Broadview
WWW.BROADVIEW.COM.CN

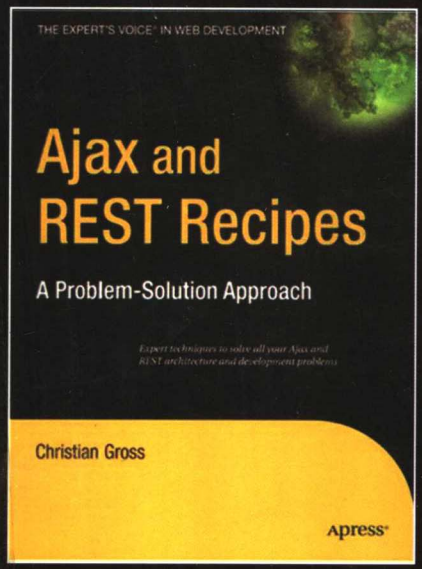
THE EXPERT'S VOICE® IN WEB DEVELOPMENT

Apress®

Ajax and REST Recipes：A Problem-Solution Approach

Ajax and REST Recipes中文版

A Problem-Solution Approach



解决您所有Ajax/REST
架构与开发问题的专家技巧

[美] Christian Gross 著
李琳骁 蔡 毅 译

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Ajax and REST Recipes 中文版

——A Problem-Solution Approach

Ajax and REST Recipes: A Problem-Solution Approach

[美] Christian Gross 著

李琳骁 蔡毅 译

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书探讨了针对 Ajax、JavaScript 和基于表现状态传输（Representational State Transfer，REST）的 Web service，以及其功能性实现的实践性解决方案。主要包含以下内容：编写高级 JavaScript 功能的技巧；构建处理动态内容的用户接口；实现 SOA 和通用 Web service 架构；针对特定情形来实现基于 REST 的 Web service。

该书实用性强，讲解全面。前半部分针对架构和开发 Ajax 应用期间无法避免的大量孤立问题，提供诸多解决方案，后半部分则将几个前后关联的诀窍（recipe）组合成较大的项目，让读者体会如何实现真实场景里的 Ajax 方案。通过此书，读者可以学习如何通过从服务器解耦客户机程序来模块化 Web 应用。

本书适合广大 Web 开发人员、Web 架构师参考使用。

1-59059-734-6 Ajax and REST Recipes: A Problem-Solution Approach by Christian Gross.

版权贸易合同登记号 图字：01-2007-3145

图书在版编目（CIP）数据

Ajax and REST Recipes 中文版：A Problem-Solution Approach / （美）格罗斯（Gross,C.）著；李琳骁，蔡毅译. —北京：电子工业出版社，2007.10
书名原文：Ajax and REST Recipes: A Problem-Solution Approach
ISBN 978-7-121-05077-0

Ⅰ. A… Ⅱ. ①格…②李…③蔡… Ⅲ. 计算机网络－程序设计 Ⅳ. TP393. 09

中国版本图书馆 CIP 数据核字（2007）第 146389 号

责任编辑：杨绣国
印 刷：北京智力达印刷有限公司
装 订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×980 1/16 印张：23 字数：400 千字
印 次：2007 年 10 月第 1 次印刷
定 价：49.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

译者序

2005 年，Jesse James Garrett 提出了“Ajax”这一术语，尽管 Ajax 不过是对现有几种技术的总结，此后却日渐走红。2000 年，Roy Fielding 对 Web 进行深入分析、研究和实践之后，在其博士论文里为超媒体应用确定了一种新的架构风格：REST（Representational State Transfer），实际上它是一组设计约束，包括无状态服务器、分布式缓存等。

随着融入式 Web 应用的普及，Web 应用架构逐渐开始背离 REST 准则，并且情况日益恶化，这促使人们反思自己的做法并设法寻找出路。Ajax 无疑是良方之一，它的特性使得 Web 应用不仅能实现快速响应、个性化的用户体验，而且只要设计得当，还能符合 REST 准则的简单性和可伸缩性等约束。本书正是围绕这两个主题展开的，开篇就从理论到实践对这两大主题之间的关联进行了深入探讨，并以问题—解法的方式（Problem-Solution Approach）逐一分析和处理在实践 Ajax/REST 过程中遇到的种种问题。

有人将遵循 REST 准则的架构称为“面向资源的架构”（Resource-Oriented Architecture），足见“资源”这一概念在 REST Web service 里的重要性，与之对应的另一个概念是“表现”，厘清这两个概念是解耦服务器/客户端使之遵循 REST 准则的关键所在，第 4 章里对此有详细的分析讨论，非常值得一读。

本书大体上由两部分组成，第一部分用大量篇幅介绍了 Ajax 的组成，包括 JavaScript、DHTML 等，针对大量常见的实践问题，详细描述了理论和背景，并提供了合理的解决方案。第二部分集中介绍 REST Web service 的具体实施，并将此前描述的各种诀窍应用于实例中，列举并详细探讨了多个实例，如海量或缓慢数据集的 Web service、Ajax 购物车和股票报价机等。

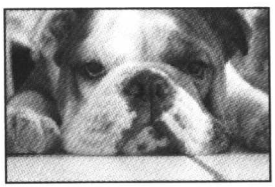
本书中文版的翻译工作由蔡毅和我合作完成，蔡毅负责前言部分、第 5 至 8 章的翻译，第 1 至 4 章的翻译由我负责。翻译期间，计划之外的事件接二连三地发生，导致翻译工作时断时续，幸得博文视点周筠女士和晓菲编辑体谅，适时调整翻译计划，最终顺利完成了全书的翻译工作。此外还要感谢责任编辑杨绣国，杨编辑细致耐心的工作，保证了本书的翻译质量。

祝各位读者阅读愉快！☺

李琳骁

2007 年 9 月 9 日于杭州

关于作者
About the Author



许多人都说，看一个人养的狗，可以大约了解这个人的性格。这幅图片上的狗就是我养的，它叫 Louys，是只英国牛头犬。嗯，我和这只英国牛头犬还真有许多共同的性格特征。

关于我的个人履历，非常简单：多数时间我喜欢坐在椅子上进行调试和编码的工作。自初次接触编程至今，我一直都很喜欢软件开发这项工作。我写过不少书，包括 *Ajax Patterns and Best Practices*[1]和 *How to Code .NET*，已由 Apress 出版社出版。

最近我喜欢上了用.NET 编写代码和做些试验，对我而言，它是非常不错的开发环境。.NET 让我感觉自己就像是在圣诞节一大早刚刚打开礼物的孩子，开心快乐。也许能猜到是什么礼物，但又不能完全确信。对于.NET 而言，虽然它不能给你短袜或汗衫[2]，但会带给你一整天的兴奋。

[1] 译注：中文名《Ajax 模式与最佳实践》，已由电子工业出版社于 2007 年 3 月出版。

[2] 译注：这里的短袜和汗衫，指的是圣诞袜和圣诞衫。关于圣诞风俗，在国外，圣诞节一大早，孩子们会收到圣诞老人送的圣诞袜或圣诞衫等礼物，这一刻往往是一年中最开心的时刻。作者用这个来比喻自己使用.NET 的兴奋程度。

本书的技术审校
About the Technical Reviewer

NICK MCCOLLUM

拥有超过 18 年的多种平台上企业级应用系统的设计开发经验。他是 Nusoft Solutions 公司的首席顾问，目前也是 *http://www.spout.com* 的架构师和核心开发人员。Nick 曾担任以下已出版图书的技术编辑：*C# COM+ Programming*（作者：Derek Beyer，出版社：John Wiley & Son，出版时间：2001 年），*Pro Ajax and the .NET2.0 Platform*（作者：Daniel Woolston，出版社：Apress，出版时间：2006 年）。他持有 MCSD 认证（Microsoft Certified Solution Developer，微软认证解决方案开发专家），并经常在微软的一些官方活动以及西密歇根州的一些本地用户组会议上发表演讲。

BERNHARD SEEFELD

于 1995 年共同创立了瑞士的搜索引擎（Swiss search engine）*http://search.ch*，并于 2004 年 10 月，发布第一个世界级的 Ajax 地图应用系统 *http://map.search.ch*。*Bernhard* 拥有瑞士伯尔尼大学理论物理学的理学硕士学位。

简　介
Introduction

写这本书，感觉很棒。本书以一种特殊的方式开头，但我喜欢这本书是因为它呈现的内容：使用 Web service 技术构建异步 JavaScript 和 XML（AJAX）应用。自 1999 年起，我就开始从事基于 Web service 的应用开发。有了 Ajax，Web service 可算找到了最佳拍档。

本书主要探讨针对 Ajax、JavaScript 和基于表现状态传输（Representational State Transfer，REST）的 Web service，以及其功能性实现的实践解决方案。我想推广这样一种开发方式，其中应用程序是解耦的（客户端代码和服务端代码互相分离），并经过充分的测试和维护，即使代码脱离 Ajax 上下文也能在客户端里运行。我信任 Ajax，不过我也相信，如果你开发的 Ajax 代码在非 Ajax 客户端上也能访问，那么与别人的应用相比，你的应用就具有明显的优势。

本书各个章节的主要内容如下。

- 第 1 章“开始起步（Getting Started）”：本章重点在于理解 Ajax、REST、Web service 和面向服务的架构（SOA）的基本定义。我给出的定义都是基于行业内观点而得出的，理解这些定义可以让你更容易理解地 Ajax/REST 的上下文及后面的章节。本章还介绍了如何使用 JavaScript 测试 Ajax/REST 应用，并展示了如何测试 Web service 契约（contract）和 JavaScript 客户端代码。
- 第 2 章“JavaScript Recipes”：本章重点介绍如何编写更为高级的 JavaScript 功能，涵盖了以下技术（其他章节也有涵盖）：使用委托（delegate）以便同时调用多个方法（method）或函数（function），将函数看作有状态的对象，并使用函数来进行初始化和判断。
- 第 3 章“动态内容 Recipes（Dynamic Content Recipes）”：本章主要阐述如何构建用来处理动态内容的用户接口（user interface）[1]。一般来说，动态内容是基于表单的内容，但也有例外。本章还会讲到验证（validation）技术，动态布局（dynamic layout）和状态管理（state-management）技术。
- 第 4 章“实现 SOA 架构（Implementing an SOA Architecture）”：本章详细探讨了 SOA 和

[1] 译注：全书将该词译作用户接口

基于 REST 的 Web service 这两个主题。本章还讨论了如何使用不同的 HTTP 动词（比如 GET 和 POST），如何测试 Web service，如何升级现有系统以便通过 REST 暴露它的功能点。

- 第 5 章“实现通用的 Web service 架构（Implementing a Universal Web Service Architecture）”：本章重点讨论在通用的上下文中使用基于 REST 的 Web service。本章还介绍了实现 Web service 必需的技术，有了这些技术，你能够将其集成到一个 mashup[2]中，能够用来生成多种数据格式，将动态 URL 集成到静态 HTML 页面。
- 第 6 章“实现海量或缓慢数据集的 Web service（Implementing Web Services for Large or Slow Data Sets）”[3]：本章重点讨论如何针对特定情形来实现基于 REST 的 Web service，其中会生成大量的结果，或者要花费很长的时间才能生成结果。本章将重点关注如何解决其中一个问题，并举例说明如何描述这些数据，实现 URL 及执行服务端的任务。
- 第 7 章“实现 Ajax 购物车（Implementing an Ajax Shopping Cart）”：本章主要讨论在线购物车系统的普遍问题。从抽象角度来看，当前购物车实现的主要问题是数据和特定用户关联在一起。本章会告诉你如何利用授权控制（authorization control）来定义和访问不与用户关联的（non-user-associated）URL。
- 第 8 章“不要提交你的表单——用 AJAX 吧（Don't Submit Your Forms-Ajax Them）”：本章重点讨论如何解决可怕的回退按钮提交问题，并详细阐述了如何管理表单提交过程，将状态关联到一个 HTML 页面，以实现更复杂的内容显示。

[2] 译注：整合不同来源的内容以提供统一完整体验的 Web 站点或应用程序被称为 Mashup。
[3] 译注：针对本章标题，“海量”表示结果集数据量很大；“缓慢”表示结果集生成的时间很缓慢。

目录一览
Contents at a Glance

目　录
Contents

第 1 章

开始起步

Getting Started

本章主要针对 Ajax（异步 JavaScript 和 XML）和 REST（表现状态传输）应用开发之前或期间不可避免的一些常见、普遍的问题和疑问提供解决方案。实际上，这些常见疑问并不一定是技术上的，反而往往与开发的理论或基本原理关系更大。这些疑问非常棘手，一旦开始思考它们，你就会被牵着鼻子绕圈走，最终则又回到起点。寻找答案的诀窍在于不要绕圈走，而是坚持某些假设并作出决定。

1.1 理解 Ajax 的定义和基本原理

Adaptive Path 公司的 Jesse James Garrett 提出了最初的 Ajax 定义[1]。根据该定义，Ajax 具有如下特征：

- 使用可扩展超文本标记语言（XHTML）和级联样式表（CSS）进行基于标准的呈现
- 使用文档对象模型（DOM）进行动态显示和交互
- 使用可扩展标记语言（XML）和可扩展样式表语言与转换（XSLT）进行数据交互和操作
- 使用 `XMLHttpRequest` 实现异步数据获取
- 使用 JavaScript 将所有东西绑定在一起

一言以蔽之，Ajax 是一种（需要 Web 浏览器的）Web 开发风格。Web 浏览器仅在必要时（only when necessary）才通过编程语言获取数据，并动态（dynamically）修改用户界面，而不是像传统做法那样，每次一有请求就刷新整个页面。在此，我要强调两个词**“动态”**和**“仅在必要时”**，因为这两个词是 Ajax 的本质所在。Ajax 和 JavaScript 是 *duck typing*[2]和 *latent-type*（隐式类型）编程技术的具体实例[3]。

---

[1] *http://www.adaptivepath.com/publications/essays/archives/000385.php*。

[2] *http://www.artima.com/weblogs/viewpost.jsp?thread=131502*。

[3] 译注：duck typing 是指“如果看起来像鸭子，听起来像鸭子，就可以把它当作鸭子”，也就是说，不管对象是什么，只在乎它做什么。引自 IBM/developWorks/《可爱的 Python：Python 之优雅与瑕疵》第 1 部分；《Programming Ruby 中文版，第 2 版》也有此论述。

Duck-typed 编程技术是指在编写代码时，预先并不知道类的定义，但知道对象具有某种特定行为。通过在运行时动态克隆（cloning）和装配（assembling）对象，我们就能实现重用。传统的面向对象程序设计则是在执行之前先定义好类型的行为（the behavior of the type）。

下面是一段动态 HTML（DHTML）和 JavaScript 应用示例的源代码，它阐明了 duck-typed 编程技术的本质。

Source: /website/ROOT/gettingstarted/PrototypeBased.html

```
<html>
<head>
   <title>Prototype-based Programming</title>
<script language="JavaScript" type="text/javascript">
function Variation1() {
   document.getElementById( "output").innerHTML = "Ran Variation 1";
}

function Variation2() {
   document.getElementById( "output").innerHTML = "Ran Variation 2";
}

var obj = new Object();

function RunVariation() {
   obj.runIt();
}
</script>
</head>
<body>
   <input type="button" value="Variation 1"
      onclick="obj.runIt = Variation1; RunVariation()" />
   <input type="button" value="Variation 2"
      onclick="obj.runIt = Variation2; RunVariation()" /><br/>
   <div id="output">Nothing yet</div>
</body>
</html>
```

在这个例子里，加粗部分的代码例示了 duck-typed 编程结构（programming constructs）[4]。在加载代码时，Web 浏览器会从上到下解析这段代码。一旦完成解析，下列类型和对象实例即被激活：

- 函数 `Variation1`、`Variation2` 和 `RunVariation` 的定义；
- 变量 `obj` 的实例化和定义，`obj` 引用了一个最基本的（plain vanilla）`Object` 实例；
- 两个按钮（`Variation 1` 和 `Variation 2`）的定义，点击时会执行一些 JavaScript 代码；
- 标识符为 `output` 的 HTML 元素 `div` 的定义。

直接调用函数 `RunVariation` 会产生异常[5]，因为 `obj` 是个纯对象实例，并未实现 `runIt` 方法。诸如 Java、C#或 C++等传统编程语言无法编译 JavaScript 代码，因为函数 `RunVariation` 对

[4] 译注：Wikipedia 的定义，Pertains to the basic elements, commands, and statements used in various programming languages. It does not include general concepts or processes。

[5] 译注：例如，去掉 `onclick="obj.runIt = Variation1; RunVariation()"` 里的 `obj.runIt = Variation1;` 部分，用 Firefox 或 IE 里浏览这段 HTML 代码时就会产生错误。

类型执行了一个该类型并不支持的方法。

对象方法在像上面的源代码那样被调用时，即称为隐式类型（latent typing）。隐式类型是指直到应用程序运行时才真正确定与变量关联的类型。以上面的源代码为例，这意味着直到应用程序执行时，obj 的准确行为才得以确定。因此，RunVariation 也许能工作，也许不能。

在示例代码里，当 input 按钮被按下时，属性 obj.runIt 随即被赋给 Variation1 或 Variation2。在该属性被赋值后，input 按钮调用函数 RunVariation，后者又调用属性 obj.runIt。如果该属性已被赋值，则函数 Variation1 或 Variation2 被调用。将函数赋给属性便是 duck-typed 程序设计的本质所在。

疑问随之而来，如果编程语言使用隐式（latent）编程技术，是否也就意味着支持 duck-typed 编程呢？如果不是，那么两者又有哪些不同？如果某种编程语言支持隐式类型，这并不意味着支持 duck-typed 编程。但如果编程语言支持 duck-typed 编程技术，那它肯定支持隐式类型。在支持隐式类型但不支持 duck typing 的编程语言中，C++便是个很好的例子[6]。下面的代码例示了隐式类型：

```
class LatentTypeCaller< T> {
   public void CallIt( T t) {
      t.LatentDefinedMethod();
   }
}
```

在这段代码里，T 是个 C++模板类型。CallIt 的实现会调用方法 t.LatentDefinedMethod。从这段代码看不出类型 T 为何物，但不管它是什么，T 必须支持方法 LatentDefinedMethod。C++并不支持 duck typing，因为 T 的 LatentDefinedMethod 方法无法被动态赋值。

随着.NET 2.0 和 Java 5.0 引入所谓的 generics 模板类型功能，你或许会笃定 generics 支持隐式类型。但在.NET 或 Java 里，也不能出现上述用 C++写的代码，否则编译器会提示未限定的类型（unconstrained type）相关错误。要去除 C#或 Java 里的编译器错误，你必须将 T 限定为支持方法 LatentDefinedMethod 的类型。

反对 duck-typed 编程设计和隐式类型的一个常见论据是，在执行代码前，你根本不知道代码是什么样子。相反，要求类型的显式定义（explicit definition）或静态检定[7]（static typing）的 C++、.NET 和 Java 编程环境则能产生稳定而健壮的代码。至少这是那些支持静态类型的人所宣扬的论据。静态检定虽能保证程序通过编译并连接组装在一起，但并不能担保这个程序的行为与预期相符。以下面的代码为例，这段代码例示了静态检定是如何被愚弄的：

```
class Math {
   public long add( long value1, long value2) {
      return value1 - value2;
   }
}
```

[6] 译注：作者所举的例子似乎有点问题，C++并不符合上面给出的隐式类型定义，C++的模板机制是种编译技术。

[7] 译注：全书大部分情况下译为静态类型，在强调动作时，则译为静态检定。

这个例子里定义了一个 Math 类，Math 只有一个方法 add，意在求两数之和。如果有程序使用了 Math，编译器就会验证 Math 是否已正确实例化，且方法 add 被正确调用。不过，编译器并不会验证两数相加的结果是否正确。欲达此目的，你需要创建一个明确的测试来验证两数相加的结果正确。一个编写良好的测试能快速捕捉到 add 的错误，并发现它实际上是做了减法。这个论据的要点并不是要嘲弄受限语言（constrained language），而是要阐明编译器无法验证程序的正确性。

受限语言所谓优势的另一个例证是 IntelliSense[8]。IntelliSense 具有如下特性：键入变量的名字，敲击键盘上的句点键，即可查看与对象相关的所有方法、属性和数据成员。借助 IntelliSense，编码的速度可以更快，因为你不必再苦思冥想一个方法到底有几个参数。不过，Charles Petzold 提出了一个反对 IntelliSense 的有趣论点[9]，声称它会令心智退化。他的主要论据是，IntelliSense 的存在会导致.NET 包含 5 000 个类、45 000 个 public 方法及 15 000 个属性。当然，这其实是在回避问题实质，有哪个开发人员能真正掌握所有这么多类/方法/属性呢？Petzold 用冗长且命名诡异的类型来说明这个问题。

使用 IntelliSense 还有一个不可避免的缺点，即造成如下趋势：它会导致开发人员采用自底向上（bottom-up）的技术创建程序。如果类型还未定义，IntelliSense 就无法正常工作，因此就必须使用自底向上技术。这样一来，自顶向下（top-down）的编程技术就会变得更加少见，因为开发人员会跌进 IntelliSense 的蜜罐，习惯于编写自底向上的代码。

面对这两个论据，你也许会想，“究竟为什么我们还要用受限语言编写代码呢？简单来说，受限语言束缚了我们”。答案是：受限语言再辅之以适当的 IDE（集成开发环境），给人一种温暖柔和的感觉，因为我们能够定义可编程契约（programmatic contract）和类型，并确定一切工作正常。

然而，现实并不是黑白分明的，总有些时候使用受限语言或 duck-typed 语言才更为合适。一言以蔽之，对于实现那些定义已非常明确的逻辑来说，受限语言就很管用。例如，计算抵押贷款（mortgage）就是一种已知的技术，只是有不少规则与该计算相关联。将大量定义了准确行为的先决条件与这个计算相关联并不是太困难。

在你希望系统里有更强的灵活性时，比如创建抵押贷款计算程序，duck typing，或更恰当地说，动态语言就非常适合。银行家们乐于创建专门的抵押贷款套餐，例如有些套餐指出客户可以怎样预先少付一些，过段时间偿还更多余款，或者预先付一大笔钱，一段时间内不用还款，然后再付清余款。这些例子与用来计算抵押贷款的数学运算别无二致。不同的只是这些计算的组合方式。

你可以用受限语言对付所有问题，也可以用 duck-typed 语言搞定一切。此外，你也可以通过 Web service 或兼容层（compatibility layer）混合搭配使用这两种语言。在编写 JavaScript 和 Ajax 代码时，你编写的就是 duck-typed 代码。Ajax 不只是 XML、JavaScript 和 DHTML 的简单组合，它还能动态创建可织入（inject）和执行的内容、代码，认识这一点非常重要。受限语言也有可能支持动态创建内容和代码，但实现难度和枯燥会令人抓狂。为了例证这一点，我们来看一个简单的 click-me（点击我）应用程序，该程序会被动态扩展以包含第二个 click-me 按钮：

[8] 译注：智能提示、智能补全之意。

[9] *http://charlespetzold.com/etc/DoesVisualStudioRotTheMind.html*。

```
public class MainFrame extends JFrame {
    public MainFrame() {
        initialize();
        setTitle(titleText);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(panel1, BorderLayout.CENTER);
        button1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    label1.setText("hello world");
                }
            }
        );
    }

    public void show() {
        pack();
        setSize(new Dimension(
            Math.max(getWidth(), windowSize.width),
            Math.max(getHeight(), windowSize.height)));
        setLocationRelativeTo(null);
        super.show();
    }

    private String getLocalizedText(String text) {
        return text;
    }

    private JButton button1; // Button,Click Me,1,,,1
    private JPanel panel1; // Panel,Title
    private JTextArea label1; // Text,,1

    // Implementation (X-develop Swing designer code)
    ...
    // End of Implementation (X-develop Swing designer code)
}
```

这段 Java 示例代码仅作辅助说明之用，.NET 也提供了同样的方法。上述代码大都可归为基础结构（infrastructure）。例如，从三个数据成员 button1、panel1 和 label1 的声明来看，你根本不知道它们会做些什么。你甚至无法推断这些数据成员的依赖（dependency）或层级体系（hierarchy）。

通过查看各个方法调用 setLayout、add、setSize 和 setLocationRelativeTo，以及大量已移除的 GUI 设计器相关代码，你可以推断出数据成员的性质。上述示例代码及其中的数据成员和各种方法突显了如下事实：对于你想（对应用系统）做的任何事情，受限语言要求你一五一十地明确告知应用系统。下面我们用 DHTML 和 JavaScript 技术实现同一功能，做法与上面大不相同：