时刻关注企业软件开发领域的变化与创新

架构师

9月 ARCHITECT

特别专题

动态语言浅析与案例分享

动态语言企业应用优缺点浅析

重构TekPub—从ASP.NET MVC框架迁移到Ruby on Rails

用Rails创建高质量Web应用

虚拟座谈：HTML5来了，JavaScript框架会如何发展

一种适用于真实世界BPM的协作方式

Ajax应用开发：实践者指南

MonoTouch中的MVC简介

测试工程师的学习之旅

InfoQueue
Enterprise Software Development Community

每月8号出版

篇首语 | Editor's letter

最美的祝福送给天下教师

夜已深沉，当我怀着感恩的心情与略带疲惫的身体，整理好本期《架构师》杂志的所有内容之时，时钟已经划过午夜的星空，新的一天来临了，而这一天是不同寻常的，因为它是 9 月 10 日——教师节。

看着窗外昏黄的灯光，不禁回想起了自己成长的往事，甚为感慨。在幻变的生命里，其实岁月才是最大的小偷，偷走的是消逝的年华，偷不走的是温暖的记忆。如今，我已近而立之年，就在二十多年前，我还是一个不会抓笔拿书的懵懂孩童，到后来长达十几年的校园生活，一点一滴都只存在于记忆深处，或模糊或清晰。而这一路走来，不知道经历了多少位老师，他们的容貌，很多已经记不清了，或者已经模糊了，但是他们的谆谆教诲，我却不敢忘怀，正如："随风潜入夜，润物细无声"，他们对我潜移默化的熏陶和浸染，至今还在影响着我，因为他们已经给我打下了烙印。在这里，我想对曾经关心我，呵护和引导我的老师们真诚地说声"老师，您辛苦了，祝您教师节快乐！"

随着时间的积淀与岁月的洗礼，我对"老师"这个词也有了新的认识。正如两千多年前的孔子曾说过："三人行，必有我师焉。"我们身边其实到处都是老师，而不仅仅是学校里的老师，每一个人都有可能成为自己的老师。小时候，父母教育我们怎么做人，如何处事。上学的时候，身边的同学都有不同的优点，于是私下里"偷师学艺"，取他人之长以补自己之短。结婚后，突然发现老婆竟成了我最好的老师，她让我渐渐改掉了急躁和好强的缺点，学会了宽容和耐心，学会了如何去欣赏生活中的美，也让我学会了用心去倾听别人，而不要急着表达自己的想法。活到老，学到老，这句话对每个人都是适用的。我曾碰到过学历不高的出租车司机为了奥运北京的形象而努力学习外语，也碰到过在小餐馆里打工的年轻人坚持不懈地学习电脑知识，还遇到过大学里的厨师抱着英语书背单词，以及牙牙学语的孩童含糊不清地跟着妈妈一次次重复"您好"、"谢谢"、"请"，还有电视上灾区孩子们渴望求知的眼神，这一幕幕场景，我都会为之欣慰与感动。所以，我们更要时刻保持对生活的激情、对知识的渴求、对学习的热爱，这样的生活自然会精彩很多。

言归正传，回到 IT 行业，架构师又何尝不是这个行业中的"老师"呢？架构师不仅要对开发技术非常了解，而且需要有良好的组织管理能力，还需要团队协作技能以及很强的沟通能力。架构师是能在关键时候对技术的选择作出及时、有效的决定。可以这样说，一个架构师工作的好坏决定了整个软件开发项目的成败。他们丰富的项目经验可以保证项目的成功实施，也常常成为新人努力的方向和前进的标杆。他们的知识、见解甚至做人做事的方法、态度都足以影响项目团队里的每一个人。做一个架构师不难，但是要做一个优秀的架构师却绝非易事。各位立志成为架构师的朋友们，虽然任重道远，但是让我们一起努力。希望我们在前进的道路上，互为人师，相互扶持，共谋进步！

本期主编：马国耀

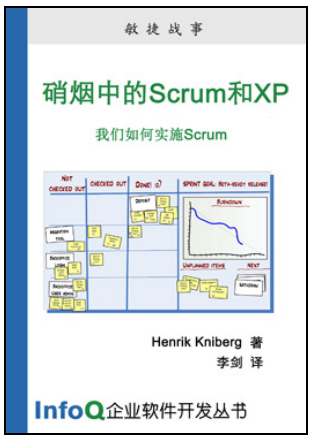
目　录

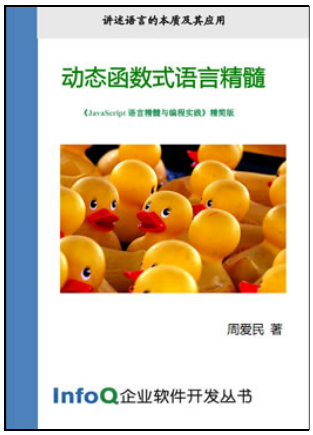
[新品推荐]

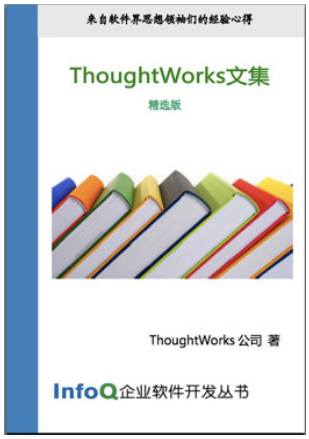
[封面植物]

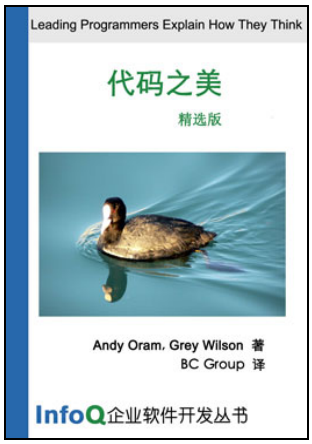
InfoQ企业软件开发丛书

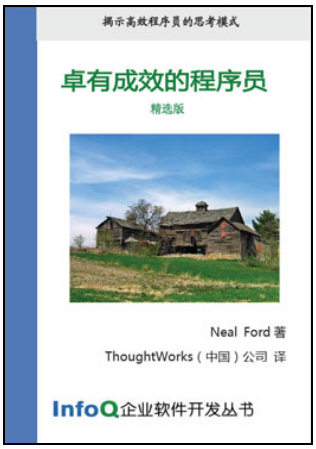
欢迎免费下载

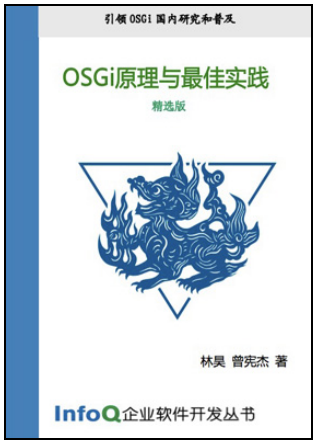












商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

汤涛滔谈系统需求收集和架构设计

MPD 大会是中国软件研发管理者自己的年会，是本土操作实践与海外研发经验分享的年会，主要面向软件开发管理人士。架构设计是前不久在上海举行的 MPD 大会的重要主题之一。在大会上，InfoQ 有幸就架构设计与架构师职业相关的问题采访了汤涛滔先生，汤先生具有相当丰富的项目管理和开发经验，曾任微软资深 咨询顾问。

汤涛滔，曾任职微软（中国）有限公司顾问咨询部、公共事业部高级行业顾问，历任资深技术顾问，技术总监，副总工程师等，具有相当丰富的项目管理和开发经验。负责过多个大型项目管理、设计和开发工作。作为技术专家设计、规划或评审多个项目，其中包括但不仅限于中国人民银行“人民银行货币调控系统”、海关总署“全国海关统计资讯系统”、“全国海关办公系统”和“移动办公系统”、人事部“全国机关事业单位工资管理系统”等，精通项目管理各个环节，对于软件需求开发管理、软件架构等具有相当丰富的实战经验。在业界具有良好的口碑。

InfoQ：今天非常很荣幸在亚太软件研发团队管理年会上采访到微软顾问咨询部资深架构师汤涛滔先生，汤先生先给我们介绍一下你自己？

汤涛滔：我最早也是从程序员开始做起，最早 CODING，包括做一些基础的工作。那后来一直往项目经理这个角色，以及架构师这个角色在转。再后来呢，就在往这个管理角色在转，就包括现在在做这个叫类似于 CEO 什么之类，类似这样的角色，等于从最基础的编程开始到项目管理到架构设计，到高级管理等。

InfoQ：众所周知，需求是架构的核心，不了解需求的话无法设计出符合需求的系统，那么又要准确而全面的去把握需求也不是一件非常容易的事情 ,那么作为你在搜集和整理需求的过程中，有没有什么经验和我们的架构师进行分享？

汤涛滔 :首先第一个问题是要摆正我的方向 ,首先方向是最重要的 ,因为如果方向出问题了，那需求做得再好都是有偏差的 ,所以要从战略的高度来看待我们软件系统的方向。而这个方向并不是说，因为我们有时候在软件开发里边，往往会遇到一些词汇上的一些差异，理解上

的差异，比如说我们一直在强调说，我们要搞清楚用户的真实需求，用户的核心需求，什么诸如此类的。实际上根据我自己的经验来理解，那么这么些年来，在这个项目里边我们发现一个共性的东西，第一用户往往阐述不清楚，他到底想要什么；第二即使他阐述清楚了，因为用户的需求往往是无限的。所谓无限的含义是什么呢？就像我们在心理学上有这样的词汇，当我们满足了一些基础需求，用户一定会有更高一层的需求。

InfoQ：肯定是。

汤涛滔：对，所以这样子，就会导致我们现在项目里边，跟着这个用户的所谓的用户的需求在跑，那这样我觉得实际上这是一个，首先我们要定义方向，就是我们的目标到底是什么，我们的目标真的是用户提的想法或者需求吗？我觉得在我们任何项目和产品设计里边不完全是这样。那我们的目标应该是什么呢？这是我觉得值得探讨的一个问题，就是我把我的一些想法来跟各位分享、沟通一下。实际上，我们在过去很多项目里边，我们会发现有一个很重要的因素，当我们去做一个产品的时候，我们这个产品的目标是什么？是为了占领市场，是为了这个叫生命力很长，长治久安这种，还是为了别的？那这个目标将会涉及我们是怎样来去做需求，以及需求做到什么程度。

假设我们说我们要像微软、像 IBM、像 HP 这样子来去做，这时候我们会发现他们有一个共性，什么样的共性呢？他们实际上往往不是完全跟着用户的需求在走，他们更多的是瞄准他们的战略目标，什么样的战略目标呢？就是我要到底实现我的什么样的商业目标，比如像微软，那微软的商业目标很简单，就是我要占领桌面市场。所以从 Windows 到 Office，它都在做同一件事情，是什么呢？替换用户的功能，我们可以感受到，目前我们几乎现在无论搞 IT 还是非 IT 还是管理，我们一大部分功能都被桌面替换掉了，就被微软替换掉了，以前我们做演讲，我们用透明的塑料膜做得的投影机，现在我们几乎很难看到了，因为这项功能已经被替换掉了。

InfoQ：还有胶片。

汤涛滔：被 PowerPoint 替换掉了，所以从这个角度讲，我们在做需求的时候，首先要搞清楚我们的战略方向，我们的目标是什么，然后才来根据这样的目标去做取舍和选择。因为有时候对用户来讲，他提出的需求，不一定是跟我们的目标相完全匹配的，这时候我们要放大跟我们目标匹配的需求，而缩小那些跟我们目标不匹配的需求，换句话说叫排除这些噪音。因为在一次软件项目里边，一定会有大量这样的噪音，那如果我们做这到一点，就意味着什么呢，我们让客户对我们产生严重的依赖。我恭喜各位，如果你找这到一点，你就会取得巨大的成功。

InfoQ：架构师不仅要懂技术，还要懂得处理与各种人之间的关系，那么要想做一个成功的企业架构师，或者是项目架构师，又必须要处理好与项目经理、开发人员、测试人员等等，甚至和 **CTO** 之间的关系，那么在这方面你有什么诀窍呢？

汤涛滔：我觉得就说在作为架构师的这个 Skill Sets，就是他的这些能力的这些集合里边，实际上有很大一块是非技术的因素 ,技术只是占其中的一小块。根据我这么多年的经验的理解，技术只是占到其中很小的一块。那其他的很大一块呢，就包含有不仅仅是说素养的问题，那很重要的一块就是个沟通的问题，我们又称之为叫架构的协作。所谓架构协作是指什么呢，就说我们设计出一个架构以后 ,就相当于是指明了我们未来这个团队在这么长时间的一个方向，也就说我们的战略方向、我们的目标，那在这样一个前提下呢，因为我们知道，要一个团队，这个要完美的来去做好一件事情，一定是需要有统一的思想认识和统一的行动方向。所以就是要拿以前毛主席一句话，叫团结一切可以团结的力量。

所以这时候我们就需要涉足我们的 CEO、客户的高管，如果我们做产品化的话，那一定要涉足到不同的部门和线条，让他们能够来支持我们。换句话说，让我们整个一个团队，不仅仅是说开发团队，我们的眼光作为架构师，一定要站在一个更高的高度来去看。这个更高的高度就是，我们看所有周围这些，凡是跟项目未来会有关联的，都是我们的资源，这叫有资源的方式来去看待。而正是由于这些是资源，所以要通过各种方式，沟通是其中一个很重要的渠道，然后把他们这个叫资源，都能够纳入到我们团队，朝着我们团队的方向前进。所以在这一块沟通和这种叫资源的协调，这是一个很重要的层面。架构师不是说我设计好了就不管了，设计好以后，一定要取得团队的认可，开发团队的认可，我们的商务团队的认可，我们的领导团队的认可，所以这一切都有赖于我们怎样来去沟通和协调这些资源，这只是其中的一个 Skill Sets 里的分支。

InfoQ：很重要的一个本质，也有人说做架构就是一个不断去寻找平衡的过程，你比如说你要想让你的系统有非常高的性能，那你可能就会牺牲安全，那你要安全性很好，你可能就会牺牲性能，更多的抽象可能会产生开发的复杂度等等这些东西。那么在寻找这个平衡的过程中，你有什么指导原则和我们的架构师进行分享的？

汤涛滔：这个核心的原则，因为我们知道架构师实际上在做架构工作的时候，最关键的就是在做一个取舍，做一个平衡的取舍。这种取舍一定有一个核心的贯穿这个取舍的一个方向，这个方向实际上跟我们前面讨论的方向几乎是一致的，就是商业目标或者战略目标。而这个战略目标一定要清晰话，这种清晰话就说，不仅是说架构师本身要来去找到这样的方向，而且要让整个团队都来去了解这样的方向。这样大家做取舍 ,才会有一个支点 ,这个支点在那，因为一个平衡，一个天平，它一定会有一个支起那个天平的那个支点，这个支点在哪，这个

很关键。这个支点将决定这个天平它的这个叫，我可以用多少斤的力量来矛动比如说几吨的力量，这个支点很重要。

所以这个支点就是我们前进的方向，我们的战略目标，或者说我们的使命。OK，好，那有这样一个大的方向和大的支点以后，接下来就是我们怎么选砝码，我选那一些作为我来去挑选来去均衡的一些砝码。这些砝码有多方面的因素，有市场的因素、技术因素、团队的因素等等，这些因素实际上在我们每一次架构设计里边，都需要来考量。有用户提出来的一个因素，比如用户的优先级、技术的优先级、市场的优先级等等。由此我们才能真正的把这个平衡把握的很好。

就像你刚才提到的安全和可用性之间，那这的确他会有，像两个葫芦一样的，按下这边，那边冒出来，按下这边，那边冒出来。那现在问题就来了，要选择一个平衡点，这个平衡点到底在哪？那就是根据综合的因素，实际上我们在做架构的时候，往往会有这样的一些因素的考量，这些因素考量现在有一些算法，或者说一些公式来去做这样的一些均衡的取舍。这些大家可以参照一下需求方法论里头，像一些这种叫用力的权重什么之类的，类似这样的东西，那里边涉及到有环境的因素、有技术的因素、有用户的因素等等。

InfoQ：在您的演讲中你也提到了架构师要有创新思维，创新同时意味着探索，在另外一种意义上来讲，也意味着风险，那么您是如何来看待创新和风险的关系的？

汤涛滔：创新一定会有风险，因为有投入，而且产出是不可预知的，对吧。但是不创新的风险更大，为什么这么说呢，因为就说如果你不创新，那么我们的竞争对手会比我们跑的更快，在竞争领域里，不是说这个我们不能单单把这个眼光放到团队内，而是要放在整个市场，放在我们的竞争对手，放在我们的整个市场环境上。如果我们不创新，别人一定会创新，并且跑的比我们还快。那对于这个创新思维呢，像 Intel 的一些总裁们，他们是对这个很有理解，所以我们看到像现在 Intel 的 CPU，他是不断自己在超越，就像摩尔定律，三个月更新，三个月更新。像这个就是他自己自我更新，不断的推出更好的、更新的，更能满足市场需要的一些新产品，所以综合起来就这一句，创新有风险，不创新风险更大。

InfoQ：架构师也有很多的分类，有企业架构师、业务架构师，还有服务架构师。那在您看来架构师应该怎么分类会更好一点？

汤涛滔：这个分类就跟我们市场细分一样，他有很多的细分的变量，这些细分的变量会使得我们分层的不一样。比如说我们从这种大的方向上来分，通常架构师是分成企业架构和解决方案架构，这样两个层面，一个企业架构就相当于我要来去规划整个企业未来的信息化，如何来支持业务，包含一大堆的系统，可能需要五年、十年怎么发展；那对于应用架构师，就

相对是我们在软件内部的，在比如说我们做一套系统，那这时候应用架构师就开始承担起相应的职责。那实际上在应用架构师里边又可以进一步细分，比如说分数据库的架构的，这个网络架构、安全架构等。当然企业架构层面也可以细分，比如说 Infrastructure，就基础架构的架构的，然后信息化的，Informational Architect，Solution（解决方案）之间怎么结合。

所以呢，在这个 Enterprise Architect 这个层次呢可以细分，那在 Solution Architect 又可以继续细分。但目前的分法呢 ,就是从大的范围来讲 ,就是 Solution Architect 和 Enterprise Architect 这样两个领域，因为他们之间差别还是蛮大的，那个 Enterprise Architect，他关注的不是说我继续去解决某一个用户的单个的问题 ,而是解决一个企业未来的发展方向和这个整体的问题。那 Solution Architect 解决的是相对细，就我某一个方向，比如说我的 HR，我的这个 ERP，这样比较具体一些。所以对于 EA，就是 Enterprise Architect 来讲呢，相对呢，他是应该说是叫在这些不同的 Architecture 之间一些关联，这样一个就是分类的一个模型。

那具体的在应用方面，当然可能还有些其他的分法，在业界目前来讲，这个 EA 和 SA 这两个领域是大家比较公认的，那至于具体内部在怎么细分，就各有各的不同的分法，微软可能有微软的分法，IBM 可能有 IBM 的分法。大概是这样。

InfoQ：现在我们知道面向服务的架构，也就是 **SOA** 越来越完善，随之也诞生了新的架构师的抬头，叫 **SOA** 架构师，那 **SOA** 架构师与传统架构师，你认为在职能上有没有什么主要的区别？

汤涛滔：因为刚才咱们提到，传统架构师应该讲它还是两个层面的，有 EA 和 SA 之分，就是 Enterprise 级别和 Solution 级别的，那我估计你提到的这个传统架构师，应该是大部分应该是叫在 SA 的这个范畴的，Solution 这个范畴的。那 Solution 这个范畴和 SOA 架构师之间，确切讲它有些重叠，有一些不同，这个重叠体现在那里呢？就说 SOA 应该它更多的面向服务的这种架构，他既可以作为我这个叫 SA 这个底层，就是我一个 Solution 的底层。比如我们假设做一套系统，这一套系统是我们企业的 ERP，因为 ERP 要跟太多的系统打交道，要跟供应链打交道，要跟 HR 打交道，要打财务打交道。那么他们之间的关联实际上就可以通过 SOA 来做，这时候 SOA 架构师呢，就跟这个 SA 的架构师它的职能就蛮重叠了。

那么还有一些就说是我们企业内的这些不同系统的整合，和企业的系统跟外部的系统的整合，这时候它就不会涉及到比如我系统内部一些细节，关注的只是系统与系统之间怎么来交互，怎么来实现流程编排。那这时候呢，SOA 的架构师的关注点就会在系统间的这个服务接口，系统间的这个交互这一块，所以这时候它跟传统的 SA 呢，就会有一些差异。这种差异呢，并不是说它本质的区别，而是说随着技术的发展的一种区别，那可能以前大家都是通过传统的架构师来去设计标准的接口，现在我们把这个设计标准接口这一块，因为我们要跨系

统、跨平台，那很难做了。我们就通过 SOA 来去实现，这时候 SOA 架构师相当于就把传统的接口设计这一块放大了，就类似于这样，所以他们之间是一个既有重叠，又有不同，这样一个关联，是这样子。

InfoQ：不论是在国外还是在国内很多架构师也都是从程序员转型过来的，刚才在您的介绍中，您也提到您也是从程序员一步一步走到现在的，那么您认为就是您的程序员的经历，对您后来作为一个构造架构师的职业有什么具体的铺垫？

汤涛滔：这个实际上是蛮关键的，这个实际上跟我们设计软件、设计产品是一样的，每一个人你首先需要有个目标，如果没有目标，那就意味着，就说你不会去关注你周围的跟这个目标相关的事情。比如假设我们说，我们想成为比如说我想学开车，如果是说这个你不定义这样一个目标，假设天天周围有大量的这些信息你不会去关注了，在心理学上有一个重要的法则叫吸引力法则，就说只有当你心里有所想，这些信息它才会来找你。因为您周围，比如说我们每个人接触的信息量太大了，就说曾经有人估算，大概是在几千万亿的比特的信息量，我们每天接受的。但实际上大概我们真正能吸收的，或者说我们能抓住的这些信息，大概只有几千亿个比特的这种信息抓在我们脑子里边。

比如说像在我们这个场馆里边，我们这边挂的这种壁画或者油画，我相信大多数人可能不会去关心。但是如果是一个油画家进来，我相信他首先就看，这个画怎么样，它的风格是什么，OK 这就是我们所谓的吸引力法则。所以作为程序员来讲，首先你需要定义你的发展方向和目标，假设是说我就以架构师，或者我以项目经理作为发展目标，那这时候呢就说，我们都知道这样的一些角色他有他的一些职业技能的一些要求，或者说我们叫 Skill Set。那这种 Skill Set 它里边有很多分支，那这些分支呢，我们就需要来去考量，比如说我，在这些技能上，哪些是我的长板，哪些是我的短板，那我们就需要来考量怎么来去补这些短板，怎么样来去补这些短板，这样子我们才能逐步的往这个方向来走。

比如我们从程序员发展，我们想来成为一个架构师，这里头就有几个关键的技能是必须的，第一个我们刚才提到的沟通能力，演讲能力，因为你需要带着团队，你需要来让团队所有人按照你的设计思想往前来走，所以呢演讲力、沟通力这是必备的。那同样我们需要有领导力，我们要把整个团队能够领导拧成一股绳，所以这个领导力又变得很关键。所谓领导是这样的，拿传统的一个思路来讲，实际上就是一个叫领袖和导师，所以这里头不仅需要带着别人往前冲，还要作为导师来去教别人。所以这个时候，类似这样子的一些，还有很多其他的，我们只是举了其中的几个小例子，这样的这些技能方向一定是要来去关注，这种关注不是说我就天天坐在那想，我要成为一个架构师，我要成为一个架构师。

不行，一定要具体化，就说作为一个架构师那需要有这些能力，OK，那我们应该怎么样来

去，我想成为架构师的话，我就必须演讲力、表达力、沟通能力提升上去。这种时候我们就要来去通过各种方式、各种手段来去改善和提高，这样子才能一步一个脚印（走上去）。实际上这对我们任何人 ,就说职业生涯规模也是一样的 ,就不单单是说从程序员发展到架构师，这个我相信是通用的。你首先需要把你的目标具体化，才具备可执行的价值，才有可实施的这个叫力量，你才会真正的往前走，那只是说天天想，我要一个别墅，我要有一个别墅，那是不行的。

InfoQ： 一定要具体化。那最后我们再简单的总结一下，你认为作为一个称职的架构师需要具备那些特质，如果说这个是一个 **Senior Developer**，就是一个高级的程序员，他想成为一个架构师，一个成功的架构师，他应该怎么做？

汤涛滔：实际上知识的学习是很重要的一个层面，我们刚才已经提到了，我们要这些知识技能都要来去学，那么学，我们说形成知识技能是不够的，一定要让它应用并且形成自己的智慧，这才是有价值的。培根说了知识就是力量，那发展到今天，我们越来越发现实际上，就像彼得•德鲁克说的，著名的管理大师说过，知识本身这个它不产生价值，哪怕它在我们脑海里装的太多太多，只是知识而已，只是装在脑袋里的知识，东西而已，让它产生价值最佳的方式是什么？应用和分享知识，才能带来价值。就像我们在这边跟大家来分享，这个分享是有价值的。

那同样，我们比如说我们成为这个要成为架构师的话，那我们可能把这些技能都开始一个一个练了。练完以后，它形成你的一个一些智慧，形成你的知识，但是我们还要作为架构师这个角色来训练。这时候可能很多人会问了，我们在企业里边我们没有这样的机会，我们每天都在忙着自己的代码呀，这里头就有个很关键的点，什么关键的点呢？主动积极去做你所想的，试图来去实现的目标的事情，也就说，虽然可能你不是一个架构师的角色，你是一个比如说程序员，或者高级程序员，或者是类似诸如此类的。那在任何一个项目里边，那一定会有架构设计团队来去做相应的工作，那这时候你把自己想象成那个组里的人，如果你是那个组里的人你会怎么做，你会不会做类似像他们这样的设计？或者说你有没有更好的想法？

如果你有，把它写下来，把它做出来。然后呢，换句话说，就叫贡献你的力量，贡献你的知识，哪怕就说这种贡献是没有任何回报，但是这种贡献对自己绝对是一次极好的锻炼和成长的机会，因为你站在架构师的角度去想问题了。也就说当你的内心改变了，你的外在才能来去改变。那这样一步一步下来，你会发现你的思维模式你的思考方式已经变的，跟你们这个团队的架构师的小组一致了，或者甚至比他们想的更多了，那试想，如果某一次做产品设计也好，做架构设计也好，你的老板的老板，你们的 CEO 发现你具备这个能力的时候，各位就不用我多说了。

原文链接：

http://www.infoq.com/cn/interviews/ttt-requirement-gathering-architecture-design

相关内容：

- 2010 年大规模技术架构的思路
- 五年Skype架构师之路的感言
- 你是个软件架构师吗？
- 又拍网架构中的分库设计

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

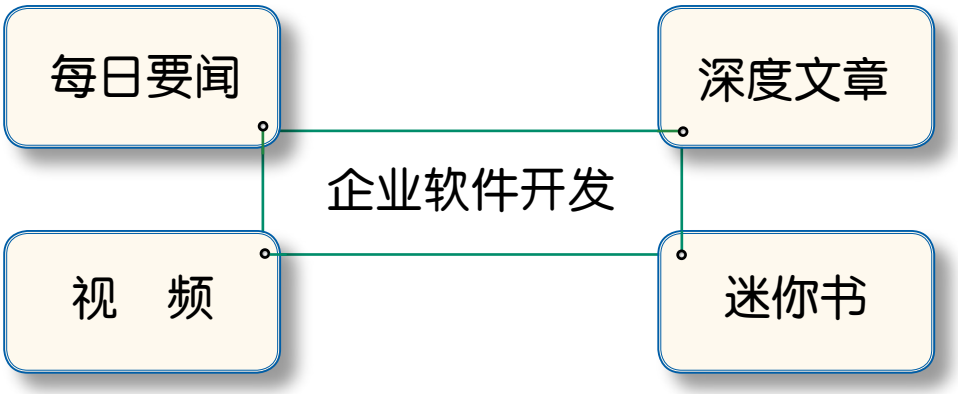
我们的**定位**：关注高级决策人员和大中型企业

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

Web 2.0 应用客户端性能问题十大根源

作者 崔康

Web 2.0 应用的推广为用户带来了全新的体验，同时也让开发人员更加关注客户端性能问题。最近，资深 Web 性能诊断专家、知名工具 dynatrace 的创始人之一 Andreas Grabner 根据自己的工作经验，总结了 Web 2.0 应用客户端性能问题十大根源，InfoQ 中文站将这十个问题做了概括整理，供 Web 开发人员借鉴和思考。

1. IE 中的 CSS 选择器（selector）运行缓慢

Web 开发人员通常使用 JavaScript 框架（如 jQuery）提供的 CSS 选择器来实现查找功能，如 var element = $(“.shoppingcart”)，但是 IE 6 和 7 没有提供这种查找方法的原生实现。所以，JavaScript 框架不得不通过遍历整个 DOM 树来达到目的。这种方式花费的时间比在其他浏览器中的消耗要多得多，而且严重依赖于 DOM 树的规模。IE 8 对 CSS 查找提供了较好的支持，所以 Web 人员最好升级相应的 JavaScript 框架版本以利用这些新特性。

2. 针对相同对象重复进行 CSS 查找

正如第一点所说，单个 CSS 查找代价高昂，在这种情况下，如果还要对相同的对象进行多次重复查找，那性能问题就可想而知了。下图是一个典型的 Web 页面中 CSS 查找功能调用统计结果：

Contributor	Tot...	Invocations
$(".ztBucket")	660.08	8
$(".subNavDD, .seeItAllDD")	177.52	1
$(".bingProd")	168.93	2
$("div.contenPrice")	166.83	10

（引自 dynatrace 博客，中间一列为查找函数总执行时间，单位毫秒，最后一列为函数调用次数）

对于这种问题，Andreas Grabner 建议将第一次查找的结果保存到变量中，在以后需要的时候重用即可，不必再重复进行查找。

3. XHR 调用太多

JavaScript 和 XmlHttpRequest 是 AJAX 技术的基础，很多 JavaScript 框架都提供了非常方便的使用方法，Web 开发人员会充分利用其异步通信优势来实现诸如分页加载等效果，避免对整个页面的操作。

Andreas Grabner 根据自己的经验指出 ,他发现这种方式被滥用了——过多的信息通过过多的调用来动态访问。例如，在一个显示 10 种商品的页面中，开发人员可能想分别加载每种商品的详细信息。这意味着，你需要和服务器端进行 10 次交流才能得到全部信息，也会对后台系统产生压力。他建议，在这种情况下，把 10 次调用合并成 1 次来减少通信压力。

4. 代价高昂的 DOM 操作

操作 DOM 是网页交互性的必要技术。拿添加 DOM 元素来说，存在多种实现方式，每种方式因为不同的浏览器类型和元素数量大小带来的性能影响也各不相同。建议大家仔细分析比较不同的方法，采用适合自身情况的技术。

5. JavaScript 文件过多

Andreas Grabner 说，对于一个典型的网站来说，存在超过 40 个单独的 JavaScript 文件并不少见。他指出，JavaScript 文件过多带来两个问题：一是浏览器在加载这些文件时需要通过 JavaScript 引擎切换上下文运行环境，二是因为下载文件而带来额外的网络通信。解决方法是：减少 JavaScript 文件数量！

6. DOM 规模庞大

DOM 规模对页面性能影响很大，具体表现在：

- 占用的内存
- 从根节点到子节点的 style 变化所花费的开销
- IE 中 CSS 查找的性能问题
- DOM 遍历操作的性能问题

所以，警惕你的 DOM 树！

7. 事件处理函数绑定过多

对于 Web 开发人员来说，绑定事件处理函数是日常工作之一。Andreas Grabner 提醒大家关注其对性能的影响：

- 绑定操作本身消耗时间（如查找对象、注册事件管理器等）。
- 当事件被触发时，事件管理器需要查找注册该事件的元素，并调用正确的事件处理函数。
- 在切换页面时，要记住对事件解绑，避免 DOM 相关的内存泄露问题。在生产集群上频繁根据垃圾邮件模式为邮件计分

8. 外部服务执行缓慢

很多网页都嵌入了外部内容（如广告栏等）或者调用外部服务，Web 开发人员通常需要在页面中包含由第三方提供商发布的 JavaScript 文件，而通常这些文件中就存在前面所提到的性能问题，我们需要擦亮眼睛，如果有问题要反馈给第三方供应商让其修改优化。

9. 滥用视觉效果

很多 JavaScript 框架都提供了绚丽的视觉特效，如动态弹出表单等，一些方法在示例代码中运行良好，但是在实际的页面中特别是 DOM 规模较大时表现不尽人意。Andreas Grabner 建议 Web 开发人员在引入视觉效果时关注其对浏览器 CPU、渲染引擎和整个网站性能的负面影响。

10. 日志和监控粒度过细

现在存在很多优秀的日志和监控工具，但是如果把粒度设得太细（如记录每次鼠标移动的详情），信息的收集过程会对 JavaScript 引擎和网络产生额外的负担。

Web 2.0 应用客户端性能问题十大根源向大家介绍完了，原文作者 Andreas Grabner 不仅是 Web 性能诊断工具 dynatrace 的创始人之一，而且参与了许多企业级 Web 应用的性能优化项目，他总结的这些问题相信会对国内 Web 开发人员带来一定的启示。

关心服务器端性能的读者朋友请参考本站编辑张龙撰写的《服务器端编程的十大性能问题》，InfoQ 中文站也会继续关注业界的最新进展。

原文链接：http://www.infoq.com/cn/news/2010/08/web-performance-root

相关内容：

- 剖析IE浏览器子系统的性能权重 、Yahoo推出开源YUI跨浏览器测试工具Yeti、Google提出Web性能优化新方法——Diffable

服务器端编程的十大性能问题

作者 张龙

今年 5 月底，瑞士计算机世界杂志上刊登了 Web 性能诊断专家 Bernd Greifeneder 的一篇文章，文章列举了其在过去几年工作中所遇到的服务器端编程的十大性能问题。Andreas Grabner 则在自己的博客上对这些性能问题给出了进一步阅读的链接。希望这些问题与相关的延伸阅读能为广大的 InfoQ 读者带来一定的启示。

问题一：过多的数据库调用

我们发现经常出现的一个问题就是在每次请求/事务中存在过多的数据库查询。有如下三个场景作为佐证：

- 在一次事务上下文中所请求的数据比实际需要的数据多出很多。比如说：请求所有的账户信息而不是仅仅查询出当前需要显示的信息。
- 多次请求同样的数据。这种情况通常发生在相同事务中的不同组件之间是彼此独立的，而每个组件都会请求同样的数据。我们并不清楚当前上下文中已经加载了哪些数据，最后只得多次发出同样的查询。
- 发出多个查询语句以获得某一数据集。通常这是由于没有充分利用到复杂的 SQL 语句、存储过程等在一次批处理中获取需要的数据所导致的。

延伸阅读：Blog on Ling2Sql Performance Issues on Database、Video on Performance Anti-Patterns

问题二：过多地使用同步

毫无疑问，同步对于应用中共享数据的保护来说是至关重要的举措。但有很多开发者却过度使用同步，比如在超大段的代码中使用同步。在低负载的情况下，这么做倒没什么问题；但在高负载或是产品环境下，过度的同步会导致严重的性能与可伸缩性问题。

延伸阅读： How to identify synchronization problems under load

问题三：过度使用远程调用

很多库都使用了远程通信。对于开发者来说，远程调用与本地调用似乎没什么区别，但如果不清楚远程调用的本质就会铸成大错，因为每一次远程调用都会涉及到延迟、序列化、网络堵塞以及内存使用等问题。如果没有经过深思熟虑而盲目使用这些远程技术就会导致严重的性能与可伸缩性问题。

延伸阅读： Performance Considerations in Distributed Applications

问题四：错误地使用对象关系映射

对象关系映射为开发者解决了很多负担，比如从数据库中加载对象以及将对象持久化到数据库中。但与其他任何框架一样，对象关系映射也有很多配置选项需要优化，只有这样才能适应于当前应用的需要。错误的配置与不正确的使用都会导致始料不及的性能问题。在使用对象关系映射框架前，请务必保证熟悉所有的配置，如果有机会，请深入到所用框架的内核，这样使用起来才有保障。

延伸阅读：Understanding Hibernate Session Cache、Understanding the Query Cache、Understanding the Second Level Cache

问题五：内存泄漏

托管的运行时环境（如 Java 和.NET）可以通过垃圾收集器进行内存管理。但垃圾收集器无法避免内存泄漏问题。“被遗忘”的对象依旧会占据着内存，最终将会导致内存泄漏问题。当对象不再需要时，请尽快释放掉对象引用。

延伸阅读：Understanding and finding Memory Leaks

问题六：使用有问题的第三方代码/组件

没有人会从头编写应用的全部功能。我们都会使用第三方程序库来加快开发进程。这么做不仅会加速产出，也增加了性能上的风险。虽然大多数框架都具有良好的文档并且经过了充分的测试，但没人能够保证这些框架在任何时候都会像预期的那样好。因此，在使用这些第三方框架时，事先一定要做好充分的调研。

延伸阅读： Top SharePoint Performance Mistakes

问题七：对稀缺资源的使用存在浪费的情况

内存、CPU、I/O 以及数据库等资源属于稀缺资源。在使用这些资源时如果存在浪费的情况就会造成严重的性能与可伸缩性问题。比如说，有人会长时间打开数据库连接而不关闭。连接应该只在需要的时候才使用，使用完毕后就应该放回到连接池中。我们经常看到有人在请求一开始就去获取连接，直到最后才释放，这么做会导致性能瓶颈。

延伸阅读： Resource Leak detection in .NET Applications

问题八：膨胀的 **Web** 前端

由于现在的 Web 速度越来越快，用户的网络体验也越来越好。在这个趋势下，很多应用的前端都提供了太多的内容，但这么做会导致差劲的浏览体验。很多图片都太大了，没有利用好或是错误地使用了浏览器缓存、过度地使用 JavaScript/AJAX 等，所有这一切都会导致浏览器的性能问题。

延伸阅读： How Better Caching would help speed up Frankfurt Airport Web Site、Best Practices on Web Performance Optimization

问题九：错误的缓存策略导致过度的垃圾收集

将对象缓存在内存中可以避免每次都向数据库发出请求，这么做可以提升性能。但如果缓存了太多的对象，或是缓存了很多不常使用的对象则会将缓存的这种优势变成劣势，因为这会导致过高的内存使用率及过多的垃圾收集活动。在实现缓存策略前，请想好哪些对象需要缓存，哪些对象不需要缓存，进而避免这类性能与可伸缩性问题。

延伸阅读：Java Memory Problems、Identify GC Bottlenecks in Distributed Applications

问题十：间歇性问题

间歇性问题很难发现。通常这类问题与特定的输入参数有关，或是发生在某个负载条件下。完全的测试覆盖率及负载与性能测试能在这些问题产生前就发现他们。

延伸阅读：Tracing Intermittent Errors by Lucy Monahan from Novell、How to find invisible performance problems

原文链接：http://www.infoq.com/cn/news/2010/08/top10-server-side-performance

相关内容：

- WebShpere 7 支持基于OSGi的应用部署和SCA集成
- JRebel 3.0 发布——热插拔重装上阵
- Oracle宣布GlassFish路线图
- Spring的IoC容器

EffiProz：面向.NET 程序员的跨平台嵌入式数据库

作者 Jonathan Allen 译者 张龙

EffiProz 是个完全由 C#编写的嵌入式数据库，它有两种模式：磁盘模式与 内存模式。这样，其开发者就可以将它移植到具备 CLR 的大多数环境中，包括.NET Compact、Mono、Windows 7 以及 Silverlight。EffiProz 的下一版本将会扩展到移动平台上。

在被问到为何要使用 EffiProz 而不是 SQL Server Compact 或是 SQLite 时，EffiProz 的作者 Irantha Suwandarathna 回应到：

> 如果人们对以下特性感兴趣，那么他们就会使用 EffiProz：
>
> - 非常棒的性能
> - ACID 事务
> - 功能完备、兼容于 SQL 的支持（支持的特性比 SQLite 和 SQLCE 还要多）
> - 完整的代码数据库引擎
> - 希望同样的数据库能够用在.NET、Compact Framework、Silverlight、Mono、Windows Phone、 Moonlight（未来还有 Android 与 iPhone）应用上（或是可以在这些平台间迁移数据库文件，由于初步兼容于 HSQL，你甚至还可以在.NET 与 JAVA 应用间交换相同的数据库文件）。

此外，EffiProz 还支持 Entity Framework，下一版本（1.3）将支持 MonoTouch。对 MonoDroid 的支持也在计划当中。

由于应用的质量在很大程度上依赖于所用的组件质量，因此我们就有关测试与质量保证过程的问题进行了咨询。

> 我们的回归测试套件的代码已经超过了 10,000 行。在目前的 1.2 版中，我们达到了 74%左右的代码覆盖率并且期望在 1.3 版中能将这一数字提升至 80%。我们会对每个存储层重复执行该测试套件，并且会重新运行这些测试用例 2 次：一次是测试事务

日志恢复，然后测试常规的关闭。除了这个自动化的回归测试套件外，我们还会进行更广泛的手工测试。

HSQLDB 是个开源的 Java 数据库，它构成了 EffiProz 产品的基础。

> 一开始，EffiProz 是 HSQLDB 到.NET 框架的移植。但现在，EffiProz 已经具备了 HSQLDB 所不具备的众多特性（比如，EffiProz 具有兼容于 SQL Server 的 UniqueIdentifier 数据类型）。HSQLDB 基于自由的 BSD 许可，而 EffiProz 对于非商业应用是免费的（比如学术、 研究等等）。如果用于商业目的，则需要购买许可。

原文链接：http://www.infoq.com/cn/news/2010/08/EffiProz

相关内容：

- Raven——一个.NET上的文档数据库
- 在.NET中使用Rails风格的数据库迁移方式
- RAD Studio 2009 通过Mono提供.NET应用跨平台开发

本地（手机）应用未来会怎样？

作者 Jean-Jacques Dubray 译者 侯伯薇

当前本地手机应用大获成功，这在业界引起了热烈的讨论。Google 的 DeWitt Clinton 说：

> 当前，本地手机应用的用户体验更好，运行得更快，还更容易用它来赚钱，而且它比手机的 Web 应用更容易找到。
>
> 但是那也是一种缺陷，而不是一种特性。这种模式在桌面应用中没能留存先来，因此我认为它也不会在手机应用中长久存在。

他的帖子是针对 GigaOM 的 Stacey Higginbotham 对 Gowalla 公司的 CEO Josh Williams 的采访做出的响应，在那个采访中提到：

> Gowalla 已经创建了 iPhone 应用，并且还为其他智能手机平台创建了“漂亮的移动站点”，但是人们更多都在使用那个应用……它从根本上驱动了“一次部署到处使用”模式，并且让市场分成了更多的部分。

Stacey 看到的是更加广泛的趋势：

> 这也是人们如何使用 Internet 以及 Internet 的本质正在发生大规模转变的征兆。

DeWitt 继续提到：

> 当前看起来很可笑的是，(90 后的) 人们没有清楚地看到在桌面计算机中，web 应用已经呈现出取代本地应用的趋势。Google、Yahoo、 Facebook、Twitter、YouTube、Amazon、Ebay、Gmail、MySpace、Craigslist、Wikipedia、 Blogger、Wordpress 等等，等等。
>
> 在你 OSX 或者 Windows 计算机上 ,你最常用的本地应用程序是什么呢？ 我喜欢的有 Chrome、 Firefox、 Opera、 IE 或者 Safari。

> 创建基于浏览器的应用程序，就可以让成千上万的人来使用它，这比为同样的用户创建多个本地的桌面应用要容易得多。

Stacey 指出，本地应用程序在构建业务模型方面非常成功，而 Web 应用在利用内容和应用程序赚钱方面有些困难。

> 应用程序（不仅仅）流行，而且（……）人们会更换应用程序，但还是不能用基于 web 的服务来赚钱。

即便是 DeWitt 也承认：

> 当前手机上的 web 浏览器和手机的 web 工具还是和本地环境中的没法比。
>
> 但是事情变化得很快：
>
> - jQuery 团队最近发布了 jQuery 的移动框架（jquerymobile.com）。
> - GWT 的移动 web 工具集（code.google.com/p/gwt-mobile-webkit）是 Google 提供的用于“支持 HTML5 和移动 Web 特性”的库。
> - 移动浏览器也发展的很快，处于领先地位的是 Opera 和 Firefox。

DeWitt 做出以下结论：

> 没有任何技术上的理由会让移动 Web 应用无法追赶上来。

很自然地，DeWitt 的帖子引起了很多评论。Anssi Porttkivi 说：

> 你无法在本地应用中为用户界面的状态设置书签，然后将它发布到 Facebook 上与他人分享。你无法开启多个 UI 会话，就像你可以打开相同站点的多个浏览器页面一样。你不能随心所欲地复制和粘贴，而其中只包含 widget 文本。你无法在页面上搜索。在浏览器中得到的改善都无法在本地应用中获得，或者说至少不是免费的，而且总体上方式都不一致。这些都是我喜欢 Web 应用的原因。

Denton Gentry 评论说：

> 我喜欢本地应用的响应速度和优雅，但是我想要它能够将数据存储在云中（或者与云中的数据同步）。 而从不会丢失数据。

Doug Purdy 指出：

> 本地和 Web 平台之间竞争的关键因素之一就在于对新硬件功能的使用。

JR Holmes 回应说：

> 人们不喜欢 web 应用程序，因为他们将 web 应用视为信息的来源，而不是完成任务的工具。其中的关系和功能都截然不同。
>
> （因此）web 浏览器能够成为人们所使用的最流行、最通用的程序，也就不足为奇了。他们花费大部分的时间以各种方式来查看信息。那样说的话，Gmail 并不是 web 应用，而只是查看 email 信息的一种方式。

本地应用是否具备作为新的计算设备的固有优势吗？它能够继续与传感器连接，并利用更快的带宽（3G、4G）所带来的优势吗？“长尾”开发者会最终使用业务模型来发挥创造力进行自我解释吗？历史是否会重演，产生新的 Web 标准吗？或者我们进入了一个新时代（复杂）应用程序的架构是由用户体验所驱动的？你对这个问题的观点如何？

原文链接：http://www.infoq.com/cn/news/2010/08/future-native-apps

相关内容：

- Rhodes 1.5：使用Ruby为智能手机开发应用，已支持iPad
- 为网站和智能手机构建FlightCaster前台应用
- Nokia基于EPL协议向Symbian基金会捐献手机版Java运行时系统
- Rhodes 2.0 带来高清音频视频流，基于MIT许可免费使用

热点新闻 | News

开源权限管理中间件 Ralasafe 发布 1.0 rc2 版

作者 崔康

Ralasafe 是一款国产开源数据级权限管理中间件，使用 MIT 协议，最近发布了 1.0 rc2 版。项目采用 Java 语言编写，解开权限与业务的耦合，将权限策略集中管理，并使用图形化的管理模式。InfoQ 中文站就 Ralasafe 的应用场景、技术架构和未来规划等问题对项目负责人汪金保进行了专访。

InfoQ 中文站：作为一款权限管理中间件，Ralasafe 的典型应用场景包括哪些？

> Ralasafe 与各种应用系统结合，提供两种权限管理服务，对应场景是：
>
> 1. 行为权限：基于角色的权限管理 ——比如：张三是销售经理，能够查询订单；李四是人力资源经理，能够修改客户资料等。
> 2. 数据级权限：更加细粒度的数据内容权限管理——比如：总公司的销售经理王大大能够查询所有订单，北京分公司的销售经理吴北北只能查询北京分公司及下属子公司订单。
>
> Ralasafe 使用 Java 编写的中间件，与 Java 系统结合非常方便。与其他系统，需要 web service 等交互方式。

InfoQ 中文站：相比同类商业软件如 TAM、OES，Ralasafe 存在哪些优势？

> TAM(Tivoli Access Manager)和 OES(Oracle Entitlements Server)发展年份非常长，属于业内老牌软件。Ralasafe 从 2004 年开始研发，相对于 TAM、OES，还非常年轻。
>
> 三款产品都定位于数据级权限管理领域，Ralasafe 与 TAM、OES 相比，做了很多创新。比如：TAM、OES 的权限策略，完全基于规则。 Ralasafe 则提出了“用户分类”和“资源”的概念，然后用规则来描述“用户分类”、“资源”。相比较，Ralasafe 具有这样的优势：
>
> 产品功能：TAM、OES 和 Ralasafe 都能控制决策权限，但只有 Ralasafe 能够控制查询权限。比如根据规则，吴北北“能够修改北京订单”，所以吴北北能够修改订单 A、

订单 B。但吴北北能够查询哪些订单呢，这种查询只有 Ralasafe 能够控制。

1. 产品定位：Ralasafe 定位为轻量级，开发人员容易集成，最终用户通过 Ralasafe 界面能自我操控授权策略。
2. 产品友好易用性：Ralasafe 独创“用户分类”和“资源”的概念，并将其配对成授权策略。该方法不仅增强了授权策略的可读性，也增加了复用性。Ralasafe 和很多开源软件不同，使用 Ralasafe 几乎不需要 JAVA/XML 编程配置工作，全程使用 Ralasafe 友好的图形化界面。

InfoQ 中文站：能否介绍一下 Ralasafe 的系统架构以及所用到的技术？

> Ralasafe 主要由两大块组成 :安全引擎和 GUI 管理界面。安全引擎用来解析授权策略，为应用系统提供决策支持，查询服务；开发人员和最终用户可以通过 GUI 管理界面来管理授权策略，还可以进行在线测试。
>
> Ralasafe 的安全引擎由 Java 编写的，使用了 beanshell、castor 技术。因为 Ralasafe 涉及的 ORM 是动态的，不能像 hibernate 那样要事先编写 hbm.xml 文件，所以 Ralasafe 自己编写了一套 ORM。
>
> Ralasafe 的 GUI 管理界面 ,使用 GWT 技术 结合 GWT-EXT、EXT 技术。我们使用的 ext2.0.2 版本，该版本使用 LGPL 协议。由于后继版本使用 GPL 或者商业授权协议，对商业并不友好。因此下一版本，我们将寻求其他前台技术。

InfoQ 中文站：在应用 Ralasafe 过程中，开发人员需要注意哪些步骤？

> 不少开发人员接触 Ralasafe 后，问得最多的问题是：
>
> 1. 怎样与应用集成
> 2. 怎样使用 Ralasafe 编程
>
> Ralasafe 是一款中间件，提供权限管理服务。就像 LOG4J 提供日志服务一样，在需要的地方调用一下服务即可。就我们的项目实践经验，认为在控制层接入 Ralasafe 服务非常合适。
>
> Ralasafe 的接口非常简单，主要是 3 个类。用户类——org.ralasafe.User、Ralasafe 服务类 ——org.ralasafe.Ralasafe 和针对 Web 的服务类——org.ralasafe.WebRalasafe。另外，Ralasafe 提供的一些 Filter，也可以大幅减少编程工作。如登录控制 Filter，Url 访问权限控制，这些都只需要对 web.xml 进行简单配置即可。

关于怎样使用 Ralasafe 编程，这个需要转换一下思想。Ralasafe 几乎不需要 JAVA/XML 编程配置工作。首先，开发人员要关注于良好的项目实践：业务逻辑与权限逻辑分离解耦。在需要权限的地方，通过 API 或者 AOP 的方式注入。再次，权限逻辑是使用 Ralasafe 管理界面里面进行管理，不必进行编程或者 XML 配置。很多开发人员都习惯于以往的开发配置模式，所以对 Ralasafe 的图形化管理模式还不太习惯。

InfoQ 中文站：未来 Ralasafe 的发展规划如何？

> Ralasafe 的发展方向是：专注于应用安全领域，保持第三方独立性，为最终用户提供“自我掌控的安全”软件及服务。Ralasafe 服务软件开发商、企业和最终用户。
>
> Ralasafe 开发团队，向社会做出两项承诺：免费、开源。不增加软件开发企业的负担，也广思集益，打造稳定最好用的产品。以开放、第三方独立性和良好易用性为目标，不断为软件商服务，为企业和最终用户提供一站式安全管理服务。

InfoQ 中文站：该项目投入到开源世界，您有何期望？

> Ralasafe 从 2004 年开始研发，2009 年正式向市场发布。当时我们使用闭源模式，商业推广。也做了几个单子，但我们也听到很多市场的声音：
>
> 1. 很多企业认为 Ralasafe 很好，但商业的软件，不愿意使用。
> 2. 有些企业尤其是大企业，像 Ralasafe 这样的安全软件，采取闭源模式不放心。
> 3. 很多开发人员呼吁我们，希望我们开源。
>
> 鉴于这些情况，我们团队慎重考虑后，决定开源免费。就像起先我们花费 5 年研发时间一样，这次开源免费，我们也会不断坚持，做个负责任的开源团队。
>
> 开源后，我们看到软件下载量不断增加，很多网友通过 email/社区/QQ/gtalk 等方式，与我们互动沟通。不仅找出我们软件的一些 BUG，也让我们学习到很多东西，让我们不断提高，软件品质不断提高。开源并不像我们原来所想，只是付出。开源更是双赢。
>
> 在地域方面，我们已经推出中英文软件和中英文网站，为全球用户无偿服务。
>
> 在商业方面，Ralasafe 的商业模式主要是赞助商和咨询定制服务。开源一个月以来（2010 年 6 月 23 日开源的），已有 10 多位客户与我们洽谈合作。这也让我们看到了希望。当前，我们团队非常期望有些企业能够成为我们的赞助商。
>
> 我们也期望中国开源软件，越来越多，更多软件走向国际，形成大气候。

InfoQ 会继续关注国内开源社区的发展动态，对 Ralasafe 项目感兴趣的朋友可以访问其官方网站或者论坛了解更多细节。

专家介绍：汪金保，Ralasafe 项目负责人，首席架构师。2000 年开始接触 Java，立马喜欢上该编程语言。从事多项银行项目研发。 2004 年辞职专注于 Ralasafe 研发。我喜欢简单，不喜欢复杂；我喜欢自由，不喜欢束缚；我喜欢不断反省自己，又不断与周边的朋友分享。除了 IT 外，还喜欢民歌，戏曲，读书，网球和乒乓球。我的 Email: wangjbao[at]gmail.com，期望与大家交流。

原文链接：http://www.infoq.com/cn/news/2010/08/ralasafe1.0

相关内容：

- Mahout 0.3: 机器学习开源项目
- 走近淘宝开源平台
- Windows 平台的开源软件包管理系统
- 云计算标准和开源项目

热点新闻 | News

讨论：测试用例的粒度——粗细之争

作者 崔康

测试用例的粒度一直是软件测试领域的热点问题，无论是粗粒度还是细粒度，都各有利弊。最近，淘宝测试团队针对该问题举行了内部辩论会，相关内容值得借鉴和思考。

正方观点：测试用例的粒度应该细点，主要体现测试细节。主要论据：

> 1. 测试用例的编写就像是织网，而 BUG 就像是鱼，网织得越密，捕捉到的 BUG 就越多。
> 2. 测试思想的学习并不是一蹴而就的。对一个新人来说，这种学习是一个渐进的过程，具体到每个项目，更需要用更精细的用例来保证测试的覆盖率。
> 3. 设计详细的用例便于执行，便于新人理解，便于知识传承。

反方观点：测试用例的料度应该粗点，主要体现测试思想。主要论据：

> 1. 粗并不等于简单。测试用例的粒度粗点，是建立在我们对需求完全理解，对设计完全掌握的基础上的粗粒度。这样我们可以避免繁琐的流程，提高测试执行的效率，把握重点需求。测试粗粒度可以避免陷入机械性的测试。
> 2. 粗粒度的测试设计可以使我们把重点关注于设计，可以让测试往前走，在时间，资源有限的情况下，更高效地进行测试，保证产品质量。

随后，双方展开了自由辩论，其中不乏精彩的言论：

> 反方：思想就像大脑，测试用例是骨骼。在时间有限，资源有限时，必须要有所取舍，抓住主干测试，所以我们会追求白盒覆盖率而不是路径覆盖率。测试技能的提高是测试思想的不断丰富，测试手段的不断完善，而不是用例越写越细。
>
> 正方：在测试领域有 8:2 原则，80%的 bug 源于经常修改的 20%代码，测试用例的数量提升有利于减少这种 bug 遗漏。并且，越精细的用例越便于定位 BUG。
>
> 反方：就是因为我们的用例过细，导致在时间，资源紧张的情况下，导致覆盖率低，

> 没有发现尽可能多的 BUG，相反，如果我们在测试设计的时候，放得粗，可以把主要精力放在测试思想上，这样就可以提高测试覆盖率，发现更多的 BUG。测试用例的设计要先搭一个整体的框架，然后再逐步完善。

最后，评委做了总结发言：

> ……从管理者角度来看，还是希望测试用例的粒度细点好。
>
> 测试用例的粒度取决于项目质量的要求、时间的要求、用户的要求。如果时间充足，就可以把用例写细一些，时间紧张，就写粗些。有个词叫测试艺术家。就是要我们掌握质量与效率之间的平衡。
>
> 我们的用例不管是细还是粗，它都是为了达到最终目的——保证产品质量。测试用例写粗点还是细点，可以用一个例子来说明。当我们刚学车的时候，什么时候打方 向盘，什么时候踩离合，什么时候踩油门。都需要教练一步步教，要一步步来，这些都很明确的。当开车有一段时间后，什么情况下要做什么动作都会很自然，一气呵成。当我们的新人在进行测试的时候，需要很明确地知道怎么做，这时候用例就得细些。当成为一个很熟练的测试工程师的时候，设计用例时就不必纠结于这些细节了。每个阶段不同，做事方式就不同，只要满足结果就好。

淘宝内部辩论会结束了，但是对于测试用例的粒度的研究还在持续，读者对于这个问题怎么看？欢迎加入到讨论中！

原文链接： http://www.infoq.com/cn/news/2010/08/test-case-discussion

相关内容：

- 测试覆盖率强迫症
- 一次敏捷测试的实战
- 百度技术沙龙第 4 期回顾：Web测试自动化（含资料下载）
- SoftLogica发布负载与压力测试工具WAPT 7.0
- 没有自动化测试的应用应该如何测试？

TIOBE 编程语言排行榜：别了，Smalltalk！

作者 崔康

最新一期的TIOBE编程语言排行榜公布，世界上最古老的纯面向对象编程语言之一Smalltalk，跌出了前50名的总榜单，而新生代语言go稳步上升。

TIOBE在公布榜单时，以“恐龙”（Dinosaur）一词来修饰Smalltalk，点出了这个最古老OO语言的昔日辉煌和现时落寞，意味深长。TIOBE同时指出，在数月之前，另一个著名的纯面向对象语言——Eiffel也跌出了榜单，如今又轮到了Smalltalk，这从另一个侧面反映出了如今编程语言的趋势——多泛型（multiparadigm）：面向对象、面向过程、最好再来点函数式编程。

排在前5名的依旧是Java、C、C++、PHP和（Visual）Basic，它们在排行榜中的权重之和超过60%，优势明显。

另一方面，新生代语言在稳步上升。Go进入了前20名榜单，其东家Google也一直在努力推进Go的发展。前不久，Go语言创始人之一Rob Pike在O'Reilly开源大会上对Java、C++的复杂性表示了不满，并指出Go的优势：“Go试图把静态语言的安全、效率与动态语言的便捷结合起来，至于效果有多好，还需要你自己来使用、判断。”

具体的排行榜如下所示：

Position Aug 2010	Position Aug 2009	Delta in Position	Programming Language	Ratings Jul 2010	Delta Jul 2009	Status
1	1	=	Java	17.994%	-1.53%	A
2	2	=	C	17.866%	+0.65%	A
3	3	=	C++	9.658%	-0.84%	A
4	4	=	PHP	9.180%	-0.21%	A
5	5	=	(Visual) Basic	5.413%	-3.07%	A

6	7	↑	C#	4.986%	+0.54%	A
7	6	↓	Python	4.223%	-0.27%	A
8	8	=	Perl	3.427%	-0.60%	A
9	19	↑↑↑↑↑↑↑↑↑↑	Objective-C	3.150%	+2.54%	A
10	11	↑	Delphi	2.428%	+0.09%	A
11	9	↓↓	JavaScript	2.401%	-0.41%	A
12	10	↓↓	Ruby	1.979%	-0.51%	A
13	12	↓	PL/SQL	0.757%	-0.23%	A
14	13	↓	SAS	0.715%	-0.10%	A
15	20	↑↑↑↑↑	MATLAB	0.627%	+0.07%	B
16	18	↑↑	Lisp/Scheme/Clojure	0.626%	0.00%	B
17	16	↓	Pascal	0.622%	-0.05%	B
18	15	↓↓↓	ABAP	0.616%	-0.12%	B
19	14	↓↓↓↓↓	RPG (OS/400)	0.606%	-0.15%	B
20	-	↑↑↑↑↑↑↑↑↑↑	Go	0.603%	0.00%	B

（图片来源：www.tiobe.com）

最后，笔者选取 TIOBE 排行榜前 20 名语言作为模板针对读者朋友做一个调查，相信您也会从调查结果中获得一些启示。

原文链接： http://www.infoq.com/cn/news/2010/08/tiobe-smalltalk

相关内容：

- 微软.NET编程语言的未来
- 编程语言：2008 年回顾和 2009 年预测
- 在多核GPU和CPU上使用Accelerator V2 执行并行编程
- 面向对象编程——走错了路？

Adobe Flash Builder 4 简体中文正式版

隆重发布

出众的创新，合适的技术

Adobe Flash 平台是一套全方位的技术，外围是一个由支持程序、业务合作伙伴和热情的用户社区构成的可靠的生态系统。它们共同为您提供了为最广的观众群创建和交付最引人注目的应用程序、内容和视频所需的一切。

现在下载使用
即可体验更多最新特性，开发更炫RIA应用
更可优先免费获得产品序列号
数量有限，先下载先获得！

InfoQ中文站提供以下Adobe Flash Flatform相关产品和工具的高速下载：

- Flash Platform一览表（PDF，231k）下载
- Flash Catalyst Beta 2 292M 下载
- Flash Builder 4 285M 本地下载 Adobe 官方网站下载
- Adobe Flex Builder 3 本地下载 Adobe 官方网站下载
- Flash Player 下载
- Flex 4 SDK 下载
- Adobe AIR 15M 下载

立即下载

特别专题

策划：**InfoQ** 中文站； 执行：马国耀

所谓动态语言，又称动态编程语言，指的是有其编写的程序在运行时可以改变其结构，如加入新代码、扩展对象及定义或更高类型系统等。目前，JavaScipt、Ruby、JRuby、IronRuby、Groovy 等多种动态语言百花齐放，争奇斗艳。同时，人们对于动态语言的讨论，甚至是争论，也在如火如荼的进行着。有人认为编程语言的未来属于动态语言，而另外一些人则不怎么看好它们。本期专题特别挑选了 InfoQ 主站上与动态语言相关，特别是与 Ruby on Rails 相关的部分文章，寄希望于通过该专题展现动态语言的现状、动态与发展，供读者参考。

动态语言企业应用优缺点浅析 - 重构 TekPub——从 ASP.NET MVC 框架迁移到 Ruby on Rails

用 Rails 创建高质量 Web 应用 - 虚拟座谈：HTML5 来了，JavaScript 框架会如何发展

特别专题

动态语言企业应用优缺点浅析

作者 李明（nasi）

动态语言的兴起已经有些年头了。现在，人们早已不再去争论动态语言是否能够取代静态语言，因为这种争论毫无意义。越来越多的开发者开始在动态语言更为擅长的领域应用它们。比如，Django 和 Ruby on Rails 等开发框架的盛行使得像 Python 和 Ruby 这样的动态语言可以在 Web 开发领域大放异彩，PHP 和 JavaScript 也早已在 Web 开发领域占有一席之地。

不过目前动态语言在企业开发中的应用还不够广泛，很多企业只是用它来做一些粘合系统的工作，并没有承担起主力开发语言的重任。尤其是在底层系统开发方面，动态语言远没有在 Web 开发方面那么风光。在运行时效率和虚拟机稳定性方面的不足，使得动态语言注定无法与编译型语言竞争，并取代它们在高性能领域的地位。然而，动态语言也有自己的优势所在。如何克服自己的劣势，将优势发扬光大，便是每一位动态语言开发者所面临的机遇和挑战。

我所在的团队用了近两年的时间，将一个电信领域的公司绝大部分的生产系统用动态语言（主要是 Python）重写。包括短/彩信消息网关、业务订阅服务、座席查询系统、销售支撑系统，乃至搜索引擎等多个核心系统，都在重写之列。重写的理由很多，一方面原有系统无论是从性能上，还是从应对需求变化的能力上，都已经不能满足业务发展的需要；另外一方面，动态语言的诸多优势，也是我们重写的动力。这里仅以开发这些系统时获得的经验，来谈谈动态语言在应用时的优缺点。

动态语言的优势

动态语言的优势有很多，归纳起来主要有以下几个方面：

1. 生产力。动态语言在开发效率方面有着无与伦比的优势，这也与动态语言“优化人的时间而不是机器的时间”这个理念相吻合。利用传统的静 态语言要开发几周的功能和特性，使用动态语言也许几天甚至几个小时就可以实现。不仅如此，动态语言在开发原型系统和常用工具方面的开发效率也非常高，尤其值得一提的是原型系统。更快地让原型系统

运转起来，不仅可以尽早验证一些假设，也能够更好地与迭代开发相结合，更及时地与需求方进行沟通，帮助需求方挖掘和了解自己真正的需求。开发效率可以说是动态语言最为吸引人的地方，这也被认为是将来开发语言的前进方向。这些年随着敏捷开发的盛行，越来越多的开发者意识到，原来动态语言的特性和敏捷开发的价值观也相当契合：缩短反馈时间，对变化的响应能力更强。所以许多动态语言团队选择了敏捷开发的实践来组织团队。我们在实际开发的过程中，以两周为一个迭代周期，基本上 1-2 个迭代就可以完成一个系统的开发和上线，而原型系统的开发通常能在 1-3 天内完成。

2. 代码量。曾有报道说，用 Ruby on Rails 写同样的项目，代码量大概只有 Java 的 1/10。且先不说这个说法是否有夸张的成分，但就实际来看，动态语言的确从代码量上来说，要比 Java/C/C++等传统静态编译型语言要少的多（当然语言的表达能力与动态静态关系并不大，静态函数式语言的表达能力也很强），可能几千行的项目就算得上是个大项目。例如，我们开发的系统中较为复杂的消息网关，生产代码行数大概在 7000 行左右；而订阅服务系统的生产代码行数不到 3000 行；借助于 Xapian 的 Python-bindings，我们的搜索引擎系统代码行数只有 800 行左右。代码量少的好处非常多：首先，这意味着将来需要花在维护上的代价更低；其次，因为代码少了，犯错误的机会也就少了，因此代码量少还意味着 BUG 的减少；再次，代码量的减少也有助于优化系统，有句话说的好，“There is no code faster than no code”。开源项目 Nebula Device 有一个设计哲学，就是“每一次 Release 要比前一次代码量更少”，窃以为高论也。

3. 测试。因为动态语言很容易实现反射等动态特性（JUnit 也是等到 Java 支持了反射以后才出现的），因此测试也更为容易实现。 Python 和 Ruby 的标准库中都带有 unittest 的框架，这几乎可以让你无成本地使用单元测试来加固代码。因为动态语言本身不具有编译过程，因此 犯下某些低级错误的几率大大增加，也为重构带来了重重困难。没有单元测试的重构如同梦魇一般，动态语言尤甚。因此，在开发语言以动态语言为主的开源项目 中，单元测试总是占有相当大的比重。还有建议称测试代码与生产代码的比率（Unit Test To Code Ratio）要达到 2:1 以上。另外，动态语言的测试环境更容易搭建，实现 Mock 也更为简单。

4. 原生数据结构。现在主流的动态语言多为脚本语言发展而来，而在这些语言中，集合、列表和词典这样的数据结构都是原生的，而静态语言的数据结构往往是通过程序类库来实现的。比如 Python 就提供了 set、tuple、list 和 dict 等原生数据结构，同时还提供了大量操作（如数组分片等），让这些数据结构使用起来非常方便。原生数据结构使得对数据的操控融入到了语言的语法当中，让程序更为易读，这也让基于代码的沟通更为顺畅。

5. 简单易学。动态语言的语法相对简单，学习成本看似也比较低。有人举例说，Python 和 Ruby 写个 Hello World 只需要一行即可，这是很多静态语言所达不到的（把多行代码写成一行的不算）。当然你可以认为这只不过是句玩笑话，不过单就语法而言，动态语言的学习门槛要比很多静态语言要低的多。可是，开发不仅仅只是语法而已。很多动态语言的初学者，能够用动态语言写一些简单的小程序小工具，却很难构建起庞大复杂的商业系统，究其原因，主要是还是因为系统设计和面向对象的功底欠缺所导致的。如何设计，如何抽象，如何重构，这些能力与语言无关，而是个人的修为。正如陆游所言，“功夫在诗外”，这些能力也不是一朝一夕、通过学学语言就能够轻易练就的。当然，动态语言的各种特性（如 Duck Typing）也使得在静态语言中不得不使用的设计模式可以很自然地表达，这些差异也增加了动态语言学习的隐性成本。

不足之处

任何事物都具有两面性，动态语言也不例外，虽然优势显而易见，动态语言的不足之处也有很多。这里列举一些我们在开发过程中所遇到的问题，以及一些初步的解决方案，来供大家参考。

1. 运行效率。运行效率低下使得动态语言饱受诟病。“跑得太慢”这顶帽子已经在动态语言的头上扣了许多年。甚至有 Benchmark 表明，在某些应用场景下，动态语言的运行效率和 C/C++、Java 等成熟的静态语言相比，相差数十倍甚至上百倍，这也为动态语言的普及埋下阴影。不少开发者因为运行效率的问题，纷纷表示“对动态语言很失望”。其实我倒是觉得大可不必纠结在这个问题上，原因有两点。第一，很多动态语言的应用场景使得运行效率的重要程度大大降低。就拿 Ruby on Rails 来说，在 Web 开发这个应用场景里，数据库的响应时间无疑是最大时延，与之相比代码运行时间就微不足道了。而且通过 Cache 和优化，基本上可以消除代码运行效率低对项目的影响。又如我们的消息网关系统，最耗时的部分就是网络通信和文件 I/O，而这两部分动态语言和静态语言相比并无明显劣势，运行效率的问题可以完全忽略。第二，如果遇到很耗 CPU 或者很耗内存的运算，完全可以通过 C/C++实现的扩展来解决。无论是 Python 还是 Ruby，都支持采用 C/C++编写扩展。通过这些扩展，可以极大地提高运行效率，从而弥补动态语言在运行效率上的不足。

2. **BUG** 难于发现。动态语言由于没有构建的过程，因此很多错误只有等到运行时才会发现。而这些错误很可能是些低级错误，比如拼写错误、没有 import 相关的类库，或者括号不匹配等等。如果每次修复这样的 BUG 都要通过去测试环境中部署来验证的话，则会浪费了大量时间。因此动态语言往往需要充分的自动化测试套件，才能够确保代码基本可用。

另外，使用动态语言的时候，一个良好的代码静态检查工具也是很有必要的。它不但可以纠正一些低级错误，而且还可以帮助你发现代码中的 Bad Smells，大大提高开发效率。对于 Python 来说，Pyflakes 或 Pylint 都是不错的选择；而 Ruby 也有众多工具可供使用。测试充分的代码也更容易重构，在重构动态语言项目时要万分小心，因为动态语言极容易犯错，稍不留意就会引入新的 BUG。保持小步前进的步伐，每次修改后都执行测试，最好再通过持续集成环境来帮助发现测试失败的情况，这样重构起来才能得心应手。

3. 专业人员少。不少使用动态语言的公司都会遭遇一个问题，那就是使用动态语言的资深开发人员很少，不但很难招聘到靠谱的员工，核心人员的离队也会对公司造成很大的损失。这是因为完全使用动态语言进行开发的公司少的可怜，只有极少数的开发者能够参与其中并获得相关的开发经验。绝大多数的动态语言使用者还处在爱好者阶段，跟着 Tutorials 写写 Demo，或者随手写个 Utils 等等。因为高水平的动态语言开发者的确是可遇不可求，因此寻找有经验的开发者也许要花上不少的时间和成本。当团队有了较为有经验的开发者以后，就需要通过内部培训、结对编程等手段，帮助公司里没有经验的开发者迅速积累经验，逐渐成为动态语言方面的靠谱人才。其实，对于动态语言的圈子，还有一个有趣的说法：因为学习动态语言的人往往都是在其他领域有了很深的积累后，在有余力的情况下才接触动态语言的，因此往往相对都比较靠谱，动态语言的圈子反而能够帮助雇主们甄选出一批高素质的开发者。

4. 不够成熟。动态语言的发展历史虽然不比静态语言差到哪里（比如 Ruby 和 Java 就同为 1995 年始创），然而由于其较为小众，因此无论是虚拟机的实现上，语言本身的机制上，还是相关的配套工具上都算不得十分成熟。例如，Ruby 虽然以其优美灵活的语法为人所称道，但也因为其虚拟机效率低下和内存泄露问题所为人诟病，使用 Ruby on Rails 的网站往往需要加配监控程序，一旦发现某个 VM 内存超标立刻重启；Python 的虚拟机虽然还算稳定，但长久以来一直受 GIL（Global Interpreter Lock）问题所困扰，完全无法发挥多核的优势，这在家用 PC 都早已多核的今天的确是个不小的问题（事实上 Ruby 也存在 GIL 问题）。不过，虽然官方实现不够成熟，现在已经有很多逐渐成熟的其他选择可供使用。比如 JRuby 就充分利用了 Java 成熟的虚拟机和 Ruby 优良的语法特性，还可以允许开发者使用 Java 背后庞大的类库。通过 multiprocessing 或 Stackless Python，甚至手工将任务切成多份，分发给多个进程运行，都可以规避掉 GIL 的问题，更充分地利用系统性能。当然，随着时间的推移，动态语言的实现将会越来越成熟，不但 MRI 逐渐完善，MagLev 和 Rubinius 等一系列优秀的 Ruby 虚拟机也开始登上舞台；Python 3000 甚至打破了向后兼容性，试图将 Python 以前的设计错误全面改写。回头去看 Java 等一批成熟开发语言的发展路线，有谁没有经历过不成熟的青春期呢？

小结

通过实践我们发现，动态语言既不是什么洪水猛兽，也不是什么奇巧玩物，它们已经逐渐成长为称手的兵器，帮助开发者们快速完成项目，进而达成商业目标。使用动态语言，已经让我们切切实实感受到了它的开发效率为我们所带来的好处。在商业机会瞬息万变的今天，谁能以最快的速度实现自己的想法，谁能尽快应对市场带来的变化，谁就能离成功更进一步。

诚然，动态语言目前还存在很多问题。但瑕不掩瑜，如果在使用时可以意识到这些问题，并善加处理的话，动态语言也可以成为复杂商业系统的主角，在企业开发中占据自己的地位。而且随着开源社区的努力，很多问题正逐一被解决。我们有理由相信，在不远的未来，动态语言一定会有一片更为广阔的天空。

感谢田乐（Tin）和赖翥翔（Jason Lai）对本文提出了大量的反馈意见，感谢霍泰稳为本文找到如此贴切的标题。

特别专题

重构 TekPub——从 ASP.NET MVC 框架迁移到 Ruby on Rails

作者 Robert Bazinet 译者 池建强

TekPub 是一个面向开发人员的站点，致力于为开发人员提供一系列主题的在线培训，主题范围非常广泛，从微软的 O/R Mapping 框架 Microsoft Entity Framework，到如何使用 Ruby on Rails 技术编写自己的日志引擎等内容都有涉及。该网站是由前微软员工 Rob Conery 与 Lounge 的老板 James Avery 创立的。

TekPub 是个很有趣的学习案例，公司开始时使用 ASP.NET MVC 框架，之后很快迁移到了 Ruby on Rails 上。InfoQ 与 Rob 和 James 探讨了这次迁移之旅。

InfoQ：和我们谈谈 **TekPub** 吧，对于哪些不熟悉你们的产品的读者，**TekPub** 意味着什么？

James Avery（简称 **JA**）：TekPub 为程序员提供了高质量的技术视频演示。我们的目标是帮助一些人在几个小时内学习一项新技术，主要方式就是观看一些牛人的演讲视频。他们对演讲的技术非常了解，演讲内容不仅仅覆盖基础知识，还深入到了这些具体技术在真实项目中的应用。与其等几个月才拿到一本很可能已经过时的书，还不如订阅我们的产品立刻获得新技术的提升。我们已经完成 ASP.NET MVC 2 的系列视频，但目前还没有与这个主题相关的书籍。

InfoQ: James Avery 和 **Rob Conery** 有什么来头？

JA：除了 TekPub 之外，我还搞了几个科技创业公司。我运营了 Lounge 和 Ruby Row 广告网络，分别关注.NET 和 Ruby 开发人员。我还帮助运营 DotNetKicks 公司，一个.NET 开发人员的社区网站。我最新的关注点是 Adzerk，这是我自己构建的 Web 服务器，用来有效的运行 Lounge 和 Ruby Row，让其他人来使用。从 90 年代中期我就开始使用 Web，.NET 发布后，我开始转移到微软的技术上。最近我一直再用.NET、Ruby on Rails 和 MongoDB 等，还有任何让我感兴趣的技术。

Rob Conery（简称 **RC**）：我自 1991 年以来一直从事软件行业，做一些数据库，CGI 和 HTML

之类的开发。从 1997 年开始 使用 ASP，从那之后一直坚持使用微软技术。2006 年我开始为微软工作，主要职责是帮助人们学习和使用新的 ASP.NET MVC 框架。2009 年我离开了微软开始做些不同的事情（更多的关注开源平台），之后和 James Avery 一起创办了 Tekpub。

InfoQ：能介绍一下你们刚创业时 **TekPub** 的架构吗？

JA：TekPub 的第一个版本是基于 Ruby on Rails 构建的，当时 Rob 花费了整整一个周末的时间。之后我们仔细的进行了讨论，最终决定放弃这个版本。由于我们都很了解 ASP.NET MVC 技术，所以决定用 ASP.NET MVC 来实现 TekPub。我相信，当开始一项新业务时是没有时间尝试和学习新技术的。Rob 和我过去都写过 Rails 应用，但真正使用时，我们都觉得 ASP.NET MVC 比 Rails 好得多。于是我们放弃了 Rails 转而使用 ASP.NET MVC 技术开发网站。后来我们取消该站点并重新构建它，因为它已经变得太复杂。在我们开始进行技术迁移时，TekPub 的版本是 3，主要技术是 ASP.NET MVC，C#和 MS SQL Server。

InfoQ：从网站使用者的角度你遇到了什么样的挑战？从网站管理者的角度呢？

RC：最初，或者说网站运营的第一天，我们就遇到了网络带宽的问题。我们在 Twitter 上宣布网站开张，这直接导致我们的 ISP 由于带宽的需求被淘汰。他们毫不夸张的让工程师“坐在开关上”来保证网站的正常运行，但最终网站还是停了。之后两小时内我把我们所有免费的内容都放到了亚马逊的 S3 上，这对我们帮助非常大。

还有，一小部分人不喜欢 Silverlight，不愿意安装它。我们在 Reddit 上投放了广告，人们却简单的认为我们的网站是微软资助的一个什么东西——这是和我们的目标背离的。对我们来说这是个大问题。

JA：ASP.NET MVC 运行的很好，Windows 架构也不错。事实上我们在这个领域没碰到什么麻烦。主要的挑战来自于我们决定使用 Silverlight 播放流媒体。很多用户不愿意安装 Silverlight，这给我们带来很大的困扰。与 Rob 确认后，我们不得不迁移到 Flash 技术。迁移到 Flash 之后还没有一个人抱怨过。我们希望 HTML5 很快面世。

InfoQ：架构如何应对用户需求？

RC：我们从来没有遇到底层框架的问题——它处理的很好。这并不是一个负载非常大的网站（在功能方面），所以我们从没真正遇到那方面的问题。

InfoQ：既然平台运行的非常好，为什么要做架构的改变呢？

RC：成本。我们加入了微软的 BizSpark 计划， 而且它的确给了我们一个很棒的起点，但是随着项目的发展，我们发现，3 年以后，从我们使用的数据库到开发环境，每一件事情都需

要付费。而且我们很可能需要 为 Silverlight（使用流媒体）提供一个分离的服务器，用来播放视频，这样就需要再支付一个使用许可的费用——此外，为了使视频流平滑输出，我们还需要买媒体编码器。

对于大公司来说，这的确不算什么，但当我们坐下来看帐单时——噢，这可是个 5 位数啊。最终我们发现，根据我们现有的业务规模，是无法承担这笔费用的。

不仅仅如此，James 和我都清楚 Rails 非常好用。我们意识到可以把所有事情都推到云端，只需支付很低的价格就可以获得很好的流媒体和吞吐量。

JA：正如 Rob 提到的，成本压力是其中一个因素，BizSpark 是很好，但却像一个定时炸弹。我觉得比成本更重要的推动因素是每天我们希望使用什么技术。ASP.NET MVC 和.NET 技术在某些领域是有缺陷的，而这些领域的工作对我们非常重要。在.NET 上做测试就并不让人满意，你不得不大费周章才能以正确的方式设计你的应用，处理测试，而且编写测试本身也不像其他语言那么清晰和有用。另一个问题是部署，虽然有很多种方式可以处理，但确实不如 Capistrano 好用。

InfoQ：那么现在 **TekPub** 平台是什么样子的？

RC：我们迁移到 Rails 2.3.5，使用了针对 MongoDB 的 MongoMapper 技术。我们做了一个报告设置，用 MySQL 来跟踪反馈使用 DataMapper 的情况。我们还植入了 New Relic RPM，来跟踪我们的站点和健康情况——所有的这些，平均成本仅仅是我们使用 BizSpark 的 1%。

JA：这解决了我们所有与使用许可有关的问题，我们每月需要为 Amazon EC2 平台上的一个 Ubuntu 实例支付$80，这是保留实例后的费用。Rob 和我都很喜欢这种技术，我们使用 Cucumber 做了大量测试，用 Capistrano 部署变得非常简单容易。

InfoQ: 能给我们讲讲你们的技术架构细节吗？

RC：使用 Rails 2.3.5 及其对应的 MongoMapper/MongoDB，使用 MySQL 5.2 和 O/R Mapping 工具 DataMapper。我们还用了 New Relic RPM 工具，非常棒。

JA：我们选择使用 Rails 2.3.5 和 Ruby 1.8.7，通过 Passenger 运 行 Rails 应用。在数据库方面，我们通过使用优秀的 MongoMapper gem 访问 MongoDB 1.3.4。我们还通过 DataMapper 实现 MySQL 5.1 的数据访问。我们并不使用很多其他的 Gem，只是用 Pony 发邮件，rpx_now 访问 RPX 做身份验证。同时我俩还都是 HAML 的爱好者，所有的界面都是基于 HAML 实现的。

InfoQ：你们的 **EC2** 怎么配置的？

JA：所有的内容都是托管在云端，相当于一个比较小的服务器，也就是一个单独的 EC2 实例（虚拟双核，2 个计算单元，7G 内存）。在这个实例上运行着 Ubuntu 9.1（karmic）和 Apache。我希望未来有更多的资源，你知道，运行在 EC2 上，想增加资源是非常容易的。

InfoQ：为什么选择 **Ruby on Rails**？

RC：James 和我都知道，插件的世界是非常吸引人的。例如 New Relic，简直是上帝送来的礼物。我不担心服务器挂掉，而且我能看到所有代码的“瓶颈点”。他们还能跟踪最高发生频率的问题，甚至会提出改进意见。

MongoMapper 是一个令人难以置信的数据工具，而 MongoDB 本身的速度比闪电还快。Rails 的平台已经成熟到了这样的地步，几乎难以说服自己*不*使用它。

JA：我们遇到的所有问题（使用许可、测试和部署）都得到了很好的解决。我们还能使用一些变通的办法，例如写自己的部署框架等。随之而来的是我们都很享受与 Rails 工作，我们喜欢 Rails 社区，还有很多工具和类库可用。经过迁移之后最棒的地方就是，这一切让我们变的非常高兴。

InfoQ：相对很多传统数据库例如 **MySQL** 或者 **PostgreSQL**，你们为什么选择 **MongoDB**？

RC：速度和扩展性。使用 MongoDB 非常简单——不用担心迁移，非常灵活。而且它不可思议的快——这一点对用户带来的感受是巨大的。

JA：事实上我们都用了，我们用 MongoDB 做那些它最擅长的（灵活存储产品、用户和订单等信息），同样我们也这么使用 MySQL，我们用 MySQL 提供正在发生的持久事务日志。

InfoQ：完成了新的设计和重写应用之后，最直接的好处是什么？

RC: 最直接的——第一件事就是人们发现网站变快了。他们还注意到一个“更严格”的设计，诚实的说，确实有些繁琐，但它让我们能更多的关注服务器端。

不仅仅如此——当出现问题时，我们可以在几秒钟之内消除它。这是由于基于 Capistrano 的部署是如此简单。问题的修复、推出——仅需几秒。

我们的测试套件使用的非常好——使用 Cucumber 做一些事情简直是一种享受，但用 ASP.NET MVC 就会很困难。例如——使用 ASP 测试 PayPal 的支付接口时就会很麻烦。

基于 Rails/Cucumber/Webrat，在这样的框架下要做的就是填空并提交——然后确认所有事情都是按照计划执行的。我想我们的用户会看到很多功能在按照预期的方式运行——这确实很棒。

最直接的好处是我们想做一些变化时，可以快速完成，并且这种响应速度直接反应到了产品上。在原来基于 ASP.NET MVC 技术的网站上，我经常会写些 SQL 脚本来处理一些复杂的支持需求，比如改变某人的 OpenID，或合并帐户什么的。但现在我只需花 30-60 分钟的时间写了几行的 Ruby 脚本就能完成。

在重写网站过程中另一个重大的改变之一是如何处理用户的文件。过去我们从自己的服务器或运行在 EC2 上的 Wowza 媒体服务器上以流的方式处理文件。这两种方案都可以，但是太贵了，而且在其他国家就运行的没那么好了。这次我们开始重新设计亚马逊服务，提供了从云端和云端流来处理私有内容的能力。现在我们的全部内容都是通过云端处理，并使用了安全签名的 URL，这就意味着世界上任何地方的用户都可以进行高速下载。这也节省了我们的成本，因为我们只需支付使用带宽的费用，不需要额外的设备或许可费用。

InfoQ: 在你们当前的实现中，**TDD** 测试方法包含哪些内容，引入模拟对象了吗？

JA：我们用 Cucumber 和 Pickle 实现测试驱动开发。这方面 Rob 应该比我更清楚。

RC：没有进行模拟测试，都是 BDD（行为驱动开发）。我痛恨那种拖拉的感觉，就像我需要做“代码覆盖率”一样——事实上我们有个商业应用在运行，我更需要确认用户获得良好的体验 这对我来说就是 BDD。Cucumber 扮演了一个真正的重要角色——就像 Pickle 和 Factory Girl 做的那样——它与 MongoMapper 配合的好极了，对比我在微软框架下做的和我现在能做的事情，这让我高兴的笑了好几次。我几乎完全在考虑业务——甚至没有注意到测试方法这回事。

InfoQ: 你们现在的测试方法和以前在微软使用的方法有什么不同？

RC：100%不同。测试微软的东西是极其痛苦的，当时你很穷，还有三个月钱就花完了，在这种情况下，测试就是变成了一个很难判断的的东西。不可否认——我为我们的第一次修改几乎写了一吨的垃圾。如果你没有客户，那么你的应用系统好不好用就无关紧要，我真正需要确定的是不出现那些不可预期的错误。所以我进行了一些测试，但并没有得到想要的结果。我们对 PayPal 的东西进行了大量地测试，这是你必须要做的。当开始使用他们的 API 时，我在他们的沙箱环境中折腾了整整三天，还是会不断的出问题。

另一边呢——感觉是黑夜和白天的对比。现在我们覆盖了所有的行为类型，我想 James 甚至抽时间写了一些。

JA：对微软产品的测试总是像打一场战争，主要是因为 C#和静态语言对测试的支持不是很好。在.NET 上测试时你总是需要在各种循环中跳来跳去。话虽这么说，.NET 的 SpecFlow 框架看上去还是很乐观，但我对它能解决.NET 的测试方面的所有问题持怀疑态度。

InfoQ: 在微软框架上做测试会有多少困难？为什么会这样？

RC：语言和工具造成的。假如微软对这一点想的多一些，测试可能会更容易一些。使用 RSpec 进行测试，没有其他语言比 Ruby 做得更好了——它让测试变得非常容易。微软可以利用 DLR 实现这些功能……但是他们没有这么做。因为这不符合“.NET Story”，那么好吧，这是他们的商业决策，我们这次的迁移也是一样。

InfoQ: 在这次全面的重新设计中，你们有什么经验教训吗？

RC：没什么——我想 James 和我都知道，我们到了需要调整和重写的那个点。这是我的第三次创业，对 James 来说应该是第五次。开始时我们经过很长的时间讨论采用什么技术。我们都同意我们要“跟随你所知道的，并且去实现它”——于是我们就做了。

我们很幸运把握住了这个机会——然后我们进入了这样一个状态：“好吧，让我们开始为之后的 3-5 年规划，进入可控的增长模式”——这就是我们现在的状态。我们不想做得很大，我们不想引入 IT 人员或一组开发人员。我们想自己做而且保持小的规模——重点放在了我们的创意上。

对于我们这样的两个家伙来说，Rails 超级简单，易于维护。我们有世界上最可靠的支撑（可自动扩展的亚马逊 EC2），我们对正在做的事情非常满意，获得了很多乐趣。

JA：我想大部分人看到这个故事，会认为我们应该从开始就使用 Rails，事实上我很高兴我们没有那么做。开始创业时我想我们要相信自己的直觉，哪些是当时我们认为最好的，我们应该关心所有的业务问题，而不是使用什么工具。现在业务已经正常运转，在这样一个坚实的基础之上，我们就有了很多美妙的时光，释放内心的渴望做一些好玩的事情。平台移植让我们享受了很多乐趣。

InfoQ: 你们现在的架构有什么缺点吗？如果今天开始重新做一遍的话，有什么改变吗？

RC：从我的观点来看没什么——我爱现在的一切。对我来说这很不可思议——我在很长一段时间内曾是一名微软员工，并被扣上了这样一顶帽子……其实这对我们来说并不意味着什么。我非常喜欢硬件成本的可扩展性——我想在很长时间之内我们不再需要其他的硬件方案了。我们的服务器非常灵活，完全根据需求动态扩展。使用微软技术你就没法做到这些（据我所知），如果你想搞一台独立的机器，那你就需要支付一大笔款项。

JA：针对现有架构，我唯一想做的改变就是把 MongoDB 从我们的单一服务器上分离出来，同时运行两个 MongoDB 对我们来说是有意义的，但现在这部分功能被废弃了，对我们来说更有意义的事就是等 MongoDB 开发新的配对策略。但是考虑到额外的两个虚拟机的成本，

以及我们的业务在现有的单一服务器上运行良好，我们暂时还不准备这么做。

InfoQ: 感谢 **Rob** 和 **James** 抽时间接受我们的访谈。

关于 TekPub 的更多信息，请参考公司网站。

特别专题

用 Rails 创建高质量 Web 应用

作者 胡振波

越来越多的企业开始选择 Rails 作为 Web 应用的框架。Rails 曾经还主要是一些轻公司的选择，但今天一些“重”企业（比如保险、金融等行业的企业） 也开始把 Rails 纳入内部应用甚至外部应用的考虑范围。我最近服务过的客户是国外某大型保险公司，该公司就选择了 Rails 来创建他们的保险销售网站。

选择 Rails 的原因，是因为它快速构建的能力，是因为它是 Web 开发的 DSL。但是否选择了 Rails 就代表了高效开发？是否在 Rails 上创建的 Web 应用就一定是高质量的？答案是否定的。从我参与过的几个 Rails 项目来看，质量可谓是参差不齐，开发速度也是判若云泥。而开发的效率低下的原因，则常常是应用本身质量的低下和设计的拙劣。

在本文中，我将逐一讨论几个影响 Web 应用质量的因素。同时，我们也可以从中领悟到 Rails 为创建高质量的 Web 应用所做的努力、它的各种设计给我们的启示，以及 Rails 3 的改进所代表的意义和趋势。

MVC

我们都知道 Rails 是一个 MVC 架构模式的 Web 框架，MVC 各部分的职责也很清楚。但问题在于我们是否真的遵循了 MVC 架构模式做到了各部分职责的明确分离？是否遵循了单一职责的原则？

在大多数代码里面，这种混沌不清的状态存在于 model 和 controller 之间：controller 承担了太多本应由 model 承担的职责。其中比较典型的例子是内嵌（多）对象表单。比如，Album 和 Photo 之间是一对多的关系，我们要创建一个含有多个 photo 的 album。在 Rails 2.3 之前，我们可能会写出类似的代码：

```
AlbumsController
  def create
    album = Album.new params[:album]
    album.photos << Photo.new params[:album][:photo]
    ...
```

如果是一个涉及更多种类型对象的表单，这里的代码可能会更加复杂。但在 AlbumsController 里面，我们真正想关心的只是 Album 的创建，而不是 Photo 或其它关联对象的创建。而且从 Album 的角度看，创建过程中 photo 跟其它 attributes 没有区别，应该得到一致地处理。

Rails 2.3 之后，我们就可以很简单地达到这个目的。在 Album 里面做这样的声明：

```
class Album < ActiveRecord::Base
  ...
  accepts_nested_attributes_for :photos
end
```

然后，controller 中的代码就可以被简化为：

```
AlbumsController
  def create
    album = Album.new params[:album]
    ...
```

从这个例子中可以看到 Rails 在推进 MVC 三部分之间职责明确上所做的努力和进步。很多人可能会说，我们的代码没有这样的问题。但 MVC 三部分之间职责开始模糊，往往出现在业务逻辑变得复杂之后。我们应该经常审视我们的代码，做到真正的职责单一。

REST

现今的互联网应用已经很难是一个独立的个体，互联网应用之间的交互越来越多。所以，建立 REST 架构风格的互联网应用变得越来越重要。Rails 的 router 很好地支持了 REST 风格的外部接口设计。Rails 3 所做的一个很大改进就是 router 的改进，以强调 REST 风格的接口设计。

REST 也让我们以资源的角度看待应用中的数据，我们的代码设计因此也产生了一些变化。当需要增加一个 invoice 的 PDF 文件下载功能的时候，我们一般会向 InvoicesController 添加这么一段代码：

```
InvoicesController
  def download_pdf
    ...
    send_data(generate_pdf(@invoice), :type => 'application/pdf')
  end
```

这段代码至少存在两个问题：第一个问题，就是我们前面所述的职责明确问题。PDF 的 generate 属于 Invoice 而不是 controller 的职责，所以我们应该把 PDF 生成的逻辑移到 Invoice 内部。第二个问题，则是语义是否恰当的问题 。如果我们以资源的角度看待 Invoice 的话，PDF 跟 HTML 或者 XML 一样，只是 Invoice 的另一种表现形式而已。而表现一个资源，在 show action 中处理最为恰当。

重写之后，代码如下：

```
def show
  ...
  respond_to do |format|
    format.html
    format.pdf { render :pdf => @invoice.pdf }
  end
end
```

重写之后的代码不仅更符合 REST 的风格，而且更加简洁优美。

JavaScript

随着 RIA 的普及以及 HTML5 时代的即将来临，JavaScript 的江湖地位正在与日俱增。从 Google 的一些应用就可以看出业界对于 JavaScript 态度的一些变化。比如 Gmail，它提供了在无 JavaScript 支持环境下的普通版本和有 JavaScript 支持的全功能版本——这是一种渐进式增强的设计理念。但随后几年推出的 Google Doc，已经完全放弃了对无 JavaScript 环境的支持。从这些变化可以看出，JavaScript 已经是 Web 应用的"必需品"。甚至有人把 JavaScript 称为当今最重要的编程语言，从某种意义上这种说法也不过分。

很久以来，我们一直以"脚本"的态度看待 JavaScript。程序员对 JavaScript 的重视程度很不够，业界对程序员的 JavaScript 能力要求也不高。现在，必须做出这种态度的转变。

Rails 3 所做的很大一个改进就是：Unobtrusive JavaScript（非侵入式的 JavaScript），以实现对 HTML 和 JavaScript 代码的分离。比如：

```
<%= link_to "delete", album_path(@album), :method => :delete, :confirm => "Are you sure?"%>
```

在 Rails 3 之前，它生成的代码应为（代码进行了省略）：

```
<a href="/albums/1" onclick="if (confirm('Are you sure?')) { var f = document.createElement('form'); f.style.display = 'none'; this.parentNode.appendChild(f); f.method = 'POST'; ...
```

可以看到，生成的 HTML 内嵌了大量的 JavaScript 代码，这是一种不好的做法。Rails 3 所做的其中一个改变，就是分离 HTML 和 JavaScript 代码，生成的 HTML 中内嵌的 JavaScript 代码消失了：

```
<a href="/albums/1" data-confirm="Are you sure?" data-method="delete" rel="nofollow">delete</a>
```

那么 JavaScript 代码到哪里去了？它们都被放到了一个叫做 Rails.js 的文件中。

跟服务端 MVC 要求职责分离一样，这个原则也应该体现在客户端的代码上。HTML、CSS 和 JavaScript 应该职责明确地各自负责数据、显示和行为。同时，这种分离也对程序员的

JavaScript 能力提出了更高的要求。

性能

从一个请求（Request）的数据传输角度看，数据一般会经历从数据库到服务器，最后到客户端这么一个过程（可能还有其它层次）。数据离客户端越近，响应速度肯定越快。因此，缓存是提升性能的一大利器。

而客户端缓存是离用户最近的地方。关于客户端缓存的一条原则是：不要缓存动态 HTML 页面，但永久缓存一切其它文件类型。Rails 对静态文件的处理很好地体现了这条原则。比如下面这段代码：

```
stylesheet_link_tag("application")
```

它生成的 HTML 是：

```
<link href="/stylesheets/application.css?1232285206" media="screen" rel="stylesheet" type="text/css"/>
```

其中 application.css?1232285206 的后缀是这个文件的时间戳。那么客户端就可以放心地永久缓存这个静态文静。因为文件一旦更新，客户端就会认为这是一个新的请求，即会去获取最新的文件。

Rails 还在其它很多方面提供了简便的方法使性能优化变得简单，比如服务端缓存机制等。但大多数时候，性能问题源自于我们自己的实现或者设计问题。比如对于 Active Record Query Interface 的的滥用，多数时候性能问题都可以通过完善数据库查询来得到很大的改进。对于数据库查询，我们应该经常关注几个问题，比如：获取的数据结果集中是否有大量无用数据，数据库查询次数是否可以减少等等。

用户体验

用户体验是 Web 应用非常重要的元素，Rails 作为 Web 开发的 DSL 在这方面也有很多关注。在 Web 应用中我们经常遇到的一个问题是：submit button（提交按钮）被多次点击。如果没有被恰当处理，就会引起表单的多次提交。用 Rails 的 form helper 方法可以很简单地避免这个问题：

```
submit_tag "Submit", :disable_with => "Submitting..."
```

代码中的:disable_with 选项的作用是：在 button 被点击之后把它 disable 掉，并且把 button 的文字替换成“Submitting”。一个简单的选项带来了显而易见的好处：不仅避免了多次点击

的问题，而且显式地告诉了用户表单正在被提交当中。

Rails 提供了很多便捷的方法，让提升用户体验变得非常容易。作为程序员，我们也应该对用户体验有更多关注，比如如何设计更好的交互来避免 AJAX 所带来的种种用户体验问题等等。

安全

Web 应用面临着很多安全隐患，比如 Session 定置（Session Fixation）、跨站请求伪造（CSRF）和日志信息泄露（Logging）问题。在 Rails 中我们可以用简单到只有一行代码的方式来避免这些安全问题。下面是各安全隐患以及对应策略。

Session 定置

攻击者通过某种方式强制用户使用他所掌握的 Session ID，在用户登录之后攻击者即可使用此 Session ID 窃取用户的信息。解决方案：

```
reset_session
```

在登录逻辑中添加此段代码，以在登录之前重置 session，这样便可以防止攻击者通过 Session Fixation 攻击来获得用户信息。

跨站请求伪造

CSRF 是指在页面中注入一些恶意代码或者链接——指向用户使用的其它站点，比如站点 A。当用户访问被污染的页面时，如果刚好站点 A 仍处于有效认证期，则用户在站点 A 的数据就会被侵犯。解决方案：

```
protect_from_forgery :secret => "123456789012345678901234567890..."
```

此代码会在非 get 请求中添加一个 security token，如果 token 不一致，则请求将失败。这种方式可以有效防止 CSRF，当然前提是我们正确地使用了 HTTP method。

日志信息泄露

默认情况下，Rails 会把所有的请求信息都记录在日志文件中。那么攻击者就可以通过窃取日志文件，以得到一些秘密信息，比如登录密码、信用卡信息等等。解决方案：

```
filter_parameter_logging :password
```

这行代码就可以过滤那些不希望被日志文件记录的信息，比如 password 等，从而避免通过日志来泄露敏感信息。

Web 应用还面临着很多其它安全问题，比如 SQL 注入，XSS 等等。我们应该更多关注 Web 应用所面临的安全问题，并尽可能避免。何况，在 Rails 中要避免大多数问题，方法都很简单。

业务模型

最后一个问题虽然与 Rails 甚至技术的关系并不大，但是却关系到一个 Web 应用质量的最关键问题：创建的 Web 应用是否符合业务模型。我们曾经在一个电子商务应用的开发过程中遇到这么一个问题：整个购买流程的最后一步是支付页面，用于完成支付并生成收据的 PDF 文件。产品交付之后，客户开始抱怨支付页面的性能问题：响应时间超过了容忍度。于是我们试图改进支付页面的性能，但因为支付页面涉及的逻辑和业务实在过多，性能提升很困难。

但当我们重新审视支付页面的业务逻辑时，我们发现这个页面其实包含了两部分功能：支付和 PDF 文件的生成。而从业务角度看，PDF 文件的生成不属于支付过程，而是支付完成之后的逻辑。而且，并不是所有的用户都需要生成 PDF，强制在支付的同时生成 PDF 是一种资源的浪费。所以，我们把支付页面进行了拆分：把生成 PDF 文件的功能移到了支付成功页面，而且只有在客户点击相应链接之后才会生成 PDF。简单的改动之后，不仅性能问题得到了解决，而且应用也更加符合真实的业务流程。

当我们遇到问题或者举步维艰之时，停下来思考一下：我们对业务的理解是否出现了问题，我们的设计是否出现了问题。很多时候我们都可以在这里找到答案。重新思考业务逻辑或者重新设计之后，实现可能会简单很多，甚至也许问题本身都已经不复存在了。

小结

以上谈到的各个元素关注了代码质量、用户体验、性能、设计等等问题。也许这些并没有涉及到什么高深的技术问题，但在一个项目中，我们大多数时候面临的都不是高深的技术难题，而是这些平常的点点滴滴。一个高质量的 Web 应用，正是从这些点点滴滴开始。

关于作者

胡振波，ThoughtWorks 公司咨询师。多年企业级应用开发经验，敏捷开发的一名忠诚实践者和思考者。关注编程技术、互联网发展。

特别专题

虚拟座谈：HTML5 来了，JavaScript 框架会如何发展

作者 Dionysios G. Synodinos 译者 王瑜珩

HTML 5 是万维网核心语言的第 5 个主要版本，早在 2004 年就由网络富文本应用技术工作组（WHATWG）发起。虽然标准仍在制定之中，但有些浏览器已经能够支持一部分 HTML 5 的特性了，如 Safari 4 beta。

除了更多的标记以外，HTML 5 还添加了一些脚本 API：

> 新增的特性充分地考虑了应用程序开发人员，HTML 5 引入了大量的新的 Javascript API。可以利用这些内容与对应的 HTML 元素相关联，它们包括：
>
> - 二维绘图 API，可以用在一个新的画布（Canvas）元素上以呈现图像、游戏图形或者其他运行中的可视图形。
> - 一个允许 web 应用程序将自身注册为某个协议或 MIME 类型的 API。
> - 一个引入新的缓存机制以支持脱机 web 应用程序的 API。
> - 一个能够播放视频和音频的 API，可以使用新的 video 和 audio 元素。
> - 一个历史纪录 API，它可以公开正在浏览的历史纪录，从而允许页面更好地支持 AJAX 应用程序中实现对后退功能。
> - 跨文档的消息传递，它提供了一种方式，使得文档可以互相通信而不用考虑它们的来源域，在某种程度上，这样的设计是为了防止跨站点的脚本攻击。
> - 一个支持拖放操作的 API，用它可以与 draggable 特性相关联。
> - 一个支持编辑操作的 API，用它可以与一个新的全局 contenteditable 特性相关联。
> - 一个新的网络 API，它支持 web 应用程序在本地网络上互相通信，并在它们的源服务器上维持双向的通信。

- 使用 JavaScript API 的键/值对实现客户端的持久化存储，同时支持嵌入的 SQL 数据库。
- 服务器发送的事件，通过它可以与新的事件源（event-source）元素关联，新的事件源元素有利于与远程数据源的持久性连接，而且极大地消除了在 Web 应用程序中对轮询的需求。

最近 InfoQ 利用电子邮件组织了一次虚拟座谈，主题为 JavaScript 框架将会如何发展，以便充分利用这些新的 API。此次座谈邀请了来自主流 JavaScript 框架的代表：

- **Dylan Schiemann**，SitePen 的 CEO，Dojo 的共同创建者
- **Matt Sweeney** 和 **Eric Miraglia**，来自 YUI 开发团队
- **Andrew Dupont**，Prototype 的核心开发者
- **Thomas Fuchs**，script.aculo.us 的创建者，Prototype 和 Ruby on Rails 的核心开发人员
- **David Walsh**，MooTools 的核心开发人员
- **Scott Blum** 和 **Joel Webber**，GWT 的领头人

下面是我们的问题以及各专家的回答。

InfoQ：由于 **HTML 5** 标准仍在制定之中，大多数开发人员对它的新特性并不熟悉，您认为对于 **web** 开发人员，哪些特性是最值得关注的？

> **Dylan**：HTML 5 包含很多东西，其中很多有价值的特性都已经在 Dojo 这样的框架中实现了。例如，内置的富表单控件将包含多文件上传和数据属性，这样人们就不会再抱怨 Dojo 使用非标准的自定义属性了，虽然这些属性也是合法的。最近 Peter Higgins 为 Dojo 解析器写了一个补丁，大概有 1KB 左右的代码量，以便当这些特性添加到浏览器时可以使用它们。对我来说最感兴趣的特性是 WebSocket，它是由 Michael Carter 提出，并由 Orbited 最先实现的。WebSocket 非常适合那些长连接应用，你可以将它看做是 web 安全的 TCP Socket。
>
> **Matt & Eric**：对那些把浏览器当成是应用平台的开发人员来说，HTML 5 包含了一些很具创新性的组件。但需要注意的是这有点超出文档语义领域，已经到达 DOM API 领域了，这不是 HTML 规范的必须部分。我们希望 HTML 5 规范能够限制在对文档语义的增强和精化上，而把对行为 API 的规定留给其它规范。
>
> 一般来说，开发人员应该知道 HTML5 提供的以下 HTML 相关的特性：

- 反对使用仅用于显示的元素和属性
- 更多有语义的元素
- 更多种输入控件和语义
- 自定义数据属性

开发人员看起来对 HTML 5 中的 DOM API 很感兴趣，如果它们最后能够被实现，其中的某些特性确实能够丰富我们的工具箱，成为很重要的工具。最早被浏览器厂商实现的 2D 绘图 API（通过 Canvas 元素）以及客户端存储 API 已经引发了广泛的关注，这些关注与浏览器厂商在标准确定之前就率先实现有关。但是还有很多其他的重要变化目前也在 草案之中：

- Iframe 沙箱
- 支持"getElementsByClassName()"
- "classList" API
- 音频和视频（通过 Audio 和 Video 元素）API
- 离线 web 应用 API
- 编辑（通过 contentEditable 属性）API
- 跨文档消息 API
- 服务器端事件 API

HTML 5 试图解决一些重大的问题，而且解决的不错，比我们上面列出的还要多。

Andrew：“web 存储”（客户端数据库）会被广泛使用——很多网站已经开始好好地利用它了。新的 form 控件（Web Forms 2）从一开始就存在于标准之中，我都等不及赶快使用了。服务器发送事件可以用来构造纯粹的“推”服务程序，而不需要依赖 Ajax 的不断轮询或不稳定的长连 接。我还喜欢自定义数据属性，虽然相比之下，这只是一个不太重要的特性。

Thomas：除了离线应用特性（主要指持久化存储），我认为 VIEDO 和 AUDIO 标签是最重要的新特性，因为我们终于可以离开烦人的 OBJECT 和 EMBED 了。当然这还需要一段时间，直到所有浏览器都支持，不过这确实是朝着正确的方向在发展，而对所有非正式标准的标准化工作也会让浏 览器表现得比现在更好。一个容易被忽略的特性

是 web 应用程序能够注册协议和媒体类型，以使它们自己成为默认处理程序，就像桌面应用程序一样（但是从 UI 的角度看，仍有很多阻碍，因为对计算机不甚了解的用户不懂什么是“协议”或者“媒体类型”）。

David：除非被广泛支持，否则没有哪个 HTML 5 的特性是非常值得关注的。这些概念值得去实现，但在只有少数几个浏览器支持的情况下使用是不明智的。这就是我们不使用 querySelectorAll 的原因：浏览器厂商的不同实现会导致更多针对浏览器的 hack，而不是简单的避免使用 QSA。

Scott & Joel：在 HTML5 中我最关注的是那些现在就可以使用，而不需要开发人员额外创建不同版本程序的特性。我对其 中的数据库和缓存 API 特别感兴趣，程序可以真正支持在线和离线两种模式了。另外对于移动 web 开发人员来说，HTML5 提供了大量特性来处理低速和连接 不稳定的移动网络。

另外一些 HTML5 的特性也很让人激动，例如<canvas>、<audio>和<video>，这些功能现在还需要插件来实现。目前这些标签还没有被主流浏览器广泛支持，但是随着浏览器支持的增强，使用的人也会增加。

InfoQ：您认为现有的 JavaScript 框架该如何发展，以支持像内置媒体、离线浏览以及更高级的服务器进程通信这样的新特性？能粗略说一下您所在项目的路线图么？

Dylan：我们的目标一直是弥补浏览器的不足。我们希望我们的框架能够越来越不重要（例如，实现统一的事件系统或者是 QuerySelectorAll）。我们发现在某些情况下，我们会随着时间的推移一点一点的删除代码，但是 Dojo 的用户并不会感到有任何的不同，他们 还是可以继续使用相同的 API，只是这些 API 会简单的调用本地实现。对于你提到的更高级的特性，Dojo 已经可以支持媒体和离线程序，并且支持 JSON-PRC 和 REST，甚至可以在 Perservere 这样的服务器端 JavaScript 环境中运行！

Matt & Eric：JavaScript 框架的作用是利用更丰富的 API 和透明的跨浏览器支持来改善编程环境。YUI 将会遵循 HTML 5 标准（特别是那些已经对浏览器产生影响的），并添加对老版本浏览器的支持，以便让新的功能可以在标准实现和推广之前就得以应用。客户端存储 API 是一个 例子，YUI 将要实现它以消除 HTML 5 和现有浏览器之间的不同。

Andrew：像 Prototype 这样的“中间件”框架，主要是把难用的、不一致的 API 包装为统一的、完善的接口，HTML5 并不会 （为 Prototype）带来太大的影响。HTML5 的特性会让某些工作变得简单，但是 Prototype 会一直保留“困难的方式”，直到最后

一个非 HTML5 浏览器不再得到支持。

另一方面，建立在 Prototype 上的库 - script.aculo.us，显然需要做出选择。Script.aculo.us 提供了一个“slider”控件，HTML5 也有。是否应该使用浏览器内置的 HTML5 slider？还是使用现有的纯 JavasScript 版本，以保正在所有浏览器中的外观一致？

HTML5 提供了一些特性的本地实现，而之前的多年我们一直用 HTML 和 JavaScript 来实现这些特性，这是一个进步。如果我们能够全部转移到 HTML5 而不管老版本浏览器，大多数的（JavaScript）库都可以移除大量的代码。但是在很长的一段时间里，我们还需要保留这些旧代码。

Prototype 和 script.aculo.us 并没有长期的路线图，但是我知道，当四种主流浏览器中至少有两个支持某一特性时，我就会认真考虑是否实现它。记住，HTML5 不会一次就全部实现的。

Thomas：是的，它们会进化的，最终当浏览器支持这些新特性时，它们也会支持。这对于框架来说并不是什么新鲜事，当新版本浏览器发布时，通常需要关注的是 bugs 而不是新特性。目前，对于 script.aculo.us 来说，最新的“舞台”将是 Canvas 元素，它可以平衡 Prototype 的功能。例如，insertAdjacentHTML() DOM API 可能会比我们目前插入 HTML 的方法更快。

David：我认为我们会像以前那样：从浏览器实现中抽象出 API，对那些不一致的实现进行修复。在某种程度上，我们为老版本浏览器开发解 决方案，以提供跨浏览器支持，但当我们无法实现时，我们会提供检测功能以处理这种情况（或许使用欺骗手段）。我们还不能忘记 IE6 拒绝灭亡的事实，还要过 很长的时间，我们才能完全依赖这些特性。

对于路线图，我们将会利用这些特性的优点，首先创建更小、更快的库。如果我们可以为支持 HTML5 的浏览器构建更快的选择器引擎，我们会将精力集中在那里，而不是一些非广泛支持的特性。将规范集成到我们的 API 将会提升性能并减少代码量，这在短期内是对我们的最大好处，直到更高级的特性被浏览器市场普遍实现。

Scott & Joel：对于 GWT 来说，我们会集中在那些被主流浏览器实现的特性上，但是我们也 可能会根据用户浏览器的不同而提供不同的实现方式。例如我们提供了抽象的存储 API，可以根据用户的环境使用 Google Gears 或 HTML 5。GWT 的好处是，最终用户只需要下载他们浏览器支持的特定实现，因为我们已经事先考虑了所有的可

能。

InfoQ：HTML 5 为客户端添加了非常多的重量级特性，有些人认为目前的 JavaScript 和 DOM 编程模型已经走到了尽头。我们是否需要为不久的将来考虑一个完全不同的编程模型？

> **Dylan**：jQuery 和 Dojo 的一个主要不同是，jQuery 本质上是以 DOM 为中心的，而 Dojo 还试图改进 JavaScript 的不足。像 Mozilla Lab 的 Bespin 这样的程序用于非 DOM 为中心的开发，我一直认为 DOM 是 JavaScript 开发人员的工具，而不是全部，如果有些事不能通过修改 DOM 来解决，那就不应该在浏览器中解决。坦白地说，我们已经通过不同的工具提供了不同的开发方式。
>
> **Matt and Eric**：浏览器环境允许不同的模型，而且事实上也有很多模型被开发出来。Flash/Flex/AIR 就是一个很好的例子，它与 HTML/DOM/CSS（同时）都是 web 成功的关键部分。当你使用 iPhone 上的 Facebook 程序而不是 Facebook 的移动网站时，你就在使用一种新的模型。到目前为止，还没有哪个其他的模型可以在访问性和简便性上超过它。我们应该在以后“考虑其它模型”么？作为开发人员，我们曾经有过争执，每次我们构建新程序时，都会考虑其他的模型。如果今天的大多数程序是 web 程序，那么就已经说明浏览器仍然有价值。我们认为浏览器中仍有许多未被发现的价值，聪明的改进规范（就像 ECMA3.1 那样）将会对目前的平台产生长远的改善。
>
> **Andrew**：这很难说。有些人希望浏览器解析标记；有些人希望浏览器在窗口上绘制像素。这取决于你构建的程序类型。这并非是要代替 DOM，而是寻找其他的模型来补充 DOM，这样你就可以使用最合适的工具来工作。我觉得 Canvas 和 SVG 可能就是这一类模型。
>
> **Thomas**：我不这样认为，HTML、CSS 和 JavaScript 的组合已经被证明是非常实用和通用的，每一项技术都在积极的进步，没有必要替换掉它们。
>
> **David**：怎么会在“不久的将来”呢？任何 HTML5 中能够改变这个模型的东西，都会在多年后才能出现。而且目前的模型已经很完善、很易于理解，更被成千上万的网页所使用。我认为目前还不会有任何改变。
>
> **Scott & Joel**：我认为目前的方式还没有任何走到尽头的迹象，反而发展的更快、更有组织了。
>
> 一方面，有很多理由去制造更好的浏览器（带有 V8 的 Chrome、带有 TraceMonky 的 Firefox、带有 SquirrelFish Extreme 的 Safari，当然还有 IE8）。不管你喜欢哪个浏览器，开发者都有一个更强大的平台。

同时，开发者在这个平台上可使用的工具也越来越好。当 GWT 出现时，我认为它是革命性的。而几周前我们刚刚发布了 GWT 1.6，它比最初的 GWT，甚至比一年前的 GWT 更强大。你可以看到，它的内部其实还是那些东西，只不过随着它的成熟而增加了一些工具。

因此我认为还有很大的发展空间。

InfoQ：有人建议使用另一种客户端语言，例如 **Ruby**。您认为 **JavaScript** 目前是否足够强大？是否需要做出大的修改？是否应该使用更多的 **DSL** 技术？

> **Dylan**：我想我们很可能会看到 DSL，甚至在服务器端也会有 JavaScript。有一个服务器端 JavaScript 讨论组对此有着 极大的兴趣。JavaScript 本身并不是问题所在，真正的问题是浏览器对 DOM 和 JavaScript 交互的实现方式，以前的 JavaScript 引 擎比现在 Mozilla、Google、Apple 和 Opera 都要慢很多。我曾经认真考虑过如果 Python 或 Ruby 成为客户端语言意味着什么，坦白 说我觉得这并不能解决问题。

Matt & Eric：JavaScript 在 ECMA 3.1 中做出了彻底的改变，这就是目前我们真正需要的。

浏览器可以自行选择实现其它的脚本引擎。不过既然它们按照规范实现了 DOM API，使用什么脚本语言也就无所谓了。一些人——包括 Yahoo 的 Douglas Crockford - 曾经争论过将来的 Web 是否需要一种新的内建安全机制的语言。

Andrew：完全的语言替换是不会发生的 - JavaScript 背后的力量很强大。我喜欢已经流产的 ES4 提案中的很多新特性——运算符重 载、Ruby 风格的 catch-all methods 等等。不幸的是，ES4 包含的太多了。我很庆幸在“妥协”后，ES3.1 包含了 getter 和 setter。但是我认为 ES 3.1 做的还不够。简单来说，我觉得应该尽量让 JavaScript 更加动态。

Thomas：是的，我觉得 JavaScript 就应该是现在这样。它不应该成为一个“真正的面向对象编程语言”，它强大的基于原型的机制已经很好了。这可以让我们使用不同的开发风格，并根据个人喜好进行定制。JavaScript 和 Ruby 有时候非常相似（JavaScript 框架 Prototype 中的某些部分就是在向 JavaScript 中添加 Ruby 风格的元素）。更多的 DSL 将会很好——我最希望未来在 JavaScript 中 能有两样东西，一样是捕获未定义函数名（就像是 Ruby 的 method_missing），另一样是内联函数（blocks）以避免总是需要写 function(){...}。

David：JavaScript 是这个星球上最成功的编程语言之一。尽管有些语言没有那么多的

问题（我们知 道 Valerio 喜欢写 Lua 代码，他已经爱上 Lua 了），JavaScript 真正的问题一直是浏览器的实现。框架为我们解决了其中的大量问题，但是显然 JavaScript 规范应 该得到更新。框架的目的有 3 个：1）抽象出浏览器的不同，并支持老版本浏览器；2）提供丰富的、更方便的 API；3）提供规范中没有的功能（例如效果、可 排序表格、图册）。

对于浏览器的错误实现，我们需要 1），对于仍不好用的 API，我们需要 2），对于规范无法提供丰富的功能，我们需要 3）。我们只是没有发现这些东西在近期有任何改变。

Scott & Joel：我认为 JavaScript 作为一种语言非常强大，甚至有些时候太强大了。你有 64 位整型数或为金融程序而内建的 BigDecimal，但是 JavaScript 面临的最大挑战在与如何构建和管理庞大的手写源代码库。当我们最初创建 GWT 时，我们打赌 JavaScript 为人们喜欢的巨大灵活性也可以使它成为一个优秀的编译器目标语言，因此也可以将它当成是 Web 上的某种汇编语言。

你可以在 JavaScript frameworks 和 Rich Internet Applications 找到更多信息。

推荐文章 | Articles

一种适用于真实世界 BPM 的协作方式

作者 Bernd Ruecker 译者 吴宇

我们在业务流程管理（BPM）领域里摸爬滚打已经很多年了，最近看到人们对它的关注不断提升，这是非常有趣的一件事。对这一趣事儿起催化作用方面的有，工具的日渐成熟、新 BPMN2.0 规范的形成、以及更多更好的相关出版物带来的人们对 BPM 的进一步理解，它们代表着 BPM 领域内最重要的进步。

厂商提供了越来越高精良的图形化工具以及由其承诺的业务流程实现自动化，无需任何编码甚至开发者参与；然而，我们也发现了使用这些“传统”的以厂商为中心方法的一个问题：它们并未履行任何承诺！

我们以前的一些项目可以佐证以上观点。公平起见，既然这些工具大都会面临相同的基本问题，就不具体点名是什么工具了，有个同事不得不使用一个小的 Web 界面工具来实现一个简单的流程。这个工具提供了一个特有的、神奇的可拖放界面设计器，刚开始用似乎还比较方便，但当我们项目快要结束时，出现了一些需要对表单上的数据进行细微校验的需求，而这个“神奇的”工具并不提供该功能。为了规避这些局限性，我们花了比使用简单的 JSF 实现整个界面更多的时间去搞这个设计器，不管怎样，我们最后还是搞定了。

在另外一个客户那里，某个开发者曾告诉我说，“他花了两天多时间尝试对一些特殊需求进行建模，而他本可以用 Java 在半小时内直接实现的！”还有一个客户尝试运行交易服务和有状态服务，而这不幸需要以 Web 服务方式调用服务。在使用 WS-Addressing，WS-AtomicTransaction 等标准进行实验并试图拼接一些框架之后，基本上他放弃了整套 BPM 工具。

而下一个客户在竞标初期就将厂商踢出了局，因为他们需要更多的资金以便自己提供一套 Java API。

所有这些例子有一个共同点：工具使得开发者的工作更困难而非更简单！这些工具没有减少开发所需的时间和金钱。它们对使得业务和 IT 相契合并没有提供进一步的帮助，这是因为所需的技术模型非常复杂以至于不能和业务流程模型完全相一致。你有看过 BPEL 模型（所

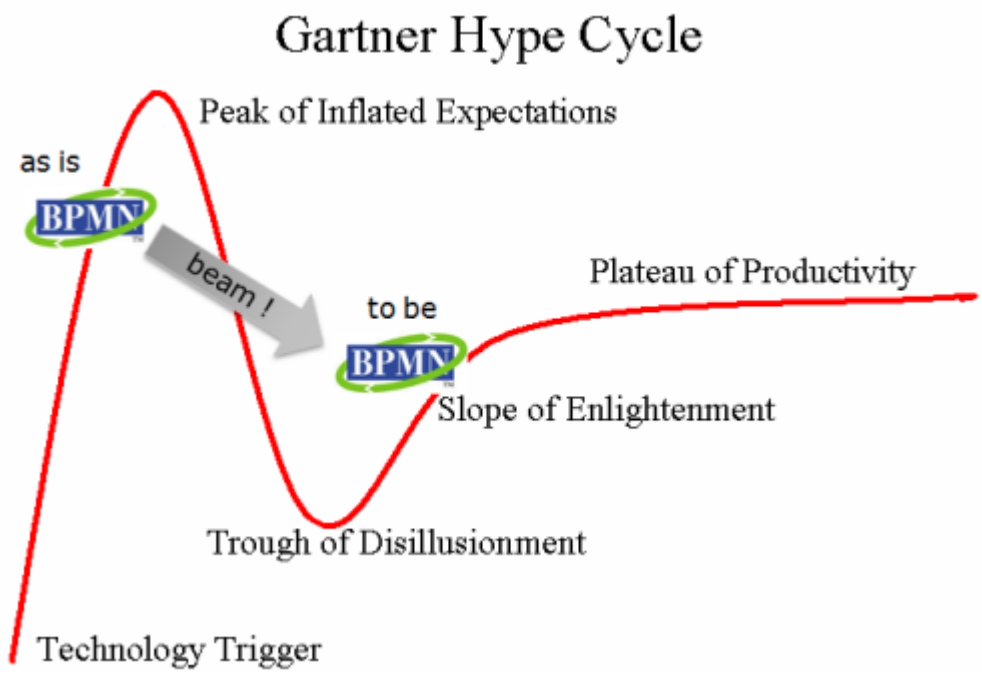
制作的流程图）和业务人员最初所画的流程图有任何共同之处的吗？

那么，BPM 不起作用吗？难道业务和 IT 契合始终是一个神话？当然不是！话虽如此，我们却不得不反思我们做 BPM 项目的方式。经过过去几年的不断反思，我们找到了 BPM 在真实项目中的运作之道。

简单说来，BPM 更多地关注流程的协作以及充分考虑协作过程中参与的不同角色，并使人们按照他们期望的方式工作。因此，它不太以工具为中心，因为就算我们需要用工具，也应该是工具来适应我们的工作方式。我们必须结束那种工具强迫我们按照厂商规定的方式来工作的情况。不同的角色使用不同的工具，所以不存在独一无二的工具。虽然这看起来显而易见，但许多工具还是试图与此背道而驰。

为了能够使得业务和 IT 相契合，我们开发了一种使用 BPMN（业务流程建模和标记）的方法论。它以流程模型为中心实现协作，我们可以进行讨论并将需求、业务规则或其他物件连接起来、使开发状态形象化、使业务驱动的测试场景得以细致明确等等。它不仅仅可以对可执行的流程进行建模，而且还对周边“池”中的各组织的视角进行建模，以使业务和 IT 视图相契合。

因此，BPMN 提供了一种很大的可能性：“池”。“池”的使用为用同一种模型来构建“业务特有的”和技术的两个方面，并为设置二者间的正确的关系提供了可能。在我们的书中对此进行了详细的描述，并且在官方 BPMN 样例文档中提供了一个例子。你也可以通过我同事写的一篇博文弄明白我现在所表达的意思。总之，我们把那些看作 BPM 的真正价值，而不是业务人员“描画”可执行的流程。



我们已经在一些客户那里实践过这种方法，有些客户甚至还在使用 Microsoft Visio 工具。当然了，在过去几个月里，我们也开发了工具在几个试点客户那印证这种方法，其中最重要的一个大项目是为一家大型德国电信公司做的。目前， 我们以通过 camunda fox 开源项目发布所有资料，其第一版本将很 快会公之于众。我们希望分享我们的经验，通过讨论吸收新观点，并帮助 BPMN 规范以得以正确采用。那样它就会为每个人创造真正的价值，而不仅仅是为那些工 具厂商的银行账户增值;-)，如果顺利的话，我们可以跳过 Gartner 的技术成熟度曲线理论中所谓的“幻灭期”，尤其针对 BPMN 2.0 而言。

让我们更具体一点看看我刚刚提到的那个电信客户项目。当时我们所面临的环境仅仅是使用了许多 EJB、一些 JMS 以及有限数量的 JBoss ESB 服务的 Java EE。流程还没有进行统一文档化，业务部门大多采用事件驱动流程链（EPC）的方式，而 IT 人员尽一切所能使用了 UML、Power Point 等等。我们的目标是针对业务和 IT，通过引入 BPMN 作为建模标记，使其成为保持模型的技术化和业务流程同步的桥梁。

我们最终达到了我们的目标！但当时是举步维艰，必须按部就班.....

技术上讲，一个“超级棒”的 JBoss SOA 平台启用，这意味着可以部署可执行流程已到 JBoss jBPM 3.2 这一著名的开源流程引擎上。我们在几次不成功的尝试使用 BPEL（其实并不适合技术环境）后做出了使用 jBPM 的决定。主要的原因根本在于那些服务 不能作为 Web Services 来用，工具也不足够成熟，价格太昂贵并且也缺乏技术诀窍。因此，jBPM 是一个不错的选择，它可以让现有的 Java 开发人员很快上手使 用，并且对开发过程的影响也非常之小。在此我要特别要指出非常重要的一点：类似 JBoss jBPM 或者不久前的 Activiti ，这样的开源流程引擎对开发人员来说更像是某种框架或库，而不是一个完整的产品套件。它们可以容易实现客户化定制并集成到你自己的架构中。它们支持单元测试，理解起来也很容易。

记住：我们希望给不同的角色提供其所需的工具。开发人员乐于接受 Java、Eclipse、JUnit、Subversion、Maven 等工具。因此他们应该能够继续使用这些工具！

业务分析人员有一个商业化工具 Signavio edition 在手，这是这些业务人员乐于使用的工具。而开发人员并不一定要使用它，因为访问存储模型的方式是多种多样的，你可以任意选择。

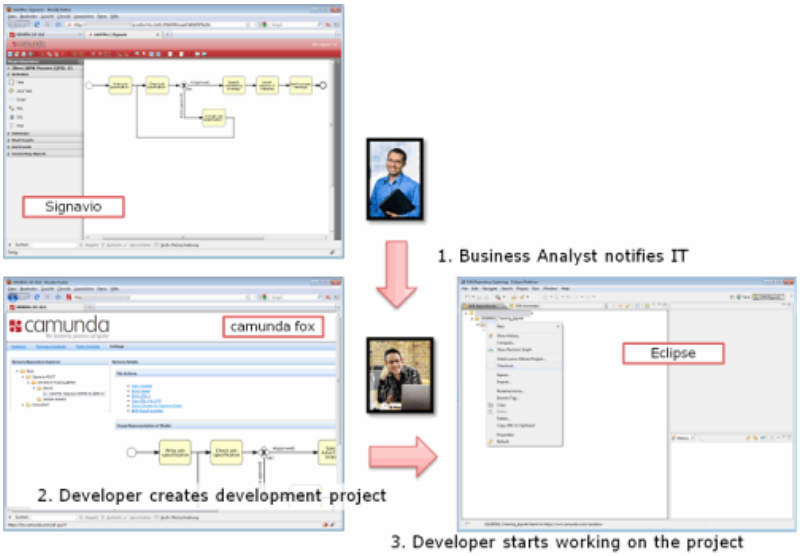
问题是：“如果我们用不同的工具而它们使用不同的存储库，那么如何才能使这些（工具产生的）模型同步呢？”解决办法也很简单：我们在不同的工具和存储库之间创建了一个粘合“层”。这一粘合包含了一个简单 web 应用，它能够访问存储流程模型的 Signavio 存储库和存储技术工件的 SVN 的。基于工件的不同 类型 ,我们做不同的特殊处理。比如从 Signavio BPMN 模型生成 jBPM 模型。这种粘合层已经成为了 camunda fox 的一部分，下面将对此展开具体

讨论。

保持业务流程模型和技术流程模型一致是非常重要的，因为这是保持流程模型版本最新的唯一方法。这极其重要，不仅仅出于归档目的，也是为了在业务流程模型一级报告 KPI（关键性能指标）或类似指标。当然，我们可以使用相对容易的粘合层达到该目标，而无需所谓的高度精良的零代码工具。

为了说明粘合层怎样发挥作用，我会列举几个截屏来演示。试想某个业务分析师创建了一个业务流程模型，并且已经完成了第一次迭代。他想要将其递交给 IT 团队以便实现该流程。他通知了技术项目头头告诉他可以开始工作了，这只需通过电子邮件发送一个链接就可以轻松搞定。好了，那现在该项目的头儿有活儿可干了，他需要对 BPMN 模型做一个操作以便在 SVN 中创建一个开发项目。这一操作可以通过项目模板来完成，而此模板是由一个从 BPMN 模型所创建的 jBPM 流程定义来生成的。我们通过一种自己实现的“特殊的映射”来实现该模板。所涉及的基础映射可以延展到全公司、整个部门、甚至是某个具体项目的规范和模式。

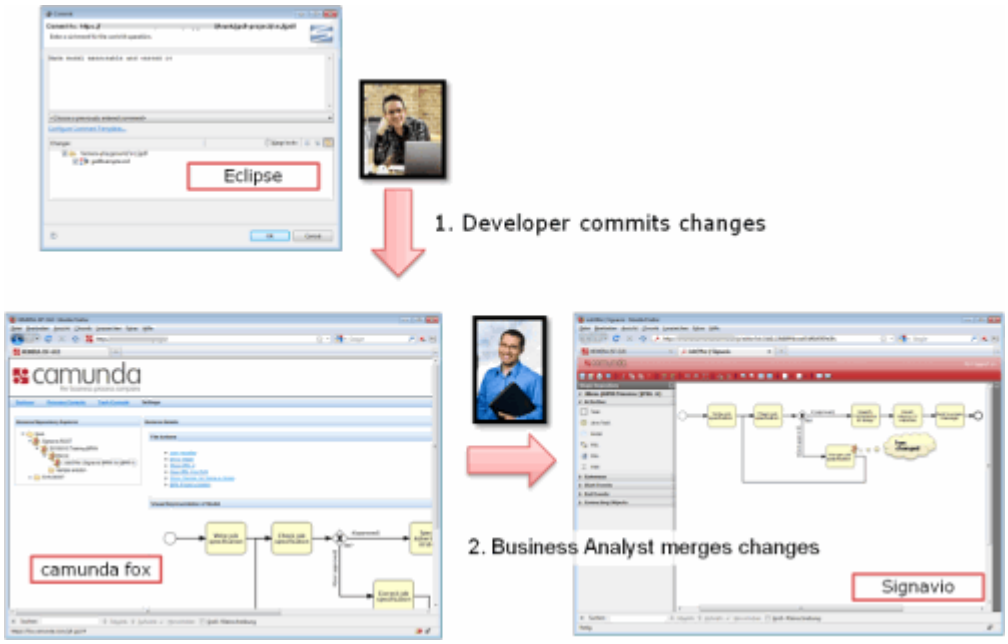
即便是 BPMN 2.0，使用映射也是有意义的，在我的博客中你也许能找到其中的原因。



接下来，开发人员开始工作，完成所有那些另人繁琐的技术细节，在流程中加入所谓的 ActionHandlers，它在流程运行的整个过程中负责执行 Java 代码。这感觉基本上和其他 Java 开发项目没什么区别了。

流程没有必要一定要部署到 SOA 平台才能测试，正常的 JUnit 测试、持续集成等等都可以。总之，常规的 Java 和 jBPM 开发都没问题。:-)

你可能已经猜到 Signavio 和 jBPM 模型之间的连接在后台保存。所以一旦开发者有更改行为，我们就可以在 camunda fox（web 应用）中看到。并不是所有更改所引起的变化都应该合并在 BPMN 模型中，所以开发人员需要向业务分析人员发送邮件或安排一个切实的会议进行通知。他们可以使用 fox GUI 来发现简单的区别并将这些更改复制回 Signavio 存储库中。

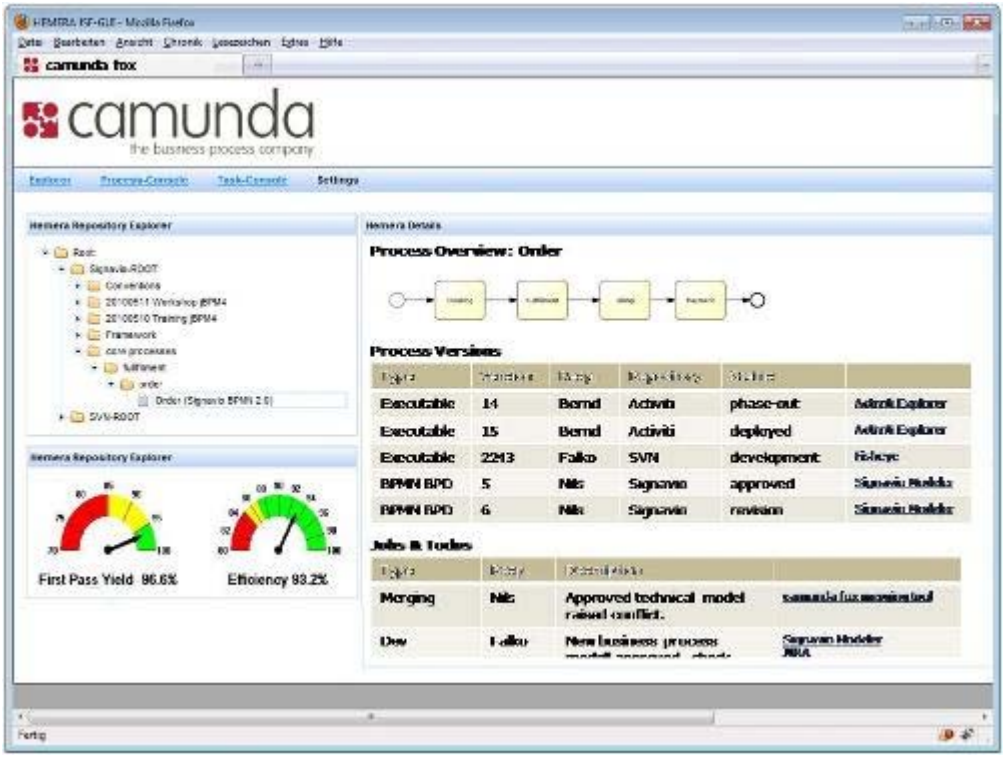


事实证明那种方法很奏效。在项目最后，我们所使用的描述性的 BPMN 图和可执行 jBPM 流程实现了同步。而大家喜欢它的原因各不相同：BPMN 在业务部门得到认可，而这种及时更新的流程描述确实大受欢迎。这种轻量级的开源流程引擎在开发过程中很实用，并且对 Java 开发人员来说很直观，我认为这是极为重要的一点。正是两者之间的粘合增加了原本缺乏的关联。与此同时，我们在另一个客户的不同项目中也尝试了该方法，并且结果证明同样非常有效。但是请记住这种方法只是个样例；如果你尝试不同的做法，尽管去尝试——使用适合的工具来配合你的需求！

接下来，我们尽力争取另一个“唾手可得”的硕果。因为在不同的模型和粘合层之间有了一个链接，我们就可以有一个中心入口点去提供流程的概貌。我们可以得知有哪些流程版本，哪个版本发布在业务端，哪个 BPMN 版本作为 jBPM 流程来部署到引擎，以及在引擎上并行

运行着哪些不同的版本等等。

就算在引擎上，也可能存在不同版本的流程同时运行。这些链接也可以显示一些用来测量运行在 jBPM 引擎上的流程在 BPMN 级别的 KPI。如下图所示。



坦白地讲，我们在这儿所做的只是冰山一角！将此作为出发点，有无限的可能对粘合层进行扩展。一个有趣的方向是对分布式团队（比如离岸外包项目）进行支持。 这种情况下你需要更多的交互功能。这些功能包括比如讨论流程模型并归档关联的讨论邮件。这离构建一个通用访问点访问存储库和工件，能够让它们互相标示、连接或与流程模型连接起来的想法只有一步之遥。这使得以 Word 文档记录的项目订单实现可追踪性，并且可以在网络硬盘上保存 BPMN 图形和技术模型。目前我们正在开发一些不同类型的标示和虚拟文档以期提高客户体验和整体感觉。你可以通过查询 Activiti Cycle Vision 继续了解这方面类似的发展方向。

一个更有趣的方向，也是我们想要在接下来的客户项目中攻克的是提供更好的流程测试支持，这在我的博文中有所描述。还记得我提到过我们目前正在做的一个数据流原型吗？它能够动态显示被处理数据的数据流，该数据在 BPMN 级别上依附于流程的 Java 类中使用或产生。

这种数据流原型会特别便于使用，如果你要实现“只需”调用服务的高度自动化流程，你会花大量时间纠结于这么一个问题：“我需要哪个数据，而这个数据是由谁在何处创建的？”最后，也是最重要的是，我觉得将流程和规则相结合具有巨大的潜力。不是什么新鲜事儿是

吧？开放的粘合层使得你很容易地在业务级别通过决策表来创建规则，并且把规则和 BPMN 流程模型关联起来。在可执行流程中，这归结为 Java 代码的一小部分，比如说，JBoss Drools，一个绝妙的开源规则引擎。虽然一些大的厂商已经提供了类似的东西，但是我们的方法也将会把这一功能引入到开源世界中。

希望我已经描述清楚了如何使得业务流程模型和技术流程模型同步这一想法。我想要概括的是有一种实用的方式创建一些简单的工具来支持你所需要的方法。此外，既然它是开源的，并且具有可扩展性，那么你就可以利用它来实现你自己的想法，扩展它以满足你的需要。我们已经有了使用这种方法的良好经验，而我们也将会继续开发 camunda fox，在未来不断的试点项目中与客户携手一起努力。

当然，camunda fox 也不是什么灵丹妙药 ;-) 但是如果你的项目里用到了（Java）开发，那我认为值得你花时间去尝试使用 camunda fox 并给我们提供反馈意见！我们每天还在学习很多新东西，而我很好奇我们最终会在哪里停步。我坚信，不管怎样，我们投入的所有精力终将会得到一个截然不同的方法用以开发以流程为中心的应用。好消息是：我认为这对大家都有好处:-)

原文链接：http://www.infoq.com/cn/articles/collaborativeBPM

相关内容：

- 书摘与采访《动态SOA和BPM：业务流程管理和SOA敏捷的最佳实践》
- Oracle BPM 11g强调流程统一、以用户为中心和社交功能
- 社交BPM，你听说过吗?
- Activiti是否有能力应对BPM的挑战？

推荐文章 | Articles

Ajax 应用开发：实践者指南

作者 成富

目前的 Web 应用开发基本上都是围绕富互联网应用（Rich Internet Application，RIA）展开。RIA 的实现技术有很多种：Ajax、Flash、JavaFX 和 Sliverlight 等。Ajax 技术的优点在于它是构建在开放标准之上，不存在厂商锁定的问题；同时也不需要额外的浏览器插件支持。Ajax 应用对搜索引擎也比较友好。对开发者 来说，Ajax 所需技术的学习曲线也较平滑，容易上手。本文简要介绍了 Ajax 应用开发的各个方面以及相关的最佳实践，但对一些细节内容没有展开讨论。

Ajax 简介

Ajax 技术的出发点在于改变传统 Web 应用使用时的“操作-等待页面加载-操作”的用户交互模式。这种交互模式会打断用户正常的使用流程，降低用户的工作效率。 Ajax 技术的交互模式是“操作-操作-操作”。用户并不需要显式地等待页面重新加载完成，而是可以不断地与页面进行交互。页面上的某个局部会动态刷新来 给用户提供反馈。整个交互过程更加平滑和顺畅。这是 Ajax 技术得以流行的一个重要原因。

Ajax 构建于一系列标准技术之上，包括 HTML、JavaScript 和 CSS 等。在这些技术中，HTML 是作为应用的骨架而存在的，展示给用户最基本的内容。CSS 则为 HTML 表示的内容提供易于用户阅读的样式。JavaScript 则为应用添加丰富的交互行为，为用户提供良好的使用体验。

Ajax 技术的出现使得应用中一部分的逻辑从服务器端迁移到了浏览器端。浏览器的作用从简单的渲染页面和表单处理，上升到了处理一部分业务逻辑。

一般来说有两种类型的 Ajax 应用风格，一种是仅少量使用 Ajax 技术来适当增强用户体验（Ajax Lite），另外一种则是大量使用 Ajax 技术来达到与桌面应用相似的用户体验（Ajax Deluxe），提供诸如鼠标右键、拖拽和级联菜单等。开发人员应该根据应用的特征选用合适的风格。

浏览器兼容性

开发 Ajax 应用的时候要面对的一个重要问题就是浏览器兼容性。虽然 Ajax 技术基于 HTML、JavaScript 和 CSS 等标准技术，但是不同的浏览器厂商对于这些标准的实现程度有着很大的差别。同一浏览器的不同版本之间也会有一些不同。新版本可能会修复旧版本上的问题，也可能会带来新的问题。不过总体的趋势是浏览器的实现越来越向标准靠拢。

解决浏览器兼容性的第一步是确定应用要支持的浏览器种类和版本。这个决策取决于应用的目标用户和特定的应用需求。对于一般的通用 Ajax 应用来说，可以按照浏览器的市场份额和支持某种浏览器所需的代价来确定。雅虎提出的分级式浏览器支持（Graded Browser Support）是一个很好的参照，从其中给出的 A 级浏览器开始是一个不错的选择。从特定的应用需求来说，某些使用了 ActiveX 控件的 Ajax 应用就只能在 IE 上运行；而开发针对 iPhone 的应用则只需要考虑移动版 WebKit 就可以了。

就 Ajax 应用的三个组成部分来说，HTML 的兼容性问题比较少，毕竟目前主流的 HTML 4.01 规范已经有 10 年的历史了；在 JavaScript 方面，JavaScript 语言核心部分基本上没什么问题，而文档对象模型（DOM）和浏览器对象模型（BOM）部分的兼容性问题相对较多，这主要是因为浏览器长生对规范的支持程度不同以及各自添加了私有实现。使用一个流行的 JavaScript 库就可以解决这些问题；CSS 方面的兼容性目前是问题最多的，而且没有比较好的库的支持。在下面会着重介绍 CSS 的兼容性问题。

富含语义的 HTML

HTML 语言本身上手比较简单，只是一些元素的集合，只需要了解这些元素及其属性的含义即可。这些元素既有与文档结构相关和富含语义的元素，也有与页面的展示相关 的元素。一个好的实践是只使用与文档结构相关和富含语义的元素。从 HTML 语言规范的历史也可以看到这个趋势。HTML 语言规范的历史比较长。在 HTML 最初的草案和 HTML 2.0 中，HTML 只包含描述文档结构的元素。在 HTML 3.2 中，很多与展示相关的元素被引入进来。HTML 4.01 规范试图解决这个问题，许多与展示相关的元素被标记为废弃的，不推荐使用。HTML 5 更进一步，引入了更多的富含语义的元素，同时移除了一些在 HTML 4.01 中被废弃的元素。应用这种实践进一步划分清楚了 HTML 和 CSS 在 Ajax 应用中的职责。

编写 HTML 文档的时候首先需要选用合适的文档类型声明（DTD）。目前来说最合适的 HTML 4.01 过渡型，即<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">。在编写 HTML 文档的时候，需要使用合适的元素。

HTML 规范 中的一些元素， 如、、<abbr>、<blockquote>、<cite> 和<code>等，对开发人员来说比较陌生。但是这些元素富含语义，有适合它们使用的场景。如果使用的是 和等通用元素，需要使用富含语义的 class 属性来增加语义，说明元素的作用。在 HTML 文档编写完成之后，最好使用 W3C 提供 的 HTML 文档验证器来验证文档。

CSS

CSS 的语法非常简单，包含的元素也非常少，其中最主要的是样式规则集。样式规则集是一系列样式声明规则的集合。每个样式规则由选择器和声明两部分组成。选择器用来选择文档中的元素。这些元素将被应用上与选择器对应的样式声明。CSS 不同版本规范所支持的选择器类型不同，尽量使用常用并且简单的选择器以获得更好的浏览器兼容性，如 ID 选择器、class 选择器和元素选择器。

使用 CSS 的时候会遇到的一个很大的问题是浏览器兼容性。经常会遇到的情况是某种样式在 A 和 C 浏览器上应用正常，而在 B 浏览器上则使用不正常。等到把 B 浏览器调好了之后，却发现 C 浏览器上显示出现错误。解决这个问题的基本原则是要首先确定几个基准的浏览器和开发基本的布局样式表。基准浏览器一般是所要支持的浏览器中对 CSS 规范支持较好的浏览器。基本的布局样式表保证在基准浏览器上应用的页面布局是正确的。目前的浏览器在 CSS 页面布局这一块的兼容性最差，尤其在盒模型（box model）、浮动定位等方面。而在显示样式，如字体大小和颜色等方面，则基本上没有什么问题。

接下来要做的是让基本的布局样式表在除基准浏览器外的其它浏览器上正确工作。这个时候就需要对某种版本的浏览器来应用特殊的样式，从而修正样式上的不一致。一种做法是利用一些招数（hack）。招数是利用浏览器本身对 CSS 规范支持的不完善或是实现上的 bug 来识别浏览器并应用样式。另外一种做法是通过 JavaScript 来检测当前浏览器并应用样式。招数可能会随着浏览器的版本更新而变得不可用，因此尽量少使用。

在一般 Ajax 应用，最常被应用招数的是 IE 6。因为 IE 6 对 CSS 规范支持不完善，而且存在比较多的 bug，但是 IE 6 的用户目前还是数量众多，还是有支持的必要。对 IE 浏览器应用特殊样式的更好做法是使用 IE 独有的条件注释。

当应用所包含的 CSS 文件比较多的时候，开发和维护这些 CSS 文件就成为一件比较困难的事情。一个解决办法是把面向对象的思想引入到 CSS 的编写过程中。两种重要的原则是组件化和单一职责。组件化的做法是开发出针对页面上某类元素的样式组件。这些样式组件可以在不同的页面中任意组合使用。单一职责指的是把表示结构和外观的样式分开。与结构相关的样式包括大小和位置，外观的样式包括字体大小、颜色和背景图片等。

DOM 查询与操作

DOM 操作是 Ajax 应用中页面动态和局部刷新的实现基础。DOM 定义了文档的逻辑结构，以及对文档进行访问和操作的方式。通过 DOM，开发人员可以在文档中自由导航，也可以添加、更新和删除其中的元素和内容。通过 DOM 规范提供的 API 就可以完成对文档的查询与操作。不过 DOM 的原生 API 使用起来比较繁琐，最好使用 JavaScript 库来完成查询和操作。

通过 DOM 操作对当前页面进行修改一般都是通过响应用户的事件而发生的。这些 DOM 操作中一部分是纯浏览器端实现的，另外一部分则需要服务器端的支持。服务器端可以选择返回数据或 HTML 片段。返回数据的好处是传输的数据量小，易于与第三方应用集成。不足之处在于浏览器端需要额外的操作来完成展示。浏览器端可以使用 DOM 操作或是模板技术来生成 HTML 片段。服务器端也可以通过 JSP 和 Apache Velocity 等模板技术来生成 HTML 片段，并直接返回给浏览器。浏览器只需要直接使用即可。这种做法的好处是浏览器端实现简单。不足之处在于与展示相关的逻辑同时存在于服务器端和浏览器端，不容易维护。

有一些比较好的实践可以提高 DOM 操作的性能。首先是使用文档片段（document fragment）。当需要插入大量节点的时候，首先把这些节点添加到一个文档片段中，再把此文档片段添加到文档上。这样可以减少页面的重新排列。其次是使用 innerHTML 来更新文档内容，速度比使用 DOM API 要快。最后是通过 cloneNode()来创建多个结构相同的元素。

事件处理

Ajax 应用与用户的交互是通过响应用户事件的方式来完成的。浏览器负责捕获用户的行为并产生各种不同的事件，应用处理这些事件。浏览器中可以产生的事件种类比较多。事件产生之后，会按照一定的过程在当前文档树中传播。事件所产生的节点称为目标节点。完整的事件传播流程是从文档的根节点开始向下传播到目标节点（捕获阶段），然后再往上传播回根节点（冒泡阶段）。当事件传播到某个节点上的时候，就会触发此节点上绑定的处理方法。（IE 只支持冒泡阶段。）需要注意的是事件处理方法中 this 所指向的对象的值，有可能是当前节点或是 window 对象。通过 JavaScript 库提供的支持来绑定事件处理方法，可以避免这些不一致。

在绑定事件处理方法的时候，可以利用事件的传播机制来减少事件监听器的数量。如当需要为一系列<li>元素添加鼠标点击的事件时，可以把该事件添加到其父节点<ul>上。在完成对事件的处理之后，可以终止事件的传播，还可以阻止浏览器的默认行为。

选用合适的 JavaScript 框架

目前存在非常多的 JavaScript 框架，有开源的也有商业的。比较流行的有 jQuery、Dojo、YUI、ExtJs、MooTools 和 Prototype 等。选用流行框架的好处是有比较大的社区支持，遇到问题的时候容易获得帮助。流行框架的文档和示例也比较丰富。使用不同的框架会给应用带来不同的实现风格。jQuery 的使用者对方法的级联情有独钟，Dojo 的爱好者则倾向于把页面上的不同部分划分成 dijit 来实现。

选用什么样的框架的因素很多，技术的和非技术的都有。一般来说，轻量级的框架，如 jQuery 和 Prototype，上手比较容易，但是可复用的组件较少；而比较庞大的框架，如 Dojo 和 ExtJs，则学习曲线较陡，但是可复用的组件非常多，适合快速开发复杂的 Ajax 应用。

构建过程

Ajax 应用也需要一个完整的构建过程。构建过程的主要目的是提高 Ajax 应用的质量和性能。这个构建过程可以包含的步骤有：

1. JavaScript 代码的潜在错误和代码风格检查。通过集成 JSLint 可以找到代码中潜在的问题。
2. JavaScript 文件的合并、缩减和混淆。通过合并可以把多个 JavaScript 文件合成一个，减少页面加载时的 HTTP 请求个数；通过缩减可以去掉 JavaScript 代码中多余的空白字符和注释等，从而减少文件大小，降低下载时间；通过混淆则是可以替换有意义的变量名称，从而进一步减少文件大小， 同时在一定程度上保护代码免被反向工程。可以执行这些操作的工具有很多，Apache Ant 就可以完成合并，JSMin 和 YUI Compressor 可以完成文件的缩减，Dojo Shrinksafe 可以进行混淆。
3. CSS 文件的合并和缩减。与 JavaScript 类似，CSS 文件也可以执行同样的合并和缩减操作，从而减少 HTTP 请求数目和文件大小。YUI Compressor 可以完成 CSS 的缩减。
4. 图片文件的压缩。通过对图片文件进行格式转换和压缩，可以在不损失质量的前提下，减少图片文件的大小。

测试

Ajax 应用的测试包含服务器端和浏览器端两部分。对于服务器端来说，测试的技术和工具都已经比较成熟。只需要根据服务器端采用的技术来进行选择即可。一个比较重 要的原则是服务器端和浏览器端尽量实行松散耦合，以方便测试。从这个角度出发，服务器端返回数

据，而不是 HTML 片段是更好的做法。可以通过工具来测试服 务器端返回的结果是否正确。

浏览器端的测试目前情况不是非常理想。已经有一些单元测试的框架，如 QUnit, Dojo D.O.H 等，也存在一些集成测试的工具，如 DOH robot 和 Selenium 等。就单元测试来说，目前对仅用 JavaScript 实现的纯逻辑代码较容易实现，而对于包含了与页面上节点交互的代码则较难实现。不管是单元测试还是集成测试，目前自动化程度都不是很高。

为了便于测试，Ajax 应用中各部分之间的耦合应尽可能的小。事件处理方法的方法体应尽可能的简单。

调试

Ajax 应用的调试一直是一个比较麻烦的问题，其主要原因是不同浏览器之间存在着各种各样的兼容性问题，同一浏览器的不同版本之间也会存在很多不同。为了在所支持 的浏览器上达到一致的效果，开发人员往往费劲了周折。目前的情况要好了不少，不同的浏览器都有了自己比较好用的调试工具，如 Firefox 上的 Firebug，IE 上的 developer toolbar 等。当出现问题的时候，可以通过这些工具来直接修改页面上的 DOM 结构和 CSS 样式来进行试验。找到正确的解法之后再用代码来实现。很多工具都支持直接在控制台输入 JavaScript 语句来执行，通过这种方式可以快速的查看程序中变量的值以及调用 JavaScript 方法来改变应用的内部状态，从而发现问题的原因。

内存泄露

Web 应用内存泄露的问题一直存在，Ajax 应用的出现把这个问题进一步暴露出来。目前很多的 Ajax 应用都是单页面应用（Single-page Application，SPA）。用户通常会在单个页面上使用比较长的时间而不关闭浏览器。在用户操作过程中产生的一些小的内存泄露会累积起来，导致浏览器占用内存不断增加，应用运行起来越来越缓慢。

面对内存泄露问题，一般来说需要注意下面几点：

1. 熟悉常见的内存泄露模式。最典型的是由于错误使用闭包造成的包含 DOM 节点的循环引用。打断循环引用就可以解决此问题。
2. 很大一部分内存泄露与 DOM 节点相关。尽量不要为 DOM 节点对象添加额外的属性，尤其是 JavaScript 方法。
3. 当内存泄露发生的时候，使用 Drip 等工具来找到发生泄露的节点并修正。

安全

Ajax 的出现并没有解决存在的一些安全问题，同时也带来了一些新的安全隐患。传统 Web 应用中存在的跨站点脚本攻击（XSS）、SQL 注入和跨站点请求伪造（CSRF）等安全问题在 Ajax 应用中仍然需要解决。对于 XSS 来说，一般的解决办法是不信任用户的任何输入。输出的时候对所有的东西进行转义（escape）。只对那些明确知道是安全的（白名单）的东西恢复转义（unescape）。对于 CSRF 的解决办法是对所有的请求添加一个验证令牌，用来确保请求是来自于自己的站点。

Ajax 带来的新的安全隐患主要与 JSON 有关。一部分 Ajax 应用的服务器端暴露 JSON 格式的数据。JSONP 允许通过<script>标签来获取数据，而不受浏览器同源策略（Same -origin Policy）的影响。不过 JSONP 可能造成数据被恶意的第三方窃取。攻击者还可能通过重定义 JavaScript 对象方法（如 Array）的方式来窃取数据。

性能

Ajax 应用的性能是一个非常重要的方面，应该在应用开始开发的第一天就把性能这个因素考虑进来，并贯穿整个开发过程。如果在开发后期才考虑性能的话，就可能与陷入一个两难的境地。一方面应用的性能达不到用户的要求，造成用户的抱怨和流失；另一个方面为了提升性能就需要对应用已有的架构做出非常大甚至是颠覆性的调整。

Ajax 应用性能的决定因素在前端。简单来说有下面几条基本的原则：

1. 减少与服务器端交互的次数与数据大小。这点主要是减少浏览器端发出的 HTTP 请求的数目和降低服务器端返回的数据内容的大小。前面提到的 JavaScript 和 CSS 文件的合并和缩减都是服务于这个目的。
2. 页面的渐进式增强。在 Ajax 应用中，HTML 文档所包含的内容对用户是最重要的，而 CSS 则帮助用户方便的查看 HTML 文档。因此这两者是要被优先加载的。JavaScript 文件可以稍后加载或延迟加载。因此，在 HTML 文档中，对 CSS 文件的引用要放在文档的上面，即<head>元素中；而 JavaScript 文件的一般作为<body>元素的最后一个子节点出现。部分的 JavaScript 文件可以等到页面完全加载成功之后再延迟加载。

Google 的 Steve Souders 在前端性能这一领域做了很多开创性的工作。他写的两本书《高性能网站》和《更快速网站》都是非常好的总结性材料，值得深入研读。

个人简介

成富，目前任职于 IBM 中国开发中心，参与 IBM 产品的开发工作。对前端开发和 Dojo 框架有着比较丰富的经验。对新兴的 Web 2.0 技术也有比较浓厚的兴趣。他的个人网站是 http://www.cheng-fu.com。

参考资料

- Ajax应用风格
- Browser Compatibility Tables
- HTML 4.01 Specification
- 富含语义的HTML
- ECMAScript Language Specification第五版
- 面向对象的CSS
- 精通CSS--高级Web标准解决方案
- Pro CSS and HTML Design Patterns
- Memory leak patterns in JavaScript
- Understanding and Solving Internet Explorer Leak Patterns
- XSS Cheat Sheet
- Ajax Security
- Unit testing Web 2.0 applications using the Dojo Objective Harness
- High Performance Web Sites

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

相关内容：

- 虚拟座谈会：RIA和Ajax技术的现状与展望
- AppengineJS：使用JavaScript访问Google App Engine Python SDK
- 虚拟座谈：HTML5 来了，JavaScript框架会如何发展
- 剖析IE浏览器子系统的性能权重

Microsoft Visual Studio

Microsoft

/*CODING完美世界*/

Code，创造了万事万物。几乎任何地方，均有它的存在，正如作为开发人员的你，眼中所见的无限可能。利用Visual Studio 2010的新工具，你设计、开发或是部署的工作方式，都将发生意想不到的变化；你心中的伟大构想，终将以前所未有的方式，付诸实现。

/*CODE，无处不在*/
用Visual Studio 2010打造你的奇迹

立即登录VisualStudio.com
或拨打咨询电话: 800-820-3800 / 021-96081318

请用手机拍摄

拍摄此标签，获取Visual Studio 2010最新消息
如手机未安装二维码应用程序，请至http://gettag.mobi免费获取 *信息及数据资费按相关标准收取

©2010 Microsoft Corporation保留所有权利。Microsoft, Visual Studio及Visual Studio徽标是Microsoft Corporation在美国和/或其他国家（或地区）的注册商标或商标。

推荐文章 | Articles

MonoTouch 中的 MVC 简介

作者 Bryan Costanich 译者 朱永光

在我们的第一篇文章中，用 MonoTouch 在 iPhone 上创建了一个应用程序。我们用到了 outlet 和 action，了解了基本的应用程序结构，并创建了一个简单的用户界面。在这篇文章中，我们将要创建另外一个简单的应用程序，不过这次要学习下如何使用 Views（视图）和 View Controllers（视图控制器）来创建一个具有多个界面的应用程序。特别地，我们将使用 **UINavigationController** 来在应用程序里的两个界面间进行导航。

在开始构建应用程序之前，让我们简单熟悉下 iPhone 应用程序所用的这个重要设计模式。

模型-视图-控制器（MVC）模式

Cocoa Touch 使用了一种修改版本的 MVC 模式来处理 GUI 的显示。MVC 模式（自 1979 年以来）已经出现很长时间了，它皆在分离显示用户界面所需的大量任务，并处理用户交互。

正如名称所蕴含的，MVC 具有三个主要部分，Model（模型）、View（视图）和 Controller（控制器）：

- 模型——模型是特定于领域的数据表现形式。比如说，我们正在创建一个任务列表应用程序。你可能会有一个 Task 对象的集合，书写为 List<Task>。你或许把这些数据保存在数据库、XML 文件，或者甚至从 Web Service 中得到，不过 MVC 不那么关心它们是在何处/如何来持久保存的（乃至它们是什么）。相反，它特别专注于如何显示这些数据，并处理与用户交互的。
- 视图——视图代表了数据如何实际地显示出来。在我们这个假设的任务应用程序中，会在一个网页（以 HTML 的方式）中来显示这些任务，也会在一个 WPF 页面中（以 XAML 的方式）来显示，或者在一个 iPhone 应用程序中显示为 UITableView 。如果用户点击某个任务，要删除之，那么视图通常会触发一个事件，或对 Controller（控制器）进行一个回调。
- 控制器——控制器是模型和视图间的粘合剂。控制器的目的就是获取模型中的数据，告

知视图来显示。控制器还侦听着视图的事件，在用户选中一个任务来删除的时候，控制着任务从模型中删除。

通过分离显示数据、持久化数据和处理用户交互的职责，MVC 模式有助于创建易于理解的代码。而且，它促进了视图和模型的解耦，以便模型能被重用。例如，在你的应用程序中，有两个界面，基于 Web 的和 WPF 的，那么你可以在两者中都使用同样的模型定义代码。

因而，在很多 MVC 框架中不管具体的工作方式如何，基本原理都大致如此的。然而，在 Cocoa（及 Cocoa Touch）中，还是或多或少有所不同，苹果用 MVC 来代表 Views（视图）、View Controller（视图控制器）和 Models（模型）；但是在不同的控件中，它们却不是完全一致的，实现的方式也不太一样。我们将在构建示例应用程序的时候了解更多细节。

在 MonoTouch 中的视图和视图控制器

我之前简短地提到，在 iPhone 应用程序中，你只能显示一个窗口。不过可以包含很多界面。要做到这点，你需要为每个界面都添加一个视图和视图控制器。

视图实际上包含了所有可视化元素，比如标签、按钮等等，而视图控制器处理在视图上的实际用户交互（通过事件），并让你在这些事件被触发的时候运行相应的代码。做一个粗略的比喻的话，这就是和 ASP.NET 或 WPF 有点类似的模型，在这些模型中，你通过 HTML 或 XAML 来定义用户界面，在后置代码中处理事件。

在你导向另外一个页面的时候，就把视图控制器放到视图控制器堆栈中。在这个要构建的应用程序中，我们将使用 Navigation View Controller（导航视图控制器，**UINavigationController**）来处理不同的界面，因为它提供了一种方式可以在界面之间非常容易地导航，通过这种基于层级模式的导航栏，让你的用户能够藉由视图控制器往后和往前进行导航。

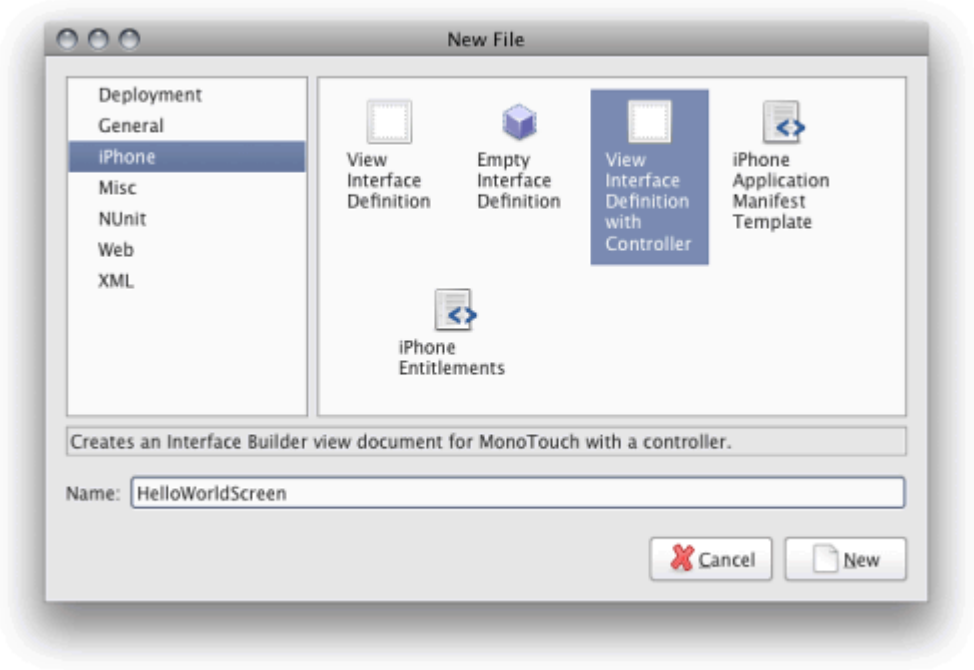
UINavigationController 在很多内置的 iPhone 应用程序都能看到。例如，在查看短信列表的时候，如果你点击其中一个，顶部导航栏将在顶部显示一个左箭头按钮，让你可以回到显示消息列表的视图。

具有多个界面的 Hello World 应用

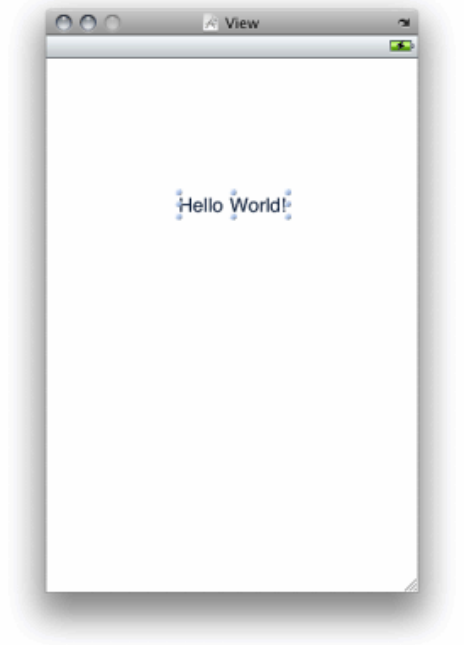
现在，在概念上了解了 MVC 的工作原理后，让我们实际地创建一个应用程序来实践下。

首先，在 MonoDevelop 中新建一个 MonoTouch iPhone 解决方案，命名为 Example_HelloWorld_2（如果你忘记如何操作可以参考一下第一篇文章）。

接着，添加两个视图控制器（以及相关的视图）来服务于我们将要执行导航的应用程序中的界面。要完成这个步骤，在项目上点击右键，选择“**Add : New File**”。

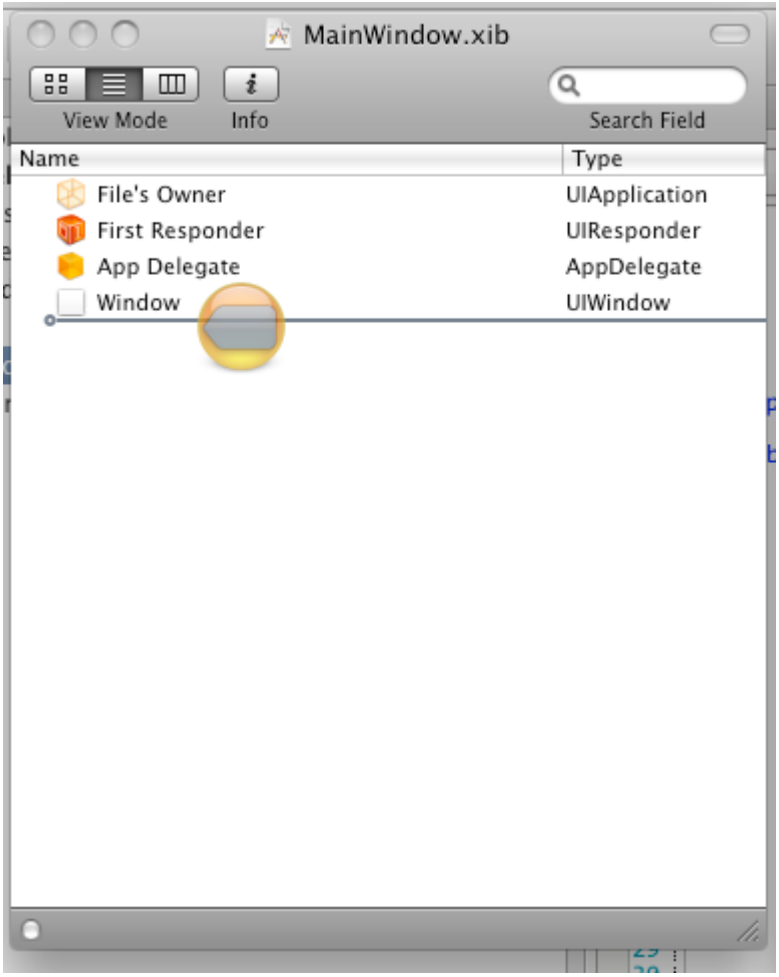


在 Interface Builder 中打开.xib 文件，添加一个标签到 **HelloWorldScreen** 上，修改文本为“Hello World”，另外添加一个文本到 **HelloUniverseScreen** 上，修改文本为“Hello Universe”，如下图所示：



现在，让我们添加一个 Navigation Controller 到 Main Window 上。方式是，在 Interface Builder

里打开 MainWindow.xib，从 Library Window 中拖一个 Navigation Controller 到 Document Window 上：

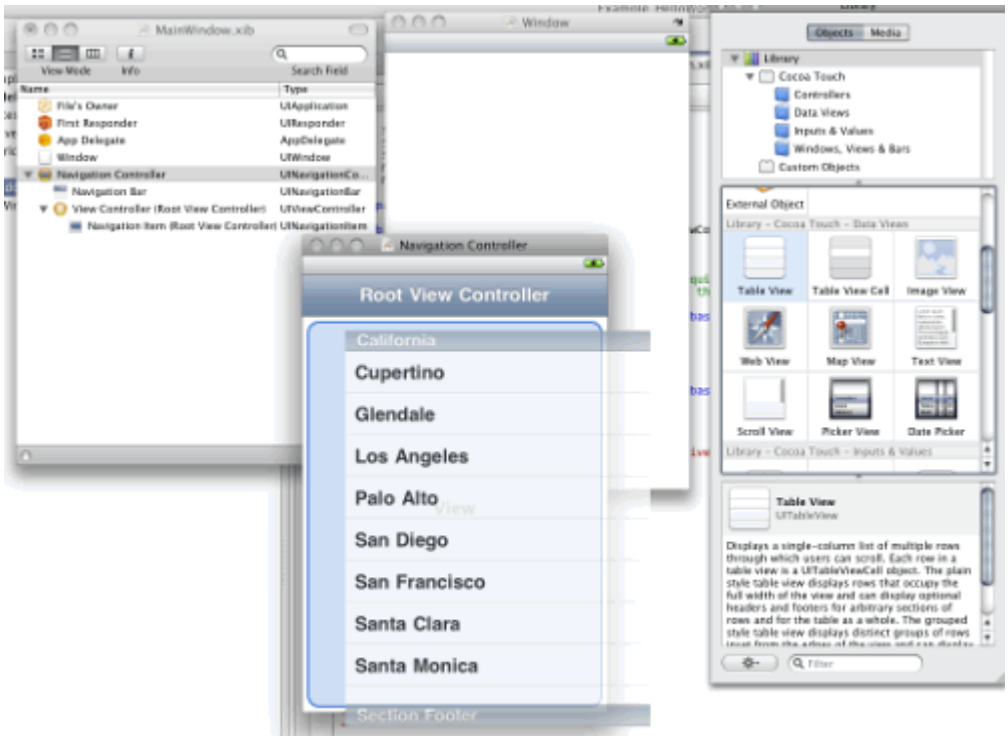


Navigation Controller 具有如下几个部分：

- **Navigation Controller**（导航控制器）——这是控制器的主要部分，处理导航事件，把所有东西糅合在一起。
- **Navigation Bar**（导航栏）——这是显示在顶部的工具条，让用户能够看到它处于导航层级的什么位置，并可以导航回去。
- 视图控制器——这个部分用来控制着视图的显示。
- **Navigation Item**（导航条目）—— 就是显示在导航栏上的部分，实际上就是用于导航的按钮，也显示相应的标题。

接下来，我们添加一个 Table View 到 Navigation Controller 上，以便能创建一个用于各个界面

的链接列表。要完成这个步骤，从 Library 中拖一个 **UITableView** 到 Navigation Controller 里的 View Controller 上：



改变一下导航栏的标题。在 Navigation Controller 上双击顶部栏，键入“Hello World Home!”：

> 我必须使用 Table View 来包含 Navigation Items 吗？
>
> 不用，你可以放任何东西到 View Controller 中。我们将在后面看到，在你导航到一个新界面的时候，你是调用 **NavigationController.PushViewController** 方法，并把要去的界面的 View Controller 传递给它。在用户点击按钮的时候，我们能轻易地实现它。

现在，我们获得了所需的 Navigation Controller 以及相关的 Table View，还需要让两者都可被后置代码访问。需要让 Navigation Controller 在代码中可访问，以便我们能把 View Controllers 传给它；也需要让 Table View 在代码中可访问，以便我们能用要导航到的界面的名称来填充它。

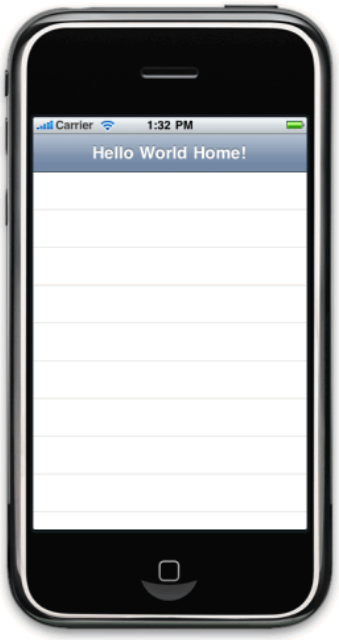
要实现这个步骤，要为它们创建 Outlets，正如我们在第一篇文章所做的那样的。我们把 Navigation Controller 取名为 **mainNavigationController**，把 Table View 取名为 **mainNavTableView**。要确保在 AppDelegate 中创建它们。在你完成后，Connection Inspector 应该看上去如下所示：

接着，需要设置在应用程序启动的时候显示 Navigation Controller。还记得之前在 Main.cs 中注释掉的 **Window.AddSubview** 代码吗？对，这就是我们现在要使用的代码。我们把那行代码改为如下：

```
// If you have defined a view, add it here:
window.AddSubview (this.mainNavigationController.View);
```

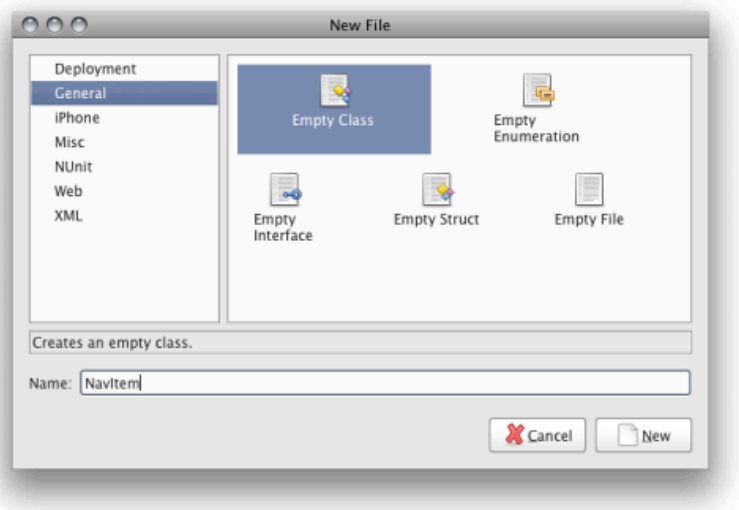
AddSubView 很像 WPF、ASP.NET 等中的 AddControl 语句。通过把它传递给 **mainNavigationController** 对象的 View 属性，我们就可告知窗口去显示这个 Navigation Controller 的界面。

现在让我们来运行一下应用程序，会看到下图所示的样子：



这样 Navigation Controller 就可显示出来了，不过还没有任何链接指向其他界面。为了设置链接，必须用数据来填充 Table View。这就需要创建一个 **UITableViewDataSource** 对象，把它绑定给 Table View 的 DataSource 属性。在传统的.NET 编程中，你可以绑定任何实现了 **IEnumerable** 接口的对象到 DataSource 属性上，并设定一些数据绑定参数（比如需要显示那些字段），这样就实现了巧妙的数据绑定。在 Cocoa 中，工作方式稍微 不同，正如我们看到的，在绑定上的对象需要创建新条目的时候，DataSource 本身都会被调用，DataSource 实际负责它们的创建。

之前，我们实现了 DataSource，现在来创建将要真正使用的条目。创建一个名为 NavItem 的类。在项目上点右键 ,选择“**Add : New File**” ,再选择“**General : Empty Class**” ,命名为“NavItem”，如下图：



现在，把如下代码写到里面：

```
using System;

using MonoTouch.UIKit;

namespace Example_HelloWorld_2 {

    //===========================================================================
    /// <summary>
    ///
    /// </summary>

    public class NavItem  {

        //=============================================================

        #region -= declarations =-

        /// <summary>
        /// The name of the nav item, shows up as the label
        /// </summary>

        public string Name  {

            get { return this._name; }
            set { this._name = value; }
        }

        protected string _name;

        /// <summary>
       ///The UIViewController that the nav item opens. Use this property if you
       ///wanted to early instantiate the controller when the nav table is built
      ///out, otherwise just set the Type property and it will lazy-instantiate
      ///when the nav item is clicked on.
      /// </summary>

        public UIViewController Controller {

            get { return this._controller; }
            set { this._controller = value; }
        }

        protected UIViewController _controller;

        /// <summary>
        /// The Type of the UIViewController. Set this to the type and leave the
        /// Controller property empty to lazy-instantiate the ViewController when
        /// the nav item is clicked.
        /// </summary>

        public Type ControllerType {

            get { return this._controllerType; }
            set { this._controllerType = value; }
        }
```

```
        protected Type _controllerType;

        /// <summary>
        /// a list of the constructor args (if neccesary) for the controller.
        /// use this in conjunction with ControllerType if lazy-creating
        /// controllers.
        /// </summary>

        public object[] ControllerConstructorArgs {

            get { return this._controllerConstructorArgs; }

            set  {

                this._controllerConstructorArgs = value;

                this._controllerConstructorTypes = new
                    Type[this._controllerConstructorArgs.Length];

                for (int i = 0; i < this._controllerConstructorArgs.Length; i++)

                {

                    this._controllerConstructorTypes[i] =
                        this._controllerConstructorArgs[i].GetType ();

                }

            }

        }

        protected object[] _controllerConstructorArgs = new object[] {};

        /// <summary>
        /// The types of constructor args.
        /// </summary>

        public Type[] ControllerConstructorTypes {

            get { return this._controllerConstructorTypes; }

        }

        protected Type[] _controllerConstructorTypes = Type.EmptyTypes;

        #endregion

//============================================================================
//============================================================================

        #region -= constructors =-

        public NavItem () {  }

        public NavItem (string name) : this() {
```

```
            this._name = name;
        }

        public NavItem (string name, UIViewController controller) : this(name)
        {
            this._controller = controller;
        }


        public NavItem (string name, Type controllerType) : this(name) {
            this._controllerType = controllerType;
        }

        public NavItem (string name, Type controllerType, object[]
controllerConstructorArgs) : this(name, controllerType) {
            this.ControllerConstructorArgs = controllerConstructorArgs;
        }
        #endregion
        //=============================================================

    }
}
```

这个类非常简单。我们首先来看一下其中的属性：

- **Name**——打算在 Navigation Table 中显示的界面名称。
- **Controller**——界面对应的实际 **UIViewController** 。
- **ControllerType**——界面对应的 **UIVeiwController** 的类型，这里只是存储着这个控制器的类型，并在需要的时候才来创建它，从而实现 **UIViewController** 的后期实例化目标。
- **ControllerConstructorArgs** ——如果你的 **UIViewController** 具有任何构造参数 ,并且你希望传递它的话，就在这个属性上设置。在我们的例子中，不需要用到这个属性，所以现在可以忽略它，不过我在这里还是列出，因为它对于需要后期创建的类是很有用的。
- **ControllerConstructorTypes** ——这是一个只读属性，读取从 **ControllerConstructorArgs** 设置的类型，其用于实例化控件。

类的剩余部分就是一些基本的构造器。

现在，我们编写好了 NavItem，就可以来为 Navigation Table View 创建一个能实际使用的 DataSource。创建一个名为 **NavTableViewDataSource** 的新类。做法和已经编好的 NavItem 的类似。

现在，把下面代码写入：

```
using System;

using System.Collections.Generic;

using MonoTouch.UIKit;

using MonoTouch.Foundation;

namespace Example_HelloWorld_2

{

    //=======================================================================
    //
    // The data source for our Navigation TableView
    //

    public class NavTableViewDataSource : UITableViewDataSource {

        /// <summary>
        /// The collection of Navigation Items that we bind to our Navigation
        /// Table
        /// </summary>

        public List<NavItem> NavItems {

            get { return this._navItems; }
            set { this._navItems = value; }
        }

        protected List<NavItem> _navItems;

        /// <summary>
        /// Constructor
        /// </summary>

        public NavTableViewDataSource (List<NavItem> navItems) {

            this._navItems = navItems;

        }

        /// <summary>

        /// Called by the TableView to determine how man cells to create for that
       /// particular section.
        /// </summary>

        public override int RowsInSection (UITableView tableView, int section)
```

```
        {
            return this._navItems.Count;

        }

        /// <summary>
        /// Called by the TableView to actually build each cell.
        /// </summary>

        public override UITableViewCell GetCell (UITableView tableView,
NSIndexPath indexPath)

        {

            //---- declare vars

            string cellIdentifier = "SimpleCellTemplate";

            //---- try to grab a cell object from the internal queue

            var cell = tableView.DequeueReusableCell (cellIdentifier);

            //---- if there wasn't any available, just create a new one

            if (cell == null) {

                cell = new UITableViewCell (
                    UITableViewCellStyle.Default, cellIdentifier);

            }
            //---- set the cell properties

            cell.TextLabel.Text = this._navItems[indexPath.Row].Name;
            cell.Accessory = UITableViewCellAccessory.DisclosureIndicator;


            //---- return the cell
            return cell;

        }
    }
//=======================================================================

}
```

快速浏览一下代码。第一部分是我们的 List<NavItem>集合。就是一个 NavItem 对象的集合。接着会看到一个基本的构造器，使用传入的 NavItems 参数来初始化 **NavTableViewDataSource** 。

接着，我们重写了 **RowsInSection** 方法。Table Views 能具有多个分段，在每个分段上都可以放置条目。**RowsInSection** 基于 section 参数传递进来的分段索引来返回条目的数量。在我们的例子中，只具有一个分段，那么我们就返回 NavItem 集合的 Count 属性。

最后一个方法是 GetCell，这里就是数据绑定实际发生的地方。这个方法被 UITableView 在构建每行数据的时候所调用。你可以利用这个方法来构建出 Table 中的每行数据，以显示出你期望的内容。

此处，我们所做的第一件事情就是通过 **DequeueReusableCell** 方法从 **TableView** 中得到 **UITableViewCell** 对象。**TableView** 保持着一个 **UITableViewCell** 对象的内部对象池，其基于 **CellIdentifiers** 来进行查找。它让你可以为 **UITableViewCell** 创建自定义模板(只用创建一次)，并重用这个模板，而不是 GetCell 每次被调用的时候都重复创建模板，这样就提高了性能。我们第一次调用 **DequeueReusableCell**，它不会返回任何东西，那么就要创建一个新的 **UITableViewCell**。之后的每次调用，**UITableViewCell** 已经存在，就只需直接重用它就行。

我们使用 Default 的单元格样式（ cell style ），其只为我们提供了很少的自定义选项，所以接下来的事情就是把 **TextLabel.Text** 属性设置为 **NavItem** 的 **Name** 属性值。接着，我们设置 **Accessory** 属性来使用 **DisclosureIndicator**，其只是一个显示在 Navigation Item 右边的简单箭头。

现在 ,我们已经得到了创建好的 **UITableViewDataSource** ,是时候使用它了。在 MonoDevelop 中打开 Main.cs，把如下的代码行添加到 **AppDelegate** 类中：

```
protected List<NavItem> _navItems = new List<NavItem> ();
```

它将保存我们的 NavItem 对象。

接下来，添加如下代码到 **FinishedLaunching** 方法中，在 **Window.MakeKeyAndVisible()**之后：

```
//---- create our list of items in the nav

this._navItems.Add (new NavItem ("Hello World", typeof(HelloWorldScreen)));

this._navItems.Add (new NavItem ("Hello Universe",
   typeof(HelloUniverseScreen)));

//---- configure our datasource

this.mainNavTableView.DataSource =
   new NavTableViewDataSource (this._navItems);
```

在这里我们做的所有这些事情，就是创建两个 NavItem 对象，并把它们添加到_navItems 集合中。接着 ,我们创建一个 **NavTableViewDataSource** 对象 ,把它绑定到 Navigation Table View。

把之前代码加入后，我们的 AppDelegate 类看上去如下所示：

```
// The name AppDelegate is referenced in the MainWindow.xib file.

public partial class AppDelegate : UIApplicationDelegate {
```

```
    protected List<NavItem> _navItems = new List<NavItem> ();

    // This method is invoked when the application has loaded its UI and its ready
    // to run

    public override bool FinishedLaunching (UIApplication app, NSDictionary
options)  {

    // If you have defined a view, add it here:

        window.AddSubview (this.mainNavigationController.View);

        window.MakeKeyAndVisible ();

        //---- create our list of items in the nav

        this._navItems.Add (new NavItem ("Hello World",
            typeof(HelloWorldScreen)));

        this._navItems.Add (new NavItem ("Hello Universe",
            typeof(HelloUniverseScreen)));

        //---- configure our datasource

        this.mainNavTableView.DataSource =
            new NavTableViewDataSource (this._navItems);

        return true;

    }

    // This method is required in iPhoneOS 3.0

    public override void OnActivated (UIApplication application){  }

}
```

如果你现在运行应用程序，你将看到如下所示的样子：

我们现在拥有了构建好的导航条目，不过在点击它们的时候不会发生任何事情。在你点击一个条目的时候，**UITableView** 会引发一个事件，不过需要我们传递给它一个特别的类，叫作 **UITableViewDelegate** ，它是检测这些事件实际处理类。要实现这个步骤，就在项目中创建一个新类，命名为“NavTableDelegate”，并写入如下代码：

```
using MonoTouch.Foundation;
using MonoTouch.UIKit;
using System;
using System.Collections.Generic;
using System.Reflection;
namespace Example_HelloWorld_2 {
//==========================================================================
// This class receives notifications that happen on the UITableView
public class NavTableDelegate : UITableViewDelegate {
    //---- declare vars
    UINavigationController _navigationController;
    List<NavItem> _navItems;
    //==========================================================================
```

```
   /// <summary>
   /// Constructor
   /// </summary>

   public NavTableDelegate (UINavigationController navigationController,
List<NavItem> navItems) {

       this._navigationController = navigationController;

       this._navItems = navItems;

   }

   //===========================================================================

   //===========================================================================

   /// <summary>
   /// Is called when a row is selected
   /// </summary>

   public override void RowSelected (UITableView tableView, NSIndexPath
indexPath) {
       //---- get a reference to the nav item

       NavItem navItem = this._navItems[indexPath.Row];

       //---- if the nav item has a proper controller, push it on to the
       // NavigationController

       // NOTE: we could also raise an event here, to loosely couple this, but
      // isn't neccessary,

       // because we'll only ever use this this way

       if (navItem.Controller != null)  {

          this._navigationController.PushViewController (navItem.Controller,
true);

           //---- show the nav bar (we don't show it on the home page)

           this._navigationController.NavigationBarHidden = false;

       } else {

           if (navItem.ControllerType != null)  {

               //----

               ConstructorInfo ctor = null;

               //---- if the nav item has constructor aguments

               if (navItem.ControllerConstructorArgs.Length > 0) {
```

```
            //---- look for the constructor

            ctor = navItem.ControllerType.GetConstructor (
               navItem.ControllerConstructorTypes);

         } else {

            //---- search for the default constructor

            ctor = navItem.ControllerType.GetConstructor (
               System.Type.EmptyTypes);

         }

         //---- if we found the constructor

         if (ctor != null) {

            //----

            UIViewController instance = null;

            if (navItem.ControllerConstructorArgs.Length > 0) {

               //---- instance the view controller

               instance = ctor.Invoke (
                 navItem.ControllerConstructorArgs)
                 as UIViewController;

            } else {

               //---- instance the view controller
               instance = ctor.Invoke (null) as UIViewController;

            }
            if (instance != null){

               //---- save the object

               navItem.Controller = instance;



               //---- push the view controller onto the stack

               this._navigationController.PushViewController (
                  navItem.Controller, true);

            } else {

               Console.WriteLine (
                  "instance of view controller not created");

            }

         } else {
```

```
                Console.WriteLine ("constructor not found");

            }
        }
    }
    //=======================================================================
}
//=======================================================================
}
```

这个类的第一部分是针对 **UINavigationController** 和 **NavItem** 对象的集合的一对声明，下面的构造器会需要用到它们。在下面的方法——**RowSelected** 中我们将看到，为什么需要它。

RowSelected 在用户点击某行的时候 **UITableView** 会调用它，并会返回给我们一个 **UITableView** 的引用，以及用户点击条目的 **NSIndexPath** 。首先，我们要根据 **NSIndexPath** 来找到相应的 **NavItem** 。接着，我们把 **NavItem** 的 **UIViewController** 传递给 **NavigationController**。如果 **Controller** 是空的，那么我们就会基于它的类型进行实例化。

最后的两个操作，就是我们为什么需要 **NavItem** 集合和 **NavigationController** 引用的原因。

现在，我们有了 **UITableViewDelegate**，就可以来组合在一起。返回到 Main.cs 文件中，在 **AppDelegate** 类中添加如下代码行到设置 **DataSource** 属性的后面：

```
this.mainNavTableView.Delegate = new NavTableDelegate (
    this.mainNavigationController, this._navItems);
```

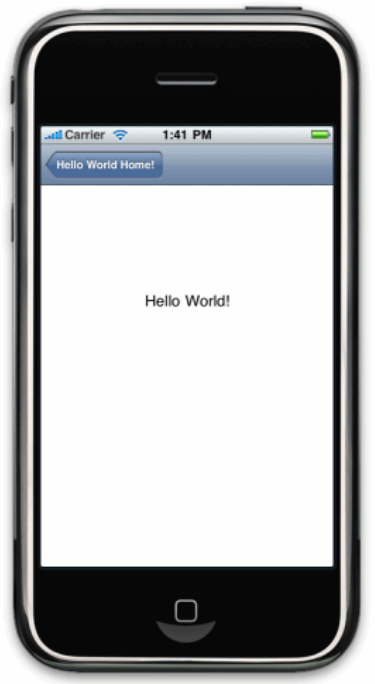
这样就创建了一个新的 **NavTableDelegate** 类，以及指向 Navigation Controller 和 **NavItems** 集合的引用，且会告知 **mainNavTable** 使用它来处理事件。

Main.cs 文件中的 **AppDelegate** 类将会如下面代码所示：

```
// The name AppDelegate is referenced in the MainWindow.xib file.

public partial class AppDelegate : UIApplicationDelegate {
    protected List<NavItem> _navItems = new List<NavItem> ();

    // This method is invoked when the application has loaded its UI and its ready
    // to run

    public override bool FinishedLaunching (UIApplication app, NSDictionary
options){

        // If you have defined a view, add it here:
        window.AddSubview (this.mainNavigationController.View);

        window.MakeKeyAndVisible ();

        //---- create our list of items in the nav

        this._navItems.Add (new NavItem (
```

```
            "Hello World", typeof(HelloWorldScreen)));

        this._navItems.Add (new NavItem (
            "Hello Universe", typeof(HelloUniverseScreen)));

        //---- configure our datasource

        this.mainNavTableView.DataSource = new NavTableViewDataSource (
            this._navItems);

        this.mainNavTableView.Delegate = new NavTableDelegate (
            this.mainNavigationController, this._navItems);

        return true;
    }

    // This method is required in iPhoneOS 3.0

    public override void OnActivated (UIApplication application){  }
}
```

现在，我们运行一下应用程序，看一下会发生什么，点击“Hello World”你将看到如下的效果：



注意，我们会自动地在顶部得到一个“Hello World Home”按钮，这样就能让我们返回到主界面上。点击“Hello Universe”将得到如下界面：

恭喜你！你现在应该已经对 MonoTouch iPhone 应用程序中多个界面是如何工作的有了一个基本的概念，以及对 **UINavigationController** 的工作原理有了一定了解。

示例代码

原文链接：http://www.infoq.com/cn/articles/monotouch-mvc

相关内容：

- ASP.NET到达新的里程碑版本：ASP.NET MVC 3 Preview 1
- ASP.NET MVC的四种视图引擎
- Spring MVC与JAX-RS比较与分析

推荐文章 | Articles

测试工程师的学习之旅

作者 Lisa Crispin 译者 崔康

软件行业发展迅猛。越来越多的团队开始重视测试，他们利用测试驱动开发。全新的或者改进的自动化测试和驱动框架层出不穷。团队在采用更多自动化回归测试之后，需要测试人员具有精湛的探索性测试技能。但是大部分人在校园里中学不到这些必要的技能，那么测试人员是如何炼成的呢？

同时，我发现一些人在努力寻找让自己满意的测试工作。测试人员经常问我如何融入敏捷开发，或者哪些技能可以帮助他们找到满意的工作。如果没有编程经验，他们会担心技术上无法立足于敏捷团队。我认为虽然技术很重要，但是态度决定一切。如果你乐于学习，并且努力帮助团队交付优秀的产品，那么作为测试人员，你前途一片光明。我的建议是抓住一切学习机会，主动获取新技能。

我发现许多人从例子中学到的东西比较多，所以这里分享本人的几个故事，讲述了我的学习动力是如何促进事业发展的，希望能够为读者自己的职业进步提供一些启发。

开发人员、测试人员还是领域专家？

测试人员的背景非常广泛。在过去十年间，随着越来越多的开发人员对测试产生兴趣，我见到许多开发人员更乐于把自己视为测试人员。还有许多测试人员来自于业务领域，他们的领域专长对于开发非常有价值。技术作者，必须弄清楚应用程序的行为才能正确表述，所以经常让自己变成测试人员。许多人都是碰巧担当了这个角色，我也是！

来说说我自己的故事吧。我的职业生涯起初是一名开发人员，而且我喜欢编程。测试自动化（本质上属于开发人员的任务）是我最喜欢的工作之一。我热爱测试。我乐于了解业务并想办法促其成功。拥有技术背景让我既适应开发团队又适应业务团队。下面的故事讲述了我的学习旅程：从早期的开发岁月到参与敏捷团队。

对测试的早期认识

和许多人一样，我是偶然进入了软件开发领域。我最初在 Texas 大学的 Austin 数据处理部门获得了“开发实习生”的职位。

我的培训老师其实比我早几周入职，也是刚刚接受培训，他们刚学会了编程，然后又教会了我。很快，我就了解了 Easytrieve、Cobo 和 4GL 还有层次型数据库的基础知识。我们以相同的方式编写代码，所以彼此的程序易于操作。现在回想起来，集体性的代码所有权非常有用。

在这次培训的数月之后，我很高兴的接受了教育协调员的工作，不仅仅监督开发人员培训，还负责培训最终用户。我们通过课程教育老师们如何执行简单的查询和报告，这节省了开发人员大量的工作。我从这一年的经历（期间我还在日常开发工作）中学到了很多：如何教授他人。

我惊讶的发现从客户和其他开发人员身上受益良多。我们（开发人员和分析人员）与客户坐在一起，讨论他们的需求，并现场画出原型。我们一一展示直到他们确认需求。我曾经加入一个团队来规划图书馆的在线编目系统，与图书管理人员坐在一起了解卡片编目系统是如何运作的。学习不同的领域是我工作中最有趣的部分。我们对测试一无所知，但是与客户的合作帮助我们在发布产品之前提高了软件的质量。

在最初的开发/分析工作中，我学会了如何领导他人。我的老板曾经告诉我做领导意味着确保其他人知道我的团队所做的贡献。我学会了以身作则。在以后的工作中我一直谨记在心，想办法让老板和其他业务上的人员知道我的团队和我自己带来的价值。

在转变中学习

几年后，我在一家大型软件公司担任技术支持，那时对测试和质量保证的概念不太了解。同事和我出于自我防范的意识做了大量测试工作：在客户发现缺陷之前最好由我们自己来找到这些问题。某天，老板问：“谁想做 DB2 培训？”没人了解 DB2，但是我主动请缨。很快，我成为了团队中 SQL 和 DB2 专家。

公司发现在客户之前找到缺陷好处多多，所以决定创建第一个测试团队。我再次自愿参与。因为我了解 SQL，所以我测试了使用 Oracle 和 Sybase 数据库的项目，这些都比我们自己的数据库产品在市场上更受欢迎。

在新的工作中，我开始学习测试的方方面面。我参加了一次测试研讨会了解了更多知识。我们开始尝试测试自动化。我们的软件适应于所有操作系统，因此我有幸学习 VAX/VMS、Wang、

OS2、AS400 和八种不同的 UNIX 系统。虽然这些经历写到简历上不是那么好看，但是在所有平台上维护测试环境是宝贵的经验。

我们的团队同时负责打包发布。我理解了发布说明和准确文档的重要性，以及如何管理 alpha 和 beta 测试。起初这些任务让人觉得很困难，即使现在我也觉得自动化测试是这样。但是我很幸运的受到了来自外部课程、自学教材和同事等各方面的培训和支持。我试着克服困难，不断想办法掌握新技能。

随着在测试、自动化、数据库和操作系统等各方面的广泛经验，我拥有了不凡的技能。这本不是我的目标，起初我只是想学习新知识！不论是技术能力还是有关业务的什么东西，我喜欢在新领域中探索，很值得去做。当公司遇到财政危机时，我找到了一份不错的新工作。

个人关系创造机会

我的新工作很有趣，而且有机会学到新技能。例如，我成为了团队的 Powerbuilder 专家。我能够花费几个月时间来学习一种测试工具并搭建自动化 GUI 测试集。最重要的是，一些过去的同事也加入了这家新公司，让我领悟到——这世界真是小啊！

几年之后，在互联网热潮中，我加入了一家 web 创业公司。我对测试 web 应用一无所知，但是因为我曾经使用过多年各种测试自动化工具，所以我在因特网上想找到适合 web 应用的工具。

当我查看工具列表网站时，“OCLC”几个字母吸引了我的眼球。当我在参与在线图书馆编目项目时深入了解了 OCLC，因为 OCLC 一直被用于编目书籍和向图书馆提供服务。奇怪的是，它们在出售一款名为 WebArt 的测试工具，我决定购买。它的开发者 Tip House 过来培训我们如何测试 web 应用和自动化测试。

和许多测试人员一样，我总是在想如何更好地及时交付高质量的软件。互联网世界比数据库产品变化快得多，我对缓慢、瀑布型的过程感到沮丧。尝试一种不同方式的机会很快就出现了。当我们的创业小公司被一家大公司收购的时候，一些同事离职选择自主创业，他们给我一本名为《Extreme Programming Explained》的书，说：“我们准备尝试极限编程。”当我读了这本书，我觉得自己必须尝试一下，请求他们带上我。

第一次加入 XP 团队之后，我开始学习在 XP 模式下测试人员应该如何工作，并分享到在线敏捷社区中（虽然那时我们还不称之为“敏捷”）。我惊讶的发现 XP 专家和其他敏捷实践者非常受欢迎。当 Bob Martin 大叔过来培训我们时，他建议我给 Ward Cunningham 打电话请教测试中的问题，并提供了他的手机号。Ward 与我讨论了一个小时！如果我听说类似 Ron

Jeffries 或者 Kent Beck 来访或者出席一个我参加的会议，我会想办法与他们见面，而他们总是很慷慨地花时间会解答问题。Brian Marick 帮助我创建了一个敏捷测试邮件组，使我受益良多。

贡献社区获得机会

当我的团队还有那些我通过会议、用户组和邮件列表结识的朋友都认识到敏捷测试技术的好处时，我决定不应该让其他的测试人员和团队继续重复昨天的故事。在 XP 社区的鼓励下，Tip House 和我合著了一本书《Testing Extreme Programming》。许多人帮忙审阅了草稿并反馈意见，包括 Janet Gregory。Janet 和我开始组织研讨会和教程。

极限编程的核心是人，而事业成功的要素也是。我运用了个人关系，并最终成为一名演讲者、教练和书籍作者。我不仅成为了一名更出色的测试人员，我还学会了沟通的最佳方式。我经常参加会议，向他人学习，并在研讨会和培训班中阐述自己的观点。这一切都因为我想学习，并花时间与朋友发展良好的工作关系。

我也体验了回报的乐趣和价值。我的第一个 XP 团队与其他组织发起了本地的 XP 用户组。我在第一次会议上做了演讲！过去十年间，我通过这个用户组见到了许多优秀的朋友并受益良多，而它只是占用了我的一些时间而已。我努力回报以前获得的所有帮助。我参加了本地用户组，志愿帮忙组织会议，维护着一个测试邮件列表，和其他公司组织一些短期的研讨会，和对测试和敏捷开发存在疑问的团队进行网络和电话会议。我发现帮助别人越多，自学的也就越多。这感觉太好了—— 回报他人就是帮助自己。

学无止境：开阔眼界

我已经从事软件测试这个职业许多年了，但是不觉得厌倦。我每天都在学习新东西：要么是技术，要么是业务运作上的发现。在同事或者用户组、会议甚至 Twitter 上的同行的协助下，我尝试了新的开源工具，并且学习了新的脚本语言。这可能很困难，但是值得付出。

例如，我努力学习 Ruby，因为我从没掌握过一种面向对象语言。我阅读相关书籍并从同事中获得帮助，通过 Ruby 编写的脚本让我有更多时间关注更有趣的测试。我参加了一些组织以改进测试工具，如关注测试自动化的 Austin Workshop 和敏捷联盟功能测试工具委员会。我不仅了解了更多工具，而且见到了许多能提供帮助的同行。

为什么如此重要？

我希望其他测试人员在看到本文时能够感受到我对自己工作的热爱（虽然有时会沮丧：希望自己掌握更多的技能！）来自早期技术支持团队的朋友惊讶于为何我找到一份新工作这么容易，而他们依然挣扎于讨厌的工作中。我不比他们任何人更聪明：我花时间学习并抓住新机会！对学习的时间投入和参与技术社区活动对我的职业发展产生了回报。

这就是我希望读者从我的学习之旅中得到的启示：对自己的职业发展负责。不要局限于技术或者测试技能。了解公司业务领域使你能够帮助他们做出正确选择。现在，请走出你的封闭角落，想一想如何帮助团队和公司。加入一个在线的测试俱乐部，或者志愿帮助本地的测试用户组。买一本新书或者阅读在线的教程。今天就启程可以让你的学习之旅走得更远一些。你会更加喜欢自己的工作，你会获得更多机会，你会让我们所有人为你感到骄傲。

下面是一些测试人员学习的资源：

- **Collaboration Explained: Facilitation Skills for Software Project Leaders**, Jean Tabaka, Addison-Wesley 2006. These skills will serve you even if you aren't a 'project leader'.
- **Everyday Scripting with Ruby: For Teams, Testers and You**, Brian Marick, Pragmatic Bookshelf, 2007
- "Writing Maintainable Automated Tests", Dale Emery, 2009,
- Continuous Integration and Testing Conference
- "Google's '20 percent time' in action", Alex K.
- Exploratory testing articles and blog posts, Jonathan Kohl,
- Agile user groups
- Software Testing Club
- Weekend Testers
- Articles about agile testing

关于作者

Lisa Crispin 与 Janet Gregory 合著了《Agile Testing: A Practical Guide for Testers and Agile Teams》，参与编写了《Beautiful Testing》，她在过去十年间在敏捷团队中担任测试人员，喜欢通过写作、演讲和参与敏捷测试社区来分享其经验。如果想了解更多有关 Lisa 的信息，请访问 www.lisacrispin.com。

原文链接：http://www.infoq.com/cn/articles/testers-learning-journey

相关内容：Yahoo 推出开源 YUI 跨浏览器测试工具 Yeti、让测试也敏捷起来

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

Yahoo 推出开源 YUI 跨浏览器测试工具 Yeti

作者 崔康

YUI 是一款企业级的 JavaScript 开发工具包，被广大 Web 前端工程师所熟知和采纳。不论是采用哪种框架 构建的 Web 应用在不同浏览器上的测试 通常是一件令人头痛的事情。最近，Yahoo 开发团队推出了开源 YUI 跨浏览器测试工具 Yeti，相信 Web 开发和测试人员会从中受益。

原文链接：http://www.infoq.com/cn/news/2010/08/yui-yeti

Spring Roo 与 GWT 同时发布新的里程碑版本

作者 张龙

近日，Spring Roo 1.1 与 Google Web Toolkit 2.1 同时发布了 M3 版本，这表明自从 Google I/O 以来，这两种技术的一种同步状况。VMware 与 Google 曾在 Google I/O 上宣布未来关于 Spring 框架与 GWT 的集成计划。这两种技术的紧密集成，再加上其他项目（如 AspectJ 与 STS）的不断参与使得在实际开发 中，同时使用 Spring 工具与 GWT 的项目呈现出不断增长的态势。

原文链接：http://www.infoq.com/cn/news/2010/08/spring-roo-and-gwt-release-m3

开源 HTML 解析工具包 jsoup 1.3.1 发布

作者 崔康

jsoup 是一款开源的 HTML 解析工具包，采用 Java 语言编写，通过精巧的 API 充分利用 DOM、CSS 和类 jquery 的方法抽取和操作数据。最近，jsoup 1.3.1 正式发布，对上一版（1.2.3）做了重要更新，包括完成自主实现（无外部依赖）、改进 Web 连接方法等。

原文链接：http://www.infoq.com/cn/news/2010/08/jsoup1.3.1

开源权限管理中间件 Ralasafe 发布 1.0 rc2 版

作者 崔康

Ralasafe 是一款国产开源数据 级权限管理中间件，使用 MIT 协议，最近发布了 1.0 rc2 版。项目采用 Java 语言编写，解开权限与业务的耦合，将权限策略集中管理，并使用图形化的管理模式。InfoQ 中文站就 Ralasafe 的应用场景、技术架构和未来规划等问题对项目负责人汪金保进行了专访。

原文链接：http://www.infoq.com/cn/news/2010/08/ralasafe1.0

Tasktop Agile Planner for Eclipse 发布了

作者 Dave West 译者 张龙

近日，Tasktop 发布了一个敏捷 规划工具，该工具使用了 MyLyn 连接器来提供规划支持 ,可以跨越多种项目管理工具 ,如 Mingle、Team Concert、Rally、Scrumworks、JIRA 及 VersionOne。其 Eclipse 集成可以将规划项目与源代码文件自动链接起来，这样开发者就可以直接从 Eclipse 中创建并管理规划了。

原文链接：http://www.infoq.com/cn/news/2010/08/tasktop-agile-planning

Web Services 框架 XINS 2.3 发布

作者 张龙

XINS 是个开源的 Web Services 框架，支持 REST、SOAP、XML-RPC、JSON 以及 JSON-RPC 等。它基于契约优先的开发模式，因此可以根据 API 规范生成代码与文档。近日，XINS 2.3 发布了。

原文链接：http://www.infoq.com/cn/news/2010/08/XINS2.3-release

推荐编辑 | SOA 社区编辑 黄璜

大家好，我是黄璜。不知不觉加入 InfoQ 的编辑团队也有两年半了。清楚的记得当时我还是参加工作不久的学生。通过浏览 InfoQ 上的文章，觉得为读者带来了不少价值，觉得自己也能为此出一份力。就联系了社区主编胡键。一开始，只是凭着对 SOA 的一股热情。后来才发现，翻译这件事绝不是一件容易的事情。在此要感谢胡主编悉心的指导，使我成长为一名合格的编辑。

InfoQ 社区是一个激情、活跃、积极贡献的社区。编辑们都是天南海北，远程的合作。虽然我没有到北京参加聚会以及参加 Qcon 大会，但我仍然能够感觉到自己作为其中的一分子，一件简单的印着 InfoQ logo 的 T-Shirt 既是一份荣誉也是一份激励，觉得自己有责任为读者贡献质量更高的内容。

随着工作的深入，慢慢也发现了自己的兴趣和发展的方向所在，InfoQ 上即时的信息和趋势就像是业界的一个晴雨表，而深度的技术文章和访谈内容又可以满足我 们这些技术爱好者想要深入挖掘和一探究竟的心理。同时也很高兴的看到 InfoQ 中文站有了越来越多的本土原创内容和来自支付宝淘宝等公司的深度贡献。

打开 Gdoc，回顾自己所翻译的新闻和文章，就像梳理自己这两年半以来的成长历程，同时，也更能审视自己的不足。这两年半的技术翻译工作我想有三个方面的收获，第一，能把更新更好的东西介绍给读者，第二，在翻译中经常会遇到各种各样的问题，提升了自己技术理解的深度和广度，第三，社区和团队，使我感受到了人们联合在一起的力量，也使我更相信以人为本以团队为根基。

就在今年，我开始的一份新的工作和一个新的征程。我想，这与我在 InfoQ 社区两年半来所受到的影响是密不可分的。同时，我也只有通过更好更多的翻译工作，来回馈读者。就让我与你一道，时刻关注和参与开发领域的变化与创新！

桃儿七

桃儿七，稀有种，“桃儿七”属于“太白七药”之一。残存于东亚，呈零星分布。由于根状茎与果实入药，而被任意采挖，天然繁殖能力较弱，随着植被的破坏而导致其生境的改变，植株日益稀少，分布区日渐缩减。

生态习性：多年生草本，高 40 一 80 厘米；根状茎粗壮，横走，通常多少结节状；不定根多数，长达 30 厘米以上，直径 2 一 3 毫米，红褐色或淡褐色；茎直立，基部具抱茎的鳞片，上部有 2（一 3）叶。通常生于海拔较高的平坦山谷及透光度好的林下、林缘或草灌丛中。高山草丛中或疏林下及林缘。适于寒冷而湿润，夏季低温多雨，冬春干冷的气候。所在地为高山草地乱石缝隙腐殖质丰富的山地灰化土、暗灰钙土、灰褐土及山地棕壤。

保护价值：桃儿七的根茎与果实均有较高的药用价值。同时也是东亚和北美植物区系中的一个洲际间断分布的物种，对研究东亚、北美植物区系有一定的科学价值。

保护措施：在太白山已建立自然保护区，应列为保护对象，严禁采挖。其他地区也应加强保护、控制采挖，积极繁殖，扩大种植。

开花动态：花单一,先叶开放；花着生于叶柄的交叉处或稍上方；花梗长 2～5cm;花萼早落；花瓣 6，白色至蔷薇红色，倒卵状长卵形，长 3～4.5cm，先端圆，基部渐狭；雄蕊 6,长约 1cm，花药长圆形，花似基部加宽；雌蕊单一；子房近圆形，柱头盾状，几无花柱。浆果卵圆形，熟时红色。种子多数。花期 5～6 月，果期 7～9 月。


1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 8 月刊

每月 8 日出版

本期主编：马国耀

总编辑：霍泰稳

编辑：李明 胡键 宋玮 郑柯 朱永光 池建强

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：马国耀，**InfoQ** 中文站 **SOA** 社区编辑

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，云计算，企业服务总线（ESB），J2EE，Java 编程，开源项目等。目前在 IBM 工作，主要工作内容是将 IBM 的产品和技术带给合作伙伴，让更多的人了解 IBM 的技术和产品，曾支持过电信，移动，金融，政府等行业的大型 SOA 与 ESB 实施项目。业余时间大多在闲敲棋子落花灯中度过；目前正在翻译《SOA 与云计算在企业中的融合》一书。您可以通过 maguoyao [at] gmail.com 联系到他。

《架构师》月刊由 InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ neue

Enterprise Software Development Community

商务合作：sales@cn.infoq.com

读者反馈/投稿：editors@cn.infoq.com