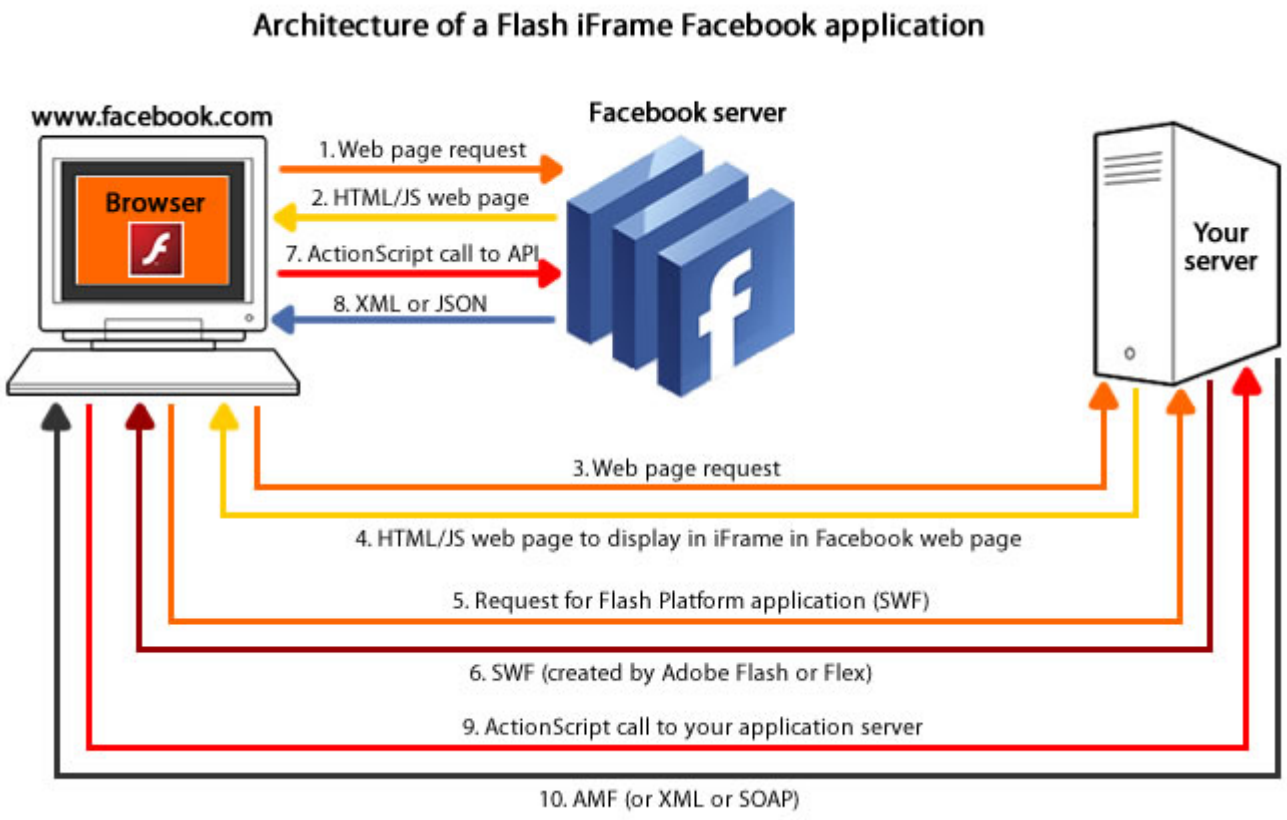



**图 3 Flash iFrame Facebook 应用**

1. 用户在 Facebook 站点上访问你的应用时，浏览器向 Facebook 服务器发出一个 HTTP 请求。
2. Facebook 服务器返回一个 HTML/JS 页面，其中包含 Facebook 站点容器和一个 iFrame HTML 标签。
3. 用户浏览器向你的服务器请求包含在 iFrame 中的页面。和前面讨论过的情况不同，这个页面不再是一个服务端页面，而是一个简单的 HTML 页面（内含 SWF 文件）。此时，Session 信息仍通过 GET 请求的 URL 参数传递；你的服务器解析这些参数，并转换为内嵌的 SWF 所需的 flash 参数，这样，在 SWF 中的 ActionScript 代码就可以根据这些参数，直接向 Facebook 服务器发出请求了。
4. 你的服务器将 HTML/JS 页面返回给用户浏览器，并在 iFrame 中展示。
5. 用户浏览器向你的服务器发出其他请求，即请求展示在 iFrame 中的 HTML 页面中内嵌的 SWF 文件。
6. 你的服务器返回 SWF 文件。

当用户和你的应用交互时，SWF 可向 Facebook 服务器、或你的服务器发送异步请求。

- SWF文件中的ActionScript脚本直接访问Facebook服务器（步骤 7-8 ）。你可以使用Google代码中的ActionScript 3.0 Library for Facebook Platform。

  出于Flash Player安全性的考虑，SWF文件只能从两类服务器获取数据：（ 1 ）提供SWF文件的服务器（这里即你的服务器）；（ 2 ）有跨域策略文件（在文件中列出了SWF来源服务器）的服务器。也就是说，若要你的SWF能直接访问Facebook服务器，Facebook服务器必须在跨域策略文件中开放了访问权限。如果看过Facebook的跨域策略文件，你会发现它通过一个通配符，授予了来自任何服务器的SWF文件的访问权：

  ```
  <cross-domain-policy>
  <site-control
  permitted-cross-domain-policies="master-only"/>
  <allow-access-from domain="*"/>
  </cross-domain-policy>
  ```

- 你的 ActionScript 访问 Facebook 服务器时，必须像前面非 Flash 的 iFrame 和 FBML 应用部分讲到的那样，传送应用 API Key 和用于说明访问来自何处的签名信息。利用 ActionScript 3.0 Library for Facebook Platform 中的类可自动生成签名；为此，你只需向 ActionScript session 类的构造函数传入应用的 API Key 和密钥。

  但是，你不应在 SWF 文件以硬编码方式写入上述数据，因为 SWF 文件可用多种软件反编译。相反，SWF 应该在运行时向你的服务器发出请求（可 HTTP 或 Flash Remoting 方式），从而获得应用的密钥，接下来再传给 ActionScript session 类的构造函数，从而生成访问 Facebook 服务器时所需的签名。

  记住，此签名由下列信息组成：传递给 Facebook 服务器的 URL 参数、Session Key（用户访问你的应用时分配）的 MD5 哈希串、应用的密钥等。

- 若需实现任何服务端处理功能（如在你的服务器上保存某些数据），可在 ActionScript 代码中通过远程过程调用方法实现（见步骤 9-10 ）。对于利用 Flex 构建的 Flash 平台应用而言，这些方法包括 HTTP、Web Service 和 Flash Remoting 请求技术。其中最便捷的方法当属 Flash Remoting——它通过开源的二进制 Action Message Format (AMF) 实现服务器和 Flash Player 间的数据交换。当然，服务端代码在向客户端返回数据之前，可根据需要访问 Facebook 服务器（此点未包含在图 3 中 ）。

**Flash FBML Facebook 应用**

除了在 iFrame 中集成你的 Flash 应用 ,还可以通过 FBML 标签<fb:swf>将其植入到 FBML 应用中。在这种情况中，你的应用会由一或多个服务端页面组成；这些页面包含 FBML（用于获取或展示 Facebook 形式的可视化内容、对话框和小控件）、不能通过简单的 FBML 标签实现的服务端 API 调用，以及包纳了你的 Flash 内容的<fb:swf>标签。

FBML应用的好处是在访问Facebook提供的社会化特性时非常简单，向Facebook传送FBML标签，就可自动渲染为HTML内容。而其不足，则是此处在多个地方编写表现层和逻辑层的代码更为复杂：SWF文件中、服务端页面中，以及发向Facebook服务器的FBML代码中（见图 4）。有关iFrame和FBML应用区别的更详细信息，请参看iFrame、FBML Flash Facebook应用比较。

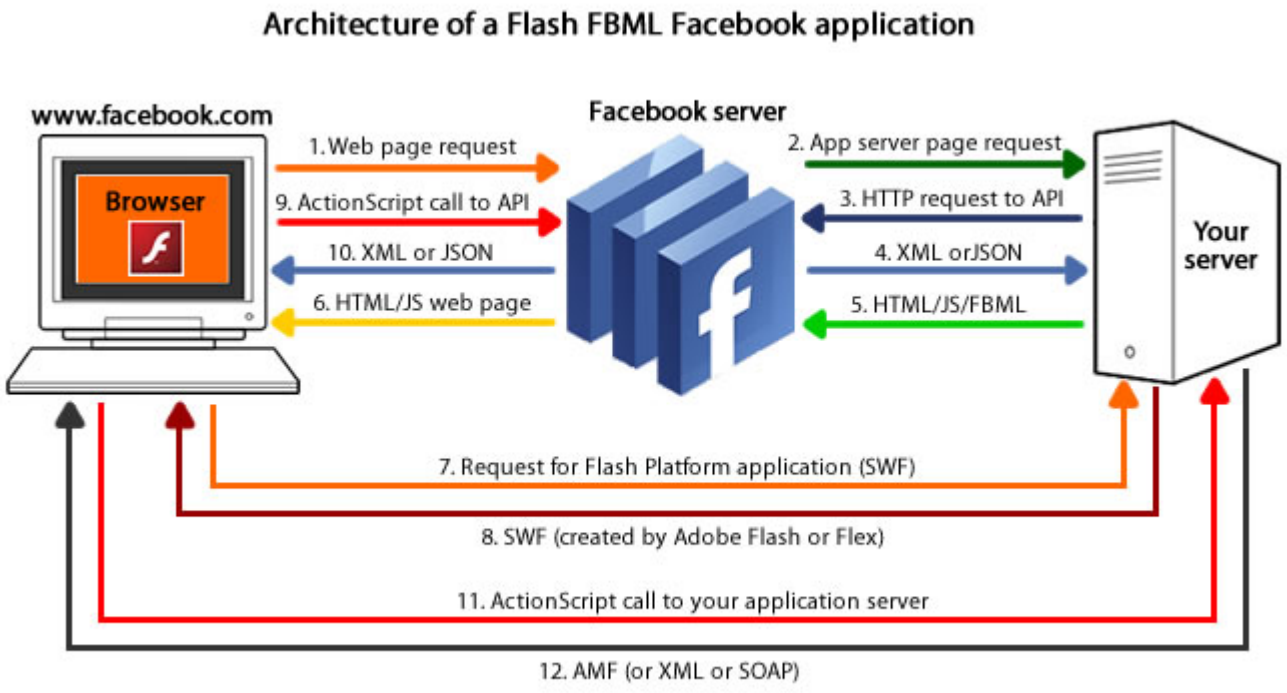



**图 4 Flash FBML Facebook 应用**

1. 用户通过 Facebook 站点访问你的应用时，浏览器向 Facebook 服务器发送 HTTP 请求。
2. Facebook 服务器向你的服务器发出请求——一般请求的是 PHP、JSP 或 ColdFusion 式的服务端页面。在这种情况下，Session 信息通过 POST 请求（而非 iFrame 中的 GET 请求）中的 URL 参数传递，那么，你的应用服务器就知道该请求来自 Facebook、是哪个用户发出的请求了。
3. 服务端页面执行时，可根据需要访问数据库和其他服务器，当然包括利用 REST API 访问 Facebook 服务器。

API 调用必须包含认证信息——包括在 Facebook 上注册应用时获得的 API Key、该调用的签名（通过传给 Facebook 方法的参数、用户请求你的应用时指定的 Session 的 MD5 哈希串生成）、应用的密钥和其他信息。

对于非Flash的iFrame和FBML应用来说，服务端页面通常可利用现成的代码库实现对Facebook的访问，以及生成签名。因此对于所有FBML应用而言，用户浏览器的所有请求都是通过Facebook代理的，你的应用在请求Facebook服务器时，就没必要每次单独调用一个API；同时，诸如获得用户姓名、图片、生成对话框等等，都可以利用FBML 标签实现。

Facebook 服务器在将页面返回给用户浏览器前，会自动将 FBML 转换为 HTML 和 JavaScript 代码。当然，不是所有功能都有标签支持的，比如要取得朋友的生日信息，还是得从你的服务器向 Facebook 发送对应的 API 调用请求。

4. Facebook 服务器将被请求数据返回给你的服务器，格式可为 XML 或 JSON。
5. 你的服务器向 Facebook 服务器返回包括 HTML/JS 和 FBML 内容的页面。
6. Facebook 服务器向用户浏览器返回 HTML/JS 页面。该页面包括了（用标签<fb:swf>指定）对你的服务器上 SWF 文件的引用。
7. 用户浏览器向你的服务器请求内嵌在 HTML 页面中的 SWF 文件。
8. 你的服务器返回 SWF 文件。

在用户和你的应用交互过程中，SWF 可向 Facebook 服务器或你的服务器发出异步调用。

- SWF中ActionScript代码直接访问Facebook服务器（见步骤 9-10），这可利用宿主在Google代码上官方支持的ActionScript 3.0 Library for Facebook Platform实现。当然，Facebook必须通过跨域策略文件开放了访问权限，且API调用中传送了所需参数。有关这部分的详细问题，前面的Flash iFrame应用中已有讨论。
- 若需实现任何服务端处理功能（如在你的服务器上保存某些数据），可在 ActionScript 代码中通过远程过程调用方法实现（见步骤 11-12）。对于利用 Flex 构建的 Flash 平台应用而言，这些方法包括 HTTP、Web Service 和 Flash Remoting 请求技术。其中最便捷的方法当属 Flash Remoting——它通过开源的二进制 Action Message Format (AMF)实现服务器和 Flash Player 间的数据交换。当然，服务端代码在向客户端返回数据之前，可根据需要访问 Facebook 服务器（此点未包含在图 4 中）。

图 5 描绘了独立的Flash Facebook站点应用的架构。其主要区别是Facebook服务器不再代为处理全部浏览器请求。另外，现在你还必须在客户端代码中用Facebook API 或Facebook Connect处理用户的登录。如果使用Facebook API处理登录，用户需在新的浏览器窗口中登录Facebook并返回到你的应用中。为了避免在Facebook站点登录，为用户提供更无缝的登录体验，你可以使用Facebook Connect。

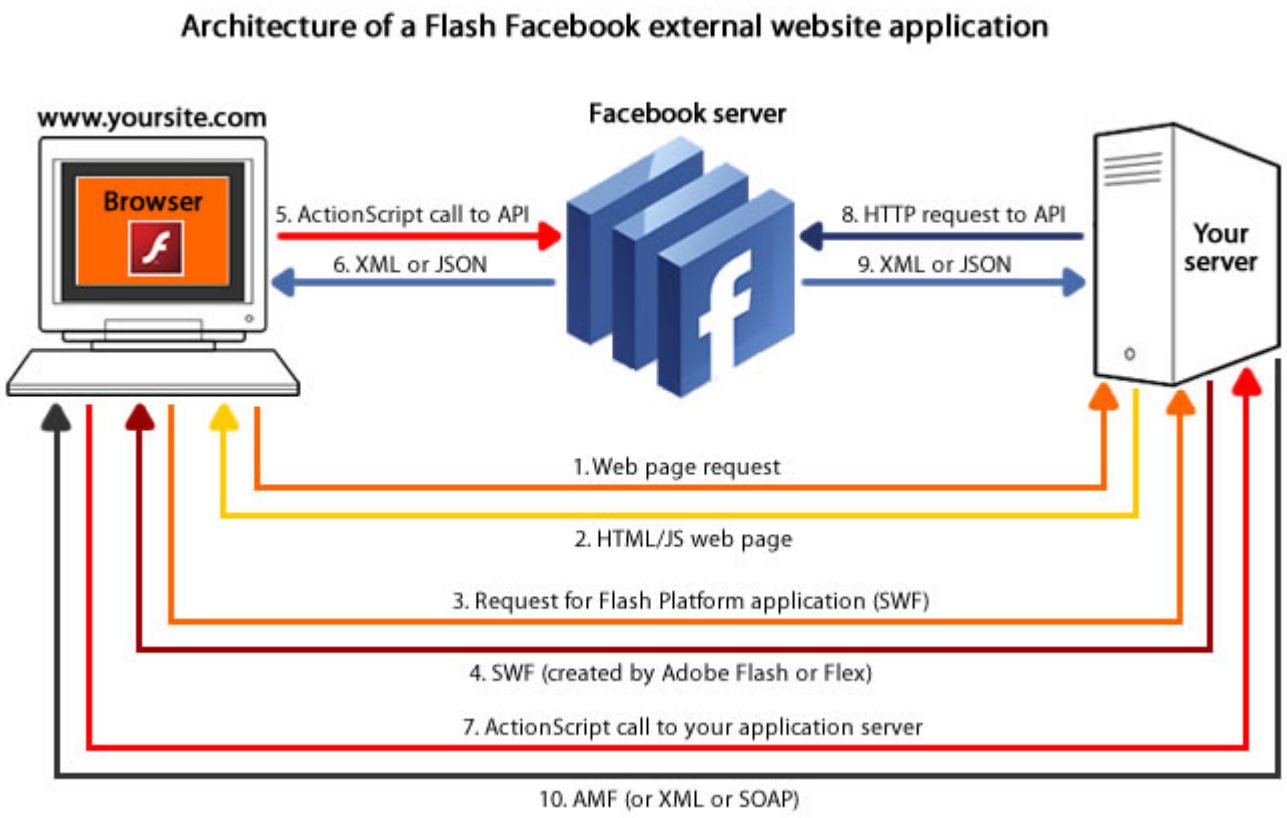



**图 5 独立的 Flash Facebook 站点应用**

1. 当用户在你的站点上访问应用时，浏览器向你的服务器发送 HTTP 请求——请求一个 HTML 或任何服务端页面。
2. 服务器返回包含了对你的SWF文件的引用的HTML/JS页面。如果使用Facebook Connect，该HTML页面会包括部分用于初始化Facebook Connect的JavaScript代码（说明）。
3. 用户浏览器向你服务器请求内嵌在 HTML 页面中的 SWF 文件。
4. 你的服务器返回 SWF 文件。
5. SWF 文件中的 ActionScript 代码直接异步请求 Facebook 服务器——方法是使用官方提供的 ActionScript 3.0 Library for Facebook Platform。你每次可以提交单独一个调用，也可以

提交成批调用。在这种情况下，最初对 Facebook 服务器的调用必须获得授权；一旦用户成功登录（最好使用 Facebook Connect），得到了 Session Key，那么后续所有 Facebook API 调用所需的签名就会由 ActionScript 3.0 Library for Facebook Platform 的类生成。当然，Facebook 必须通过跨域策略文件开放了访问权限，且 API 调用中传送了所需参数。有关此问题的更多信息，请参看前面在 Flash iFrame 应用部分的讨论。

6. Facebook 服务器向你的 Flash 应用返回 XML 或 JSON 格式的数据，并由你的应用处理这些数据。
7. 若需实现任何服务端处理功能（如在你的服务器上保存某些数据），可在 ActionScript 代码中通过远程过程调用方法实现（可以是 HTTP、Web Service 和 Flash Remoting）。其中最便捷的方法当属 Flash Remoting——它通过开源的二进制 Action Message Format (AMF) 实现服务器和 Flash Player 间的数据交换。
8. 若有必要，服务器可与 Facebook 服务器进行其他通讯。
9. 你的服务器处理 Facebook 服务器返回的结果数据。
10. 你的服务器将数据返回给用户浏览器中的 Flash 应用。图 5 中，我们利用 Flash Remoting 和 AMF 交换数据，当然你也可用 Web Service、SOAP、HTTP 实现文本或 XML 格式的数据交换。

**Flash Facebook 桌面应用**

最后，让我们来讨论Flash Facebook桌面应用的架构。基于Flash平台的桌面应用，就是AIR应用（这个地方请再斟酌一下）。有关构建AIR应用的更多信息，请参阅AIR文档和AIR开发者中心。Flash Facebook桌面应用（如图 6）的架构，和前面讨论过的独立Flash Facebook站点应用非常类似，唯一的不同是此时不需要浏览器，SWF文件也存在于安装了AIR应用的用户本地计算机上。

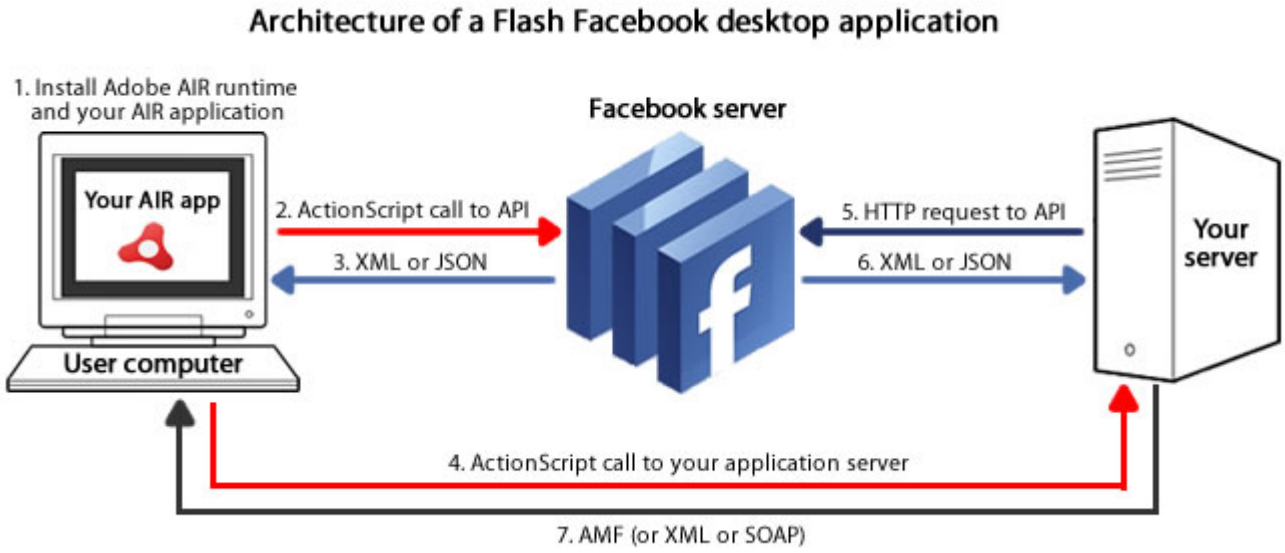



**图 6 Flash Facebook 桌面应用**

1. 用户安装并运行 AIR 桌面程序。
2. SWF文件中的ActionScript 代码直接异步请求Facebook服务器——方法是使用宿主在Google代码上官方提供的ActionScript 3.0 Library for Facebook Platform。你每次可以提交单独一个调用，也可以提交成批调用。在这种情况下，最初对Facebook服务器的调用必须获得授权；一旦用户成功登录（最好使用Facebook Connect），得到了Session Key，那么后续所有Facebook API调用所需的签名就会由ActionScript 3.0 Library for Facebook Platform的类生成。当然，Facebook必须通过跨域策略文件开放了访问权限，且API调用中传送了所需参数。有关此问题的更多信息，请参看前面在Flash iFrame应用部分的讨论。
3. Facebook 服务器向你的 Flash 应用返回 XML 或 JSON 格式的数据，并由你的应用处理这些数据。
4. 若需实现任何形式的服务端处理功能（如在你的服务器上保存某些数据），可在 ActionScript 代码中通过远程过程调用方法实现（可以是 HTTP、 Web Service 和 Flash Remoting）。其中最便捷的方法当属 Flash Remoting——它通过开源的二进制 Action Message Format (AMF)实现服务器和 Flash Player 间的数据交换。
5. 若有必要，服务器可与 Facebook 服务器进行其他通讯。
6. 你的服务器处理 Facebook 服务器返回的结果数据。
7. 你的服务器将结果数据返回给 Flash 桌面程序。图 6 中利用 Flash Remoting 和 AMF 交换

数据，当然你也可用 Web Service、 SOAP、 HTTP 实现文本或 XML 格式的数据交换。

**总结与引申**

本系列文章介绍了三类基于 Flash 和 Facebook 平台的应用：基于 Facebook 的嵌入式应用、Web 站点式的独立应用和桌面应用。对于任何 Facebook 应用，你都可将 Flash 程序包纳在 iFrame 或 FBML 应用中。具体来说，从架构和处理流程角度可分为六种子类型？(本文的架构图和流程处理适用于）基于 Facebook 非 Flash 的 iFrame/FBML 应用 基于 Facebook 的 Flash iFrame/FBML 应用，以及 Flash 站点应用、 Flash 桌面应用。

有关iFrame和FBML应用区别的更多信息，请参考iFrame、FBML Flash Facebook应用比较。有关构架基于Facebook的Flash应用的详细步骤，请观看快速构建Facebook应用视频，或阅读利用Flexible构建Facebook应用快速入门。

**原文链接：**

http://www.infoq.com/cn/articles/facebook-arch-overview-part1

http://www.infoq.com/cn/articles/facebook-arch-overview-part2

http://www.infoq.com/cn/articles/facebook-arch-overview-final

**相关内容：**

- RichClient/RIA原则与实践（上）
- RichClient/RIA原则与实践（下）
- Flex体系架构深度剖析
- 把WPF作为一种富客户端技术
- 表现层模式新探

推荐文章 | Articles

# 使用iTest2 重构自动化功能测试脚本

作者 Zhimin Zhan 译者 金明

## 介绍

众所周知，自动测试脚本很难维护。随着敏捷方法学在企业软件项目中的广泛应用，其核心实践之一——自动化功能测试已经证明了它的价值，同时却也对项目提出了挑战。传统的“录制 - 回播”类型的测试工具也许能帮助测试人员很快地创建一系列的测试脚本，但这些测试代码最后却很难维护。原因就是：应用程序在不断变化。

在编程的世界中，“重构”（在不影响软件外在行为的前提下，改善软件内部结构的一种方法）已经成为程序员之间频繁使用的词汇。简而言之，通过重构，程序员让代码变得更易于理解、设计也更灵活。经验丰富的敏捷项目经理会给程序员分配一定的时间来重构代码，或者把重构作为完成用户故事的一部分。大部分的集成开发环境（IDE）已经对多种重构方式提供了内置支持。

开发或者维护自动测试脚本的测试人员就没有这份惬意了，虽然他们也有使自动测试脚本变得可读和可维护的要求。软件发布新版本，会伴随新特性、bug 修复和软件变更，要想跟踪与之对应的测试脚本，这很难（而且，测试脚本越多，这项工作就越困难）。

## 测试重构

对功能测试的重构目标和流程与代码重构一样，但有自己的特点：

- 目标受众
  测试工具的最终用户包括测试人员、业务分析师，甚至还有客户。事实是测试人员、业务分析师和客户一般都不掌握编程技能，整个范式因此而改变。
- 脚本语法
  代码重构主要是在编译型语言（比如 Java 和 C#）上得到支持。函数式测试脚本，可能是 XML、厂商专有脚本、编译型语言或者脚本语言（比如 Ruby）。根据测试框架不同，

重构的使用形式也不同。

- 功能测试专属重构
  很多通用的代码重构技巧，比如“重命名”，可以用在功能测试脚本里面，它们特定于测试意图，比如“Move the scripts to run each test case”。

**iTest2 IDE**

iTest2 IDE是一款新的功能测试工具，专为测试人员设计，让他们能够很轻松地开发和维护自动测试脚本。iTest2 完全致力于web测试的自动化，它支持的测试框架是使用RSpec语法的rWebUnit（是广为流行的Watir的一款开源插件）。

iTest2 背后的哲学是：容易、简单。试用显示：没有编程经验的测试人员在指导下，平均只需要少于 10 分钟的时间就能编写他们第一个自动化测试脚本。借助于 iTest2，测试人员可以开发、维护和验证功能需求的测试脚本；开发人员可以验证特性可用；业务分析师/客户通过查看测试运行结果（在真实的浏览器下，比如 IE 或者 Firefox）来验证功能需求。

由 iTest2 创建的测试脚本可以从命令行运行，也能集成在持续构建服务器上。

**演练**

事实胜于雄辩。下面我们就来看看如何使用 iTest2 提供的重构工具创建两个测试用例，使它们变得更易理解和维护。

**测试计划**

为了练习，我们给 Mecury's NewTour 网站开发了一些典型但是简单的 web 测试脚本。

| | |
|---|---|
| 站点 URL | http://newtours.demoaut.com |
| 测试数据： | 用户登录：agileway / agileway |
| 测试用例 001： | 一个注册客户可以选择单程航行方式，从纽约前往悉尼。 |
| 测试用例 002： | 一个注册客户可以选择往返方式，从纽约前往悉尼。 |

| | |
|---|---|
| 自动化测试 | |
| 测试脚本框架： | rWebUnit（开源的 Watir 扩展） |
| 测试执行方法： | 通过命令行或 iTest2 IDE |
| 测试编辑器/工具： | iTest2 IDE |

**创建测试用例 001**

**1. 创建项目**

首先，我们创建一个 iTest2 项目，指定网站 URL。一个简单的测试脚本文件就会被创建出来，如下所示：

```
load File.dirname(__FILE__) + '/test_helper.rb'

test_suite "TODO" do
  include TestHelper

  before(:all) do
    open_browser "http://newtours.demoaut.com"
  end

  test "your test case name" do
    # add your test scripts here
  end
end
```

**2. 使用 iTest2Recorder 录制测试用例 001 的测试脚本**

我们使用 iTest2Recorder，这是 Firefox 的一个插件，能录制用户在 Firefox 浏览器中的操作，并记录为可执行的测试脚本。

```
enter_text("userName", "agileway")
enter_text("password", "agileway")
click_button_with_image("btn_signin.gif")
click_radio_option("tripType", "oneway")
```

```
select_option("fromPort", "New York")
select_option("toPort", "Sydney")
click_button_with_image("continue.gif")
assert_text_present("New York to Sydney")
```

**3. 把录好的测试脚本贴到一个测试脚本文件里面，运行**

```
# ...
test "[001] one way trip" do
  enter_text("userName", "agileway")
  enter_text("password", "agileway")
  click_button_with_image("btn_signin.gif")
  click_radio_option("tripType", "oneway")
  select_option("fromPort", "New York")
  select_option("toPort", "Sydney")
  click_button_with_image("continue.gif")
  assert_text_present("New York to Sydney")
end
```

现在运行测试用例（右键单击，然后选择“Run [001] one way trip”），它通过了！

**使用 Page 对象进行重构**

上面的测试脚本可以工作，而且 rWebUnit 语法也非常易读。有人可能对重构的要求提出质疑，也许还会问“使用 Page”是怎么回事？

首先，以现在的格式来看，测试脚本并不易于维护。假设我们已经有了数百个自动测试脚本，而新发布的软件修改了用户认证方式，使用客户邮箱作为用户名登录，这意味着我们需要在测试脚本里面使用‘email’，而不再是‘userName’。在数百个文件里面查找替换，那可不是个好主意。况且，项目成员也喜欢使用项目里面的通用词汇，有一个很美妙的名字来称呼它们：领域专属语言（DSL）。在测试脚本里面也使用这些词汇就太美妙了。

使用 Page 对象能很好地做到这一点。一个我们所说的 Page 对象代表了一个逻辑上的 web 页面，它包含了最终用户在该页面上可以执行的操作。举例来说，在我们例子里面的主页就包含了三个操作：“输入用户名”、“输入密码”和“点击登录按钮”。“使用 Page 对象进行重构”是指把操作抽取到特定 Page 对象的过程，而 iTest2 提供了对这样的重构支持，你可以很容易做到这一点。

### 1. 抽取到 HomePage 对象

登录功能是发生在主页上面，我们把这事交给 HomePage。用户登录是一个很常见的功能，我们用了三行语句（输入用户名、输入密码和点击登录按钮）完成这个操作。选中这三行代码，然后在“Refactoring”菜单下单击“Extract Page...”（快捷键是 Ctrl+Alt+G）。

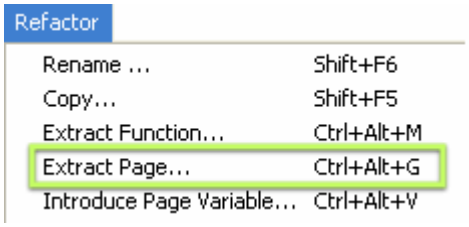



**图 1. “Refactor”菜单——“Extract Page”**

如下图所示，这样会弹出一个窗口，让你输入 Page 对象的名字和功能名。这里，我们分别输入“HomePage”和“login”。

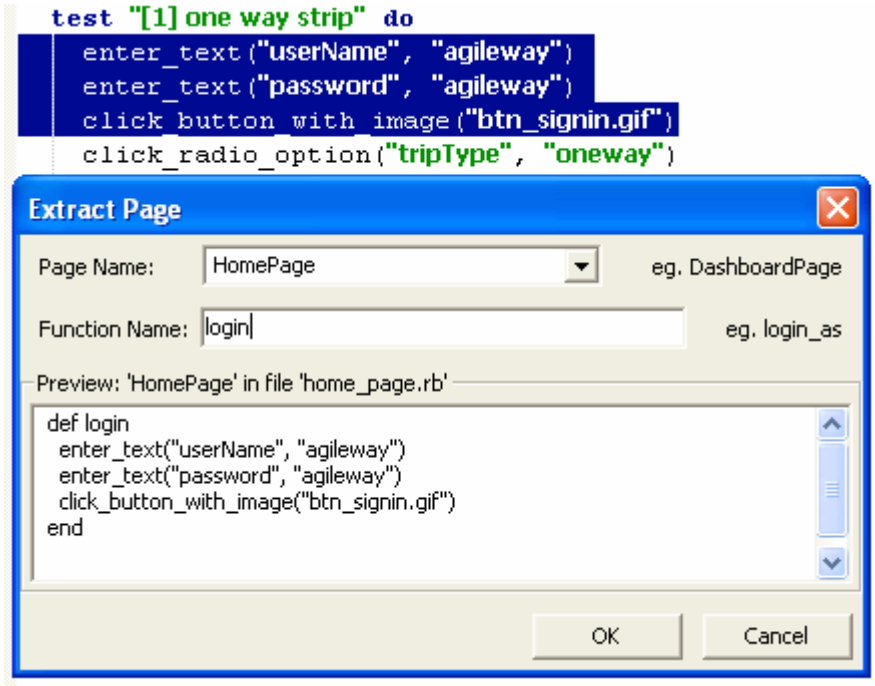



**图 2. “Extract Page”对话框**

选中的 3 行代码就被替换成：

```
home_page = expect_page HomePage
home_page.login
```

这将会自动创建一个新文件“pages\home_page.rb”，其内容如下：

```
  class HomePage < RWebUnit::AbstractWebPage
```

```
  def initialize(browser)
    super(browser, "") # TODO: add identity text (in quotes)
  end

  def login
    enter_text("userName", "agileway")
    enter_text("password", "agileway")
    click_button_with_image("btn_signin.gif")
  end
end
```

再次运行测试用例，它应该还是可以通过。

注意：正如 Martin Fowler 指出，重构的节奏：测试、小的改动、测试、小的改动。正是这种节奏保证了重构的迅速和安全。

**2. 抽取 SelectFlightPage**

登录成功之后，顾客进入了航班选择页面。与登录页面不同，这里的每个操作很可能被不同的开发人员修改，所以我们把每个操作都抽取为一个函数。把光标移到这一行

```
click_radio_option("tripType", "oneway")
```

再次执行“Extract to Page...”重构命令（Ctrl+Alt+G），给新的 Page 对象和函数名输入“SelectFlightPage”和“select_trip_oneway”。

```
select_flight_page = expect_page SelectFlightPage
select_flight_page.select_trip_oneway
```

**3. 继续抽取更多的操作到 SelectFlightPage 对象**

继续把“SelectFlightPage”上的操作重构成函数：“select_from_new_york”、“select_to_sydney”和“click_continue”。

```
test "[1] one way trip" do
  home_page = expect_page HomePage
  home_page.login
  select_flight_page = expect_page SelectFlightPage
```

```
  select_flight_page.select_trip_oneway
  select_flight_page.select_from_new_york
  select_flight_page.select_to_sydney
  select_flight_page.click_continue
  assert_text_present("New York to Sydney")
end
```

跟往常一样，我们再一次运行测试用例。

**编写测试用例 002**

在重构完测试用例 001 之后 我们现在有了 2 个 Page 对象（“HomePage”和“SelectFlightPage”），因此（通过重用它们）编写测试用例 002 会容易很多

**1. 使用已有的 HomePage**

iTest2 IDE 内置支持 Page 对象，输入“ep”再敲“Tab”制表键（称为“snippets”），就能自动补全为“expect_page”并且弹出所有已知的 Page 对象以供选择。

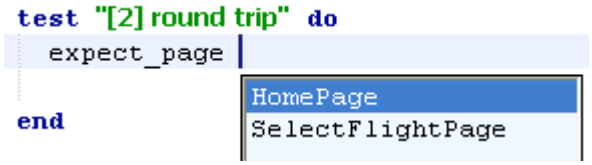



**图 3. 自动补全 Page 对象**

我们就能得到

```
expect_page HomePage
```

为了使用 HomePage，我们需要持有它的句柄（在编程世界中，也被称为‘变量’）。执行“Introduce Page Variable”重构动作（Ctrl+Alt+V）创建一个新变量。

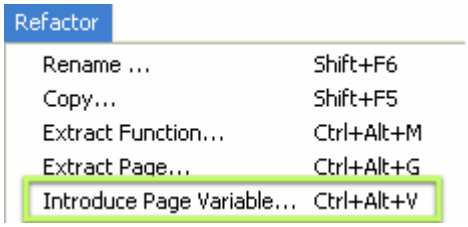



**图 4. ‘Refactor’菜单 - “Introduce Page Variable”菜单项**

```
home_page = expect_page HomePage
```

现在在新行中输入“home_page.”，会自动提示这个 Page 对象中定义的函数供你选择。

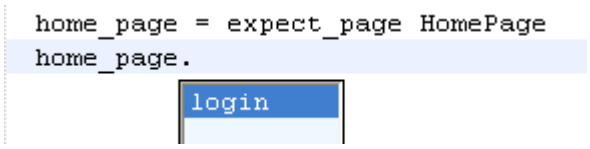



**图 5. Page 对象函数查找**

### 2. 添加测试用例 2 需要的方法

测试用例 002 跟测试用例 001 很像，区别只在于旅行类型的选择和断言。借助于 Recorder，我们可以定义出新的函数：

```
click_radio_option("tripType", "roundtrip")
```

把它重构成 SelectFlightPage 的一个新功能

```
select_flight_page.select_trip_round
```

就变成了

```
test "[2] round trip" do
  home_page = expect_page HomePage
  home_page.login
  select_flight_page = expect_page SelectFlightPage
  select_flight_page.select_trip_round
  select_flight_page.select_from_new_york
  select_flight_page.select_to_sydney
  select_flight_page.click_continue
  assert_text_present("New York to Sydney")
  assert_text_present("Sydney to New York")
end
```

运行测试用例 2 的测试脚本（在测试用例 2 的任意一行之上单击右键，选择“Run ...”），测试也通过了！

### 把应用复原为原始状态

但是等一等，我们还没有完成。测试用例 1 通过了，测试用例 2 也通过了，但是当把它们一起运行的时候，测试用例 2 却失败了，为什么？

我们没有把 web 应用复原回初始状态，在运行完测试用例 001 之后用户还是保持登录的状态。为了让测试之间互相保持独立，我们要确保每次运行测试都要以登录开始，以退出结束，

有始有终。

```
test "[001] one way trip" do
   home_page = expect_page HomePage
   home_page.login
   # . . .
   click_link("SIGN-OFF")
   goto_page("/")
end


test "[002] round trip" do
  home_page = expect_page HomePage
  home_page.login
  # . . .
  click_link("SIGN-OFF")
  goto_page("/")
end
```

**删除重复代码**

测试脚本存在着明显的重复。RSpec 框架允许用户在每个测试用例运行之前或之后执行某些操作。

选中首部两行（登录功能），按下“Shift + F7”以执行“Move Code”重构。

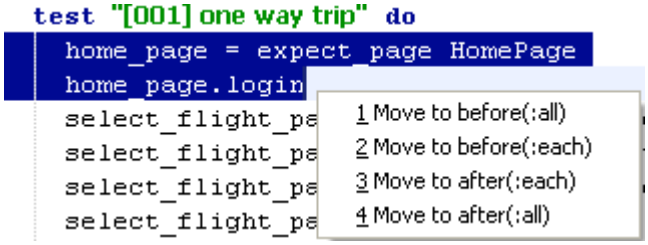



**图 6. 重构菜单“Move code”**

选择“2 Move to before(:each)”，把这部分操作移到

```
before(:each) do
  home_page = expect_page HomePage
```

```
  home_page.login
end
```

正如名字所示，这两步操作会在每个测试用例运行之前执行，所以测试用例 002 里面的前面两行也就没有存在的必要了。我们还可以执行相似的重构，完成“after(:each)”的相关部分。

```
after(:each) do
click_link("SIGN-OFF")
goto_page("/")
end
```

**最终版本**

以下是测试用例 001 和 002 的完整的（经过充分重构的）测试脚本。

```
load File.dirname(__FILE__) + '/test_helper.rb'


  test_suite "Complete Test Script" do
    include TestHelper


    before(:all) do
      open_browser "http://newtours.demoaut.com"
    end


    before(:each) do
      home_page = expect_page HomePage
      home_page.login
    end


    after(:each) do
      click_link("SIGN-OFF")
```

```
    goto_page("/")
  end


  test "[001] one way trip" do
    select_flight_page = expect_page SelectFlightPage
    select_flight_page.select_trip_oneway
    select_flight_page.select_from_new_york
    select_flight_page.select_to_sydney
    select_flight_page.click_continue
    assert_text_present("New York to Sydney")
  end


  test "[002] round trip" do
    select_flight_page = expect_page SelectFlightPage
    select_flight_page.select_trip_round
    select_flight_page.select_from_new_york
    select_flight_page.select_to_sydney
    select_flight_page.click_continue
    assert_text_present("New York to Sydney")
    assert_text_present("Sydney to New York")
  end


end
```

**适应变化**

我们的世界并不完美。在软件开发行业，事物频繁发生变更。幸运的是，以上的工作使得测试脚本不仅仅更易读，而且也更容易适应变化。

**1. 客户修改了术语**

众所周知，项目使用同一套语言是一个好的实践，即使在测试脚本里面也是如此。举例来说，客户现在更倾向于使用“Return Trip”这个名词，而不再是“Round Trip”。借助于重构测试脚本，这很容易做到。

把光标移到“SelectFlightPage”类（pages\select_flight_page.rb）的“select_trip_round”函数，在“Refactoring”菜单下选择“Rename ...”项（Shift+F6）

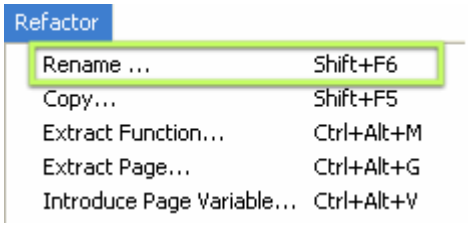



**图 7. “Refactor”菜单 - “Rename”**

然后输入新的函数名字“select_return_trip”。

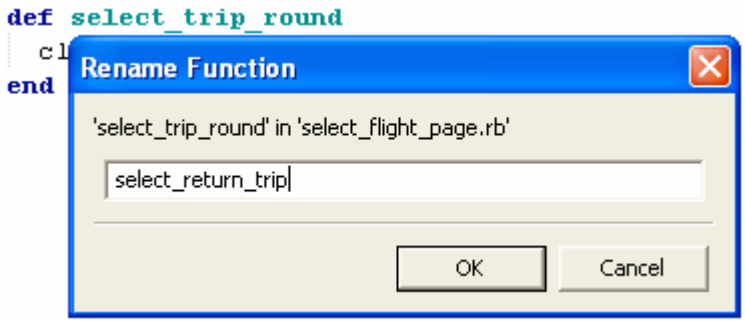



**图 8. “Rename Function”对话框**

测试脚本其他引用“select_trip_round”的地方就都更改为

```
select_flight_page.select_return_trip
```

**2. 应用程序的修改**

应用程序（来自程序员）的修改就更普遍了。举例来说，程序员基于某些原因修改了航班选择页面，导致 HTML 页面上出发城市的属性从

```
<select name="fromPort">
```

改成

```
<select name="departurePort">
```

虽然用户不会察觉到任何变化，测试脚本（任何访问这个页面的测试用例）现在却会失败。如果你直接用录制的脚本文件作为测试脚本，修改的操作将会非常乏味，而且易于引入错误。

定位到“SelectFlightPage”的“select_from_new_york”方法（使用快捷键 Ctrl+T 选中“select_flight_page”，再输入快捷键 Ctrl+F12 选择“select_from_xx”），把“fromPort”改成“departurePort”。

```
def select_from_new_york
select_option("departurePort", "New York") # from 'fromPort'
end
```

看上去还不赖！

**结论**

本文我们介绍了在自动化功能测试中使用 Page 对象，以使测试脚本易于理解和维护。通过一个使用 iTest2 IDE 改善测试脚本过程的实际例子，我们演示了其提供的丰富的重构功能。

**引用文献**

Fowler, Martin, et al. Refactoring: Improving the design of existing code, Reading, Mass.: Addison-Wesley, 1999

**原文链接：**http://www.infoq.com/cn/articles/refactoring-test-scripts

**相关内容：**

- 给测试归类
- 调试失败Selenium测试的技巧
- Kent Beck建议超短项目跳过测试
- 用Cucumber脚本做故事驱动开发
- 代码契约组件使用详解

时刻关注企业软件开发领域的变化与创新

InfoQ中文站
www.infoq.com/cn

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

Java社区：企业Java社区的变化与创新

.NET社区：.NET和微软的其它企业软件开发解决方案

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

SOA社区：关于大中型企业内面向服务架构的一切

Agile社区：敏捷软件开发和项目经理

Architecture社区：设计、技术趋势及架构师所感兴趣的话题

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

# 新品推荐 | New Products

## Apache Wicket 1.4 发布了

**作者** Dionysios G. Synodinos **译者** 张龙

近日 Apache Wicket 项目发布了 1.4 版，这是一个开源、面向组件的 Java Web 应用框架，同时该版本对 JDK 的要求也首次提升到了 Java 5+，这样就可以充分利用 Java 5 的新特性如泛型等来增强 API 的类型安全性。

**原文链接：**http://www.infoq.com/cn/news/2009/08/wicket-1.4

## JBoss Netty 3.1 发布

**作者** Craig Wickesser **译者** 崔康

JBoss 社区最近发布了 Netty 3.1.0，为用户提供了编写客户/服务网络应用的另一选择。。

**原文链接：**http://www.infoq.com/cn/news/2009/08/jboss-netty-3.1

## 连贯NHibernate正式发布 1.0 候选版

作者 Abel Avram 译者 王波


连贯 NHibernate 项目的创始人 James Gregory 宣布项目已经到达了另一个里程碑——1.0 版本，当前已发布候选版本。

**原文链接：**http://www.infoq.com/cn/news/2009/08/Fluent-NHibernate-1.0-RC

## Moonlight 2.0 Beta 1 发布并包含Silverlight 3 部分功能

作者 Abel Avram 译者 王波


针对 Linux 和 Unix/X11 系统实现的微软 Silverlight 开源版本——Moonlight，已经进入 2.0 的 Beta1 阶段。由于 Moonlight 2.0 API 包含了 Silverlight 3.0 功能，故他是 Silverlight 2.0 API 的超集。

**原文链接：**http://www.infoq.com/cn/news/2009/08/Moonlight-2.0-Beta-1

## Google Wave预览 9 月 30 号开放 - 有何期待

作者 Dionysios G. Synodinos 译者 曹云飞


Google Wave 预览计划在 9 月 30 号对公众开放，Wave API 的技术领导人 Douwe Osinga 在 Wave 的 Google 用户组上公告了小组正在做什么以及未来的几个发展方向。

**原文链接：**http://www.infoq.com/cn/news/2009/08/wave-preview-opens

## Expression Studio 3 入门学习包

作者 朱永光


微软的加拿大用户体验团队，最近发布了 4 个针对 Expression Studio3 的入门学习包，为渴

望掌握 Expression Studio 3 新特性的同学们带来了福音。

**原文链接：**http://www.infoq.com/cn/news/2009/08/expression-3-starter-kits

## Gestalt：使用Ruby，Python和XAML编写网页脚本

**作者** 赵劼


最近微软在 Mix 在线实验室上发布了一个名为 Gestalt 的项目，希望借助高级语言与编译器的强大功能，在保留前端开发人员原有工作方式与习惯（编写 => 保存 => 刷新）的同时，提高构建复杂 AJAX 或 RIA 应用程序的生产力。

**原文链接：**http://www.infoq.com/cn/news/2009/08/gestalt

## CodePlex站点的Wiki引擎现已开源

**作者** Jonathan Allen **译者** 王波


来自微软开源站点 CodePlex 的 Wiki 呈现引擎现已经作为 API 开源了。

**原文链接：**http://www.infoq.com/cn/news/2009/08/WikiPlex

## ASP.NET MVC 2 Preview 1 发布

作者 赵劼


微软发布了 ASP.NET MVC 2 的 Preview 1 版本 ,并在论坛中向社区征求反馈意见和建议。ASP.NET MVC 2 的“主题”是“提高生产力”，提供多个有用的功能。令人放心的是，ASP.NET MVC 2 Preview 1 能够与 ASP.NET MVC 1.0 RTM 共存，不会影响后者的正常使用。

**原文链接：**http://www.infoq.com/cn/news/2009/07/asp-net-mvc-2-preview-1-released

架构师大家谈

# 架构之我见

现如今，架构一词算是软件行当的宠儿，而架构师则成了代表最高技术水平的职位。可是，在树立成为优秀架构师理想的同时，你真的对“架构”一词的含义搞清楚了么？本期，我们就：

- 什么是架构？
- 理想的架构应该需要具备怎样的特点？
- 如何才能得到理想的架构？

3 个问题请来了几位在行业中摸爬滚打多年的社区同行，请他们谈谈他们心目中的架构。他们分别是：

- 辛鹏 :东方易维技术总监兼首席咨询顾问、中国开放流程社区/用户组( China-OPC/OPUG ）发起人、jBPM-side 开源项目发起人
- 陈义：项目经理，SOA草根论坛发起人之一，开源力量（www.opensourceforce.org）EAI 版大版主
- 张凯峰：IBM 中国软件开发中心，软件开发工程师，InfoQ 中文站 Java 社区编辑
- 朱永光：InfoQ 中文站.NET 社区首席编辑

**1. 什么是架构?**

- 辛鹏：在 open party 的一次活动上，周爱民曾经用三句话讲过架构：架构是从无到有，架构是从表及里，架构是从粗到细，实际上他是从架构的过程来讲。从架构要做的事情上来讲，我认为：架构就是描述实体、实体的行为、实体与实体之间的关系与交互，这个实体是一个很粗的概念，在软件领域指软件中的层次、子系统、模块、 组件、接口等实体。

- 陈义 :架构就是我们通常所说的软件架构 ,目前 IT 界还没有明确的定义 ,维基百科( wiki ) 对它的定义是：软件架构是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。架构可以分为：应用架构、业务架构、基础架构和数据架构。每种架构的侧重点不同，应用架构专注于应用的设计与实现，业务架构侧重于业务流程的创建、组合和编排，基础架构是整个软件的基础设施。数据架构注重数据的存储、交换和维护。

- 张凯峰：架构应该是一套软件构建的生态系统，静态看有不同的层次结构，从底层到中间层到前端的 UI。动态看，覆盖了软件构建的整个生命周期，包含配置管理、构建、自动化、开发等等内容。

- 朱永光：Architecture 作为一个源于建筑世界的词，在软件世界中意义其实也很类似。就是回答如何搭建软件系统这个“大厦”的问题。不外乎：结构形式（系统结构，子系统，功能模块），主要材料（主要技术平台），连接材料（特定技术解决方案），内饰和外观（内外部接口），以及是否节能环保（投资回报高），经久耐用（可靠性），进出方便（用户体验），消防安检合格（安全性）。

**2. 理想的架构应该具备怎样的特点?**

- 辛鹏 :这个首先设定的前提是软件架构 ,软件架构应该具备的特点基本上也是公认的了，包括：可靠性、安全性、可伸缩性、可扩展性、可定制性、可维护性等、还有清晰统一的架构风格及模式。

- 陈义：理想的架构一直是软件架构师说追求的，它至少应具备可靠性、安全性、可扩展性、可维护性、敏捷性、前瞻性等特点。可靠性和安全性是架构的基础，它们决定了架构的成与败；可扩展性、可维护性、敏捷性和前瞻性为架构提供了更大的灵活性，它们决定了架构的好与坏。一个成功的架构只有在实践中不断地锤炼才能成为一个好的架构。

- 张凯峰：耦合度低，可伸缩，可扩展，维护性高，自动化程度高。

- 朱永光：首先，要适合，符合应用目的，你不能用一个体育场的架构去盖医院；其次，要稳固，不能因为架构不合理出现“莲花河畔” ；最后，要经济，要易于开发，不能因为结构复杂，难以施工，工期拖太久。

**3. 如何才能得到理想的架构？**

- 辛鹏：按照分类，架构分为：逻辑架构、物理架构、系统架构。逻辑架构关注的是层次、

子系统、模块等的划分策略、接口、与交互；物理架构关注的是物理元素、它们之间的关系，软件与硬件相匹配的部署策略；系统架构关注的是可扩展性、可靠性、鲁棒性、灵活性、性能等。因此从上面的三个方面去做好自己的关注点，就能得到一个理想的架构。

- 陈义：架构设计不仅是一门科学也是一门艺术，要想设计出理想的架构，必须逐步完善理想架构的几个特点。但不可盲目地追求理想化的架构，满足客户需求的架构就是好架构。架构只有在实践中逐步完善才能成为理想的架构。
- 张凯峰：首先是严密切合实际的业务需要，在此基础上，做最大可能性的技术选型和优化，在设计甚至研发过程中，依然能够一定程度调整架构以适应业务的变化。理想的架构不仅仅来自于先进的技术，更重要的是架构师自身的业务经验和技术能力的配合。
- 朱永光：两点，需求分析准确，构架师能力。

如何？以上发言是否让你有所感悟？期望下一次采访的对象就是你 !也期望咱们的这个栏目能在国内的架构师之间架起相互交流的桥梁。如果您有什么关于本栏目的好点子以及想讨论的主题，我们也非常欢迎您给出，邮件请发送至：editors [at] cn.infoq.com。

## 推荐编辑

马国耀，2007 年毕业于北京大学信息技术学院，硕士学位。他感兴趣的技术领域是 SOA，ESB，J2EE，Java 编程，开源项目等。业余时间爱好五子棋，围棋。他热情乐观，愿与天下各路豪杰结为朋友，可以通过 maguoyao [at] gmail.com 联系到他。

### SOA 社区编辑 马国耀

很荣幸能在这一期的《架构师》上写上自己的感受。我是来自 InfoQ 的 SOA 社区的马国耀，负责 SOA 和 Arch 社区的新闻和文章的本地化工作。人的一生中总会有很多偶然，偶然的事情多了就变成了必然。我和 InfoQ 的相识就是这样一个由很多偶然构成的必然。如果不是那次在公司的技术大会上被临时安排成认证考试的监考，我就不会认识张兰女士；如果我平时不喜欢在技术论坛上发表几篇文章分享自己的一些心得，那么在她跟我介绍这个网站的我可能就不会动心；如果不是才华横溢的胡键的个人魅力和他认真负责态度感染和吸引着我，我的热情可能会慢慢淡去。但是正是这些看似偶然其实必然的事件一直牵引着我，激励着我在这里默默而快乐地为社区贡献自己的一点绵薄之力。在了解 InfoQ 之前，我所熟悉的就是本公司的技术和理念，对外面的技术了解不多，慢慢就有了井底之蛙的感觉。成为正式编辑之前我做了两件事情，第一：认真的通读网站上关于我所熟悉并感兴趣的领域（SOA）的相关新闻和文章。第二：也就是加入 InfoQ 编辑团队必须要通过的考核——试译若干新闻和一篇文章。完成第一件事情的过程中，我“听”到了很多不同的声音，有来自社区的，有来自其他公司的，这些声音就像汩汩清泉，注入我的身体，让我倍感清爽。第二件事情我很幸运地通过了，这里面当然少不了胡键的悉心指导和帮助。就这样，我成为了 InfoQ 的正式编辑。加入这个集体之后，让我深受感动的是这里的每个人。他们热情洋溢，才情四溢，责任感极强，互相之间的友谊真切而诚挚。每每在翻译时碰到了问题，大家热烈讨论。有些人甚至通过自己的努力为大家创造便利，让程序代替人力完成一些“程序化的”准备的工作。这个组织不需要太多的管理，但合作无间，且效率极佳。刚刚加入这个集体时就被这种合作方式深深吸引了，现在我为成为这个集体的一员而自豪。

感谢张兰女生和胡键先生的引荐，让我有机会了解当前流行的技术和很多的技术，并为社区做点贡献。能为国外最新的企业级技术和解决方案在中国的传播贡献自己的一点力量让我感到无比光荣和荣幸。此外，对我而言，翻译的过程也是我自身学习和提升的过程。我目前在 IBM CDL 北亚太地区 SOA 战略合作部工作，主要工作内容是负责 SOA 的落地项目的支持和 IBM WebSphere 产品及技术在被亚太的传播。如果您想和我联系，您可以通过 maguoyao@gmail.com 联系到我。

## 封面植物

# 天女木兰

天女木兰自然分布于长江以南和长城以北，多产于辽宁、安徽、江西、广西北郊，也是中国东北地区惟一的野生木兰属植物。朝鲜、日本也有分布。近年来在吉林、河北也陆续发现了天女木兰自然林。天女木兰为濒危植物，属国家三级重点保护植物。

天女木兰一直鲜为人知，长在深山人不识。 1985 年，科研人员首次于桓仁发现这一稀世花种的消息，曾引起国内外的震动。

天女木兰又名天女花，是一种木兰科木兰属之落叶小乔木。树高 4 — 7 米，高者可达 10 米。天女木兰多垂直分布在 400 — 850 米之间的阴坡郁闭中等阔叶杂木林中。在桓仁满族自治县的老秃顶、花脖山，本溪满族自治县境内的草河掌、关门山等均有发现，从而使本溪成为天女木兰的故乡。天女木兰是本溪市市花,叶片宽大翠绿.白色花朵,粉心,味道芳香!即可观花又可观叶,是小区和别墅绿化的首选花木品种。

每年六七月份，是天女木兰盛开的季节，呈白色花纹的灰色树干，伸展出毛绒绒的灰色枝条，无数根枝条托起无数片绿叶，那绿叶呈椭圆形，片片晶莹，片片肥厚，隆起的叶脉，似在展示她生命的活跃。在两三片叶的扶衬下，但见一个八九厘米长的花梗蹿出，将一朵美丽的花朵高高举在头顶，此花不负重望，使出浑身解数，将一朵美丽的花朵高高举在头顶，此花不负重望，使出浑身解数，显示其婀娜多姿。瞧！那九片硕大而洁白的花瓣，重叠为三层，在红黄相环的花瓣点缀下，散发着沁人心脾的芳香，她是那样的圣洁，那样的高贵，那样的豁达，真是神来之笔也难述其美！

天女木兰绝非只盛开在高山之上孤芳自赏，同样适于人工栽植。她一般每年只开一次花，而明山区卧龙镇农民钟家义栽植的一株天女木兰，每年竟能绽开两次花朵，让人大开眼界。 1988 年春，钟家义从山上挖出一株野生天女木兰幼苗 ,栽于自家果园旁边。经过精心培育， 1991 年开出几朵小花。 1992 年开始，这株天女木兰每年竟开两次花，第一期在 5 月下旬，第二期在 7 月下旬，各持续一个月左右。花本是稀世奇花，奇花又每年绽开两次，堪称奇中之奇。这大概是为感谢主人的精心照顾，而给予的特殊回报吧！


1kg.org
多背一公斤
爱自然 | 更爱孩子


# 架构师 9月刊

**每月 8 日出版**

本期主编：胡键

总编辑：霍泰稳

总编助理：刘申

编辑：宋玮 朱永光 郑柯 李明 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：胡键，InfoQ 中文站 SOA 社区首席编辑

西安交大硕士，开源软件的坚定支持者。既曾任职于国内某大型通信公司，亦曾与友人合办公司，但终究一事无成。所学虽杂，然忘性颇大，直至如今大都不甚了了。而今又好上了中医养生，但也不过是蜻蜓点水，浮光掠影而已。曾参与《开源软件技术选型》的编写，并主持翻译《SOA实践指南——应用整体架构》。自去年接触Groovy/Grails，便一发不可收拾，同好可访问我的Groovy博客。今年 8 月，刚刚在GAE上发布GroovyLive！，一个带交互式Groovy教程的Groovy控制台。同时还在Google上托管了一个简单的js工具：WebCommand，用它可在浏览器中实现基本的命令行界面。该工具受TryRuby中命令行界面的启发而编写，也在我的GroovyLive！得到了应用。至于中文博客，这些年来我也换了不少，最后在我的MSN空间上落了脚，欢迎大家前来作客。如想和我联系，请发送邮件。

InfoQ中文站
www.infoq.com/cn

《架构师》月刊由InfoQ中文站出品。