时刻关注企业软件开发领域的变化与创新

架构师

11月 ARCHITECT

Paul Hudak
谈Haskell

SOA宣言发布

如何进行平台
型网站架构设计

HTML5来了
JS框架如何发展

每月技术综述

InfoQ中文站
www.infoq.com/cn

每月8号出版

篇首语 | Editor's letter

修炼技术全局观

很高兴能在这期的架构师中与大家见面。这次，我想和大家谈谈“技术全局观”这样一个概念。下面，我将简单探讨一下何谓技术全局观、为什么要具备技术全局观及如何修炼技术全局观。

所谓技术全局观，指在某个技术领域里，对其中的技术有一个全面了解，对这些技术具体情况做到心中有素；但是，全局观并不要求对所有技术中的细枝末节都烂熟于心。重要的是，学习技术不仅要知其然，还要知其所以然；理解各种技术相互之间的关系，它们搭配协同的方式；了解各种技术演变的历史，发展的趋势；明白各种技术的适用情况，知道它们的优缺点；等等。

对于架构师这样的角色来说，具备技术全局观是必需也是必然的要求。由于架构师（特指应用或解决方案架构师）的职责就是要技术和业务结合起来，把正确的技术用来解决业务问题。假设，对涉及领域的技术没有一个全局的理解，试问如何把技术正确应用起来呢？而且，也只有具备了技术全局观，才能对不断推陈出新的新技术做到快速学习，快速融入到技术体系中，并用来事半功倍地解决实际问题。

冰冻三尺非一日之寒，滴水石穿非一日之功。要具备技术全局观也需要一个长期的学习积累过程，需要不断的实践和总结。不过，要修炼技术全局观，还有一点不能忽视——主动意识。要主动全面深入地学习技术基础原理，要主动思考各个技术点间的关系，要主动弄清楚各个技术存在、设计和演化的缘由，要主动分析各种技术的不足并尝试思考解决方法，要主动开放地接受本领域甚至跨领域的新技术并加于适当应用。

本期主编： 朱永光

目　录

[篇首语]

修炼技术全局观 I

[人物专访]

PAUL HUDAK谈HASKELL 1

[热点新闻]

INTELLIJ IDEA开源！ 9

SUN将在JAVA 7 中摒弃SWING APPLICATION FRAMEWORK 11

使用GESTALT直接在HTML中嵌入PYTHON、RUBY和XAML 13

BRAD ABRAMS终于完成了.NET RIA SERVICES的开发系列文章 14

RUBY ENTERPRISE EDITION新版本切换至RUBY 1.8.7 15

SOA宣言发布 17

支持云应用程序服务的PHP API 19

ZDNET发布中国最受欢迎 50 大技术博客 21

如何进行平台型网站架构设计？ 24

[推荐文章]

虚拟座谈：HTML5 来了，JAVASCRIPT框架会如何发展 26

基于AZURE云计算平台的网格计算，第 2 部分：开发网格应用 33

企业级开发，PHP准备好了吗？ 45

HTTPS连接最初的若干毫秒 49

[每月新品]

OSGI 4.2 发布了 67

开源SERVLET容器JETTY 7.0 发布 67

XMEMCACHED——一个新的开源JAVA MEMCACHED客户端 67

连接JAVA和.NET的RESTFUL BRIDGE发布了 68

VS 2010 BETA 2 发布，预计 2010 年 3 月将发布RTM 68

微软发布MICROSOFT AJAX脚本库和脚本缩小器 68

[每月技术综述]

十月技术综述 69

[封面植物]

海菜花 72

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ中文站
www.infoq.com/cn

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

人物专访 | Interview

Paul Hudak谈Haskell

这篇访谈首先讨论了何时引入如 monads 等复杂的 Haskell 概念，转而讨论了高阶编程背后的哲学，以及 Haskell 所取得的成功和产生的影响和主流世界中的使用，最后探讨了同步教授计算机音乐与 Haskell 语言这一想法来作为总结。

Paul Hudak 是耶鲁大学计算机系的教授，Haskell 编程语言的主要设计者之一。他热爱艺术，开发了用于计算机音乐的 Haskore 函数语言。他最近在耶鲁助创立了计算与艺术专业。

InfoQ：我是 Sadek Drobi，这里是旧金山的 QCon，和我在一起的是 Paul Hudak。Paul，何不介绍一下您自己以及您所从事的工作呢？

Paul：首先，感谢大会邀请我来到这里。Eric Meijer 邀请我来这里作一个采访，我认识 Eric 已经很多年了，差不多是从我们参与到函数式编程早期的黄金年代开始。我在耶鲁大学已经工作了 26 个年头了，一切都进行得非常顺利，我仍然在从事 Haskell 以及函数式编程方面的研究工作，在也是我来到这里的原因。

InfoQ：我阅读了您关于 Haskell 的著作，令我惊讶的是，直到全书的后面你才提到类型类与 monads。您认为学习 Haskell 的步骤就是这样的吗？

Paul：是的，这就是我这样安排的原因，我觉得这是教授这些概念的最佳方式。在后面才讲 monads 而不是在前面，是因为我发现大多数的读者都已经有一些命令式编程的经验，并且期望能看到一些命令式计算的概念。事实上，我在这本书的前面就介绍了这一概念，大概在第 3 章，全书共 20 章节，在那里我只是将它们称作动作（action）。那里提到的 IO 动作就是这样的概念，只要是熟悉命令式编程的读者，理解这一部分是不会有什么困难的。

我在前面章节所保留的是 monad 的正式定义，当然如果要了解它，就必须要了解什么是类型类，才能真正从最广泛的意义上理解它。在本书的前面章节中我甚至有意避免使用“monad”这一术语，因为它会让人感到害怕。在函数式编程的社区里流传着这样一个笑话，我们应该从一开始就叫它们“温暖绒球”（warm fuzzy things：让人友好而有感情的事物），而不是叫它们“monads”，也许这是对的，谁知道呢。

这是一方面，但说到类型类，事实上，我认为应该在正式介绍什么是类型类之前介绍在介绍类型类上下文的概念。换言之，如果你看到的函数的类型签名有上下文或者约束等，就能联想到这是一个类型类。这一概念的介绍实际上能够提前，放在介绍类型类的声明以及什么是实例声明等等之前。

我想这也许是个好主意，因为我了解到的人们（并不只来自我的书，也来自学习 Haskell 的人们）的抱怨之一就是当你得到类型错误时，程序会丢出错误而错误信息自然地指向类型类或者其他。如果你不理解那些错误信息的话，那你就很不幸了。如果我重写这本书的话，我也许会做出一些小小的调整。回到原始的问题上：是的，我这么做是有意为之。

InfoQ：我们逐渐看到 Haskell 对其它主流语言所带来的启发。比如我们看到 Philip Wadler 为 Java 设计的泛型，以及 Eric Meijer 在微软的.NET 平台上所进行的大量工作。这是否可以看作是 Haskell 的成功呢？

Paul：我认为这是极大的成功。我认为衡量一个语言的成功并不仅仅是从语言本身被使用或者被接受等维度来衡量。Haskell 本身对其它语言所产生的影响或者说对整个编程系统的影响是巨大的。在语言设计方面有很多这类的例子，包括最近平地而起的一些新语言，如 Scala。另一方面是我经常听见人们说，实际上中午吃饭的时候，我还听到有人说："我一直在使用 Haskell 编程；我并不是在日常的工作中使用它，但是我喜欢这门语言。我工作中可能没用到它，但我理解它并且欣赏它的思想，而且我发现它影响了我使用 Java 或者 C#进行编程的方式"。这真是一件很酷的事情。

Haskell 给我的感觉非常好。在我看来，特别是近几年来，我觉得说它是成功的是没有任何问题的。就算它此时此刻结束了自己的历史使命，它所产生的影响也已经证明了它自己。我并不是说我们会止步不前解甲归田了，而是说明它的成功。在我看来确实如此。

InfoQ 您认为 Haskell 的目标是作为一门启发其它语言的研究用编程语言这一已实现的目标，或者它能真正用于主流的企业(开发)，比如软件？

Paul：它当然能用于主流开发，事实上它也一直主流开发中使用，并且赢得了越来越多的亲睐。我们最终有了能够更好的支持主流开发的各种库，各种实现以及相应的编程环境。这方面长时间以来都是一个障碍，但是像 Simon Peyton Jones 以及他所领导的微软小组在这方面做了令人难以置信的工作，他们为在业界需要使用这一语言的人们创建了相应的工具。另外，我们过去也常常担心性能可能是个大问题，而现如今的编译器也已经变得越来越好了，所以发自内心的讲，我实在想不出不选择 Haskell 的理由。我们将看来越来越多的人使用它，谁知道呢？

InfoQ：能给我们进一步谈一谈高阶编程吗？

Paul：能不能多讲讲高阶编程？它是一种思维状态，是一种思考的方式，它认为任何事物都是头等公民。当你在某一层次上做某事时，你可以将它抽象到一个更高的层次上。一旦你领悟了这一思想，它就无处不在。如果我突然不能使用 Haskell 或者被迫去使用比如 Java 或者 C#等语言——尽管这些语言已越来越多的在朝着函数式的方向努力——我将无比想念的不是类型或者延迟求值，但离开了高阶函数我就非常难熬。

它对于计算抽象而言是如此关键，并且它引领人们用一种极佳的方式思考问题和编写程序，因此我不知道人们离开了高阶函数会如何继续。我可以忍受失去一些其它特性，但高阶函数是不可缺失的。我们说“高阶编程”的时候，它同时也意味着一些别的事物。比如高阶类型等等，而如果你想知道更多的话，最近 Haskell 在类型系统与高阶编程方面着实走得比较远了。

我们直到最近才发现它所蕴含的强大力量。我认为故事才刚刚开始而已。高阶函数实际上非常直接，一旦你有了 lambda 算子，你就可以对任何事物加以抽象。而由于可判定性的问题和一些其它原因，类型系统相对而已就没有这么容易了。我想可能是由于我们的设计还没有臻于完善吧，目前仍有各种扩展和思潮在不断的发展当中。然而，当我们看待高阶类型些，它的能力就变得如此的明显，可以很肯定的说这就是我们需要的东西而且我们将不断的发展下去，谁知道发展到头会是什么样子呢？

InfoQ：当我们谈到函数与高阶函数时，我们同时也谈到函数式交互动画。您能为我们进一步解释一下吗？

Paul：动画，从图形学的角度上讲是我们所泛指的函数式反应性编程的一类实例。这一思想源于 Conal Elliot，是他将这一思想带入了 Haskell，我们相互合作并开发了 FRAN，它一个函数式反应性动画语言。在耶鲁大学，我们也在积极探索这一思想的其它应用，因为这其实是一个相当一般性的概念。而 Conal 一早就意识到了这一点，虽然他最初主要将其应用于动画领域。

我们将这一思想引入了机器人设计，现在又开始探索它在信号处理与声音合成方面的应用——我们已将其应用到了并行编程中。这是一个非常强大并且普遍的思想，其基本意思是，不把程序里面的变量想成值，而是想成静态的，因此一个给定的变量在一次迭代、函数的一次调用，或者递归中执行一次时仍保持原值。这样你完全可以将它想像成随着时间的每一次极微小的变化时都是保持一致的，这就是真正的持续值。

这就是我们想要展现的抽象。当然，这并不是我们的底层实现方式，但只要你这么做，各方

面的事情就都会套用到这一范型，再给你一个动画方面的简单例子。一个图像通常被当作是一个静态的值，但随时间变化的图像却是一个动画。有很多这样的例子，而我们最近所使用的是图形化用户界面的例子，关注在音乐或者是进行声音合成的计算机音乐应用这一领域。典型的是，我们通常用把手和滑棒来代表控制音量的值。

这只是些信号——我们习惯这么称呼——或者说行为中的一种。它是一个随时间变化的量，使用这一框架可以优雅的捕捉到它。当你提起任何事物，或执行任何操作，不管是加法这样的算术运算，还是积分或微分这样的有状态操作，你得到的都是一种完全不同的编程风格。当整个世界不再永远是连续的时候，真正的挑战来自于你如何将这一点与离散时间所发生的事件这一事实集成起来。比如，如何将点击鼠标或者敲击键盘集成起来？

要做到这一点，最终是使用了一种非常优雅的方式，引入了离散事件这一概念，但却是把这些离散事件看作是事件流——就像我们有连续变化的值一样，我们现在有连续的潜在事件流。可以开发出允许你在离散与连续之间进行仲裁的“话务员”，我们称它为交换器，它在基于事件的基础上会调用连续的行为来作出改变。它工作的如此之好，创造出一种高阶的编程方式。

要用 Haskell 实现它将完全依赖于高阶函数等元素，但不仅仅是实现它，而是将它概念化。这是一种正确的思考方式，而且这是一种很好的做法。

InfoQ：您的意思是只要我们知道一个事件流是用户事件，或者是其它的企业相关的 GUI 事件，它们可以被应用到 Web 应用上，对吗？

Paul：是的，我是这么认为的。最近我在这一领域作了许多工作。虽然在这一问题上还谈不上能以专家的身份来作什么评论，但同样还是我们之前所探讨的那些概念，包括希望 web 应用是无状态的这一事实，都能很好的适合这一模型，甚至包括对 web 应用的基于连续的理解这一思想，自然也是符合于这一模型的。

正如我所说，人们还在不断开发各种各样这样的概念。我认为这一领域非常有意义的一点在于，它不仅限于 web 应用，而适合普遍的因特网事务。当产生重叠时，就会产生特别的面向时间的事件序列，可能会碰上并发事件，可能会出现超时的情况或者某些事件使得你不得不停止其它正在进行的工作。这就直接引出了在 Haskell 社区里正讨论的如火如荼的另一个重要概念，那就是并发。

令人兴奋的意外收获是，当你处理好所有与实现高阶函数等等相关的问题之后，一旦这些机制都全部就位了，要加入并发就是一件再容易不过的事情了。因为要用高阶函数处理延迟求值，你必须要有一种封装计算的方式，挂起，延迟——不管你想怎么命名——，或者说闭

包(closure)。一旦有了这些机制，并发不过就是这种思想的另一种形式而已了。Simon Peyton Jones 和他的小组在这方面作出了杰出的工作，并为 Haskell 提供了这样的线程库，大大简化了这一类工作。

InfoQ：函数式抽象的能力与 OOP，面向对象相比，差别在哪里呢？

Paul：这两者截然不同。OOP 所带来的很多东西在函数式的世界实际上是不怎么存在的。或许 Haskell 的类型系统有点面向对象里的类的意味这一点会让人觉得有点疑惑，但它们没有相关的状态与之联系。我觉得 OOP 语言的一个问题是——这只是我个人的观点——他们搞混了几个不同类别的问题，而且我觉得完美的 OOP 语言还没有被设计出来。

关于状态，封装，以及对象与继承等问题都太容易造成人们的困惑，因此你会看到对象在 Java 这样的语言中以不同的方式被使用。使用它们的思想在概念上有着真正的不同这一事实也许正说明了它有什么不对劲的地方。再次声明，这只是我个人的观点而已，而且我认为学习面向对象编程是困难的。实际上，我在耶鲁教了够多的 Java 和 C#，因此我才知道它的不易之处以及学生容易产生困惑的地方。

教授这些语言的最近一些趋势是，至少直到学期的结尾以前，都不去强调这些语言面向对象的方面，这样学生才能首先掌握计算的基本思想。显然，面向对象编程有不少优秀的思想，我同样也不想贬低它们。为了达到同样的效果，比如继承，在函数式语言中必须要花费更多的功夫才行。你可以用高阶函数来模拟它，我曾经这么做过，比如我在计算机音乐库 HasKore 里就这么做过。

也许在面向对象的语言中能更容易进行表达，但这并不意味着我已经准备好转入到面向对象的语言中，不过可以肯定的是，它也有其精华的部分。

InfoQ：现在，我们越来越多的看到 monads 的应用。我的意思是，当 monads 初来乍到的时候，一些人认为它是一个奇怪的概念，而现在，仅仅过了几个年头，我们已经在 LINQ 或者其它的主流语言里面看到了它的应用。您认为 monads 是可以应用在任何地方的新的 API 抽象吗？

Paul：当然，它们可以被用于任何适合的地方。试着很好的理解 monads 以去理解哪些是适合/不适合使用它们的地方是一个挑战，而且这么做可以教给人们相关的技术并让他们理解 monad 到底是什么。遗憾的是，许多人都把 monads 当成或者是基本跟所谓的“状态 monad”——一种特殊的 monad——等同起来，因为它最主要是跟 IO 和命令运算联系在一起，但其实“状态 monad”不过是 monad 的一种而已。

真正要很好地处理到底 monad 的真正抽象意义何在，我觉得是一个挑战。意思是，我确实

认为 monads 应当被理解并且可以对帮助人们理解他们的软件产生影响，但这里还存在超越了 monads 的一些事。所以，真正重要的事情是要理解，monad 在某种意义上讲是对一种抽象计算的捕获，而这种计算可以非常正式的用 monadic 的运算符和 monadic 规则进行描述，但这并不是唯一的计算类型。

举例来说，最近在一些特定背景下变得越来越流行的东西就是所谓的 arrows，它是来自 John Hughes 的思想。arrows 可以被看作是一般化的 monads。你可以将任何 monad 编码成 arrows，但反过来则不行。目前 Haskell 实际已经在 monads 语法的基础上，又加入了 arrows 的语法。所以，这样一来，我们又得了另一种抽象计算。我们需要去理解它，我们需要去理解什么时候适合使用 monad，什么时候适合使用 arrow。

当然，自然而然，像“还有其它别的东西吗？”这样的问题就会接踵而来。是的，确实有。适用函子(Applicative functor)就是另一个例子，但它却是朝另一个方向在发展——它甚至有着更加简单的概念，比 monads 还要简单——因此更加适合于特定的背景。可能会出现这种情况，程序员可能会想到：“哇，我应该使用 monad!”而实际上，使用一个适用函子就已经足够了。而且适用函子有着更为简单的代数规则集合。因为它这些更简单的运算符，更易于使用的特点，使用得变得理所当然。杀鸡焉用牛刀呢？

与此同时，可能会出现这样的情况，一个应用应当需要一个 arrow，却错误的强行使用了 monad 去迎合它，造成的结果要不是无法工作，就是不能够表达你想要的东西。要处理所有这些问题，函数式编程社区本身也仍然在整理细节。最近相关的论文层出不穷，这些思想都是有待他们去解决的。

话说回来，这不是说因为 monads 可能不再是当初让人害怕的词汇，主流就不为之激动了，他们可以看到在其它的环境里已经发生的很好的例子，他们也可以尝试在新的环境里运用它们，这非常棒。事情本该如此。我想说的是，需要更多这样的思想涌现。而这些东西最后是否都会加入到主流里呢？我说不好，但可以肯定的是对像我这样的研究编程语言的人来说，这些东西都是足以让人兴奋的。

InfoQ：我们知道您致力于和对非程序员一起开发嵌入式 DSL，能给我们分享一下这方面的经验吗？

Paul：这方面我的经验也不能说非常丰富，但是今早我还谈到了这样的一个例子。实际上，我是和音乐家一起工作，比如，使用我们的计算机音乐软件库比如 Haskore 和 HaskSound 这样的 DSL，他们爱不释手，至少当我在一边旁观并提供一些帮助的情况下确实如此。这可以说是非常积极的一种经验。

我不得不承认我在所有的与完全的新手一起工作的领域中最具挑战的一项，以机器人为例吧：如果你跟一个机器人专家交谈的话，你不会找到没有从事过编程的机器人专家。比如 Greg Hager 和他的研究小组(Greg 现在在约翰霍普金斯，他曾经在耶鲁)：他在耶鲁的时候，我们和他一起合作过一种叫 FRAD 的 DSL，它使用我之前描述过的函数式反应性编程模型来控制机器人。

在那个时候，他所有的编程都是用 C 和 C++完成的。在此之前他从未使用过 Haskell，因此在这个叫做 FRAD 的嵌入 DSL 的环境中他是第一次使用 Haskell。他也非常喜欢 Haskell，不少他的学生也是，他们即使在离开耶鲁以后还继续着 FRAD 的工作。这并不是一拍即散的合作方式或是其他什么——相反他们真正发现了这是一种高效的编程方式。

诚恳的说，我认为我们应该积累更多这方面与非程序员的经验——当然，非函数式编程的程序员，并观察会有怎样的结果。就音乐这一背景来说，我现在所做的一项工作同时也算作一项挑战吧，就是写一本主题与编程无关的书，但我同时也认为这样的事情应当越多越好。也许我们可以在这个过程当中，也教授编程，但这本书的主要目的却是教授其它的主题。就我这个例子而言，我所说的就是计算机音乐。

实际上我是在将我先前的教材《Haskell School of Expression》在计算机音乐的背景下进行了重写，并且彻底地介绍了计算机音乐的思想。这样做意味着对于跟 Haskell 无关的事物有很多潜在的好处，但与此同时又完整地传授一门语言，不是以一种流于表面的方式，而是让他们真正的理解函数式编程的能力，因此一旦他们理解了 Haskell 或者理解了计算机音乐的思想，就能做出很酷的东西来。这并不非常事儿。如果投入更多的时间和精力的话，我们可以做得更好。我们也期盼着这些努力能有成效。

InfoQ：你从一开始就使用 Haskell，如今历经岁月沉淀，你认为真正的收获是什么？你如何看待 Haskell 的今天？有没有什么你想要删除或添加的功能？接下来的时间里，有没有哪个受 Haskell 启发的语言会引起你的兴趣？

Paul：首先，我感觉 Haskell 非常好。它多年作为一种生僻的研究语言，而从未受到主流的过多关注，最近所收获的成功则令人愉快而满意的惊喜。如果不是它最近所获得的这种关注，也不会有今天我在这里坐着接受采访了。就算退一步而言，它并没有获得这种程度的关注，我也没能在这里接受采访，我想我也不会因此而感到不安，因为我钟情于这一门语言而且一直都在使用它，它仍然是我最爱的语言——如果我想要编什么程序的话，我的抉择就是它。

正如我们之前谈到的那样，我对于它能影响这么多其它的语言感到非常欣慰，但除此以外也有很多非常酷的语言和功能强大的框架。模仿别人的工作就是最好的恭维，确实人们都在从

Haskell 吸收精华，这非常棒，但也没什么理由让我们不能从 Ruby on Rails 或者任何其它的新技术那里学习经验并影响我们的社区。我认为 Scala 就是一门非常棒的语言。

我们会看到越来越多这样的事物涌现出来，但现在令人高兴的是，人们都在互相借鉴对方的长处，而且有很多相辅相成的开发正在进行中。整个编程语言的范式非常有趣。我曾经从 DARPA 得到过很多赞助，而 DARPA 曾经有一段时间设计过 Ada 编程语言。Ada 的目的就是设计一个终结版的万能编程语言。试图达到 Ada 之后，再无语言。这是不现实的。

之后 DARPA 又有一个项目是要标准化一门原型语言。同样的错误又是，“让我们团结起来设计一门语言并让它成为最后的乐章。我们一定行!”这一次又是失败。在我 26 年的经历里我许多次听到 DARPA 或者赞助单位说：“编程语言研究已死”或者“编译器研究已死，我们已到了尽头”。非也！它仍然在不断进步，并且正变得越来越让人兴奋。我认为在许多不同的方向上如今都有令人欣喜的进展，并不仅仅是 Haskell，这可真是棒极了。

观看完整视频：

http://www.infoq.com/cn/interviews/paul-hudak-haskell

相关内容：

- 为什么函数式编程没有流行起来？
- 迷你书免费下载：动态函数式语言精粹
- .NET 4 中的模式匹配
- 使用Haskell和Hubris加强Ruby
- 超越F#基础——活动模式

热点新闻 | News

IntelliJ IDEA开源！

作者 Scott Delap 译者 霍泰稳

在逐渐增长的开源环境下，作为一个创新商业产品打拼了多年之后，IntelliJ IDEA终于步入开源之地。在它官方的新闻稿件中，这样提到：

> 从即将发布的 9.0 版本开始，IntelliJ IDEA将为市场提供两种版本：免费和开源的社区版，完全功能的旗舰版（即从前的IntelliJ IDEA）。 此举的最重大的意义在于，社区版的引入，完全清除了那些挡在广大使用IntelliJ IDEA进行纯Java开发的用户面前的障碍——价格。这个版本不仅免费，而且——这也是尤其重要的——完全开源。
>
> “一直以来，我们都通过公共早期采用计划（Early Access Program，EAP）、问题追踪者、论坛等对社区开放。这也使得我们和用户之间形成了紧密而又直接的反馈链条，即使在当时业界还没有广泛认可这一方式时，我们就开始这样做了。从那以后，我们通过免费的产品授权为数百个开源项目提供了支持，为 Groovy 和 Scala 等项目贡献源代码，我们自己也开发了许多开源的 IntelliJ IDEA 插件。” JetBrains 首席执行官 Sergey Dmitriev 表示，“所以说，大家能够看到我们是如何通过开源授权协议，免费为社区提供 IntelliJ IDEA，又如何致力于社区的发展的。开源已经成为主流，我们会继续拥抱这一令人激动的挑战。简单来说，我们并没有改变我们的方向——我们一直在努力发展！”
>
> 这一全新的社区版基于 IntelliJ 平台构建，包括其源代码。JetBrains 通过业界熟悉的 Apache 2.0 授权协议，让社区版和 IntelliJ 平台的源代码尽可能便于获得和使用。

除了社区版，JetBrains还将继续提供IntelliJ的商业旗舰版。该版本包含更多的功能支持，比如Android、GWT、Flex、JEEp和OSGi等。InfoQ会继续关注这一话题，及时报道来自社区的反馈。

原文链接：http://www.infoq.com/cn/news/2009/10/intellij-open-source

相关内容：

- 社区反应：IntelliJ开源，亡羊补牢？
- 一体化IDE发布
- IntelliJ IDEA的DSM工具使架构可视化
- IntelliJ IDEA 7 增加了对Groovy和Grails的支持
- IntelliJ IDEA 7.0 增加Spring/Hibernate支持、Eclipse互操作和Maven集成

热点新闻 | News

Sun将在Java 7 中摒弃Swing Application Framework

作者 Charles Humble 译者 张龙

Sun已经决定在Java 7 中放弃JSR 296：Swing Application Framework（SAF）。规范的领导者Alexander Potochkin在其博客中写到：

> 在多次讨论后，大家并没有就 Swing Application Framework API 达成共识，我们觉得还需要进一步设计才行。
>
> 我们已经将 SAF API 提交至 JDK 7 M5 了，进度就在那摆着呢，时间太紧了，所以我们得把 SAF 从 JDK 7 的里程碑版中移除。

只有两个特性（椭圆曲线加密技术及Swing JXLayer组件）加到了原来的M5 的计划中。结果Sun合并了M5 和M6 并将M5 的周期延至本月 29 号。现在的M5 具备最完整的特性，还将包含Project Coin的新特性和Swing JXDatePicker并更新至JAXP、JAXB及JAX-WS API（构成了Java XML技术栈）上，这一切使其成为最稳定的版本。在月底发布完整特性的构建版之后，Sun计划再发布 14 个构建版（直到明年 2 月份）以为RC版做准备。大家可以从OpenJDK的站点上找到完整的Java 7 特性列表。

自从Potochkin宣布放弃SAF的决定后，至少又出来了两个框架（BSAF及SAFF）。Jonathan Giles在其博客上表达了失望之情：

> 我们离开了最初的 SAF 项目，他们根本就不理睬（尽管这种情况在 Sun 不经常发生）这两个活跃的项目。如果一切可以重来，我们本可以将其集成到 JDK 7 中的，但我们都快等死了，现在只有期盼 JDK 8，JDK 7 是没啥指望了。

Giles在SAF邮件列表上进一步表示希望能将精力放在这两个框架其中之一上，这样所有的努力才不会白费。

> 假如这两个框架的拥有者同意，那我们就可以选择其中之一了并称之为 SAF.next。即刻停止对落选框架的开发才是明智之举。

Potochkin还向InfoQ证实另一个呼声很高的特性：在Swing组件中使用基于CSS的样式（很可能成为M5 Swing更新包的一部分）也将被Java 7 抛弃。JavaFX具备这个功能，而Ethan Nicholas为Swing创建了一个功能全面的原型并于 2008 年的夏天在其博客上谈到过该项目。Nicholas还在项目的站点上发布了一个教程并提供代码下载。

原文链接：http://www.infoq.com/cn/news/2009/10/java7_m5

相关内容：

- Jigsaw蓄势待发
- OpenJDK 7 / JDK 7 里程碑版 3 发布
- WindowBuilder Pro 7.0 发布了
- Sun停止资助SwingX的举动激怒社区
- Nimbus外观：Java的矢量用户界面

热点新闻 | News

使用Gestalt直接在HTML中嵌入Python、Ruby和XAML

作者 Jonathan Allen 译者 丁雪丰

Javascript 这门语言已经过了它的鼎盛时期，许多开发者现在更青睐 Ruby 或 Python 这样的语言，只有在为浏览器写代码时才会回到 Javascript 上来。当然，也有人尝试在浏览器中支持其他语言，比如支持 VBScript，不过它们从没有真正被用起来过。与此同时，HTML 语言又远远不能满足交互式应用程序的需要。虽然有可缩放的矢量图形（Scalable Vector Graphics），但没有 Internet Explorer 的支持，它就和 VBScript 一样没用。这让类似 Flash 的组件技术占据了统治地位，它们被 HTML 引用，但并不使用 HTML。

Gestalt是一个能够改变这一切现状的Javascript库。与Silverlight结合后，它可以让开发者直接在HTML中嵌入Python和Ruby。在查看站点源代码时，你会看到类似这样的东西：

```
<script language="python">
```

将这句话包含在页面顶部，这就是在着手写 Python 前你所要做的所有准备。它会将内联的 Python 或 Ruby 代码传给 Silverlight 运行时，该运行时支持 Dynamic Language Runtime。虽然 Gestalt 还只是一个演示，但它能让你在 HTML 中直接嵌入 XAML。XAML 代码被放在一个 XML 标签内，该标签的 class 属性值为“xaml”。微软的 Harry Pierson 表示，这种直接在 HTML 中嵌入 Silverlight 兼容代码的模型是 IronPython 和 IronRuby 项目的最终目标。

原文链接：http://www.infoq.com/cn/news/2009/10/Gestalt

相关内容：

- Gestalt：使用Ruby，Python和XAML编写网页脚本
- WPS：在网页中使用PostScript绘图
- 书评：用WPF勾勒世外桃源

热点新闻 | News

Brad Abrams终于完成了.NET RIA Services的开发系列文章

作者 朱永光

Brad Abrams是微软CLR和.NET框架团队的创始成员，也是多部重要书籍的合著者。他通过 4 个月的写作，完成了多达 26 篇关于如何使用Silverlight 3 RTM和.NET RIA Services July Update 进行开发的系列文章。这些文章通过一些示例或操作步骤循序渐进地讲述了开发RIA的各个方面，其中还蕴含了大量的最佳实践。由于他对.NET 运行时和框架深刻的理解，而且他的著作大都偏向设计向导方面的（如《框架设计向导，1、2 版》），所以这些示例除了告诉大家如何开发，也告诉大家如何设计。

这些文章包括如下内容：1，导航基础；2，丰富的数据查询；3，登录验证；4，SEO、导出到Excel及脱离浏览器运行；5，Astoria、添加服务引用和WinForms；6，数据传输对象（DTO）；7，用ADO.NET数据服务访问数据储存；8，用WCF访问数据存储；9，POCO和验证提供程序；10，使用LinqToSql；11，在客户端处理数据；12，使用DataSet；13，新建类库项目来组织代码；14，Visual Basic和WPF的支持；15，使用ASP.NET MVC；16，暴露WCF服务；17，逐步演化应用程序；18，自定义Linq提供程序；19；使用ASP.NET动态数据；20，使用NHibernate；21，访问层级数据；22，分离解决方案文件；23，使用Azure云服务；24，使用存储过程；25，使用ViewModel模式；26，加强验证功能和进行个性化。

要完成以上示例中的练习，需要安装VS 2008 SP1（包括Sql Express 2008）、Silverlight 3 RTM、.NET RIA Services July '09 Preview、Windows Azure Tools for Microsoft Visual Studio July 2009 CTP、Azure的账号、ASP.NET MVC 1.0、ASP.NET Dynamic Data Preview 4 Refresh、NHibernate（包括NHibernate Linq）和Fluent NHibernate。Brad也提供了源代码下载（请访问每个文章，以下载单独的源代码）和一个在线演示的地址。

原文链接：http://www.infoq.com/cn/news/2009/11/net-ria-services-guides

相关内容：

- 微软发表.NET RIA Services的路线图

热点新闻 | News

Ruby Enterprise Edition新版本切换至Ruby 1.8.7

作者 Mirko Stocker 译者 丁雪丰

Phusion发布了新的Ruby Enterprise Edition ,版本号为 1.8.7-20090928。之前的版本都基于Ruby 1.8.6，该版本开始基于Ruby 1.8.7。Ruby 1.8.7 发布已经有段时间了，为什么现在才切换到 1.8.7 呢？Phusion在发布声明中解释了这个问题：

> 我们起先并不愿意切换到 1.8.7 很多人报告说在引入 1.8.7 后遇到了各种不兼容的情况，而且其他几个 Ruby 实现都抵制 1.8.7。然而，现在的情况不同了。Rails 3.0 将不再支持 Ruby 1.8.7 之前的版本 ,JRuby 最近也决定支持 1.8.7 ,并且 OS X Snow Leopard 和所有 Linux 发行版都自带了 1.8.7。我们已经收到了来自社区的很多请求，希望能有一个基于 1.8.7 的版本。

本次发布还有一些其他的消息——新版本集成了Brent Roman的MBARI补丁，它能改善性能、减少内存开销。为了提升多线程速度，该版本中还包含了一个作为可选实验特性的补丁（作者是Joe Damato和Aman Gupta）。

这些补丁究竟起了多少改善作用呢？Twitter对该版本做了测试，正如Evan Weaver报告的那样，它在吞吐量上带来了显著的提升。Evan还补充道"比起-O2 或-O3（针对速度优化），用-Os（针对大小优化）编译的Ruby更快一些。Phusion的Hongli指出Ruby的指令局部性很糟，主要是靠将指令塞进指令缓存中来改善性能的"。

InfoQ 向 Ruby Enterprise Edition 团队咨询了在升级到新版本时是否存在一些问题。Phusion 的 Ninh Bui 回答道：

> 呃，除了从 1.8.6 到 1.8.7 的兼容性问题，tcmalloc——我们使用的内存分配器——在 Snow Leopard 上还无法正常工作。

Ninh 的同事 Hongli 还补充说：

> 目前 Xen 的支持方面还存在一些问题：有报告说 REE 在 Xen 中运行时输出了很多（无害的）警告信息。我们计划在以后的版本中加入 Xen 特有的编译标志来解决这个问题。至于 1.8.6 和 1.8.7 的不兼容性，这是由语义变化造成的，例如，一些方法现在不再返回 Array，改为返回 Enumerable；一些标准库类有细微变化，等等。
>
> 社区已经接受了这些变化，因此我并不认为会有不兼容的问题。。

您可在www.rubyenterpriseedition.com获取到Ruby Enterprise Edition的新版本。

原文链接：http://www.infoq.com/cn/news/2009/10/ree

相关内容：

- Ruby VM综述：MacRuby进展、IronRuby及Ruby 1.9.2 延期
- JRuby综述：1.4 的新特性、JRubyConf议程及MLVM
- Gregg Pollack和他的Scaling Rails教学视频”
- 基于Rails的企业级应用剖析
- 书摘与采访：Rails for .NET Developers

热点新闻 | News

SOA宣言发布

作者Mark Little 译者 黄璜

敏捷宣言成为敏捷软件开发者的首选参考主要出于两方面的原因 :它是由思想深远的领袖撰写的，并且有着简短而易获取的格式。宣言将两两事物之间进行对比，来表述孰重孰轻，接着提供了原则，对这些核心价值进行了解释与扩展。通过这一格式，突出了敏捷软件开发的核心价值。SOA最近年逐步走向成熟，最近，一群SOA实践者/报道者/起草者注意到适合使用敏捷宣言的这一格式，来起草SOA宣言，并尝试能对SOA开发者和使用者社区起到同样的帮助。最近，在鹿特丹的第二届国际SOA讨论会上，制定了SOA宣言。在参与鹿特丹会议之前，起草者基于他们自己的见解与同行之间的意见，都准备了他们自己的宣言。

就像这里所报道的一样当时的争论异常激烈，就像你能想像到的其它重要的，同时也是经常如SOA一般被错误定义的事物一样。然而，人们达成了共识，虽然就像其它的任何一个工作组一样，并不是每个人都得到了他们想要的每一点。尽管鹿特丹的争论激烈异常，最终这个工作组却达成了令人惊讶的高度共识。这一宣言在这次SOA讨论会上首次宣告，这次宣告还被录制下来：

注：视频来自Youtube，由于众所周知的原因未能显示，请自行设法观看：http://www.youtube.com/watch?v=TCg16oTZSV0。

在这一宣言发布的短短时间之内，就引起了众多的评论，有正面的，也有负面的。为了遵照原始的敏捷宣言风格，SOA宣言也保持了简短与直截了当的原则。然 而，这同样也会造成在表达方面的不足。要用少量的词汇描述大量的信息，产生的表述不可避免地会具有歧义。比如，“内在的互操作性”可以被理解为购买ESB的强烈建议，整个的互操作性都基于它标准化的能力与格式。然而，从鹿特丹的讨论看来，小组的成员似乎从一开始就考虑要为服务本身设计互操作性。后一种解释与陈述产品不能带给你SOA的原则不谋而合。

如果 SOA 宣言要得到普遍的接受，SOA 社区首先要统一这些陈述应该如何来理解。如若不是，争论将不会休止，而 SOA 宣言也无法为填补 SOA 社区的空白作 出贡献 :对于 SOA 核心

价值的共同认识。正是如此，完整的把握整个宣言，以及其起草者后续地对于他们所要达成的目的进行更深的讨论，才显得如此重要。如果 不及时地做这项工作，很有可能出现的情况是 ,宣言本身将会被更多的口头争论而掩盖 ,而这些争论其实都是基于对宣言片面的理解。

注，Herbjörn Wilhelmsen对本文亦有贡献。

原文链接：http://www.infoq.com/cn/news/2009/10/soa-manifesto-released

相关内容：

- SOA宣言的发展
- SOA联盟改弦易张以填平业务与IT之间的沟壑
- 与WOA融合——走出SOA困境
- SOA的管理策略
- 拉近SOA和BPM的距离

热点新闻 | News

支持云应用程序服务的PHP API

作者Abel Avram译者 赵劼

自称“PHP公司”的Zend Technologies发起了一个开源的Simple API for Cloud Application Services项目，希望可以帮助PHP开发人员在构建应用程序时访问各主流云平台。微软已经为PHP开发人员提供了Windows Azure SDK。

许多技术供应商，如Zend、IBM、微软、Nirvanix、Rackspace和GoGrid参与了Zend领导的项目，希望可以创建易于使用的API来访问各种云资源。一开始，这个API提供了对Amazon Web Service的文件存储、文档数据库存储及简单队列服务，Rackspce云文件、Windows Azure和Nirvanix存储分发网络的支持。Zend希望可以将这个API作为Zend Framework的新组件，并称之为Zend Cloud。

微软在 7 月份发布了他们的Windows Azure SDK for PHP Developers ,这是一套帮助PHP程序员使用Azure的工具。不久后 ,微软又向Zend的简单API提供了SDK。这个SDK由微软与RealDolmen联合开发，目前使用New BSD协议存放在CodePlex站点中。

PHP 的 Windows Azure SDK 也提供了存储以外的支持：

- 访问 Windows Azure 大文件、表格及队列的 PHP 类（ CRUD 操作 ）
- 用于 HTTP 传输、AuthN/AuthZ、REST 及错误管理的辅助类。
- 管理、工具及日志支持。
- 支持将 PHP 会话存放在 Azure 表格存储中。

目前，PHP 开发人员可以同时使用 Simple API 和这个 SDK，尤其是在那些面向存储以外的任务上。不过，将来的 Simple API 也会包含越来越多的功能。Zend 建议目前在产品环境中小心使用这个 API，因为它们还处于早期状态，很可能会有所改变。

这些接口可以很容易从 PHP 移植到其他 OO 语言上 ,因为它们原本就是以 OO 的方式设计的。

观察这个项目是否会延伸出面向 C#或 Java 的项目也是一件有趣的事情。

原文链接：http://www.infoq.com/cn/news/2009/10/PHP-Simple-API-Cloud

相关内容：

- 实现“软件+服务”（S+S）的注意事项
- 微软凭借新增的两个数据中心进军云计算
- SQL Azure万事俱备，用户需要在 12 月前迁移到新的CTP上
- PHP 6 将全面支持Unicode和国际化
- 纯GET的REST集成模式——是同步，还是集成？

热点新闻 | News

ZDNet发布中国最受欢迎 50 大技术博客

作者 霍泰稳

日前，ZDNet联合InfoQ、博客园等国内技术社区，对外发布了“最受欢迎中国技术博客(PB50)”榜单，主要包括软件、网络、安全、存储、服务器和企业管理信息化等 6 个技术领域，InfoQ中文站编辑赵劼、高昂、王翔、肖桦等名利其中。InfoQ中文站编辑就该榜单相关话题采访了其主要发起人，ZDNet主编李宁。

在该榜单的简介中，发布者解释说这一榜单不是中国技术博客的绝对排行榜，但可以给关注中国技术博客的用户一个参考，可以大体了解在技术社区中有哪些人在笔耕不辍，努力为他人分享自己在技术领域的见解。在最受欢迎技术博客评选第一期中，排名前十位的博客为：

1. 李会军：软件公司项目经理。多本技术图书作者译者。擅长基于.NET平台的Web开发，ASP.NET、 AJAX、 Silverlight等。
2. 赵劼：资深软件架构师，InfoQ中文站.NET社区编辑。目前担任上海柏盛网络技术有限公司架构师。
3. 周爱民：系统架构师，就职支付宝。十余年软件开发、项目管理经验。多本技术图书作者译者，如《大道至简》。
4. 吕建伟：CTO，10 年以上商业软件从业经验，专注行业管理信息化领域。《走出软件作坊》一书作者。
5. 陈皓：技术主管。多年银行电信业务系统开发经验。多年网格计算/分布式平台研发经验。
6. 盖国强：Oracle ACE 总监。目前从事独立数据库服务咨询，擅长数据库诊断、性能调整与SQL优化等。
7. 胡长城：系统架构师。就职于普元。多年J2EE项目架构和实施经验，以及工作流研发经验。长期进行工作流培训推广。

8. 陈广琛：就职于百度。Web工程师，多本技术图书译者。擅长基于.NET平台的Web开发。
9. 张大志：haopinpro创始人。从事研发人员考核的培训与咨询。
10. 冯强：信息化咨询/IT行业研究。

其他 40 名博客可直接浏览榜单。InfoQ中文站编辑就该榜单制作的初衷、博客主和读者的反馈、评价标准，以及未来的构想等和ZDNet主编李宁进行了沟通。

InfoQ：制作“最受欢迎的 50 大中国技术博客”的初衷是什么？

李宁：博客已经是人们获取、分享信息的重要渠道。在国内的技术圈里更是涌现出了许多不错的技术博客，他们拥有专业的知 识背景，以自己的视角评述焦点事件、解读 IT 技术。为了让这些优秀的博客被更多的网友了解，所以，ZDNet 联合国内主流技术网站，尤其是 InfQ 中文 站，启动了“最受欢迎的 50 大中国技术博客”PB 50（POP Blogger 50）计划。通过该计划定期评选出最受网友欢迎的 50 个技术博客，帮助博主与网友建立起更多的交流机会。

InfoQ：在第一期评选的过程中，你收到了哪些有价值的反馈？

李宁：

1. 很多博主发现自己的博客被网友投了很多赞成票，他们表示自己博客能被网友认可是一件非常开心的事情。
2. 很多博主表示，通过这个活动，也一定程度的激励自己更加勤奋的通过博客的形式与大家分享技术带来的快乐。
3. 很多参与投票的网友表示，通过这次活动，发现了很多国内优秀的技术博客，可以认识很多牛人，感觉很有收获。
4. 这次评选，参与活动的主要技术社区，还得到了来自国内 Sina，Sohu，163 等 20 多家网站的大力支持，他们很感兴趣对于这样的活动。
5. 不足，参赛博客数量太多，来自的领域也比较广泛，博客的内容很比较多样，所以有博主反应是否以后可以增加更多专项的评选。

InfoQ：如何让这次评选尽可能公平、公正？

李宁：公平是、公正是在整个活动中，最重要考虑的一个问题。在很多地方，都进行了相关的设置。例如，在投票环节，设置 网友投票，检审核组，两轮投票。技术方面，每天都分

析日志，防止恶意刷屏的行为。目前来看，第一期投票没有发现相关的问题。当然也会不断优化评选规则，做 到尽可能的公平，公正。

InfoQ：给读者推荐几个企业级开发领域比较有价值的博客。

李宁：其实，参赛的很多博主都是非常的有价值，从某种意义上说，排名并不是绝对的。这儿我推荐几位我经常阅读的博客：赵劼、周爱民、吕建伟、胡长城。

InfoQ：对该榜单的下一步发展，有没有什么设想？

李宁：评选不会脱离初衷，帮助博主与网友建立起更多的交流机会。在评选环节设计上，会不断优化评选流程，让评选更具公正、公开。在评选项目上，会考虑增加例如，按照技术领域划分的单项评选等，从而增加评选的权威性。

原文链接：http://www.infoq.com/cn/news/2009/10/zdnet-top-bloggers

相关内容：

- Enterprise 2.0，一个时髦的新词儿
- CodePaste.NET——代码片段交换站点
- 采访：Miko Matsumura谈AlignSpace
- 挑战 2009——创造商业价值的架构趋势
- 软件自由日 2008 北京站即将举行

热点新闻 | News

如何进行平台型网站架构设计？

作者 霍泰稳

在欧拉的“平台网站架构设计之我所见”的博客中，他从选择技术方案和物理架构、平台研发和架构优化等方面阐述了他在多年的平台型网站架构设计过程中的经验心得。

欧拉先是分析了在选择技术方案和物理架构中的几个常见问题，以来解决如何提高开发效率，使平台具有高性能、高负载性的问题：

> 开发语言和数据库：我个人觉的最关键是你和你的团队最擅长的开发语言和数据库是哪个，古语有云：“工欲善其事,必先利其器！”，趁手的开发语言和数据库有助于事半功倍。试想如果你选择了一个并不很熟悉的语言，也许这个语言和数据库在基础性能上的确比你掌握的语言好，但是在研发过程中学习曲线肯定长。
>
> 成熟框架还是自己实现：我个人的一些经验是，尽量使用开源的成熟框架，因为平台研发初期使用成熟的开源框架，能提高开发效率，并且在质量上有保证。我曾经接手过一个平台的改版，框架是前面开发人员自己写的，里面的一些设计思想不是很成熟，导致平台在负载增高后性能很差，整改起来很麻烦。

除此之外，Web Server/DB Server/Cache Server 的选择也是很重要的一点，欧拉认为这一部分一定要使用具有前瞻性、易配置、能监控和维护的产品，并提出几个选型的标准：丰富和深入的配置选 项、基于高并发模型、支持负载均衡和请求分发、高效的缓存机制、实时的状态监控机制等。而对于操作系统的选择，则要稳定安全、易管理和维护、易监控等。对 于物理架构，即服务器的搭建方式，欧拉同样认为前瞻性是非常重要的：

> 平台初期的话，我想大部分访问量都不高，Web Server/DB server/Cache Server 放在一台服务器上都没问题。但是自己心里最好能预估一下这个平台会发展到什么样的规模，在做架构设计的时候，按照事先预估的来决定怎么做物理架构，并为以后的架构升级做准备。说到这里，想到前百度架构师雷鸣说过的一句话，当你的会员数达到目前的 5 倍或 10 倍的时候，架构就要升级。

在实质性的研发过程中，需要注意的是平台网站研发的模式和传统 IT 项目研发的不同，以前是开发过程中要和客户、需求人员等打交道，而现在关注的是产品设 计。对于平台网站研发项目的管理，欧拉推荐使用敏捷开发方法，通过设立短的发布周期进行迭代开发，并使用 JIRA 等成熟的项目管理系统进行管理等。对于团 队研发需要注意的地方，欧拉总结如下：

- 合适的开发工具；
- 如何控制代码质量
- （根据）需要引入新框架；
- 知识总结和培训。

对网站研发过程中的架构优化、存储和搜索等关键点，欧拉也分享了自己的经验，比如对于网站速度慢，而却又不知如何下手的问题，欧拉提出的解决方案是：

> 我的经验是从数据开始，从最外围开始画圈，找到源头。先从外围开始收集日志，比如 access_log 访问日志或 sql_log 数据库操作日志，找出访问最多的 10 条日志和执行时间最长的 10 条日志，然后根据日志去反查到底是什么引起的操作，然后一条条的解决。如果解决不了，那么就考虑重构。

在文章的最后，欧拉来分享了一些他认为有价值的资料，比如新型的大型BBS架构（Squid+Nginx）、Nginx图片服务器的架构方案、校内相册发展过程及核心技术分析爆料、架构设计贵在务实、大型网站架构不得不考虑的 10 个问题等。

原文链接：http://www.infoq.com/cn/news/2009/10/how-to-design-platform-websites

相关内容：

- 与冯大辉谈数据库架构
- 豆瓣网技术架构变迁
- 架构师的唐诗三百首
- 架构风格和架构模式速览

推荐文章 | Articles

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。

原文链接：http://www.infoq.com/cn/articles/js-for-h5

相关内容：

- OSGi原理与最佳实践（精选版）
- Anissa和Judy谈Glassfish的开发与测试
- IE中使用Google Chrome Frame运行HTML 5
- SproutCore：一个HTML 5 应用框架
- Google Wave会影响RIA/Silverlight吗？

InfoQ中文站

www.infoq.com/cn

我们的**使命**：成为关注软件开发领域变化和创新的专业网站

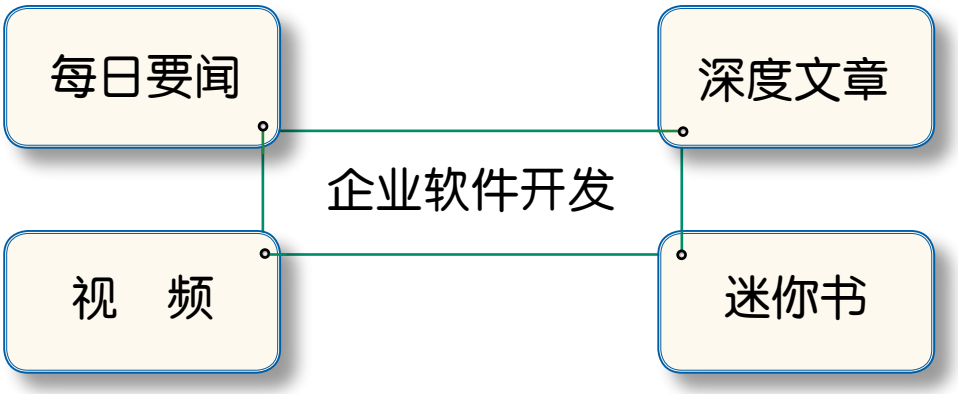
我们的**定位**：关注高级决策人员和大中型企业

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

我们的**特质**：个性化RSS定制、国际化内容同步更新

我们的**团队**：超过30位领域专家担当社区编辑

……



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

推荐文章 | Articles

基于Azure云计算平台的网格计算，第 2 部分：开发网格应用

作者 David Pallmann 译者 朱永光

在本系列的第 1 部分，我们介绍了在Azure上进行网格计算的设计模型。在这篇文章中，我们将用C#来开发一个网格应用程序以实现这个模式；而在第 3 部分，我们将首先在本地运行这个应用程序，接着在云中运行。为了实现这些功能，我们需要网格计算框架提供的辅助功能。

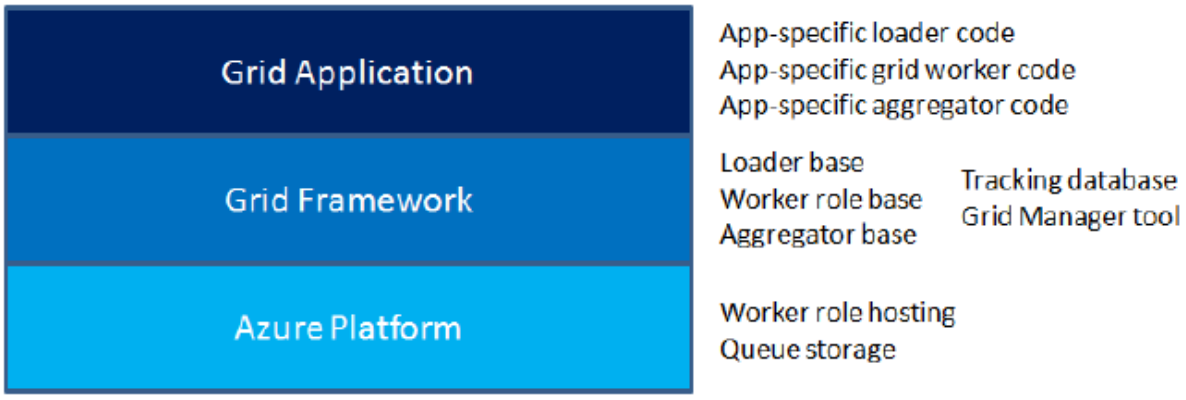
网格框架的角色

除非你准备编写大量的底层基础软件，那么应该为你的网格应用程序选用一个框架，来消除繁重的工作，让你集中精力于应用程序代码的编写。虽然 Azure 实现 了你想在网格计算基础结构中所需的很多服务，但仍然需要在 Azure 和网格应用程序之间添加一些特定于网格的功能。一个优良的网格计算框架应该为你完成如下工作：

- 提供对工作运行的计划调度和控制能力
- 从底层存储中检索输入数据。
- 为网格执行器生成任务以便执行
- 把任务分发到可用的执行器
- 在网格执行应用程序的时候跟踪任务的状态
- 从执行器中收集结果
- 把结果存储到底层存储中

下图显示了框架如何把网格应用程序和 Azure 平台结合到一起。应用程序开发人员只需编写

应用程序特定的代码去加载输入数据、生成任务、执行任务和保存结果数据。这个框架提供了全部所需功能——这些功能极大地利用了 Azure 平台的特点。



在本篇文章中，我们将利用Azure Grid，一个Neudesic Grid Computing Framework的社区版本。Azure Grid提供了 4 个软件组件，来实现列在下面的所有功能：

- 加载器，让你可以添加自己的代码，来从底层资源中提取输入数据并生成任务。
- 执行器角色，让你可以添加自己的代码，来执行应用程序任务。
- 聚合器，让你可以添加自己的代码，来把结果存储回底层资源。
- 网格管理器，让你启动工作运行，并监测它们的执行情况。

Azure Grid 只在你的网格应用程序执行期间才使用云资源，使你的费用尽量最低。底层存储保存着输入数据、结果和 Azure Grid 的跟踪数据库。云存储用于与执行器通信过程的参数传递和结果收集，且在你的网格应用程序执行的时候把它们都清空。一旦你的网格应用程序执行完成， 在空闲的时候，你也可以挂起网格执行器的运行实例，那么就无需为存储和计算时间支付持续的费用。

应用程序：Fraud Check

我们将要编码的应用程序是一个虚构的欺诈检查（fraud check）程序，使用某些规则对申请者数据进行计算，以求出欺诈可能性分数。每个申请者的记录都作为一个网格任务来进行处理 。申请者记录具有这样的结构：

LastName	FirstName	State	Country	Age	SSN	Relation	MosEmployed
Beethoven	Ludwig	CA	USA	112	111-11-1111	Applicant	48
Bach	J.S.	NY	USA	27	156-77-8935	Applicant	0
Mozart	Wolfgang	Tirol	Austria	23		Co-Applicant	3
Strauss	Johann	GA	USA	48	095-65-7844	Co-Applicant	150
Tchaikovsky	Pyotr	TX	USA	17	110-44-15435	Applicant	6
Chopin	Fryderyk	NY	USA	90	110-44-15435	Applicant	1
Gershwin	George	NY	USA	81	110-44-15435	Applicant	14
Haydn	Joseph	NY	USA	37	110-44-15435	Co-Applicant	7

通过在申请者记录上应用业务规则，Fraud Check 程序可算出一个 0 到 1000 之间的欺诈可能性分数，而 0 表示最坏可能的分数。如果分数低于 500，那么申请可能被拒绝。

设计网格应用程序

在你设计网格应用程序的时候，你需要确定能把工作划分到可并行执行的独立任务的最好方法。你首先要考虑 2 个关键问题：

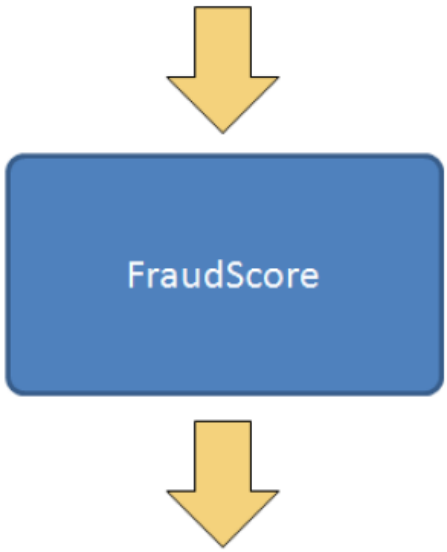
- 你基于什么基础来划分工作为任务？
- 有多少种不同类型的任务？

在 Fraud Check 这个例子中，为每个申请者记录创建单独的任务是很有道理的：为每个记录评出欺诈分数是一个原子操作，而且在所有的记录处理完成后，它们的顺序如何也无所谓。

对于 Fraud Check 而言，只需要一种任务类型，我们将其命名为“FraudScore”。FraudScore 任务就是为申请者记录算出欺诈分数。

这些任务需要读取输入数据，生成结果数据。FraudScore 的输入数据也即申请者记录，而结果数据则是欺骗分数加上一个文本字段来解释得到这个分数的原因。FraudScore 所需的参数和返回结果，连同其名称一起显示在下面。

Parameters	
LastName	Bach
FirstName	J.S.
State	NY
Country	USA
Age	27
SSN	156-77-8935
Relation	Applicant
MonthsEmployed	0



Results	
Score	700
Notes	Employed less than 1 year
Accepted	1

在某些网格计算应用程序中，任务在完成工作的时候可能也需要访问额外的资源，比如数据库或 Web Services。FraudScore 没有这样的需求，不过如果需要的话，可以通过输入参数来提供必需的信息，如 Web Service 地址和数据库连接字符串。

开发网格应用程序

现在，我们的网格应用程序的输入参数、任务和结果字段已经定义好了，我们可以继续编写应用程序了。Azure Grid 只要求我们编写加载器（Loader）、应用程序任务和聚合器（Aggregator）的代码。

编写加载器代码

加载器代码负责读取输入数据，并生成附带参数的任务。大部分时候，这些数据都来自于数

据库，不过 Fraud Check 编写成从电子数据表中读取输入数据。

Azure Grid 通过一个名为 AppLoader 的类，为你的加载器提供了一个可以开始编码的模板。需要实现 GenerateTasks 方法 ,来获取你的输入数 据 ,生成带有任务类型名称和参数的任务。你的代码创建 Task 对象，并作为数组返回。在基类中，GridLoader，把你的任务处理为队列后放到任务执 行位置的云存储中。

```
#region Application-Specific Loader Code

/// <summary>
/// Your applications' Loader code goes here.
/// Responsibilities:
/// 1. Read input data necessary to create tasks with parameters from local
resources.
/// 2. For each task generated, create a Task object with input parameters.
/// 3. Return an array of Task objects.
/// </summary>
/// <param name="jobId">Job id</param>
/// <returns>Task[] array</returns>

public Task[] GenerateTasks(string jobId)
{
    List<Task> tasks = new List<Task>();
    Dictionary<string, string> parameters = new Dictionary<string,string>();

    // TODO: implement Loader


    // Example task creation:

    parameters["Param1"] = "Value1";
    parameters["Param2"] = "Value2";
    parameters["Param3"] = "Value3";
    tasks.Add(new Task(ProjectName, jobId, 1, "GridTask1",
Task.Status.Pending, parameters, null));

    return tasks.ToArray();
}

#endregion
```

为了实现 Fraud Check 的加载器，我们用下面的代码替换任务创建的示例代码，以从电子数据表 CSV 中读取记录，并为每条记录创建一个任务。

```
using (TextReader reader = File.OpenText("FraudInput.csv"))
{
    string[] names = null;
    string[] values = null;

    string line;
    int lineCount = 0;
    int nextTaskId = 1;

    // Read lines from CSV file until empty. Line 1 contains parameter names.

    while ((line = reader.ReadLine()) != null)
    {
        lineCount++;

        if (lineCount == 1)
        {
            // Reader header row of parameter names.
            names = line.Split(',');
            int n = 0;
            foreach (string name in names)
                names[n++] = name;
        }
        else
        {
            if (!String.IsNullOrEmpty(line))
            {
                // Load latest values for this row and generate a task
                values = line.Split(',');

                parameters = new Dictionary<string, string>();

                for (int i = 0; i < names.Length; i++)
                    parameters[names[i]] = String.Empty;

                for (int i = 0; i < values.Length; i++)
                    parameters[names[i]] = values[i];

                tasks.Add(new Task(ProjectName, jobId, nextTaskId++,
"FraudScore", Task.Status.Pending, parameters, null));
            }
        }
    }
}
```

输入的电子数据表的首行应该包含参数名称，而后面的行应该包含值，正如之前显示的那样。创建任务的过程很简单，就是初始化一个 Task 对象，并构造器中赋给它如下信息：

- Project Name：你的应用程序的项目名称。这从配置文件设置中读取。
- Job ID：工作运行的编号，一个字符串。这个值是由外部提供给 GenerateTasks 方法的。
- Task ID：这个任务的唯一标识符，一个整数。
- Task Type：要运行任务的名称。

- Task Status：应该设置为 Task.Status.Pending，以表明这是一个还未运行的任务。
- Parameters：参数名称和值的字典集合对象。
- Results：NULL——结果将由网格执行器在执行任务后来设置。

把 Task 添加到一个列表集合中，就完成了这部分工作。一旦所有的任务都生成好，把 `List.ToArray()` 作为结果传递给加载器，它就会把这些任务排队到云存储中。

编写聚合器代码

编写好加载器之后，就是聚合器，其处理任务结果，并在本地存储它们。

Azure Grid 通过一个名为 AppAggregator 的类，为你的聚合器提供了一个可以开始编码的模板。需要实现 3 个方法：

- OpenStorage ,在第一个结果已经准备好可以处理的时候调用 ,让你有机会打开存储资源。
- StoreResult，在每个结果需要保存的时候调用。输入参数和结果都用 XML 来传递。
- CloseStorage，在最后一个结果已经保存好后调用，让你有机会关闭存储资源。

在基类中，GridAggregator 处理来自云存储中的结果，并调用你的方法来存储这些结果。

```
#region Application-specific Aggregator Code

/// <summary>
/// You application's Aggregator code goes here.
/// Responsibilities:
/// 1. OpenStorage - open local storage.
/// 2. StoreResult - store a result.
/// 2. CloseStorage - close local storage.
/// </summary>
/// <param name="resultXml"></param>

protected override void OpenStorage()
{
    // TODO: open storage
}

protected override void StoreResult(string parametersXml, string resultsXml)
{
    // TODO: store result
}

protected override void CloseStorage()
{
    // TODO: close storage
}

#endregion
```

在 StoreResult 中，当前任务的参数和结果以如下格式的 XML 来传递：

```
<Parameters>
  <Parameter name="LastName" value="Bach"/>
  <Parameter name="FirstName" value="J.S."/>
  ...
</Parameters>

<Results>
  <Result name="Score" value="700"/>
  <Result name="Approved" value="1"/>
  <Result name="Notes" value=" "/>
</Result>
```

为了实现 Fraud Check 的聚合器，我们将完成同加载器相反的事情，即把每个结果添加到电子数据表 CSV 文件中。

- 在 OpenStorage 中，打开一个.csv 文件来接受输出，把结果写出到电子数据表 CSV 文件的行列中。
- 在 StoreResult 中 ,结果(以及包含在这个上下文中的输入参数的第一个和最后一个名称 ）从 XML 里提取出来，写出到 CSV 文件中。
- 在 CloseStorage 中，文件被关闭。

```
#region Application-specific Aggregator Code

/// <summary>
/// You application's Aggregator code goes here.
/// Responsibilities:
/// 1. OpenStorage - open local storage.
/// 2. StoreResult - store a result.
/// 2. CloseStorage - close local storage.
/// </summary>
/// <param name="resultXml"></param>

TextWriter tw = null;

protected override void OpenStorage()
{
    tw = File.CreateText("FraudOutput.csv");
    tw.WriteLine("Last Name,First Name,Score,Accepted,Notes");
    tw.Flush();
}

protected override void StoreResult(string parametersXml, string resultsXml)
{
    XElement parameters = XElement.Parse(parametersXml);
    XElement results = XElement.Parse(resultsXml);

    // Write values

    tw.WriteLine("{0},{1},{2},{3},\"{4}\"",

parameters.XPathSelectElement("/Parameter[@name='LastName']").Attribute("valu
e").Value,

parameters.XPathSelectElement("/Parameter[@name='FirstName']").Attribute("val
ue").Value,

results.XPathSelectElement("/Result[@name='Score']").Attribute("value").Value
,

results.XPathSelectElement("/Result[@name='Accepted']").Attribute("value").Va
lue,

results.XPathSelectElement("/Result[@name='Notes']").Attribute("value").Value
);

    tw.Flush();
}

protected override void CloseStorage()
{
    if (tw != null)
        tw.Close();
}

#endregion
```

编写应用程序任务代码

在编写好加载器和聚合器后，还有一块功能需要编写：应用程序代码本身。AppWorker 类用来包含应用程序任务代码。当前任务被传递给一个名称为 Execute 方法，其检查任务类型，以决定执行哪些任务代码。

```
        #region Application Code

        /// <summary>
        /// Application code to execute a task. The switch statement uses the
task's TaskType string to determine the appropriate code to execute.
        /// </summary>
        /// <param name="task">The task to execute.</param>

        public override void Execute(Task task)
        {
            switch (task.TaskType)
            {
                case "Task1":
                    Task1(task);
                    break;
                case "Task2":
                    Task2(task);
                    break;
                default:
                    // Ignore unknown task.
                    break;
            }
        }

        private void Task1(Task task)
        {
            // TODO: implement task1
        }

        private void Task2(Task task)
        {
            // TODO: implement task 2
        }

        #endregion End Application Code
```

对于 Fraud Check，在我们的应用程序中使用 switch 语句来检查我们任务的类型——FraudScore，并执行代码基于在输入参数中的申请者数据来计算欺诈可能性分数。

```
public override void Execute(Task task)
{
    switch (task.TaskType)

        case "FraudScore":
            FraudScore(task);
            break;

        default:
            // Ignore unknown task.
            break;
    }
}
```

FraudScore 代码的首要业务逻辑就是提取输入参数，在 Task 对象中，可以通过名称和字符串值的一个字典集合来逐一访问。

```
private void FraudScore(Task task)
{
    StringBuilder notes = new StringBuilder();

    bool rejected = false;
    int score = 1000;

    string firstName = task.Parameters["FirstName"];
    string lastName = task.Parameters["LastName"];
    string state = task.Parameters["State"];
    string country = task.Parameters["Country"];
    int age = Convert.ToInt32(task.Parameters["Age"]);
    string ssn = task.Parameters["SSN"];
    string relation = task.Parameters["Relation"];
    int monthsEmployed = Convert.ToInt32(task.Parameters["MonthsEmployed"]);
```

接下来，执行一系列的业务规则算出分数。下面是一个摘录：

```
// Rule: if age < 18 or age > 100, automatic rejection

if (age < 18 || age > 100)
{
    score = 0;
    notes.Append("Age out of range, automatic rejection. ");
    rejected = true;
}

// Rule: is SSN missing, reduce score by 300.

if (string.IsNullOrEmpty(ssn))
{
    score -= 300;
    notes.Append("SSN missing. ");
}

  ...

// Check score. If below 500, reject application.

if (score < 0) score = 0;

if (!rejected && score < 500)
{
    rejected = true;
    notes.Append("Score below 500, rejection. ");
}
```

最后，FraudScore 更新任务的结果属性。也是简单地在字典集合中设置名称和字符串值。

```
        // Store task results.

        task.Results["Score"] = score.ToString();
        task.Results["Notes"] = notes.ToString();

        if (rejected)
            task.Results["Accepted"] = "0";
        else
            task.Results["Accepted"] = "1";
    }
```

GridWorker 这个基类和 WorkerRole 实现了把结果排队到云存储中，稍后将被聚合器取回。

准备运行

我们已经开发好了自己的网格应用程序，准备来运行它了。稍微回顾一下我们刚刚完成的事情：使用一个框架，实现了加载器、聚合器和任务代码。我们只需编写特定于应用程序的代码。

剩下的事情就是要来运行应用程序。对于网格应用程序，你应该总是仔细测试，且首先在本地用少量任务来运行。一旦你对自己的应用程序设计和代码完整性有把握了，就可以移步到云中大规模的执行了。我们将在本系列的下一篇文章(第 3 部分)中来讲述应用程序的运行。

原文链接：http://www.infoq.com/cn/articles/Grid-Azure-2-David-Pallmann

相关内容：

- 基于Azure云计算平台的网格计算，第 1 部分
- Oracle Coherence 3.5 带来增强的WebLogic支持和万亿级数据网格
- 面向服务的虚拟网格简介
- SOA Agents：当网格遇上SOA
- 云计算的虚拟研讨会

推荐文章 | Articles

企业级开发，PHP准备好了吗？

作者 Dionysios G. Synodinos 译者 曹如进

虽然PHP是Web应用开发中最广泛使用的环境，但它还是一度被认为无缘企业级开发。InfoQ组织了一个虚拟座谈小组来讨论语言/平台的演变及PHP在企业环境下的适用性。

参加此次虚拟座谈小组的人有：

- **Zeev Suraski**，Zend Technologies 公司创始人，该公司主要关注 PHP 的进展
- **Rob Nicholson**，高级技术研员，曾为 IBM 编写过程序设计语言运行时
- **Derick Rethans**，PHP 开发小组成员，eZ 组件的项目负责人

InfoQ：企业软件的一个关键元素就是互操作性，它可以让软件与其他平台交换信息。大家都认为 PHP 在这方面表现欠佳，因为它的 WS-*支持相对来说比较新且功能较少，成熟度不高。关于这点你们是怎么考虑的？它会不会有所改变？

Zeev：我觉得相比 WS-*而言，互操作性涉及的要更加多些。事实上，我们只看到了很少的基于 SOAP 的 Web 服务请求，而更多的则来自 于其他标准，这主要是因为部署 SOAP 的过程较为复杂。PHP 极好地支持了互操作，并且为此提供了很多不同的接口（REST，优秀的 XML 支 持 ,SOAP ,以及为 web 服务提供的 ZF 组件等等)。据说 PHP 从 2004 年开始就为 SOAP 提供了非常好的基础支持 ,从 2006 年开始就通过 Axis2 扩展为 WS-*提供了广泛的支持。我 只能说我还从没有碰到过用户抱怨缺乏互操作性的情况，如果真的有，那也一定是赞美吧。

Rob :我觉得这只是部分人的观点。PHP 源于其简单性。它是一门只需必要的复杂度 ,就能“解决 web 问题”的语言。因此 PHP 程序员会更 多的选择 REST 而不是 SOAP。传统的企业软件正逐步向位于中间的 PHP 靠拢。比方说 ,IBM 的许多企业级软件产品在去年都提供了 RESTful 交互支持，包括 Atom 发布协议，这样的话就多了一个选择。在该用 WS-*的地方使用它，而在开发的简单性和速度至关重要时，应使用 REST。我们也饶有兴趣的看到了 PHP 被用 来直接加强企业连通性。IBM 的 Message Broker 可以当作一个“万能转换器”，它能够将一个东

西连接到另外一个东西，而现在它的消息转化流中也提供了对 PHP 计算节点的支持。所以现在是可以 在企业软件内部中使用 PHP 语言简单而又强大的语法和语句的。我们最近为 IBM 的 CISC 事务处理器发布了一个 SupportPac，用以支持 PHP 语 言。CISC 正如软件一样，具有“企业级“的性质。它运行于主机上，可以由一些像银行，政府和医疗保健部门的组织来使用，用以处理一些最重要的可能影响到 日常生活的事务。

Derick：我觉得这里没什么太大问题。PHP 已经为所有的 WS 技术如 SOAP，XML-RPC 和 JSON 提供了支持。

InfoQ：过去的几年里，将脚本语言移植到 JVM 中并以利用它丰富的监控，安全等功能已经成为了一种趋势。这对于 PHP 开发而言并不陌生，因为现实世界中存在好些个运行在 JVM 中的 PHP 应用。制造商们对于提升性能的话题各抒己见。你们是怎么看待这种趋势的？

Zeev：我们在.Net 中也看到了类似的趋势，但是这些脚本语言并没有很远地脱离原始的实现。我想对于 JVM 上的 PHP 也是同样的道理。事实上我们可以看到原生实现的 PHP 相比综合改造后的 PHP 所拥有的性能优势——尤其是对内存的需要以及在现实世界中长期运行的表现。尽管如此，标准实现 最大的优势是在于它所拥有的强大社区支持（包括在代码贡献和使用上），这是其他实现所缺少的一个东西。

Rob：它的一切都令人如此兴奋，我相信它会有很好的未来。在数以千计的已经被实现的语言中，只有少量语言在自然选择过程中幸存下来，原因 是它们特别适合于某个用途。因此开发者改进创新某种语言的实现是一个很自然的事情。如果我们看看 Ruby 社区就会发现，这门语言的成功归因于至少半打以上 的实现，还有这些实现中的共享测试和性能调整，它们帮助明确了语言的规范，而相互间”最快 Ruby“头衔的竞争也功不可没。我想我们正在 PHP 上见证同样 的事情发生。我们已经看到了 PHP 实现间协作所带来的巨大好处，例如在过去两年里由社区产生了大量的全新测试用例以及为改善某些 API 而作的努力，我相信 这种现象在未来会持续下去。我现在正工作于 PHP 在 JVM 上的一个实现，它已经用在了 IBM 的 ProjectZero 孵化器（incubator ）中，WebSphere sMush 产品，以及我前面提到的 CISC PHP SupportPac 和 MessageBroker 计算节点中。我认为对于某些类型的问题，在 JVM 上运行 PHP 会非常有意义。我们看到我们的合作伙伴和 客户正在使用它来耦合现有的基于 Java 的系统，这样做他们可以在轻松重用 Java 库和 API 的同时，享受 PHP 所带来的便捷。

Derick：尽管性能方面“可能”会有所提高，但是可扩展性却始终是个问题。PHP 的整体思想是在无共享架构的情况下轻松实现可扩展性。在 JVM 上跑 PHP 会移除掉它的无共享架构。不幸的是 PHP 社区中只有一个叫做 PHP-on-JVM 的项目在尽可能的贡献着测试用例。

InfoQ：从 PHP 4 到 PHP 5 的升级不是一个简单的迁移过程。关于那些犹豫是否对即将发布的 PHP 6 进行投资的公司，你们想说些什么？

Zeev: 实际上我并不同意关于 4->5 迁移是个非常困难过程的说法。整个过程并没有太多的兼容性破坏问题，而只是相对简单地修补应用程序。事实上想要利用新 的功能，多花一点工作是在所难免的，也是意料之中的。在 6 的版本中我们实际上更多的考虑了兼容性破坏问题——目前这个问题在 6 中要比在 5 中更具实质性。这 就是我们需要花时间去做的事情。

Rob：我认为 PHP5 在今后很长一段时间都会存在。即将发布的 5.3 版本已经尽可能的设计为无痛升级，且增加了原本定在 PHP6.0 中的 几乎全部的功能，只差移除掉一些不用的功能和增加 PHP 6.0 中的 unicode 了。我非常渴望看到 unicode 版本的的 PHP，因为它可以让基于 PHP 的 JVM 具有更加直接的兼容性，之所以更加直接是因为 JVM 原生地采用了 unicode 来表示字符串，但是我怀疑采纳过程在 PHP 5 和 PHP 6 中都将会非常缓慢且持续许多年。

Derick：虽然大家对此总是怀疑，但是我们会努力减少这些问题，通过引入向前兼容的功能来转移到 PHP 6。如果大家能够向我们反馈一下自己在当前开发版本中碰到的问题的话，那么可以帮助我们将迁移过程变得更加简单。

InfoQ：在所有的建立的语言中，社区中的人们都推动增加了许多高级的功能。而另一方面 PHP 一直被认为易学的功能较少。你们认为这种情况需要改变吗？

Zeevv：我绝对不认为它应该改变，因为它是 PHP 成功的一个关键因素。希伯来语中有句谚语大致这么说“给的越多，拿的越多”，我坚信这 句话对于 PHP 是适用的，至少在语言结构与语法上如此。通过使用扩展和框架，PHP 可以无止境的扩展，在我来看，这些扩展和框架正是 PHP 最佳和最有趣的“最后前 沿”。我觉得完全使用 PHP 的大型复杂网站（Facebook，Yahoo，Flickr），完全基于 PHP 的复杂现有应用 （SugarCRM，OpenPro，CMS's），以及公司网站或内部系统依赖于 PHP 的企业证明了这样一个事实：PHP 的功能集已经成熟，并且我们应 该朝着这个方向走下去。

Rob：在我们着手为 IBM 的脚本产品 WebSphere sSmash 选择脚本语言时，就因为 PHP 如此广泛的使用面而特别选择了它。我们希望能够让数以百万的 PHP 程序员们能与企业或者企业软件紧密联系在一 起，并且我们希望支持一种能够让新人程序员快速上手的语言。PHP 的强大在于它的简单性。一门语言如果不想灭亡的话就一定需要不断的演变。如果 PHP 5 没有支持面向对象编程的话，肯定会丧失很多吸引力。伴随着 PHP 5.3 的发布，PHP 肯定可以在这些新的特性上潜在的增加其复杂性。我想未来更多的工作是去了解怎样使用它们和在此之上形成的语句。鉴于新版本被采纳的滞 后性，因此在大部分主流应用程序转移到使用 5.3

功能之前，还需要等上若干年，我想在这段时间里 PHP 程序员将会用大量实例来掌握这些新功能，并将它们用 在简化常见的编程任务上。

Derick：不，它不需要改变，这两类开发人员都存在。增加新功能并不一定需要提高入门的门槛。

InfoQ：PHP 作为一门语言，在这些年里一直追随优秀的范型而演变，并从一个简单的预处理器演变成了一个强大的 OO 语言。随着函数式编程风格崭露头角，你们觉得这种这种范式是否会走进未来 PHP 的世界？

Zeev：不会。PHP 仍然支持过程式开发，并且不太可能会消失；我们早在启动 PHP（PHP 3）中就增加了 OO 的支持，虽然它现在跨越到了 PHP 5 中。lambda 大概是最接近函数式范式的东西了，而这正是我们需要完成的。这也映衬了我前面回答的一个问题——我们不想要一个一劳永逸的语言，只是想 要一个能够完成工作的简单语言。

Rob：这在一定程度上已经发生了。PHP 5.3 中闭包的概念就是来源于函数式编程的世界里。PHP 社区混杂了大量经过"经典训练"的计算机科学专业人员以及一些业余自我训练的程序员。看到这个多 样的社区中闭包的诞生和常用语句的演化其实是一件很有趣的事。我相信我们最终会完成一套被广泛接受的模式和语句，它可以很优雅的解决 web 开发中的常见问题，而程序员在使用时都不会想到其实这一切都源于函数式编程。

Derick：我不确定，我认为它不会特别合适。但是如果它对 PHP 应用程序有意义的话，也许可以找到进入 PHP 的出路。PHP 在集成其他语言中有趣和有用的理念方面一直做得很优秀。

你是怎么认为的呢，选择**PHP**是企业的明智之举吗？

原文链接：http://www.infoq.com/cn/articles/enterprise-php

相关内容：

- HTTPS连接最初的若干毫秒
- PHP与微软云计算
- 微软：PHP在IIS 7 上雄起
- Neal Ford谈编程语言和平台
- 使用 Irony 编写新的.NET 语言

推荐文章 | Articles

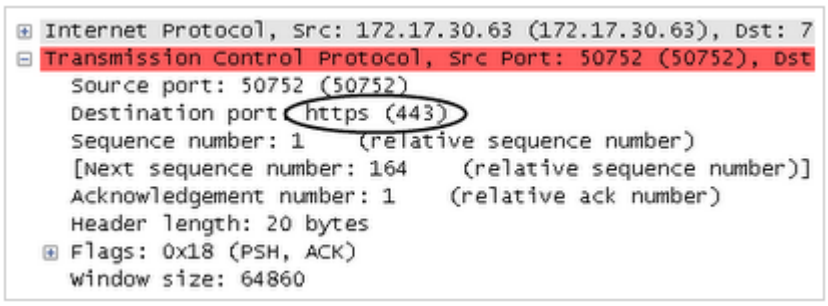
HTTPS连接最初的若干毫秒

作者 Jeff Moser 译者 马国耀

当你在浏览了一个网站上面的商品之后，点击“继续并结帐”时会发生什么？本文即将对（浏览器）与 Amazon 建立安全连接的整个过程中最初的若干毫秒进行分析。当你点击继续按钮时一个新的页面将被加载：

在短暂的 220 毫秒内，发生了很多有趣的事情，Firefox修改了地址栏的颜色，并在其右下角放置了一个锁状的图标。在我最喜爱的网络工具Wireshark以及略微修改的Firefox调试版的帮助下，我们可以对正在发生的事情看个究竟。

根据RFC 2818协议的规定，Firefox明白“https”意味着它应该连接Amazon.com的443 端口：

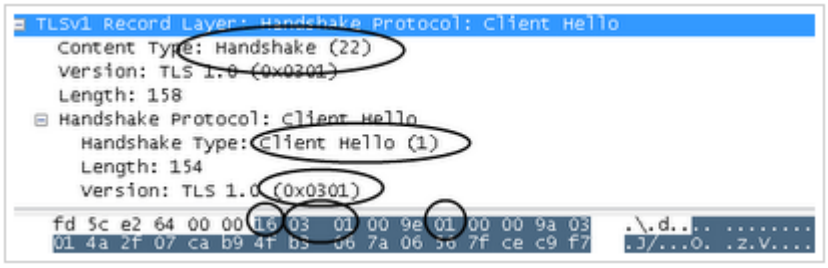


大多数人将HTTPS和SSL(Secure Sockets Layer)联系起来，SSL是Netscape公司在 90 年代中期发明的。随着时间的推移这种说法就渐渐变得不准确了。由于Netscape失去了市场份额，它将SSL的维护工作移交给因特网工程任务组（IETF）。第一个后Netscape版本被重新命名为安全

传输层协议（TLS），TLS1.0 是在 1999 年 1 月份发布的。由于TLS诞生都 10 年了， 所以真正的“SSL”传输其实是几乎见不到。

Client Hello

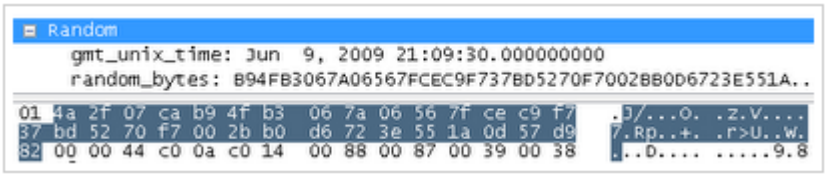
TLS将所有的网络传输打包成不同的“记录”类型。我们看到从浏览器出来的第一个字节是一个十六进制（hex）字节 0x16=22，这说明它是一个“握手”记录：



后面的两个字节是 0x0301，意味着它的版本是 3.1，事实上 TLS1.0 就是 SSL3.1。

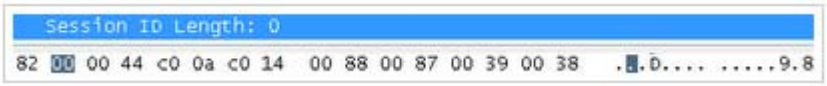
“握手”记录被分解成若干消息。第一个就是“Client Hello”消息（0x01）。这里面有很重要的几点：

1、**随机数：**



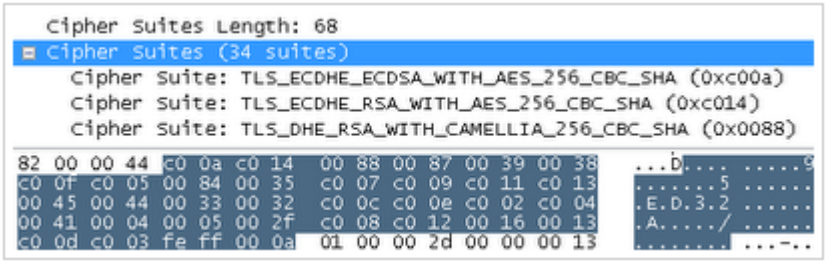
前面的四个字节是当前的协调世界时（Coordinated Universal Time，UTC），它的格式是Unix时间戳 ,也就是从 1970 年 1 月 1 日起到此刻所经历的秒数。在本例中的该数字是 0x4a2f07ca。跟随其后是 28 字节的随机数，它将在后面使用。

2、会话标识（session id）：



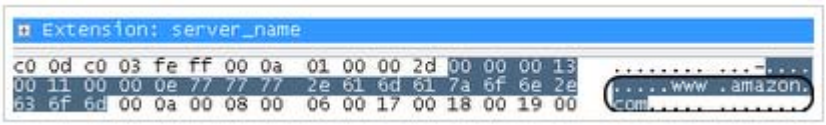
在这里它是空值或者是 null。如果在几秒前该浏览器曾连接过 Amazon.com，它就可能继续使用前面的会话，而不需要重新执行整个“握手”过程。

3、密码套件：



它是浏览器所支持的密码算法的一个列表。最上面的是一个很强大的组合“TLS_ECDHE_ECDSA_与_AES_256_CBC_SHA，下面还有该浏览器支持的另外 33 个选择。如果现在不明白它们的含义也不必担心，后面你将会发现Amazon并没有选择最上面的强大组合。

4、server_name extension：

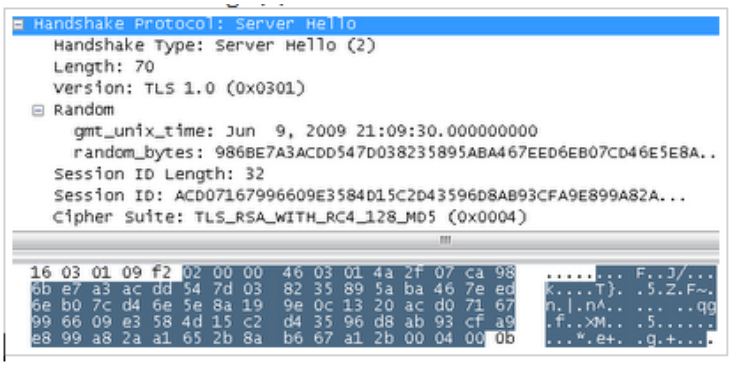


通过它告诉Amazon.com，浏览器正要连接https://www.amazon.com/。这样做非常方便，因为TLS“握手”发生在所有的HTTP传输之前 。HTTP协议中有一个“Host”头，这就允许了Internet托管公司出于成本的考虑将上百个网站绑定在同一IP 地址上。传统意义上SSL要求每一个地址有一个不同的IP，而这个扩展就使得服务器能够正确响应浏览器所请求的（服务器）证书。最后注意一点，对于 IPv4 的地址这一扩展的有效期大概能再多一周左右。

Server Hello

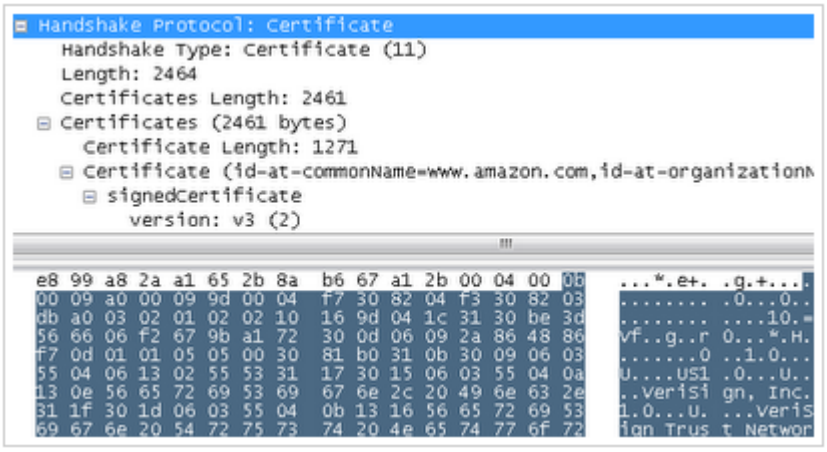
对应于 Client Hello，Amazon.com 也返回一个“握手”记录，它是两个庞大的数据包(2,551 字节)。该记录的版本号是 0x0301，也就意味这 Amazon 支持我们使用的 TLS1.0 版本的请求。这个记录分成三个子消息，他们包含着一些有趣的信息。

1、"Server Hello" 消息（2）：

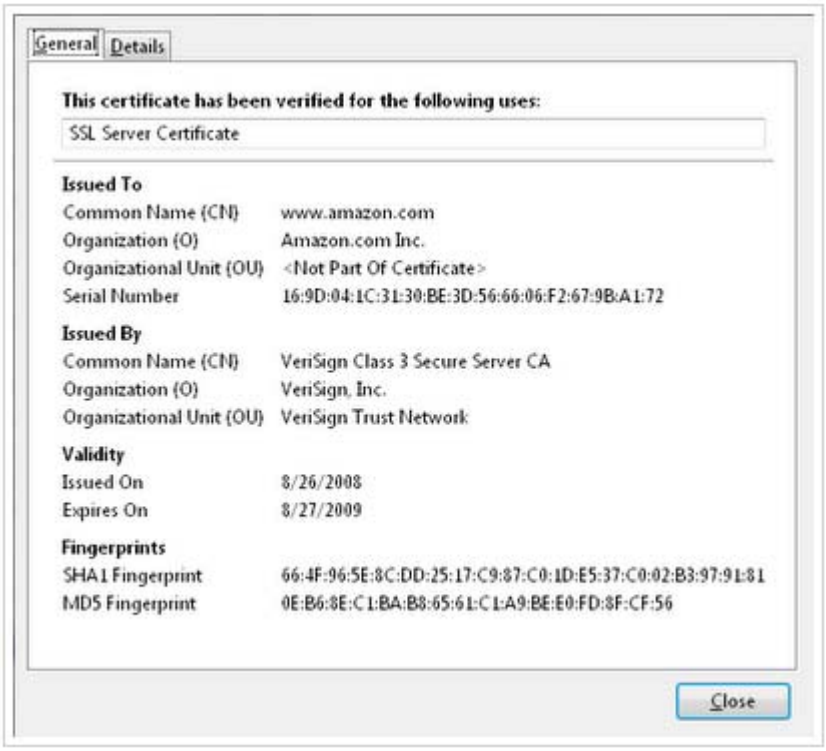


- 我们得到服务器返回的四字节的 Unix 时间戳以及 28 字节的随机数，它也将在后面使用（译注：注 Clien Hello 是发了一个随机数，这两个随机数将用于产生最后的对称密钥）。
- 一个 32 字节的会话标识，有了它，随后重连服务器就不需要再执行一个完整的握手过程了。
- 从我们提供的 34 个可选的密码套件中，Amazon选择的套件“TLS_RSA_WITH_RC4_128_MD5”（0x0004）。也就是说它将使用“RSA”公钥算法来验证证书以及交换密钥，用RC4加密算法对数据进行加密，使用MD5哈 希算法来校验消息内容。后面会对它们做详细介绍。我个人认为Amazon选择这个密码套件的原因是出于自私的考虑（译注：这里说“自私”的意思 是，Amazon出于性能的考虑，降低了安全的强度）。这个套件是这 34 个套件中消耗CPU最少的一个，因此Amazon选择它就是为了节省一些CPU来 处理更多的用户连接。还有一个不大可能的原因是他们要特别地向Ron Rivest致敬，这个套件中所用到的三个算法都是他发明的。

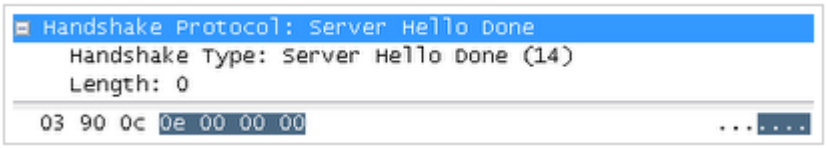
2、证书消息（11）：



这个巨大的消息占据了 2464 个字节，它是一个证书，客户端用它对 Amazon 进行认证。它并不是什么花哨的玩意，通过浏览器就可以看到它的大部分内容：



3、“Server Hello 结束”消息（14）：



这是一个 0 字节的消息，它告诉客户端“hello”过程已经完成，也就意味着服务端将不验证客户端的证书。

校验证书

浏览器必须要确定它是否要信任Amazon.com，在这里是通过证书判断的。它检查Amazon的证书并且检查现在的时间是否在证书的有效期之内，该有效期规定的是“不能早于”2008 年 8 月 26 日且“不能晚于”2009 年 8 月 27 日。此外，它还要确保该证书的公钥已经被授权在进行密钥交换的过程中使用。

为什么我们应该相信这个证书呢？

在证书上附有一份“签名”，这个签名事实上是一个big-endian（译注：Big-Endian和Little-Endian是一种二进位资料储存或传输的格式，Big-Endian的最高位字节在最前头，而Little-Endian的最高位字节在最后面）格式的长整型数：

任何人都有可能发给我们这些字节，为什么我们要相信这个签名呢？为了回答这个问题，有必要先到这里mathemagic land去速成一些相关知识：

穿插：简短的，不那么可怕的RSA向导

有时候人们想知道数学和编程有何联系。证书就是应用数学的一个非常实际的例子。Amazon的证书告诉我们应该用RSA算法来校验其签名。RSA是MIT的教授Ron Rivest，Adi Shamir和Len Adleman在 70 年代创造出来的，他们三人创造了一个巧妙的方法将2000 多年来; 数学发展的思想汇合起来形成了一个漂亮而简单的算法：

首先选择两个很大的质数“p”和“q”，并对他们求积得到“n=p*q”。接下来，取一个较小的“e”作为指数，它用作“加密指数”，而对e的进行特殊的逆反函数计算所得到的“d”作为“解密指数”。然后将“n”和“e”公开出去，而对“d”要保密，对于“p”和“q”你可以把它们扔掉，也可以像“d”一样保密起来。真正重要的要记住“d”和“e”是相互的逆反。

现在，如果你有一些消息，那么你只需要将该消息的字节翻译成一个数“M”，若要对这个消息进行“加密”形成“密文”的话，你就这么计算：

```
C ≡ Me (mod n)
```

它意思是先求“M”的“e”次方，然后对它应用模数“n”求余。举个例子，`11AM +3 hours ≡ 2 (PM)(mod 12 hours)`。接收者知道（解密用的）“d”，而“d”可以对已加密的消息进行反转并还原消息：

```
Cd ≡ (Me)d ≡ Me*d ≡ M1 ≡ M (mod n)
```

另一件有意思的事情是拿着“d”的人可以对一个文档进行签名，其做法是文档“M”求“d”次幂然后应用模数“n”求余数“S”：

```
Md ≡ S (mod n)
```

这种做法成立的原因是“签名者”将“S”，“M”，“e”以及“n”公开出去，任何人都可以通过这样一个简单的计算来验证“S”是否由“签名者”所签（译 注：因为每个“e”都是某个签名者所独有的，而“e”和“d”是成对出现的，所以签名者用自己的“d”签名，只能拿这个“d”所对应的“e”才能还原消 息，这就到了签名的作用）：

$$Se \equiv (Md)e \equiv Md*e \equiv Me*d \equiv M1 \equiv M \pmod{n}$$

像RSA这样的公钥密码算法经常被称之为“非对称”算法，因为加密的密钥（本例子中的“e”）和解密的密钥是不相同的（本例子中是“d”）。对所有的数应用“mod n”的目的是使攻击者不可能使用简单的技术（如过去我们使用的对数）破解它。RSA的神奇之处是你可以很快计算/加密$C \equiv M^e \pmod{n}$，而在不知道“d”的情况下计算/解密$C^d \equiv M \pmod{n}$是非常困难的。如前面所说，“d”来自于对“n”的因子分解，而“n”来自于“p”和“q”，求解“d”是一件复杂的事情。

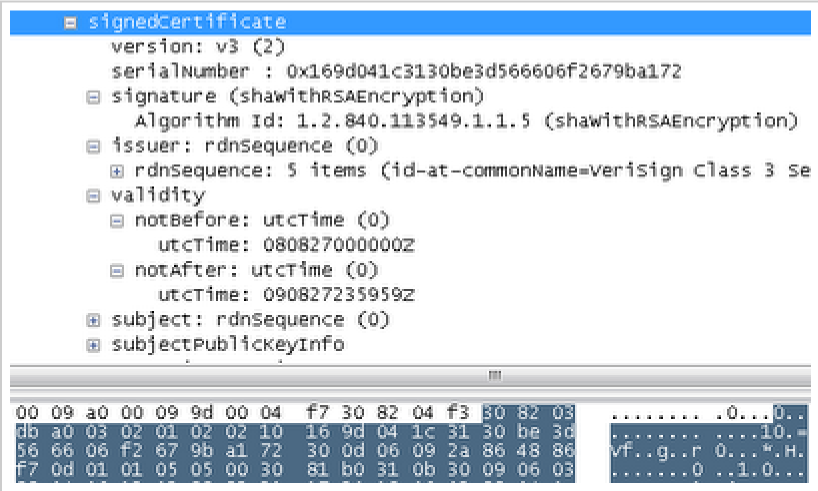
签名验证

在现实世界里使用RSA时应该谨记的重要的一点是，前文提到的所有数字都应是很大的数，只有这样才能保证，即使用目前最好的算法也 很难攻破上述算法。到底多大才算很大的数呢？Amazon.com的证书是由“VeriSign Class 3 Secure Server CA.” 签名的。从这个证书中我们可以看出，VeriSign所使用的模数“n”是一个 2048 比特长的数字，表示成 10 进制数的长度是 617 位：

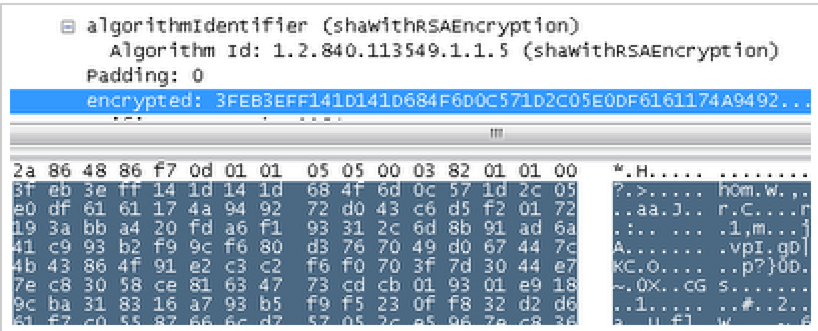
```
1890572922 9464742433 9498401781 6528521078 8629616064 3051642608
4317020197 7241822595 6075980039 8371048211 4887504542 4200635317
0422636532 2091550579 0341204005 1169453804 7325464426 0479594122
4167270607 6731441028 3698615569 9947933786 3789783838 5829991518
1037601365 0218058341 7944190228 0926880299 3425241541 4300090021
1055372661 2125414429 9349272172 5333752665 6605550620 5558450610
3253786958 8361121949 2417723618 5199653627 5260212221 0847786057
9342235500 9443918198 9038906234 1550747726 8041766919 1500918876
1961879460 3091993360 6376719337 6644159792 1249204891 7079005527
7689341573 9395596650 5484628101 0469658502 1566385762 0175231997
6268718746 7514321
```

（如果你要试图从这个“n”中找到“p”和“q”那就祝你好运，假如你能找到他们，那么你能产生像真的一样的 VeriSign 证书了）

VeriSign的“e”的值是 2^16 + 1 = 65537。当然，他们对相应的“d”保密，也许被放在一个安全的硬件设备上，由视网膜扫描机和武装部队保护着。在签名之前，VeriSign要通过现实世界的“握手”来验证Amazon.com声明的证书内容，包括审查他们的很多商务文档。一旦VeriSign对这些文档满意了，他们使用SHA-1哈希算法对包含所有声明的证书进行计算获得一个哈希值。在Wireshark中看到整个证书的样子如下图“signedCertificate”部分所示：



上面的说法（整个证书如同 signedCertificate 所示）说有点用词不当，因为证书的实际含义是签名者将要进行签名的部分，而不是已经包含了签名信息的字节。



实际的签名，也就是“S”，是 Wireshark 包中简单地称为“encrypted”的部分（见上图）。如果我们求“S”的“e”次幂（“e”是 VeriSign 公开的），然后在对该结果取模“n”并得到余数，那么我们得到的“已解密”十六进制签名是：

0001FFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFF00302130
0906052B0E03021A 05000414C19F8786 871775C60EFE0542 E4C2167C830539DB

基于PKCS #1 v1.5 标准， 最开始的字节是“00”，它“保证了加密块被转换成整数之后，大小不会超过模数”。第二个字节“01”说明这是一个私钥操作（如，它是一个签名）。后面是 一大堆“FF”，用他们填充这个结果，使之足够大。填充字节以“00”结束，后面跟的是"30 21 30 09

06 05 2B 0E 03 02 1A 05 00 04 14"，PKCS #1 v2.就是这样来指定SHA-1哈希算法。最后的 20 个字节是对“signedCertificate”部分的SHA-1 哈希摘要。

因为解密的值有着正确的格式并且最后的字节和我们独立计算得到的哈希值相同 ,我们可以认为这就是由拥有“VeriSign Class 3 Secure Server CA”私钥的人所签名的，而我们隐含地信任只有VeriSign知道这个私钥“d”。

但我们为什么要信任它呢？因为在信任链上已经没有更高级别了。

最上层的“VeriSign Class 3 Public Primary Certification Authority”是自签名的。这个证书自从网络安全服务（Network Security ServicesNSS）库中certdata.txt的 1.4 版本之后就已经被嵌入Mozilla的产品中作为一个隐含的受信证书。它是由NetScape的Robert Relyea在 2000 年 9 月 6 日登记进去的，当时他还留下以下评注：

> 用这个框架来收集其他网络安全服务，它包括这一个“实时的” certdata.txt ，里面存放了那些我们已经获得许可权可以推向开源的证书（当我们从所有者那里获得许可权之后，会有更多的证书加入这个文档中）

由于该证书的有效期范围是从 1996 年 1 月 28 日到 2028 年 8 月 1 日，所以这个决定是意义深远的。

Ken Thompson在他的“Reflection on Trusting Trust”中解释的非常好 ,你最终不得不隐式地信任某人。在这个问题上别无选择。本例中我们隐含地信任Robert Relyea做了个好的选择，我们也希望Mozilla的内建证书策略对其他的内建证书也是合理的。

这里要记住的一点是所有这些证书和签名都简单地用来形成一个信任链。在公共网络中，在你还未上任何网站之前，Firefox 就已经隐含地信任了 VeriSign 的根证书。在一个公司内，你可以创建你自己的根证书 CA，并将它安装在每个人的电脑上。

或者，你还可以付钱给VeriSign这样的公司并把所有的证书信任链的工作交给它。证书用来通过可信的第三方（这里是VeriSign）建立信任 关系。如果你可以安全地和别人共享密钥，

比如“在某人耳边轻声告诉他一长串密码”，那么你就可以使用预共享密钥（PSK，Pre-shared Kay）认证来建立信任关系。TLS对此也有扩展，比如TLS-PSK，以及我本人所喜欢的TLS的安全远程密码扩展。然而，这些扩展没能被广泛地使用和支持，所以他们通常是不切实际的。此外，这些方法增加了负担，它们要求必须通过安全的方式交换密钥，而这比通过TLS来建立信任连接的负担更重（要不然我们何不都这么用呢？）。

我们要做的最后的检查是验证证书上的主机名是否是我们期待访问的。 Nelson Bolyard 在 SSL_AuthCertificate 函数中的注释是这样解释的：

> /* 证书是没问题的。这是 SSL 连接的客户端。
>
> *现在要检查证书中的名字和所期待的主机名是否一致。
>
> *注意：这是我们防范中间人（Man-In-The-Middle）攻击的唯一途经*/

这个检查防范了中间人攻击，因为我们隐含地相信拥有证书的人不会干坏事，例如，只有它真是Amazon.com，它才会对一个证书签名说它是Amazon.com。如果一个攻击者能够通过类似 DNS缓存破坏的技术修改我们的DNS服务器，你可能被蒙蔽了，以为自己正访问正确的网站（如Amazon.com），因为你的浏览器的地址栏看起来就是这个网站。这最后的检查就隐含地信任证书的授权机构可以防止坏事的发生。

Pre-Master Secret

现在我们已经验证了Amazon.com的那些声明，并且知道其公开的加密指数“e”和模数“n”。任何在网络上侦听的人当然也知道这两个数（这点 很明显，因为我们是通过Wireshark获取它们的，别人也可以）。现在我们要创建一个攻击者或偷听者不能辨别的随机密钥，这件事做起来不像听起来那么 简单。在 1996 年，研究者发现Netscape Navigator1.1 仅使用三个数据源来作为他们的伪随机数生产器的种子（PRNG），它们是：当前时间，进程号和父进程号。研究显示，这些“随机”源并不那么随机，而且相对比较容易被猜出来。

由于其他的所有东西都是从这三个“随机”源产的，在 1996 年当时的机器上可能只需 25 秒中就可以“攻破”SSL“安全”。如果你还不相信寻找随机数是件困难的事，只要问问Debian OpenSSL维护者就知道了。如果随机数搞乱了，那么基于它之上的所有安全都是可疑的。

在Windows上，用于加解密的随机数是通过调用CryptGenRandom函数对来自 125 个源的抽样数据求hash值产生的。Firefox将该函数产生的随机数和它自己的函数产生的比特码一起作为伪随机数生成器的种子。

虽然这个 48 字节的“pre-master secret”随机数并不直接使用，但是保证其私密性是非常重要的，因为很多东西都是由它而来的。毫无疑问，Firefox让我们很难看到这个数。我不得不编译了一个调试版本并设置SSLDEBUGFILE和SSLTRACE两个环境变量才能看到它。

在本次会话中，SSLDEBUGFILE 中显示的这个 pre-master secret 是这样的：

```
4456: SSL[131491792]: Pre-Master Secret [Len: 48]
03 01 bb 7b 08 98 a7 49 de e8 e9 b8 91 52 ec 81 ...{...I.....R..
4c c2 39 7b f6 ba 1c 0a b1 95 50 29 be 02 ad e6 L.9{......P)....
ad 6e 11 3f 20 c4 66 f0 64 22 57 7e e1 06 7a 3b .n.? .f.d"W~..z;
```

需要指出的是这并非全是随机数， 按约定，最前面的两个字节是TLS版本（03 01）。

密钥交换

现在要把这个密钥传给 Amazon.com。由于 Amazon 选择的是“TLS_RSA_WITH_RC4_128_MD5”，我们将使用 RSA 算 法来做这件事。输入消息可以是正好 48 字节的 pre-master secret，但是公钥密码学标准（PKCS）#1，RFC1.5 要求用一些随机数来填充这个数使其长度等于模数的长度（1024 比特，或者 128 字节）。这样做就使得攻击者更难找出我们的 pre-master secret 了，另外，万一有人做傻事，比如重用相同的密钥，它还为我们提供最后一层保护。通过这样一填充，即使我们重用密钥，偷听者在网络上两次看到的 数也是不一样的。

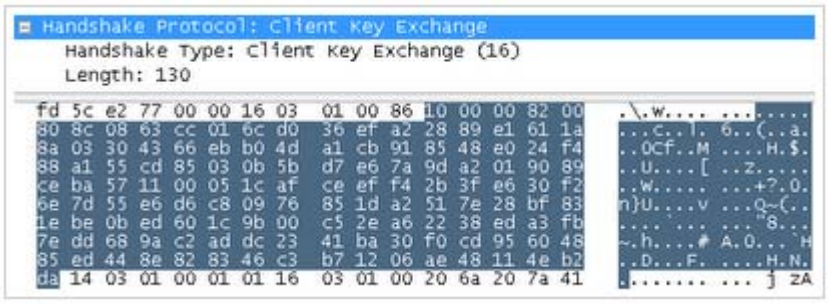
又，因为Firefox让我们很难看到这些随机数，我不得不在填充函数中加入一些调试语句才能看到发生的事情：

```
wrapperHandle = fopen("plaintextpadding.txt", "a");
fprintf(wrapperHandle, "PLAINTEXT = ");
for(i = 0; i < modulusLen; i++)
{
    fprintf(wrapperHandle, "%02X ", block[i]);
}
fprintf(wrapperHandle, "\r\n");
fclose(wrapperHandle);
```

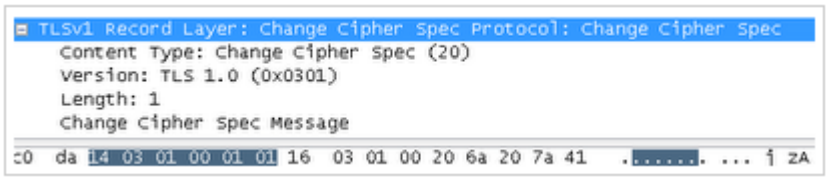
在这次会话中，填充后的消息如下所示：

```
00 02 12 A3 EA B1 65 D6 81 6C 13 14 13 62 10 53 23 B3 96 85 FF 24 FA
CC 46 11 21 24 A4 81 EA 30 63 95 D4 DC BF 9C CC D0 2E DD 5A A6 41 6A
4E 82 65 7D 70 7D 50 09 17 CD 10 55 97 B9 C1 A1 84 F2 A9 AB EA 7D F4
CC 54 E4 64 6E 3A E5 91 A0 06 00 03 01 BB 7B 08 98 A7 49 DE E8 E9 B8
91 52 EC 81 4C C2 39 7B F6 BA 1C 0A B1 95 50 29 BE 02 AD E6 AD 6E 11
3F 20 C4 66 F0 64 22 57 7E E1 06 7A 3B
```

Firefox拿这个数进行计算“$C \equiv M^e \pmod n$”，得到的数是我们在客户密钥交换记录中所看到的



最后Firefox发出最后一条没加密的消息，一个“Change Cipher Spec”记录：



Firefox 使用这种方式通知 Amazon，它将使用前面协商好的密钥来加密下一条消息。

生成Master Secret

如果前面所有事情都正确完成，则双方（且只有这双方，客户端和服务端）现已知道这个48 字节的pre-master secret。从Amazon的角度来看这里有一个很小的信任问题：pre- master secret只包含客户端生成的比特位，他们没有考虑到服务端或者我们前面所提到的那些随机数（译注：Client hello和Server hello过程中产生的随机数）。我们将通过计算“master secret”来解决这个问题。根据规约，应该这么计算：

```
master_secret = PRF(pre_master_secret, "master secret",
ClientHello.random + ServerHello.random)
```

“pre_master_secret”是客户端之前发送的；“master secret”用的是一个字符串的ASCII值（例如：“6d 61 73 74 65 72 ...”）。然后我们将本文最开始看到的ClientHello和ServerHello发送的随机数拼接起来。

PRF是一个“伪随机数函数”，这个函数很聪明，在规约中也有定义。它使用基于哈希的消息验证码（HMAC）的MD5和SHA-1两种哈希函数将密钥，ASCII字符以及我们给的种子结合起

来。对每个哈希函数发送一半的输入。说它聪明的原因是即使面对MD5和SHA-1 的弱点，它的防攻击能力还很强。这个过程可以自我反馈并不停地循环，而且我们要多少字节就能生成多少。

依照这个过程，我们获得以下 48 字节的“master secret”：

```
4C AF 20 30 8F 4C AA C5 66 4A 02 90 F2 AC 10 00 39 DB 1D E0 1F CB E0
E0 9D D7 E6 BE 62 A4 6C 18 06 AD 79 21 DB 82 1D 53 84 DB 35 A7 1F C1
01 19
```

多个密钥的生成

现在双方都有了“master secrets”，规约描述了我们如何生成会话所需的所有的密钥，我们需要使用PRF函数来创建一个“key block”，然后从这个块中提取所需的密钥：

```
key_block = PRF(SecurityParameters.master_secret, "key expansion",
SecurityParameters.server_random +
SecurityParameters.client_random);
```

“key_block”被用来提取以下密钥：

```
client_write_MAC_secret[SecurityParameters.hash_size]
server_write_MAC_secret[SecurityParameters.hash_size]
client_write_key[SecurityParameters.key_material_length]
server_write_key[SecurityParameters.key_material_length]
client_write_IV[SecurityParameters.IV_size]
server_write_IV[SecurityParameters.IV_size]
```

由于这里使用的是序列密码而非分组密码（如高级加密标准AES），所以不需要初始向量（Initialization Vectors IV），而只是双方各需要一个 16 字节（128 比特）的消息验证码（Message Authentication Code MAC），这是因为指定的MD5 哈希摘要大小是 16 字节。另外，RC4 加密算法使用的 16 字节的密码也是双方都需要的。最后，我们需要key block中的 2*16 + 2*16 = 64 个字节：

运行 PRF，我们得到：

```
client_write_MAC_secret = 80 B8 F6 09 51 74 EA DB 29 28 EF 6F 9A B8
81 B0
server_write_MAC_secret = 67 7C 96 7B 70 C5 BC 62 9D 1D 1F 4A A6 79
81 61
client_write_key = 32 13 2C DD 1B 39 36 40 84 4A DE E5 6C 52 46 72
server_write_key = 58 36 C4 0D 8C 7C 74 DA 6D B7 34 0A 91 B6 8F A7
```

准备加密！

客户端发出的最后一条“握手”消息是“Finished message”。 这个消息非常巧妙，它不仅能证明没有人篡改了握手过程，还能证明我们确实知道密钥。客户端将所有的“握手”消息放入一个 “handshake_messages”缓存区，然后使用伪随机函数，“client finished”字符串以及MD5 和 SHA-1 哈希运算后“handshake_messages”计算出 12 字节的“verify_data”：

```
verify_data = PRF(master_secret, "client finished",
MD5(handshake_messages) + SHA-1(handshake_messages)) [12]
```

我们在这个结果的前面加上一个记录头字节“0x14”指明“完成”和长度字节“00 00 0c”指明我们将发送 12 字节的“verify data”。然后，像所有接下来的加密消息一样， 我们要确保解密后的内容没有被篡改。因为选择使用的密码套件是 TLS_RSA_WITH_RC4_128_MD5，所以我们将使用 MD5 哈希函数。

有些人一听到MD5 就感到恐慌，因为它存在一些弱点，我原先也很不提倡使用它。然而，TLS很聪明，他并不直接使用MD5，而使用它的HMAC版本。TLS是这样使用MDB进行计算的：

```
HMAC_MD5(Key, m) = MD5((Key ⊕ opad) ++ MD5((Key ⊕ ipad) ++ m)
```

（ ⊕指的是异或（XOR），++指的是拼接，“opad”是一串“5c 5c ... 5c”字节，“ipad”是另一串“36 36 ... 36”）。

这里我们对以下内容进行计算：

```
HMAC_MD5(client_write_MAC_secret, seq_num + TLSCompressed.type +
TLSCompressed.version + TLSCompressed.length +
TLSCompressed.fragment));
```

也许你已经看到，我们加入了一个序列号（“seq_num”）和明文消息（这里被称为 TLSCompressed）的一些其他属性。序列号可以迷惑 攻击者，他可能会在中途把一个先前加密的消息插入。如果他这么干，则序列号一定和我们所期待的不一样，这就保护了我们不受攻击者们扔消息的攻击。

剩下就是加密消息了。

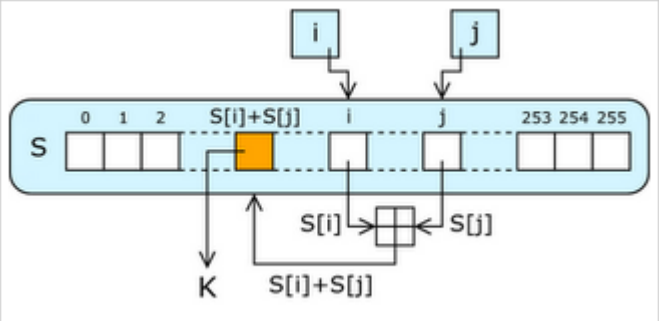
RC4 加密算法

从前文已知双方协商的密码套件是TLS_RSA_WITH_RC4_128_MD5。这就意味着我们将使用 Ron's Code #4 (RC4)对传输信息进行加密。 Ron Rivest开发了基于 256 字节的密钥生成随机数

的RC4 加密算法。这个算法非常简单，以至于几分钟内你就可以记住它。

RC4 从创建一个 256 字节的数组“S”开始，并从 0 到 255 对其进行填充。然后从“S”的第 0 位开始循环，将“S”和密钥中的字节进行混合，这样做是为了创建用于产生“随机”数的状态机。为了生成随机数，我们将“S”数组进行洗牌（译注：参考百度百科RC4）

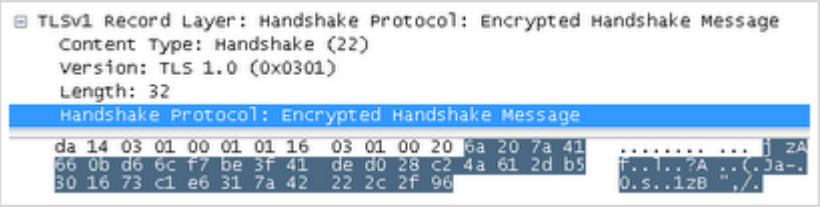
图形化描述是这样的：



对一个字节进行加密，我们对伪随机字节和要加密的字节进行异或运 算。记住将一个比特和 1 进行异或的话是使这个比特反转（译注：0^1=1, 1^1=0）。因为前面产生的是随机数，所以大约会有一半的比特码会被反转，这种随机的比特反转在加密数据时非常有效。你已经看到，这并不复杂，而且运行 起来很快。我想这也许就是Amazon用它的原因吧。

回想一下，我们有“client_write_key”和“server_write_key”。这意味这我们需要创建两个 RC4 实例，一个用于加密浏览器发送的消息，另一个用于解密服务器返回的消息。

“client_write”最前面的随机字节是“E 20 7A 4D FE FB 78 A7 33 ...”，如果我们用这些字节和未加密的消息头（经查证，该消息的字节为“4 00 00 0C 98 F0 AE CB C4 ...”）进行异或运算的话，我们将得到在 Wireshark 中看到的已加密的部分：



服务器做的事情几乎一样。它发出一条“Change Cipher Spec”消息，然后发出的“Finished Message”消息，这条消息包括所有的“握手”消息，以及解密的客户端发过来的“Finished Message”，这也向客户端表明了服务端可以正确解密客户端发过来的消息。

欢迎回到应用层！

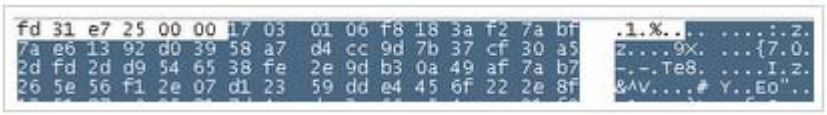
现在，220 毫秒过去了，我们最终为应用层准备好了，现在我们可以发送通过 TLS 层使用 RC4 的写实例加密过的普通 HTTP 消息，也可以解密服务端 RC4 写实例发过来的消息。此外，TLS 层还会通过计算消息内容的 HMAC_MD5 哈希值来校验每一条消息是否被篡改。

至此，“握手”过程已经完成，现在 TLS 记录的 content type 变成 23（0x17）。加密的数据流以“17 03 01”开始，这几个字节表明了记录类型和 TLS 版本，后面是包含了 HMAC 哈希的加密数据。

对如下明文的加密：

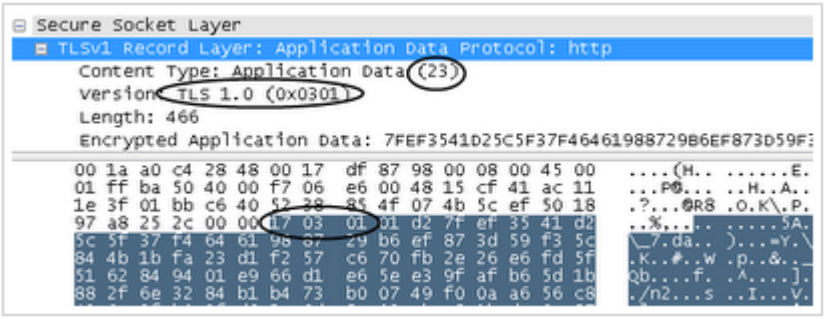
```
GET /gp/cart/view.html/ref=pd_luc_mri HTTP/1.1
Host: www.amazon.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
rv:1.9.0.10) Gecko/2009060911 Minefield/3.0.10 (.NET CLR 3.5.30729)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

将得到下面这些字节：



剩下的唯一有趣的事是每个消息的序列号是依次递增的，现在是 1（下一个消息将会是 2，以此类推）。

服务端做的事情是一样的，不过它使用“server_write_key”这个密钥。我们看到它的响应消息，包括泄露秘密的应用数据头。



这个消息解密后得到：

```
HTTP/1.1 200 OK
Date: Wed, 10 Jun 2009 01:09:30 GMT
Server: Server
...
Cneonction: close
Transfer-Encoding: chunked
```

这是一个普通的HTTP返回消息，它包含一个非描述性“Server: Server”HTTP头和一个来自Amazon负载均衡器的“Cneonction: close”头，显然拼写有误。

TLS就在应用层之下。HTTP服务器上的软件可以就像发送非加密数据一样工作，而唯一的区别是它们（指软件）将数据写到一个库，由这个库统一做加密。OpenSSL是TLS的一个非常流行的开源库。

在双方发送和接受加密数据的过程中，链接一直保持活动状态，直到任何一端发出一条“closure alert”消息并接着关闭连接。如果在断开后的很短时间内重连，则可以重用前面协商好的密钥（前提是服务器还缓存着他们），这样就不需要公钥操作，否则，就需要重新执行一次完全的握手过程。

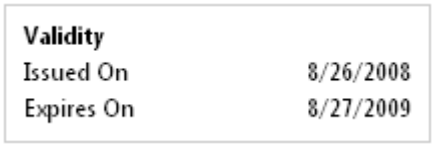
应用数据消息可以是任何信息，这点很重要。“HTTPS”特别的唯一原因是因为Web太流行了，还有其他基于TCP/IP的协议运行在TLS之上。例如，TLS也用于FTPS和SMTP的安全扩展。使用TLS肯定比你自己发明解决方案要好，另外，使用经得起仔细的安全分析的协议，你能从中获益。

……顺利完工！

TLS RFC中还包含很多我们这里遗漏的 内容，它是一份可读性非常好的规范。我们覆盖的内容是对Firefox和Amazon之间的 220 毫秒的舞蹈进行观察所看到的一条路径。这个过程很大程度 上受Amazon在ServerHello消息中所选择的TLS_RSA_WITH_RC4_128_MD5 密码套件的影

响，这是一个合理的选择，因为它 对速度的倾向比对安全的倾向更多一点。

如我们所看到的，如果有人能将 Amazon 的模数“n”分解成“p”,“q”，那么他就可以有效解密所有的“安全”流通，直到 Amazon 换掉它的证书。所以 Amazon 通过每年替换一次证书的方式来平衡这方面的担心。



其中有一个密码套件是“TLS_DHE_RSA_WITH_AES_256_CBC_SHA”，它使用Diffie-Hellman密钥交换算法，这个算法具有良好的 “正向安全”的特点。也就意味这如果某人破解了密码交换的数学原理 ,对于解密下一个会话也没有什么益处。这个算法的缺点是需要计算更大的数字，所以对于一个很忙的服务器来说计算成本更大。“高级加密标准”（ AES ）出现在我们提供的很多密码套件中。它与RC4 的不同在于它一次加密 16 字节的“数据块”而不是一个字节，由于它的密钥可以大到 256 比特，所以很多人认为它比RC4 要更安全。

仅需 220 毫秒中 ,网络中两个端点连接到一起 ,提供了足够资料来信任对方 ,建立加密算法，并开始进行加密的信息传输。

我写了一个程序对文章中所提及的握手协议走了一遍。

原文链接：http://www.infoq.com/cn/articles/HTTPS-Connection-Jeff-Moser

相关内容：

- .NET安全漏洞波及Firefox
- 基于声明的身份认证指南
- SharePoint的列级安全性
- 虚拟技术与安全
- 深入理解Flash Player的安全沙箱

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

OSGi 4.2 发布了

作者Alex Blewitt **译者** 张龙

近日 OSGi 联盟发布了 OSGi 4.2 规范。虽然早期草案已经发布了，但此次则为最终版。此次发布都包含什么呢？InfoQ 此前曾报道过关于草案的相关信息，但现在规范最终版已经发布了，我们一起来看看 OSGi 4.2 最终版中都有哪些值得关注的新特性。

原文链接：http://www.infoq.com/cn/news/2009/10/osgi-4-2-released

开源Servlet容器Jetty 7.0 发布

作者 Alex Blewitt **译者** 胡键

10 月 8 号，Jetty 7.0 宣告发布，人们可由位于 Eclipse.org 上的新主页或通过 Maven Repository 下载。该版本不仅代表了 Jetty 6.0 的一次演变，而且还代表了代码库的重大调整和诸多性能改进。此外，延续（Continuation）API 现在也已移植到不同服务器之上。

原文链接：http://www.infoq.com/cn/news/2009/10/jetty-7-0-released

XMemcached——一个新的开源Java memcached客户端

作者 霍泰稳

XMemcached 是一个基于 Java nio 的 memcached 客户端。它线程安全，结构简单，支持所有的 memcached 文本协议和二进制协议，并且有比较优异的性能表现。它还支持一些高级特性，如 JMX、动态增删节点、客户端统计以及 nio 连接池等。InfoQ 中文站编辑

采访了该工具的核心开发人员 Dennis Zhuang。

原文链接：http://www.infoq.com/cn/news/2009/10/xmemcached-introduction

连接Java和.NET的RESTFul Bridge发布了

作者 Dilip Krishnan **译者** 张龙

近日位于法国的咨询服务公司 Noelios Technologies 发布了 Restlet 开源项目(面向 Java 的轻量级 REST 框架)的新版本 ,该版本包含了针对 ADO.NET Data Services 的 Restlet 扩展。此举是微软和 Noelios Technologies 协作的成果，这样 Java 开发者就可以轻松使用 ADO.NET Data Services 了。

原文链接：http://www.infoq.com/cn/news/2009/10/restlet-extension-micorosoft

VS 2010 Beta 2 发布，预计 2010 年 3 月将发布RTM

作者 Abel Avram**译者** 朱永光

伴随着一个新的图标，Visual Studio 2010 Beta 2 于近日登场亮相，其带来了性能的提升和更好的稳定性，并为 2010 年 2 月 22 日发布的 RTM 做好了相应准备。

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

微软发布Microsoft Ajax脚本库和脚本缩小器

作者 崇桦

微软的 ASP.NET 开发团队近日发布了 Microsoft Ajax 库的一个重大更新版本，该更新版本包括了客户端 AJAX 库的一堆新增功能和改进。此外，微软还同时发布了一个新的免费的 Microsoft Ajax 脚本缩小工具——Microsoft Ajax Minifier。

原文链接：http://www.infoq.com/cn/news/2009/10/microsoft-ajax-library

每月技术综述

十月技术综述

从本期开始，构架师将对各个技术领域在过去一个月的发展做一个总结。

SOA社区

话说 10 月的 SOA 社区，最大的新闻应该就要算是《SOA 宣言》的发布了。在经历了 SOA 的疯狂时代和年初的“SOA 已死”风波之后，社区终于开始考虑形成一份关于面向服务原则、意图和目标 ,以及面向服务架构模型的一份正式声明。考虑到工作组中有大名鼎鼎的 Thomas Erl，以及象 IBM 和 Oracle 这样重量级 SOA 厂商的代表，其重要性可想而知。但至于其影响是否能如当年的《敏捷宣言》一样深远，尚需等待时间的检验。另一个值得注意的现象就是，SOA 项目有和其他具有强业务驱动力类型的项目（如 BPM 和 MDM）融合的趋势，纯 SOA 项目的风光不再。最近的 SOA 联盟更名事件恰恰是这一趋势的反应。这也反应了这样一种老生常谈：“业务驱动才是王道。”

Java社区

我认为十月份有两件事情是大家可以关注一下的：一是 IntelliJ IDEA 开源了；二是 Android 出了新版本 2.0。IntelliJ IDEA 不用多说，是非常优秀的开发工具，其开源肯定会吸引一批粉丝转而投入其阵营，另外，开源也可能会促进 IntelliJ IDEA 的进一步发展。只是在 Eclipse 和 NetBeans 的夹击下，不知道还来不来得急。Android 2.0 则又加入了一些新特性，包括支持多种屏幕尺寸、支持多点触摸等等，Android 浏览器还号称支持 HTML5。再加上中移动与 Google 在 Android 上的持续合作关系，想必 Android 在中国会有一番作为。

Architecture社区

亚马逊开始提供 MySQL 服务，对担忧 MySQL 前景的人来说，这份信心保障恐怕比 Oracle 增加投入的承诺更实在。NoSQL 运动开始从斩钉截铁的“说不”迈向拥抱多样化的 NOSQL—

—Not Only SQL。Yahoo!从箱底翻出来一个 HTTP 缓存服务器 Traffic Server 送去 Apache 回炉。性能指标看上去颇有吸引力，我们的读者中是不是有人已经在做功课了？

.NET社区

对于微软而言，2009 年 10 月是一个新的起步。23 日 Windows 7 正式上市，作为一个从 Vista 痛苦中走过来的 Vista 优化版，Windows 7 获得了大量的好评。Windows 7 在给普通用户带来更佳用户体验的同时，也为 Windows 开发人员带来了一些有趣的开发新特性。我们不仅可以使用这些开发新特性开发出具备优异 UX 的应用程序，还可以为未来的多点触摸、传感器集成等特性做好准备。这为我们打好了第一个坚实基础。另外，微软于上个月公布了 VS 2010 Beta 2，一个带着 Go Live 许可的、整体界面用 WPF 重写、支持完整 Silverlight 开发和面向.NET 4.0 开发的全生命周期开发工具。它为让我们在开发下一代应用程序的过程中，打好了第二个坚实基础。

Ruby社区

在语言方面，各种 Ruby 实现在十月里都取得了不小的进步：JRuby 1.4 RC1 于十月发布，兼容 Ruby 1.8.7，并改进了 Java 集成；MacRuby 0.5 Beta 也已经发布，增加新的 VM、JIT 和 AOT 支持，同时删除了 GIL；IronRuby 也离 1.0 发布之日为期不远。随着各种 Ruby 实现工作越来越稳定，对标准的支持越来越好，相信对 Ruby 的推广有很大帮助。在社区方面，各 Gem Hosting 开始着手统一 Gem 发布，从而将会对 Ruby 的生态系统进行一次全面的改进，目前 RubyForge 即将停止工作，将会逐步迁移至 RubyGems.org 上，这对于广大 Ruby 开发者来说，应该算是一个不错的消息。

推荐编辑 | .Net 社区编辑 王瑜珩

各位读者好，很高兴能在这一期的《架构师》中与大家见面。我是 InfoQ 中文站.NET 社区的编辑王瑜珩，目前就职于 ThoughtWorks。我关注的技术领域有 Web 开发、设计模式、项目管理、敏捷方法以及自动化测试等。

最初认识 InfoQ 是在 07 年，由于身边的朋友正好是 InfoQ 的编辑，便对 InfoQ 产生了兴趣。不过那时工作比较忙，对自己能否胜任也有所怀疑，所以一直只是默默关注。直到 09 年初，工作上有了些闲暇，才决定尝试一下，从此便成为.NET 社区的一名小编。

初入 InfoQ，真的是什么都不懂，幸好 InfoQ 的编辑都很热心，主动帮助我解决各种工作上的问题，而平时邮件组中的各种积极的讨论也让我收获良多。然而让我印象最深刻的，还是大家对工作认真负责的态度。不管是对新闻的采集还是翻译，都务求能给读者最大的价值。有时为了一个英文单词的翻译也要讨论上半天，只为能够准确表达原文的意图。今年 QCon 和 AgileChina 大会的成功举办，给了我们一个相聚的机会，我也有幸客串了一把分会场的主持人，体会到了与平时不一样的工作感觉。

在这个日新月异的时代，技术的迅猛发展也让我们不得不加快追随的脚步。InfoQ 以高质量的内容及时提供业界的趋势信息，我也很高兴能为 InfoQ 和广大读者尽我的一点力。

封面植物

海菜花

海菜花，或名海菜，为我国特有植物，国家 3 级重点保护植物，曾经广泛分布于中国西南、华南诸省。特别是在广西西南部高原台地的河溪与云南湖泊当中能形成稳定的沉水植物群落。但 60 年代以来由于水体污染及其它因子影响，海菜花分布面积日逐缩小。

海菜长年生长于水中，其根伸成茎，茎伸为藤，藤生化为独片鹅掌形阔叶，四季轮开黄蕊白瓣小花，夏季结爪形肉质“菜果”。其叶翠绿欲滴，茎白如玉，花朵清香宜人，是一种蛋白质丰富和富有多种维生素及微量元素的天然野生水菜。食用方法包括炒、汆、烩、煮等多种方式。

在民间，常吃的菜谱如：“龙凤呈祥”汤。即在腊猪骨熬汤中，先放进蚕豆米煮化，再装入洗净不切的海菜藤茎，其味鲜爽甘香，食后能明目养肝、止咳化痰，是中老年人常吃的一种汤菜。另外“龙袍加身”这道菜也很不错。做法是以鲜海菜配火腿丁炒得半熟时，加淀粉及作料，兑酸醋烩焖片刻后出锅，其味酸香鲜美，对心血管疾病与尿频有辅助治疗作用。还有“龙爪拱目”、“龙藤碧海”等海菜的营养价值都很高。

海菜这种“富贵”菜，生长条件苛刻，只能生长在纯净的活泉水中，水质稍有污染或农田里施有化肥都会影响其生长。所以人们往往把是否生长海菜来判别水质是否受到污染。环保部门称其为“环保菜”。凡是泉水中成长着海菜，其水质一定是高标准的。在云南大理州的鹤庆像新华村这样的龙潭有 18 个，个个都生长着丰茂的海菜，当地人把捞摘到海菜作为招待客人的一种美食。


1kg.org
多背一公斤
爱自然 | 更爱孩子

架构师 11 月刊

每月 8 日出版

本期主编：朱永光

总编辑：霍泰稳

总编助理：刘申

编辑：胡键 宋玮 郑柯 李明 郭晓刚

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

交流群组：
http://groups.google.com/group/infoqchina

商务合作：sales@cn.infoq.com 13911020445

本期主编：朱永光，InfoQ 中文站.Net 社区首席编辑

IT自由人和环境保护者，微软最有价值专家（MVP）和MCSD。他有 15 年以上的编程实践经历，擅长微软相关技术和产品，目前主要关注动态语言、函数式语言、并行计算、云计算和RIA。个人博客为：http://redmoon.cnblogs.com。现在他作为共同创始人经营着一家环境保护技术公司，致力于用IT技术促进环境保护，并把环境保护理念带入IT业界。