HTML5: Up and Running

Dive into the Future of Web Development

HTML5 揭秘

Mark Pilgrim 著

常可 胡金埔 赵静 译 赵泽欣 审校

O'REILLY® | Google™ PRESS

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

HTML5揭秘

如果你还不熟悉HTML5的新特性，现在是时候去了解了。本书提供了实用的信息，告诉你如何及为什么说HTML的最新版本将会显著地改变Web开发的方式。

HTML5仍然还在发展中，但是像Safari、Mozilla、Opera及Chrome这些浏览器已经可以支持它的许多特性——移动设备上的浏览器甚至更加支持它。《HTML5揭秘》通过包含许多代码、图表及截图的实战范例，细致地引导你学习这门变化巨大的新语言。你将会学到如果使用HTML5代码来给网页加上视频、离线功能等——并且能立刻上手。

- 学习了解新增的语义元素。例如<header>，<footer>及<section>
- 认识Canvas（画布），一种可以通过JavaScript编程控制的二维绘图界面
- 无须用到第三方插件就能在网页上嵌入视频
- 使用Geolocation（地理定位）来让网站访客分享他们的地理位置
- 全面超越cookies的本地存储（local storage）
- 构建离线Web应用程序，在网络断开的情况下仍可以运作
- 学习多种新的表单输入框类型
- 使用微数据（microdata）来创建你自己的HTML5词汇表

建议本书读者具有一定的编程经验。

“本书不仅权威实用，而且很漂亮。我自己也使用过书中的各种经验……并且以后也会继续使用。”

——Tim Bray
Google开发者关系团队

“……无论是初学者还是高手，这本书都绝对适用。”

——Brad Neuberg
Google软件工程师

Mark Pilgrim是Google开发者关系团队中的一员，他精通开源技术和开放标准。著有多本技术书籍，包括“Greasemonkey Hacks”、“Dive Into Python”和“Dive Into Python 3”。

Google™ PRESS

O'REILLY®
oreilly.com

图书分类：Web开发

策划编辑：卢鹏翔
责任编辑：杨绣国

Broadview®
WWW.BROADVIEW.COM.CN
www.phei.com.cn

ISBN 978-7-121-12408-2

定价：45.00元

O'REILLY®

HTML5揭秘

HTML5: Up and Running

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书全面而深入地对HTML5相关的技术进行详细介绍和剖析。“从开始到现在”道出HTML5的坎坷发展史；“HTML5特性检测”介绍了多种针对不同特性的检测方法；“本地存储”揭开了如何把“数据库”搬到客户端的神秘面纱；“离线应用”展示了脱机状态下依然能让Web应用完好无损的神奇技艺；“疯狂的表单”大秀了一把炫酷无比的下一代Web表单……它几乎涵盖了HTML5标准中描述的所有新特性。本书在以诙谐的文字结合生动的实例介绍HTML5特性的同时，还深入剖析其内部原理。让读者不仅知其然，而且知其所以然。无论是刚接触Web前端技术的新人，还是经验丰富的老手，只要是HTML5技术爱好者，都会从本书中受益。

版权贸易合同登记号　图字：01-2010-6579

图书在版编目（CIP）数据

HTML5揭秘 / 皮尔格林（Pilgrim,M.）著；常可，胡金埔，赵静译.
—北京：电子工业出版社，2010.12
书名原文：HTML5: Up and Running
ISBN 978-7-121-12408-2
Ⅰ. ①H… Ⅱ. ①皮… ②常… ③胡… ④赵… Ⅲ. ①超文本标记语言，HTML一程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字（2010）第227311号

策划编辑：卢鹞翔
责任编辑：杨绣国
项目管理：杨绣国
印　　刷：北京市天竺颖华印刷厂
装　　订：三河市鑫金马印装有限公司
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**：14　　**字数**：280千字
印　　次：2010年12月第1次印刷
定　　价：45.00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.

O'Reilly Media, Inc.介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly Media, Inc. 授权电子工业出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在Unix、X、Internet和其他开放系统图书领域具有领导地位的出版公司，同时也是在线出版的先锋。

从最畅销的“The Whole Internet User's Guide & Catalog”（被纽约公共图书馆评为20世纪最重要的50本书之一）到GNN（最早的Internet 门户和商业网站），再到 WebSite（第一个桌面 PC 的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为 O'Reilly Media, Inc.紧密地与计算机业界联系着，所以O'Reilly Media, Inc. 知道市场上真正需要什么图书。

译者序

HTML5 可能是眼下 IT 前端圈子里最为热门的词语，在乔布斯看来，未来的客户端也将是 HTML5 的天下。作为 Web 开发工程师更有理由来更新自我的知识体系，学好这门新技术。在如今这个信息爆炸的时代，一本好书往往就是一位良师益友，能帮你解开学习道路中的一个又一个谜题。《HTML5 揭秘》就是一本这样的书。

本书全面而又深入地对 HTML5 技术作了介绍和剖析。从文档类型到全新 HTML 标签，从视频到画布，从本地存储到离线应用，涵盖方方面面；从 MIME 类型的发展史到其详细含义，从本地存储方式到其内部实现原理，从 Form2.0 的全新标签到其使用注意事项，讲解深入浅出。

本书不是一本关于 HTML5 的手册，而是一本可以教会你使用 HTML5 的书。如果以学习修车为例，本书很少教你一步步如何操作，取而代之的是有关引擎的内部原理及其发展历史。比如第 4 章“Canvas 绘图”，并非只教给你怎么画，更多的是教你绘图的整个过程和原理；第 5 章“网络视频”，讲到了很多平时被我们忽视的视频容器、编解码器，以及一些相关历史。从而，我们越发地觉得，在一个领域用心研究的人，他的记忆是有时间点的，也就是熟悉那个领域相关技术的发展史。

作为译者，我们一方面深感荣幸，另一方面也倍感压力。我们知道，中文和英文思维有着很大的区别，倘若带着英文思维去直译，往往读者就会觉得晦涩，甚至误解了原文的本意，这对于译者而言就是莫大的罪过了。正如 Gilbert Highet 所说：“一本书写得不好，那只是一种过错，而将一本好书翻译得糟糕那就是犯罪了”。带着这种心态，我们希望能在保持原文意思的基础上，尽可能地采用中文思维来翻译，呈现给读者一个中文的语境。

在这里，特别感谢博文视点的编辑们，是他们的见识使这本好书有可能尽早地与国内读者见面。由于本书覆盖面广，翻译难度也较大，我们虽然在翻译中下了不少功夫，但译文仍难免会出现一些疏漏，恳请认真阅读的同行和朋友们不吝赐教。

常可 胡金埔 赵静

2010 年 11 月

前言
Preface

引言
Diving In

HTML5 是什么？HTML5 是下一代的 HTML，将会取代 HTML 4.01，XHTML 1.0 及 XHTML 1.1。HTML5 将会提供现代 Web 应用程序所必需的新功能。它同时也会将许多用于 Web 平台上的技术进行标准化，开发者们已经使用了这些技术多年，但从未有标准委员会对其进行审核和归档。（“Window 对象从未被正式写进标准文档里”——你听到这个会不会惊讶？除了新功能/特性，还有许多那些已经出现多年，“既成事实”的浏览器标准功能，HTML5 将会首次尝试将这些东西都正式地文档化。）

和它的前辈一样，HTML5 也被设计为是跨平台的。你并不须要一定运行 Windows 或者 Mac OS X 或者 Linux 或者 Multics 或者其他任何特定的操作系统，即可享受 HTML5 的优势。唯一需要的就是一个现代的浏览器。所有主流操作系统上都有免费的现代浏览器。你或许已经在使用一个支持部分 HTML5 特性的浏览器了。比如最新版的 Apple Safari、Google Chrome、Mozilla Firefox 及 Opera，它们都支持许多的 HTML5 特性。（在本书中附有详细的浏览器兼容性列表。）在 iPhone、iPad 及 Android 系统上预装的移动设备浏览器也都对 HTML5 提供极好地支持。甚至微软也已经宣布即将问世的 Internet Explorer 9 会支持部分 HTML5 的功能。

本书的内容集中在以下八个主题：

- 新的语义元素，如<header>、<footer>和<section>（详见第 3 章）。
- Canvas，一个可以通过 JavaScript 编程控制的二维绘图界面（详见第 4 章）。
- 无须安装第三方插件就可以在网页中嵌入视频（详见第 5 章）。
- 地理定位，使得访问者可以通过 Web 应用程序分享他们的所在位置（详见第 6 章）
- 无须使用第三方插件就能实现的永久性本地数据存储（详见第 7 章）。

- 离线 Web 应用程序，即使在没有连接网络的情况下也可以运作（详见第 8 章）。
- 经过改进的 HTML Web 表单（详见第 9 章）。
- 通过微数据创建你自己的 HTML5 扩展词汇表，赋予网页新的语义性（详见第 10 章）。

HTML5 被设计为尽可能地后向兼容现有的 Web 浏览器。新的特性都建立在已有特性的基础之上，使得你可以为旧版本的浏览器提供容错内容。如果需要更全面的控制，你还可以检测浏览器对各项 HTML5 特性的支持情况（参见第 2 章），这只需几行 JavaScript 就能做到。不要依赖脆弱的对浏览器的侦测来断定 HTML5 的支持情况！相反，使用 HTML5 本身来测试你想要用到的功能。

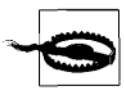
本书中所用的约定
Conventions Used in This Book

本书使用以下字体排版约定：

斜体（*Italic*）
: 表示专业词汇、链接（URL）、文件名和文件扩展名。

等宽字体（`Constant width`）
: 表示广义上的计算机编码，它们包括变量或函数名、数据库、数据类型、环境变量、语句和关键字。

该图标表示一个提示、建议或一般的说明。



该图标表示警告或提醒。

示例代码的使用
Using Code Examples

本书的目的是为了帮助你搞定工作任务。在一般情况下，你都可以在程序和文档中使用本书中的代码。除非需要复制这些示例代码的相当大部分，否则无须联系我们以获得许可。比如说，当你编写的程序用到了本书中的若干示例代码，这并不需要特别许可。但是，销售或分发含有 O'Reilly 书籍附带的示例程序的光盘则需要获得许可。当你在回答他人问题时援引本书内容，或者引用书中的范例代码，也不用申请许可。而如果要把本书中的代码大量地引用到你的产品文档中，则需要许可。

对于引用时注明出处，我们表示感谢，但并不硬性要求。署名信息通常包括书名、作者、出版商及 ISBN。例如："HTML5: Up and Running，由 Mark Pilgrim 撰写。O'Reilly 媒体公司 2010 年版权所有，978-0-596-80602-6。"

如果你觉得你使用示例代码的情况超出了以上描述的不需要许可的范围，请随时联系我们 *permissions@oreilly.com*。

关于本书的版本说明

A Note on the Editons of This Book

本书源自最初的 HTML5 网页版本，位于 *http://diveintohtml5.org/*，由作者本人维护。电子书和 Safari Books Online 版本包括所有的原始超链接，而印刷版只包括了其中一部分。如果你阅读的是印刷版，请参照其他版本，或原始在线版本，以获得更多信息。作者所维护的 *http://diveintohtml5.org/*网站上包括了本书中的范例的现实版本，本书中，许多相关内容为了便于出版经过了一些修改。请访问 *http://diveintohtml5.org/*来查看这些例子，但是要注意，不同的浏览器可能有不同的呈现结果。

如何联系我们

How to Contact Us

请将对本书的有关意见和问题告知出版商：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

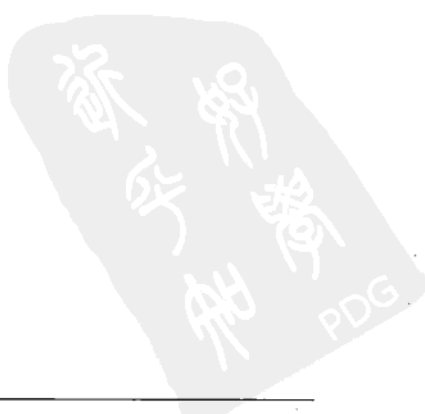
与本书有关的在线信息如下所示：

http://oreilly.com/catalog/9780596806026/（原书）
http://www.oreilly.com.cn/book.php?bn=978-7-121-12408-3（中文版）

北京博文视点资讯有限公司（武汉分部）

湖北省 武汉市 洪山区 吴家湾 邮科院路特1号 湖北信息产业科技大厦1402室

邮政编码：430074
电话：(027)87690813　　传真：(027)87690013
读者服务网页：*http://bv.csdn.net*
E-mail：
reader@broadview.com.cn（读者信箱）
bvtougao@gmail.com（投稿信箱）



目录

Table of Contents

第 1 章

从开始到现在

How Did We Get Here?

1.1 引言

Diving In

前不久，我偶然读到某 Mozilla 开发人员的一个观点，内容是关于在创建一种标准的过程中存在的冲突（*http://lists.w3.org/Archives/Public/public-html/2010Jan/0107.html*）：

> 一份技术规范和它的具体实现必须要做到步调一致。实现先于规范完成不是什么好事情，因为人们会开始依赖这些已实现的细节，这样会对规范形成制约。然而，你也不希望在规范已经完成时还没有任何相关的具体实现和实践经验，因为这样规范就得不到任何反馈。这里面如果存在着无法避免的冲突，而我们也只能硬着头皮去克服了。

请你先牢记这番话，接下来我开始讲述 HTML5 是如何横空出世的。

1.2 MIME 类型

MIME Types

本书的内容是关于 HTML5，不是旧版本的 HTML，也不是任何版本的 XHTML。但要理解 HTML5 的历史及其背后的发展动力，你需要先了解一些技术细节。具体来说，就是 MIME 类型。

每当 Web 浏览器请求一个页面时，Web 服务器会在发送实际页面内容之前，先发送一些头信息（header）。这些头信息通常是不可见的，当然如果你有兴趣，也可以借助许多 Web 开发工具来查看到这些信息。看不到不代表不重要，事实上，浏览器需要这些信息来决定如何解析随后的页面内容。最重要的头信息是 `Content-Type`，比如：

```
Content-Type: text/html
```

“`text/html`”即是这个页面的“内容类型（content type）”，或者称作“MIME 类型”。这条头信息将唯一确定某个资源本质上是什么，也因而决定了它应该如何被呈现。图片文件有自己的 MIME 类型(JPEG 图像的类型是 `image/jpeg`,PNG 图像则是 `image/png`，等等）。JavaScript 文件有它自己的 MIME 类型。CSS 样式表也有自己的 MIME 类型。任何

资源都有自己的 MIME 类型。整个 Web 都依靠 MIME 类型运作。

当然，现实情况比这还要复杂得多。非常早期的 Web 服务器（我指的是 1993 年）并不会发送 Content-Type 头信息，因为它当时压根还不存在。（1994 年以前还没有“内容类型”这一概念。）为了兼容到这样的情况，某些流行的浏览器还会在一些特定情况下忽略 Content-Type 头信息。（这一行为被称为“内容嗅探”。）但作为一个一般性的经验法则，凡是在一个网页上能看到的东西，比如 HTML，图片，脚本，视频，PDF 文件，所有这些可以用一个 URL 地址定位的东西，都具有特定的 MIME 类型——在 content-type 头信息中指明。

把上面这些内容先塞到脑子里吧。在后面还会复习到。

1.3 很长的题外话：一份标准是如何诞生的？
A Long Digression into How Standards Are Made

为什么会存在<img>这个元素？这个问题想必没有人会经常考虑。显然，必定是有人创造了它。这种东西才不会自己从石头里蹦出来。每个元素，每个属性，每个你所用过的 HTML 特性都是这样——某些人创造了它们，决定它们的用法，并且把全部相关事项都记录了下来。这些人不是造物主，不会完美无暇。他们是人——当然，是很聪明的人——但毕竟也只是人。

“公开”地制定标准有一大好处，你可以重返过去，寻找上述问题的答案。与之相关的讨论都会出现在邮件列表里，这些邮件通常留有存档，并且可以被公开搜索。所以，我决定做一下对电子邮件的考古，来尝试回答这个关于<img>元素的问题。为此我得追溯到万维网联盟(W3C)诞生之前。回到 Web 时代最开始的日子，那时候 Web 服务器的总数还屈指可数——好吧，也许再加上脚趾。

在 1993 年 2 月 25 日，马克·安德雷森（Marc Andreessen）写道：[1]

我想提议一个新的可选 HTML 标签：

IMG

其必需参数是 SRC="url"。

它指明一个位图或像素图文件，让浏览器尝试通过网络获取并解析为一幅图像，显示于页面上标签所在的位置。

例如：

<IMG SRC="file://foobar.com/foo/bar/blargh.xbm">

（没有结束标记，它是一个独立标签。）

[1]译注：*http://1997.webhistory.org/www.lists/www-talk.1993q1/0182.html*。点击“下一条消息”和“上一条消息”的链接可以跟踪阅读这份长达数页的讨论。

这个标签可以像其他东西一样嵌在锚点元素里，此时，它就变成一个图标链接，和文字链接一样可以被激活。

对于支持何种图像格式，浏览器应该有足够的灵活性。例如，Xbm 和 Xpm 就应该被支持。如果浏览器无法解析给定的格式，它可以自行决定做何种代替处理（X Mosaic 会显示一个默认图像作为占位符）。

对这个标签提供支持是 X Mosaic 的必要功能；我们已经将其实现，并且至少会在内部使用它。我也乐于听到任何有关如何在 HTML 中处理图像的建议；如果你有更好的主意，烦请告知。我知道，关于图像格式这一点，这里的提议还有些模糊，但我也想不出其他办法，只能说“让浏览器自行处理”，然后等待完美解决方案的出现（MIME？也许吧）。

以上内容需要一些解释。Xbm 和 Xpm 是当时 Unix 系统上流行的图像格式。

“Mosaic”是最早的浏览器之一。（“X Mosaic”是其运行于 Unix 系统上的版本。）当马克在 1993 年初写下这篇帖子时，他尚未成立那家将让他一举成名的公司：Mosaic 通信公司，也还没有开始设计公司的主打产品“Mosaic Netscape。”（你可能更熟悉它们后来的名字，“网景公司”（Netscape Corp.）和“网景浏览器。”（Netscape Navigator））

“MIME？也许吧”这句话影射的是内容协商(content negotiation)，这是一个 HTTP 特性，其中：客户端（如 Web 浏览器）借此告诉服务器（如 Web 服务器）它支持什么类型的资源（如 image/jpeg），然后服务器就可以按客户端需要的格式返回内容。最初的超文本传输协议(HTTP)定义于 1991 年（1993 年 2 月时有了唯一的实现版本），而其中还没有办法能让客户端告诉服务器它所能支持的图像格式，这就是马克在设计上所面临的困境。

几个小时后，托尼·约翰逊（Tony Johnson）回复道：

我在 Midas 2.0（用于 SLAC，很快就要对外发布了）中也有很类似的东西，除了名称不同，以及还有一个额外参数 NAME="name"。在功能上，和你提出的 IMG 标签则几乎是一模一样的。例如：

```
<ICON name="NoEntry" href="http://note/foo/bar/NoEntry.xbm">
```

给出这个 name 属性是为了允许浏览器使用一系列的“内置”图像。如果 name 匹配了“内置”图像，浏览器就直接使用它，而不用再去网络上抓取。这个属性也可以作为针对“行模式”浏览器的提示信息，告诉它在图像位置显示什么符号。

我不太关心具体参数或者标签名称，不过如果我们都用相同的命名，倒是更为明智。我也不很在意缩写，比如为什么不用写全的 IMAGE=以及 SOURCE=。不过我倒是更倾向于使用 ICON 作为标签名称，因为它意味着应该用较小的图片。不过也许 ICON 又是个意思太多的词？

这里提到的 Midas 是另一个早期的 Web 浏览器，和 X Mosaic 同时代。它是跨平台的，在 Unix 和 VMS 上都能运行。“SLAC”是斯坦福直线加速器中心——现在的斯坦福国家加

速器实验室——的名字缩写。美国第一台 Web 服务器（也是欧洲以外的第一台 Web 服务器）就位于这里。在托尼写下这篇帖子时，SLAC 已经是万维网的老前辈了，它的 Web 服务器上已存放有五个网页达 441 天之久。

托尼继续写道：

既然我们正在讨论关于新的标签这一话题，我还有一个类似的标签，想在 Midas 2.0 中提供支持。它大致是：

```
<INCLUDE HREF ="...">
```

这个标签的作用是把另一个文档包含进来，置于当前文档中该标签所出现的位置。原则上，所包含的文件可以是任何类型，但其主要目的还是为了在文档中嵌入图片（此例中图片可以是任意大小）。此外，当 HTTP2 成形时，包含文档的具体格式就可以另外协商了。

“HTTP2”指的是 1992 年诞生的基本超文本传输协议(Basic HTTP)。1993 年初的时候，这一协议很大程度上仍然没有任何实现。被称为“HTTP2”的规范草案持续进化，并最终标准化，成为为“HTTP 1.0”。HTTP 1.0 确实包括有用于内容协商的请求头信息（reuqest headers），也就是那个“只是也许”的“MIME”。

托尼继续说：

我正在考虑另一种选择是：

```
<A HREF ="..." INCLUDE>See photo</A>
```

我不是很喜欢给<A>标签添加更多的功能，这里的写法是为了那些对不支持 INCLUDE 参数的浏览器也保持兼容性。我的想法是，那些支持 INCLUDE 参数的浏览器，会把链接里的文字（本例中的“See photo”）替换为将被包含的文件（图片），那些老旧的、愚蠢的浏览器就干脆完全忽略掉 INCLUDE 标签。

这一提议从未付诸实施，尽管这个“在图像缺失的情况下显示一段替代文本“是一个重要的，提高网页可用性的技术。而这一点在马克最初的<IMG>提案里是没有的。多年以后，这个特性被规定为<img alt>属性，Netscape 则错误地将它实现成一个 tooltip 提示信息。

在托尼发帖几个小时之后，蒂姆·伯纳斯·李回帖说：

我原本觉得图像应该这样标记

```
<a name=fig1 href="fghjkdfghj" REL="EMBED, PRESENT">Figure</a>
```

其中 rel 属性的值的意思是：

```
EMBED 呈现时嵌入链接所指内容
PRESENT 呈现源文件时，始终呈现这个标签内容
```

请注意，这些值可以任意组合，如果浏览器一个都不支持，它也不应该崩溃。

[我]发现要使用这种方法创建可选择的图标，就意味着嵌套的链接标记。嗯……但我不想要一个特殊的标签。

这一建议也从未付诸实施，但 rel 属性仍然存在（参见第 36 页“朋友和（链接）关系”）。

吉姆 · 戴维斯（Jim Davis）补充说道：

要是有一种方法可以指明内容类型就好了，比如说：

<IMG HREF="http://nsa.gov/pub/sounds/gorby.au" CONTENT-TYPE=audio/basic>

当然我也完全赞同必须通过文件扩展名来区分内容类型。

这一建议也从未付诸实施，但 Netscape 随后开始支持通过<embed>标签来嵌入任意的媒体对象。

杰伊 · C · 韦伯（Jay C. Weber）提出了疑问：

虽然我非常希望浏览器能够支持图像文件，但我并不赞同每次都专门指明内容类型。对 MIME 类型判断机制失去热情了么？

马克 · 安德雷森回答说：

MIME 即将成为标准的文档类别判断机制，这里所提的并非是个替代方案，而是提供一种必要的，简单的，独立于 MIME 的功能实现。

韦伯又回复道：

姑且放下 MIME 不谈，我们似乎偏离了重点。我所反对的是讨论“我们该如何支持嵌入图像”，而不是“我们该如何支持嵌入各种媒体”。

否则，下周就可能有人建议“让我们增加一个新的标签 <AUD SRC="file://foobar.com/foo/bar/blargh.snd">” 来支持音频文件吧。

本来是很有普适性的东西，何必搞得这样劳民伤财。

现在我们可以后知后觉地说，韦伯的忧虑似乎也不无道理。虽然并不是在“一周以后”，但 HTML5 最后确实增加了新的<video>和<audio>元素。

戴夫 · 拉吉特（Dave Raggett）回复韦伯的帖子：

没错！我们应该考虑到所有的图像文件类型，并且最好用上格式协商。蒂姆提到支持图片内的可点击区域，这一点也很重要。

1993 年晚些时候，戴夫 · 拉吉特提出了 HTML+，作为 HTML 标准的一次进化。这项提议从未付诸实施，取而代之的是 HTML 2.0 。HTML 2.0 是一个“复古规范“，就是说它规范化了那些已经很常用的功能特性：“这份规范汇集，澄清并正规化了一组特性，它们基本上涵盖了到 1994 年 6 月以前常用的那些 HTML 的功能。”

之后，在 HTML+的基础之上，戴夫编写了 HTML 3.0。除了一份被称为 Arena 的 W3C 内部参考实现以外，HTML 3.0 基本上算是没能发布。它随后被 HTML 3.2 所取代，后者同样是一份“复古规范”：“HTML 3.2 中增添了一些已经被广泛使用的特性，例如表格，applet 以及图文绕排，同时提供和现有 HTML 2.0 标准的完全后向兼容性。”

戴夫后来还参与起草了 HTML 4.0 规范，开发了 HTML Tity，并一直致力于 XHTML，XForms，MathML 以及其他现代 W3C 规范的编写。

再回到 1993 年，马克回答戴夫说：

> 其实，我们或许应该考虑一种通用的过程式图形语言，可以用来嵌入图标，图像或者文本等任何东西，同时这些东西都还可以附带任意超链接。有人注意到 Intermedia 在这方面的能力了么？

Intermedia 是布朗大学的一个与超文本相关的项目。它的开发工作进行于 1985 至 1991 年间，运行在 A/UX——一个面向早期 Macintosh 计算机的类 Unix 操作系统之上。

这个“通用过程式图形语言”的想法在后来的确成真了。现代的浏览器同时支持 SVG（含内嵌脚本的声明性标记语言）和<canvas>（一种过程化，直接模式的图形 API），不过，后者始于一个私有的扩展，后来才被 WHAT 工作小组给标准化了。

比尔 · 詹森（Bill Janssen）说：

> 其他具有这个（相当有价值的）概念的系统还包括 Andrew 和 Slate。Andrew 构建于“小插图”（insets）之上，它们各自有一些有趣的类型，如文本、位图、绘画、动画、信息、电子表格等。任意递归嵌入也被支持，这样任何类型的小插图都可以嵌入其他类型的小插图之中，只要对方支持嵌入。例如，一个小插图可以嵌入一个文本部件（text widget）中的任意位置，一个图形部件上的任意矩形区域内，或者一份电子表格上的任意单元格里。

“Andrew”是 Andrew 用户界面系统（Andrew UI System）的一个参考实现，不过当时它就只是简单地被称作 Andrew 项目。

与此同时，托马斯 · 法恩（Thomas Fine）提出了不同的想法：

> 以下是我的意见。在 WWW 中操作图像的最佳途径是使用 MIME。我敢肯定，在 MIME 中已经有了 PostScript 这个子类型，它可以很漂亮地处理图文混排。
>
> 但你想说，它是不可点击的？是的，你说得对。不过我认为这个问题已经有了答案，就是 Display PostScript。它虽然还不是标准 PostScript 的扩充，但这并不重要。可以定义一个锚命令（anchor command）来指定 URL，并使用当前路径作为一个按钮的闭合区域。PostScript 可以很好地处理路径，因此做出任意形状的按钮都不是问题。
>
> Display PostScript 是一种屏幕渲染技术，由 Adobe 和 NeXT 共同开发。

这一建议从未付诸实施，但这样的想法——即解决 HTML 中所存在问题的最好方式就是用另外的东西来完全取代之——仍然不时地产生。

1993 年 3 月 2 日，蒂姆 · 伯纳斯 · 李（Tim Berners-Lee）又回复说道：

> HTTP2 允许文档中包含任何类型——不仅仅是注册了的 MIME 类型，也包含用户声明了他能够处理的各种类型。所以人们可以试验。是的，我觉得 PostScript 和超文本就是一个例子。我不知道 Display PostScript 是否已经够用。我知道的是 Adobe 正在试图建立自己的基于 PostScript 的“PDF 格式”，其中可以含有链接，用它们自己的阅读器可以查看。
>
> 我想，一个针对锚点的通用覆盖语言（比如说基于 HyTime？）可以使得超文本和图形/视频标准分别演进，这对双方都好。
>
> 让 INCLUDE 取代 IMG 标签，并且使它可以用于任意文档类型。或者用 EMBED 标签，因为也许 INCLUDE 看上去像是 C++的语言成分——人们可能还会期望它提供可以用于分析的 SGML 源码——而这个标签压根不是干这事情的。

HyTime 是早期一种基于 SGML 的超文本文档系统。它在当时关于 HTML 以及后来关于 XML 的讨论中经常涌现。

蒂姆的这个<include>标记提案从未付诸实施，尽管你现在可以看到和它相呼应的<object>，<embed>标签和<iframe>元素。

最后，1993 年 3 月 12 日，马克 · 安德雷森重新激活了这个帖子：

> 再回到这个关于内嵌图像的帖子——我很快就要发布 Mosaic 0.10 版了，如先前所述，它支持内嵌 GIF 和 XBM 图像/位图的显示。[...]
>
> 我们暂时还不准备支持 INCLUDE/EMBED[...]所以大概会使用<IMG SRC="url">（不用 ICON，因为并非所有的内嵌图像都算是真正意义上的图标）。就目前而言，内嵌图像还不会具有明确的 content-type；不过接下来我们计划会对其提供支持（以及对 MIME 的一般性接纳）。实际上，我们读取图片的部分会即时判断图像格式，因此文件扩展名无所谓。

1.4 未曾间断的路线

An Unbroken Line

这段差不多有 17 年历史的对话分外令我着迷，它促成了一个 HTML 元素的诞生——而这个元素几乎被用在所有的网页上。请看：

- HTTP 依然存在。它成功地从 0.9 版进化到 1.0 版，接着是 1.1 版，并且还在继续进化（*http://www.ietf.org/dyn/wg/charter/httpbis-charter.html*）。
- HTML 依然存在。这种原始的，简陋的数据格式（它甚至不支持内嵌图片！）成功地演变成 2.0，3.2 和 4.0 版。HTML 的发展未曾间断。尽管经历了诸多曲折——它的进化树上长了许多“死亡分支”，着眼于创造标准的人们在这些地方有点过于超前了（标准的起草者和实现者没法跟上）——但尽管这样，在 2010 年的今天，那些 1990 年代的网页仍然可以在现代浏览器中呈现。我在一只最新潮的 Android 手机里打开一张过时的老页面，并不会遇到类似“请稍候，正在转换旧格式……”的提示。

- 一直以来 HTML 就是一个话题中心，参与者包括浏览器厂商、开发者、标准制定人士及那些喜欢谈论尖括号的家伙们。成功的 HTML 版本大都是属于“复古规范”，在追赶世界潮流的同时尝试将它带往正确方向。如果有人告诉你 HTML 应当保持“纯粹”（比如说无视浏览器厂商或者开发者），那他一定是在误导。HTML 从来没有纯粹过，所有曾经的这方面尝试都无奈地失败了，就像那些企图取代 HTML 的尝试。
- 1993 年的时候人们使用的那些浏览器到今天已经荡然无存了。Netscape Navigator 在 1998 年被放弃，随后被全面重写，变成了 Mozilla 套件，在后来又分支成为了 Firefox。Internet Explorer 低调地在 Microsoft Plus! for Windows 95 中登场，同一些桌面主题和一个弹珠游戏捆绑在一起；当然，这个浏览器现在要高调许多了。
- 1993 年时所用的一些操作系统至今还存在，不过都和现代 Web 毫不相关。今天，大多数人们都通过一台装有 Windows 2000 及以上版本或者某个 Linux 发行版的 PC，装有 Mac OS 的 Mac，或者像 iPhone 这样的手持设备来“体验”互联网。在 1993 年时，Windows 的版本还是 3.1（正在与 OS/2 竞争），Mac 还在运行系统 7，而 Linux 尚处在通过 Usenet 分发的时期。（想找乐子吗？去找个老龄 geek，跟他提“Trumpet Winsock”或者“MacPPP”。）
- 当年的一些老面孔，比如说曾积极参与“Web 标准”的制定并已经将近 20 年，现在仍然活跃。他们中的一些人甚至从 1980 年代起就已经参与建立 HTML 的前身。
- 这里说到前身……随着 HTML 和 Web 最终的流行，人们很容易忘记同时代的其他一些格式和系统——尽管 HTML 和 Web 曾受其启发。在阅读本章内容之前，你曾经听说 Andrew、Intermedia，以及 HyTime 吗？而 HyTime 并不是什么微不足道的大学研究项目；它是用于军事领域的，经过认证的一份 ISO 标准。很有来头的。可以访问这个链接了解详情：*http://www.sgmlsource.com/history/hthist.htm*。

但是上面这些都还没有回到我们最初那个问题：为什么会有<img>这个元素？为什么不是一个<icon>元素？或者<include>元素？为何不是一个带有 include 属性，或者多个 rel 属性值的超链接？为什么一定是<img>元素呢？很简单，因为马克 · 安德雷森发布了代码来处理这个元素，能给出东西的人就是赢家。

但这并不是说，只要谁给出代码就能当赢家；毕竟 Andrew、Intermadia 及 HyTime 都发布了各自的代码。能给出代码是获得成功的必要条件，但还不是充分条件。而我当然也不是说，在标准诞生之前就发布代码是最佳解决方案。马克的<img>元素没有指定常用图形格式；没有定义文本如何绕排；也不支持面向老旧浏览器的替代文本或者默认内容。即使 17 年后的今天，我们仍然在内容侦测这个议题上纠结，它仍然是一个安全漏洞的疯狂来源。这一切都追溯到浏览器大战，再一路回到 1993 年 2 月 25 日这天：马克 · 安得雷森随手写

下：“MIME，也许吧”，然后就发布了他的代码。

1.5 HTML 发展史：从 1997 到 2004 年

A Timeline of HTML Development from 1997 to 2004

1997 年 12 月，万维网联盟（W3C）发布了 HTML 4.0，接着立刻就解散了 HTML 工作小组。不到两个月后，一个单独的 W3C 工作小组发布了 XML 1.0。短短的三个月之后，针对“W3C 已经放弃 HTML 了吗”这个问题，W3C 举办了一个研讨会，主题是“塑造 HTML 的未来”，并给出回答：

> 经过讨论，与会者一致认为，进一步扩展 HTML 4.0 将会比较困难，同样地，将 4.0 转化为一个 XML 应用程序也很困难。所建议的用来打破这些限制的方法是，在一套 XML 标签组的基础之上，创造崭新的下一代 HTML。

W3C 重新成立了 HTML 工作小组来创建这“一套 XML 标签组”。1998 年 12 月，小组成员们的第一步就是起草一份临时规范，它只是简单地用 XML 重新表达 HTML，而不增加任何新的元素或属性。这份规范后来被称为“XHTML 1.0”。它定义了一种用于表示 XHTML 文档的新 MIME 类型，`application/xhtml+xml`。然而，为了使得已有 HTML 4 页面的迁移工作更加容易，这份规范的附录 C 中又“总结了一些设计指南，专门针对那些希望自己的 XHTML 文档也能在 HTML 用户代理下正确呈现的开发者们”。其中说道，XHTML 网页依然可以以 `text/html` 这个 MIME 类型来发布。

下一个目标是 Web 表单。1999 年 8 月，同一个 HTML 工作小组发表了 XHTML 扩展表单（extended forms）的初稿。小组成员们在草案文档的第一句话里就设定了他们的期望（*http://www.w3.org/TR/1999/WD-xhtml-forms-req-19990830#intro*）：

> 经过慎重考虑，HTML 工作小组已决定，下一代表单对那些为早期版本的 HTML 所设计的浏览器将不提供后向兼容性。我们的目标是要提供一个干净的全新表单模型（“XHTML 扩展表单”），它基于一组精心制定的需求。本文档重所描述的需求建立在一个非常广泛的与表单应用程序相关的经验之上。

几个月后，“XHTML 扩展表单”被重新命名为“XForms”，并成立了自己专门的工作小组。该小组与 HTML 小组并行工作，最终在 2003 年 10 月发布了 XForms 1.0 的第一版规范。

同时，随着向 XML 过渡的完成，HTML 工作小组的成员们将他们的目光锁定在创建“下一代的 HTML”上面。2001 年 5 月，他们发布了 XHTML 1.1 规范的第一版，其中只增添了一些 XHTML 1.0 基础之上的细小特性，不过却消除了那个“附录 C”漏洞。从这个 1.1 版开始，所有的 XHTML 文档都必须以 `application/xhtml+xml` 这个 MIME 类型发布。

1.6 你所知道的关于 XHTML 的一切都是错误的

Everything You Know About XHTML Is Wrong

MIME 类型为什么很重要？为什么我总是反复提到它？很简单：德拉科式（严格）的错误处理。浏览器对 HTML 从来都是很“宽容”的。即使你忘记了在 HTML 页面中写上 <title>，浏览器还是会显示页面，尽管<title>元素在任何版本的 HTML 中都是不可或缺的成分。有一些标签不允许被嵌在另一些标签之内，但是如果你确实这样做了，浏览器还是会以某种方式处理好一切，而不会给出错误信息。

正如你所想到的，因为格式错误的 HTML 仍然可以在浏览器中显示，所以导致了开发者们写出错误的页面，大量的错误页面。据估计，今天的互联网上超过 99%的 HTML 页面中都有至少一个错误。但由于这些错误不会导致浏览器显示可见的错误信息，从来没有人修复它们。

W3C 认为这是 Web 的一个根本性问题，并决定着手纠正。1997 年发布的 XML 打破了对用户的宽容传统，强制要求所有使用 XML 的程序都必须将所谓的“格式完好”错误当作致命问题对待。这个每当遇到一个错误程序就立刻全面终止的概念后来被称为“德拉科式错误处理”，它源自希腊领导人德拉科（Draco），在此人的法律中即便较轻的违法行为也会被判处死刑。当 W3C 将 HTML 用 XML 的词汇表重构时，负责的人强制要求所有那些以新的 application/xhtml+xml 这个 MIME 类型发布的文档都必须遵循这个严格的错误处理规定。如果 XHTML 页面上出现哪怕只有一个错误，Web 浏览器也将别无选择，只能停止继续处理并给用户显示一个报错信息。

这个想法没有受到普遍的欢迎。由于多达近 99%的页面都存在错误，用户获得错误提示的可能性几乎一定存在，加之 XHTML 1.1 和 1.0 相比所带来的新功能还并不值得为此付出那么大代价，Web 开发者们基本上选择了忽略 application/xhtml+xml。但这并不意味着他们同样完全忽略了 XHTML。嗯，应该说绝对没有忽略。XHTML 1.0 规范的附录 C 给全世界的 Web 开发者们留下了一个漏洞：“使用 XHTML 的语法，但是将页面以 text/html 的 MIME 类型发布。”而这正是成千上万的 Web 开发者们所做的事情：他们将页面代码的语法“升级”为 XHTML，但是用 text/html 的 MIME 类型发布。

即使在今天，纵然许多网页声称它们是 XHTML 的——页面第一行以 XHTML doctype 开始，使用小写的标签，在属性值外加上引号，以及在空标签内加上结束斜线，比如
和<hr />——但其中却只有极小一部分是以 application/xhtml+xml 这个 MIME 类型发布的，而只有这个 MIME 类型才会引发 XML 的德拉科式错误处理。任何以 text/html 这个 MIME 类型发布的页面，不管它用的什么 doctype，语法或者编码风格，都将被一个“宽容”的 HTML 解析器所处理，悄无声息地忽略掉任何代码错误，绝不会给出警告，即便网页从技术角度看是破烂不堪的。

XHTML 1.0 里包含的这个漏洞在 XHTML 1.1 中被关闭，而从未定型的 XHTML 2.0 也延续了要求德拉科式错误处理的传统。这就是为什么有数十亿的网页声称是 XTHML 1.0 的，但只有很少数声称自己是 XHTML 1.1（或者 XHTML 2.0）的。那么，你真的在使用 XHTML 吗？检查 MIME 类型。（实际上，如果你不知道你所用的是什么 MIME 类型，我几乎可以保证你还在使用 `text/html`。）除非你使用 `application/xhtml+xml` 这个 MIME 类型来发布你的页面，否则你所谓的“XHTML”只是名义上的 XML 而已。

1.7 一个竞争愿景
A Competing Vision

在 2004 年 6 月，W3C 举行了关于 Web 应用程序和复合文档（Compound Documents）的研讨会。参会的包括多家浏览器厂商，Web 开发公司以及其他 W3C 成员的代表。一个由有关各方，包括 Mozilla 基金会和 Opera 软件组组成的小组，介绍了他们对 Web 未来的竞争愿景：一个现有 HTML 4 标准的进化，以支持面向现代 Web 应用程序开发者们的新特性（*http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html*）：

以下七条原则代表了我们认为是这项工作的最关键的要求：

后向兼容性，明确的迁移路径

Web 应用程序技术应当建立在开发者们已经熟悉的技术之上，包括 HTML，CSS，DOM 和 JavaScript。

基本的 Web 应用程序特性应当可以通过行为、脚本以及样式表在今天的 IE6 中实现，这样开发者们能有一个明确的迁移路径。任何那些无法参与当前高端市场的用户代理所实现的方案（无须借助二进制插件），都几乎不可能获得成功。

明确定义的错误处理

Web 应用程序中的错误处理必须被定义到足够的细节层次，用户代理（UA）无须再重新设计自己的错误处理机制或者通过逆向工程使用其他用户代理的机制。

开发方面的错误不应当暴露给用户

规范必须针对每个可能的错误场景指明确切的错误恢复行为。大部分的错误处理应当被设计为优雅错误恢复式（如同 CSS），而不是明显和灾难性的失败式（如同 XML 那样）。

实际应用

进入到 Web 应用程序规范中的每个特性都要由实际的用例来验证。反之则不一定成立：针对每个用例并不一定需要增加一个新的特性。

用例最好是基于一个实际的网站，其中开发者曾经不得不用一些差劲的方案来勉强解决问题。

这里可以使用脚本

但如果某处使用声明性标记代码更加方便的话，就应当避免脚本。脚本应当是和设备及表现无关的，除非是用于特定设备上（例如用在 XBL 中）。

应当避免针对特定设备的功能区分

开发者应当能够在同样 UA 的桌面和移动版之中用到同样的特性。

开放的过程

Web 发展于一个开放的环境，它也从中受益。Web 应用程序将是 Web 的核心，它的发展也应当位于开放环境中。邮件列表、邮件存档和规范草案都应该对公众开放。

在一个非正式投票中，参会者被问到，“W3C 是否应该开发 HTML 和 CSS 的描述性扩展以及 DOM 的命令式扩展，用以满足中级 Web 应用程序的需求，而不是复杂的，全功能的操作系统级 API？”投票结果是 11 票赞成 8 票反对。在研讨会的总结中（*http://www.w3.org/2004/04/webapps-cdf-ws/summary*），W3C 成员们写道：“目前，W3C 不打算把精力和资源投入第三次非正式投票的主题——针对 Web 应用程序的 HTML 和 CSS 扩展，那些各工作小组现在正在进行的开发工作除外。”

面对这一决定，那些提议发展 HTML 和 HTML 表单的人们只剩下两种选择：放弃，或者在 W3C 之外继续他们的工作。他们选择了后者，注册了 whatwg.org 域名，接着在 2004 年 6 月，WHAT 工作小组诞生了。

1.8 WHAT 工作小组？

What Working Group?

到底什么是 WHAT 工作小组？我会让它自我解释（*http:www.whatug.org/news/start*）：

> Web 超文本应用程序技术（Web Hypertext Application Technology，缩写是 WHAT）工作小组是一个松散的，非官方的，开放的协作组织，由 Web 浏览器厂商和有关各方组成。该小组致力于发展基于 HTML 及其相关技术的规范，以减轻可互操作的 Web 应用程序的部署的困难度，并意图将成果提交到标准组织。然后所提交的成果可以作为在标准中正式扩展 HTML 的工作的基础。
>
> 本论坛的建立承接自数月来通过私人邮件所进行的对这些技术的规范制定。目前为止的工作焦点是扩展 HTML4 表单，以支持开发者们所要求的特性，并且不会破坏对现有内容的后向兼容性。这个小组的建立是为了确保这些规范的未来发展将会是完全开放的，这基于一个面向公众开放存档的邮件列表。

这里的关键词是“不破坏后向兼容性”。XHTML（去掉附录 C 漏洞）是和 HTML 不兼容的。它要求一个全新的 MIME 类型，还强制对所有具有这个 MIME 类型的内容使用德拉科式错误处理。XForms 也和 HTML 表单不兼容，因为它只能用在以新的 XHTML MIME 类型发布的文档中，意味着 XForms 也强制使用德拉科式错误处理。所有这些都涉及了 MIME。

为了不抹杀十多年以来在 HTML 上的成果，以及不让 99%的现有网页变得无法使用，WHAT 工作小组决定采用不同的方法：将浏览器实际所用的那些“宽容”的错误处理算法文档化。Web 浏览器一直都对 HTML 中的错误宽容对待，但是从来没有人费心去记录下它们到底是怎么做的。NCSA Mosaic 有自己的算法来应对有错误的页面，Netscape 也试图做到同样的事情与之媲美。然后 Internet Explorer 试图媲美 Netscape。再然后，Opera 和 Firefox 也试图媲美 IE。再然后 Safari 试图媲美 Firefox。以此类推，一直到今天。一路走来，开发者们耗费了无数时间来让自己的产品与竞争对手的产品相兼容。

这听起来像是近乎疯狂的工作量，实际上也是如此。更确切地说，早就是这样了。尽管花费了好几年时间，但 WHAT 工作小组还是成功地将“如何以兼容现有 Web 内容的方式解析 HTML”文档化了（除去一些个别情况）。最终的算法中完全没有强制 HTML 停止处理并显示错误信息的步骤。

在这些逆向工程进展的同时，WHAT 工作小组也在悄悄进行着其他一些工作。其中之一是一份规范，最初被称作 Web Forms 2.0，它给 HTML 表单新增了一些控件类型。（你将在第 9 章中了解 Web 表单的更多信息。）另一个是一份规范草案，称为“Web Application 1.0”，其中包括重要的新特性，比如一个支持直接模式绘图的画布（canvas，详见第 4 章）以及对音频和视频的无须插件的原生支持（详见第 5 章）。

1.9 回到 W3C

Back to the W3C

数年时间之中，W3C 和 WHAT 工作小组基本上互相无视对方的存在。WHAT 小组着眼于 Web 表单和新的 HTML 特性，W3C 的 HTML 工作小组则忙于 XHTML 的 2.0 版。到了 2006 年 10 月，WHAT 小组明显已经取得了重大成果，而 XHTML 2 还煎熬在草稿之中，没有任何主流浏览器为之提供实现。2006 年 10 月中，W3C 的创始人蒂姆·伯纳斯·李宣布，W3C 将和 WHAT 小组合作来开发下一代 HTML：

> 有些东西，过若干年再来看，会更加清楚。HTML 应当逐步进化。企图让全世界一下子切换到 XML——包括给属性值加上引号，给空标签加上结束斜线以及使用命名空间——这行不通。大多数 HTML 开发人员不为所动，主要是因为浏览器并

没有抱怨。一些大型社区确实转换过去了，并享受着结构良好的系统带来的甜头，但并不是所有人。渐进式地维护 HTML 是重要的，当然，持续不断地最终过渡到良好结构的世界也同样重要，并且要在这个世界里发展出更多力量。

合作的计划是成立一个全新的 HTML 小组。和以往的那个不同，这一个将致力于渐进地同时改善 HTML 和 XHTML。它有不同的负责人员。它的工作会同时涉及 HTML 和 XHTML。该小组将得到许多人的大力支持，包括浏览器厂商。

此外也会有工作涉及到表单。这是一个复杂的领域，因为现有的 HTML 表单和 XForms 都是表单语言。HTML 表单无处不在，XForms 也拥有许多实现和用户。同时，Webforms 提案也给出了对 HTML 表单很有意义的扩展。根据 WebForms 所说，该计划是扩充 HTML 表单的内容。

新组建的 W3C HTML 工作小组的最初几个决定之一就是将“Web Applications 1.0”更名为“HTML5。”于是我们来到了这里，开始试水 HTML5。

1.10 后记
Postscript

2009 年 10 月，W3C 解散了 XHTML 2 工作小组，并发表声明，解释这一决定（*http://www.w3.org/2009/06/xhtml-faq.html*）：

当 W3C 在 2007 年 3 月宣布成立 HTML 和 XHTML 2 工作小组时，我们表示会继续密切留意 XHTML 2 所拥有的市场。W3C 重视向社区发出关于 HTML 的未来的明确信号。

尽管我们认可 XHTML 2 工作小组多年来所贡献的价值，但在经过讨论后，W3C 管理层决定到 2009 年底解散这个小组，不再继续设立。

交付出东西的人才是赢家。

1.11 扩展阅读
Further Reading

- “Web 的历史”（The History of the Web：*http://esw.w3.org/topic/HTML/history*），Ian Hickson 所撰写的草稿，有点年头了
- “HTML/ 历史”（HTML/History：*http://hixie.ch/commentary/web/history*），由 Michael Smith，Henri Sivonen 等人撰写
- “HTML 简史”（A Brief History of HTML：*http://www.atendesigngroup.com/blog/brief-history-of-hlml*），由 Scott Reynen 撰写

第 2 章

HTML5 特性检测

Detecting HTML5 Features

2.1 引言

Diving In

你或许会问："老的浏览器都不支持 HTML5，我怎么才能开始使用它呢？"，其实这个问题本身就带有误解。HTML5 只是一些独立特性的集合，因此你不能检测浏览器是否支持"HTML5"，因为这没有任何意义。但是你可以分别检测浏览器是否支持诸如"画布（canvas）"、"视频（video）"、"地理位置(geolocation)"等 HTML5 特性。

2.2 检测技术

Detection Techniques

当浏览器渲染 Web 页面的时候，它会构造一个文档对象模型(Document Object Model，简称 DOM），用一个对象的集合来表示页面上的 HTML 元素。每一个诸如<p>、 、等页面元素都会被表示成 DOM 中不同的对象（除此之外，还有像 window 和 document 这些不和特定的页面元素绑定在一起的全局对象）。

所有的 DOM 对象都共享一些公共属性，但是有些对象会拥有其特有的属性。在支持 HTML5 特性的浏览器中，特性相关的 DOM 对象就会有特定的属性。从这些 DOM 对象中你可以快速检测出哪些 HTML5 特性是被支持的。

有四种基本技术可以用于检测浏览器是否支持某种 HTML5 特性，从简单到复杂依次是：

1. 检测全局对象（诸如 window 或者 navigator）是否拥有特定的属性。以检测地理位置特性为例，参见第 24 页的"地理位置"一节。
2. 创建一个元素，然后检测该元素的 DOM 对象是否拥有特定的属性。以检测画布特性为例，参见第 16 页的"画布"一节。
3. 创建一个元素，然后检测这个元素的 DOM 对象是否拥有特定的方法，同时调用这个方法并检查它的返回值。以检测支持的视频格式为例，参见第 19 页的"视频格式"一节。

4. 创建一个元素，给这个元素的 DOM 对象设定特定的属性值，然后检查浏览器是否保留了该属性值。

以检测支持的<input>类型为例，参见本书第 25 页的“输入框类型”一节。

2.3 Modernizr：一个 HTML5 特性检测库

Modernizr: An HTML5 Detection Library

Moderniz 是一个基于 MIT 许可证书发布的开源 JavaScript 类库，用于检测浏览器是否支持 HTML5 及 CSS3 特性。在本书撰写的时候，它的最新版本是 1.1 版，建议你使用它最新的版本。使用它的时候，只要将下面高亮的<script>元素放到页面顶部就好：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Dive into HTML5</title>
  <script src="modernizr.min.js"></script>
</head>
<body>
  ...
</body>
</html>
```

Modernizr 是自动运行的，无须调用诸如 modernizr_init()之类的初始化方法。当它运行的时候，会创建一个名为 Modernizr 的全局对象，这个对象为每一个可检测的特性都创建了一个对应的布尔类型的属性。例如：如果你的浏览器支持画布 API（参见本书第 4 章），那么 Modernizr.canvas 的属性值就为 true，反之，则为 false：

```
if (Modernizr.canvas) {
  // 开始画些什么吧!
} else {
  // 浏览器没有提供原生 canvas 支持 :(
}
```

2.4 画布

Canvas

HTML5 中定义<canvas>元素为：“它是依赖分辨率的位图画布，你可以在 canvas 上面绘制任意图形，甚至加载照片。”在网页中，一个 canvas 就是一个矩形区域。你可以在这个区域内通过 JavaScript 画你想画的任何东西。HTML5 标准中定义了一系列的 canvas API，用于绘制简单图形、定义路径、创建渐变及应用图像变换。

检测浏览器是否支持 canvas API 可以使用检测方法 2（参见第 15 页“检测技术”一节）。如果浏览器支持 canvas API，那么被创建的<canvas>元素对应的 DOM 对象就会拥有 getContext()这个方法（参见第 58 页“简单的图形”一节），反之，该对象只会拥有一些公共属性而没有任何 canvas 特定属性。你可以使用下面的函数来做检测：

```
function supports_canvas() {
  return !!document.createElement('canvas').getContext;
}
```

supports_canvas()方法中首先创建一个虚拟的<canvas>元素：

```
return !!document.createElement('canvas').getContext;
```

这个元素永远不会被附加到页面上，因此对于用户来说它永远是不可见的。它只是浮游在内存中无处可去也无所事事，好比平静湖面上的一叶孤舟。

创建完虚拟的<canvas>元素之后，紧接着测试该元素是否拥有 getContext()方法，这个方法只有当浏览器支持 canvas API 时才会存在：

```
return !!document.createElement('canvas').getContext;
```

最后，用双重否定来强制让这个检测方法返回一个布尔值：

```
return !!document.createElement('canvas').getContext;
```

这个函数能够检测出浏览器是否支持大多数的 canvas API，包括：图形（参见第 58 页“简单的图形”一节）、路径（参见本书第 61 页“路径”一节）、渐变（参见第 67 页“渐变”一节）和模式等。但它不会检测出类似 explorercanvas 这样的在 IE 中模拟实现 canvas API 的第三方库（参见第 73 页“IE 怎么办？”）。

如果不想自己写这个方法的话，可以使用 Modernizr 库（前一节提到的）提供的方法检测浏览器是否支持 canvas API：

```
if (Modernizr.canvas) {
  // 开始画些什么吧!
} else {
  // 浏览器没有提供原生 canvas 支持 :(
}
```

有一项专门针对 canvas 文本 API 的检测，将在下一节介绍。

2.5 画布文本

Canvas Text

浏览器支持 canvas API 并不意味着它也一定支持 canvas 文本 API，原因是 canvas API 一直在不断完善中，绘制文本的函数后来才被纳入规范。一些浏览器在文本 API 最终确定之前就引入了对 canvas 的支持。

要检测浏览器是否支持 canvas 文本 API，同样可以使用检测方法 2（参见第 15 页“检测技术”一节）。如果浏览器支持 canvas API，被创建的<canvas>元素对应的 DOM 对象会拥有 getContext()方法（参见本书第 58 页“简单的图形”一节），反之，该对象只会拥有一些公共属性而没有 canvas 特定属性。你可以使用下面的函数来检测：

```
function supports_canvas_text() {
  if (!supports_canvas()) { return false; }
  var dummy_canvas = document.createElement('canvas');
  var context = dummy_canvas.getContext('2d');
  return typeof context.fillText == 'function';
}
```

这个函数开始先使用上一节中介绍的 supports_canvas()方法来检测浏览器是否支持 canvas API：

```
if (!supports_canvas()) { return false; }
```

如果你的浏览器不支持 canvas API，那么它肯定不支持 canvas 文本 API。

接下来，创建一个虚拟的<canvas>元素并获取绘图上下文。这个操作一定会成功，因为在 supports_canvas()函数中已经检测过 canvas 对象上存在的 getContext()方法：

```
var dummy_canvas = document.createElement('canvas');
var context = dummy_canvas.getContext('2d');
```

最后，检测之前取得的绘图上下文是否拥有 fillText()方法，如果有，那就表示当前浏览器支持 canvas 文本 API：

```
return typeof context.fillText == 'function';
```

同样的，如果不想自己写这个方法的话，可以使用 Modernizr 库（参见本书第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持 canvas 文本 API：

```
if (Modernizr.canvastext) {
  // 让我们画些文字吧!
} else {
  // 浏览器没有提供原生 canvas 文本支持：(
}
```

2.6 视频
Video

HTML5 标准中定义了一个新的元素：<video>，用来将视频内嵌到 Web 页面中。在过去，要想在页面中内嵌视频，没有诸如 Apple 的 Quick Time 或者是 Adobe Flash 播放器这些第三方插件的话根本是不可能的。

<video>元素的可用性有意设计为不需要任何脚本来检测。你可以指定多个视频文件，支持 HTML5 video 的浏览器会选择一个它支持的视频格式[1]来进行播放。

不支持 HTML5 video 的浏览器会完全忽略<video>元素，在这种情况下可以指定用第三方插件来播放视频。Kroc Camen 正是利用这一点提出了一个被称为“全民视频（Video for Everybody!）”的视频播放解决方案，该方案提出：对于支持 HTML5 video 的浏览器就使用该特性来播放视频，否则就使用第三方的插件来播放视频。此方案完全不用 JavaScript，并且在包括移动终端浏览器在内的几乎所有的浏览器上都工作得很好。

但是，如果你想对视频做更多的操作而不仅仅是简单把视频内嵌到页面上并播放的话，就需要使用 JavaScript。检测浏览器是否支持 HTML5 video 可以使用检测方法 2（参见第 15 页“检测技术”一节）。如果浏览器支持 HTML5 video，被创建<video>元素对应的 DOM 对象会有一个名为 canPlayType()的方法，反之，该对象只会拥有一些所有元素都有的公共属性。你可以使用下面的函数来检测：

```
function supports_video() {
  return !!document.createElement('video').canPlayType;
}
```

如果不想自己写这个方法的话，可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持 HTML5 video：

```
if (Modernizr.video) {
  // 播放某个视频!
} else {
  // 浏览器不提供 video 原生支持：(
  // 检查 QuickTime 或 Flash 可否顶上
}
```

有一项专门针对浏览器可以播放何种视频格式的检测，将在下一节介绍。

2.7 视频格式
Video Formats

视频格式好比语言文字。一份英文报纸和一份西班牙文报纸可能报导的是相同的内容，但是如果你只看得懂英文的话，那么只有英文的那份对你才是真正有帮助的。同样，对于一个视频而言，浏览器要想播放它，必须要理解视频是用哪种“语言”编写的。

[1]注：了解更多不同的视频格式请参见《视频编码概述》一文的第一部分：“容器格式”（*http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats*）和 第二部分：“有损视频编码”（*http://diveintomark.org/archives/2008/12/19/give-part-2-lossy-video-codecs*）。

编写视频的语言被称为：“编码算法（codec）”———一种用来将视频编码成比特流的算法。目前世界上有许许多多的编码算法，应该使用哪一个呢？遗憾的是，各种浏览器没有达成一致，使用统一的视频编码算法。但好在分歧已缩减到两种算法上：一种是 Safari 和 iPhone（如果你使用“视频无处不在”的解决方案的话还有 Adobe 的 Flash）遵循的需要收费（因为专利的缘故）的编码算法，另外一种是诸如 Chromium 和 Mozilla Firefox 这类开源浏览器遵循的免费的编码算法。

要检测浏览器是否支持某种视频格式，可以使用检测方法 3（参见第 15 页“检测技术”一节）。如果浏览器支持 HTML5 video，被创建的<video>元素对应的 DOM 对象会有一个叫 canPlayType()的方法，利用这个方法可以知道浏览器是否支持某种特定视频格式。

这个函数检测 Mac 和 iPhone 支持的受专利保护的格式：

```
function supports_h264_baseline_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

这个函数开始先使用上一节中介绍的 supports_video()方法来检测浏览器是否支持 HTML5 video：

```
if (!supports_video()) { return false; }
```

如果浏览器不支持 HTML5 video，那么自然也就没有支持什么视频格式的问题了。

紧接着，该方法创建了一个虚拟的<video>元素，然后调用它的 canPlayType()方法，这个操作一定会成功，因为 supports_video()函数已经检测过了：

```
var v = document.createElement("video");
return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
```

一种“视频格式”实际上组合了好几种不同的东西。用技术术语来说，是在问浏览器是否能播放封装在 MPEG-4 容器里的“H.264”编码格式的视频和 AAC LC 编码格式的音频。[2]

调用 canPlayType()方法不会返回布尔值。因为视频格式非常复杂，所以这个方法返回一个字符串：

```
"probably"
```

表示浏览器有充分把握可以播放此格式。

[2]注：我会在第 5 章详细解释这些概念，如果你想了解更多，可以阅读《视频编码概述》这篇文章。

"maybe"

表示浏览器有可能可以播放此格式。

""（空的字符串）

表示浏览器确定无法播放此格式。

第二个函数用于检测被 Mozilla Firefox 和其他一些开源浏览器支持的开放视频格式。检测过程大致相同，唯一的不同点是给 canPlayType()方法传入的参数不同，用技术术语来说的话，你是在问浏览器是否能播放封装在 Ogg 容器内的“Theora”编码格式的视频和“Vorbis”编码格式的音频：

```
function supports_ogg_theora_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/ogg; codecs="theora,vorbis"');
}
```

最后，要提一下 WebM（*http://www.webmproject.org*），它是一种新的开源（不受专利约束）视频编码格式，并会在 Chrome，Firefox 和 Opera 这些主流浏览器的下一个版本得到广泛支持。你可以利用同样的技术来检测浏览器是否支持 WebM 视频格式：

```
function supports_webm_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

如果不想自己写这些函数的话，可以使用 Modernizr 库提供的方法检测浏览器是否支持上述提到的各种 HTML5 视频格式（注意：Modernizr 还不支持检测 WebM 视频格式）：

```
if (Modernizr.video) {
  // 可以播放视频了，但播放哪一种格式呢?
  if (Modernizr.video.ogg) {
    //尝试在 Ogg 容器中使用 Ogg Theora+Vorbis
  }else if (Modernizr.video.h264) {
    //尝试在 MP4 容器中使用 H.264 视频+AAC 音频
  }
}
```

2.8 本地存储

Local Storage

HTML5 本地存储（*http://dev.w3.org/html5/webdatabase/*）允许网站把信息存储到本地的计算机上，以便在以后需要时获取。这个概念和 cookie 相似，但它是为了更大容量的存储设计的。Cookie 的大小是受限的，并且每次请求一个新页面时，cookie 信息都会被发送回服务器（不仅耗费额外的时间，而且占用宝贵的带宽）。而 HTML5 本地存储将信息存储在用户本地计算机上，网站可以在页面加载完毕后通过 JavaScript 获取。

专家答疑

问：本地存储真的是 HTML5 标准的一部分吗？为什么它有单独的一份标准？

答：简单来说，本地存储是 HTML5 标准的一部分。稍微详细一点的解释是：本地存储原先是作为 HTML5 标准中的一部分，但后来部分 HTML5 工作组的成员抱怨 HTML5 标准太庞大了，于是将本地存储剥离出来成为一份单独的标准。这听起来就好像是为了减少卡路里总量而把一块大馅饼分成很多小块……很古怪吧？欢迎来到古怪的标准世界！

检测浏览器是否支持本地存储可以使用检测方法 1（参见第 15 页“检测技术”一节）。如果浏览器支持 HTML5 本地存储，全局 window 对象会有一个 localStorage 属性，反之，window 对象的该属性值为 undefined。你可以使用这个函数来检测本地存储是否被支持：

```
function supports_local_storage() {
  return ('localStorage' in window) && window['localStorage'] !==null;
}
```

如果不想自己写这个方法的话，可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持 HTML5 本地存储：

```
if (Modernizr.localstorage) {
  // window.localStorage 存在!
} else {
  // 浏览器没有提供本地存储的原声支持 :(
  // 也许可以尝试使用 Gears 或者其他的第三方解决方案
}
```

注意：JavaScript 是大小写敏感的。Modernizr 库中的属性名称是 localstorage（全部小写），而 DOM 中的属性名称是 window.localStorage(大小写都有)。

专家答疑

问：HTML5 本地存储的安全性如何？是不是所有人都可以读取存储的数据？

答：任何物理上能访问你计算机的人都可以查看（或者修改）你 HTML5 的本地数据库。在浏览器中，任何网站都可以读取和修改它们自己存储的数据，但是不同站点的存储数据是不能相互访问的。这被称作“同源限制（same-origin restriction）（*http://bit.ly/bYiOQP*）”。

2.9 Web Workers

Web Workers

Web workers（*http://bit.ly/bYiOQP*）提供了一种标准的方式让浏览器能够在后台运行JavaScript。通过Web Worker，你可以产生多个“线程”，并让它们同时运行。（就像计算机能够同一时间运行多个应用程序一样）。这些“后台线程”可以在页面响应用户的滚屏、点击或者输入操作的同时做些诸如复杂的数学运算、发送网络请求或者操作本地数据库的事情。

检测浏览器是否支持Web Worker可以使用检测方法1（参见本书第15页“检测技术”一节）。如果浏览器支持Web Worker API的话，全局window对象会有一个名为Worker的属性，反之，window对象上该属性的值为undefined：

```
function supports_web_workers() {
  return !!window.Worker;
}
```

如果不想自己写这个方法的话，可以使用Modernizr库（参见本书第16页“Modernizr：一个HTML5特性检测库”）提供的方法检测浏览器是否支持Web Worker：

```
if (Modernizr.webworkers) {
  // window.Worker 存在!
} else {
  // 浏览器没有提供 web worker 的原生支持 :(
  // 也许可以尝试使用 Gears 或其他第三方解决方案
}
```

注意：JavaScript是大小写敏感的。Modernizr中的属性名称是“webworkers”（全部小写），而DOM中的属性名称是window.Worker（“Worker”中首字母“W”是大写）。

2.10 离线Web应用

Offline Web Applications

离线查看一个静态的Web页面很容易：连入互联网，载入这个Web页面，随后断开网络，开着车去一个僻静的度假别墅，然后悠闲地阅读它（为了节约时间，你也许希望省掉开车去别墅这一段）。可是对于像Gmail或者Google Docs这样的Web应用来说，怎么做到离线访问呢？感谢HTML5，它让所有人（而不是只有Google！）都可以构建一个能离线访问的Web应用程序。

离线Web应用首先要从在线Web应用开始。在第一次访问具备离线访问功能的Web站点时，Web服务器会告诉浏览器哪些文件是保证离线正常工作所必需的，这些文件可以是任意文件——HTML、JavaScript、图片甚至视频（参看第18页“视频”一节）。一旦浏览器下载了这些必需的文件之后，下次哪怕在没有网络的情况下也能正常访问该站点。浏览器会识别到当前是离线状态，并使用那些之前下载下来的文件。当你重新上线的时候，所有改动都会上传到远程Web服务器。

检测浏览器是否支持离线可以使用检测方法#1（参见本书第 15 页“检测技术”一节）。如果浏览器支持离线 Web 应用，全局的 window 对象上会拥有一个名为 applicationCache 的属性，反之，window 对象上该属性的值为 undefined：

```
function supports_offline() {
  return !!window.applicationCache;
}
```

如果你不想自己写这个方法的话，可以使用 Modernizr 库（参见本书第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持离线 Web 应用：

```
if (Modernizr.applicationcache) {
  // window.applicationCache 存在!
} else {
  // 浏览器没有提供原生的离线支持 :(
  // 也许可以尝试使用 Gears 或者其他第三方解决方案
}
```

注意：JavaScript 是大小写敏感的。Modernizr 库中这个属性名称是 applicationcache（全部小写），但是 DOM 中的属性名称是 window.applicationCache（大小写都有）。

2.11 地理位置

Geolocation

地理位置能够神奇地定位出你正待在世界上的什么地方，并且在你允许的情况下把该位置信息分享给你信任的人。定位的方法有很多——IP 地址、利用基站获取手机无线网络的接入位置、通过能够利用卫星定位获得经纬度信息的 GPS 设备。

专家答疑

问：地理位置是 HTML5 标准的一部分吗？为什么要介绍它？

答：浏览器在实现对 HTML5 新特性支持的同时，也纷纷加入了对地理位置特性的支持。严格地说，地理位置特性并不属于 HTML5 标准的一部分，它由地理位置工作组（Geolocation Working Group）（*http://www.w3.org/2008/geolocation/*）负责制定标准，这个工作组和 HTML5 工作组没有关系。尽管如此，在本书中我还是打算介绍它，因为地理位置特性也是当下整个 Web 的一部分。

检测浏览器是否支持地理位置特性可以使用检测方法#1（参见第 15 页“检测技术”一节）。如果浏览器支持地理位置 API 的话，全局的 navigator 对象上会有一个名为 geolocation 的属性，反之，navigaor 对象上该属性的值为 undefined：

```
function supports_geolocation() {
  return !!navigator.geolocation;
}
```

如果不想自己去写这个方法的话，可以使用Modernizr库（参见本书第16页“Modernizr：一个HTML5特性检测库”）提供的方法检测浏览器是否支持地理位置API：

```
if (Modernizr.geolocation) {
  // 让我们找出你在哪?
} else {
  //  浏览器没有提供原生的地理位置支持 :(
  //  也许可以尝试使用Gears或者其他第三方解决方案
}
```

如果浏览器没有提供对地理位置 API 的原生支持，利用 Gears(*http://tools.goolge.com/gears/*)也能达到同样的目的。Gears是Google开发的开源浏览器插件，兼容Windows、Mac、Linux、Windows Mobile和Android平台。它为那些对本章中介绍的超酷特性没有提供原生支持的早期浏览器提供了第三方解决方案。其中一个提供支持的特性就是地理位置API。尽管它和navigator.geolocation API并不相同，但它实现了同样的目的。

还有一些绑定在特定移动手机平台上的地理位置API，这些平台包括BlackBerry、Nokia、Palm、OMTP BONDI等。

第6章会具体介绍如何使用这些不同的API。

2.12 输入框类型

Input Types

想必你对如何创建一个Web表单一定了如指掌：首先创建一个<form>元素，然后创建一些诸如：<input type="text">、<input type="password">这样的元素，最后再以一个<input type="submit">按钮来结束。

可是你也许不知道，HTML5标准中定义了很多的新输入框类型：

`<input type="search">`

表示搜索框，具体参见 *http://bit.ly/9mQt5C*。

`<input type="number">`

表示数字类型输入框，具体参见 *http://bit.ly/aPZHjD*。

`<input type="range">`

表示范围选择滑块，具体参见 *http://bit.ly/dmLiRr*。

`<input type="color">`

表示颜色选择器，具体参见 *http://bit.ly/bwRcMO*。

`<input type="tel">`

表示电话号码输入框，具体参见 *http://bit.ly/amkWLq*。

`<input type="url">`

表示网址输入框，具体参见 *http://bit.ly/cjKb3a*。

`<input type="email">`

表示 email 地址输入框，具体参见 *http://bit.ly/aaDrgS*。

`<input type="date">`

表示日期选择器，具体参见 *http://bit.ly/c8hL58*。

`<input type="month">`

表示月份输入框，具体参见 *http://bit.ly/cDgHRI*。

`<input type="week">`

表示星期输入框，具体参见 *http://bit.ly/bR3r58*。

`<input type="time">`

表示时间戳输入框，具体参见 *http://bit.ly/bfMCMn*。

`<input type="datetime">`

表示精确日期/时间戳输入框，具体参见 *http://bit.ly/c46zVW*。

`<input type="datetime-local">`

表示当地日期和时间输入框，具体参见 *http://bit.ly/aziNkE*。

检测浏览器是否支持新的输入框类型可以使用检测方法#4（参见第 15 页“检测技术”一节）。首先，创建一个虚拟的`<input>`元素：

```
var i = document.createElement("input");
```

`<input>`元素默认为“文本类型”，这一点被证明非常重要。

接下来将`<input>`元素的类型设置成你要检测的类型：

```
i.setAttribute("type","color");
```

如果浏览器支持此特定的输入框类型，那么设置的 type 属性值会被保留。反之，浏览器会忽略你设置的值，type 属性的值依然为“text”：

```
return i.type !== "text";
```

针对 13 种不同输入框类型的检测就意味着要写 13 个检测方法，如果不想自己去写这些方法的话，你可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持这些新的输入框类型。Modernizr 库重用一个`<input>`元素来高效检测 13 种输入框类型，然后构建了一个名为 Modernizr.inputtypes 的散列表，这个散列表中包含 13 个键值对，其中，键表示类型（即 HTML5 新增的输入框类型属性名），值是布尔值（true 表示支持该类型，false 则表示不支持该类型）：

```
if (!Modernizr.inputtypes.date) {
  // 浏览器没有提供<input type="date"> 原生支持 :(
  // 也许你需要自己用 Dojo 或 JqueryUI 来构造一个类似的控件
}
```

2.13 占位文本

Placeholder Text

除了新的输入框类型，HTML5 标准中还给现有的表单控件做了一些小改进，其中一个改进就是允许给输入框设置占位文本。占位文本会在输入框为空或者失去焦点的时候显示出来，一旦用户点击输入框（或者利用 Tab 键使其获得焦点），占位文本就会消失。如果你不太明白这是怎么回事，可以看看第 147 页“占位文本”一节里的截图。

检测浏览器是否支持占位文本可以使用检测方法 2（参见第 15 页“检测技术”一节）。如果浏览器支持占位文本的话，创建的<input>元素对应的DOM对象会有一个placeholder的属性（即使没有在<input>元素上显式地设置 placeholder 属性），反之，该 DOM 对象则不会有此属性：

```
function supports_input_placeholder() {
  var i = document.createElement('input');
  return 'placeholder' in i;
}
```

如果不想自己去写这个方法的话，可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持占位文本：

```
if (Modernizr.input.placeholder) {
  // 也许你已经看到占位文本了!
} else {
  // 占位文本还不被支持，退而使用脚本解决方案
}
```

2.14 表单自动聚焦

Form Autofocus

很多网站使用JavaScript脚本让Web表单的首个输入框自动获得焦点，例如Google.com首页会让搜索框自动获取焦点以便直接输入搜索关键字，而无须定位光标到搜索框里。这对于多数普通用户来说的确很方便，但是对于那些高级用户或者有特殊需求的用户来说却未必如此。比如你原想按下空格键来滚动页面，但是因为焦点已经被聚到了表单中的输入框，因此页面并不会如你所愿地滚动（而是会在输入框中输入空格）；再比如你在页面还在加载的时候把焦点聚焦到了其他输入框，你正想要输入什么的时候，网站的自动聚焦脚本可能“帮倒忙”地把焦点移动到原本设定自动对焦的输入框中，从而打断了你的输入，并导致文字输入到了错误的位置。

因为自动聚焦是用 JavaScript 脚本来实现的，因此可以使用一些小技巧来处理所有这些极端的情况，但那些不希望 Web 页面“偷”走焦点的人们就无计可施了。

为了解决这个问题，HTML5 标准中为所有的表单控件引入了一个 autofocus 属性。顾名思义，这个属性的意思就是将焦点自动聚焦到特定的输入框中。由于是直接采用标记而不是 JavaScript 实现这一功能，因此所有网站的行为保持一致。而且，浏览器提供商（或者扩展程序作者）也可以为用户提供一个禁用该功能的方式。

检测浏览器是否支持自动聚焦可以使用检测方法#2（参见第 15 页“检测技术”一节）。如果浏览器支持自动聚焦的话，创建的<input>元素对应的 DOM 对象会拥有一个 autofocus 的属性（那怕你没有在<input>元素上包含此属性也是如此），反之，该 DOM 对象就不会有此属性：

```
function supports_input_autofocus() {
  var i = document.createElement('input');
  return 'autofocus' in i;
}
```

如果不想自己去写这个方法的话，可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法检测浏览器是否支持自动聚焦：

```
if (Modernizr.input.autofocus) {
  // autofoucs 可以工作!
} else {
  // 浏览器不支持 autofocus，退而采用脚本解决方案
}
```

2.15 微数据

Microdata

微数据（*http://bit.ly/ckt9Rj*）是一种标准化的为 Web 页面提供额外语义的方式。例如，可以使用微数据来声明一张图片是被特定的创作公用许可证授权使用的。在第 10 章你还会看到，微数据可以用来标记一个“关于我”的页面。浏览器、浏览器扩展程序及搜索引擎能把微数据的标记转化成 vCard 形式（一种用于共享联系人信息的标准格式）。当然，你也可以定义属于自己的微数据词汇。

HTML5 微数据标准包含 HTML 标记（主要用于搜索引擎）和一系列的 DOM 方法（主要用于浏览器）。在页面中使用微数据标记不会带来什么损害，因为那顶多就是一些被适当放置的属性罢了，并且如果搜索引擎无法识别这些微数据属性，就会直接忽略掉。但是如果想要通过 DOM 访问和操作微数据，那就得确保浏览器支持微数据的 DOM API。

检测浏览器是否支持 HTML5 微数据 API 可以使用检测方法 1（参见第 15 页“检测技术”一节）。如果浏览器支持 HTML5 微数据 API 的话，全局 document 对象上会有一个 getItems()的方法，反之，该方法就不存在：

```
function supports_microdata_api() {
  return !!document.getItems;
}
```

Modernizr 目前还不支持检测微数据 API，所以必须用类似这样的一个函数来检测。

2.16 扩展阅读

Further Reading

标准文档：

- <canvas>元素（*http://bit.ly/9JHzOf*）。
- <video>元素（*http://bit.ly/a3kpiq*）。
- <input>元素类型（*http://bit.ly/akweH4*）。
- <input placeholder>属性（*http://bit.ly/caGl8N*）。
- <input autofocus>属性（*http://bit.ly/db1Fj4*）。
- HTML5 存储（*http://dev.w3.org/html5/webstorage/*）。
- Web Workers（*http://bit.ly/9jheof*）。
- 离线 Web 应用（*http://bit.ly/d8ZgzX*）。
- 地理位置 API（*http://www.w3.org/TR/geolocation-API*）。

JavaScript 库：

- Modernizr（*http://www.modernizr.com/*），一个 HTML5 特性检测库。
- geo.js（*http://code.google.com/p/geo-location-javascript*），一个封装地理位置 API 的 JavaScript 库。

其他一些文章和教程：

- 全民视频（*http://camendesign.com/code/video_for_everybody*）。
- 视频编码概述（*http://diveintomark.org/tag/give*）。
- 视频类型参数（*http://wiki.whatwg.org/wiki/Video_type_parameters*）。
- 本书附录。

第 3 章

从这一切的含义

What Does It All Mean?

3.1 引言

Diving In

本章将用到一个绝对没有任何错误的 HTML 页面，对其加以改进。它的一些部分会变得简短，一些部分会变得更长，而它整体将会有更好的语义性，变得不同凡响。

这就是我们的这个范例页面：*http://diveintohtml5.org/examples/blog-original.html*。了解它，活化它，爱上它。在一个新标签页里打开该页面，除非你至少点击了一次“查看源文件”，否则不要再继续读下去。

3.2 文档类型

The Doctype

从最顶部开始：

```
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

这就是 doctype。doctype 身后有着悠久的历史——以及一种黑色艺术。在 Internet Explorer 5 for Mac 的开发过程中，微软遇到了一个令人惊讶的问题。其即将推出的新版浏览器如此之多地改进了对标准的支持，以至于旧一些的网页不再能够正确呈现。或者说，它们是被正确渲染了（依据规范），但人们期望看到的却是错误的渲染结果。那些页面本身是依照当时占统治地位的浏览器——主要是 Netscape 4 和 Internet Explorer 4——的有问题的渲染行为（quirks）而编写的。IE5/Mac 却是如此先进，它事实上破坏了 Web。

微软想出了一个新奇的解决方案。在开始渲染页面之前，IE5/Mac 会先检查“doctype”，通常即是 HTML 源码的第一行内容（甚至在<html>之前）。老旧的网页（面向旧版本浏览器 quirks 模式的）通常根本没有 doctype。IE5/Mac 便像旧版本浏览器那样渲染这些网页。为了“激活”新的标准模式，网页作者必须在<html>之前加上正确的 doctype。

这个方案像野火一般蔓延，很快所有主要的浏览器都有了两种渲染模式："quirks 模式"和"标准模式"。当然，这是网络时代，事情很快就一发不可收拾。当 Mozilla 将要发布其浏览器的 1.1 版时，它发现有的网页尽管以标准模式渲染，它们实际上是依赖于某个特定的 quirk 模式。Mozilla 刚刚修正了自己的渲染引擎以消除这个 quirk，于是数以千记的网页在 Mozilla 浏览器中就变得支离破碎了。于是乎，所谓的"准标准模式"（almost standard mode）被创立了——这不是我杜撰的。

Henri Sivonen 在他的大作"使用 Doctype 激活浏览器的渲染模式"（Activating Browser Modes with Doctype）（*http://hsivonen.iki.fi/doctype/*）一文中总结了不同的浏览器模式：

Quirks 模式

在 Quirks 模式中，浏览器违反了 Web 格式规范，以避免那些依照 90 年代末期的流行做法编写的网页不至于被呈现得无法阅读。

标准模式

在标准模式中，浏览器尽力尝试用符合标准规范的方式去渲染页面文档。HTML5 里将这个模式称为"非 quirks 模式"。

准标准模式（Almost Standards Mode）

Firefox、Safari、Chrome、Opera（7.5 版起）和 IE8 也具有被称为"准标准模式"（almost standards mode）的渲染模式，其中以传统方式实现了表格单元格的竖直尺寸，而没有严格遵循 CSS2 规范。HTML5 中将这种模式称为"有限 quirks 模式"（limited quirks mode）。

注意：你应该阅读 Henri 的文章的其余部分，因为我在这里给出的内容经过大量简化。即使在 IE5/Mac 里，也还存在一些很老的 doctype，已经不被标准所考虑。随着时间的推移，能引发 quirks 模式的 doctype 的名单越来越长。我上次数的时候，有 5 个 doctype 会引发"准标准模式"，有 73 个会引发 quirks 模式。不过我可能漏掉了一些，并且我甚至不打算提到 Internet Explorer 8 在它的 4 种——没错，4 种！——不同的渲染模式之间切换的机制。这里有一个示意流程图：*http://hsivonen.iki.fi/doctype/ie8-mode.png*。杀死它，用火杀死它。

回到正题。我们说到哪了？啊，是的，doctype：

```
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

这就是 15 个能在所有现代浏览器里触发标准模式的 doctype 之一。它没有什么问题。如果你喜欢它，可以继续使用。或者你也可以把它改成 HTML5 的 doctype——更加简短美观，并且同样可以触发所有现代浏览器的标准模式。

这就是 HTML5 的 doctype：

```
<!DOCTYPE html>
```

就是这样。仅仅 15 个字符。非常简单，可以直接用键盘打出来而不用担心会敲错。

专家答疑

Doctype 必须位于 HTML 源文件的第一行。如果它前面还有东西——哪怕一个空行——某些浏览器都会认为该页面不具有 doctype。没有指明 doctype 的话，浏览器就会以 quirks 模式渲染页面。这是个很难捕捉到的错误。额外的空格在 HTML 中通常无所谓，所以开发者往往不会去注意它们，但在这种情况下它就非常重要了！

3.3 根元素

The Root Element

一个 HTML 页面包含一系列嵌套的元素。页面的整个结构就像一棵树。元素互相之间可以是“兄弟”，好比从同一个树干延伸出的两个分支。一些元素也可以是另一些元素的“子元素”，好比从较大分支延伸出小的分支。（另外的情况是，一个元素如果包含其他元素，它就被称为其直接子元素的“父节点”，以及它的非直接子元素的“祖先节点”。）不含有子元素的节点称为“叶节点”。最外层的元素，它是页面上的所有其他元素的祖先，被称为“根元素。”一个 HTML 页面的根元素始终是<html>。

在我们的示例页面（*http://diveintohtlm5.org/examples/blog-original.html*）中，根元素看起来是这样的：

```
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="en"
      xml:lang="en">
```

这个标记没有什么问题。同样，喜欢的话你就继续这么写。它是有效的 HTML5 标记。但是其中一些部分对于 HTML5 已经不再是必须的了，所以可以删除以再节省几个字节。

首先要讨论的是 xmlns 属性。这是 XHTML 1.0 的遗迹。它的意思是，在这个页面上的元素位于 XHTML 命名空间 *http://www.w3.org/1999/xhtml* 之中。但是 HTML5 的元素都是具有这个命名空间的，因此不必再显式写明了。不管有没有这个属性，HTML5 页面在所有现代浏览器中将会有一致的表现。

去掉 xmlns 属性后的根元素是这样的：

```
<html lang="en" xml:lang="en">
```

这里的两个属性 lang 和 xml:lang，都用于定义该 HTML 页面的所用语种。en 表示英语[1]。为什么定义同一个东西要用到两个属性呢？这是 XHTML 的遗迹。在 HTML5 中只有 lang 属性有效果。如果乐意，你当然可以保留 xml:lang 属性，但这么做必须确保它和 lang 的属性值相同。

> 为了简化和 XHTML 之间的互相迁移，在 HTML 文档中，开发者可以给没有命名空间的元素指定无前缀的属性，以及具有字面本地名称（literal localname）“xml:lang”的属性，但是要指定这样的属性，必须要先指定没有命名空间的 lang 属性，并且两个属性的值要相同（不区分大小写）。没有命名空间的属性，不管是没有前缀还是有字面本地名称“xml:lang”，都不会影响页面语种处理。

你是否准备去掉这个属性呢？现在根元素就是这样了：

```
<html lang="en">
```

这就是我要说的一切。

3.4 <head>元素

The <head> Element

根元素的第一个子元素通常是<head>。<head>元素包含元数据——关于网页本身的信息，而不是网页的主体。（页面的主体毫无疑问是包含在<body>元素中的。）<head>元素本身相当无聊，在 HTML5 里也没有变得有趣一点。藏在<head>元素里面的才是好东西。为此再回到我们的示例页面：

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>My Weblog</title>
  <link rel="stylesheet" type="text/css" href="style-original.css" />
  <link rel="alternate" type="application/atom+xml"
                        title="My Weblog feed"
                        href="/feed/" />
  <link rel="search" type="application/opensearchdescription+xml"
                        title="My Weblog search"
                        href="opensearch.xml" />
  <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

首先亮相的是：<meta>元素。

[1] 注：页面不是用的英语？没关系，可以从 *http://www.w3.org/International/questions/qa-choosing-language-tags* 找到你所使用语言的代码。

3.4.1 字符编码

当提到“文字”的时候，或许你想到的是“我在我的电脑屏幕上所看到的字母和符号。”但其实电脑所处理的并不是字母和符号，而是位和字节。每个你能在电脑屏幕上看到文字本质上是以某种字符编码存储着。字符编码有很多种，有些专门面向特定语言，比如俄文、中文或英文，有些则可用于多种语言。粗略地讲，字符编码为你在屏幕上所看到的东西，以及电脑实际存储在内存和磁盘上的东西之间提供了一种映射。

在现实中，这要复杂得多。有一些字符在多种编码中都很常见，但是每种编码却可能使用不同的字节序列来实际存储这些字符。所以，你可以将字符编码看做某种给文字解密的密钥。当有人给你一段字节序列并声称它是一段“文字”，你就需要知道他所用的字符编码，这样才能将字节解码成字符并显示出来（或者做其他任何处理）。

那么，浏览器是如何实际确定 Web 服务器发来的字节流的编码呢？很高兴你问到。如果熟悉 HTTP 头信息，你可能见过：

```
Content-Type: text/html; charset="utf-8"
```

简单地说，这表示 Web 服务器认为它在向你发送一个 HTML 文件，并且认为这个文件使用的是 UTF-8 字符编码。不幸的是，在整个宏伟的万维网世界，没有几个开发者能够有 HTTP 服务器的控制权。考虑一下 Blogger（*http://www.blogger.com*）的情况：文章内容是由个人用户提供的，但是服务器由 Google 运营。因此，HTML 4 中提供了一个方法来指定在 HTML 文件本身的字符编码。你可能也见过这个：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

简单地说，这表示网页开发者认为她的页面使用的是 UTF-8 字符编码。

以上两种方法在 HTML5 中仍然有效。使用 HTTP 头信息是首选，并且它会覆盖<meta>标签的作用。但并不是每个人都可以设置 HTTP 头信息，因此<meta>标记仍然有用。事实上，在 HTML5 中这个变得稍微更容易了一点。现在看来就是这样：

```
<meta charset="utf-8" />
```

这适用于所有浏览器。而该缩略语法从何而来呢？以下是我能找到的最佳解释（*http://lists.w3.org/Archives/Public/public-html/2007Jul/0550.html*）：

> <meta charset="">属性组合存在的理由是，许多 UA 已经实现了它，因为人们往往习惯不加上引号，例如：
>
> <META HTTP-EQUIV=Content-Type CONTENT=text/html; charset=ISO-8859-1>

如果你不相信浏览器已经这么做了的话，甚至还有一些<meta charset>的测试用例（*http://simon.html5.org/test/html/parsing/encoding*）可以说明。

专家答疑

问：我从来没有用过什么怪异的字符。那我还需要显示声明字符编码吗？

答：是的！在每一个 HTML 页上你都应该总是指明所用的字符编码。不指定编码可能会导致安全漏洞（*http://code.google.com/p/doctype/wiki/ArticleUtf7*）。

总结：字符编码是复杂的，多年以来，许多只会复制粘帖的开发者大量使用垃圾软件，因此情况一点也没改善。你应该总是指定每个 HTML 文档的字符编码，否则难免会发生不好的事情。有多种方法可以做到，可以使用 HTTP Content-Type 头信息，<meta http-equiv>声明，或更短的<meta charset>声明，但是请一定要做。Web 会因此感谢你。

3.4.2 朋友和（链接）关系

普通的链接（<a href>）只是简单地链到另一个网页。链接关系（link relations）提供了一种方式以解释为什么要链接到那个页面。完成下面这句话："我链到另一个网页，是因为……"

- ……它是一个样式表，包含浏览器应当应用于当前文档上的 CSS 规则。
- ……它提供一个包含页面内容的标准订阅格式（比如 RSS）。
- ……它是本页面的某种语言的翻译版本。
- ……它是页面内容的 PDF 格式版本。
- ……它是一本在线电子书的下一章节，本页也是该书的一部分。

及诸如此类的信息。HTML5 的链接关系分为两类（*http://bit.ly/d2cbiR*）：

> 有两种类别的链接可以由 link 元素创建。外部资源链接引入那些用于加强当前文档的资源，超链接则指向其他文档。[...]

外部资源链接的确切行为取决于确切的关系，这也定义为相关链接类型（relevant link type）。

在我前面给出的示例中，只有第一个（rel="stylesheet"）是外部资源链接。其余的都是指向其他文档的超链接。你可以跟踪这些链接，也可以不，但它们对查看当前页面并不造成任何影响。

大多数情况下，链接关系都在一个页面的<head>区域里的<link>元素中可见。一些链接关系也可用于<a>元素，尽管允许，也十分不常见。HTML5 还允许在<area>元素里使用关系，但这更加罕见。（HTML 4 中不允许在<area>元素里添加 rel 属性。）请参考链接关系全图（*http://bit.ly/a3nsqi*）以查看在哪些地方可以使用特定的 rel 属性值。

专家答疑

问：我可以随意增添我想要的链接关系吗？

答：关于新的链接关系似乎总是有无穷的点子出现。为了试图阻止人们随意增加东西，WHAT 工作小组维护着一个注册表，其中包含所提议的 rel 属性值（*http://wiki.whatwg.org/wiki/RelExtensions*），并制定了审核和认可程序（*http://bit.ly/da3pse*）。

rel=stylesheet

让我们来看看示例中的第一个链接关系：

```
<link rel="stylesheet" href="style-original.css" type="text/css" />
```

这是全世界最常用的链接关系。<link rel="stylesheet">指向一个包含 CSS 规则的外部独立文件。在 HTML5 中你还可以做一个小小的优化，就是去掉 type 属性。用于 Web 的样式表语言只有一种，就是 CSS，因此它就是 type 属性的默认值了：

```
<link rel="stylesheet" href="style-original.css" />
```

这适用于所有浏览器。（我猜想或许有一天有人会发明一种新的样式表语言，但如果真的发生这事，你只需再把 type 属性写明即可。）

rel = alternate

继续看示例页面：

```
<link rel="alternate"
      type="application/atom+xml"
      title="My Weblog feed"
      href="/feed/" />
```

这个链接关系也相当常见。<link rel="alternate">，其 type 属性值可以是 RSS 或者 Atom 等媒介类型，由此可以开启所谓的“聚合内容自动发现机制”。这可以使得像 Google Reader 这样的聚合内容阅读器可以发现网站的最新内容聚合信息。大多数浏览器也支持聚

合内容，它们会在地址栏 URL 旁边显示一个特殊的图标。（和 rel="stylesheet"不同，这里的 type 属性是必须的，不能省略掉！）

即使在 HTML 4 中，rel="alternate"这个链接关系一直有着奇怪的混合用例。HTML5 里，它的定义已得到澄清，并扩展到能更准确地描述现有的 Web 内容。正如你刚才所看到的，结合使用 rel="alternate"以及 type=application/atom+xml 就可以指明当前页面的 Atom 聚合地址。当然你也可以将 rel="alternate"与其他 type 属性值结合使用来指明同样内容的其他格式，例如 PDF。

HTML5 也了结了长期存在的混乱，即关于如何链接页面文档的翻译版本。在 HTML 4 中可以结合使用 lang 属性与 rel="alternate"来指明所链接文档的语种，但这并不正确。HTML 4 勘误里面列举了规范（以及若干修订版）中的四个完全的错误；其中之一就是如何指定通过 rel="alternate"链接的文档的语种。（HTML 4 勘误以及现在 HTML5 中所描述的正确方式，是使用 hreflang 这个属性。）不幸的是，这些勘误从未重新集成到 HTML 4 的规范中，因为 W3C HTML 小组已经没有人还在为 HTML 工作了。

HTML5 中其他的链接关系

rel="archives"

（*http://bit.ly/clzlyG*）“表示所引用的文档描述了一组收藏，包括记录、文档或者有历史价值的材料。一个 Blog 的索引页可以使用 rel="archives"来链接到该 Blog 过去所发表文章的索引目录。”

rel="author"

用于链接到该页面作者的相关信息。这可以是一个 mailto：地址，但不是必须的。它也可以简单地链接到一个联系表格或者“关于作者”页面。

rel="external"

（*http://bit.ly/dBVO09*）“表示该链接指向一个和当前文档不同的，并不属于站点一部分的文档。”我相信它是随着 WordPress 而流行开来的，WP 会给访客留言中所含有的链接加上这个属性。

rel="start", rel="prev", and rel="next"

（*http://www.w3.org/TR/html401/types.html#type-links*）定义同属于一个系列的页面之间的关系（比如书本的章节，或者一篇 Blog 帖子）。而唯一被正确使用的是 rel="next"。过去人们使用 rel="previous"来代替 rel="prev"；使用 rel="begin"以及 rel="first"来代替 rel="start"；使用 rel="end"来代替 rel="last"。哦——还有他们自创的——使用 rel="up"来指向父页面。

HTML5 中包含有 rel="first"，这是最常见的表示“第一页”的说法。（rel="start"是一个不合格的代名词，目的是为了后向兼容。）它还包括有 rel="prev"和 rel="next"，

如同 HTML 4（为了后向兼容也支持 rel="previous"），此外还有 rel="last"（最后一页，和 rel="first"相呼应）以及 rel="up"。

要最好地理解 rel="up"，看一下（或者想象）“面包屑导航”（breadcrumb navigation）。你的首页可能是面包屑导航的起始，而当前所在页面位于末尾。那么 rel="up"就指向面包屑导航里位于倒数第二的那页。

rel="icon"

（*http://bit.ly/diAJUP*）这是第二流行的链接关系（*http://code.google.com/webstats/2005-12/linkrels.html*），仅次于 rel="stylesheet"。它通常和 shortcut 一起使用，像这样：

```
<link rel="shortcut icon" href="/favicon.ico">
```

所有的主流浏览器都支持这个用法，它们会将页面与一个小图标相关联。通常这个图标会显示在地址栏的网址旁边，或者浏览器标签卡上，或者两处都显示。

HTML5 中另外新增的特性：icon 链接关系中还可以使用 size 属性，来指明图标的尺寸（*http:/bit.ly/diAJUP*）。

rel="license"

（*http://bit.ly/9n9Xfv*）它由微格式（microformats）社群所发明。“表示所引用的文件正是当前文档所采用的版权许可证。”

rel="nofollow"

（*http://bit.ly/cGjSPi*）“表示链接未经页面的作者或出版者的认可，或者表示该链接主要是由于与对方的商业合作关系才被添加的。”这个链接关系由 Google 发明，并由微格式社群所标准化。最初的想法是，如果“nofollow”的链接不被 PageRank 理会，Blog 上收到的垃圾留言可能就会少很多。这个想法虽然未能如愿，但 rel="nofollow"还是保留了下来。许多流行的 Blog 系统都会默认给评论中的链接添加 rel="nofollow"。

rel="noreferrer"

（*http://bit.ly/cQMSJg*）“表示点击这个链接时不会泄漏 referer 信息。”目前市面上已经发布的浏览器中还没有支持这个的，不过 WebKit 的每日构建版本最近已经增加了对此的支持，所以它最终会出现于 Safari、Google Chrome，以及其他基于 WebKit 的浏览器上。在 *http://wearehugh.com/public/2009/04/rel-noreferrer.html* 可以找到一个 rel="noreferrer"的测试用例。

rel="pingback"

（*http://bit.ly/cIAGXB*）指定一个“通告（pingback）”服务器的地址。如同在 Pingback 的规范中所解释的（*http://hixie.ch/specs/pingback/pingback-1.0*），“当被其他网站链接时，Blog 可以自动接收到通知，这就是 pingback 系统的作用。”[...]它使得反向链接

——一种在成串的链接中返回而不是仅能深钻的方式——变得可能。Blog 系统，尤其是 WordPress，实现了这个 pingback 机制，在创建新帖子的时候告诉其他作者你链接了他们。

`rel="prefetch"`

（*http://bit.ly/9o0nMS*）“表示预先获取并缓存特定的资源可能是有好处的，因为用户极有可能会请求这个资源。”搜索引擎有时会给某些搜索结果加上`<link rel="prefetch" href="<emphasis>URL OF TOP SEARCH RESULT</emphasis>">`，因为它觉得这个结果要比其他的受欢迎得多。例如：使用 Firefox 浏览器，在 Google 搜索 CNN，查看页面源码，并搜索源码中的“prefetch”关键字。Mozilla Firefox 是目前唯一支持`rel="prefetch"`的浏览器。

`rel="search"`

（*http://bit.ly/aApkaP*）“表示所引用的文档提供了用于搜索文档及其相关资源的专门接口。”特别是如果你想让`rel="search"`做任何有用的事情，那就得让其链接指向一个 OpenSearch 文档，其中描述了浏览器如何针对一个给定关键字构造 URL 来搜索当前网站。微软 IE 浏览器从版本 7 开始，Mozilla Firefox 从版本 2 开始都已经支持 OpenSearch（以及指向 OpenSearch 描述文档的`rel="search"`链接）。

`rel="sidebar"`

（*http://bit.ly/azTA9D*）“表示所引用的文件如果被获取，那么其目的是要显示在次要浏览范围（如果有的话），而不是在当前浏览范围内。”这是什么意思呢？在 Opera 和 Mozilla Firefox 中，它的意思是“如果我点击这个链接，那么就提示用户创建一个收藏夹项目，当从收藏夹里选择该项目时，所链接的文档将在浏览器的侧边栏中打开。”（Opera 实际上称之为“面板”而不是“侧栏”。）IE、Safari 和 Chrome 则会忽视`rel="sidebar"`，将其看作一般的链接。在 *http://wearehugh.com/public/2009/04/rel-sidebar.html* 可以找到一个`rel="sidebar"`的测试用例。

`rel="tag"`

（*http://bit.ly/9bYlfa*）“表示这个标签（由所引用的文档代表）被应用于当前文档。”如此使用 rel 属性来标注“标签”（表示类别的关键字），最初是由 Technorati 发明的，目的是为了帮助对 Blog 文章进行分类。因此早期的 Blog 和教程都称它们为“Technorati 标签。”（你没看错：一家商业公司说服了整个世界来添加元数据，从而使该公司的工作变得更容易。这很棒，如果你能理解的话！）它的语法后来由微格式社群所标准化，被简单地称为`rel="tag"`。大多数支持文章分类，关键字和标签的 Blog 系统都会使用`rel="tag"`链接。它们确实是为了给搜索引擎提供一个关于页面内容的信号而设计的，浏览器并不特别对待它们。

3.5 HTML5 中新增的语义元素

New Semantic Elements in HTML5

HTML5 并不仅仅是把现有的标记变得更简短（虽然这方面它确实做了不少）。它还定义了一批新的语义元素。其中包括：

`<section>` section 元素代表文档或应用程序中一般性的“段”或者“节”。“段”在这里的上下文中，指的是对内容按照主题的分组，通常还附带标题。一些例子包括书本的章节，带标签页的对话框上的每个标签页，或者一篇论文的编号节。网站的主页也可以分为不同的节，比如介绍、新闻列表和联系信息。

`<nav>` nav 元素代表页面的导航链接区域，其中含有链接指向该页面上的其他各部分。并不是页面上所有的链接组都需要位于 nav 元素中——只有那些由主要的导航区块所组成的部分才适合。常见的情况是，页脚区域中含有一个简短的链接列表，指向网站的不同区域，比如服务条款、首页和版权页等。这种情况下 footer 元素自身就够用了，不需要另外使用 nav 元素。

`<article>` article 元素代表一个在文档，页面，应用程序或者网站中自成一体的内容组成，其目的是为了可以独立分发或者重用，例如用于聚合。这可以是一篇论坛帖子，一篇杂志或者报纸上的文章，一篇 blog 帖子，一条用户发表的评论，一个交互性的小部件，或者其他任何独立的内容项目。

`<aside>` aside 元素代表一个页面区域，其中包含和页面主要内容相关，但又可以单独存在的那些内容。在排印业界，这样的区域通常表现为侧边栏。这个元素可以用于实现一些字面特效，比如要点归纳或者侧栏，还可以用于广告、导航元素组，以及其他可以和页面主要内容分开的内容。

`<hgroup>` hgroup 元素代表“段”或“节”的标题。当标题有多个层级时，该元素可用于将 h1 到 h6 元素归组，用于实现比如子标题，替代标题或者题头标语。

`<header>` header 元素代表了一组介绍性内容或者导航辅助。一个 header 元素通常用来包含各节的标题（一个 h1-h6 元素或者 hgroup 元素），但这并不是必须的。header 元素也可以用来包裹一节的目录列表、一个搜索框，或者任何相关的 logo。

`<footer>` footer 元素代表一个节的末尾（“脚”）部分。页脚通常含有该节的一些基本信息，例如作者、相关文档的链接、版权资料等。页脚并不一定要出现在节的最底部，尽管通常都是这样。如果页脚元素包含了整个节，那么它们就代表附录、索引、长的题跋、冗长的许可协议，以及其他类似的内容。

`<time>` time 元素表示 24 小时时间上的一刻或者一个精确的公历日期，还可以附带时间以及时区偏移量。

`<mark>` mark 元素代表在文档中被突出/高亮标记的文字，作为参考之用。

我知道你急于开始使用这些新的元素，不然你也不会阅读本章。但首先我们还需要绕点圈子。

3.6 题外话：浏览器如何处理未知元素

A Long Digression into How Browsers Handle Unknown Elements

每个浏览器都有一份清单列举了它所支持的 HTML 元素。例如 Mozilla Firefox 的这个清单保存于 *nsElementTable.cpp*（*http://mxr.mozilla.org/seamonkey/source/parser/htmlparser/src/nsElementTable.cpp*）这个文件中。不在该清单上的元素都将被视为“未知元素”。对于未知元素存在两个基本的问题：

应当如何设定元素的样式？

默认情况下，<p>会在顶部和底部留白，<blockquote>会左侧缩进，<h1>会使用较大的字号。

元素的 DOM 是什么样？

Mozilla 的 *nsElementTable.cpp* 中包括了关于“每种元素可以包含哪些其他元素”的信息。如果你使用了像<p><p>这样的代码，第二个段落元素就会隐含地结束第一个段落元素，它们二者将会是兄弟而不是父子关系。但如果你写<p>，span 就不会结束段落，因为 Firefox 知道<p>是一个块级元素，可以包含像这样的行内元素。因此在最后的 DOM 中这个会是<p>的子元素。

对于上述两个问题，不同的浏览器会有不同的回答。（令人震惊吧，我知道。）当前的主流浏览器中，IE 在这方面的问题最多。

第一个问题的答案应该比较简单：不给未知元素添加任何特殊的样式。只要让它们继承任何当前页面位置上有效的 CSS 属性即可，并且让网页作者通过 CSS 指明所有的样式。不幸的是，Internet Explorer（版本 9 之前）并不允许给未知元素设定样式。例如，如果你写了这样的页面代码：

```
<style type="text/css">
  article { display: block; border: 1px solid red }
</style>
...
<article>
<h1>Welcome to Initech</h1>
<p>This is your <span>first day</span>.</p>
</article>
```

IE（直到 IE 8）并不会给 article 元素加上红色边框。在我撰写本书时，Internet Explorer 9 仍处于 beta 测试阶段，但微软已表示（开发者们也已证实），IE 9 不会再有这个问题。

第二个问题是浏览器在遇到未知元素时所创建的 DOM。同样，最有问题的浏览器是 Internet Explorer。如果 IE 不能明确识别一个元素，它就会在 DOM 中插入一个没有子元素的空节点。

所有那些你原本认为将会是这个未知元素的子元素的元素实际上会称为其兄弟元素。

下面的 ASCII 图展示了这个差异。这是 HTML5 想要的 DOM 结构：

```
article
|
+--h1 (article 的子节点)
|  |
|  +--文本节点"Welcome to Initech"
|
+--p (article 的子节点，h1 的兄弟节点)
   |
   +--文本节点"This is your "
   |
   +--span
   |  |
   |  +--文本节点"first day"
   |
   +--文本节点"."
```

但 Internet Explorer 实际创建的 DOM 结构却是：

```
article (无子节点)
h1 (article 的兄弟节点)
|
+--文本节点"Welcome to Initech"
p (h1 的兄弟节点)
|
+--文本节点"This is your "
|
+--span
|  |
|  +--文本节点"first day"
|
+--文本节点"."
```

针对此问题有个奇妙的弥补方案。如果你在使用<article>标签之前先用 JavaScript 创建一个虚假的<article>元素，Internet Explorer 就会奇迹般地可以识别<article>了，并允许你用 CSS 给它们设定样式。这个虚假的元素并不需要真正插入到 DOM 中。只需要在每个页面上创建一次，就足以教会 IE 来给这个它不认识的元素设定样式了。例如：

```
<html>
<head>
<style>
  article { display: block; border: 1px solid red }
</style>
<script>document.createElement("article");</script>
</head>
<body>
<article>
<h1>Welcome to Initech</h1>
<p>This is your <span>first day</span>.</p>
```

```
</article>
</body>
</html>
```

这适用于所有版本的 IE，甚至包括 IE6！利用这一技巧，我们可以为所有的新 HTML5 元素都一次性创建一份虚假副本——同样，它们无须被添加到 DOM 中，所以并不可见——然后就可以自如地使用它们，而无须担心那些不能良好支持 HTML5 的浏览器了。

Remy Sharp 的“HTML5 enabling script”就是帮你做这些事情的。该脚本经历了几次修改，但是其基本想法如下：

```
<!--[if lt IE 9]>
<script>
  var e = ("abbr,article,aside,audio,canvas,datalist,details," +
    "figure,footer,header,hgroup,mark,menu,meter,nav,output," +
    "progress,section,time,video").split(',');
  for (var i = 0; i < e.length; i++) {
    document.createElement(e[i]);
  }
</script>
<![endif]-->
```

这里<!--[if lt IE 9]>和<![endif]-->是条件注释。Internet Explorer 会将其解析为一个判定语句：“如果当前的浏览器是 Internet Explorer 版本 9 之前，就执行后面的代码块。”所有其他浏览器则会将整个代码块视作一段普通的 HTML 注释。最终结果是 Internet Explorer（直至并包括第 8 版）将执行这个脚本，但其他浏览器将完全忽略它。于是你的页面在那些无须这个 hack 的浏览器中将会加载得更快。

上面的 JavaScript 代码本身相对简单。变量 e 是一个字符串数组，其中保存有"abbr"、"article"、"aside"等。然后我们循环遍历这个数组，通过调用 document.createElement() 来创建每个命名元素。但由于忽略了返回值，这些元素就并不会真正放进 DOM 中。但这样已经足以让 IE 能正确地处理这些元素了，我们已经可以接下来在页面中使用它们。

这个“接下来”很重要。上述脚本代码需要位于页面起始部分——最好是<head>中——而不是页面底部。这样，Internet Explorer 将会在解析页面上的标签和属性之前，先运行脚本代码。要是你把脚本放在页面最下方，那它的执行就太迟了。Internet Explorer 会已经解析完全部的页面代码，并建立了错误的 DOM 结构，并且不会再因为这个脚本而返回去重新修改。

Remy Sharp 已经“最小化”压缩了这个脚本，并托管在 Google Project Hosting 上（*http://code.google.com/p/html5shiv/*）。（此外，这些代码本身是开源的，遵循 MIT 协议，所以你可以用于任何项目中。）如果喜欢，你甚至可以直接外链这个脚本，获取其托管版本：

```
<head>
  <meta charset="utf-8" />
  <title>My Weblog</title>
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
```

现在，我们已经准备就绪，可以开始使用这些新的 HTML5 语义元素了。

3.7 页头
Headers

回到我们的示例页面。确切地说，让我们看看页头部分：

```
<div id="header">
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
</div>

...

<div class="entry">
  <h2>Travel day</h2>
</div>

...
<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

这些代码没有任何错误。如果喜欢，你可以继续使用它们。它们也是合法的 HTML5。但 HTML5 还提供了一些额外的，专门用于页头和章节的元素。

首先，让我们换掉这个 。这样的写法十分常见，但它没有多少意义。div 元素和 id 属性都没有任何语义。（用户代理不能从 id 属性的值上推断出任何含义。）你也可以写成 ，它还是会具有同样的语义：即压根没有语义。

HTML5 为此定义了<header>元素。在 HTML5 的规范里可以找到数个使用<header>元素的现实例子。下面就在我们的示例页面上使用它：

```
<header>
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
  ...
</header>
```

很好。它告诉了所有人这是一个页头。但是题头标语（tagline）怎么办呢？这是另外一

种常用的模式，但至今也还没有标准的标注代码，它要难办一些。一个题头标语类似一个二级标题，但它又“依附”于主标题。也就是说，这个二级标题并不创建属于自己的区域。

像<h1>和<h2>这样的页头元素可以使页面结构化。两者结合使用，可以为页面创建出一个大纲，使其具备更好的可视性（或者可导航性）。屏幕阅读器软件会使用这样的大纲，来帮助盲人用户浏览页面。也有一些在线工具（*http://gsnedders.html5.org/outliner/*）和浏览器扩展（*http://chrispederick.com/work/web-developer/*），可以将页面的大纲可视化展现。

在 HTML 4 中，创建文档大纲的唯一方式是使用<h1>到<h6>元素。示例页面的大纲看起来是这样的：

```
My Weblog (h1)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

看上去不错，但这也意味着没有办法标注“A lot of effort went into making this effortless”这个题头标语。如果将其标注为一个<h2>，那么文档大纲中会多一个“幽灵”节点：

```
My Weblog (h1)
|
+--A lot of effort went into making this effortless. (h2)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

但这并不是该文档应有的结构。题头标语并不代表一个“节”，它只是一个副标题。

那么如果将题头标语标注为一个<h2>，同时将其他的文章标题换成<h3>呢？不，这样会更糟糕：

```
My Weblog (h1)
|
+--A lot of effort went into making this effortless. (h2)
   |
   +--Travel day (h3)
   |
   +--I'm going to Prague! (h3)
```

现在我们的文档大纲里还是存在一个幽灵节点，它还“偷窃”了两个本该直接从属于根节点的子节点。问题就出在这里：HTML 4 并没有提供一个方法来标注副标题，它们无法避免地会被加到文档大纲里。无论怎么规避，“A lot of effort went into making this effortless”这句话都会出现在上面的结构图里。最后我们不得不使用像<p class="tagline">这样毫无语义性的代码。

HTML5 则为此提供了解决方案：<hgroup>元素。<hgroup>元素能起到“包装”的作用，可以将两个或者更多相关联的标题元素置于其中。那么这里的“相关联”是什么意思呢？即，它们合在一起可以组成文档大纲中的单一节点。

像下面这样的代码：

```
<header>
  <hgroup>
    <h1>My Weblog</h1>
    <h2>A lot of effort went into making this effortless.</h2>
  </hgroup>
  ...
</header>

...

<div class="entry">
  <h2>Travel day</h2>
</div>

...

<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

所创建的文档大纲就是：

```
My Weblog (其所处 hgroup 里的 h1)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

你可以用 HTML5 Outliner 来测试自己的页面，以确保你正确使用了标题元素。

3.8 文章

Articles

继续看我们的示例页面，对于以下的代码可以做些什么改进呢：

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
      rel="bookmark"
      title="link to this post">
      Travel day
    </a>
  </h2>
  ...
</div>
```

同样，这些也都是合法的 HTML5 代码。但 HTML5 还提供专门的元素，用于标注常见于页面上的“文章”——元素的名字非常恰当<article>：

```
<article>
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="link to this post">
       Travel day
    </a>
  </h2>
  ...
</article>
```

噢，它其实并不是那么简单。你还应该做一个修改。我先展示给你看，然后再解释：

```
<article>
  <header>
    <p class="post-date">October 22, 2009</p>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  ...
</article>
```

你看出来了吗？我把<h2>元素改成了<h1>，并将它包在一个<header>元素之中。前面你已经实际了解过<header>元素了。其目的是包裹所有那些组成文章头部的元素（在本例中即文章的发表日期以及标题）。但是...但是...但是...每个文档难道不是应该只有一个<h1>吗？这样做会不会搞乱文档大纲？不会，但要理解为什么不会，我们需要回退一步。

在 HTML 4 中，创建文档大纲的唯一办法是使用<h1>到<h6>元素。如果想让大纲中只有唯一一个根节点，那就只能在代码中写一个<h1>。但 HTML5 规范定义了算法用于创建一个文档大纲，其中能够包含那些新的 HTML5 语义元素。根据这个算法，一个<article>元素将创建一个新的“节”，也就是文档大纲中一个新的节点。而在 HTML5 中，每个节都可以有自己的<h1>元素。

同 HTML 4 相比，这是一个剧烈的改变，但下面会说明为何这个改变是好事情。许多的页面其实都是通过模板自动生成的。来自不同源头的内容片段被插入页面上的不同区域。许多教程里也是这样说的。“这里有一些 HTML 代码。只需复制并粘贴到您的页面里。”对于很少量的内容来说，这样做可行，但如果要复制粘帖的整个“节”呢？在这种情况下，

教程就会这样说："这里有一些 HTML 代码。只需要将它复制并粘贴到一个文本编辑器中，然后修改标题标签，以便使它们和页面上已有的标题保持正确的嵌套关系。"

让我换一种方式表达。HTML 4 里没有一个一般性的标题元素。它只有六个严格编号的标题元素，从<h1>到<h6>，它们必须以这样的顺序嵌套。这是件糟糕的事情，尤其是如果你的页面是"拼装"而不是"撰写"而成。HTML5 新增的分节元素,以及针对已有标题元素的新的使用规则，就正好用于解决这样的问题。如果你使用新的章节元素，那就可以用这样的代码：

```
<article>
  <header>
    <h1>A syndicated post</h1>
  </header>
  <p>Lorem ipsum blah blah...</p>
</article>
```

然后你可以将其粘帖在页面任何地方而无须另做修改。这段代码里尽管包含一个<h1>元素，但这并不构成问题，因为所有东西都位于一个<article>容器内。该<article>元素定义了一个自包含的页面大纲节点，<h1>元素给这个节点提供了标题，页面上已有的章节元素也会保持其原有嵌套层级。

专家发言

如同 Web 上的一切东西深入下去都会变得更加复杂。新的"显式"分节元素（比如包裹于<article>中的<h1>），可能会以意料之外的方式同旧的"隐式"分节元素（<h1>到<h6>）互相影响。为了避免麻烦，你最好只用二者其中之一，而不是同时使用。如果不得不同时用，那就要确保用 HTML5 Outliner 检查页面代码，并确认文档大纲是正确无误的。

3.9 日期和时间

Dates and Times

这是激动人心的，对吗？我的意思是，这不是"裸体在珠峰滑雪同时倒背星条旗永不落的歌词"的那种兴奋，但这对于标记语言来说确实很令人振奋。继续我们的示例页面。我这次要强调的是：

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>Travel day</h2>
</div>
```

同样的老故事，对吗？一种常用的模式——标记一篇文章的发表日期——使用没有任何语义的通用代码，页面作者只能加上一些自定义的 class 属性。同样，这些也还是合法的 HTML5 代码。你并不一定非得改写它们不可。但 HTML5 确实针对这样的情况提供了一种专门解决方案——<time>元素：

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

<time>元素有三个组成部分：

- 一个机器可识别的时间戳
- 人可识别的文本内容
- 一个可选的 pubdate 标记

在这个例子中，datetime 属性只能指定一个日期，而不是时间。其格式是四位数字的年份，两位数字的月份和两位数字的天数，由减号分隔：

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

如果你还想包含时间，那就在日期后面加上字母 T，然后跟 24 小时格式的时间值，以及时区偏移量：

```
<time datetime="2009-10-22T13:59:47-04:00" pubdate>
  October 22, 2009 1:59pm EDT
</time>
```

日期/时间的格式非常灵活。HTML5 规范中包含了数个合法的日期/时间字符串的例子。

请注意，上例中我改变了文本内容——位于<time>和</time>标签之间的东西——以匹配机器可识别的时间戳。这并不是必须的。文本内容可以是任何你喜欢的东西，只要给 datetime 属性赋予一个机器可识别的日期/时间戳即可。因此，这是合法的 HTML5 代码：

```
<time datetime="2009-10-22">last Thursday</time>
```

这同样是合法的 HTML5 代码：

```
<time datetime="2009-10-22"></time>
```

最后是这个 pubdate 属性。它是一个布尔属性，如果需要它，写上属性名即可，像这样：

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

如果你不喜欢这样的“裸体”属性，等价的写法是：

```
<time datetime="2009-10-22" pubdate="pubdate">October 22, 2009</time>
```

这个 pubdate 属性是什么意思呢？它意味着两件事情中的一件。如果该<time>元素位于一个<article>元素之中，那么它表示该时间戳是这篇文章的发表时间。如果不在一个<article>里，那就表示整个文档的发布时间。

以下是经过改写，充分利用了 HTML5 的优点的完整文章代码：

```
<article>
  <header>
    <time datetime="2009-10-22" pubdate>
      October 22, 2009
    </time>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  <p>Lorem ipsum dolor sit amet...</p>
</article>
```

3.10 导航

Navigation

导航栏对于任何网站来说都是最重要的组成部分之一。CNN.com 的每个页面顶部都有一行“标签”，链接到不同的新闻区域——“技术”、“健康”、“体育”等。Google 的搜索结果页面在顶部也有类似的区域，使得你可以在不同的 Google 服务中搜索——“图像”、“视频”、“地图”等。我们的示例页面在页头部分也有一个导航栏，包含指向这个虚拟网站的不同区域的链接——“home”、“blog”、“gallery”和“about”。

这是最初的导航栏代码：

```
<div id="nav">
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</div>
```

同样，这是合法的 HTML5 代码。但是尽管它们标注了一个含有四个项目的列表，你却无从得知这些代表的是一个站点导航。乍一看，以及阅读链接文字后，你可能会猜测到这是页头的一部分。但是语义上，这个链接和其他任何一个链接没有什么区别。

那都有谁在乎网站导航的语义性呢？其一，残疾人士（*http://diveintoaccessibility.org*）。这是为什么？考虑这种情况：你的运动能力受限，使用鼠标很困难或者根本不可能。为此，你可以使用一个浏览器插件，它可以让你在主要的导航链接之间跳转。或者考虑另外一种

情况：你的视力受限，所以使用专门的“屏幕阅读器”程序，通过将文本转换成语音来“听”页面内容。当你知道页面总标题后，下一个关于该页的重要信息片段就是主要导航链接。如果你想快速浏览，你会告诉屏幕阅读器跳转到导航栏，并开始阅读。如果你想很快地浏览，你可能会告诉屏幕阅读器跳过导航栏，开始阅读主要内容。无论哪种方式，很重要的一点都是要能够确定导航链接的位置。

因此，虽然这里使用 来标注站点导航并没有什么错误，但也并不算有多正确。某些方面还是会影响到某些人。HTML5 提供了一种语义化的方法来用于标注导航区域——<nav>元素：

```
<nav>
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</nav>
```

专家答疑

问：跳过链接（skip links）还能与<nav>元素兼容吗？在 HTML5 中是否还需要跳过链接？

答：跳过链接浏览者可以直接略过导航区域。它们对于有残障的，使用第三方软件来阅读网页及不使用鼠标导航的用户来说很有帮助。在这里可以学习如何及为何提供跳过链接：*http://www.webaim.org/techniques/skipnav*。

一旦屏幕阅读器经过更新可以识别<nav>元素后，跳过链接就将变得过时，因为屏幕阅读器将能够自动略过导航部分——<nav>元素了。然而，这得是一段时间之后的事情了，需要等待所有的残障 Web 用户都升级到支持 HTML5 的屏幕阅读器，所以你还是应当继续提供自己的跳过链接机制，来略过<nav>区域。

3.11 页脚
Footers

终于走到示例页面的末尾了。最后我要说的东西也是页面的最后一个部分：页脚。示例中的页脚起初是这样标注的：

```
<div id="footer">
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</div>
```

这是合法的 HTML5 代码。如果喜欢，可以继续使用它们。但 HTML5 还为此提供了一个更专门一些的元素——<footer>元素：

```
<footer>
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</footer>
```

哪些东西适合放在<footer>元素里面呢？差不多就是任何你现在放进 里面的那些。好吧，这是个绕圈子的回答。但真的，就是如此。HTML5 规范说到："页脚通常包含类似作者、相关文档链接、版权数据等信息。"以上正是例子页面里页脚部分的内容：简短的版权声明及指向"关于作者"页的链接。这就是在这个例子中页面的页脚：短版权声明，和一个到笔者页面的链接。看看一些热门的站点，就能发现页脚内容多种多样：

- CNN 网站的页脚部分包含版权声明、不同语种页面的链接、服务条款链接、隐私、"关于我们"、"联系我们"及"帮助"页。所有这些都适合放在<footer>中。
- Google 的页面以简洁而出名，但在页面底部还是有链接指向"广告计划"、"商业解决方案"及"关于 Google"；版权声明；以及指向 Google 隐私政策的链接。所有这些也都可以置于<footer>里面。
- 我自己的 Blog（*http://diveintomark.org*）的页脚部分包含指向我的其他网站的链接，外加版权声明。也绝对适合放在一个<footer>元素里。（注意，这些链接本身不应该再用<nav>元素包裹，因为它们并不是站点导航链接；它们只不过是一些我的其他项目和网站的链接的集合。）

近年来很流行在页脚部分包含大量内容（*http://ui-patterns.com/pattern/FatFooter*）。看一看 W3C 网站（*http://www.w3c.org*）的页脚。它包含三列，分别标有"导航"、"联系 W3C"和"W3C 更新"。其代码差不多是这样：

```
<div id="w3c_footer">
  <div class="w3c_footer-nav">
    <h3>Navigation</h3>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>Contact W3C</h3>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
```

```
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>W3C Updates</h3>
    <ul>
      <li><a href="http://twitter.com/W3C">Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
  </div>
  <p class="copyright">Copyright © 2009 W3C</p>
</div>
```

要将这个转化成语义良好的 HTML5，我会做下列改动：

- 将最外围的 改为<footer>元素。
- 将前两个 改写为<nav>元素，第三个改为<section>元素。
- 将<h3>标题改为<h1>，因为它们各自都位于一个分节元素的内部。<nav>元素也可以在文档大纲中创建分节，就像<article>元素那样（参见第 47 页“文章”一节）。

最后的代码看起来像这样：

```
<footer>
  <nav>
    <h1>Navigation</h1>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </nav>
  <nav>
    <h1>Contact W3C</h1>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </nav>
  <section>
    <h1>W3C Updates</h1>
    <ul>
      <li><a href="http://twitter.com/W3C">Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
  </section>
  <p class="copyright">Copyright © 2009 W3C</p>
</footer>
```

3.12 扩展阅读
Further Reading

本章中使用的示例页面：

- 原始（HTML 4）（*http://diveintohtml5.org/examples/blog-original.html*）。
- 改进（HTML5）（*http://diveintohtml5.org/examples/blog-html5.html*）。

关于字符编码：

- Joel Spolsky：“The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)”（*http://www.joelonsoftware.com/articles/Unicode.html*）。
- Tim Bray：“On the Goodness of Unicode”，“On Character Strings”，and “Characters vs. Bytes”（*http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF*）。

关于在 Internet Explorer 中开启对 HTML5 的支持：

- Sjoerd Visscher：“How to style unknown elements in IE”（*http://wopus.com/devblog/2008/style-unknown-elements.html*）。
- John Resig：HTML5 shiv（*http://ejohn.org/blog/html5-shiv/*）。
- Remy Sharp：HTML5 enabling script（*http://remysharp.com/2009/01/07/html5-enabling-script/*）。

关于标准模式及 doctype 侦测：

- Henri Sivonen：“Activating Browser Modes with Doctype”（*http://hsivonen.iki.fi/doctype*）。这是关于该主题你唯一应当阅读的文章。也有许多其他同主题的文章，但它们要么太过时，不完整，要么就是错误的。

支持 HTML5 的校验器：

- Validator.nu (X)HTML5 Validator（*http://html5.validator.nu*）.

第 4 章

Canvas 绘图

Let's Call It a Draw(ing Surface)

4.1 引言

Diving In

HTML5 中定义<canvas>元素 *http://bit/ly/9JHzOf* 为："它是依赖分辨率的位图画布，你可以在 canvas 上面绘制任意图形，甚至加载照片。"在网页中，一个 canvas 就是一个矩形区域。你可以在这个区域内通过 JavaScript 任意地绘制图形。下表是本书写作之时浏览器对 canvas 的基本支持情况：

IE[a]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0 以上版本	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 支持[1]。

那么，canvas 看上去像什么呢？什么也没有，真的。<canvas>元素既没有自己的内容也没有自己的边框。标记看起来像这样：

```
<canvas width="300" height="225"></canvas>
```

图 4.1 显示了一个 canvas，我们加上虚线边框以便识别。

同一页面里可以有多个<canvas>元素。每个 canvas 将出现在 DOM 中并维护自己的状态。如果你给每个 canvas 加上 id 属性，就可以像获取任何其他元素一样取得它。

让我们加上 id 属性吧：

```
<canvas id="a" width="300" height="225"></canvas>
```

现在，我们可以很容易地从 DOM 中获取这个<canvas>元素：

```
var a_canvas = document.getElementById("a");
```

[1]译注：在本书翻译之时，IE9beta 已经提供了对<canvas>的支持。更多详情信息参见：*http://msdn.microsoft.com/zh-cn/ie/ff468705.aspx#_HTML5_canvas*。

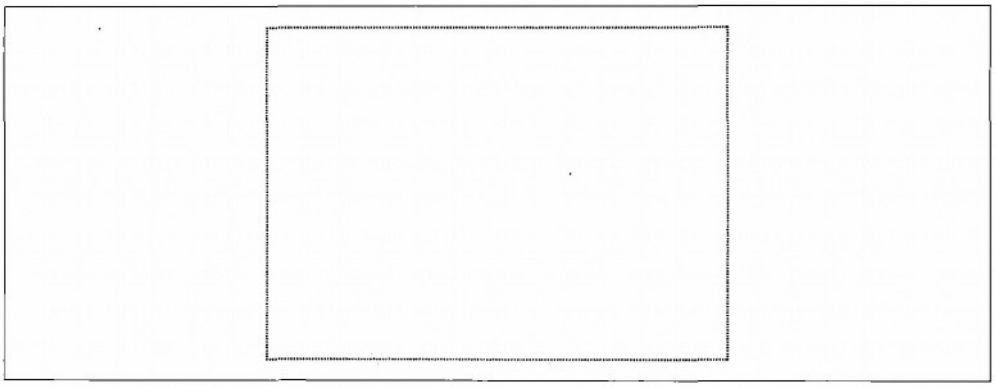


图 4.1 带边框的 canvas

4.2 简单的图形
Simple Shapes

IE[a]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 支持。

每个 canvas 初始时就是一块白板上面什么也没有。让我们来画点什么吧。你可以使用 onclick 事件处理器去调用一个函数来绘制一个矩形（在线示例见 *http://diveintohtml5.org/canvas.html*）：

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

函数的第一行没有什么特别，它只是从 DOM 中获取<canvas>元素。第二行让它变得更加有趣些。每个 canvas 都有一个上下文环境，这是趣事诞生之地。一旦你从 DOM 中找到一个<canvas>元素（使用 document.getElementById()或任何你喜欢的方式），你就可以调用它的 getContext()方法。你必须要给 getContent 传递一个字符串"2d"作为参数。

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

专家答疑

问：是否有 3D 的 canvas？

答：目前还没有。虽然个别厂商已经推出它们自己的 3DcanvasAPI 的实验性版本，但还没有被标准化。HTML5 的规范提到，“该规范的未来版本有可能会定义一个 3D 的上下文环境。”

当你创建一个<canvas>元素后，就拥有了它的绘图上下文。绘图上下文包含所有绘制方法和属性的定义。有一整套属性和方法专门用于绘制矩形：

- fillStyle 可以设置为 CSS 颜色、一个图案或一种颜色渐变。（稍后会详细介绍颜色渐变。）fillStyle 默认是纯黑色，你可以设置成你喜欢的任意颜色。只要页面打开着，每个绘图上下文都会记录自己的属性，除非你重置过它。
- fillRect(*x*, *y*, *width*, *height*)绘制一个矩形，并以当前的 fill Style 来填充。
- strokeStyle 和 fillStyle 一样，可以设置为 CSS 颜色、一个图案或一种颜色渐变。
- strokeRect(*x*, *y*, *width*, *height*)使用当前的 storke style 来绘制一个矩形，strokeRect 并不填充中间区域，而只绘制矩形的边缘。
- clearRect(*x*, *y*, *width*, *height*)清除指定矩形区域的像素。

专家答疑

问：我可以“重置”一个 canvas 吗？

答：当然可以。只要你设置<canvas>的高度或者宽度就可以让绘图上下文里的属性值重置为默认值。你甚至不需要改变它的宽度，只要简单地设置为当前的宽度就行，就像这样：

```
var b_canvas = document.getElementById("b");
b_canvas.width = b_canvas.width;
```

再回到前面的例子：

```
var b_canvas = document.getElementById("b");
var b_context = b_canvas.getContext("2d");
b_context.fillRect(50, 25, 150, 100);
```

这里调用 fillRect()方法来绘制一个矩形，并使用当前 fill style 填充，默认为黑色，除非你改变了 fill style。该矩形从左上角（50，25）这个点开始，宽 150 像素，高 100 像素。要想更深入理解，让我们一起来看 canvas 的坐标系。

4.3 Canvas 坐标系
Canvas Coordinates

canvas 是一个二维的网格。坐标（0，0）位于 canvas 的左上角。沿 X 轴，取值从左至右递增。沿 Y 轴，取值由上到下递增。

图 4.2 是用<canvas>元素绘制的坐标系。它包括：

- 一组垂直的灰色直线。
- 一组水平的灰色直线。
- 两段水平的黑色直线。
- 两条黑色斜线形成了一个箭头。
- 两段垂直的黑色直线。
- 两条黑色斜线形成了另外一个箭头。
- 字母“X”。
- 字母“Y”。
- 左上角附近的文字“(0，0)”。
- 右下角附近的文字“(500，375)”。
- 左上角右下角各一个黑点。

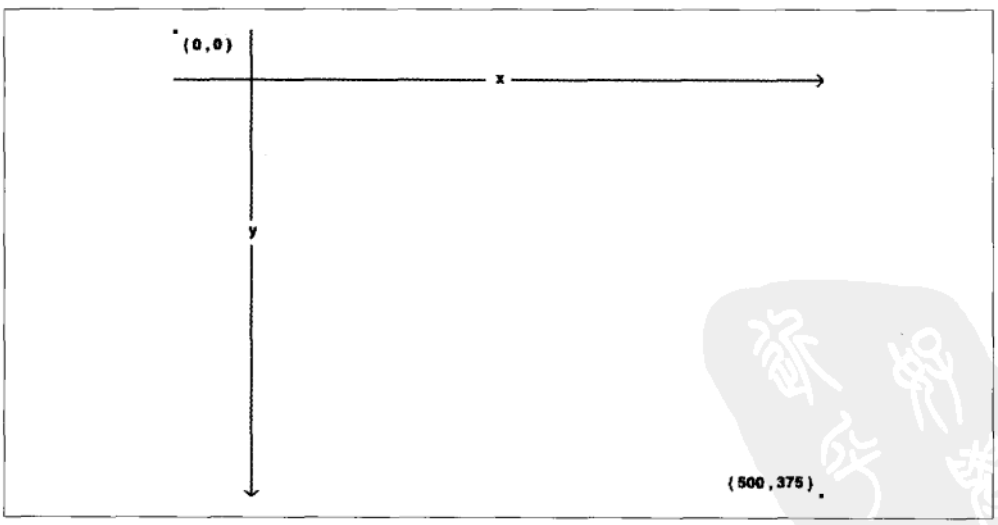


图 4.2 canvas 坐标图

在下面的章节中，我们将探讨如何创建上图所示的效果。首先需要定义一个<canvas>元素，设置它的高和宽，并绑定一个 id 以方便以后获取它。

```
<canvas id="c" width="500" height="375"></canvas>
```

然后，我们需要一段脚本从 DOM 中获取<canvas>元素，并得到其绘图上下文：

```
var c_canvas = document.getElementById("c");
var context = c_canvas.getContext("2d");
```

现在我们可以开始画线了。

4.4 路径
Paths

IE[a]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 支持。

想象你正在用笔作画。刚开始还有些拿不准，你不想直接就用笔画上去。因此，你先用铅笔勾画轮廓，直到你满意了，才再用画笔描绘这些线条。

每个 canvas 都有一个路径。定义路径就如同用铅笔作画。你可以任意地画，但它不一定是最终作品的一部分，直到你拿起画笔沾上墨水描绘这条路径。

用铅笔画直线，可以使用以下两种方法：

- moveTo(*x* , *y*)把铅笔移动到指定点并作为线条的开始点。
- lineTo(*x* , *y*)绘制线条到指定的结束点。

moveTo()和 lineTo()方法被调用得越多，路径信息也就越多。这些都是“铅笔”方法，你可以任意使用，但 canvas 上不会有任何图案，除非你调用了某一个“画笔”的方法。

让我们先来绘制灰色的网格：

```
for (var x = 0.5; x < 500; x += 10) {
  context.moveTo(x, 0);
  context.lineTo(x, 375);
}
for (var y = 0.5; y < 375; y += 10) {
  context.moveTo(0, y);
  context.lineTo(500, y);
}
```

这些都是“铅笔”方法。其实没有什么被真正的画到 canvas 上去。我们需要一个“画笔”方法把图案绘制到 canvas 上去：

```
context.strokeStyle = "#eee";
context.stroke();
```

stroke()就是一种“画笔”方法。它会把由 moveTo()和 lineTo()方法调用而产生的复杂的路径绘制到 canvas 上。strokeStyle 控制线条的颜色。图 4.3 显示了绘制结果。

图 4.3　在 canvas 上绘制的网格

专家答疑

问：为什么 x 和 y 的数值从 0.5 开始？为什么不是 0 呢？

答：想象每一个像素都是一个很大的正方形。整数的坐标（ 0，1，2 ...）是这些正方形的边缘。如果你在坐标系中按照整数来画一条 1 像素宽度的直线，其实这条直线会落在这个像素正中间，两边对称，各占据一半的宽度[2]，结果导致得到了一条 2 像素宽度的直线。想画一条 1 像素宽的直线，必须相应移动 0.5 个像素。

[2]译注：这里就是 0.5 个像素，但由于显示器无法显示半个像素，所以会被迫往两边扩展。

现在来画水平方向的箭头。同一条路径上的所有的线条和曲线都会是相同的颜色（或相同的渐变颜色，稍后很快就会讲到颜色渐变）。我们想要画一个黑色的箭头，而不是白色，所以要开始一个新的路径：

```
context.beginPath();
context.moveTo(0, 40);
context.lineTo(240, 40);
context.moveTo(260, 40);
context.lineTo(500, 40);
context.moveTo(495, 35);
context.lineTo(500, 40);
context.lineTo(495, 45);
```

垂直方向的箭头大同小异。由于垂直方向的箭头和水平方向的箭头是一个颜色，所以这里无须创建新的路径。两个箭头共用同一个路径就行：

```
context.moveTo(60, 0);
context.lineTo(60, 153);
context.moveTo(60, 173);
context.lineTo(60, 375);
context.moveTo(65, 370);
context.lineTo(60, 375);
context.lineTo(55, 370);
```

之前说过这些箭头应该是黑色，但目前 strokeStyle 仍是白色。（创建新的路径，fillStyle 和 strokeStyle 不会被重置。）没关系，我们还只是用“铅笔”的方法。但在画到画布上之前，应该先把 strokeStyle 设置成为黑色。不然箭头就是灰色了，这会让我们很难看清楚它。以下几行就能把颜色更改为黑色并把线条绘制到画布上去：

```
context.strokeStyle ="#000";
context.stroke();
```

图 4.4 显示了绘制后的结果。

4.5 文本

Text

IE[a]	火狐[b]	Safari	Chrome	Opera	iPhone	Android
7.0 以上	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 的支持。

[b] Mozilla 公司的 Firefox 3.0 需要少许兼容性的处理。[3]

除了可以在 canvas 上绘制线条，你还可以绘制文本。不同于周围网页上的文本，它是没有盒模型的。这意味我们所熟悉的 CSS 布局技术在这里完全无用：没有浮动，没有边距，没有留白，也没有自动换行。（也许你认为这是一个好事！）但你还是可以设置一些字体属性，然后在 canvas 中选择一个点来绘制你的文本。

[3]译注：Firefox3.5 正式提供了对 fill Text 方法的支持，更多信息请参见 *https://developer.mozilla.org/en/drawing_text_using_a_canvas*。

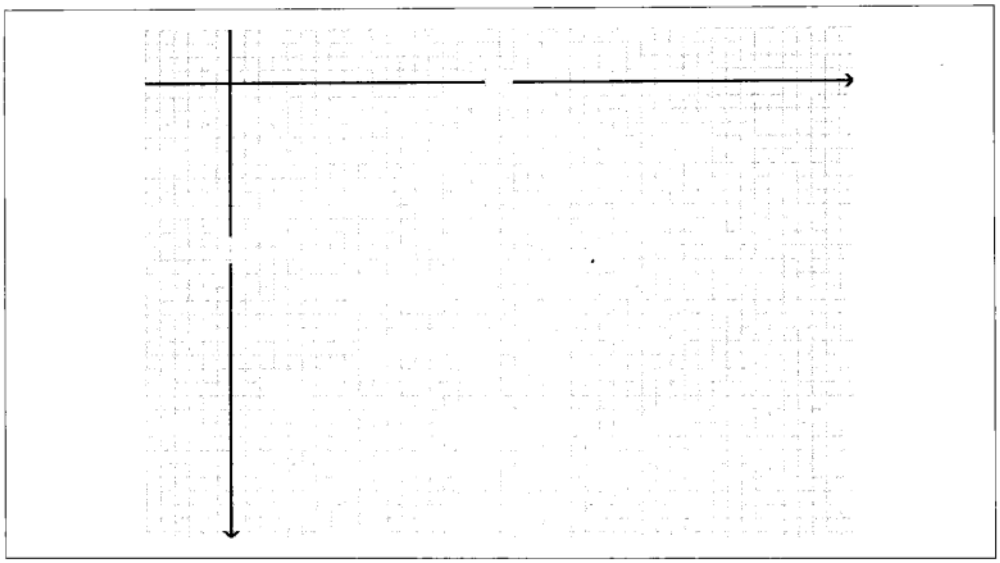


图 4.4　画在 canvas 上的尚未标识的坐标轴

在绘制上下文里有以下字体属性（参见第 58 页“简单的图形”一节）：

- font 可以是 CSS 字体规则中的任何值。包括：字体样式（font style）、字体变种（font variant）、字体大小与粗细（font weight & font size）、行高（line height）和字体名称（font family）。
- textAlign 控制文本的对齐方式。它类似于（但不完全相同）CSS 中的 text-align。可能的取值为：start，end，left，right 和 center。
- textBaseline 控制文本相对于起点的位置。可能的取值有 top，hanging，middle，alphabetic，ideographic 和 bottom。

textBaseline 的设置挺复杂，因为文本变化多端。（好吧，我承认英文文本没有问题，但你可以绘制任何 Unicode 字符的文本，而 Unicode 本身就很麻烦。）HTML5 的规范解释了不同的文本基线：[4]

> em 方块的顶部一般是字体字形的顶部，悬挂基准线是诸如अ的一些字体字形的边界，中线位于 em 方块正中间，字母基线是诸如 Á、ÿ 、f 和Ω的一些字体字形的边界，象形基准线是诸如私及達等字体的字形边界，em 块的底部一般是字体字形的底部。边界盒的顶部和底部可以远远超出这些基准，完全取决于字形需要超出 em 方块的大小（见图 4.5“文本基线”）。

[4]注：*http://bit.ly/aHCdDO*。

图 4.5 文本基线

对于简单的英文字母，你可以放心地使用 top：middle 或 bottom 作为文本基线。

现在来让我们一起画一些文字吧。canvas 中的文字会继承<canvas>元素的字体大小和样式风格，但你可以设置 font 属性来覆盖继承的值：

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

fillText()方法绘制实际的文本：

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

专家答疑

问：我可以使用相对字体大小 来绘制 canvas 上的文本吗？

答：可以。和其他 HTML 元素一样，<canvas>元素会基于页面的 CSS 规则计算自己的字体大小。如果你设置 context.font 为相对字体大小 1.5em 或者 150%，浏览器会将<canvas>元素本身的字体大小乘上这个倍数。

对于左上角的文本，假设我们希望文字的顶端位于左上角，y=5 的位置。但是我们比较懒，不想计算文本的高度和基线。实际上，我们可以设置 textBaseline 为 top，并设置左上角的坐标来实现：

```
context.textBaseline = "top";
context.fillText("( 0 , 0 )", 8, 5);
```

现在来实现右下角的文本。假设我们希望右下角的文本位于坐标(492,370)——与canvas右下角只有几个像素的距离。同样，我们不想计算文本的高度或者宽度。就可以设置文本的textAlign为右，textBaseline为bottom，然后调用fillText()方法，并传入坐标即可：

```
context.textAlign = "right";
context.textBaseline = "bottom";
context.fillText ( " ( 500, 375 ) ", 492, 370 ) ;
```

图 4.6 显示了对应的效果。

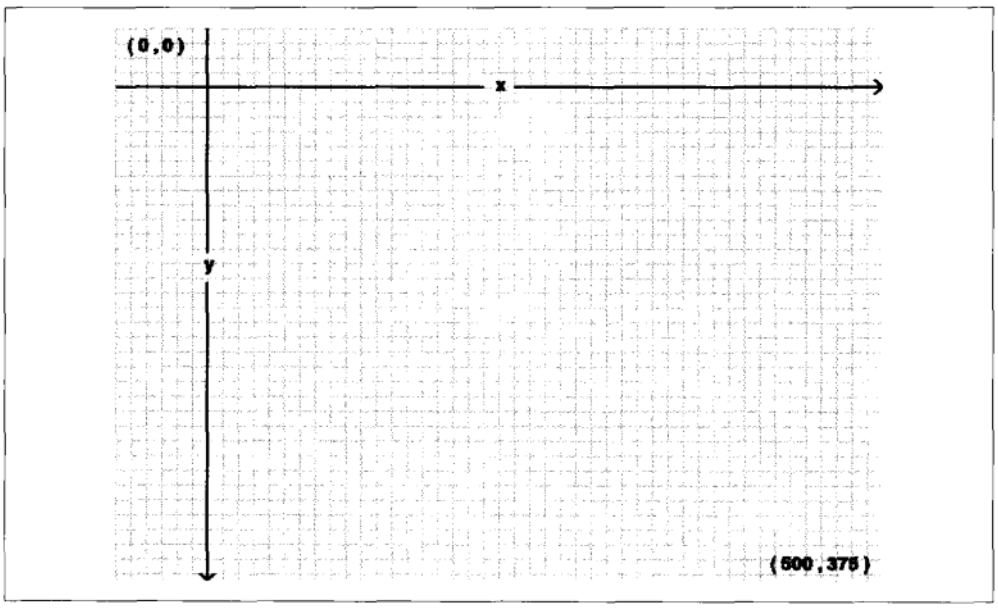


图 4.6 cavnas 上带标识的坐标轴

哦！我们把顶角的点给忘了。稍后我们会学习如何画一个小圆圈，不过这里我们就先用矩形代替一下吧（参见第 58 页“简单的图形”一节）：

```
context.fillRect ( 0, 0, 3,3 ) ;
context.fillRect ( 497, 372, 3, 3 ) ;
```

好吧，我们的作品完成了！图 4.7 显示了最终的作品。

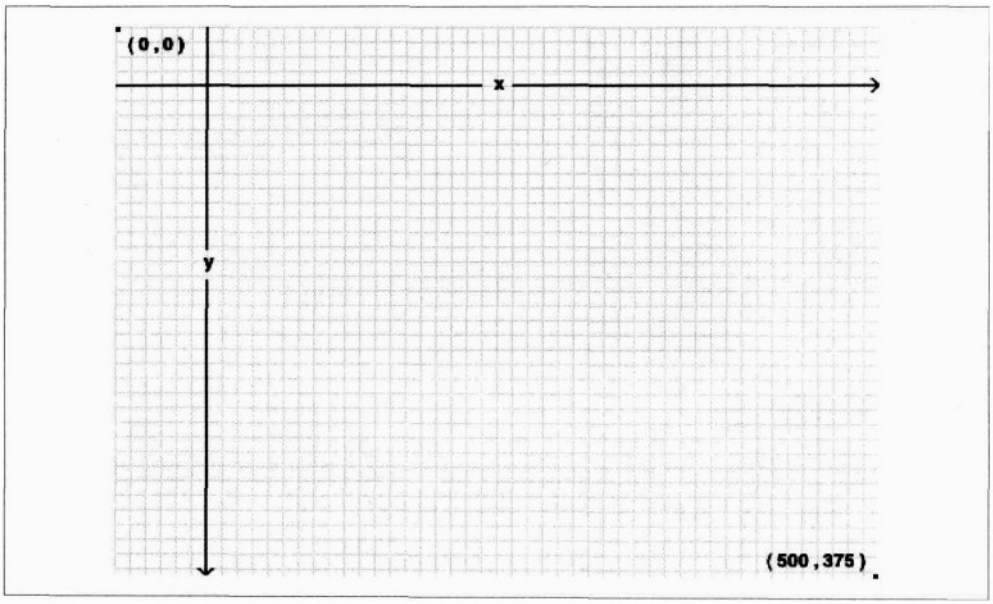


图 4.7　canvas 上的 canvas 坐标图

4.6 颜色渐变
Gradients

	IE[a]	Firefox	Safari	Chrome	Opera	iPhone	Android
线性渐变	7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
径向渐变	·	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 支持。

在前面的几节中，你学会了如何绘制一个纯色的矩形（见 58 页“简单的图形”一节），学会了怎么画纯色的线条（参见第 61 页“路径”一节）。但图形和线条不仅限于纯色。你可以用颜色渐变绘出魔幻般的色彩效果。图 4.8 演示了一个例子。

标记看起来和其他 canvas 一样：

```
<canvas id="d" width="300" height="225"></canvas>
```

首先，我们需要获取<canvas>元素并取得它的绘图上下文：

```
var d_canvas = document.getElementById("d");
var context = d_canvas.getContext("2d");
```

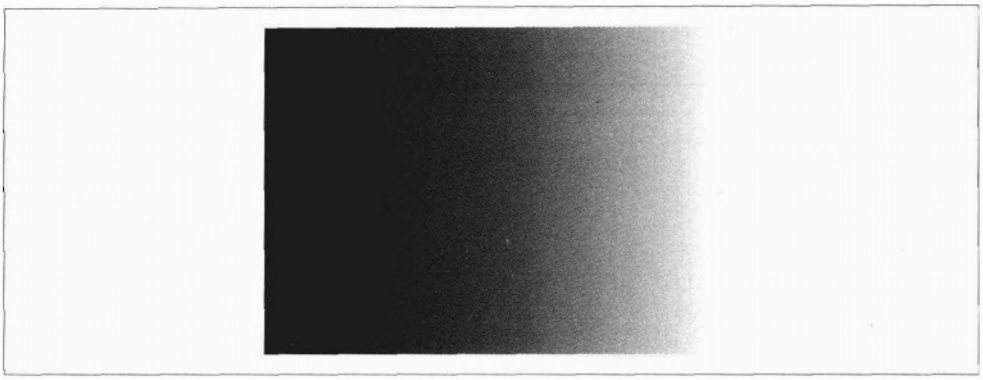


图 4.8　从左至右的线性渐变

一旦我们拥有了绘图上下文，就可以开始定义一个颜色渐变。渐变是两种或更多颜色的平滑过渡。canvas 的绘图上下文支持两种类型的渐变：

- createLinearGradient(*x0* , *y0* , *x1* , *y1*)沿着直线从(*x0* , *y0*)至(*x1* , *y1*)绘制渐变 。
- createRadialGradient(*x0* , *y0* , *r0* , *x1* , *y1* , *r1*) 沿着两个圆之间的锥面绘制渐变。前三个参数代表开始的圆，圆心为(*x0* , *y0*)，半径为 *r0*。最后三个参数代表结束的圆，圆心为(*x1* , *y1*)，半径为 *r1*。

让我们先创建一个线性渐变。渐变可以是任意大小，但我们想让这个渐变和画布一样宽，所以设置为 300 像素，像这样：

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

由于 y 的值（第二个和第四个参数）都是 0，这个渐变会沿水平从左往右。

有了一个渐变的对象，接下来我们就可以定义渐变的颜色。一个渐变可以有两种或更多种的色彩变化。沿着渐变方向颜色可以在任何地方变化。要增加一种颜色变化，你需要指定它在渐变中的位置。渐变位置可以在 0 和 1 之间任意取值。

让我们定义一个渐变，色调从黑到白过渡：

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

定义了一种渐变后，它只是保存在内存当中，而不会直接在 canvas 上画出任何东西。要让颜色渐变产生实际效果，你需要设置你图形的 fillStyle 为这个渐变对象，并绘制这个图形，比如画一个矩形或直线：

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

图 4.9 演示了这个效果。

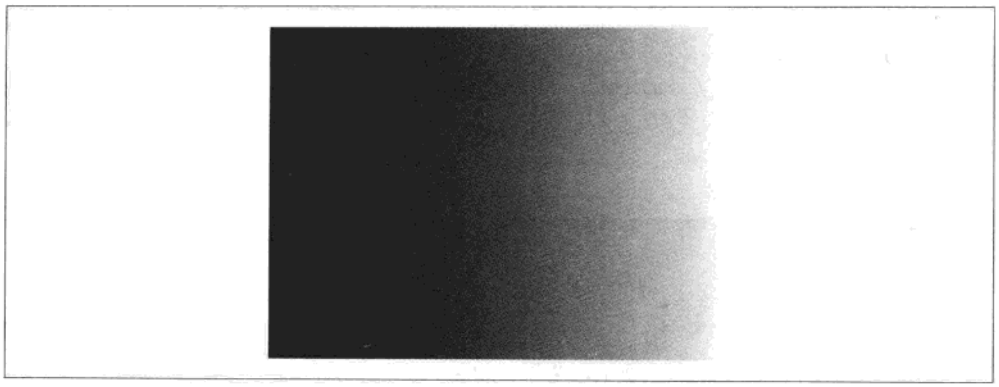


图 4.9 从左到右的渐变

假如你想有一个色调从上到下变化的渐变，那你创建渐变对象时，应该让 x 的值（第一个和第三个参数）为一个常数，而让 y 的值（第二个和第四个参数）从 0 到 canvas 的高度之间取值：

```
var my_gradient = context.createLinearGradient(0, 0, 0, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

图 4.10 演示了这个效果。

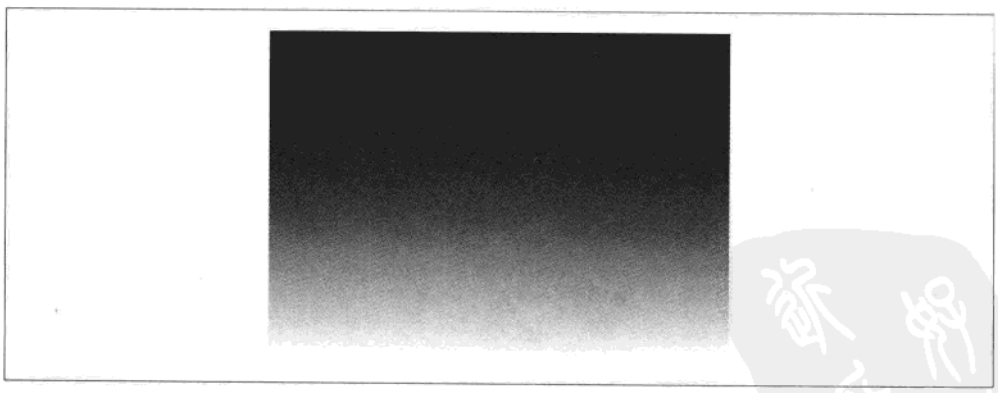


图 4.10 一个由上到下的线性渐变

你还可以创建沿对角线的渐变。例如：

```
var my_gradient = context.createLinearGradient(0, 0, 300, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

图 4.11 演示了这个效果。

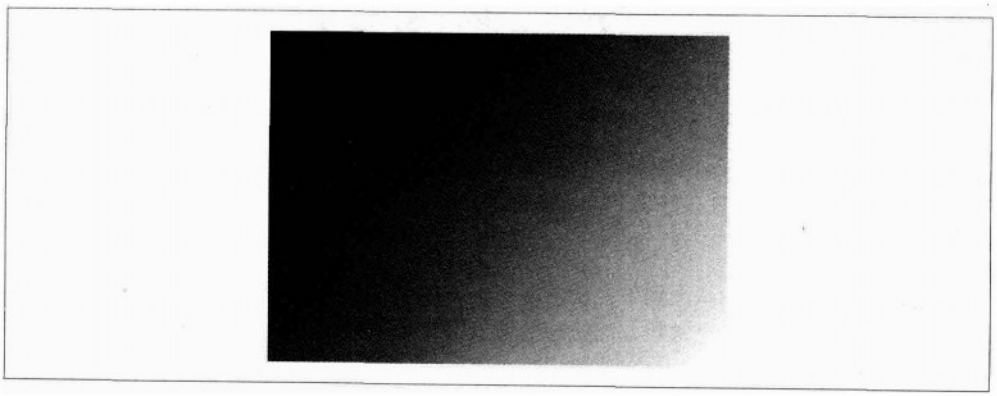


图 4.11 对角线线性渐变

4.7 图片

Images

IE [a]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

[a] Internet Explorer 需要第三方库 explorercanvas 的支持。

图 4.12 显示了一只猫，它是用<img>元素引入的。

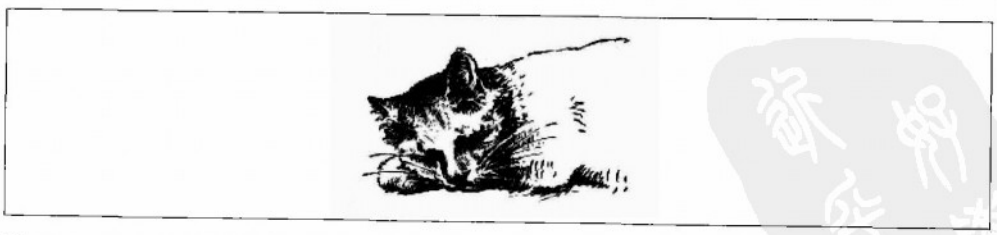


图 4.12 <img>元素显示的猫

图 4.13 显示了同样的一只猫，它绘制在一个<canvas>元素中。

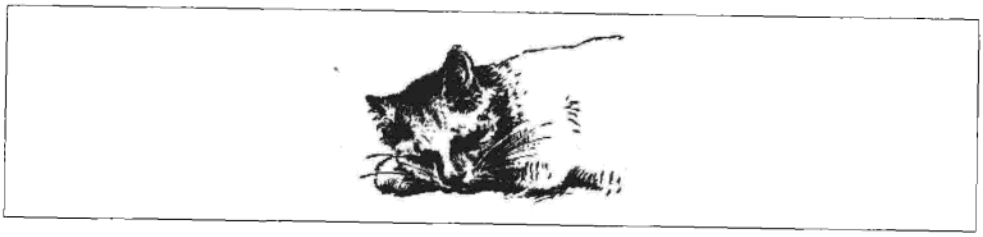


图 4.13 <canvas>元素里绘制的猫

canvas 的绘图上下文中定义了几种绘制图片的方法：

- drawImage(*image* , *dx* , *dy*)接受一个图片，并将之画到 canvas 中。给出的坐标(*dx* , *dy*)代表图片的左上角。比如，坐标(*0*, *0*)将把图片画到 canvas 的左上角。
- drawImage(*image* , *dx* , *dy* , *dw* , *dh*)接受一个图片，将其缩放为宽度 *dw* 和高度 *dh*，然后把它画到 canvas 上的(*dx* , *dy*)位置 。
- drawImage(*image* , *sx* , *sy* , *sw* , *sh* , *dx* , *dy* , *dw* , *dh*)接受一个图片，通过参数(*sx* , *sy* , *sw* , *sh*)指定图片裁剪的范围 ，缩放到(*dw* , *dh*)的大小，最后把它画到 canvas 上的(*dx*, *dy*)位置 。

HTML5 的规范这样解释 drawImage()的参数（*http://bit/ly/9WTZAp*）：

原矩形在图片中的四个顶点为：(*sx* , *sy*)，(*sx* + *sw* , *sy*)，(*sx* + *sw* , *sy* + *sh*)，(*sx*, *sy*+*sh*)。

目标矩形在 canvas 中的四个顶点为(*dx* , *dy*)，(*dx* + *dw* , *dy*)，(*dx* + *dw* , *dy* + *dh*)， (*dx* , *dy* + *dh*)。

图 4.14 给出了这些参数的可视化表示。

要在 canvas 上绘制图像，你需要先有一个图片。这个图片可以是已存在的<img>元素，或者通过 JavaScript 创建。无论采用哪种方式，你需在绘制到 canvas 之前，完全加载这张图片。

如果你使用已存在的<img>元素，你可以在 window.onload 事件处理器里安全地把它绘制到 canvas 中：

```
<img id="cat" src="images/cat.png" alt="sleeping cat" width="177" height="113">
<canvas id="e" width="177" height="113"></canvas>
<script>
window.onload = function() {
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = document.getElementById("cat");
  context.drawImage(cat, 0, 0);
};
</script>
```

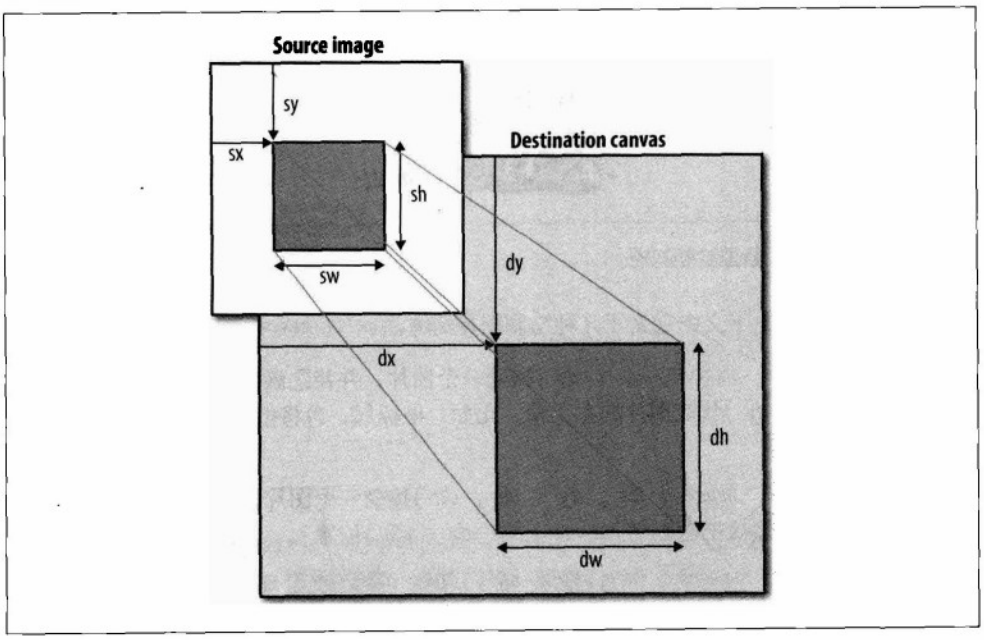


图 4.14　drawImage()方法如何把一张图片映射到 canvas

如果你通过 JavaScript 来创建图片对象，那你可以在 Image.onload 事件处理器中安全地把图片绘制到 canvas 中：

```
<canvas id="e" width="177" height="113"></canvas>
<script>
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = new Image();
  cat.src = "images/cat.png";
  cat.onload = function() {
    context.drawImage(cat, 0, 0);
  };
</script>
```

drawImage()方法的第三个和第四个可选的参数控制图像的缩放。图 4.15 显示了同样的一幅猫的图像，猫的图像被缩小到了原来的一半，并且被重复绘制在同一个 canvas 中的不同坐标位置上：

实现的脚本如下：

```
cat.onload = function() {
  for (var x = 0, y = 0;
       x < 500 && y < 375;
       x += 50, y += 37) {
    context.drawImage(cat, x, y, 88, 56);
```

不过这样也引出了一个问题：为什么起初你要使用 canvas 去绘制图像呢？为什么不使用<img>元素，并附加一些 css 样式来实现呢？即使是“叠猫图”的效果也可以通过重复 10 个<img>元素完成。

一种简单的回答是：这与你为何在 canvas 上绘制文本的理由一样。（参见第 63 页“文本”一节）。canvas 坐标系图表（参见第 60 页“Canvas 坐标系”一节 ）中包含了文字，线条和图形；文本只是其中的一小部分。更复杂的图表可以简单地通过调用 drawImage() 来绘制小图标、合成背景图或者其他一些图形。

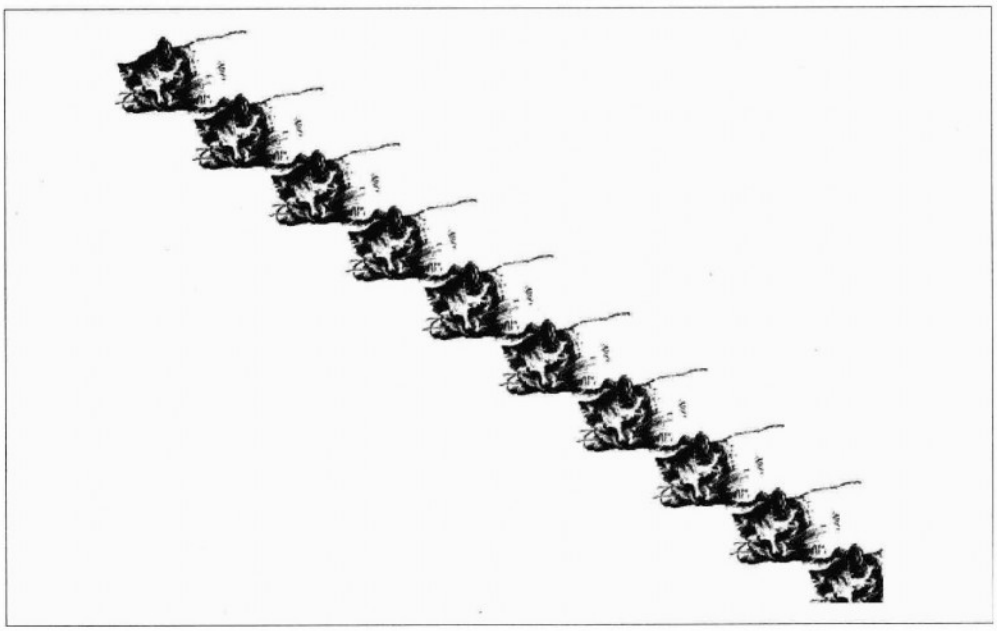


图 4.15 叠猫图

4.8 IE 怎么办？
What About IE?

虽然 IE 浏览器（在我写这本书的时候，最新的版本是 IE8）并不支持 canvas API，但是它自己有一个叫做 VML 的技术，可以完成<canvas>元素能做的大部份工作。因此，*excanvas.js* 就诞生了。

ExplorerCanvas (*http://code.google.com/p/explorercanvas/*) —excanvas.js—是一个开源的，采用 Apache license 许可的 JavaScript 类库，它在 IE 中实现了 canvas 的 API。在页面的顶部引入下面的<script>元素就可以使用它了。

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML 5</title>
  <!--[if IE]>
    <script src="excanvas.js"></script>
  <![endif]-->
</head>
<body>
  ...
</body>
</html>
```

<!--[if IE]>和<![endif]-->是条件注释。IE 会把它们当作 if 语句一样处理："如果当前的浏览器是任何版本的 IE 浏览器，就执行这一段代码。"而其他类型的浏览器都会把他们当作 HTML 的注释。最终的结果就是，IE 浏览器会下载 excanvas.js 的脚本，而其他类型的浏览器则完全忽略这段脚本（不会下载，也不会执行，什么都不会做）。这对于原生支持 canvas API 的浏览器来说可以加快页面的载入。

在<head>里引入 *excanvas.js* 之后，你不需要再做任何事情来迁就 IE 浏览器。只需要添加<canvas>元素或者通过 JavaScript 动态创建它。遵循本章介绍的知识，取得<canvas>元素的绘图上下文，然后你就可以在<canvas>元素中任意绘制图形，文本和图案了。

这多少有点夸张，其实还是有一些限制的：

- 颜色渐变（参见第 67 页"颜色渐变"一节）只能是线性的。不支持径向渐变。
- 模式必须在两个方向上重复。
- 不支持裁剪。
- 非等比缩放，不保证 stroke 也做对应缩放。
- 它很慢。这对任何人来说并不特别意外，因为相比其他浏览器，IE 的 JavaScript 解释器原本就很慢。当你试图用脚本库去模拟绘制复杂的图形，必然会是一场灾难。对于简单的画条线或者转换一个图片，你可能不会发现性能有什么下降，但一旦你开始在 canvas 上做动画或其他快速变化的事情，你就会立刻感觉到。

在我使用 excanvas.js 完成本章的例子的过程中，我发现了一个问题，希望读者也注意一下。一旦你引入 excanvas.js，它就会立刻初始化那些伪 canvas 接口。但这并不意味着你可以立马开始使用它绘图。在某些情况下，伪 canvas 接口还没有初始化完全，你就开始使用这些接口，比如获取上下文环境，那 IE 浏览器就会告诉你“对象没有此属性或方法”。

最简单的方式就是在 onload 事件触发后再执行 canvas 相关的所有操作。如果页面有很多图片或者视频的话，这会等待一段时间，但它能保障 ExplorerCanvas 的使用。

4.9 一个完整的例子

A Complete Example

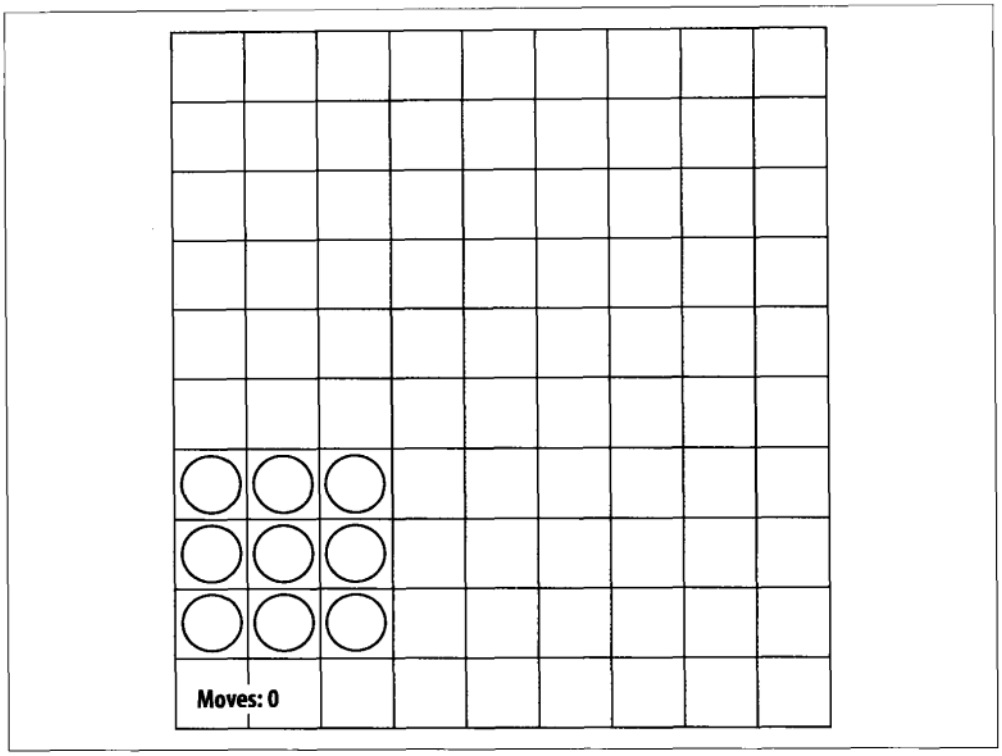
Halma 是一个古老的棋盘游戏。现在有很多变体。在下面的例子中，我会创建一个单人游戏的版本，在 9×9 的棋盘里放上一些棋子。比赛开始时，棋子全部放置于棋盘左下角 3×3 的区域。游戏的目标是用最少的移动步骤把这些棋子移动到棋盘的右上角 3×3 的区域。

有两种合法的移动：

- 选中一个棋子，移动到相邻的空格子里。一个“空的”格子是目前还没有放置棋子的格子。“相邻的”格子表示紧挨着它的东、南、西、北以及西北、东北、西南和东南 8 个方向的格子。（棋盘里棋子是无法越界的。如果一个棋子已经是最左边了，那它就不能再往西、西北或者西南移动。）
- 选中一个棋子，从相邻的一个棋子头上跳过去，可以重复。也就是说，如果你跳过了一个相邻的棋子后，你可以继续再跳过一个棋子达到新的位置，但这只算一次移动。事实上，任何次数的跳跃都只算一次移动。（既然游戏的目标就是最少的移动，那就应该较好的构建一个布局，让棋子可以多次方便的跳跃相邻的棋子。）

图 4.16 是一张游戏截图，你也可以在线玩（*http://diveintohtml5.org/examples/canvas-halma.html*），如果你想通过浏览器开发工具来探个究竟的话。

它是怎样工作的呢？很高兴你提到这个问题。我不会在这里贴出所有代码（你可以在 *http://diveintohtml5.org/examples/halma.js* 看到全部代码）。我会跳过大部分的游戏代码，但我想着重说明那些在<canvas>元素上绘图以及响应鼠标点击事件的代码。



4.16 Halma 游戏开始时的样子

页面加载过程中，我们先对游戏做一些初始化，包括设置<canvas>的宽度和高度，以及保存一个绘图上下文的引用。

```
gCanvasElement.width = kPixelWidth;
gCanvasElement.height = kPixelHeight;
gDrawingContext = gCanvasElement.getContext("2d");
```

然后我们给<canvas>元素添加一个事件处理器来监听点击事件——这是之前你没有见过的。

```
gCanvasElement.addEventListener("click", halmaOnClick, false);
```

当用户点击 canvas 中的任何位置时，halmaOnClick()函数就会被调用。函数传入的参数是 MouseEvent 对象，该对象包含了用户点击的位置信息。

```
function halmaOnClick(e) {
    var cell = getCursorPosition(e);

    // 余下的是游戏逻辑部分
    for (var i = 0; i < gNumPieces; i++) {
```

```
        if ((gPieces[i].row == cell.row) &&
            (gPieces[i].column == cell.column)) {
            clickOnPiece(i);
            return;
        }
    }
    clickOnEmptyCell(cell);
}
```

接下来的步骤是获取 MouseEvent 对象并计算是 Halma 棋盘的哪一个方格被点击了。Halma 棋盘占据了整个 canvas，所以 canvas 中的任何点击都位于棋盘内。我们要做的是计算出点击的具体是哪个方格。这有些棘手，因为不同浏览器对鼠标事件有不同的实现：

```
function getCursorPosition(e) {
    var x;
    var y;
    if (e.pageX || e.pageY) {
      x = e.pageX;
      y = e.pageY;
    }
    else {
      x = e.clientX + document.body.scrollLeft +
           document.documentElement.scrollLeft;
      y = e.clientY + document.body.scrollTop +
           document.documentElement.scrollTop;
    }
```

在这里，x 和 y 是相对于文档的（也就是整个 HTML 页面）。但希望得到相对于 canvas 的坐标值。我们可以这么做：

```
x -= gCanvasElement.offsetLeft;
y -= gCanvasElement.offsetTop;
```

现在得到了相对于 canvas 的坐标值（参见第 60 页“Canvas 坐标”一节）。也就是说，如果 x 和 y 都是 0，那就说明用户点击的是 canvas 的左上顶角。

如此一来，我们就可以计算出用户点击的是哪一个 Halma 方格，进而采用相应的行动：

```
    var cell = new Cell(Math.floor(y/kPieceWidth),
                        Math.floor(x/kPieceHeight));
    return cell;
}
```

嘿！虽然鼠标事件棘手，但是你可以把同样的逻辑（事实上，同样的代码）用在其他基于 canvas 的应用中。记住：鼠标点击→文档相对坐标→canvas 相对坐标→应用程序特定的代码。

OK，让我们来看看主绘图程序。考虑到图形很简单，所以游戏中的任何改变，我都重绘了整个棋盘。但这不是必须的。无论你在 canvas 上面绘制过什么，甚至用户的鼠标滚动

导致 canvas 离开了当前屏幕，或者切换到其他 Tab 后再切换回来，canvas 的绘图上下文都会保存所有信息。如果你正在开发一个有复杂图形界面的游戏（比如街机游戏），你可以记录 canvas 中发生变化的区域且只重绘这个区域来优化游戏的性能。但这超出了本书讨论的范围。下面是棋盘清理的代码：

```
gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);
```

棋盘绘制函数应该看起来很熟悉吧。这和我们绘制 canvas 坐标系图表十分类似（参见第 60 页“Canvas 坐标系”一节 ）：

```
gDrawingContext.beginPath();

/* vertical lines */
for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
    gDrawingContext.moveTo(0.5 + x, 0);
    gDrawingContext.lineTo(0.5 + x, kPixelHeight);
}

/* horizontal lines */
for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
    gDrawingContext.moveTo(0, 0.5 + y);
    gDrawingContext.lineTo(kPixelWidth, 0.5 +  y);
}

/* draw it! */
gDrawingContext.strokeStyle = "#ccc";
gDrawingContext.stroke();
```

真正有意思的是如何绘制每一个棋子。一个棋子就是一个圆，这是我们前面没有画过的。而且，当用户选择好一个棋子并打算移动它时，我们需要填充这个圆。这里，参数 p 代表一个棋子，有行和列两个属性，用来标识棋子在棋盘中的位置。我们使用一些游戏中的常量来把(column, row)换算为坐标系中的坐标（x, y），然后绘制这个圆，（如果被选择的话）最后填充它。

```
function drawPiece(p, selected) {
    var column = p.column;
    var row = p.row;
    var x = (column * kPieceWidth) + (kPieceWidth/2);
    var y = (row * kPieceHeight) + (kPieceHeight/2);
    var radius = (kPieceWidth/2) - (kPieceWidth/10);
```

这就是游戏最终的逻辑。现在我们把 canvas 中圆的圆心与坐标系(x, y)建立好了联系。但 canvas API 中并没有 circle()方法，只有一个 arc()方法。是的，只有一个 arc 方法，这能画出一个圆吗？好好回想一下你的初等几何知识。arc 方法接受 3 个参数，圆心的的坐标(x, y)，一个半径，一个用弧度为单位的起始角和终止角及其方向标（false 为顺时针方向，true 为逆时针方向）。我们可以使用 Javascript 内置的 Math 模块来计算弧度。

```
gDrawingContext.beginPath();
gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
gDrawingContext.closePath();
```

别急！这样还不能绘制出图形。和 moveTo()、lineTo()方法一样，arc 方法只是一个“铅笔”方法（参见第 61 页“路径”一节）。要实际绘制圆形，我们需要设置 strokeStyle 和调用 stroke()方法：

```
gDrawingContext.strokeStyle = "#000";
gDrawingContext.stroke();
```

如果棋子被选择了呢？我们可以重用路径信息并用颜色填充它：

```
if (selected) {
    gDrawingContext.fillStyle = "#000";
    gDrawingContext.fill();
}
```

好吧，程序就介绍到这里。剩下的部分都是一些游戏特定的逻辑，比如判断移动是否合法，记录移动的次数，检测游戏是否结束等。9 个圆，一些直线，加上 onclick 处理函数，我们就在<canvas>中创建了整个游戏，太棒了！

4.10 扩展阅读

Further Reading

- Mozilla 开发者中心的 Canvas 教程。
  (*https://developer.mozilla.org/en/Canvas_tutorial*)
- Mihai Sucan 编写的“HTML5 canvas 基础知识”
  (*https://dev.opera.com/articles/view/html-5-canvas-the-basics/*)。
- Canvas 演示：HTML <canvas> 元素的演示，制作工具和教程
  (*https://www.canvasdemos.com*)。
- HTML5 标准草案中的“canvas 元素”
  (*https://bit.ly/9JHzOf*)。

网络上的视频
Video on the Web

5.1 前言
Diving In

过去四年当中，只要访问过 YouTube.com 的人就会知道网页是可以嵌入视频的。但在 HTML5 之前，还没有一个标准的方式。因此，在网页中看到的视频，都是通过第三方插件的方式嵌入的。可能是 QuickTime，也可能是 RealPlayer 或者 Flash（YouTube 使用的是 Flash）。浏览器很好地整合了这些插件，你完全意识不到它们的存在，除非你使用的平台不支持这些插件。

HTML5 定义了一个在网页中嵌入视频的标准方式，这就是使用<video> 元素。对<video>元素的支持工作还在进行中，谨慎地说，目前<video>元素还无法工作（至少还不能在所有浏览器中被支持）。但是不要绝望！还是有一些选择。表 5.1“<video>元素支持情况”显示了在写本书时哪些浏览器支持<video>元素。

表 5.1 <video>元素支持情况

IE9	IE8	IE7	Firefox3.5	Firefox3.0	Safari 4	Safari 3	Chrome	Opera
√	·	·	√	·	√	√	√	√

对<video>元素的支持这件事本身就只是故事的一小部分。在我们讨论 HTML5 中的视频之前，需要先来了解一下视频本身的知识（如果你已经了解视频的相关知识，就可以直接跳到第 88 页“在网页中怎么工作”）。

5.2 视频容器
Video Containers

你可能会认为视频文件就是“AVI 文件”或“MP4 文件”。但事实上，“AVI”和“MP4”仅仅只是容器的格式。视频容器格式只决定怎么将视频存储起来，而不关系存储的内容，

这和 ZIP 压缩文件一样。（不过，也没有那么简单，并不是所有的视频流都与每种格式兼容，不过目前我们不用太过关心。）

一个视频文件通常包含多个轨道：一个视频轨道（无音频），加上一个或多个（不含视频的）音频轨道。轨道是相互关联的。音频轨道包含了一些标记，以此辅助音频与视频同步。每条轨道都可以有各自的元数据，例如视频轨道的纵横比或音频轨道的语言。容器也可以有自己的元数据，比如视频的标题，视频的艺术封面，（电视剧的）集数等。

存在很多视频容器格式。但我们只列举一些最流行的：

MPEG-4

通常以.mp4 或者.m4v 为扩展名。MPEG-4 容器是基于旧的苹果 QuickTime 容器格式（.mov）。苹果网站的电影预告片仍然使用旧的 QuickTime 容器，但从 iTunes 里买到的影片使用的则是 MPEG-4 容器格式了。

Flash 视频

通常以.flv 为扩展名。Flash 视频显然使用的是 Adobe 的 Flash 容器格式。Flash 9.0.60.184 版本之前（也就是 Flash 播放器第 9 版本第 3 次更新），它是 Flash 视频唯一支持的容器格式。最近一些版本的 Flash，也开始支持 MPEG-4 容器格式。

Ogg

通常以.ogv 为扩展名。Ogg 是一个开放的标准，它完全开源且不受任何已知专利的束缚。Firefox 3.5、Chrome 4 和 Opera 10.5 都对其提供了原生支持，这样对于 Ogg 容器格式、Ogg 视频（也称“Theora”）、Ogg 音频（也称“Vorbis”）不需要任何平台的特殊插件来支持。桌面电脑中主流的 linux 发行版都支持 Ogg 格式，Windows 和 Mac 上面也可以通过安装 QuickTime 组件或者 DirectShow 过滤器来播放 Ogg 格式的视频。当然，对于极为优秀的 VLC（*http://www.videolan.org/vlc/*）而言，在什么平台都可以播放。

WebM

以.webm 为扩展名。WebM 是一个比较新的容器格式，技术实现上和 Matroska 很类似。2010 年的 Google I/O 大会中，WebM 公开亮相。它设计为 VP8 视频编解码器和 Vorbis 音频编解码器专用的容器格式（稍后再详细介绍）。WebM 将在下个版本的 Chromium，Google 的 Chrome，Mozilla 的 Firefox 及 Opera 中被原生支持。Adobe 公司也宣称在下一个版本的 Flash 中支持 WebM 视频。

音频视频交错（Audio Video Interleave）

通常以.avi 为扩展名。AVI 容器格式是微软发明的，那个时候，电脑可以播放视频是一件很神奇的事情。但微软官方并不支持现在很多其他视频容器格式的特性。元数据也

不被支持。不仅如此，它甚至连当今大部分的视频或音频编解码器都不支持。随着时间的推移，各家公司都试图扩大对各种视频音频的兼容性，而 AVI 目前仍然是一些流行的编码器的默认容器格式，比如 MEncoder（*http://www.mplayerhq.hu/DOCS/HTMHen/encoding-guide.html*）。

5.3 视频编解码器
Video Codecs

当我们谈到“看一部视频”的时候，往往指的是视频和音频的组合。但只有一个“视频”文件，而不是两个不同的文件。比如 AVI 文件或者 MP4 文件。正如前面一节所说，这些都只是一种容器的格式，如同 ZIP 文件一般，可以用来存放各种形式的文件。同时，容器格式也规定了怎样在一个文件中存储视频和音频流。

当你想要“观看一个视频”的时候，视频播放器会做下面几项工作：

- 解析容器格式以找出可以使用的视频和音频轨道，并分析出它们的存储结构，以便接下来的解码工作。
- 对视频流解码，并在屏幕显示一幅幅的图像。
- 对音频流解码，同时给扬声器传输声音信号。

一个视频编解码器（“编解码器”一词是个合成词，由“编码”和“解码”合成。）其实就是一个算法，依据它来对视频流编码。上面的第二条依据的规则就是这个算法。而视频播放器则根据视频的编码算法来进行视频流的解码，从而我们就可以在屏幕中看到一系列的图像，或者是“帧”。现在的视频编解码器会使用各种技巧减少来从一帧到另一帧过程中传递的信息数量。比如，不会存储每一帧的所有信息（就像快照一样），而只是存储两帧之间的差异信息。大部分的视频在两帧切换时并不会变更所有信息，这就可以保证一个比较高的压缩比，从而得到了更小的文件。

编解码器包括有损和无损两种。无损视频文件一般太大了，在网页中没有优势，所以我们重点研究有损编解码器。有损视频编解码器中，信息在编码过程的丢失是无法避免的。这就好比从一个磁带复制音频，每次复制都会失去一些原来音频的信息，复制音频的质量也会越来越差。因此要避免不停的“斯斯”的拷贝磁带，反复地对视频编码会导致其画面不均匀，特别是有一些大量动画的场景（实际上，很难避免，即使你直接从原版开始编码，但如果选择了一个较差的视频编解码器，或者传递了错误的参数，结果都会导致视频质量的下滑）。从好的方面来看，有损视频编解码器可提供惊人的压缩率。同时也可以使用很多技术来“欺骗”用户，让用户看到流畅的画面而感觉不到信息的缺失。

视频编解码器如恒河沙数，和我们这里最有关联的是 H.264、Theora 和 VP8。

5.3.1 H.264

H.264(*http://en.wikipedia.org/wiki/H.264*)被称为“MPEG-4 的第 10 部分”，又名“MPEG-4 AVC”、“MPEG-4 高级视频编码”。H.264 由 MPEG 组研发并于 2003 年标准化。它的目标是支持一切设备，无论是低带宽，低 CPU（比如手提电话），还是高带宽、高 CPU（比如现在的桌面电脑），或者在两者之间的设备。要做到这一点，H.264 标准被分成不同的几种“配置（profiles）”，不同的配置决定了文件的大小。高配使用了更多的特性，这会导致在解码过程中更加消耗 CPU，但视频文件本身会更小，视频效果也更好。

为了让读者对不同的配置情况有个粗略了解，可以看看这些内容：苹果 iPhone 手机满足基本配置（Baseline），苹果电视机顶盒支持基本配置（Baseline）和主配置（Main）两种，桌面电脑中的 Aodobe Flash 支持基本（Baseline）、主（Main），以及高级（High）三种配置。YouTube（隶属于谷歌，我的雇主）现在采用 H.264 来编码高清晰视频，并通过 Adobe Flash 播放，YouTube 也提供移动设备上 H.264 编码的视频，包括苹果的 iPhone 及运行谷歌 Android 操作系统的手机（High）。此外，H.264 也是蓝光规格指定的视频编解码器之一，蓝光光盘一般使用它的高级配置。

大多数非 PC 设备（比如 iPhone、蓝光播放器）由于它们的 CPU 还远不足以实时解码 H.264 视频，所以只能通过专用的芯片实现解码。许多桌面电脑的显卡也支持硬件解码 H.264 视频。目前，有很多优秀的 H.264 编码器，包括开源的 x264 库。H.264 标准申请了专利保护，通过 MPEG LA 组来授权。H.264 视频能够嵌入在大多数流行的容器格式（参见第 81 页“视频容器”一节），包括 MP4（主要用于苹果的 iTunes 商店）和 MKV（非商业视频爱好者）。

5.3.2 Theora

Theora (*http://en.wikipedia.org/wiki/Theora*) 由 VP3 编解码器演变而来，随后由 Xiph.org 基金会开发。Theora 无许可费，也无任何专利束缚，其原型 VP3 是受专利保护的，不过也免版权费用。虽然标准自 2004 年以来已经被“冻结”，Theora 项目（其中包括开放源码的编码器和解码器）2008 年 11 月才发布版本 1.0，而 2009 年 9 月才发布版本 1.1。

Theora 视频可以嵌入至任何容器格式中，但它多嵌入到 Ogg 容器。所有主流的 Linux 发行版都无缝支持 Theora，Mozilla 的 Firefox3.5 对于基于 Ogg 容器格式的视频，提供 Theora

的原生支持。所谓“原生”，是指“在所有平台不需要任何特定的插件”。只要安装 Xiph.org 的开源的解码器软件你也可以在 Windows 或者 Mac OS X 上播放 Theora 视频。

5.3.3 VP8

VP8（*http://en.wikipedia.org/wiki/VP8*）也是由 On2 公司开发，也就是开发 VP3 的那家公司（后来开发 Theora）。从技术角度看，它相当于 H.264 基本配置的质量，不过有大量改进的潜力。

2010 年，谷歌收购了 On2，将其视频的编码解码规范公开，同时将一个简单的视频编码器和解码器开源。此外，谷歌还“开源”了 On2，授权免费使用其有关 VP8 的专利许可。（对于申请过专利的技术这是最好的事情了。为使他们开源友好，授权其免版权费用，以后任何人都可以使用这项技术并不需要支付任何费用了。）从 2010 年 5 月 19 日开始，VP8 成为一个现代的、免版权费用的编解码器，而且不受任何专利限制，而 On2(目前隶属于 Google）还是受专利保护的，不过已经免版权费用了。

5.4 音频编解码器
Audio Codecs

除非你坚持要发 1927 年以前的片子，否则肯定想要你的视频有声音。和视频编解码器一样，音频编解码器也就是一个编码算法，用于编码音频流。正如视频编解码器，也包括有损和无损音频编解码器两种。和无损视频一样，对于放到网上，无损音频实在是太大了，所以我们重点讨论有损音频编解码器。

其实，我们可以进一步聚焦，因为有损音频编解码器也有不同的种类。相比视频，音频应用的范围更加广泛（比如电话），所以对于不同的分类，都会有一套音频编解码器的编码速度优化方案。翻录音乐 CD 时，你显然不会使用这些编解码器，倘若使用的话，会导致声音如同四岁小孩在胡乱歌唱一般。但受宝贵的带宽所迫，你会在 Asterisk PBX 中使用，将语音压缩到一个可以接受的程度。然而，由于浏览器及第三方插件都没有提供很好的支持，语音优化的音频编解码器还没有基于网页的方案。因此，我们可以把讨论的范围集中到通用的音频编解码器。

正如我在 83 页“视频编解码器”一节提到，当你“观看视频的时候”计算机会同时做以下几件事：

1. 解析容器格式。
2. 对视频流解码。

3. 对音频流解码，同时给扬声器传输声音信号。

音频编解码器对于第 3 步是这样执行的：对音频流解码并转换为电子波形，然后扬声器将电子波形转化为声音。为了减少音频流中存储的信息量，也会如视频解码器一般采用各种技巧。而且，由于我们讨论的是有损音频编解码器，音频信息会随着录音→编码→解码→收听这些环节不断丢失。不同的音频编解码器扔掉不同的信息，但它们都具有共同的目的：欺骗你的耳朵，使你注意不到缺少了信息。

对于音频有一个概念是视频没有的，那就是声道。我们正在向扬声器发送声音信号，对吗？那么，你有多少个扬声器呢？如果你坐在电脑旁，可能只有两个：左一个，右一个。我的台式机有三个：左右各一个，地板上还有一个。“环绕声”系统有六个或更多的扬声器，按照一定策略放在房间的四周。每个扬声器负责原始音频的一个声道。理论上，当你坐在 6 个扬声器中间时，将被 6 个不同的声道发出的声音环绕，这时你的大脑会将它们合成起来，由此你就会有身临其境的感觉。现实中，真的可以做到吗？一个数十亿美元的产业，似乎这样认为。

大部分通用的音频编解码器可以处理两个声道。在录制过程中，声音分为左、右声道，而在编码过程中，两个声道都存储在同一个音频流中；在解码时，两个声道都会被解码，并被发送到相应的扬声器。某些音频编解码器可以处理超过两个的声道并负责把每个声道定位到准确的扬声器。

音频编解码器恒河沙数。我也是这样描述视频编解码器的吧？先忘记那些。虽有大量的音频编解码器，但网络上，其实只有三种是你需要了解的：MP3、AAC 和 Vorbis。

5.4.1 MPEG-1 音频层 3

MPEG-1 音频层 3（*http://en.wikipedia.org/wiki/MPEG-1 Audio Layer 3*）通常被称为“MP3”。如果你还没有听说过 MP3，那我也没有办法了。沃尔玛出售一些便携式音乐播放器，并称其为“MP3 播放器”。沃尔玛总该知道吧……

MP3 的歌曲可以最多包含两个声音声道。它们可以被编码为不同的比特率：64 kbps、128 kbps、192 kbps 等从 32 至 320 中间的一种。比特率较高意味着文件更大、音质更好，但音质和比特率的关系并非线性关系（128 kbps 的音质是 64 kbps 的两倍以上，但 256 kbps 的音质并非为 128 kbps 两倍）。由于 MP3 格式（1991 年标准化）允许使用可变的比特率编码，这就意味着某些编码的部分可以比其他部分压缩得更小。举例来说，音符间的停顿可以用一个非常低的比特率来编码，而当有多个设备需要播放一段复杂的和弦时，就可以

提升到一个比较高的比特率。MP3 音乐也可以用恒定的比特率编码，毫不奇怪，这被称为恒定比特率编码。

MP3 标准并没有定义究竟如何编码（但它定义了如何解码），不同的编码器会使用不同心理声学模型，由此产生很不相同的音效。尽管编码千差万别，但都可以用同一个播放器解码。开源 LAME 项目可以说是最好的免费的编码器，对于低比特率而言，甚至可以说是最好的。

MP3 格式受专利保护，这就是为什么 Linux 不能直接播放 MP3 文件的原因了。相当多的便携式音乐播放器支持独立的 MP3 文件和嵌入于任何视频容器中的 MP3 音频流。AdobeFlash 可以支持独立的 MP3 文件和嵌入于 MP4 视频容器中的 MP3 音频流。

5.4.2 高级音频编码

高级音频编码(*http://en.wikipedia.org/wiki/Advanced-Audio-Coding*)常称为“AAC 格式”。它于 1997 年标准化，当时苹果把 AAC 作为 iTunes 的默认存储格式，使它一举成名。最初，所有从 iTunes 商店里“购买”的 AAC 文件都使用了苹果公司专有的数字版权管理方法进行了加密，这种技术叫 FairPlay。iTunes Store 中许多歌曲现在是无保护的 AAC 文件，苹果称之为“iTunes Plus”，因为它听起来比“iTunes Minus”更好。AAC 格式有专利保护，授权价格可在网上找到。

AAC的目的是在相同的比特率下提供比 MP3 更好的音质，它可以按任意比特率编码(而 MP3 仅限于固定在一定范围内的比特率，320 Kbps 就是其上限）。AAC 理论上可以编码高达 48 个声道，但目前实际上并做不到。AAC 格式和 MP3 还有一点不同，那就是它可以自定义多个配置文件，这和 H.264 如出一辙。“低配”是为一些低 CPU 的实时设备而准备的，对于高配而言，就可以在相同比特率下提供更好的音质，代价就是更慢的编码和解码。

目前所有的苹果产品，包括 iPod、Apple TV，以及 QuickTime 都在一定配置下支持独立的 MP3 音频文件和嵌入于 MP4 视频容器中的 MP3 音频流。Adobe Flash 同样支持任何配置下 MP4 容器中的 AAC 音频流，同样开源的 MPlayer 和 VLC 视频播放器也是如此。如果要对 AAC 格式编码，开源的 FAAC 库是一个不错的选择，而且它支持 mencoder 和 ffmpeg 的编译时间（Compile-time）选项。

5.4.3 Vorbis

Vorbis（*http://en.wikipedia.org/wiki/Vorbis*）经常被称为是“Ogg Vorbis”但这从技术讲是不正确的，因为“Ogg”只是一个容器格式（参见第 81 页“视频容器”一节），而且 Vorbis

音频流也可以嵌入到其他视频容器。Vorbis 没有受专利保护，因此在众多 Linux 发行版中，都可以直接播放，它也可以在装有开源的 Rockbox 固件的便携式设备中播放。Mozilla 公司的 Firefox3.5 支持 Ogg 容器中的 Vorbis 音频文件，或者 Ogg 视频中 Vobis 音轨。Android 的手机也能独立播放 Vorbis 音频文件。Vorbis 音频流通常嵌入在 Ogg 或者 WebM 容器中，但它也可以嵌入于一个 MP4 或者 MKV 的容器，更甚至通过一些技巧，嵌入到 AVI 容器中。Vorbis 支持任意数量的声道。

有很多开源的 Vorbis 编码器和解码器，包括 OggConvert、ffmpeg、aoTuV 和 libvorbis。还有 OS X 上 QuickTime 组件及 Windows 中的 DirectShow 过滤器。

5.5 在网页中怎么工作

What Works on the Web

如果你还没有两眼发呆，那已经比很多人强了。你可以感觉到，视频（以及音频）是一个很复杂的主题，而这里只是一些简单的介绍。我敢断言，你会思考这和 HTML5 有什么关系。嗯，HTML5 中有一个<video>元素，可以在网页中嵌入视频。对于视频编解码器，音频编解码器，或者视频的容器格式都没有任何约束。每个<video>元素都可以链接多个视频文件，浏览器会选择第一个可以播放的文件。你应该了解每个浏览器都支持哪些容器格式和编解码器。

在撰写本文时，HTML5 视频目前被支持的情况如下：

- Mozilla 公司的 Firefox 浏览器（3.5 和更高版本）在 Ogg 容器格式中支持 Theora 视频和 Vorbis 音频。
- Opera（10.5 及更高版本）在 Ogg 容器格式中支持 Theora 视频和 Vorbis 音频。
- Google Chrome（3.0 和更高版本）在 Ogg 容器格式中支持 Theora 视频和 Vorbis 音频。同时，它还支持 MP4 容器格式中所有配置的 H.264 视频和 AAC 音频。
- 在撰写本文时（2010 年 6 月 9 日），Google Chrome “开发版”，每日构建版的 Chromium，每日构建版的 Firefox，Opera 的实验构建版都可以在 WebM 容器中支持所有的 VP8 视频和 Vorbis 音频。（访问 *webmproject.org* 可以获得更多信息和 WebM 兼容的浏览器下载链接。）
- Mac 和 Windows 上 Safari（3.0 和更高版本）可以支持任何 QuickTime 支持的格式。从理论上讲，你可以要求用户安装 QuickTime 第三方插件。但实际上，很少有人愿意这样做。因此你还是“无法直接播放”QuickTime 支持的格式。这是一个长长的清单，但不包括 Theora 视频、Vorbis 音频和 Ogg 容器。然而，QuickTime 确实可以支持 H.264 视频（主配置）和 MP4 中的 AAC 音频。

- 移动设备，如 iPhone 和 Google Android 手机都支持（基本配置的）H.264 视频和 MP4 容器中的（低配）AAC 音频。
- Adobe Flash（9.0.60.184 和更高版本）支持（所有配置的）H.264 视频，以及 MP4 容器中的（所有配置的）AAC 音频。
- IE 9 支持一些还没有形成规范配置的 H.264 视频及 MP4 容器中的 AAC 音频。
- IE 8 不支持 HTML5 视频元素，但所有的 IE 用户都通过安装 Adobe Flash 插件实现视频播放。本章后面的部分，会讲述怎么使用 HTML5 视频元素并优雅地兼容 Flash。

表 5.2“浏览器中视频编解码器的支持情况”以摘要的形式表述了上述信息。

表 5.2 浏览器中视频编解码器的支持情况

编解码器/容器	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora + Vorbis + Ogg	·	3.5+	·	5.0+	10.5+	·	·
H.264 + AAC + MP4	·	·	3.0+	5.0+	·	3.0+	2.0+
WebM	·	·	·	·	·	·	·

往后一年，局势会有意义重大的改变：WebM 将得到众多浏览器的支持，这些浏览器会陆续发布支持 WebM 的版本，用户也会升级到新的版本。预期的编解码器支持情况在表 5.3 “即将推出的浏览器中视频编解码器的支持情况”中。

表 5.3 即将推出的浏览器中视频编解码器的支持情况

编解码器/容器	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	·	3.5+	·	5.0+	10.5+	·	·
H.264+AAC+MP4	·	·	3.0+	5+	·	3.0+	2.0+
WebM	9.0+[a]	4.0+	·	6.0+	11.0+	·	[b]

[a] “当用户安装了 VP8 编解码器”后，IE9 将才支持 WebM 容器格式，这意味着微软不会直接在 IE9 中包含 VP8 编解码器。

[b] 谷歌一直致力于在 Android “未来的版本”中支持 WebM 视频，但目前还没有公布具体时间。

听累了吧，那休息一下。

专家发言

目前还没有一种编解码器和容器组合能应用于所胡 HTML5 浏览器中。

在近期也不太可能改变。

为了跨平台和设备观看你的视频，你可能需要不止一次进行视频编码。

为了获得最大的兼容性，处理视频的流程大概会是这样：

1. 制作一个 Ogg 容器中使用 Theora 视频和 Vorbis 音频的版本。
2. 制作另外一个版本，使用 WebM 视频容器（VP8＋Vorbis）。
3. 再制作一个版本，使用 MP4 视频容器，并使用 H.264 基本配置的视频和 AAC 低配的音频。
4. 链接上面 3 个文件到同一个<video>元素，并向后兼容基于 Flash 的视频播放器。

5.6 H.264 视频的授权问题

Licensing Issues with H.264 Video

在这之前，我必须指出，一个视频进行两次编码是有一定成本的。也就是说，除了明显的成本外，两次编码一个视频，会导致需要更多的计算机和时间。然而，还有一个更现实的成本，那就是 H.264 的版权费用。

还记得我第一次解释 H.264 视频时（参见第 84 页“H.264”一节）就提到过它是受专利保护的并由 MPEG LA 组织负责授权的吗？这很关键。要理解这为什么很关键，我给你看看 H.264 授权迷宫[1]：

> MPEG LA 组织将 H.264 授权分拆为两部分：一部分是对于编码器和解码器的生产商，一部分是对于内容的发行商。[...]
>
> 对于内容发行商的部分又被分拆为 4 个部分，其中两个部分（订阅和按集收费）与最终用户视频服务付费相关，另两个部分（“免费”电视和国际广播）与视频音频制作人而非最终观众相关。[...]
>
> “免费”电视授权费用是两种版权费用中的一种。第一种是对于每个 AVC 传输编码器一次性付款 2 500 美元，它可以把已经授权的 AVC 视频传输至最终用户，然后用户再负责解码。那是否会双向收费呢？答案是肯定的。编码器制造商和传播者都应该付相应的费用。
>
> 第二种是按年交的传播费。[...]按每年的收视率大小收费：
>
> - 100 000–499 999 个家庭电视为 2 500 美元/年
> - 500 000–999 999 个家庭电视为 5 000 美元/年
> - 1 000 000 或者更多个家庭电视为 10 000 美元/年

[1]注：*http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-H.264-Licensing-Labyrinth-65403.aspx*.

[...]既然围绕着“免费”电视有那么多问题，为什么会有人涉足广播之外的传播方式呢？像我前面提到的，加盟费适用于所有形式的内容传播。在把“免费”电视赋予更多的内涵之后，MPEG LA 接着把因特网广播的加盟费定义为“通过互联网把内容传播到终端用户的 AVC 视频而终端用户不需为此付费”换句话说，任何公共广播、无论是无线、有线、卫星还是互联网，都是收费对象[...]

对互联网传播收费可能会过于昂贵，因为网上传输的速度要远远比通过有线电视或卫星传输的速度快得多。把“免费电视”市场传播费用加上额外费用之外，MPEG LA 授权暂缓许可期限第一阶段之外的费用，该阶段在 2010 年 12 月 31 日终止；同时，他们指出“第一阶段期限之后的版税不会比同样时间内的免费电视的可支付版税的经济等价物更高。”

最后，国际传播的费用结构已经被修正。MPEG LA 组织最近宣称互联网视频免费，直至 2015 年 12 月 31 日。那之后呢？谁知道？

5.7 使用 Firefogg 编码 Ogg 视频

Encoding Ogg Video with Firefogg

（本节中，我们将使用“Ogg 视频”作为“Ogg 容器中 Theora 视频和 Vorbis 音频”的简写。这种编解码器和容器的组合是 Firefox 和 Chrome 原生支持的。）

Firefogg 是一个开放源码，GPL 许可的 Firefox 扩展，用于编码 Ogg 视频。要使用它，需要安装 Mozilla Firefox 3.5 或更高版本，然后访问 Firefogg 网站，如图 5.1 “Firefogg 主页”。

点击“Install Firefogg”，Firefox 将提示你是否真的允许该网站安装扩展。点击“Allow”继续（图 5.2，“允许 Firefogg 安装”）。

Firefox 将显示标准软件安装窗口。点击“Install Now（现在安装）”继续（图 5.3，“安装 Firefogg”）。

点击“Restart Firefox”，完成安装（如图 5.4，重新启动“Firefox”）。

一旦重新启动完 Firefox，Firefogg 网站将提示 Firefogg 安装成功（图 5.5，“安装成功”）。

点击“Make Ogg Video”开始编码过程（图 5.6，“让我们做一些视频！”），然后点击“Select file”选择源视频（图 5.7，“选择视频文件”）。

图 5.1　Firefogg 主页

图 5.2　允许 Firefogg 安装

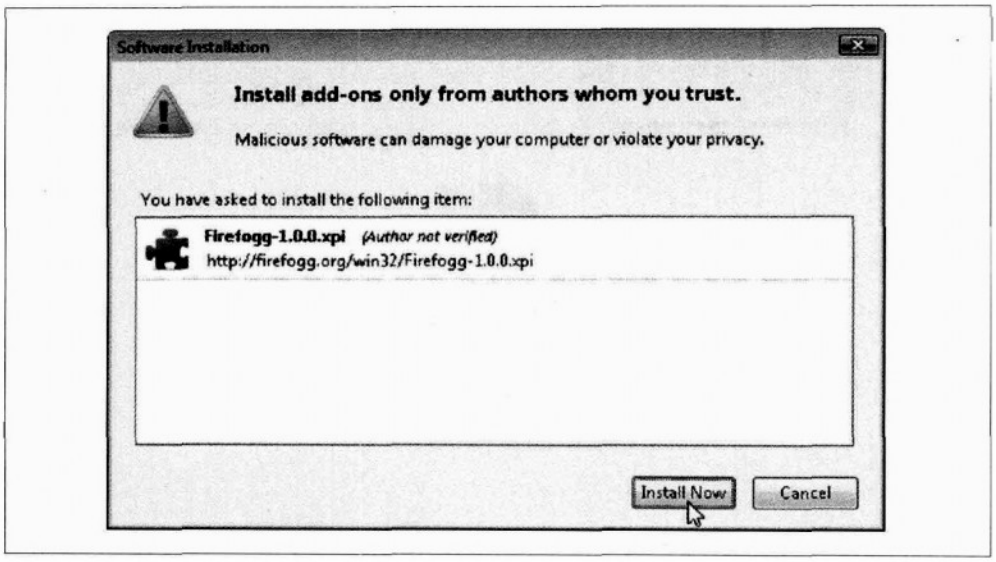


图 5.3 安装 Firefogg

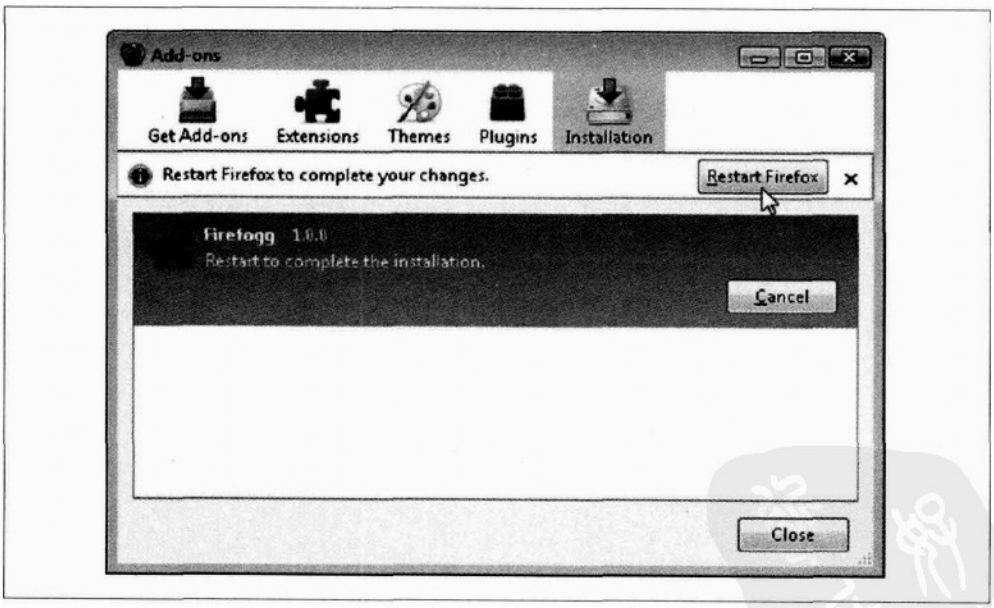


图 5.4 重新启动 Firefox

图 5.5　安装成功

图 5.6　让我们制作一些视频！

图 5.7　选择视频文件

Firefogg 主界面有六个“标签”（如图 5.8，“让我们对视频编码”）：

预设（Presets）

默认的设置是“网络视频”，符合我们的需求。

编码范围（Encoding range）

视频编码可能需要较长的时间。刚开始时，你可能只希望对一小段视频进行编码（比如，最开始的 30 秒钟），直到找到了一组你想要的设置。

基本质量和分辨率控制（Basic quality and resolution control）

这里包含了大部分重要的选项。

元数据（Metadata）

我不会在这里介绍，不过可以往编码的视频中加入元数据，比如标题，作者名等。也可能使用 iTunes 或其他音乐管理器添加元数据到你的收藏夹。这是一回事。

高级视频编码控制（Advanced video encoding controls）

不要改变这些设置，除非你明白你在做什么。（Firefogg 提供了有关这些选项的互动性的帮助。点击每个选项旁边的“i”符号可以看到它的相关帮助。）

高级音频编码控制（Advanced audio encoding controls）

再次强调，不要改变这些设置，除非你明白你在做什么。

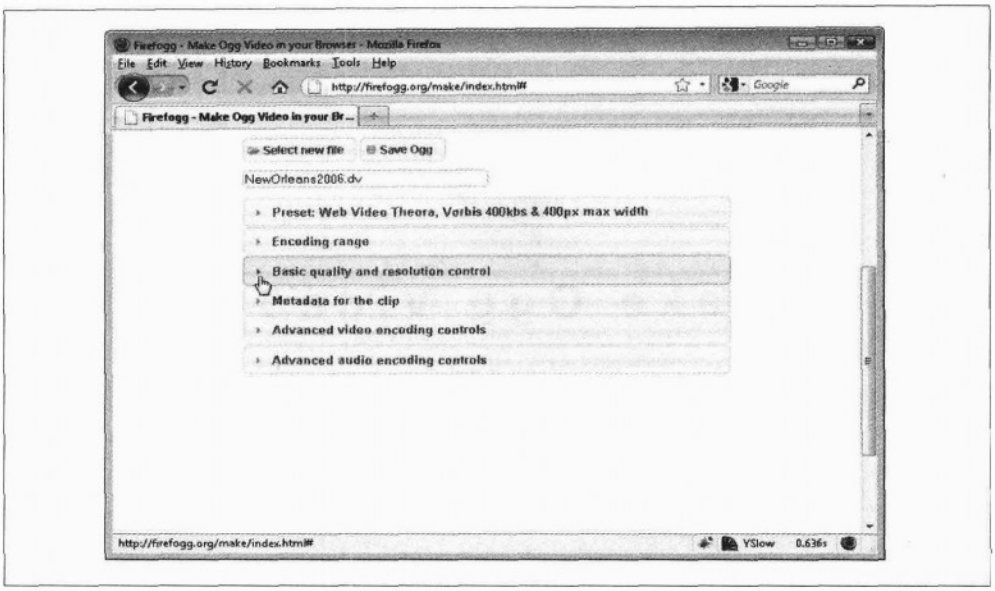


图 5.8　让我们对视频编码

我们只是要看看“基本质量和分辨率控制”选项卡（图 5.9，“基本质量和分辨率控制”），就行了，其中包含了所有重要的选项：

视频质量（Video Quality）

评估范围从 0（最低质量）到 10（最高质量）。数字越大，意味着文件越大，所以你需要做些实验，以确定满足需求的最佳大小/质量比。

音频质量（Audio Quality）

评估范围从–1（最低质量）到 10（最高质量）。数字越大，意味着文件越大，就像视频质量一样。

视频编解码器（Video Codec）

应该永远都设置为“theora”。

音频编解码器（Audio Codec）

应该永远都设置为“vorbis”。

视频的宽度和高度（Video Width and Video Height）

默认为原始视频的高度和宽度。如果需要在视频编码过程中调整视频的高度和宽度，可以改变这里的参数。Firefogg 会自动调整其他方面以保持原有的比例，因此视频不会走样。

如图 5.10“调整视频的宽度和高度”，我调整了该视频的宽度为原始宽度的一半。请注意 Firefogg 是如何自动调整视频高度以保证原宽高比的。

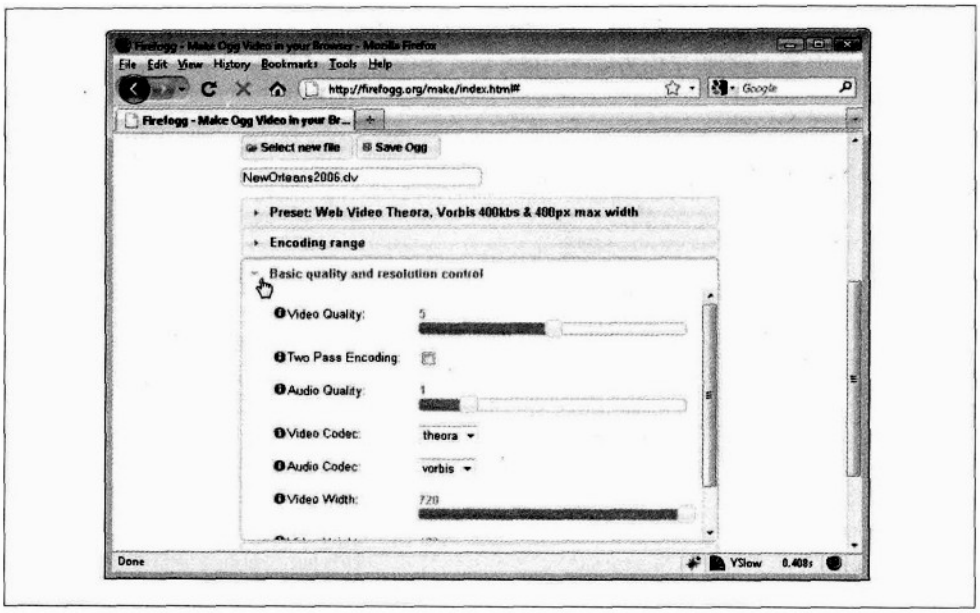


图 5.9　基本质量和分辨率控制

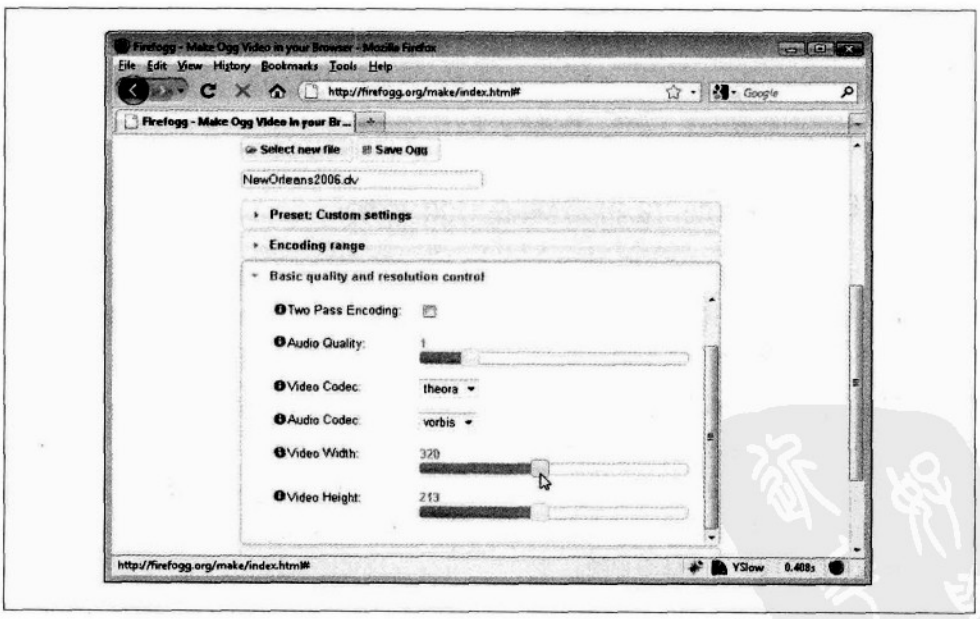


图 5.10　调整视频的宽度和高度

一旦设置完了所有选项，就可以点击“Save Ogg”，开始编码（图 5.11 点击“Save Ogg”，开始编码）。Firefogg 会提示你为编码的视频输入文件名。

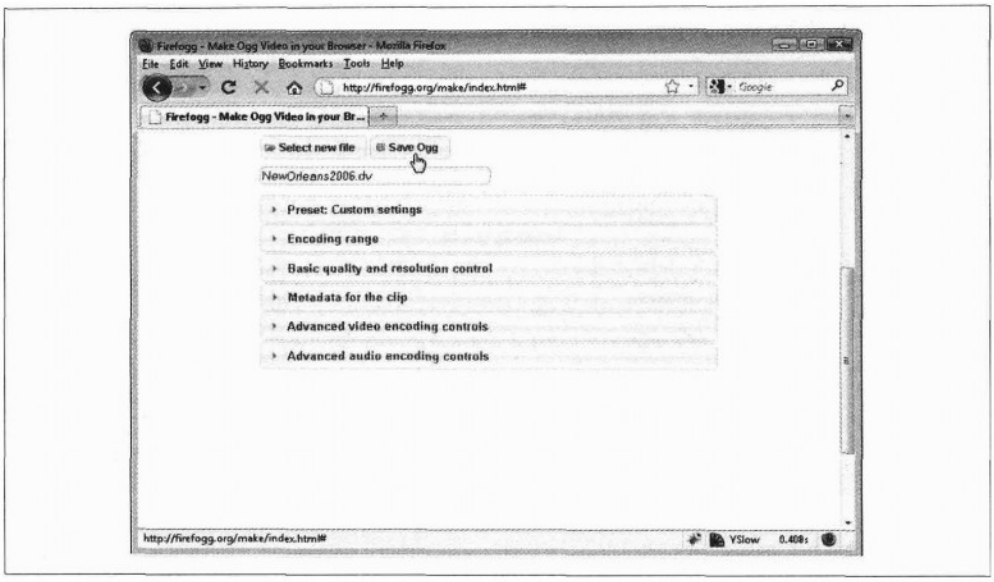


图 5.11　点击“Save Ogg”后开始了编码

在视频编码过程中，Firefogg 将显示一个漂亮的进度条（图 5.12，“编码进行中”）。接下来，你要做的就是等待（等待，再等待）！

5.8　使用 ffmpegtheora 批量编码 Ogg 视频

Batch Encoding Ogg Video with ffmpeg2theora

（和上节一样，我们将使用“Ogg 视频”作为“Ogg 容器中 Theora 视频和 Vorbis 音频”的简写。这种编解码器和容器的组合是 Firefox 和 Chrome 原生支持的。）

有很多离线 Ogg 视频编码器。但如果你希望批量编码多个视频文件并且全自动化，一定要看看 ffmpeg2theora（*http://v2v.cc/~j/ffmpeg2theora/*）。

ffmpeg2theora 是开放源码，基于 GPL 许可的 Ogg 视频编码应用程序。也可用于 Mac OS X，Windows 和 Linux 发行版的预编译二进制文件可以下载。几乎任何视频文件都可以作为输入，包括许多消费级 DV 拍摄的视频。

ffmpeg2theora 只能在命令行终端使用。（Windows 下，点击开始菜单→程序→附件→命令行提示符。Mac OS X 下，打开应用程序→实用工具→终端。）

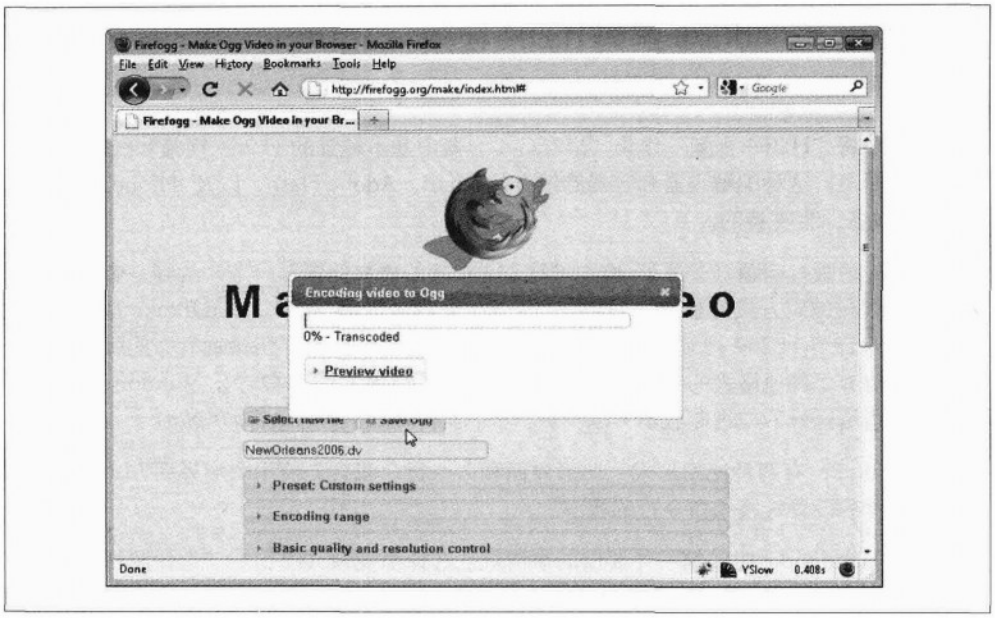


图 5.12　编码进行中

ffmpeg2theora 有很多命令行选项。（输入 ffmpeg2theora --help 可以获取所有标识说明。）我将重点介绍其中三个：

- --video-quality Q，其中 Q 取值从 0–10。
- --audio-quality Q，其中 Q 取值从–2–10。
- --max_size= W x H，其中 W 和 H 代表最大宽度和高度。（之间的 x 实际上就是字母“x”）。ffmpeg2theora 将调整视频比例以符合尺寸，从而编码后的视频可能比 W×H 小。例如，编码一个 720×480 像素的视频传递--max_size 320×240 选项，将产生一个 320 像素宽，213 像素高的视频。

因此，这里把选项设置成与前一节（参见第 91 页“使用 Firefogg 编码 Ogg 视频”一节）一样：

```
you@localhost$ ffmpeg2theora --videoquality 5
                             --audioquality 1
                             --max_size 320x240
                             pr6.dv
```

编码后的视频将被保存在与原始视频相同目录下，并以.ogv 为扩展名。你可以通过传递一个--output= /path/to/encoded/video 命令行选项给 ffmpeg2theora 指定不同的位置/或文件名。

5.9 使用 HandBrake 编码 H.264 视频

Encoding H.264 Video with HandBrake

（本节将“H.264 视频”作为“MPEG-4 容器中基本配置的 H.264 视频和低配的 AAC 音频”的简写。这种编解码器和容器的组合是 Safari、Adobe Flash，以及 iPhone 和 Google Android 设备原生支持的。）

我们先把版权问题（参见第 90 页“H.264 视频的授权问题”一节）放到一旁，最简单的 H.264 视频编码方式就是使用 HandBrake（*http//handbrake.fy*）。HandBrake 是一个开放源码，GPL 许可 H.264 视频应用程序（曾也支持很多视频格式，但最近发布的版本中，开发者已经放弃了其他格式的支持，专注于 H.264）。有用于 Mac OS X，Windows 和 Linux 发行版的预编译好的二进制版本可以下载（*http//handbrake.fy/downloads.php*）。

HandBrake 有两种使用形式：图形界面和命令行。我会先使用图形界面方式，然后再把一些推荐的设置转换为命令行的版本。

打开 HandBrake 时，第一件要做的事情就是选择源视频（图 5.13“选择源视频”）。点击“Source”，然后从下拉菜单中选择“Video File”，并选择一个文件。几乎任何视频文件都可以作为输入，包括许多消费级 DV 拍摄的视频。

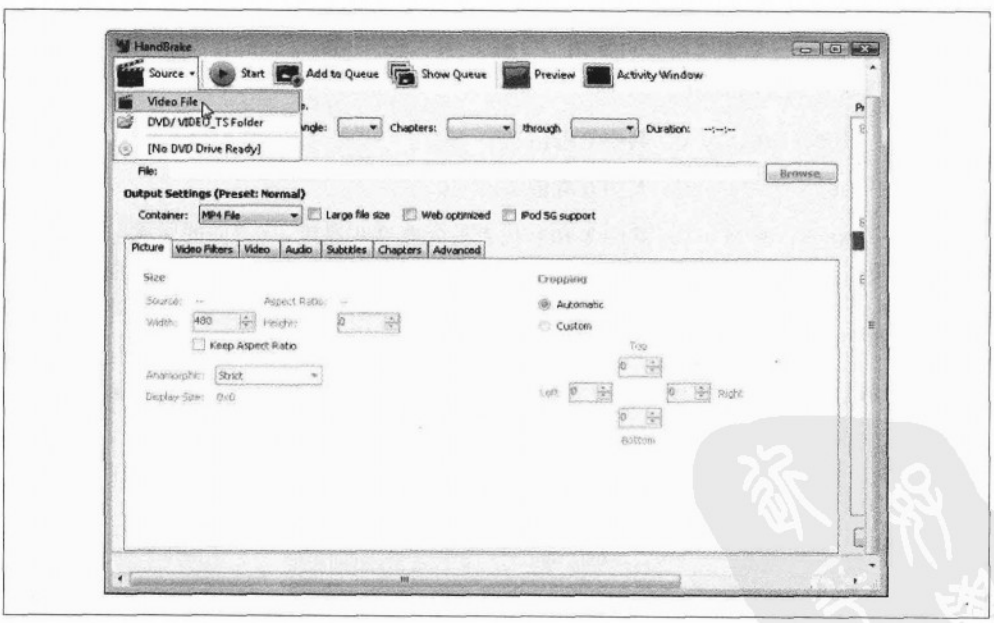


图 5.13 选择源视频

HandBrake 会弹出警告说你还没有设定一个默认目录来保存编码的视频（图 5.14，“忽略警告”）。可以放心地忽略此警告，或者可以打开选项窗口（在“Tool”菜单）并设置一个默认的输出目录。

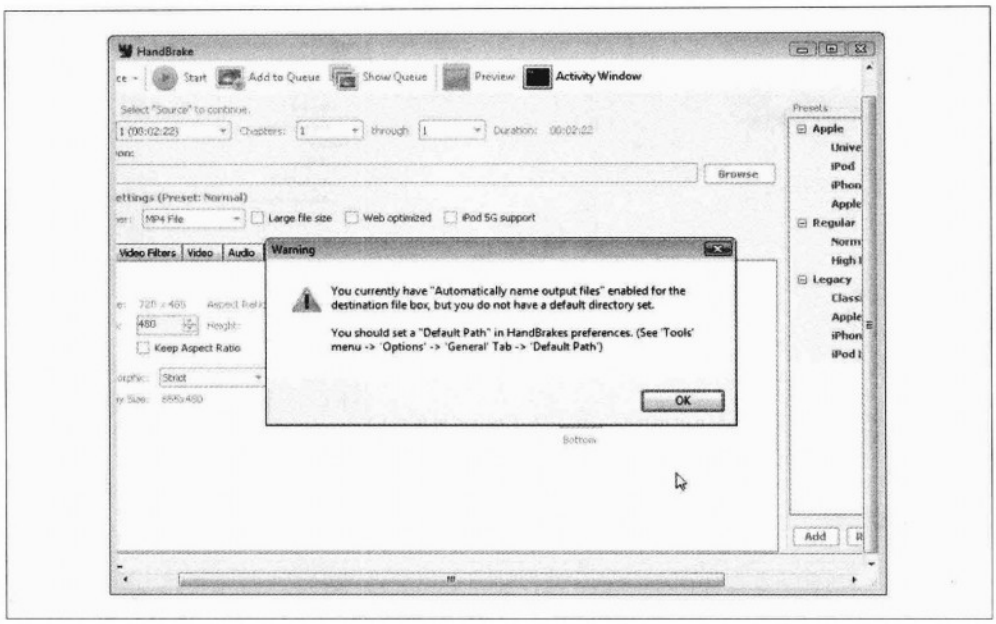


图 5.14　忽略警告

在右手边是预置配置一览表。选择“iPhone & iPod Touch”，如在图 5.15“选择 iPhone 预设”，将大多数需要的选项设置完成。

“Web optimized”这个选项默认是关闭的。勾选如图 5.16，“始终 Web 最佳化”，记录一些编码后视频的元数据可以实现边下载边播放。我强烈建议总是勾选此选项。它不影响编码后视频的质量或文件的大小，因此实在没有理由不这样做。

在“Picture”标签，可以设置编码后视频的最大宽度和高度（图 5.17，“设定宽度和高度”）。也应该选择“保持纵横比”选项，以确保在调整大小的时候 HandBrake 不会扭曲视频。

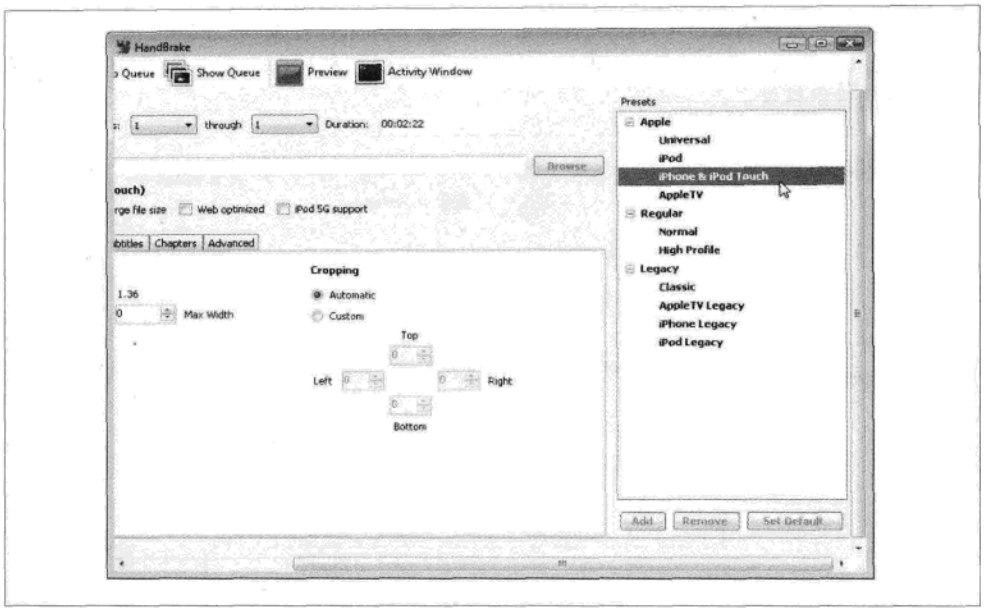


图 5.15　选择 iPhone 预置

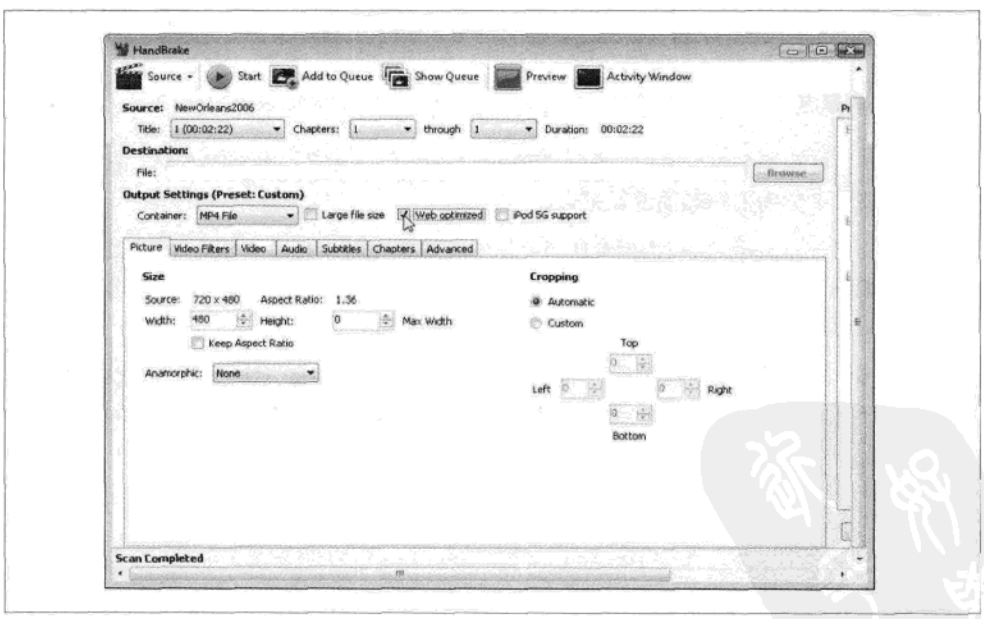


图 5.16　始终为网页优化

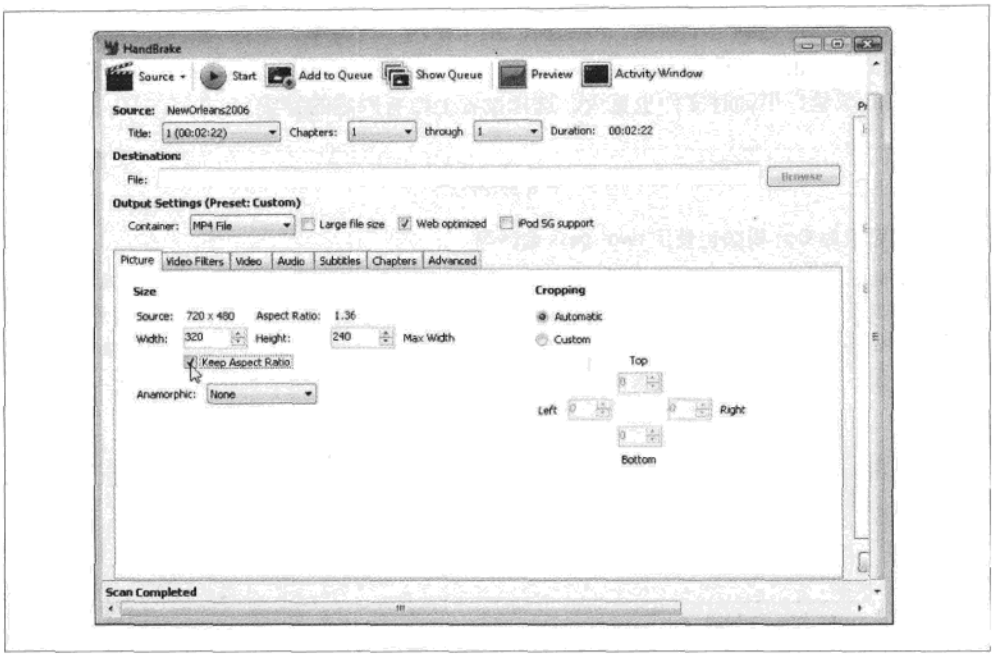


图 5.17　设置宽度和高度

如图 5.18“Video”标签所示，你可以设置几个重要的选项：

视频编解码器（Video Codec）

确保这是“H.264（x264）”。

2-pass 编码（2-Pass Encoding）

如果被选中，HandBrake 将运行视频编码器两次。第一次，它只分析视频，搜寻一些信息，如色彩构成、运动及停顿的画面。第二次，它会使用之前搜集的信息对视频进行编码。正如你所想的那样，这会花去两倍长的编码时间，但它可以在不增加文件大小的情况下提高视频的质量。我总是启用 H.264 视频两次编码。除非你正在构建下一个 YouTube，或者一天 24 小时地编码视频，否则你都应该使用 two-pass 编码。

加速首次分析（Turbo First Pass）

如果你启用了 two-pass 编码，其实还是可以稍微节约一点时间的。只要减少第一次分析信息的部分工作，这只会稍微影响视频质量。我通常会启用此选项，但是如果质量最重要，应该禁用它。

质量（Quality）

有几种不同的方法来指定编码后视频的“质量”。你可以设置目标文件大小，HandBrake 会尽力让编码后的视频文件大小小于这个值。也可以设置平均“比特率”，HandBrake

会一点不差地达到这个数值。（之所以用“平均”比特率，是因为一些时间段需要更多的位。）也可以指定一个恒定的质量，取值在0%到100%之间。更高的取值会得到更好的质量，但同时文件也更大。选择设置上没有固定的答案。

专家答疑

问：我可以对Ogg视频也使用two-pass编码吗？

答：是的，但由于编码器的实现不一样，你可能没有这个必要。Two-Pass H.264编码几乎可以产生最高质量的视频。编码后的Ogg视频只在想有一个特定的文件大小时才适用。（也许这是你感兴趣的，但我不打算举例说明，对网页视频意义也不大。）为了获得最佳的Ogg视频质量，可以使用视频质量设置，而且不用担心two-pass编码。

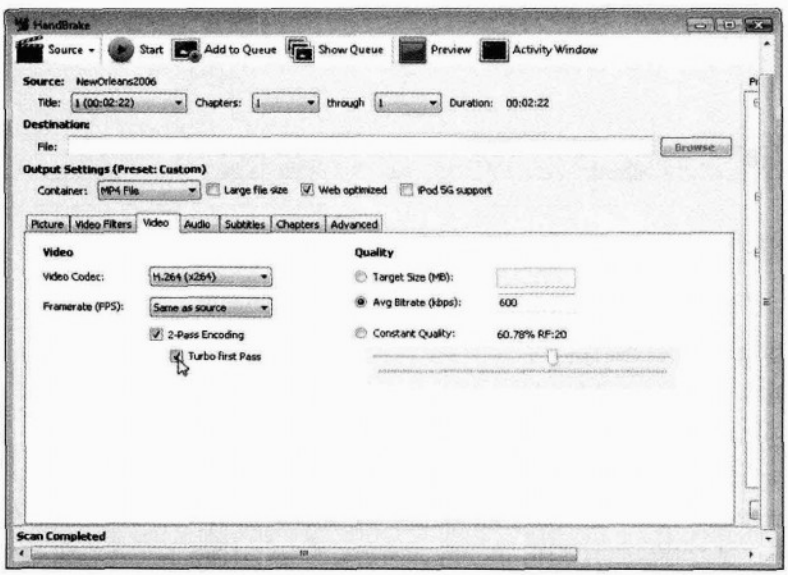


图5.18 视频质量选项

在这个例子中，我选择了 600 kbps 的平均比特率，相当于一个 320 × 240 的编码后的视频。也启用 Turbo First Pass。

在“Audio”选项卡中，如图 5.19 所示的“音频质量选项”，你可能不需要进行任何更改。如果源视频有多个音轨，可能需要选择哪一个是要引入编码后的视频的。如果视频是个人谈话（相对于音乐或者环绕声），可以使用比较低的比特率，如 96kbps 左右。除此之外，从“iPhone”预设里得到的配置应该相当合理。

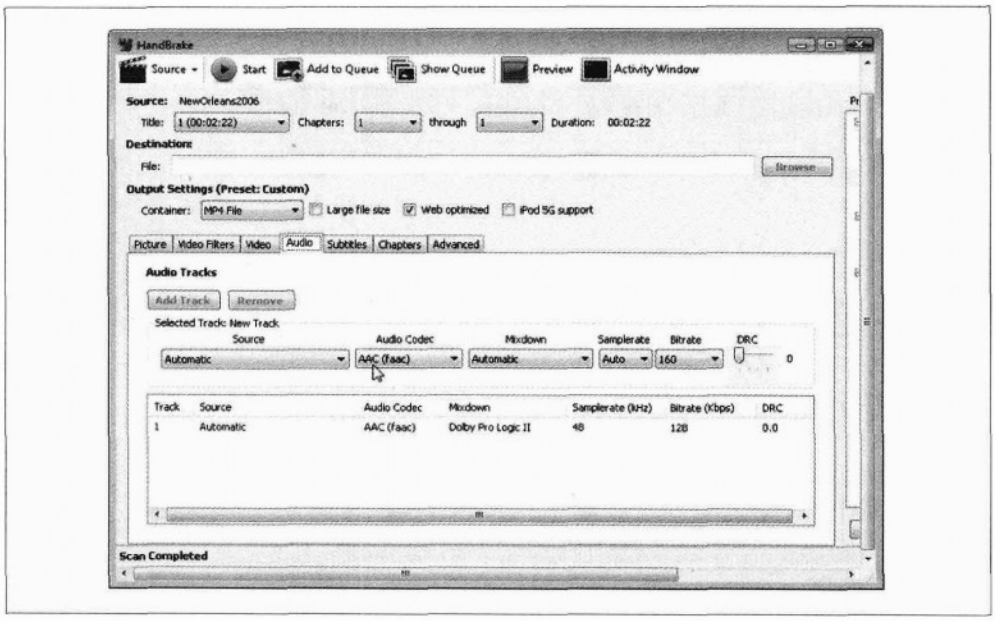


图 5.19　音频质量选项

接下来，点击“Browse”按钮，选择目录和文件名保存编码的视频（图 5.20，“设定目标文件名”）。

最后，单击“Start”开始编码（图 5.21，“让我们制作一些视频！”）。

在视频编码过程中，HandBrake 将显示一些进展中的统计数据（图 5.22，“请耐心等待”）。

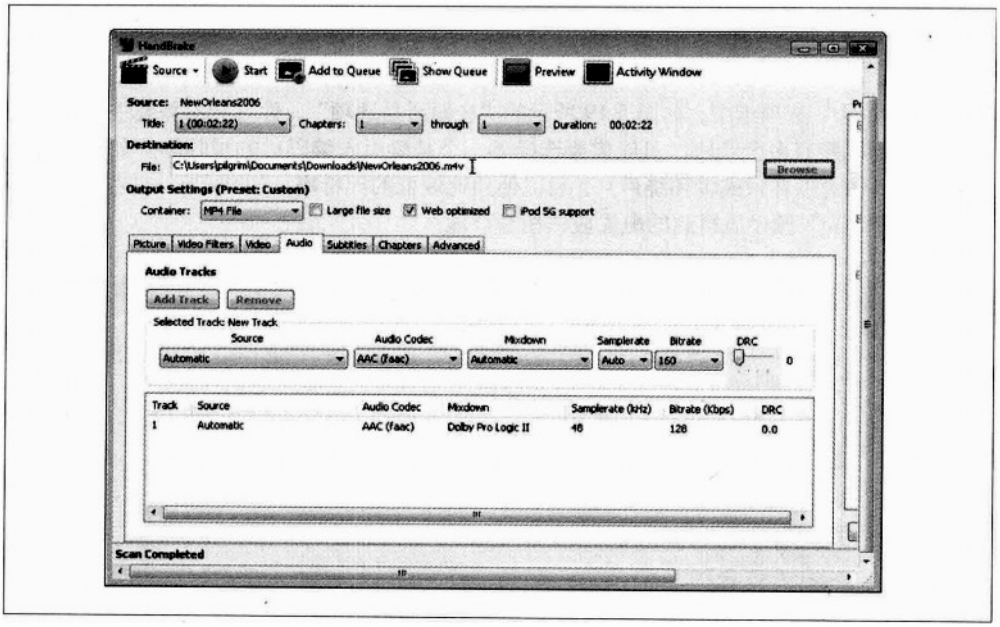


图 5.20　设定目标文件名

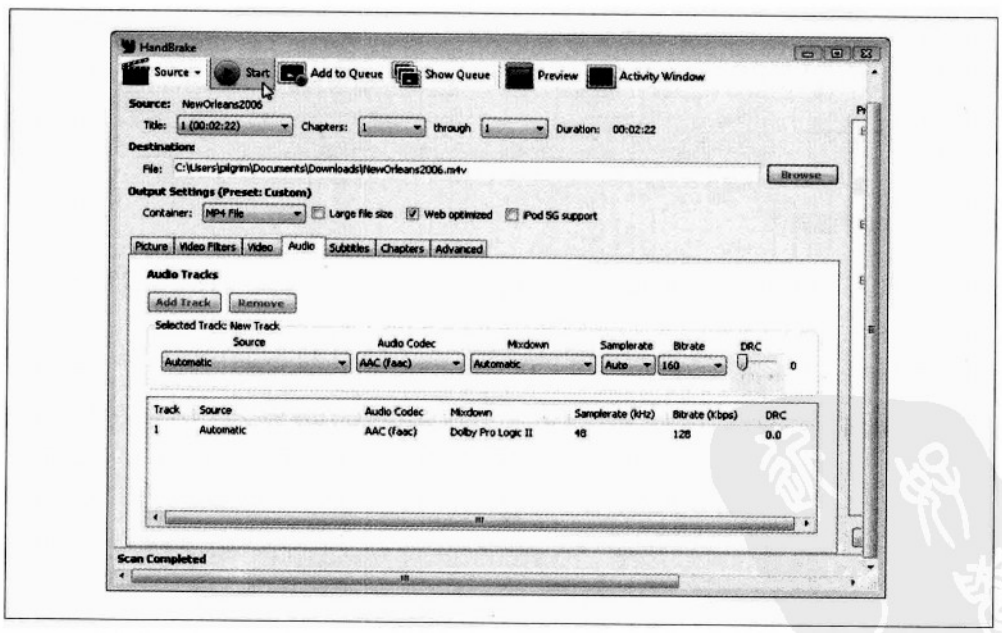


图 5.21　让我们制作一些视频！

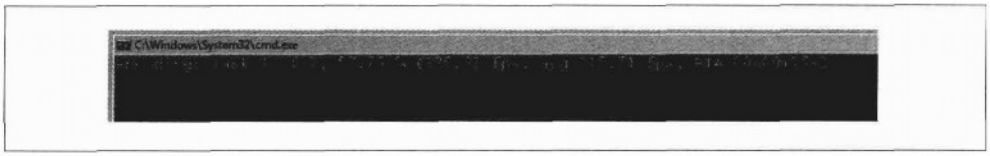


图 5.22　请耐心等待

5.10　使用 HandBrake 批量编码 H.264 视频
Batch Encoding H.264 Video with HandBrake

（和上节一样，我们使用“H.264 视频”作为“MPEG-4 容器中基本配置的 H.264 视频和低配置的 AAC 音频”的简写。这种编解码器和容器的组合是 Safari、Adobe Flash，以及 iPhone 和 Google Android 设备原生支持的。）

正如我在前面一节中提到的，HandBrake 也有命令行的版本。如同图形版一样，需要从 *http://handbrake.fr/downloads2.php* 下载一个近期的快照。

和 ffmpeg2theora（参见第 98 页“使用 ffmpegtheora 批量编码 Ogg 视频”一节）相似，命令行版的 HandBrake 提供了大量的选项。（在命令行中输入 `HandBrakeCLI --help` 可以得到帮助信息）我将仅介绍其中几个：

- `--preset " X "`，其中" X "是 HandBrake 预设配置的名字（引号很重要）网页中的 H.264 视频的预设配置名为“iPhone & iPod Touch”。
- `--width W`，其中 W 是编码后视频的宽度。HandBrake 会自动调整高度，以保持原始视频的比例。
- `--vb Q`，其中 Q 是平均比特率（单位为千比特每秒）。
- `--two-pass`，使用 two-pass 编码。
- `--turbo`，在使用 two-pass 编码时使用 turbo first pass。
- `--input F`，其中 F 是源视频的文件名。
- `--output E`，其中 E 是编码后的视频的文件名。

下面是使用 HandBrake 命令行的例子，传入了一些参数，使其效果和图形界面的版本一样：

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
                            --width 320
                            --vb 600
                            --two-pass
                            --turbo
                            --input pr6.dv
                            --output pr6.mp4
```

从上到下的参数为："iPhone & iPod Touch"预设配置，调整视频大小为320x240，设置平均比特率为600kbps，设置two－pass编码，并同时打开turbo选项，输入文件为pr6.dv，输出文件为pr6.mp4。嘿！

5.11 使用ffmpeg编码WebM视频

Encoding WebM Video with ffmpeg

我写这段话的时间是2010年5月20日。WebM格式昨天刚好发布。因此还没有多少编码工具可供选择，文档也相当少。但随着新的工具被编写（以及现有工具升级）能够支持WebM。这种情况将会好转。目前来说，需要下面一些工具：

- libvp8，一个打了很多补丁的ffmpeg（补丁使其可以与libvp8通信），但目前没有合入ffmpeg的官方源：

  —对于Linux平台的说明（*http://lardbucket.org/blog/archives/2010/05/19/vp8-webm-ffmpeg-windows/*），我在Ubuntu 10.04"Lucid"中测试了。

  —对于Windows平台的说明（*http://www.ioncannon.net/meta/1128/compiling-webm-ffmpeg-windows/*），我没有亲自测试过。

- 最新的版本为mkclean（*http://www.matroska.org/downloads/mkclean.html*）。

准备好了吗？OK。在命令行，不带任何参数运行ffmpeg，可以检查是否支持VP8：

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg developers
  built on May 19 2010 22:32:20 with gcc 4.4.3
  configuration: --enable-gpl --enable-version3 --enable-nonfree --enable-postproc
  --enable-pthreads --enable-libfaac --enable-libfaad --enable-libmp3lame
  --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora
  --enable-libx264 --enable-libxvid --enable-x11grab --enable-libvpx-vp8
```

如果没有看到咒语--enable-libvpx-vp8，就说明还没有使用支持VP8的ffmpeg特定版本。（如果你确信编译没有问题，那就检查一下，看是否安装了两个版本的ffmpeg。没关系，它们不会互相冲突，你只要全路径调用一个支持VP8的版本就行了。）

我将展示如何进行two-pass编码（见100页"使用HandBrake编码H.264视频"一节）。第一遍扫描输入文件（-i pr6.dv），然后把分析的统计信息写入日志文件（自动命名为pr6.dv-0.log文件）。你可以通过-vcodec来指定视频编解码器：

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8 -an
               -f rawvideo -y NUL
```

ffmpeg大部分的命令行参数并不支持VP8或者WebM。libvp8支持一些特别的VP8选项，可以把这些传递给ffmpeg，但目前我还不知道怎么让它们工作。一旦我找到了这些选

项不错的解释文档，我将在本书的配套网站中加入一个链接。

第二遍，ffmpeg 会读取统计信息并编码视频合音频。结果产生一个 MKV 文件，之后你需要把它再转换为 WebM 文件。（将来 ffmpeg 肯定可以直接生成 WebM 文件，不过目前还做不到。）这是第二遍时的参数：

```
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
               -b 614400 -s 320x240 -aspect 4:3 -acodec vorbis -y pr6.mkv
```

这里有五个重要的参数：

-vcodec libvpx_vp8

指定我们使用的是 VP8 视频编解码器。WebM 总是使用 VP8 编码的视频。

-b 614400

指定比特率。不像其他的格式，libvp8 比特率的是按 bit 计的，而不是 kb。如果你想要一个 600 kbps 的视频，用 600 乘以 1024 得到 614400。

-s 320x240

指定目标大小，高度乘宽度。

-aspect 4:3

指定视频的宽高比。标准分辨率的视频通常是 4:3，但大多数高分辨率的视频是 16:9 或 16:10。我在测试过程中发现必须手动设置而不能依靠 ffmpeg 自动检查。

-acodec vorbis

指定使用 vorbis 音频编解码器。WebM 始终使用 Vorbis 音频。

现在我们有一个有 VP8 视频和 Vorbis 音频的 MKV 文件。这已经非常接近我们的需求。从技术上说，WebM 容器格式与 MKV 相当接近。事实上，WebM 是其子集。我们只需要使用前面提到过的 mkclean 来调整一些位信息以创建最终的 WebM 视频文件。

```
you@localhost$ mkclean --doctype 4 pr6.mkv pr6.webm
```

只需写这样短短一句！

5.12 最后，标记
At Last, the Markup

我确信这是一本关于 HTML 的书。但怎么没有讲 HTML 标记呢？

HTML5 中有两种方法让你在网页中引入视频。这些都涉及到了<video> 元素。如果只有一个视频文件，可以简单地通过 src 属性链接它。这与使用<img src="...">标签几乎一样。

```
<video src="pr6.webm"></video>
```

从技术上讲，这就是你所有需要做的。可是就像<img>标签，<video>标签也应该包括 width 和 height 属性。高和宽可以是你编码时指定的最大值。

```
<video src="pr6.webm" width="320" height="240"></video>
```

不要担心实际的视频大小与其不符。浏览器会让<video>标签内的视频居中。同时保持视频比例，而不会让视频走样。

<video>元素默认不会提供任何播放控制。可以使用平常的 HTML、CSS 和 JavaScript 创建独特的控制器。<video>元素拥有 play()和 pause()方法和一个可读写的 currentTime 属性。当然还有可读写的 volume 和 muted 属性。所以，你拥有了创建满足所需界面的一切。

如果不想自己创建界面，可以使用浏览器内置的控制界面。只需要在<video>标签内加上 controls 属性：

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

在进一步学习之前，我还想介绍两个属性：preload 和 autoplay。耐心一点，让我来解释为什么这两个属性很有用。preload 属性告诉浏览器当页面加载时就开始下载视频。如果整个页面就是为了播放视频，这就很有意义。然而如果页面的访问量很少，就可以设置 preload 属性为 none，以节约带宽。

下面是加载时就下载视频的例子（没有自动播放）：

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

而下面这个例子，当页面加载时，并不会去下载对应视频：

```
<video src="pr6.webm" width="320" height="240" preload="none"></video>
```

autoplay 属性的作用和其名字一样：告知浏览器，只要页面加载就开始下载视频，且一旦可行就立马播放。有人喜欢这样，有人讨厌它。但请让我来解释为什么 HTML5 中一定要有这样一个属性。一些人会愿意自动播放他们的视频，尽管一些访客并不愿意如此。如果 HTML5 标准没有定义这样的方法，就会有人使用 JavaScript 来模拟自动播放。比如，通过在 window 的 load 事件处理函数里调用视频的 play()方法。这会让访客更加头痛。从另一方面说，你可以轻而易举地添加一个浏览器的插件（如果有必要，写一个也无妨）来“忽略 autoplay 属性”，我就从来都不喜欢自动播放网页中的视频。

下面是一个页面加载完，就开始边下载边播放的例子。

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

这里有一个 Greasemonkey 的脚本（*http://www.greasespot.net*），可以让你的 Firefox 浏览器阻止 HTML5 的视频被自动播放。它使用 HTML5 定义的 DOM autoplay 属性，这是 HTML 标记中 autoplay 属性的对等物。

```
// ==UserScript==
// @name           Disable video autoplay
// @namespace      http://diveintomark.org/projects/greasemonkey/
// @description    Ensures that HTML5 video elements do not autoplay
// @include        *
// ==/UserScript==

var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
    var elmVideo = arVideos[i];
    elmVideo.autoplay = false;
}
```

打住。如果你认真学习了整个章节，就会知道，你不只链接了一个视频文件，往往是 3 个。第一个是使用 Firefogg（参见第 91 页“使用 Firefogg 编码 Ogg 视频”一节）或者 ffmpeg2theora（参见第 98 页“使用 ffmpeg2theora 批量编码 Ogg 视频”一节）创建的.ogv 文件。第二个是由 HandBrake（参见第 100 页“使用 HandBrake 编码 H.264 视频”一节）生成的.mp4 文件。第三个是使用 ffmpeg（参见第 108 页“使用 ffmpeg 编码 WebM 视频”一节）生成的.webm 文件。HTML5 提供了一种链接这 3 个视频文件的方法，那就是使用<source>元素。每个<video>可以包含任意数目的<source>元素。浏览器会按照顺序下载对应视频文件，并播放最先能播放的那个文件。

这又产生了一个新的问题：浏览器怎么判断自己能否播放某种文件？最坏的情况是，

加载每个视频文件并尝试播放它们。这会浪费大量的带宽。如果可以告诉浏览器每部视频的容器类型，那就可以节约大量带宽了。可以通过<source>元素的 type 属性来指定。

下面是一个完整的例子：

```
<video width="320" height="240" controls>
  <source src="pr6.mp4"  type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="pr6.ogv"  type='video/ogg; codecs="theora, vorbis"'>
</video>
```

我们一起来分析一下。<video>元素指定了视频的宽度和高度，但它实际上并没有链接到任何一个视频文件。在<video>内部有三个<source>元素。每个<source>元素（通过 src 属性）链接到一个独立的视频文件，并（通过 type 属性）给出了相应的格式信息。

type 属性看起来很复杂，像地狱一般。它包括三部分信息：容器格式（参见第 81 页“视频容器”一节），视频编解码器（参见第 83 页“视频编解码器”一节），音频编解码器（参见第 85 页“音频编解码器”一节）。让我们从最下面的<source>开始讲解。对于.ogv 视频文件，容器格式为 Ogg 格式，这里用 video/ogg 代表（从技术上讲，这是 Ogg 视频文件的 MIME 类型）。视频编解码器为 Theora，音频编解码器为 Vorbis。这很简单吧，除了该属性的值的格式有点别扭。codecs 的值需要用引号围起来，所以对于 type 的值需要使用不同的引号来区别于 codecs 的值：

```
<source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"'>
```

对于 WebM 文件而言，其<source>元素也很类似，不同的是 MIME 类型（video/ogg 替换成 video/webm）和 codecs 参数里的视频编解码器（theora 替换成 vp8）：

```
<source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
```

对于 H.264 文件而言，<source>元素变得更加复杂。还记得我说过，无论是 H.264 视频（参见第 84 页“H.264”一节）还是 AAC 音频（参见第 87 页“高级音频编码”一节）都可以有不同的配置？我们这里使用低配的 AAC，而使用基本配置的 H.264，然后再一起打包进 MPEG-4 的视频容器。所有这些信息都包含在 type 属性里：

```
<source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

上面如此麻烦的指示，最大的好处就是浏览器可以检查 type 属性，直接得知是否可以播放此视频文件。如果浏览器确定无法播放此文件，就完全不用下载此文件。为你节省大量的带宽，访客也可以更快地看到视频。

如果你按照本章的参数编码视频，那直接从 type 属性里复制就行。否则，需要自己制定 type 属性。

专家发言

在撰写本文的时候（2010 年 5 月 20 日），iPad 有一个 bug，它只考虑 video 的视频列表中的第一个。这意味着你需要把 MP4 文件放置到第一位，然后才是几个免费的视频文件。哎。

5.12.1 MIME 类型很重要

这么多信息才拼出了一个视频，太麻烦了。但问题是，错误的配置会导致服务器不听使唤，而你在本地还觉得一切正常。如果遇到了这样的麻烦，那可能就是 MIME 类型惹的祸。

在第 1 章我提到过 MIME 类型（参见第 1 页“MIME 类型”一节），当时也许你还在发呆，没有认识到它的重要性。所以，这次算是复习了。

专家发言

视频文件必须使用正确的 MIME 类型！

什么才是正确的 MIME 类型呢？你已经看到，就是<source>元素里的 type 属性的值。但光设置 HTML 标签里的 type 属性是不够的。还应该确保服务器在 Content-Type 头信息中也有相应的设置。

如果使用 Apache 或者其衍生的服务器，可以在全站的 httpd.conf 配置文件或者存储视频文件的目录下的.htaccess 文件中使用 AddType directives 指令（如果使用的是其他服务器，请查看相关设置 Content-Type 头信息的文档来设置 MIME 类型）。AddType 指令这样使用：

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

第一行表示使用 Ogg 作为视频容器。下一行是 MPEG-4 容器。第三行是 WebM。设置一次，我们就再也不用管了。如果不这样做，视频将无法在浏览器里播放，即使你在 HTML 标签里写了 type 属性。

如果你想了解更多服务器配置的相关细节，建议你阅读 Mozilla/开发者中心/的这篇文章“Configuring servers for Ogg medio”（*http://developer.mozilla.org/en/Configuring_sercers_for_Ogg_media*）使用原书英文标题和链接。（文章里的建议对 MP4，WebM 都有用。）

5.13 IE 怎么办？
What About IE?

当我写这篇文章的时候，微软已经发布了 IE 9 的“开发者预览”版。它虽然还不支持 HTML5 的<video>元素，但微软已经承诺（*http://blogs.msdn.com/ie/orchive/2010/03/16/ htmls-hardwark-accelerated-first-ieg-playform-preview-available-for-developers.aspx*）IE 9 的最终版本会支持 MPEG-4 容器内的 H.264 视频和 AAC 音频，这和 Mac 上的 Safari 在 iOS 上的 Mobile Sofari 策略一样。

但对于老版本的 IE 怎么办呢？也就是对于 IE 8，以及以下的所有版本怎么应对呢？大多数用户的 IE 浏览器都安装了 Adobe Flash 插件。最近版本的 Adobe Flash（从 9.0.60.184 开始）支持 MPEG-4 容器内的 H.264 视频和 AAC 音频。一旦你为 safari 编码好了 H.264 视频文件（见 100 页“使用 HandBrake 编码 H.264 视频”一节），就可以通过基于 Flash 的播放器来兼容不支持 HTML5 的浏览器。

Flowplayer（*http://flowplayer.org*）就是一个开放源码，GPL 许可的，基于 Flash 的视频播放器。（商业许可证也可使用，详情请访问 *http://flowplayer.org/download/*。）FlowPlayer 对<video>元素完全不了解。它也无法神奇般地把一个<video>标签转换成 Flash 对象。但 HTML5 考虑到了这点，你可以在<video>元素内内嵌一个<object>元素。对于不支持 HTML5 视频的浏览器而言就会忽略掉<video>元素，从而只渲染<object>的部分，并调用 Flash 插件，通过 FlowPlayer 播放视频。而那些支持 HTML5 视频的浏览器，就直接播放而忽略内嵌的<object>元素。

最后由我来解开谜题：HTML5 规定，对于任何内嵌于<video>元素的标签，都将被忽略（<source>元素除外）。这就在新的浏览器使用 HTML5 视频的同时又兼容了老的浏览器，而不需要任何 JavaScript 的技巧。你可以好好看看这篇文章，把这个技术讲得很清楚（*http://camendesign.com/code/video-for-everybody*）。

5.14 完整的例子
A Complete Example

让我们一起看个例子，使用上了前面的技术，还扩展了使用原书标题链接这篇文章的代码，加入了 WebM 格式的视频。使用这些命令，我把同一部视频编码成了 3 种格式：

```
## Theora/Vorbis/Ogg
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240 --output pr6.ogv
               pr6.dv

## H.264/AAC/MP4
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch" --vb 200 --width 320
               --two-pass --turbo --optimize --input pr6.dv --output pr6.mp4
```

```
## VP8/Vorbis/WebM
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
               -an -f rawvideo -y NULffmpeg --videobitrate 200
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
               -b 204800 -s 320x240 -aspect 4:3 -acodec vorbis -ac 2 -y pr6.mkv
you@localhost$ mkclean --doctype 4 pr6.mkv pr6.webm
```

最好使用<video>元素，并内嵌<object>元素保持向后兼容：

```
<video id="movie" width="320" height="240" preload controls>
  <source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"' />
  <object width="320" height="240" type="application/x-shockwave-flash"
    data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-3.2.1.swf" />
    <param name="allowfullscreen" value="true" />
    <param name="flashvars" value='config={
     "clip": {"url": "http://wearehugh.com/dih5/good/bbb_480p.mp4",
     "autoPlay":false, "autoBuffering":true}}' />
    <p>Download video as <a href="pr6.mp4">MP4</a>,
     <a href="pr6.webm">WebM</a>, or
     <a href="pr6.ogv">Ogg</a>.</p>
  </object>
</video>
```

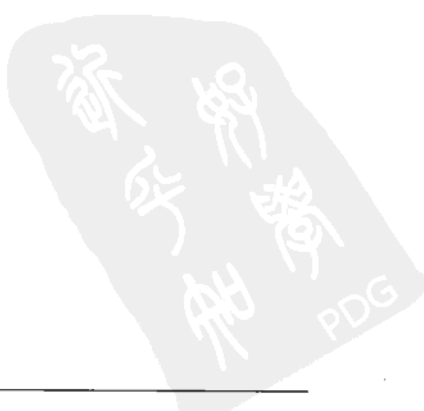
使用 HTML5 和 Flash 的组合，就可以在绝大多数浏览器和相关设备中观看视频了。

5.14 扩展阅读

Further Reading

- HTML5 规范中的<video>元素（*http://bit.ly/a3kpiq*）。
- 给所有人的视频！（*http://camendesign.com/code/cideo_for_everybody*）。
- 视频编码的一个不错的介绍（*http://diveintomark.org/tag/give*）。
- Christopher Blizzard 写的“关于 Theora 1.1 发布——你应该知道的事”（*http://hacks.mozilla.org/2009/09/theora-1-1-released/*）。
- “为 Ogg 媒体配置服务器”（*http://developer.mozilla.org/en/Configuring_servers_for_Ogg_media*）。
- “使用 x264 编解码器编码”（*http://www.mplayerhq.hu/DOCS/HTML/en/menc-feax-x264.html*）。
- “视频类型参数”（*http://wiki.whatwg.org/wiki/Video_type_paraneters*）。

- Zencoder Video js 的（*http://videojs.com*）定制 HTML5 视频控制器（*http://hideois.com*）。
- Simon Pieters 写的“关于 HTML5 音频和视频你需要知道的一切”（*http://dev.opera.com/articles/view/everything-you-need-to-fnow-obout-htm(5-video-end-audio/*）。



第 6 章

地理位置

You Are Here
(And So Is Everyboby Else)

6.1 引言

Diving In

地理位置特性能够识别出你所在的地理位置并且在你允许的情况下，把位置信息分享给别人。识别地理位置的方法有很多——通过 IP 地址、利用基站获取手机网络的接入位置、通过利用卫星定位获得经纬度信息的 GPS 设备。

专家答疑

问：地理位置特性听起来会泄露隐私，不是很安全，可以禁用吗？

答：提及把自己的位置信息共享出来，的确会有隐私泄露方面的担心。但是关于地理位置 API 的标准（*http://www.w3.org/TR/geolocation-API/#security*）明确说明了浏览器必须要在经过用户允许的情况下才可以发送用户位置信息给服务器。换句话说，要不要共享你的位置信息出来由你自己决定。

6.2 地理位置 API

The Geolocation API

调用地理位置 API 能够将用户地理位置信息发送给用户信任的站点。通过 JavaScript 可以获取用户所在地理位置的经纬度信息，随后可以将该信息发送给服务器去做一些诸如寻找周边的商家以及在地图上显示用户位置等等与位置相关的应用。

从表 6.1 中可以看到，一些 PC 以及移动设备上主流的浏览器都支持地理位置 API。另外，一些早期的浏览器也能通过第三方的库来支持地理位置 API，这部分会在本章后面作介绍。

表 6.1　浏览器对地理位置 API 支持情况一览

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	3.5+	5.0+	5.0+	.	3.0+	2.0+

除了对标准地理位置 API 的支持之外，针对其他一些移动平台还有许多特定的 API 用来获取地理位置信息。我会在后面一一介绍。

6.3　代码展示

. Show Me the Code

调用地理位置 API 从全局 navigator 对象的 geolocation 属性开始：navigator.geolocation。

最简单的使用方式如下：

```
function get_location() {
  navigator.geolocation.getCurrentPosition(show_map);
}
```

以上代码没有做任何的检测、容错处理，也没有选项参数。在实际应用中应该首先要进行检测和容错处理。可以使用 Modernizr 提供的方法来检测浏览器是否支持地理位置 API（参见第 24 页“地理位置”一节）：

```
function get_location() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(show_map);
  } else {
    // 浏览器没有提供原生支持，想获取可以试试 Gears ?
  }
}
```

如何处理浏览器不支持地理位置的情况由你决定。我等下会介绍一种叫 Gears 的备选方案，不过在此之前我想先讲讲在调用 getCurrentPosition()方法的时候究竟发生了什么。正如我在本章最开始提到的那样，对地理位置的支持本身就是可选的，这就意味着浏览器绝对不会在未经允许的情况下强制将用户当前地理位置信息发送回远程服务器，而是会有所提示，但是具体提示样式就因浏览器而异了。在 Mozilla Firefox 中，当调用 getCurrentPosition()方法的时候，浏览器窗口最上面会弹出一个“信息条”，如图 6-1 所示：

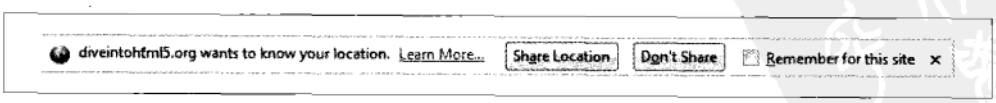


图 6-1　地理位置信息条

作为终端用户，此信息条告诉了你很多信息：

- 网站想要知道你的地理位置。
- 哪个网站想要知道你的地理位置。
- 可以点击查看更多（Learn More...）连接到 Mozilla“地理位置感知浏览”的帮助页面来了解更多这方面的信息。
- 可以选择共享你的地理位置。
- 可以选择不共享你的地理位置。
- 可以勾选“在本站点记住我的选择”（Remember for this site），之后信息条就不会在该站点再次出现。

而且，这个信息条是：

- 非模态的，因此不会阻止你切换到另外一个浏览器窗口或者标签页。
- 标签绑定的，因此当你切换到另外一个浏览器窗口或者标签页的时候，它就会消失，而当你切换回的时候又会出现。
- 无条件的，因此浏览器是无法阻止该信息条弹出的。
- 阻塞式的，因此直到你确认分享位置信息之前，浏览器都无法确定你的地理位置。

调用 getCurrentPosition()方法是一个简单的函数调用，会触发浏览器弹出信息条，该方法需要一个回调函数作为参数（文中是 show_map()方法），此方法会立刻返回，但是这并不意味着就获取到了用户的位置信息。真正获取位置信息是通过如下这个回调函数：

```
function show_map(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  // 让我们显示地图或做些有趣的事!
}
```

调用此回调函数需要传入 coords 和 timestamp 这两个属性的对象作为参数，其中 timestamp 是指计算位置信息的日期和时间（由于各项操作都是异步的，用户阅读信息条、同意共享她的地理位置以及备有 GPS 的设备去连接 GPS 卫星等等操作都是需要时间的，因此无法提前知道何时会获取到用户位置信息）coords 对象有像 latitude 和 longitude 这样的属性，顾名思义，这两个属性就是表示用户的地理位置。表 6.2 中列出了 position 对象的属性：

表 6.2 position 对象的属性

属性	类型	备注
coords.latitude	double	度（十进制）
coords.longitude	double	度（十进制）
coords.altitude	double 或 null	米（海拔）
coords.accuracy	double	米
coords.altitudeAccuracy	double 或 null	米
coords.heading	double 或 null	度（顺时针，以正北为基准）
coords.speed	double 或 null	米/秒
timestamp	DOMTimeStamp	类似 Date()对象

表中只有前三个属性是确保不为空的（coords.latitude、coords.longitude 和 coords.accuracy）。其余属性因用户设备和后台定位服务器而异，可能属性值会是 null。其中 heading 和 speed 这两个属性如果有的话是基于用户前一次位置信息计算所得的。

6.4 容错处理

Handling Errors

获取地理位置是个非常复杂的过程，其中很多地方都有可能会出错。之前，我也提到了“用户同意”的观点，用户总是处于主动的有利地位，如果你想要获取用户地理位置信息，但是用户不同意，就会出现问题。那么这些在代码里面又是如何体现的呢？这里就有必要提到 getCurrentPosition()的第二个参数——容错处理的回调函数：

```
navigator.geolocation.getCurrentPosition(
  Show_map, handle_error)
```

在获取地理位置过程中有任何错误，都会调用该回调函数，并且会传入一个 PositionError 对象作为参数，表 6.3 列出了这个对象的属性。

表 6.3 PositionError 对象的属性

属性	类型	备注
code	short	可枚举
message	DOMString	与终端用户无关

其中 code 属性有以下属性值：

- PERMISSION_DENIED (1)：用户点击了信息条上的“不共享”的按钮或者直接拒绝被获取位置信息。
- POSITION_UNAVAILABLE (2)：网络不可用或者无法连接到获取位置信息的卫星。

- TIMEOUT (3)：网络可用但是花了太长的时间在计算用户的位置上。怎么样算“太长”呢？我会在下一节介绍如何定义“太长”。
- UNKNOWN_ERROR (0)：发生其他未知错误。

举个例子：

```
function handle_error(err) {
  if (err.code == 1) {
    // 用户说不!
  }
}
```

专家答疑

问：地理位置 API 对于国际空间站、月球或者其他星球有效吗？

答：地理位置的标准文档（*http://www.w3.org/TR/geolocation-API/#coordinates_interface*）申明：“地理位置相关接口调用的地理坐标参照系统是世界大地测量系统(2d)[WGS84]。不支持其他的坐标参照系统”由于国际空间站是环绕地球的，因此在空间站（*http://twitter.com/Astro_TJ*）的宇航员是可以用经纬度以及海拔来描述他们的位置的。但是对于在月球和其他星球的位置就没法描述了，因为世界大地测量系统是以地球为中心的。

6.5 方案！我要方案！

Choices! I Demand Choices!

一些主流移动设备，诸如 iPhone 和 Android 手机，支持两种获取用户地理位置的方式。第一种方式是通过距离用户相对邻近的不同信号基站来计算用户位置信息，这些信号基站是由该用户的手机运营商建造并维护的。这种方式速度快而且不需要专用的 GPS 硬件，但是获取的只是大致的位置信息，不够精确。这里所指的“大致位置信息”可能是精确到用户所属的城市街区，也有可能只是指出用户在方圆一公里的范围内，这都取决于用户所处位置的信号基站的密度。

第二种方式是使用用户设备上的专用 GPS 设备通过和专用的环地球 GPS 定位卫星通讯来获取位置信息。GPS 通常能够在几米的误差内准确定位用户位置，但是缺点是专用的 GPS 芯片会很耗电，因此手机和其他获取位置信息的专用设备都会默认关闭该芯片，在需要的时候才会开启。这就意味着使用它的时候会有些延迟，因为启动芯片的时候需要初始化和 GPS 定位卫星的连接。如果你在 iPhone 或者其他智能手机上使用过 Google 地图的话就会有这种感觉了，在你看到最终自己位置的时候，你能明显的感到以下几个步骤：一开始你看到一个近似你位置的大圆圈（正在寻找最近的信号基站），随后是大圆圈变成一个比较小的圆圈（利用其他信号基站计算更接近的位置），最后地图以一个点来表示你的精确位置（从 GPS 定位卫星获取到）。

之所以介绍上述两种定位方式，主要是为了说明是否需要高精度的地理位置信息取决于你的 web 应用。举个例子：如果你的应用是提供用户所在地附近的电影院信息，那就不需要知道用户准确的地理位置，因为一个地区不可能有很多的电影院，哪怕在密集的城市也是如此，可以说电影院的个数都是屈指可数的。反之，如果你的应用是给用户实时导航的，那就必须要知道用户精确的地理位置，因为只有这样，你才能提供用户“行驶 20 米后右转”之类的导向。

getCurrentPosition()方法还接受第三个可选的参数——PositionOptions 对象，该对象有三个属性（参见表 6.4）。这三个属性都是可选的，也就是说你既可以全部设置也可以全部不设。

表 6.4 PositionOptions 对象的属性

属性	类型	默认值	备注
enableHighAccuracy	boolean	false	设成 true 可能会使得获取位置的速度变慢
timeout	long	没有默认值	单位：毫秒
maximumAge	long	0	单位：毫秒

enableHighAccuracy 属性顾名思义表示开启高精度定位。如果该属性被设置成 true，且当设备支持地理位置 API 同时用户同意共享其位置时，设备就会尝试着去获取用户高精度的位置信息。但是，iPhone 和 Android 手机对低精度和高精度定位都会区分不同的许可权限，因此当把 enableHighAccuracy 属性设置成 true 的情况下，getCurrentPosition()方法可能会调用失败，而如果设置成 false 则会成功。

timeout 属性表示 web 应用获取用户位置信息的最长等待时间。只有当用户允许计算位置信息时，计时器才会开始计时。所以，计时器所计算的并不是用户操作的时间，而是网络请求的时间。

maximumAge 属性允许设备将缓存的位置信息快速返回给 Web 应用。举个例子：假设你正好在上午 10：00 的时候成功通过调用 getCurrentPosition()获取了用户位置信息，1 分钟以后也就是上午 10:01 的时候你又调用了 getCurrentPosition()方法同时将 maximumAge 属性设置成 75000：

```
navigator.geolocation.getCurrentPosition(
  success_callback, error_callback, {maximumAge: 75000});
```

这表示你并不一定非要用户当前的位置信息，而是只需要他 75 秒（75000 毫秒）内的位置信息就可以了。在这种情况下，因为之前第一次就已经通过调用 getCurrentPosition()方法获取过用户位置信息了，设备其实已经知道用户 60 秒（60000 毫秒）前的位置了，由于这是在指定的时间窗口内，因此设备就不需要再去重新计算用户位置，而是直接把第一次获取的位置信息返回，包括经纬度、精度和时间戳（上午 10:00）。

综上所述，在要获取用户位置信息前，应当想清楚是否需要高精度的位置信息，然后相应设置 enableHighAccuracy 属性值。如果需要多次获取用户位置信息，应当想清楚上一次位置信息的有效时间大概是多久，然后相应地去设置 maximumAge 属性值。如果需要持续不断地获取用户位置信息，那就应当使用 watchPosition() 方法而不是 getCurrentPosition()方法。

watchPosition()方法和 getCurrentPosition()方法结构相同，都需要三个参数：必须提供的在成功获取用户信息后的回调函数、可选的用于容错处理的回调函数和可选的 PositionOptions 对象。不同之处在于，你不用主动地去获取用户位置信息，每次用户位置改变，这些回调函数就会被调用。设备会确定一个轮询的最佳时间间隔，定时检测用户位置是否改变，如果改变就去调用相应的回调函数。你可以利用这一特性实现你想要的功能，比如在地图上实时更新用户位置的标记，或者给用户提供实时的位置导航。

watchPosition()方法会返回一个数字，建议把这个数字保存下来，当你想停止监测用户位置的时候就可以调用 clearWatch()方法，并把保存下来的数字作为参数传入，这样设备就不会再调用回调函数了。如果你曾经用过 JavaScript 中的 setInterval()与 clearInterval()方法，你会发现它们的用法是一样的。

6.6 IE 怎么办？

What About IE?

IE 浏览器并不支持上文提到的 W3C 标准的地理位置 API（参见第 117 页“地理位置 API”一节）。但别沮丧，还有 Gears。Gears 是 Google 开发的兼容 Windows、Mac、Linux、Windows Mobile 以及 Android 平台的开源浏览器插件。它为早期的浏览器提供了包括地理位置 API 在内的诸多特性。尽管该地理位置 API 和 W3C 标准的地理位置 API 有所不同，但功能是一样的。[1]

应当指出，在传统的移动手机平台上有许多设备特有的地理位置 API。比如：BlackBerry、Nokia、Palm 和 OMTP BONDI 都提供了它们私有的地理位置 API。当然，它们和 Gears 以及 W3C 标准的地理位置 API 也完全不同。杯具！

6.7 geo.js 来拯救

Choices! I Demand Choices!

geo.js 是一个使用 MIT 协议的开源 JavaScript 库。它封装了 W3C 标准、Gears 以及移动平台所提供的各种不同类型的地理位置 API，提供了统一的调用接口。要使用该 JavaScript

[1] 译注：Google Gears 开发者博客在 2010 年 2 月份的博文(*http://gearsblog.blogspot. com/2010/02/hello-html5.html*)中证实：Gears 已停止研发，并鼓励大家拥抱 HTML5。Google 会继续修复现有 Gears 代码中的 bug，但不会再添加任何新特性。

库，只须在页面底部加入两个<script>元素（理论上，<script>元素放在页面任意位置都可以，但是如果放在<head>里会减慢页面的加载，因此千万不要这么做！）

这两个脚本中第一个是 gears_init.js(*http://code.google.com/apis/gears/gears_init.js*)，用于初始化 Gears（对于安装了 Gears 的设备），另一个脚本是 geo.js：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML 5</title>
</head>
<body>
  ...
  <script src="gears_init.js"></script>
  <script src="geo.js"></script>
</body>
</html>
```

使用 geo 就不需要再考虑不同地理位置 API 的情况了，使用统一提供的接口就可以：

```
if (geo_position_js.init()) {
    geo_position_js.getCurrentPosition(geo_success, geo_error);
}
```

一步一步来解释，上述代码中，首先显式调用了 init()函数。如果当前设备支持任意一种地理位置 API，该函数就会返回 true：

```
if (geo_position_js.init()) {
```

调用 init()函数只是检测当前设备是否支持地理位置 API，并没有真正去获取用户的位置信息，要获取用户位置信息需要调用 getCurrentPosition()方法：

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

调用 getCurrentPosition()方法会触发浏览器询问用户是否愿意分享位置信息。如果是通过 Gears 来获取位置信息的话，浏览器当前窗口会出现一个弹出层，询问用户是否信任当前使用 Gears 的站点；而如果是通过 W3C 标准的地理位置 API（也就是浏览器原生支持的）获取用户信息的话，那么提示会因浏览器而异，比如 Firefox 3.5 就会在浏览器当前页面窗口的最上面显示一个信息条询问用户是否愿意分享位置信息。

getCurrentPosition()方法接受两个回调函数作为参数。如果通过 getCurrentPosition()方法，在经过用户允许的情况下成功获取了用户的位置信息，那么第一个回调函数就会被调用，本例中这个回调函数就是 geo_success 方法：

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

此回调函数只接受一个参数，这个参数包括了所有获取到的位置信息：

```
function geo_success(p) {
  alert("Found you at latitude " + p.coords.latitude +
      ", longitude " + p.coords.longitude);
}
```

如果调用 getCurrentPosition()没有成功获取到用户位置信息（由于用户不同意共享位置信息或者因为地理位置 API 因为某些原因调用失败），第二个回调函数会被调用，本例中这个回调函数就是 geo_error：

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

失败的回调函数不需要任何参数：

```
function geo_error() {
  alert("Could not find you!");
}
```

geo.js 目前不支持 watchPosition()方法。如果要实时获取用户位置信息，那就不得不自己去定时调用 getCurrentPosition()方法。

6.8 一个完整的例子

A Complete Example

让我们来演示一个利用 geo.js 获取用户信息并将其显示在地图上的实例。

当页面被载入的时候，需要调用 geo_position_js.init()来确定设备是否支持地理位置 API。如果支持，那么在页面上可以设置一个链接，用户可以点击这个链接获取自己的位置信息。链接被点击的时候会调用如下 lookup_location()函数：

```
function lookup_location() {
  geo_position_js.getCurrentPosition(show_map, show_map_error);
}
```

如果用户同意共享位置信息，且服务端能够成功获取用户信息，geo.js 会调用第一个回调函数——show_map()，同时传入一个 loc 对象作为参数，loc 对象有一个 coords 属性，该属性包含了经度、纬度以及精度信息（本例中不需要用到精度信息）。show_map()方法中利用 Google 地图 API 将地图内嵌到页面中：

```
function show_map(doc) {
  $("#geo-wrapper").css({'width': '320px', 'height': '350px'});
  var map = new GMap2(document.getElementById("geo-wrapper"));
  var center = new GLatLng(loc.coords.latitude, loc.coords.longitude);
  map.setCenter(center, 14);
  map.addControl(new GSmallMapControl());
  map.addControl(new GMapTypeControl());
  map.addOverlay(new GMarker(center, {draggable: false, title: "You are here (more or
```

```
    less)" }));
}
```

如果 geo.js 无法获取用户位置信息，那么第二个回调函数——show_map_error()就会被调用：

```
function show_map_error() {
  $("#live-geolocation").html('Unable to determine your location');
}
```

6.9 扩展阅读

Further Reading

- W3C 地理位置 API（*http://www.w3.org/TR/geolocation-API/*）。
- Gears（*http://tools.google.com/gears/*）。
- BlackBerry 地理位置 API（*http://www.tonybunce.com/2008/05/08Blackberry-Browser-Amp-GPS.aspx*）。
- Nokia 地理位置 API（*http://www.forum.nokia.com/infocenter/index.jsp?topic=/Web_Developers_Library/GUID-4DDE31C7-EC0D-4EEC-BC3A-A0B0351154F8.html*）。
- Palm 地理位置 API（*http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition*）。
- OMTP BONDI 地理位置 API（*http://bondi.omtp.org/1.0/apis/geolocation.html*）。
- geo.js（*http://code.google.com/p/geo-location-javascript/*），一个封装了地理位置 API 的脚本。

第 7 章

Web 应用本地存储的过去、现在和未来

The Past, Present, and Future of Local Storage for Web Applications

7.1 引言

Diving In

持久化本地存储一直以来都是本地客户端应用相较于 Web 应用的优势之一。对于本地客户端应用，操作系统提供了一个抽象层，用于存储和检索诸如配置或者运行状态等一些应用程序相关的数据。这些数据在不同的平台上可以被存储在注册表、INI 文件、XML 文件或者其他地方。这类存储都以键/值对的形式进行存储，如果本地客户端应用需要以非键/值对的形式来存储数据，就需要搭建数据库，实现自己的文件格式，或者寻找其他的解决方案。

纵观历史，Web 应用一直没有实现真正意义上的本地存储。早期发明的 cookie 可以用来持久化存储少量的数据。但是，它有一些潜在的硬伤：

- 每一次 HTTP 请求都会发送 cookie 信息。这样，一次又一次无意义地传输同样的数据，会使得 Web 应用变慢。
- 每一次 HTTP 请求中发送的 cookie 信息都以未加密的形式在因特网上传输，这样很不安全（除非整个 Web 应用都是基于 SSL 的）。
- Cookie 信息量最大不能超过 4KB——这足以拖慢 Web 应用（参见第一点），但对于实用性而言还是太小了。

我们真正需要的是：

- 足够大的存储空间。
- 存储在客户端。
- 在页面刷新的情况下也能将存储的数据持久化。
- 存储的数据不会每次都传输给服务器（不会产生额外的 HTTP 请求）。

为了达到这样的目的，人们做了许多尝试，但最终的解决方案都不那么尽如人意。

7.2 HTML5 之前的伪本地存储简史

A Brief History of Local Storage Hacks Before HTML5

起初，世界上只有 IE 浏览器，至少微软是希望全世界都这么认为。最终，在第一次浏览器大战期间，微软有许多不错的新发明，并且把它们加入到终结这场战争的 IE 浏览器中，其中有一个叫做 DHTML 行为（DHTML Behaviors）[1]，而这些行为之中有一个被称作：userData。

UserData 允许 Web 页面在每个域下存储最大 64KB 的 XML 层次结构化数据（对于信任的域下，如内网站点，可以存储 10 倍的数据，也就是 640KB，看上去这个空间已经足够，而事实上还是太小）。IE 不会弹出任何形式的权限提醒对话框，也不允许提高该存储空间。

2002 年，Adobe 在 Flash 6 中引入了一个新特性。该特性不幸被冠以一个错误的名字："Flash cookies"。在 Flash 运行环境中，该新特性更应该被称为：本地共享对象（Local Shared Objects）或者 LSOs。简单来说，它允许 Flash 对象在每个域下存储最大 100KB 的数据。2005 年，Brad Neuberg 开发出了桥接 Flash 和 JavaScript 的早期原型，称为：AJAX 大型存储系统，简称 AMASS。但是因受限于 Flash 设计上的缺陷，该系统没有得到很好地发展。到了 2006 年，随着 Flash8 中 ExternalInterface 的问世，在 JavaScript 中访问 Flash 的本地共享对象变得越来越容易和快速。于是 Brad 重写了 AMASS，并将它集成到流行的 Dojo 工具集下的 dojox.storage 包中。有了 AMASS，就相当于每个域下"免费"拥有了 100KB 的存储空间。除此之外，它还能提示用户授权增加更大量级的存储空间（1MB，10MB，等等）。

随后，2007 年，Google 启动了 Gears 项目，一个开源的浏览器插件，旨在为浏览器提供额外的新特性。我们之前在讨论"为 IE 浏览器提供了地理位置 API"时已经介绍过了 Gears（参见第 123 页"IE 怎么办？"一节）。Gears 为内置的 SQL 数据库（基于 SQLite）提供了 API。得到用户许可后，每个域下 Gears 可以在 SQL 数据库表中存储无限（没有大小限制）的数据。

在此期间，Brad Neuber8 和其他成员继续尝试让 dojox.storage 为不同类型的存储插件和 API 提供统一的接口。到了 2009 年，dojox.storage 可以自动检测（提供了统一的接口）Adobe Flash、Gears、Adobe AIR，以及只有在老版本的 Firefox 浏览器上才能实现的 HTML5 存储的早期原型。

从以上这些对本地存储的解决方案中，不难看出有个共同点：要么是针对特定浏览器，要么是依赖于第三方插件。尽管 dojox.storage 做了很大努力来统一这些本地存储的实现

[1]译注：微软在 IE5.5 中开始引入 DHTML 行为，它们是封装了某些特殊功能或页面行为的组件集。详细内容请参见 *http://msdn.microsoft.com/en-us/library/ms531079(v=VS.85).aspx*。

差异，但是本身这些差异暴露出了很多问题，如接口不同、存储限制不同，以及用户体验不同。而 HTML5 就是来解决这些问题的：提供统一的标准 API，跨浏览器的原生实现，并且完全不需要依赖第三方插件。

7.3 HTML5 存储介绍

Introducing HTML5 Storage

我这里指的“HTML5 存储”其实是一份标准文档，名字叫“Web 存储（Web Storage）”。它原先是 HTML5 标准的一部分，后来由于某些政治原因被抽离出来作为单独的一份标准。某些浏览器提供商又把它叫做“本地存储”或者“DOM 存储”。它的命名由于某些相近、类似的新兴标准就变得更加复杂，关于这部分会在本章后面作介绍。

那么什么是 HTML5 存储呢？简单地说，它提供了一种方式，让 Web 页面在客户端浏览器中以键值对的形式在本地存储数据。和存储在 cookie 中的数据一样，无论用户是离开该站点、关掉浏览器标签页、退出浏览器还是其他的操作，存储的数据都会存在。但是和 cookie 不同的是，这些数据不会随着每次 HTTP 请求发送到远程服务器（除非手动将其发送到远程服务器），并且不像之前所提到的持久化本地存储方案（上一节中所描述的），它是浏览器原生实现的，因此不需要依赖第三方浏览器插件。

如表 7.1 所示，包括 IE 在内几乎所有浏览器的最新版本都支持 HTML5 存储。

表 7.1　支持 HTML5 存储特性的浏览器及其版本号列表一览

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
8.0+	3.5+	4.0+	4.0+	10.5+	2.0+	2.0+

利用 JavaScript，通过全局 window 对象上的 localStorage 对象可以操作 HTML5 存储。不过在这之前，应当先检测浏览器是否支持该特性（参见第 21 页“本地存储”一节）：

```
function supports_html5_storage() {
  return ('localStorage' in window) && window['localStorage'] !== null;
}
```

如果不想自己写这个方法，可以使用 Modernizr 库（参见第 16 页“Modernizr：一个 HTML5 特性检测库”）提供的方法，检测浏览器是否支持 HTML5 存储特性：

```
if (Modernizr.localstorage) {
  // window.localStorage 可用!
} else {
  // 没有本地 HTML5 存储支持 :(
  // 也许可以测试 dojox.storage 第三方方案
}
```

7.4 使用 HTML5 存储

Using HTML5 Storage

HTML5 存储是基于键值对的形式的。存储和检索数据都是通过指定的键名：

```
interface Storage {
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
};
```

键名的类型是字符串类型。值可以是包括字符串、布尔值、整数，或者浮点数在内的任意 JavaScript 支持的类型。但是，最终数据是以字符串类型存储的。所以，如果需要把存储或者读取的数据当成其他非字符串类型使用，需要利用类似 parseInt()、parseFloat() 的函数强制将其转化为期望的 JavaScript 数据类型。

调用 setItem()时，如果指定的键名已经存在，那么新传入的数据会覆盖原先的数据。调用 getItem()时，如果传入的键名不存在，那么会返回 null，而不会抛出异常。

与其他 JavaScript 对象类似，localStorage 对象也可以被看做是一个关联数组。利用这一点，就可以简单使用方括号来替换调用 getItem()和 setItem()方法实现存储和读取数据。例如，下面这段代码：

```
var foo = localStorage.getItem("bar");
// ...
localStorage.setItem("bar", foo);
```

可以用方括号的语法重写成如下形式：

```
var foo = localStorage["bar"];
// ...
localStorage["bar"] = foo;
```

还有其他一些方法用于删除指定键名的数据，以及清除整个存储数据（等于一次性删除所有的键值对）：

```
interface Storage {
  deleter void removeItem(in DOMString key);
  void clear();
};
```

调用 removeItem()时如果传入的键名不存在，那么什么都不会发生。

最后，localStorage 对象上还有一个 length 属性，用来获取当前存储区所有数据的个数。同时可以通过索引值来遍历所有的键名：

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
};
```

如果调用 key()方法传入的索引值越界了(不在 0~length-1 之间)，该方法会返回 null。

7.4.1 跟踪 HTML5 存储区的改动

如果想要以可编程方式跟踪存储区的改动，可以通过捕获 storage 事件来进行。任何时候 setItem()、removeItem()或者 clear()方法被调用，并且真的改动了数据时，都会在 window 对象上触发 storage 事件。例如：如果给一个条目设置一个原本存在的值，或者在没有数据的情况下调用 clear()方法，storage 事件不会被触发，因为事实上数据区域根本没有发生变动。

只要支持 localStorage 对象就一定支持 storage 事件，IE8 也不例外。因为 IE8 不支持 W3C 标准的注册事件方法：addEventListener(尽管 IE9 将会支持)，因此在注册 storage 事件时首先需要检测浏览器支持哪种事件机制。如果你对不同的事件机制很了解，可以直接跳至本节最后。捕获 storage 事件和捕获其他事件的方式是一样的。如果相比其他的 JavaScript 库，你更倾向于用 jQuery 来注册事件，同样可以用 jQuery 来注册 storage 事件：

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

上述代码中，handle_storage()回调函数被调用时，会传入一个 StorageEvent 对象作为参数，但是 IE 不同，IE 下面的事件对象存储在 window.event 上：

```
function handle_storage(e) {
  if (!e) { e = window.event; }
}
```

这里的 e 就是 StorageEvent 对象，表 7.2 列出了这个对象上很多有用的属性。

表 7.2 StorageEvent 对象的属性

属性	类型	描述
key	字符串	加入、修改或者删除的键名
oldValue	任意	之前的数据（被覆盖的情况）或者 null（如果有新数据项加入）
newValue	任意	新数据或者 null（如果数据项被删除）
url[a]	字符串	调用这个触发数据区变动的函数的所属页面地址

[a] url 属性最初叫 uri。在标准作出改动前，有些浏览器称该属性为 uri，因此为了兼容性考虑，应该首先检测 url 属性是否存在，如果不存在再检测 uri 属性是否存在。

storage 事件是无法撤销存储区改动的。在 handle_storage()回调方法中，没有办法停止正在发生的对存储区的改动，该方法被回调仅仅表示浏览器告诉你："嘿，存储区数据正在发生变动，但是对此你什么也不能做，我只是通知你一下。"

7.4.2 现有浏览器的局限性

之前介绍利用第三方插件实现本地存储功能的历史时（参见第 128 页“HTML5 之前的伪本地存储简史”一节），提到过每种技术存在的局限性，比如存储空间的局限等。但是到目前为止都没有提到过任何关于现在标准的 HTML5 存储的局限性。

每个域默认拥有 5MB 的存储空间。这个大小在各浏览器下难得的惊人一致，尽管它目前只是作为对 HTML5 存储标准的建议（还未被正式列入 HTML5 的存储标准中）。要记住的是，数据最终会以字符串类型，而不会以它原来的数据类型存储。例如，存储大量整型或浮点型数据时，在存储区中，会把每个数字存储为单个字符，而不会是整数或者浮点数。

存储数据时如果超过存储空间的配额，就会抛出 QUOTA_EXCEEDED_ERR 异常。你可能会问“能否调整存储空间的大小？”，很遗憾地告诉你“不能”。截止到本书撰写时，没有任何一款浏览器支持开发者以编程的方式扩大存储空间大小。部分浏览器，比如 Opera，允许用户控制每个站点的存储空间配额，但也纯粹是一种用户驱动的行为，而非 Web 开发者在其应用中可以控制的。

7.5 HTML5 存储实践

HTML5 Storage in Action

让我们来看一个 HTML5 存储实践的例子。回想我们在第四章构建的 Halma 游戏（参见第 75 页“一个完整的例子”）。Halma 游戏存在一个小问题：如果游戏过程中浏览器窗口被关闭了，之前的游戏记录都会丢失。但是，有了 HTML5 存储，就可以在本地浏览器中把记录存储下来。可以看个在线的示例：（*http://diveintohtml5.org/examples/localstorage-halma.html*），示例中，移动一些棋子，然后关掉当前浏览器标签页，再重新打开示例页面。如果浏览器支持 HTML5 存储，页面上就会显示出你经过上次移动后当前棋子的位置，同时也记住了移动的步数、每个棋子移动后的位置，甚至还记住了哪个棋子当前被选中。

这是如何实现的呢？请看下面的函数，游戏过程中每发生一次操作，该方法都会被调用：

```
function saveGameState() {
    if (!supportsLocalStorage()) { return false; }
    localStorage["halma.game.in.progress"] = gGameInProgress;
    for (var i = 0; i < kNumPieces; i++) {
      localStorage["halma.piece." + i + ".row"] = gPieces[i].row;
      localStorage["halma.piece." + i + ".column"] = gPieces[i].column;
  }
  localStorage["halma.selectedpiece"] = gSelectedPieceIndex;
  localStorage["halma.selectedpiecehasmoved"] = gSelectedPieceHasMoved;
  localStorage["halma.movecount"] = gMoveCount;
  return true;
}
```

上述方法中，使用 localStorage 对象将当前游戏是否在进行中的状态保存下来（gGameInProgress 是个布尔值）。如果游戏在进行中，就去迭代每一个棋子（gPieces 是一个 JavaScript 数组）然后将每个棋子位于第几行第几列的信息保存下来。之后，又保存了其他一些游戏状态，包括哪个棋子被选中（gSelectedPieceIndex 是个整数）、棋子是否处于跳格中的状态（gSelectedPieceHasMoved 是个布尔值），以及游戏到目前为止总的移动步数（gMoveCount 是个整数）。

页面载入时，我们会调用 resumeGame()函数，而不是自动调用 newGame()函数，以至于那些变量被重置。使用 HTML5 存储，resumeGame()方法会检测本地是否存储了某个进行中的游戏的状态，如果存储了，就用 localStorage 对象把状态恢复出来。

```
function resumeGame() {
    if (!supportsLocalStorage()) { return false; }
    gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
    if (!gGameInProgress) { return false; }
    gPieces = new Array(kNumPieces);
    for (var i = 0; i < kNumPieces; i++) {
      var row = parseInt(localStorage["halma.piece." + i + ".row"]);
      var column = parseInt(localStorage["halma.piece." + i + ".column"]);
      gPieces[i] = new Cell(row, column);
    }
    gNumPieces = kNumPieces;
    gSelectedPieceIndex = parseInt(localStorage["halma.selectedpiece"]);
    gSelectedPieceHasMoved = localStorage["halma.selectedpiecehasmoved"] == "true";
    gMoveCount = parseInt(localStorage["halma.movecount"]);
    drawBoard();
    return true;
}
```

此函数中最重要的一部分，之前在本章中已经提到过，现在再重复一次：数据最终是以字符串形式存储的。如果存储了其他非字符串类型的数据，获取时需要手动做类型转换。比如：用来标示游戏是否在进行中的变量（gGameInProgress）是一个布尔值。在 saveGameState()方法中，存储该数值时是不用担心类型问题的：

```
localStorage["halma.game.in.progress"] = gGameInProgress;
```

但是在 resumeGame()方法中，获取该数值后就需要手动将其从字符串转换成布尔值：

```
gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

同样，移动的步数保存在整型变量 gMoveCount 中。在 saveGameState()函数中，我们不用担心类型问题：

```
localStorage["halma.movecount"] = gMoveCount;
```

但是在 resumeGame()方法中获取该数值后，就需要利用 JavaScript 内置的 parseInt()方法将其从字符串转换为整数：

```
gMoveCount = parseInt(localStorage["halma.movecount"]);
```

7.6 超越键值对的存储形式

Beyond Named Key/Value Pairs: Competing Visions

相比之前介绍的很多以前混乱的伪本地存储的实现（参见第 128 页“HTML5 之前的伪本地存储简史”一节），现在的 HTML5 存储好太多了。新的 API 已经被标准化，并在所有主流的浏览器、平台和设备上实现。对于 Web 开发者来说，这种情况不是那么常见的。但是未来的持久化本地存储绝对不会仅仅是以这 5MB 的键值对形式存储。目前就已经呈现出了很多竞争方案。

其中一个就是大家熟知的 SQL。2007 年，Google 启动了 Gears，一个开源的跨浏览器插件。它内置了基于 SQLite 的数据库。这个插件的早期原型后来对 Web SQL 数据库标准的制定产生了重要的影响。Web SQL 数据库（Web SQL Database）（前身叫“WebDB”）对 SQL 数据库进行了封装，允许以如下形式使用 JavaScript 操作数据库：

```
openDatabase('documents', '1.0', 'Local document storage', 5*1024*1024,
function (db) {
  db.changeVersion('', '1.0', function (t) {
    t.executeSql('CREATE TABLE docids (id, name)');
  }, error);
});
```

正如上述代码所示，对数据库的操作作为字符串参数传入 executeSql()方法来执行。该参数可以是任何包括 SELECT、UPDATE、INSERT 和 DELETE 在内的合法 SQL 语句。最酷的是可以直接通过 JavaScript 像后台数据库编程那样操作数据库。

表 7.3 列出了目前已经支持 Web SQL 数据库标准的浏览器。

表 7.3 浏览器对 Web SQL 数据库的支持情况

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	.	4.0+	4.0+	10.5+	3.0+	.

如果你使用过多种数据库产品，那你一定清楚“SQL”更多是一个市场营销术语，而不是一种明确的标准。有人或许会说“HTML5 不也是这样吗？”。好吧，先不考虑这个。事实上也存在真正的 SQL 标准——SQL-92——但是世界上没有一个数据库服务器完全遵循或只遵循这个标准。不同的 SQL 产品，比如：Oracle's SQL、Microsoft's SQL、MySQL's SQL、PostgreSQL's SQL 及 SQLite's SQL 都有自己的一套 SQL 标准，而且不断加入新的 SQL 特性，因此，说“SQLite's SQL”其实也不是非常恰当，而是要说“SQLite X.Y.Z 版本所带的 SQL 版本”。

正因如此，在 Web SQL 数据库标准文档开始的部分就有如下这段免责申明：

> 这份标准目前陷入了这样一个僵局：所有的标准实现者都是用同样的 SQL 后台（Sqlite），但是我们更需要基于不同 SQL 后台的实现来促进这份标准不断完善。否则，对于 SQL 的表述就变成是专门针对 Sqlite 的了，这是无法成为一份标准的。

下面我要介绍另一个截然不同的为 web 应用提供高级、可持久化的本地存储方案：Indexed Database API，之前叫“WebSimpleDB”，现在被称作“IndexedDB”。

Indexed Database API 引入了“对象存储”的概念。对象存储的概念其实和 SQL 数据库中的很多概念是相通的。在数据库中有记录，每条记录有一系列字段，每个字段在数据库被创建的时候都被指定数据类型。你可以查询数据子集，然后通过“游标”来枚举这些记录。对象存储中的变化在“事务”中处理。。

如果你做过 SQL 数据库编程，那么对这些概念一定很熟悉了。对象存储和 SQL 数据库最大的区别就是对象存储没有结构型的查询语言。因此，像“`SELECT * from USERS where ACTIVE = 'Y'`”这样的查询语句是不支持的。但是可以使用对象存储提供的方法，在名为 USERS 的数据库上打开一个游标，通过它来枚举所有的记录，并且过滤掉有被激活的用户，随后通过存取方法来获取余下数据中每一个字段的值，这样就达到了之前那句查询语句一样的效果。Mozilla 的文章“初探 IndexedDB”（*http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/*）对 IndexedDB 和 Web SQL 数据库作了详细的对比，是一本很好的介绍 IndexDB 工作原理的教程。

截至本书撰写时，主流浏览器都没有实现对 IndexedDB 的支持。2010 年 6 月上旬，Mozilla 曾说过会在未来几个星期内做些这方面的尝试，但是到现在也没有说肯定会在 Firefox4 中支持 IndexedDB。相反，Mozilla 明确表示永远不会支持 Web SQL 数据库。Google 宣称将考虑在 chrome 中加入对 IndexedDB 的支持，甚至微软也表示“IndexedDB 将会推动整个 Web 的发展”。

作为 Web 开发者，你会使用 IndexedDB 吗？就目前而言，它连影子都还没有呢。或许一年后会有些眉目吧[2]。想要了解更多本地存储的内容，请阅读下面“深入阅读”小节。

7.7 扩展阅读

Further Reading

HTML5 存储：

- HTML5 存储标准（*http://dev.w3.org/html5/webstorage/*）
- MSDN 上《DOM 存储介绍》（*http://msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx*）

[2]译注：截至本书翻译时，Firefox 4.0 beta 和 Chrome 8 dev 版本分别以私有属性的方式（window.moz_indexedDB & window.webkitIndexedDB）加入了对 IndexedDB 的支持。Modernizr 库的最新版本也依据现状增加了对 IndexedDB 的检测。

- Shwetank Dixit 撰写的《Web 存储：更简单更强大的客户端数据存储》（*http://dev.opera.com/articles/view/web-storage/*）。
- Mozilla 开发者中心的《DOM 存储介绍》（*https://developer.mozilla.org/en/dom/storage*）。（注意：此文大量篇幅都在介绍 Firefox 中对 globalStorage 对象的原型实现，该对象是一个非标准的 localStorage 对象的前身。Mozilla 在 Firefox3.5 中加入了标准的 localStorage 接口）。
- IBM 开发者作品上的一篇教程《利用 HTML5 实现手机 web 应用的本地存储》（*http://www.ibm.com/developerworks/xml/library/x-html5mobile2/*）。

Brad Neuberg 早期的工作（在 HTML5 诞生之前）：

- 《IE 浏览器原生支持持久化数据存储?!?!》（*http://codinginparadise.org/weblog/2005/08/ajax-internet-explorer-has-native.html*），这是一篇介绍 IE 浏览器中 userData 对象的文章。
- Dojo 存储（*https://docs.google.com/View?docid=dhkhksk4_8gdp9gr#dojo_storage*），Dojo 离线库教程（现在已经停止了）的一部分。
- dojox.storage.manager API 参考（*http://api.dojotoolkit.org/jsdoc/HEAD/dojox.storage. manager*）。
- dojox.storage 的 svn 仓库（*http://svn.dojotoolkit.org/src/dojox/trunk/storage/*）。

Web SQL 数据库：

- Web SQL 数据库标准（*http://dev.w3.org/html5/webdatabase/*）
- Remy Sharp 撰写的《Web SQL 数据库介绍》（*http://html5doctor.com/introducing-web-sql-databases/*）。
- Web 数据库示例（*http://html5demos.com/database*）。
- persistence.js（*http://zef.me/2774/persistence-js-an-asynchronous-javascript-orm-for-html5gears*），基于 Web SQL 和 Gears 的《异步 JavaScript ORM》。

IndexedDB：

- IndexedDB 数据库 API 参考（*http://dvcs.w3.org/2006/webapi/IndexedDB/*）。
- Arun Ranganathan 和 Shawn Wilsher 撰写的《超越 HTML5：数据库 API 以及通往 IndexedDB 之路》（*http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/*）。
- Arun Ranganathan 撰写的《Firefox4：初步尝试 IndexedDB》（*http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/*）。

第 8 章

离线 Web 应用

Let's Take This Offline

8.1 引言

Diving In

什么是离线 Web 应用？听起来似乎有些矛盾。Web 页面是被下载下来后渲染的，下载就意味着需要联网。那又如何做到离线状态下下载呢？当然，在离线状态下是肯定没法下载的，但是可以在在线的状态下下载，在离线的状态下使用。HTML5 离线应用就是这样工作的。

> 简单来说，离线 Web 应用其实只是一个 URL 列表文件。列表中的 URL 分别可以指向 HTML、CSS、JavaScript 文件、图片或者是其他资源。该列表文件被称作“清单文件（manifest file）”，它是一种文本文件，可以被放置在 Web 服务器的任何位置。离线 Web 应用的首页会引入该清单文件，支持 HTML5 离线应用的浏览器就会从清单文件中读取 URL 信息，并下载对应的资源，将它们缓存到本地，同时会保证本地和服务器资源的一致性。这样，在离线状态下访问 Web 应用的时候浏览器就会自动切换到从本地直接读取这些资源。

支持 HTML5 离线应用的浏览器提供了一个 DOM 对象上的属性，该属性标记了当前访问状态是在线还是离线以及提供了对应的事件，这些事件会在访问状态发生变更的时候被触发。这对于 Web 开发者实现离线 Web 应用已经足够了。在实现过程中，大部分处理方式都取决于 Web 开发者。比如：当 Web 应用需要创建或者存储数据时，是选择在离线状态下先将数据存储到本地（参见第 7 章），等到在线后再同步到远程服务器取决于 Web 开发者本身。换句话说，HTML5 提供了离线应用特性，具体怎么去实现还是由 Web 开发者自己决定。表 8.1 列出了浏览器对 HTML5 离线应用的支持情况。

表 8.1　浏览器对 HTML5 离线应用的支持情况

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	✓	✓	✓	.	✓	✓

8.2 缓存清单
The Cache Manifest

离线 Web 应用的核心是缓存清单文件。正如之前提到的，缓存清单文件包含了所有离线状态下 Web 应用有可能会访问的资源。要下载并缓存这些资源，只需将<html>元素上的 manifest 属性指向清单文件。

```
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
<body>
...
</body>
</html>
```

缓存清单文件可以放在 Web 服务器上的任意位置，但必须要求服务器支持 text/cache-manifest 内容类型。对于基于 Apache 的 Web 服务器，需要在 Web 应用根目录下的.htaccess 为文件中增加如下这行代码：

```
AddType text/cache-manifest .manifest
```

然后确保缓存清单文件的文件名后缀为 manifest。而对于其他类型的服务器或者其他配置方式的 Apache 服务器，需参见对应的配置文档中关于控制 Content-Type 头的部分。

专家答疑

问：我的 Web 应用由多个页面组成，那我需要给每个页面设置 manifest 属性还是只需给首页设置该属性就可以了？

答：Web 应用中的每个页面都需要将 manifest 属性指向缓存清单文件。

正如专家答疑中所提到的，每个 HTML 页面都需要设置 manifest 属性，将其指向缓存清单文件，并通过恰当的 Content-Type 头来获取。接下来，让我们来看看缓存清单文件中所包含的内容。

每个缓存清单文件都以如下这一行代码开始：

```
CACHE MANIFEST
```

其余内容分为三部分：“显示”段、“默认”段和“线上白名单”段（又称“网络”段）。每段都有段头声明，如果清单文件中没有任何段头声明，则文件中列出的所有资源都默认属于“显示”段。所以，建议最好都写上段头声明。

下面是一个合法的清单文件，文件中列出了三个资源：一个 CSS 文件、一个 JavaScript 文件和一张 JPEG 格式的图片：

```
CACHE MANIFEST
/clock.css
/clock.js
/clock-face.jpg
```

该缓存清单中没有任何头部声明，因此所有列出的资源都默认属于“显示”段。属于“显示”段的资源会在在线的状态下被下载下来，然后缓存到本地，在离线状态被使用。对于上述的缓存清单文件，浏览器会将 clock.css、clock.js 和 clock-face.jpg 文件从 Web 服务器根目录下载下来缓存到本地。之后，你把网线拔掉再去刷新页面，会发现这些资源在离线状态下依然可用。

专家答疑

问：我需要将 HTML 页面本身列在缓存清单文件中吗？

答：视具体情况而定。对于单页面 Web 应用，只需确保该页面的 manifest 属性指向缓存清单文件。因为当访问带有 manifest 属性的页面时，该页面本身就是 Web 应用的一部分，因此无需将它列在缓存清单文件中。而对于多页面的 Web 应用，就需要将所有的 HTML 页面都列在缓存清单文件中，否则浏览器无法知道是否还有其他需要被缓存到本地的 HTML 页面。

8.2.1 “网络”段

考虑个复杂点的例子：假设有个计时程序利用 tracking.cgi 脚本来统计用户的访问时间，该脚本是通过<img src>属性动态载入的。如果将这个脚本列入缓存清单中在本地缓存起来，显然就和目标背道而驰了。所以，该脚本不管是在线还是离线的状态永远都不该被缓存，而是每次都应该从服务器获取。那么，如何保证资源永远不被本地缓存呢？看如下代码：

```
CACHE MANIFEST
NETWORK:

/tracking.cgi
CACHE:
/clock.css
/clock.js
/clock-face.jpg
```

上述清单文件中包含了一些段头声明。NETWORK 是“线上白名单”段的声明，属于该段的资源将永远不会被缓存（离线状态下请求这些资源会报错）；CACHE 是“显示”段的声明，属于“显示”段的那三个文件均会被缓存到本地，供离线状态下使用。

8.2.2 “默认”部分

缓存清单文件中还包括一个“默认”段。在“默认”段中，可以为那些由于某些原因无法被缓存或者缓存失败的资源指定替代资源。HTML5 标准文档中提供了如下这个很清晰的例子：

```
CACHE MANIFEST
FALLBACK:
/ /offline.html
NETWORK:
*
```

上述代码是什么意思呢？首先，考虑一个由众多页面组成的 Web 站点，就好比是维基百科。对于维基百科来说，肯定不可能也不会想把整站的资源都下载下来缓存到本地。但是假设让其中一部分被下载下来缓存到本地，你会选择哪些资源呢？看看这样的选择：假设维基百科站点是离线状态下可用的，那么就选择将用户访问过的页面及其资源下载并缓存下来。这些资源包括每个访问过的维基百科条目、讨论页面（针对维基百科条目的进行临时讨论的页面）及编辑页面（对维基百科条目进行编辑的页面）。

上述代码就是实现了这样的目的。假设维基百科站点上每个 HTML 页面（条目展示页面、讨论页面、编辑页面、历史页面）都指向上述缓存清单文件。如果是第一次访问这些页面，浏览器就会建立一个新的离线“appcache”（应用缓存：application cache 的简写），将所有缓存清单文件中列出的资源下载，并把当前页面加入到该应用缓存中。而访问的页面是之前已经访问过的，那浏览器只需将该页面加入到当前的应用缓存中。换句话说，所有访问过的页面最终都会被加入到应用缓存中。这点非常重要，这意味着离线状态下访问的页面是以“懒加载”的方式加载进来的。不需要将所有的 HTML 页面都列入到缓存清单文件中。

现在让我们来看看“默认”段。上述缓存清单文件中的“默认”段只包含一行内容。这一行内容可以分为两部分来看，第一部分是“/”（空格前面的内容），它并不表示一个 URL 而是 URL 匹配模式。它匹配了站点上所有的页面（不仅仅是首页）。当离线状态下去访问站点页面时，浏览器会去应用缓存中查找该页，如果找到了（因为之前在线状态下访问的时候浏览器已经将其下载，并添加到了应用缓存中），那就直接将页面显示出来；如果没有找到，那就显示一个错误页面。“默认”段的第二部分就指定了改错误页面为“/offline.html”。

最后我们再来看看“网络”段。“网络”段也只包含了一行内容，这一行内容只有一个“*”。该字符在“网络”段中具有特殊的含义，称为“线上白名单通配符”。它表示所有不在应用缓存中的资源仍可以在在线的状态下被下载下来。这对于“开放式”离线 Web

应用来说是非常重要的。在我们的例子中就意味着在线的状态浏览器会去获取图片、视频以及其他内嵌的资源，包括不同域下的资源。（资源在不同域下的情况对于大型的 Web 应用，即便是非离线 Web 应用来说是很常见的：HTML 页面在一个域，而图片和视频在另一域的 CDN 上）。如果没有该通配符，那么在线的状态下，该站点就不会加载任何外部图片或者视频。

继续看我们的例子，维基站点绝不仅仅只是 HTML 页面，每个页面还会包括一些常规的 CSS、JavaScript 和图片。这些资源都需要在“显示”段中列出来，这样在离线状态下这些资源就能够被正确显示出来。而“默认”段的作用实际上是对“显示”段资源的一个扩展。

8.3 事件流

The Flow of Events

到目前为止，我已经大致给大家介绍了离线 Web 应用、缓存清单和离线应用缓存（“appcache”）。资源被下载下来之后就交给浏览器，一切都“搞定了”。呵呵，当然没这么简单，我们讨论的是 Web 开发，这其中还有很多事情要做。

首先，让我们来讨论事件流，特别是 DOM 事件。当浏览器访问一个指向缓存清单文件的页面时，window.applicationCache 对象上的一系列事件会被浏览器触发，具体我在下面做了描述。我知道这看起来好像很复杂，但是我敢保证，这一定是最简单扼要又不失重点的版本。

1. 一旦浏览器访问的页面<html>元素设置了 manifest 属性，checking 事件就会被触发。（所有这里列出的事件都是在 window.applicationCache 对象被触发的。）不管之前访问过该页面还是访问过指向同一个缓存清单文件的其他页面，checking 事件都会被触发。
2. 如果浏览器从未访问过该缓存清单，那么：
   - 浏览器会触发 downloading 事件，然后去下载缓存清单中的资源。
   - 在下载资源过程中，浏览器会阶段性地触发 process 事件，通过该事件可以获取当前下载进度信息（当前已经下载了多少文件，以及还有多少文件在下载队列中）。
   - 当所有的资源都成功下载完毕后，浏览器会最后触发 cached 事件。表示当前

Web 应用所有资源已经下载完毕，准备好在离线状态下使用了。

3. 另一方面，如果之前浏览过该页面或者访问过指向同一个缓存清单文件的其他页面，那么就意味着浏览器访问过该缓存清单文件，而且有可能已经将部分，事实上可能是全部资源添加到了应用缓存中。因此，现在的问题是：该缓存清单在上一次访问后有没有发生变动？

   - 如果未发生变动，浏览器就只会立即触发一个 noupdate 事件。
   - 如果发生了变动，浏览器就会触发 downloading 事件，并开始将缓存清单中的资源重新下载一遍。

在下载的资源过程中，浏览器会阶段性地触发 process 事件，通过该事件可以获取当前下载进度信息（当前已经下载了多少文件，以及还有多少文件还在下载队列中）。

当所有的资源都成功重新下载完毕后，浏览器会触发最后一个 updateready 事件。表示当前 Web 应用所有资源已经下载更新完毕，准备好在离线状态下使用了。要想使用更新后的资源，同时又能实现“热切换”（不需要用户强制刷新浏览器），可以手动去调用 window.applicationCache.swapCache()方法。

如果过程中发生了错误，浏览器就会触发 error 事件并停止。下面列出了有可能会发生的错误：

- 访问缓存清单文件时发生 HTTP 404（找不到页面）或者 HTTP 410（永远不可用）错误。
- 访问到缓存清单文件，同时该文件也未发生变动，但是指向该文件的 HTML 页面下载失败。
- 访问到缓存清单文件，该文件发生变动。但是在无法下载缓存清单中列出的资源。

8.4 调试的艺术——杀了我！现在就杀了我！

The Fine Art of Debugging, a.k.a. "Kill Me! Kill Me Now!"

这里我想强调两点。第一点，一旦缓存清单文件中列出的某个资源下载失败，整个缓存过程都会失败。尽管浏览器会触发 error 事件，但是没有任何信息提示失败的原因。这就使调试变得异常困难。第二点，有些情况下其实并不是错误而更像是浏览器的 bug。这与浏览器如何检测缓存清单文件是否发生变动有关。此过程分为三步，尽管有些枯燥，但却非常重要，因此要予以重视。下面是该过程的具体内容：

1. 浏览器通过普通的 HTTP 语义来检测缓存清单文件是否已过期，就和检测其他通过 HTTP 请求的文件一样。Web 服务器会在响应头中包含描述该文件过期时间的信息。这样浏览器就可以通过头信息（Expires 或者 Cache-Control）得知该文件可以被缓存多久、什么时候过期，而不需要再去“询问”服务器。这种方式的缓存并不是为离线 Web 应用专门提供，而是所有的 HTML 页面、样式文件、脚本文件、图片以及其他 Web 资源都会使用这种方式。

2. 如果缓存清单文件已经过期（通过 HTTP 头信息），那么浏览器“询问”服务器是否有新的版本，有的话就会去把它下载下来。如何实现这点呢？浏览器会向服务器发送一个 HTTP 请求，该请求的请求头中包含了缓存清单文件最后一次修改时间的信息。（此信息是上一次浏览器下载该文件的时候，服务器通过 HTTP 响应头“告诉”浏览器的。）接下来服务器获取到最后一次修改时间，再和服务器上当前最新缓存文件的最后修改时间做个对比，计算出该文件是否发生了改动。如果未发生改动，那就简单地返回一个 304（没有修改）的状态码。再重申一次，这些处理方式都不是专门为离线 Web 应用提供的，而是所有的 Web 资源都是这么处理的。

3. 如果服务器计算出缓存清单文件发生了改动，那会返回一个 HTTP 200（完成）状态码，同时会将新文件内容信息以及新的 Cache-Control 和最后一次修改时间 HTTP 响应头信息，返回给浏览器。这些信息确保了下一次上述第一步和第二步能够正常工作。（HTTP 协议很酷；Web 服务器也总是很有远见：当 Web 服务器不得不向浏览器发送一个文件的时候，它总是想尽办法避免同样的情况下再次发送该文件。）下载完最新的缓存清单文件后，浏览器会去读取里面的内容，如果内容和前一个版本一致，没有变化，就不会去重新下载清单文件中的资源。

在开发和测试离线 Web 应用时，一定要对上述三步中任何一步的细节都非常了解，否则就会陷入困境。举个例子，假设在部署完缓存清单文件十分钟后，突然想要在该文件中再多加资源，没错，很简单，只需将该资源对应的 url 加到缓存清单中，然后再重新部署就行了。但是，这个时候问题来了：当用户再次访问该页面的时候，浏览器发现 manifest 属性，然后触发 checking 事件，然后……就结束了。浏览器会“坚信”缓存清单文件没有发生改动。为什么呢？因为 Web 服务器会默认将静态资源的过期时间设置成几个小时（通过 Cache-Control 头信息）。这就意味着浏览器永远不会到达上述过程的第二步。尽管 Web

服务器很清楚缓存清单文件已经发生改动，但是过期时间被设置成了几个小时，因此十分钟后，浏览器发现该文件并没有过期，因此根本不会去“询问”服务器是否有新的版本。

针对上述的情况，你就非常难调试了，因为很有可能你在几个小时内重新刷新了无数次的页面，结果由于页面还没有过期，浏览器还是不会去获取最新版本的缓存清单文件，你还是毫无头绪。这其实并不是一个 bug，而是特性如此。每个“角色”都各司其职，很正常。

那么如何才能避免上述情况呢？这就需要重新配置 Web 服务器，让缓存清单文件不用 HTTP 头信息的方式来实现缓存。对于基于 Apache 的服务器，在.htaccess 配置文件中加入如下两行就可以了：

```
ExpiresActive On
ExpiresDefault "access"
```

但是这样配置就意味着.htaccess 文件所在的目录以及子目录都不会以 HTTP 头信息的方式来实现缓存，这显然不是你想要的。因此需要指定<Files>标识，使其只作用于缓存清单文件或者创建一个单独的目录，然后只将.htaccess 文件和缓存清单文件放置在该子目录下。不同的浏览器配置不同，对于其他服务器需要查看对应的配置文档。

但是，禁止使用 HTTP 头信息方式来实现对缓存清单文件的缓存并没有解决全部问题。在这样的情况下：你改变了其中一个已经被缓存到应用缓存中的资源文件，但是该资源文件的 URL 并没有变，因此缓存资源文件并不会发生改动。这个时候浏览器根本不会知道其中的一个资源已经发生了改动。比如下面的例子：

```
CACHE MANIFEST
# rev 42
clock.js
clock.css
```

假设修改了 clock.css 并且也重新部署了，但还是不会发生任何变化。因为缓存清单文件并没有发生改动。所以，每次对资源文件进行修改后都要手动去修改缓存清单文件，哪怕只是改动一个字符。最简单的方式是：在缓存清单文件中加上一行注释来表明当前的版本，这样每次修改了资源文件后只需要修改版本号就可以了。这样，浏览器就会知道缓存清单文件发生了改动，然后就通过上述三步去重新下载所有缓存清单文件中的资源：

```
CACHE MANIFEST
# rev 43
```

```
clock.js
clock.css
```

8.5 让我们来构建一个离线 Web 应用！
Let's Build One!

还记得在第 4 章（参见第 75 页“一个完整的例子”一节）介绍的，后来通过持久化本地存储实现状态保存（参见第 132 页“HTML5 存储实践”一节）的 Halma 游戏吗？接下来，我们让 Halma 游戏实现在离线状态下依然可以玩。

首先，我们需要一个缓存清单文件将所有游戏所需的资源都包含进来。这些资源会是：一个 HTML 页面、一个 JavaScript 文件，包含了所有的实现游戏的代码以及……这里不需要图片，因为这些都是利用画布 API（参见第 4 章）通过编程的方式实现的；这里也不需要 CSS 文件，因为所有必须的样式都利用<style>元素内联在 HTML 页面了。这样，我们的缓存清单文件就包含了如下信息：

```
CACHE MANIFEST
halma.html
../halma-localstorage.js
```

对于目录结构有必要说明如下。我在 examples/目录下创建了 offline/子目录，该缓存清单文件就在此子目录下。因为 HTML 页面需要一个副本才能离线工作（稍后详细讨论），我又复制了一个 halma.html 页面，也将它放在了此子目录下，作为离线版本的页面。但是因为我们增加了本地存储支持（参见第 132 页“HTML5 存储实践”），Java Script 代码本身没有变化，所以我直接重用了在 examples/目录下的.js 文件。上面提到的这些文件的目录结构大致如下：

```
/examples/localstorage-halma.html
/examples/halma-localstorage.js
/examples/offline/halma.manifest
/examples/offline/halma.html
```

在缓存清单文件中（/examples/offline/halma.manifest），我们想要引用两个文件：一个是离线版本的 HTML 页面（/examples/offline/halma.html），因为该文件和缓存清单文件在同一目录下，因此只需要将该 HTML 页面的文件名加进来就可以了；另一个是 JavaScript，该文件在缓存清单文件所在目录的上级目录中（/examples/halma-localstorage.js），因此需要在文件名前加上指向上级目录的相对路径：../，就和使用相对路径设置<img src>属性一样。当然，也可以用绝对路径（从当前应用的根目录开始），甚至是绝对 URL（指向其他域下的资源）。

现在，我们需要将 HTML 页面的 manifest 属性设置成指向缓存清单文件：

```
<!DOCTYPE html>
<htm lang="en" manifest="halma.manifest">
```

大功告成！当支持 HTML5 离线应用的浏览器去加载该 HTML 页面时，会去下载对应的缓存清单文件，然后开始下载所有缓存清单文件中包含的资源，把它们保存到离线应用缓存中。至此以后，无论什么时候再去请求该 HTML 页面都会直接从应用缓存中获取。这样你就可以在离线状态下玩 Halma 游戏了，同时由于该游戏支持本地存储游戏状态，你可以随意地离开和进入游戏。

8.6 扩展阅读
Further Reading

标准文档：

- HTML5 离线 Web 应用标准文档（*http://bit.ly/cCkWZa*）。

浏览器提供商提供的文档：

- Mozilla 开发者中心的“Firefox 中的离线资源”（*https://developer.mozilla.org/En/Offline_resources_in_Firefox*）。
- “HTML5 离线应用缓存”（*http://developer.apple.com/library/safari/#documentation/iPhone/Conceptual/SafariJSDatabaseGuide/OfflineApplicationCache/OfflineApplicationCache.html*），“Safari 客户端存储及离线应用编程教程”（*http://developer.apple.com/library/safari/#documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Introduction/Introduction.html*）的一部分。

教程及其例子：

- “基于 HTML5 的移动版 Gmail 系列：使用离线应用缓存构建离线 Gmail——第一部分”（*http://googlecode.blogspot.com/2009/04/gmail-for-mobile-html5-series-using.html*）。
- “基于 HTML5 的移动版 Gmail 系列：使用离线应用缓存构建离线 Gmail——第二部分”（*http://googlecodė.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-2.html*）。
- “基于 HTML5 的移动版 Gmail 系列：使用离线应用缓存构建离线 Gmail——第三部分”（*http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-3.html*）。
- Jonathan Stark 的“调试 HTML5 离线应用缓存”（*http://jonathanstark.com/blog/2009/09/27/debugging-html-5-offline-application-cache/*）。
- Paul Rouget 的“一个基于 HTML5 的离线图片上传和编辑应用”（*http://hacks.mozilla.org/2010/02/an-html5-offline-image-editor-and-uploader-application/*）。

第 9 章

疯狂的表单

A Form of Madness

9.1 引言

Diving In

想必你对如何构建一个 Web 表单一定了如指掌：首先创建一个<form>元素，然后创建一些诸如：<input type="text">、<input type="password">这样的元素，最后再以一个<input type="submit">按钮结束整个表单的构建。

但是你对 HTML5 表单的构建一定知之甚少，因为 HTML5 标准中定义了许多新的输入框类型可以用在表单中。这里“用”的意思是“现在”就可以用了。但是不要高兴得太早，并不是所有的浏览器都支持全部这些新的输入框类型。主流浏览器全部支持；而传统浏览器对少部分输入框类型不支持。也就是说，通过对传统浏览器采用优雅降级的方式基本可以实现让所有的浏览器（包括 IE6）支持这些新的输入框类型。

9.2 占位文本

Placeholder Text

HTML5 标准中对 Web 表单做了些改进。其中一项改进就是允许输入框设置占位文本。在给输入框设置占位文本后，占位文本就会在输入框为空或者失去焦点的时候显示出来，而一旦用户点击了输入框（或者利用 Tab 键使其获得焦点），占位文本就会消失。

或许你对占位文本并不陌生。如图 9.1 所示，Mozilla Firefox 3.5 在地址栏中设置了“在书签和历史记录中查找”这样一段占位文本。

图 9.1 Firefox 地址栏的占位文本

当通过鼠标点击（或者按 Tab 键）使地址栏获得焦点时，该占位文本就会消失。如图 9.2 所示：

图 9.2　占位文本在地址栏获得焦点时就消失了

遗憾的是，Firefox 3.5 并不支持在自定义 Web 表单中设置占位文本。表 9.1 列出了浏览器对占位文本的支持情况。

表 9.1　浏览器对占位文本的支持情况

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	3.7+	4.0+	4.0+	.	.	.

如下代码展示了如何在 Web 表单中设置占位文本：

```
<form>
  <input name="q" placeholder="Search Bookmarks and History">
  <input type="submit" value="Search">
</form>
```

对于不支持占位文本的浏览器会忽略上述 placeholder 属性。

专家答疑

问：placeholder 属性值支持 HTML 标签吗？比如我想插入一张图片或者改变占位文本的颜色？

答：不可以。placeholder 属性值只支持纯文本。但是有些浏览器提供了专属的 CSS 扩展（*http://trac.webkit.org/export/37527/trunk/LayoutTests/fast/forms/placeholder-pseudo-style.html*），可以允许改变占位文本的颜色。

9.3　自动聚焦

Autofocus Fields

很多网站使用 JavaScript 脚本让 Web 表单的首个输入框自动获得焦点，例如 Google.com 首页会让搜索框自动获取焦点以便于直接输入搜索关键字。这对于多数普通用户来说的确很方便，但是对于那些高级用户或者有特殊需求的用户来说却未必如此。比如你原想按下空格键来滚动页面，但是因为焦点已经被聚到了表单中的输入框，因此页面并不会如你所愿地滚动（而会在输入框中输入空格）；再比如当页面还在加载的时候把焦点聚焦到了其

他输入框，而正当你想要进行输入操作的时候，脚本却又将焦点自动聚焦到了原本设定自动对焦的输入框中，此时你输入的文字就不会出现在预想的输入框中。

因为自动聚焦是用 JavaScript 脚本来实现的，因此可以使用一些小技巧来处理所有这些极端的情况，但这样却无法满足那些不想要网页自动聚焦的用户。

为了解决这个问题，HTML5 标准为所有的表单控件都引入了一个 autofocus 属性。顾名思义，这个属性的意思就是将焦点自动聚焦到特定的输入框中。由于直接采用标记而不是 JavaScript 实现了这一功能，因此所有网站对于自动聚焦的实现都是一致的。浏览器提供商（或者扩展程序作者）也可以提供用户一个禁用该功能的方法。表 9.2 展示了不同浏览器对自动聚焦的支持情况。

表 9.2　浏览器对自动聚焦的支持情况

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
.	.	4.0+	3.0+	10.0+	√	√

如下代码展示了如何为一个表单字段设置自动聚焦：

```
<form>
  <input name="q" autofocus>
  <input type="submit" value="Search">
</form>
```

不支持自动聚焦的浏览器会忽略上述 autofocus 属性。

如何才能让所有的浏览器都实现自动聚焦功能呢？其实很简单，对于不支持自动聚焦的浏览器就采用 JavaScript 脚本的方式来实现。因此只要完成以下两步就能让所有浏览器实现自动聚焦：

- 对想要自动聚焦的元素对应的 HTML 标签上加上 autofocus 属性。
- 检测浏览器是否原生支持 autofocus 属性（参见第 27 页“表单自动聚焦”一节）。如果不支持，就运行实现自动聚焦的 JavaScript 脚本：

```
<form name="f">
  <input id="q" autofocus>
  <script>
    if (!("autofocus" in document.createElement("input"))) {
      document.getElementById("q").focus();
    }
    </script>
    <input type="submit" value="Go">
</form>
...
```

可以参看一个按照如上两个步骤让所有浏览器实现 autofocus 的例子：*http://diveintohtml5.org/examples/input-autofocus-with-fallback.html*。

专家发言

很多实现自动聚焦的脚本都是等到页面载入完成触发 window.onload 事件后执行的。但是 window.onload 事件是直到页面上所有的图片都载入之后才会被触发。因此，对于有大量图片的页面，就会出现用户在页面载入过程中就手动聚焦在页面某个位置，而之后等到页面载入完毕 window.onload 事件被触发后，脚本又把焦点聚集到了原本设置自动聚焦的位置（这就是为什么高级用户很讨厌实现自动聚焦的脚本的原因）。考虑到这点，我们之前的例子是采用将脚本直接插入在要自动聚焦的表单字段后面，这样该表单字段渲染出来后即会执行该脚本。但是如果由于某些原因无法实现这样将脚本插在页面中间的话，应当将脚本写在类似于 jQuery 的$(document).ready()这样的自定义事件中而不是 window.onload 事件中。

9.4 Email 地址
Email Addresses

在 HTML5 之前，Web 表单只支持少数输入框类型——表 9.3 中列出了其中常用的。

表 9.3 HTML4 中定义的输入框类型

类型	HTML 代码	备注
复选框	<input type="checkbox">	可以被选中和撤销
单选按钮	<input type="radio">	可以和其他类型输入框组成构成一组
密码输入框	<input type="password">	输入的内容会以“点”的形式展现
下拉菜单	<select><option>...	
文件选择器	<input type="file">	弹出一个“打开文件”对话框
提交按钮	<input type="submit">	
纯文本输入框	<input type="text">	type 属性可以忽略

上述所有的输入框类型在 HTML5 表单中仍然可用。这就意味着，即使在“升级到用 HTML5 的标准来渲染页面”（可以通过修改文档类型，参见第 31 页“文档类型”一节）的情况下，它们仍然能够很好地工作，而不需要对其做任何改动。

不过，HTML5 标准中定义了很多新的输入框类型，而且现在都可以使用它们了。

其中一个就是 email 地址类型的输入框。如下所示：

```
<form>
  <input type="email">
  <input type="submit" value="Go">
</form>
```

原本我想说“对于不支持 type="email"的浏览器……”。但我没说，为什么呢？因为我自己都不确定怎么样叫“浏览器不支持 type="email"”。事实上，所有的浏览器都“支持”type="email"。只不过有些浏览器对其作了特殊处理罢了（后面会给大家展示相应的

例子）。但是对于无法识别 type="email"的浏览器会以 type="text"来处理并将其渲染成纯文本类型输入框。

我没法说这有多么的重要。目前许多 Web 表单都是采用<input type="text">这类纯文本输入框来作为 email 地址的输入框。如今，HTML5 标准中专门定义了 type="email"——用于 email 地址的输入框类型。因此，对于要用作 email 地址的输入框可以“升级”采用 type="email"来实现了。因为所有不支持未知 type 属性的浏览器都会以 type="text"来处理，包括 IE6 也不例外。

那么怎样算是浏览器支持 type="email"呢？这绝非只字片语能够说明白，且听我慢慢道来。HTML5 标准中并没有对新类型的输入框样式做强制的规定。Opera 浏览器会将 email 地址类型的输入框渲染成：前面带有小的表示 email 的图标；而其他诸如 Safari 和 Chrome 这样的浏览器则是简单地将其渲染成与纯文本类型输入框完全一样（从样式上完全看不出区别，除非看代码）。

接着说说 iPhone。

iPhone 并未提供物理键盘，所有的“输入”操作都是通过屏幕键盘来完成的。该屏幕键盘会在适当的时候（如表单输入框获得焦点时）弹出来。苹果将 iPhone 上的浏览器做得很智能，一旦浏览器识别了 HTML5 新的输入框类型，就会根据其类型相应地改变屏幕键盘来方便用户输入。

举个例子来说：email 地址其实也是段文本，这毫无疑问。但是它是一段特殊的文本：所有的 email 地址都包含一个“@”符号、至少一个“.”同时不能有任何空格。因此，当 iPhone 下<input type="email">输入框获得焦点时，弹出的屏幕键盘中会有一个比一般情况下小的空格键，同时旁边有一个“@”键和“.”键。如图 9.3 所示。

总的来说，现在将所有用于输入 email 地址的输入框都用 type="email"实现是没有任何坏处的。而且用 type="email"实现，用户根本不会察觉，除了 iPhone 用户（当然也有可能察觉不到）。不过就算察觉，他们也会很满意这样的用户体验的。

9.5 Web 地址

Web Addresses

网址：大家所熟知的 URL，也可以说是 URI——另外一类特殊的文本。相关互联网标准中定义了网址的语法格式。就比如表单要求输入一个网址，那你肯定会输入类似 *http://www.google.com* 而绝对不会输入“125 Farwood Road”。网址中斜杠（“/”）和点（“.”）是很常见的，但是不允许空格。另外，所有的网址都会有诸如“.com”或者“.org”这样的域名后缀。

<input type="url">表示网址类型的输入框。该输入框在 iPhone 下浏览器的样式如图 9.4 所示：

图 9.3　输入 email 地址时的屏幕键盘

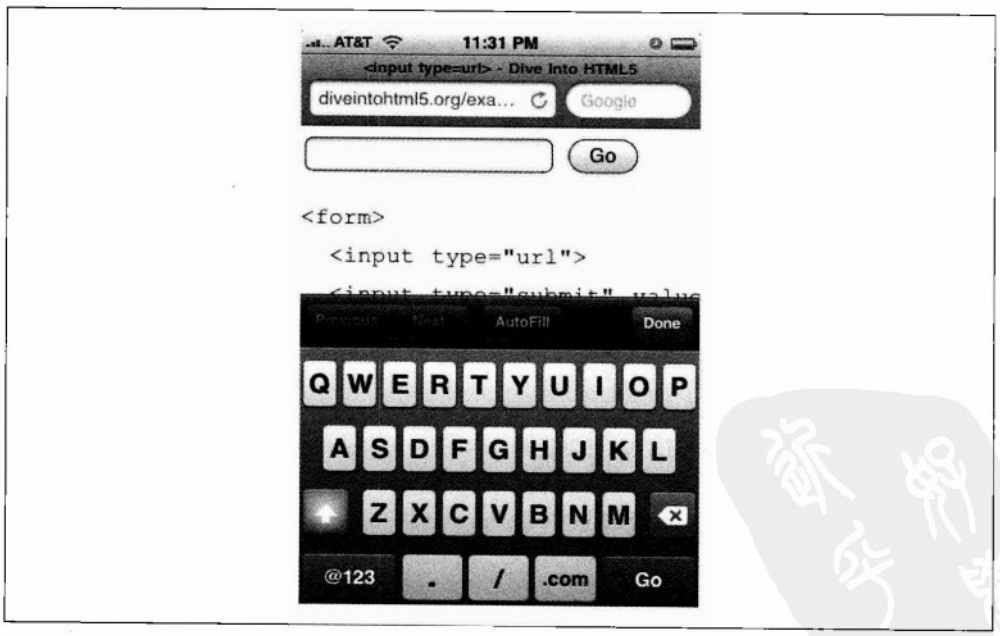


图 9.4　输入网址时候的屏幕键盘

和针对 email 地址类型输入框一样，iPhone 针对网址类型输入框对屏幕键盘也做了相应的优化。如图中：可隔间已经消失了，取而代之的是一个点（“.”）、斜杠（“/”）和“.com”按钮。其中长按“.com”按钮会出现常用的诸如“.org”或者“.net”这样的域名后缀按钮。

对于不支持 type="url"的浏览器会以 type="text"来处理。因此，采用 HTML5 中的网址类型的输入框是没有任何坏处的。

9.6 数字类型输入框：数字选择器

Numbers As Spinboxes

要求用户输入数字要比输入 email 地址或者网址的情况麻烦得多。比如，是要求用户输入具体的-1 呢？还是 1~10 范围内的数字，是要求输入 7 呢？还是小数又或者是 π……怎么样，够麻烦吧！

实际上，在我看来，大多数情况下都是要求用户输入一定范围内的某个数字——比如：十进制的整数或者是 10 的倍数。HTML5 标准中就定义了这样的类型的输入框，如下所示：

```
<input type="number"
       min="0"
       max="10"
       step="2"
       value="6">
```

让我们来依次看下该 input 元素中的属性（同时，你也可以亲自去写个例子实践一下）。

- type="number"，表示这是个数字类型的输入框。
- min="0"，表示输入的数字不能小于 0。
- max="10"，表示输入的数字不能大于 10。
- step="2"，结合 min 的值，定义了选择范围在 0、2、4……一直到最大值。
- value="6"，表示默认值是 6。和一般输入框的 value 属性是一样的（之所以提到这点旨在告诉大家 HTML5 标准的制定也是基于早期 HTML 标准的。因此，它完全兼容早期 HTML 标准中定义的内容）。

这就是数字类型输入框。要记住的是：所有的属性都是可选的。如果要求用户输入的数字区间有下限但没有上限，那就可以只设置 min 属性，而不设置 max 属性；step 属性默认值为 1，因此如果不想改变这个默认值的话就直接忽略该属性；value 属性没有默认值，该属性值可以为空或者可以忽略。

HTML5 标准中还提供了相应的方法来操控该数字类型输入框控件，如下所示：

`input.stepUp(n)`

将该输入框内的值加上 n

`input.stepDown(n)`

将该输入框内的值减去 n

`input.valueAsNumber`

以浮点数类型返回输入框内的值（默认该值为字符串类型）

同样的，对于数字类型输入框的样式也是因浏览器而异的。在 iPhone 上，数字类型输入框获得焦点时，浏览器就会对屏幕键盘做相应的优化，如图 9.5 所示：

图 9.5　输入数字时的屏幕键盘

而桌面版的 Opera 浏览器则会把数字类型输入框渲染成数字选择器。该数字选择器有“向上”和“向下”的小箭头，点击这两个小箭头可以改变输入框中的数值，如图 9.6 所示：

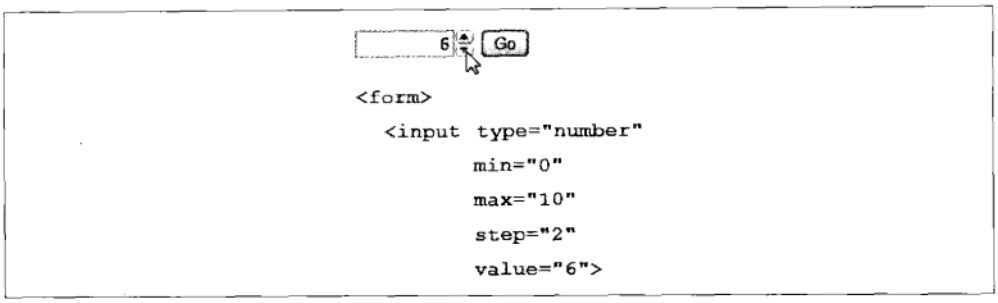


图 9.6　数字选择器

Opera 要求数字类型输入框必须设置 min、max 及 step 属性。这样能保证总能获取到有效的数字值：当输入框中的值达到所设的最大值时，数字选择器中“向上”的小箭头就会变成灰色不可用状态，如图 9.7 所示：

图 9.7　达到最大值时的数字选择器

与针对本章前面所介绍的其他类型的输入框一样，对于不支持 type="number"的浏览器会以 type="text"来处理。该输入框中的值仍然有效（因为是存储在 value 属性上的），而对于其他诸如 min、max 这样的属性就会忽略。当然，你可以手动实现一个数字选择器，也可以重用许多 JavaScript 框架中提供的数字选择器。使用如下代码可以检测浏览器是否原生支持 HTML5 中数字类型输入框（参见第 25 页“输入框类型”一节）：

```
if (!Modernizr.inputtypes.number) {
   // 对于 type="number"没有原生支持
   // 或许可以尝试 Dojo 或其他 JavaScript 框架
}
```

9.7　数字类型输入框：滑块
Numbers As Sliders

上一节介绍的数字选择器并非数字类型输入框的唯一展现形式。如图 9.8 所示是另一种滑块的展现形式：

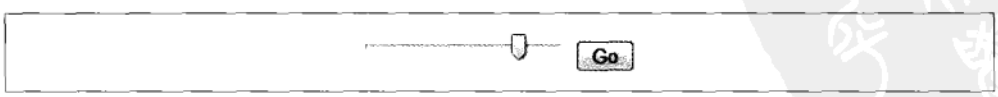


图 9.8　滑块

现在就可以在 Web 表单中使用滑块了。如下所示，其代码和数字选择器非常相似：

```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6">
```

其中 min、max、step、value 这些属性和数字选择器中的是一模一样的。唯一的不同在于展现形式上。支持 type="range"的浏览器会将其渲染成一个滑块而不是输入框。截止到本书撰写时，最新版的 Safari、Chrome 和 Opera 浏览器都已支持 type="range"的输入框并会将其渲染成滑块的样式（遗憾的是，iPhone 上只会将其渲染成简单的纯文本输入框，甚至不对屏幕键盘做任何相应的优化）。而对于其他不支持 type="range"的浏览器会以 type="text"来处理。所以，还等什么呢？现在就可以放心大胆地使用 type="range"吧！

9.8 日期选择器

Date Pickers

HTML4 中并未提供日期选择器控件，许多诸如 Dojo、jQuery UI、YUI 及 Closure Library 这样的 JavaScript 框架提供了日期选择器控件。但是，要使用其日期选择器控件就不得不使用对应的框架。

HTML5 最终定义了一种方式可以使用不包含任何 JavaScript 脚本的原生日期选择器控件。事实上，相类似的有六种选择器控件：日期、月、星期、时间、日期+时间、日期+时间-时区。如表 9.4 所示，目前只有 Opera 支持这些控件：

表 9.4 浏览器对日期选择器的支持情况

输入框类型	Opera 浏览器	其他浏览器
type="date"	9.0+	.
type="month"	9.0+	.
type="week"	9.0+	.
type="time"	9.0+	.
type="datetime"	9.0+	.
type="datetime-local"	9.0+	.

<input type="date">表示日期选择器，其在 Opera 浏览器下展现形式如图 9.9 所示。

<input type="datetime">表示日期和时间选择器，其在 Opera 浏览器下展现形式如图 9.10 所示。

<input type="month">表示只选择年份和月份（一般用于选择信用卡过期时间）。其在 Opera 浏览器下展现形式如图 9.11 所示。

图 9.9 日期选择器

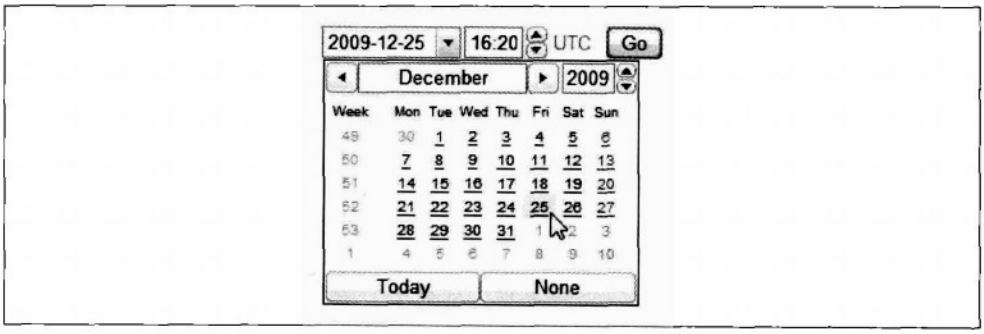


图 9.10 日期 / 时间选择器

图 9.11 月份选择器

<input type="week">表示只选择年份和星期，一般不常用。其在 Opera 浏览器下的展现形式如图 9.12 所示。

最后是<input type="time">表示选择时间，其在 Opera 浏览器下的展现形式如图 9.13 所示。

图 9.12　星期选择器

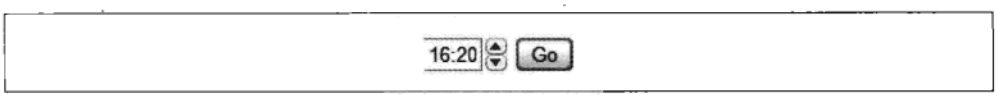


图 9.13　时间选择器

与之前提到的 type="email"（参见第 150 页“Email 地址”一节）及其他输入框类型一样，对于不支持 type="date"及相关类型的浏览器都会以 type="text"来处理。相信最终其他浏览器会支持这些输入框类型。所以现在也可以使用<input type="date">等类型输入框，来提高 Opera 用户的体验，而同时对其他浏览器进行检测（参见第 25 页“输入框类型”一节），如果不支持，就采用 JavaScript 脚本的解决方案：

```
<form>
  <input type="date">
</form>
...
<script>
  var i = document.createElement("input");
  i.setAttribute("type", "date");
  if (i.type == "text") {
    // 无原生日期选择器支持 :(
    // 使用 Dojo/jQueryUI/YUI/Closure 或其他方式创建一个，然后动态替换<input>元素。
  }
</script>
```

9.9　搜索框
Search Boxes

搜索框有些微妙。其想法很简单，但是实现起来并不容易，情况如下。

这里所指的搜索框并非仅仅是 Google 搜索的搜索框或者雅虎搜索的搜索框而是任意网站，任意页面上的任意一个搜索框。亚马逊网站上有搜索框、新蛋网站上有搜索框、大多数博客站点上也有搜索框，它们是怎么实现的呢？目前大多采用<input type="text">的方式，与纯文本类型输入框一样。但是在 HTML5 中提供了专门用于搜索框的输入框类型！

```
<form>
  <input name="q" type="search">
  <input type="submit" value="Find">
</form>
```

其样式在部分浏览器上还是和一般的纯文本类型输入框没什么区别，但在 Mac OS X 操作系统下的 Safari 浏览器中的样式却不同，如图 9.14 所示：

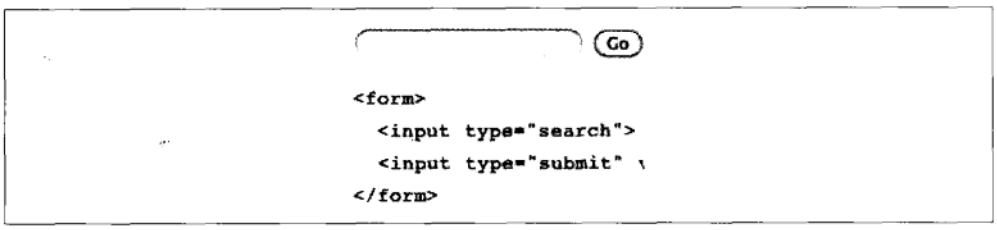


图 9.14　搜索框

能看出区别吗？没错，该输入框是圆角的。但是这并没有令人兴奋的地方。最关键的是，当开始在搜索框中输入内容时，在搜索框的右侧就会出现一个小“x”按钮，点击该按钮会清空输入框中的内容（与 Safari 采用同一内核的 Google Chrome 浏览器也是如此）。Mac OS X 中 Safari 之所以对搜索框这样处理是为了和其他 Mac OS X 客户端应用中对搜索框的处理方式保持统一（如图 9.15 所示）。

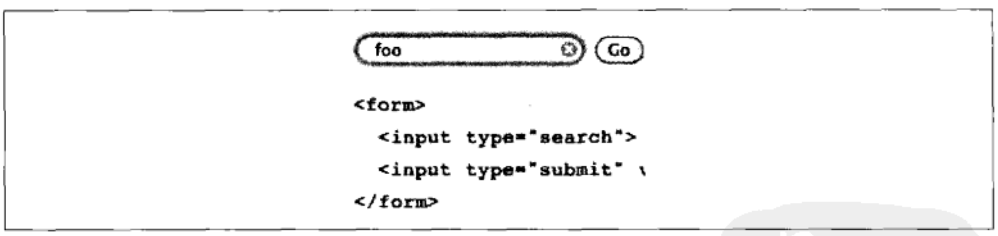


图 9.15　获取焦点后的搜索框

苹果网站使用<input type="search">来实现搜索框以呈现一种“苹果风”的感觉。但是 HTML5 中定义的搜索框并非是 Mac 专属的。不同平台的不同浏览器可以根据平台的具体情况自定义搜索框的展现形式。同样地，对于不支持 type="search"的浏览器会以 type="text"来处理。因此现在也可以大胆放心地使用<input type="search">了。

专家发言

有一个理由可能会让你放弃使用<input type="search">，那就是 Safari 不允许通过常用 CSS 来控制搜索框的样式（我所说的“常用”是指边框、背景色、背景图片、内间距等）。但是你可以得到圆角！

9.10 颜色选择器
Color Pickers

HTML5 中还定义了<input tpe="color">作为颜色选择器，允许你拾取颜色并返回其 16 进制表现形式。但是遗憾的是，目前还没有浏览器支持，因为我很喜欢 Mac OS 中的颜色拾取工具。或许以后会有。

9.11 还有一点……
And One More Thing...

本章中，我已经给大家介绍了 HTML5 中新的输入框类型及诸如自动聚焦这样新的表单特性。接下来我给大家介绍应该是 HTML5 表单中最酷的特性：自动输入验证。通常对用户在表单中填写的比如 email 地址，都会采用 JavaScript 在前端对其合法性进行验证，同时在后台也会采用诸如 PHP、Python 或者其他脚本对其进行验证。其中利用 JavaScript 在前端进行 email 地址合法性验证存在两个很大的问题：

- 有惊人数量的用户（大约 10%左右）将其浏览器设置成禁用 JavaScript 脚本。
- 这样做会出错。

这样做真的很容易出错。验证一串未知的随机字符是否是一个合法的 email 地址是件很复杂的事情（*http://www.regular-expressions.info/email.html*）。越复杂的字符串就越难验证（*http://www.ex-parrot.com/pdw/Mail-RFC822-Address.html*）。这真的非常复杂（*http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx*），而且非常容易使得浏览器崩溃。

图 9.16 所示的是一张在 Opera 10 浏览器中的截图。尽管从 Opera 9 开始就已经支持“自动输入验证”的特性。要实现该特性只需将输入框的 type 属性设置成“email”就可以了（参见第 150 页“Email 地址”一节）。当 Opera 用户提交一个带有<input type="email">的表单时，Opera 会自动对 email 地址采用 RFC 定义的合法性进行检测。即使在 JavaScript 脚本被禁用的情况下也会进行自动验证。

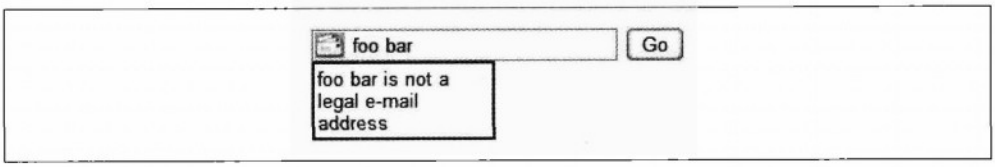


图 9.16 Opera 验证 type="email"

Opera 还提供了对网址输入框(<input type="url">)中网址及数字类型输入框(<input type="number">)中数字的自动验证。对数字的验证同时还包括 min 和 max 属性验证。如果表单中数字类型输入框中输入的数字大于设置的最大值,Opera 就会禁止用户提交该表单(如图 9.17 所示)。

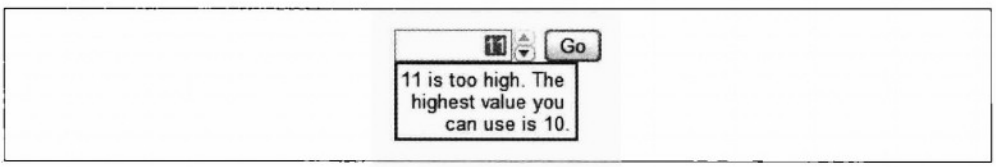


图 9.17 Opera 验证 type="number"

不幸的是,其他浏览器目前还不支持 HTML5 表单自动验证。因此目前只能使用 JavaScript 脚本来实现。

9.12 扩展阅读
Further Reading

标准文档:

- <input>类型 (*http://bit.ly/akweH4*)。
- <input placeholder>属性 (*http://bit.ly/caGl8N*)。
- <input autofocus>属性 (*http://bit.ly/db1Fj4*)。

JavaScript 库:

- Modernizr (*http://www.modernizr.com*),一个 HTML5 检测库。

PDG

第 10 章

“分布式”、“可扩展性”及其他华丽词藻

"Distributed," "Extensibility," and Other Fancy Words

10.1 引言

Diving In

HTML5 包含超过 100 个元素（*http://simon.html5.org/html5-elements*）。有些是纯粹的语义元素（参见第 3 章），有些则是脚本化 API 的容器（参见第 4 章）。纵观 HTML 的历史（参见第 1 章），那些制定标准的学究们一直在争论哪些元素应当被放进这个语言里。HTML 应该包含<figure>元素？<person>元素？还是<rant>元素呢？最后做出了决定，写出了规范，给出了实现，Web 就这样不断向前发展。

当然，HTML 无法取悦所有人。没有任何标准可以做到这一点。一些想法没能入选标准。例如，HTML5 中并没有<person>元素。（也没有<rant>元素，该死的！）你当然可以不受限制地在页面里写一个<person>，但这样做会使页面不能通过合法性校验，也无法保证跨浏览器的正确渲染（参见第 42 页“题外话：浏览器如何处理未知元素”），并且它还可能与今后新的 HTML 规范产生冲突。

因此，如果自行添加元素不是答案，那什么是 Web 开发者从语义方面倾向于做的呢？曾经有人试图扩展以前版本的 HTML。最流行的方法是使用微格式（microformats，*http://microformats.org*），它利用了 HTML 4 中的 class 和 rel 属性。另一种选择是 RDFa（*http://www.w3.org/TR/rdfa-syntax/*），它原本是设计为在 XHTML 中使用（参见第 14 页“后记”一节），但现在也被移植到了 HTML 中。

微格式和 RDFa 各有自己的优缺点。它们采取完全不同的做法来实现同样的目标：通过额外的、并不属于核心 HTML 的一部分语义成分来扩展 Web 页面。我并不想在这一章里搞格式大战。（这样的话<rant>元素必不可少！）相反，我希望把重点放在第三个选项，即本身是 HTML5 的一部分，并且紧密集成于其中的：微数据（microdata）。

10.2 什么是微数据？

What Is Microdata?

下面句子中的每个字都很重要，所以要看仔细。

> **专家发言**
>
> *微数据使用来自自定义词汇表的、带作用域的名/值对来给 DOM 做标记。*

这是什么意思呢？让我们从后往前看。微数据围绕着自定义词汇表（custom vocabularies）做文章。可以将“HTML5 中所有元素的集合”看作一个词汇表。这个词汇表包括可以代表章节或者文章的元素（参见第 41 页“HTML5 中新增的语义元素”），但不包括可以代表一个人或者活动的元素。如果你想在网页上表示一个“人”，你就需要定义自己的词汇表。微数据可以让你做到。任何人都可以定义一个微数据词汇表，然后就可以开始在自己的网页里嵌入自定义的属性。

接下来要了解的关于微数据的事实就是，它以名/值对的形式运作。每一个微数据词汇表都定义了一组命名属性。例如，一个“人”的词汇表可以定义类似姓名和照片这样的属性。要在网页上包含一个特定的微数据属性，你需要将其属性名置于特定的地方。根据你声明属性名的位置，微数据会有提取属性值的规则。（关于这方面详见下一节。）

除了命名属性，微数据还很大程度地依赖于“作用域”的概念。理解微数据作用域的最简单方式就是参照 DOM 中的元素之间的自然父子关系。<html>元素（参见第 33 页“根元素”）通常包含两个子元素，<head>（参见第 34 页“<head>元素”）和<body>。<body>元素通常包括多个子元素，其中每个也可以再有自己的子元素。例如，页面可能含有一个<h1>元素，它位于<hgroup>元素里，后者又位于<head>元素中（参见第 45 页“页头”），<head>自己又位于<body>元素内。同样，一个数据表格含有<td>元素，它们位于<tr>元素内，<tr>又位于<table>元素内（再位于<body>之内）。微数据重用了 DOM 本身的层级结构，提供了一种方式来表达“所有该元素的属性都来自这个词汇表”。这允许在同一个页面上使用多个微数据词汇表。甚至可以将词汇表嵌套，完全和 DOM 结构的用法一样。（贯穿整章，我会演示多个嵌套词汇表的例子。）

我已经提到了 DOM，下面详细说明一下。微数据的本质是给那些已经在页面上可见的数据再施加额外的语义。微数据并非是被设计为一种独立的数据格式。它是 HTML 的一种补充。在下一节里你会看到，当已经在正确使用 HTML，但 HTML 的词汇表不太够用时，这时使用微数据就可以取得最好的效果。微数据最适合用于精细地优化那些已经在 DOM 里

的数据的语义性。如果数据还不在 DOM 里，就需要退一步，重新评估使用微数据是否是个正确方案。

明白“专家发言”里所说的话了吗？但愿如此。现在让我们来实战一番。

10.3 微数据的数据模型
The Microdata Data Model

定义自己的微数据词汇表很容易。首先，需要一个命名空间，它其实就是一个网址而已。命名空间 URL 可以指向一个实际存在的网页，但这并非严格要求。比方说，我想创建一个微数据词汇表来描述一个人。如果我拥有 *data-vocabulary.org* 这个域名，我就使用 *http://data-vocabulary.org/Person* 这个 URL 来作为我的微数据词汇表的命名空间。这是创建一个全局唯一标识符的简单方法：挑一个你自己拥有的域名。

在这个词汇表中，我需要定义一些命名属性。先从三个基本属性开始：

- `name`（用户的全名）
- `photo`（一个指向用户图片的链接）
- `url`（一个链接，指向和用户相关的网站，比如 Blog 或者 Google profile 页）

这三个属性中有两个是 URL，另外一个是纯文本。它们中的每个都可以对应一个自然的标记形式，甚至在你开始考虑微数据或者词汇表之前。想象一个个人资料页或“关于”页面。你的名字可能被标记为一个标题，例如一个`<h1>`元素。照片可能用一个`<img>`元素，因为你希望人们能看到它。此外与你的个人资料相关的任何 URL 可能都会被标记为超链接，以便人们可以点击查看。为了讨论方便，假设你的全部个人资料也都位于一个`<section>`元素内，以便同页面上的其他内容相分隔。因此：

```
<section>
  <h1>Mark Pilgrim</h1>
  <p><img src="http://www.example.com/photo.jpg" alt="[me smiling]"></p>
  <p><a href="http://diveintomark.org/">weblog</a></p>
</section>
```

微数据的数据模型就是名/值对。一个微数据的属性名（本例中的 `name`、`photo` 或 `url`）总是声明在一个 HTML 元素上。相对应的属性值则从 DOM 中取得。对于大多数 HTML 元素而言，属性值其实就是元素的文本内容。不过，也有极少数的例外，如表 10.1“微数据的属性值来自何处”所示。

表 10.1 微数据的属性值来自何处

元素	值
<meta>	content 属性
<audio>	src 属性
<embed>	
<iframe>	
<img>	
<source>	
<video>	
<a>	href 属性
<area>	
<link>	
<object>	data 属性
<time>	datetime 属性
所有其他元素	文本内容

“添加微数据”到网页就是给已有的那些 HTML 元素增添一些属性。第一件要做的事就是声明你使用的是哪个微数据词汇表，这可以通过添加一个 itemtype 属性来完成。第二件要做的事是声明词汇表的作用域，这要通过 itemscope 属性。在这个例子中，所有我们希望更好的语义化的数据都位于一个<section>元素内，因此,我们将在<section>元素上声明 itemtype 和 itemscope 属性：

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

你的名字是<section>元素内的第一个数据。它被一个<h1>元素所包裹。这个<h1>元素不需要任何特殊处理，所以它适用于表 10.1 里的“所有其他元素”规则，其微数据属性值就是一个元素的文本内容而已（如果名字包裹在<p>， 或者元素内，也同样可行）：

```
<h1 itemprop="name">Mark Pilgrim</h1>
```

在自然语言中，上面这行代码的意思就是：“这是一个 name 属性，它属于 *http://data-vocabulary.org/Person* 这个词汇表。它的值是 Mark Pilgrim。”

接下来是 photo 属性。它应该是一个 URL。根据表 10.1，一个<img>属性的“值”是它的 src 属性。嘿，瞧，你的照片的 URL 已经在<img src>属性里写好了！需要做的仅仅是声明这个<img>元素是 photo 属性：

```
<p><img itemprop="photo"
        src="http://www.example.com/photo.jpg"
        alt="[me smiling]"></p>
```

在自然语言中，这行代码的意思是："这是词汇表 *http://data-vocabulary.org/Person* 里的 photo 属性。该属性的值是 *http://www.example.com/photo.jpg*"。

最后，url 属性值自然也是一个 URL。根据表 10.1，一个<a>元素的属性值是它的 href 属性。同样的，这些都能完美地集成到现有页面代码中。你还需要做的事情就是说明这个<a>元素是 url 属性：

```
<a itemprop="url" href="http://diveintomark.org/">dive into mark</a>
```

自然语言中，这行代码说的是："这是 *http://data-vocabulary.org/Person* 词汇表里的 url 属性。该属性的值是 *http://diveintomark.org/*"。

当然，如果你的页面代码看起来有点不同，那也不是什么问题。微数据属性和值可以添加到任何 HTML 代码中，即便是上个世纪、表格排版、让设计师抓狂的网页。尽管我并不推荐使用这种页面，但它们还是很常见，并且你仍然可以往里面添加微数据：

```
<TABLE>
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
</TABLE>
```

要标记 name 属性，只需要给包含姓名的那个表格单元格加上一个 itemprop 属性即可。依照微数据属性值表格，"单元格"并没有特殊的规则，因此它们取默认值。即微数据属性就是文本内容：

```
<TR><TD>Name<TD itemprop="name">Mark Pilgrim
```

添加 url 属性就稍微麻烦一点。这段代码没有正确使用<a>元素。它没有将链接目标作为 href 属性的值，而是在 onclick 属性中使用 JavaScript 调用一个函数（没有给出）来取出 URL，并导航过去。继续一点废话，我们架设这个函数还会打开一个不带滚动条的小弹出窗口。上个世纪的互联网就是这么的有趣！

无论如何，你仍然可以将这个转换成微数据属性，只需要一点点创意。现在已经无法直接使用<a>元素。href 属性的值不是链接目标，并且也没有办法修改"在<a>元素中，微数据属性值就在 href 中"这个规则。但是你可以在这整个混乱的代码之外添加一个包裹元素，并使用它来加上 url 微数据属性：

```
<TABLE itemscope itemtype="http://data-vocabulary.org/Person">
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <span itemprop="url">
      <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
    </span>
</TABLE>
```

由于针对元素没有特殊的处理规则，它便符合默认规则“微数据属性就是文本内容”。“文本内容”并不意味着“所有在这个元素里的代码”（如同你可以通过 innerHTML 这个 DOM 属性所得到的）。它的意思是“仅仅是文本。”在这种情况下，*http://diveintomark.org/* 是元素内的<a>元素的文本内容。

小结：你可以往任何页面代码上添加微数据属性。如果你正确地使用了 HTML，那你会发现添加微数据变得更容易。

10.4 标注“人”
Marking Up People

上一节里的简单例子并不是完全臆造的。确实存在一个微数据词汇表专门用于标注关于人的信息，就是这么简单。让我们来仔细看看。

将微数据整合到一个个人网站最简单的方式就在“关于”页面上。你应该确实有一个“关于”页面吧？如果没有，你可以跟随我使用额外的语义来扩展这个示例“关于”页面（*http://diveintohtml5.org/examples/person.html*）。最后的成品在这里：*http://diveintohtml5.org/examples/person-plus-microdata.html*。

让我们看看原始的页面代码，尚未添加任何微数据属性：

```
<section>
  <img width="204" height="250"
       src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
       alt="[Mark Pilgrim, circa 2000]">

  <h1>Contact Information</h1>
  <dl>
    <dt>Name</dt>
    <dd>Mark Pilgrim</dd>

    <dt>Position</dt>
    <dd>Developer advocate for Google, Inc.</dd>

    <dt>Mailing address</dt>
    <dd>
      100 Main Street<br>
      Anytown, PA 19999<br>
      USA
    </dd>
  </dl>
  <h1>My Digital Footprints</h1>
```

```
    <ul>
      <li><a href="http://diveintomark.org/">weblog</a></li>
      <li><a href="http://www.google.com/profiles/pilgrim">Google profile</a></li>
      <li><a href="http://www.reddit.com/user/MarkPilgrim">Reddit.com profile</a></li>
      <li><a href="http://www.twitter.com/diveintomark">Twitter</a></li>
    </ul>
  </section>
```

你需要做的第一件事情，就是声明你所使用的词汇表，以及你所想要添加的属性的作用域。这可以通过给最外层的元素添加 itemtype 和 itemscope 属性来做到。在本例中，这是一个<section>元素：

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

注意：本节中所做的修改你都可以在线跟踪。之前：*http://diveintohtml5.org/examples/person.html*；之后：*http://diveintohtml5.org/examples/person-plus-microdata.html*。

现在你可以开始定义来自 *http://data-vocabulary.org/Person* 词汇表的微数据属性了。但那些属性都是什么呢？你可以访问 *http://data-vocabulary.org/Person*，来查看一个属性列表。微数据规范并不要求这一点，但我认为这肯定一个“最佳实践”。毕竟，如果你希望其他开发者们实际使用你的微数据词汇表，那你就需要提供文档。而提供文档的最佳地方当然就是词汇表的 URL 本身了。表 10.2“Person 词汇表”列出了 Person 词汇表里的属性。

表 10.2　Person 词汇表

属性	说明
name	姓名
nickname	昵称
photo	一个图像链接
title	此人的头衔（例如，“财务经理”）
role	此人的角色（例如，“会计”）
url	链接到一个网页，如此人的主页
affiliation	一个与此人有关的组织名称（例如雇主）
friend	表示此人和另一人之间的一种社会关系
contact	表示此人和另一人之间的一种社会关系
acquaintance	表示此人和另一人之间的一种社会关系
address	此人的位置（可以有子属性：街道地址、地方、区域、邮编和国家名称）

这个示例“关于”页面上的第一样东西就是我的照片。自然地，我们用一个<img>元素来标注它。要声明这个<img>元素是我的照片，我们需要做的事情就是给它加上itemprop="photo"：

```
<img itemprop="photo" width="204" height="250"
    src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
    alt="[Mark Pilgrim, circa 2000]">
```

微数据的属性值自来哪里呢？它已经存在，在 src 属性。回顾表 10.1，<img>元素的“值”就是它的 src 属性。每个<img>元素都有一个 src 属性——否则图像就不会显示出来——并且 src 总是一个网址。看到了吧？如果你正确地使用了 HTML，添加微数据就变得非常简单。

此外，这个<img>元素并不孤单。它是页面上<section>元素的子元素，后者含有我们刚刚声明的 itemscope 属性。微数据会重用页面上元素的父子关系，来定义微数据属性的作用域。用自然语言表述，我们做的就是：“这个<section>元素代表一个人。你在该<section>元素的子元素中发现的任何微数据属性，都是这个“人”的属性。”如果有帮助，你还可以将<section>元素看作一个句子的主语。itemprop 属性则代表句子的谓语动词——类似“被描绘为”——然后微数据属性就代表句子的宾语：

这个人[明确的，来自<section itemscope ItemType ="...">]

被描绘为[明确的，来自<img itemprop="photo">]

http://diveintohtml5.org/examples/2000_05_mark.jpg [隐含的，来自<img src>属性]

主语只需要被定义一次，通过在最外层的<seciton>元素上添加 itemscope 和 itemtype 属性来做到。谓语动词的定义是通过将 itemprop="photo"添加到<img>元素上来完成。宾语则完全不需要任何特别的标记代码，因为根据表 10.1，一个<img>元素的属性值就是其 src 属性。

继续下面的，我们可以看到一个<h1>标题以及一个<dl>列表的开始。<h1>和<dl>都不需要再用微数据标注。并不是每个 HTML 元素都需要微数据属性。微数据里重要的是属性本身，而不是属性周围的代码或者标题。这个<h1>不代表任何属性，它就是一个标题而已。同样，<dt>里写着“Name”，但它只是一个文本标签，不是属性：

```
<h1>Contact Information</h1>
  <dl>
    <dt>Name</dt>
    <dd>Mark Pilgrim</dd>
```

那么，真正的信息在哪里？在<dd>元素里，所以那里才是我们需要增加 itemprop 属性的地方。用哪个属性呢？name 属性。属性值在哪里？<dd>元素的文本内容就是。需要特别的标记吗？

根据表 10.1，不需要了，对<dd>元素不用做特殊处理，属性值直接就是它的文本内容。

```
<dd itemprop="name">Mark Pilgrim</dd>
```

我们刚刚做的事情用自然语言表达的话是什么？“这个人的名字是 Mark Pilgrim。”很好。继续往前。

接下来的两个属性是有点棘手。这是添加微数据之前的代码：

```
<dt>Position</dt>
<dd>Developer advocate for Google, Inc.</dd>
```

根据 Person 词汇表里的定义，文本“Developer advocate for Google, Inc.”实际上包括了两个属性：title（“Developer advocate”）以及 affiliation（“Google, Inc.”）。如何通过微数据表达这些呢？简短的回答是：做不到。微数据没有办法将一行文本分割成不同的属性。不能说“这行文本前 18 个字符是一个微数据属性，后 12 个字符是另一个微数据属性”。

这也不算太坏。设想一下你想要给“Developer advocate”和“Google, Inc.”分别设定不同的字体，CSS 也无法做到。那么怎么办？你首先需要将文本的不同部分用额外元素包裹起来，比如用一个，然后给不同的设定不同的 CSS 规则。

这个技巧对使用微数据也很有用。这里有两部分不同种类的信息：一个 title 和一个 affiliation。如果你将每部分都用一个元素包裹，便可以在每个上分别声明微数据属性了。

```
<dt>Position</dt>
<dd><span itemprop="title">Developer advocate</span> for
    <span itemprop="affiliation">Google, Inc.<span></dd>
```

搞定！用自然语言表达，这些代码说的是：“这个人的头衔是‘Developer advocate’，受雇于 Google 公司”。两句话，两个微数据属性。尽管多用了一点标签，但这代价是值得的。

这个技巧对于标注街道地址同样有用。Person 词汇表定义了 address 属性，它本身是微数据项目。这意味着 address 有自己的词汇表（*http://data-vocabulary.org/Address*），并定义自己的属性：街道地址、地方、区域、邮编，以及国家名称。

如果你是一个程序员，那你可能对使用“点”操作符定义对象及其属性感到熟悉。想想这样的关系：

- Person
- Person.address
- Person.address.street-address
- Person.address.locality
- Person.address.region

- Person.address.postal-code
- Person.address.country-name

在这个例子中，整个街道地址都是包含在一个单独的<dd>元素内。（再提一次，<dt>元素只是一个文字标签，因此使用微数据增加语义和它无关。）标上 address 属性很容易。只需给<dd>元素添加一个 itemprop 属性即可：

```
<dt>Mailing address</dt>
<dd itemprop="address">
```

但记住，address 属性本身也是一个微数据项目。这意味着我们还需要添加 itemtype 属性和 itemscope 属性：

```
<dt>Mailing address</dt>
<dd itemprop="address" itemscope
  itemtype="http://data-vocabulary.org/Address">
```

之前我们已经见过，但仅限于顶级项目。一个<section>元素定义了 itemtype 和 itemscope，然后所有该<section>之内的、定义微数据属性的元素也就位于这个特定词汇表的“作用域”之内了。但这是我们第一次看到嵌套的作用域——在一个现存作用域（<section>元素）内定义一个新的 itemType 和 itemscope（在<dd>元素上）。这个嵌套作用域的工作方式和 HTML DOM 完全一样。这个<dd>元素有数个子元素，所有都处于<dd>上定义的词汇表的作用域之内。当<dd>元素由对应的</dd>标签所结束，作用域又回复到父元素（本例中即是<section>）上的那个词汇表。

Address 词汇表的属性会遭遇到与前面 title/affiliation 属性同样的问题。这里只有一整行文本，我们却想将其标注为数个单独的微数据属性。解决方案也相同：给每个信息片段包上一个元素，再在各个上声明微数据属性：

```
  <dd itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
    <span itemprop="street-address">100 Main Street</span><br>
    <span itemprop="locality">Anytown</span>,
    <span itemprop="region">PA</span>
    <span itemprop="postal-code">19999</span>
    <span itemprop="country-name">USA</span>
  </dd>
</dl>
```

用自然语言表达：“此人有一个邮政地址。地址的街道部分是‘100 Main Street’。地区部分是‘Anytown’。州是‘PA’。邮政编码是‘19999’。国家的名称是‘美国’。”十分简单。

专家答疑

问：邮政地址的格式只能是美国式的吗？

答：不。Address 词汇表的属性是通用的，可以描述世界上绝大多数邮政地址格式。当然并不是所有的地址都有全部属性值，但这没关系。有些地址可能需要在一个属性里写上多个值，这也没关系。例如，如果你的邮政地址包含街道名和单元号，它们都可以写在 street-address 子属性中：

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">
    100 Main Street
    Suite 415
  </span>
  ...
</p>
```

示例“关于”页面还有一样东西：一组 URL 列表。Person 词汇表有对应的属性，叫做 url。一个 url 属性可以是任何东西，真的。（当然，它必须是一个网址，这个你可能已经猜到了。）我的意思是，url 属性的定义是非常宽松的。该属性可以是任何 URL，只要它和这个人相关：一个 Blog、照片库、像 Facebook 或者 Twitter 那样的其他个人资料页面。

另外一个需要注意的重要事情是，一个人可以有多个 url 属性。从技术上讲，任何属性都可以出现一次以上，但是到目前为止，我们还没有用到这一点。例如，你可以有两个 photo 属性，每个指向一个不同的图像 URL。这里，我想列出四个不同的 URL：我的 Blog、我的 Google 个人资料页、我在 Reddit 的用户资料页，以及我的 Twitter 帐户。在 HTML 代码中，以上就是一个链接列表：四个<a>元素，每个都位于自己的<li>元素内。至于微数据，则每个<a>元素都有一个 itemprop="url"属性：

```
<h1>My Digital Footprints</h1>
<ul>
  <li><a href="http://diveintomark.org/"
         itemprop="url">weblog</a></li>
  <li><a href="http://www.google.com/profiles/pilgrim"
         itemprop="url">Google profile</a></li>
  <li><a href="http://www.reddit.com/user/MarkPilgrim"
         itemprop="url">Reddit.com profile</a></li>
  <li><a href="http://www.twitter.com/diveintomark"
         itemprop="url">Twitter</a></li>
</ul>
```

根据表 10.1，<a>元素有特殊的处理规则。微数据属性值是其 href 属性值，而不是其文本内容。微数据处理器实际上会忽略链接文字。因此，上述代码用自然语言表达就是："这个人有四个 URL 地址，分别是 *http://diveintomark.org/*、*http://www.google.com/profiles/pilgrim*、*http://www.reddit.com/user/MarkPilgrim*，以及 *http://www.twitter.com/diveintomark*"。

10.4.1 Google Rich Snippets 介绍

我想暂时退后一步询问："我们为什么做这些事情？"难道我们只是为了增加语义而增加语义？不要误会，我和你们一样喜欢摆弄尖括号。但我们为什么要了解和使用微数据呢？何苦？

一共有两类主要的应用程序会消费 HTML 及其扩展，以及 HTML5 微数据：

- Web 浏览器
- 搜索引擎

对于浏览器，HTML5 定义了一组 DOM API，用于从一个网页上提取微数据项目、属性以及属性值。但在我撰写本书的时候，还没有任何一个浏览器实际支持这一 API。所以这是一个死胡同……直到有浏览器实现这些客户端 API。

另一个主要的 HTML 消费者就是搜索引擎。一个搜索引擎会用关于一个人的微数据属性来干什么呢？想像一下：搜索引擎可以集成并显示一些结构化的信息，而不仅仅是显示页面标题和一些摘要文字。这些结构化信息包括：姓名、职称、工作单位、地址，甚至一个资料照片缩略图。这会引起你的兴趣吗？至少我会很有兴趣。

Google 支持微数据，作为其 Rich Snippet 计划原书链接的一部分。当 Google 的 Web 抓取工具分析到你的网页，并找到符合 *http://data-vocabulary.org/Person* 词汇表的微数据属性，它就会解析属性数据，同页面其余数据一并存储。Google 甚至提供了一个方便的工具，能显示 Google 找到了哪些你的微数据。用它测试我们的示例页面（*http://www.diveintohtml5.org/example/person-plus-microdata.html*），输出结果是：

```
Item
  Type: http://data-vocabulary.org/person
  photo = http://diveintohtml5.org/examples/2000_05_mark.jpg
  name = Mark Pilgrim
  title = Developer advocate
  affiliation = Google, Inc.
  address = Item( 1 )
  url = http://diveintomark.org/
```

```
    url = http://www.google.com/profiles/pilgrim
    url = http://www.reddit.com/user/MarkPilgrim
    url = http://www.twitter.com/diveintomark

  Item 1
    Type: http://data-vocabulary.org/address
    street-address = 100 Main Street
    locality = Anytown
    region = PA
    postal-code = 19999
  country-name = USA
```

全都找到了：源自<img src>的 photo 属性，所有四个源自<a href>的 URL，甚至还有 address 对象（显示为“Item 1”）及其全部五个子属性。

那么 Google 如何使用这些信息呢？那要看情况。并没有硬性规定说微数据属性应当如何显示，那些应当显示，或者它们究竟该不该显示。如果有人搜索“Mark Pilgrim”，并且 Google 认为这个“关于”页面应该显示在结果中，此外 Google 还觉得这些源自该页面上的微数据属性值得显示出来，那么搜索结果就会看上去像图 10.1“含有微数据 Person 的示例搜索结果”那样。

About Mark Pilgrim
Anytown PA - Developer advocate - Google, Inc.
Excerpt from the page will show up here.
Excerpt from the page will show up here.
diveintohtml5.org/examples/person-plus-microdata.html - Cached - Similar pages

图 10.1　含有微数据 Person 的搜索结果页

第一行，“About Mark Pilgrim”,实际上是网页的标题，位于<title>元素中。这并不令人兴奋，对每个网页 Google 都会这样做。但是，第二行就完全是直接从我们添加到页面上的微数据标记中提取出的信息了。“Anytown PA”的是邮政地址的一部分，使用 *http://data-vocabulary.org/Address* 词汇表标注。“Developer advocate”和“Google, Inc.”是两个来自 *http://data-vocabulary.org/Person* 词汇表的属性（分别是 title 和 affiliation）。

这实在是相当惊人的。你并不需要是一家大型企业，从而得到搜索引擎公司的优待，为你提供定制的搜索结果页。仅仅需要 10 分钟来添加几个 HTML 属性，标注出那些你反正已经提供了的数据。

专家答疑

问：我按照你说的做了，但我的 Google 搜索结果看上去并没有什么变化。这是怎么回事呢？

答：Google 并不保证给定网页或者站点上的代码会一定被用于搜索结果”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170*）。但即便 Google 决定不使用你的微数据标注，其他搜索引擎可能会采用。就像 HTML5 的其他特性一样，微数据是一个开放的标准，每个人都可以提供实现。你的职责就是提供尽可能多的数据，让别人自己决定怎么利用。而它们可能会使你吃惊！

10.5 标注“组织”
Marking Up Organizations

微数据并不仅限于单一的词汇表。“关于”页面很不错，但你恐怕只能有一个。还想要更多？下面就来学习如何标注组织机构和商业单位。

我创建了一个商业单位列表的示例页面。下面先来看看原始 HTML 代码，不含有微数据：

```
<article>
  <h1>Google, Inc.</h1>
  <p>
    1600 Amphitheatre Parkway<br>
    Mountain View, CA 94043<br>
    USA
  </p>
  <p>650-253-0000</p>
  <p><a href="http://www.google.com/">Google.com</a></p>
</article>
```

注意：你可以在线跟踪本节所做的全部页面修改。之前：*http://diveintohtml5.org/examples/organization.html*；之后：*http://diveintohtml5.org/examples/organization-plus-microdata.html*。

简短而亲切。所有关于该组织的信息都包含在<article>元素内，我们就从这里开始：

```
<article itemscope itemtype="http://data-vocabulary.org/Organization">
```

和标注“人”一样，你需要在最外层元素内设置 itemType 和 itemscope 属性。本例中，最外层的是一个<article>元素。itemtype 属性声明你所使用的微数据词汇表（本例中是 *http://data-vocabulary.org/Organization*），而 itemscope 属性声明了所有你在子元素中设置的属性都归属于该词汇表。

那么，Organization 词汇表中有些什么呢？其实很简单明了。它们中的一些应该十分眼熟了。表 10.3“Organization 词汇表”列出了相关的属性。

表 10.3 Organization 词汇表

属性	说明
name	该组织的名称（例如“Initech”）
url	指向该组织主页的链接
address	该组织的地址。可以包含子属性：街道地址，地方，区域，邮编，国家名称
tel	该组织的电话号码
geo	该组织地址的地理坐标位置。总是包含两个子属性：纬度和经度

最外层<article>元素内的第一个子元素是<h1>。这个<h1>元素包含商业机构的名称，因此直接给它放上一个 itemprop="name"属性：

```
<h1 itemprop="name">Google, Inc.</h1>
```

根据表 10.1，<h1>元素不需要任何特殊处理。微数据的属性值就是<h1>元素的文本内容。用自然语言表达：“这个组织的名称是‘Google, Inc.’”。

接下来是一个街道地址。标注一个组织机构的地址和标注一个人的地址完全一样。首先，给街道地址的最外层元素（本例中是一个<p>）加上一个 itemprop="address"，指明这是该组织机构的地址属性。但是，address 自己的属性呢？还需要定义 itemtype 和 itemscope 属性来说明这是一个有自己的属性的 Address 项目。

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
```

最后，需要将每份独立的信息片段用元素包起来，以便可以加上合适的微数据属性名（street-address，locality，region，postal-code，以及 country-name）：

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">1600 Amphitheatre Parkway</span><br>
  <span itemprop="region">CA</span>
  <span itemprop="region">的 CA </跨度>
  <span itemprop="postal-code">94043</span><br>
  <span itemprop="country-name">USA</span>
</p>
```

用自然语言表达，我们刚刚说的是：“该组织有一个地址。街道部分是‘1600 Amphitheatre Parkway’。城市是‘Mountain View’。州是‘CA’。邮编是‘94043’。国家名称是‘美国’。”

下一步：组织机构的电话号码。电话号码是出了名的棘手，因为各个国家的格式有所不同（如果你想打电话到另一个国家，就更加麻烦）。在这个例子中，我们有一个美国的电话号码，适用于从美国国内拨打：

```
<p itemprop="tel">650-253-0000</p>
```

（嘿，如果你没有注意到，Address 词汇表的作用域随着<p>的关闭而结束。现在我们回到 Organization 词汇表了。）

如果你想列出多个电话号码——可能一个面向美国客户，一个面向国际客户——你也可以做到。任何微数据属性都可以重复。只要确保每个电话号码位于自己的 HTML 元素内：

```
<p>
  US customers: <span itemprop="tel">650-253-0000</span><br>
  UK customers: <span itemprop="tel">00 + 1* + 6502530000</span>
</p>
```

根据表 10.1，<p>和元素都不需要做特殊处理。微数据属性 tel 的值就是文本内容。Organization 微数据词汇表也没有打算对电话号码再做分割。整个 tel 属性就是自由文本内容。如果你想将区号放在括号中，或者使用空格代替连线来分隔号码，你完全可以那样做。如果一个微数据客户端要解析电话号码，具体怎么做随便它。

接下来还有一个熟悉的属性：url。如同前一节中所描述的给一个人关联一个 URL 那样，你也可以给一个组织机构关联一个 URL。它可能是该公司的主页、联系信息页、产品页，或其他任何网址。如果该 URL 是关于、来自或从属于这一组织机构，那么就给它标注一个 itemprop="url"：

```
<p><a itemprop="url" href="http://www.google.com/">Google.com</a></p>
```

根据表 10.1，<a>元素有特殊处理规则。微数据的属性值是 href 值，而不是链接文本。这行代码用自然语言表达就是“该组织机构与网址 *http://www.google.com/*相关联。”它也没有给出关于这个关联更具体的信息，并且也不包含链接文本“Google.com”。

最后，我想谈谈地理位置。不，这不是 W3C 的 geolocation API（参见第 6 章）。我要谈的是如何使用微数据来标注一家组织机构的物理位置。

到目前为止，我们所有的例子都着眼于标注可见的数据。也就是说，你有一个写在<h1>里的公司名称，于是你给这个<h1>加上一个 itemprop 属性，来声明这个（可见的）标题文本实际上是组织机构的名称。或者你有一个<img>元素呈现了一张照片，你给这个<img>元素加上 itemprop 属性来声明这个（可见的）图像是一个人的照片。

在这个例子中，地理位置信息则不是这样的。并没有可见的文字给出组织机构所在的确切纬度和经度（精确到小数点后四位！）实际上，标注微数据之前的组织机构页面上压根就没有地理位置信息。它有一个指向 Google 地图的链接，但这个链接里也不含有纬度和经度坐标（类似信息显示为一种 Google 专有的格式）。即使有一个链接指向一个假想的在线地图服务，它接受将纬度和经度组表作为 URL 参数传递，微数据也没有办法分离出 URL 的不同部分。你无法声明 URL 的第一个参数是一个纬度值，第二个是经度值。

为了处理这种比较极端的情况，HTML5 提供了一种标注不可见数据的方法。这个技巧只适合作为最后手段。你应该尽可能地使用可见的数据。那些不可见的，只有机器可以读到的数据容易很快变得陈旧。这话的意思是，可见的文字内容会不时得到更新，但是更新者可能会忘记修改那些不可见的内容。这种情况比想象的更加常见，它也很可能发生在你身上。

尽管如此，某些时候人们无法避免使用不可见数据。也许你的老板就是想要机器可读的地理位置信息，但是又不想在界面上显示难看的两个 6 位数字。这时不可见数据就是唯一选择了。可以聊以慰籍的是，你可以将这些不可见数据放在紧挨着它所描述的可见文本的后面，这或许可以帮助提醒更新文本内容的人，他还需要修改文本后面的不可见数据。

本例中，我们可以创建一个元素，将不可见的地理位置数据放在这个里：

```
  <span itemprop="geo" itemscope
    itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="37.4149" />
    <meta itemprop="longitude" content="-122.078" />
  </span>
</article>
```

地理位置信息定义在自己的词汇表里，就像 Person 或 Organization 里的 address 那样。因此，这个元素需要三个属性：

itemprop="geo"

表示该元素代表所属组织机构的地理属性。

itemtype="http://data-vocabulary.org/Geo"

表示该属性所符合的微数据词汇表。

itemscope

表示该元素是一个微数据项目的内含元素，该微数据项目具有自己的词汇表（在itemtype 属性中给出）。所有该元素内的回溯性都是 Geo 词汇表中定义的属性（*http://data-vocabulary.org/Geo*），而不是外围的 Organization 词汇表（*http://data-vocabulary.org/Organization*）里的属性。

接下来这个示例需要回答的大问题就是“如何标注不可见的数据？”。这需要用到<meta>元素。在先前版本的 HTML 中，<meta>元素只能用在页面的<head>区域内（参见第 34 页的“<head>元素”一节）。在 HTML5 中，你可以在任何地方使用<meta>元素。而这正是我们要做的：

```
<meta itemprop="latitude" content="37.4149" />
```

根据表 10.1，<meta>元素具有特殊的处理规则。微数据属性值位于 content 属性里。这个属性不会被显示出来，我们可以在这里设定无限多的不可见数据。权利越大，责任越大。在这种情况下，你的责任就是确保这些不可见数据和它们周围的可见数据保持同步。

Google Rich Snippets 里对 Organization 词汇表并无直接支持，所以这里我不能给出漂亮的搜索结果页。但是在后面两个案例分析“事件”和“评价”里，组织机构占据重要角色，而 Google Rich Snippets 支持“事件”和“点评”。

10.6 标注“事件”

Marking Up Events

生活中会发生许多事情，有些发生在指定的时间。如果你可以告诉搜索引擎一件事情所要发生的确切时间，那岂不是很好？有专门的 HTML 标记用于这个需求。

我们来看一个日程表示例，它是关于我的演讲活动（*http://diveintohtml5.org/examples/event.html*）：

```
<article>
  <h1>Google Developer Day 2009</h1>
  <img width="300" height="200"
       src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
       alt="[Mark Pilgrim at podium]">
  <p>
    Google Developer Days are a chance to learn about Google
    developer products from the engineers who built them. This
    one-day conference includes seminars and "office hours"
    on web technologies like Google Maps, OpenSocial, Android,
    AJAX APIs, Chrome, and Google Web Toolkit.
  </p>
```

```
  <p>
    <time datetime="2009-11-06T08:30+01:00">2009 November 6, 8:30</time>
      &ndash;
    <time datetime="2009-11-06T20:30+01:00">20:30</time>
  </p>
  <p>
    Congress Center<br>
    5th května 65<br>
    140 21 Praha 4<br>
    Czech Republic
  </p>
  <p><a href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
      GDD/Prague home page</a></p>
</article>
```

注意：你可以在线跟踪本节所做的全部修改。之前：*http://diveinto-html5.org/examples/event.html*；之后：*http://diveintohtml5.org/examples/event-plus-microdata.html*。

所有关于该事件的信息都包含在这个<article>元素内，所以这里就是需要添加itemtype和itemscope属性的地方：

```
<article itemscope itemtype="http://data-vocabulary.org/Event">
```

Event词汇表的URL是*http://data-vocabulary.org/Event*，其中也有一个好看的小图表描述了词汇表中定义的属性。那都有些什么属性呢？参见表10.4。

表10.4 Event词汇表

属性	说明
summary	事件的名称
url	指向事件详情的链接
location	时间发生/活动举行的地点或场所。可以由一个嵌套的组织机构（Organization词汇表）或地址（Address词汇表）来表示
description	对事件的描述
startDate	事件开始日期和时间，ISO格式
endDate	事件结束的日期和时间，ISO格式
duration	事件的持续时间，ISO格式
eventType	事件类别（例如“音乐会”或者“讲座”）。这是一个自由文本属性，不是枚举属性
geo	事件地点的地理位置坐标。总是包含两个子属性：纬度和经度
photo	一个指向与事件相关的照片或图像的链接

该事件的名称位于一个<h1>元素中。根据表 10.1，<h1>元素没有什么特殊的处理规则。微数据属性值就是<h1>元素的文本内容。因此，我们需要做的就是添加 itemprop 属性来声明该<h1>元素包含事件的名称：

```
<h1 itemprop="summary">Google Developer Day 2009</h1>
```

自然语言表达就是：“该事件/活动的名称是‘Google 开发者日 2009’”。

该事件有一张照片，它可以用 photo 属性来标注。如你所期望的，照片已经用<img>元素来标记了。如同 Person 词汇表里的 photo 属性（参见第 165 页“微数据的数据模型”一节），Event 的 photo 属性也是一个 URL。根据表 10.1 所说，<img>元素的属性值位于其 src 属性中，我们需要做的唯一一件事就是给<img>元素加上 itemprop 属性：

```
<img itemprop="photo" width="300" height="200"
     src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
     alt="[Mark Pilgrim at podium]">
```

用自然语言表达就是：“该事件/活动的照片位于 *http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg*”。

接下来是一段较长的时间描述，形式上是自由文本内容：

```
  <p itemprop="description">Google Developer Days are a chance to
learn about Google developer products from the engineers who built
them. This one-day conference includes seminars and "office
hours" on web technologies like Google Maps, OpenSocial,
Android, AJAX APIs, Chrome, and Google Web Toolkit.</p>
```

再接下来是个新东西。事件通常发生在给定的日期，有给定的开始和结束时间。在 HTML5 中，日期和时间应该使用<time>元素来标注（参见第 49 页“日期和时间”一节），这里已经这样做了。所以问题变成：如何给<time>元素添加微数据属性？回顾表 10.1，我们看到，<time>元素有特殊处理规则。一个<time>元素的微数据属性值位于其 datetime 属性中。并且，Event 词汇表中的 startDate 和 endDate 属性采用 ISO 格式，与<time>元素的 datetime 属性一致。再一次地，核心 HTML 词汇表的语义和我们自定义的微数据词汇表的语义很好地衔接了。使用微数据标注起始和结束时间很简单：

1. 首先是正确使用 HTML（用<time>元素标注日期和时间）。
2. 增加一个 itemprop 属性：

```
<p>
  <time itemprop="startDate" datetime="2009-11-06T08:30+01:00">2009 November 6, 8:30</time>
    &ndash;
  <time itemprop="endDate" datetime="2009-11-06T20:30+01:00">20:30</time>
</p>
```

用自然语言表达就是："该事件/活动开始于 2009 年 11 月 6 日上午 8:30，持续到当天傍晚 20:30（布拉格当地时间，GMT+1）。"

接下来是 location 属性。Event 词汇表中定义说，这可以是一个 Organization 或者 Address。本例中，该活动在一个会议中心举行：布拉格会议中心。将它标注为一个 Organization 还可以让我们继续包含这个地点的名称和位置。

首先，我们声明包含地址的<p>元素就是 Event 的 location 属性，这个元素本身具有自己的微数据格式，符合 *http://data-vocabulary.org/Organization* 词汇表：

```
<p itemprop="location" itemscope
   itemtype="http://data-vocabulary.org/Organization">
```

其次，将组织机构的名称包在一个元素内，并给它添加 itemprop 属性：

```
<span itemprop="name">Congress Center</span><br>
```

这里"name"定义于 Organization 词汇表，而非 Event 词汇表。<p>元素定义了 Organization 词汇表作用域的开始，此处这个<p>尚未被关闭。我们这里定义的任何微数据属性都位于最"近"的词汇表作用域内。嵌套的词汇表就像一个栈。此处我们还没有出栈，因此这些属性仍然属于 Organization 词汇表。

事实上，我们即将往这个栈里添加第三个词汇表——Address，用于这个 Event 下的 Organization 中：

```
<span itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
```

再提一遍，我们想将地址的每个部分都标注为单独的微数据属性，所以需要引入一些元素（如果我在这里讲解得太快，请翻回来阅读关于标注人的地址（参见第 17 页"标注人"）和标注组织地址（参见第 177 页"标注组织"）的内容）：

```
<span itemprop="street-address">5th kvetna 65</span><br>
<span itemprop="postal-code">140 21</span>
<span itemprop="locality">Praha 4</span><br>
<span itemprop="country-name">Czech Republic</span>
```

没有更多 Address 属性了，因此关闭启用 Address 作用域的同时回到栈的上一级：

```
</span>
```

也没有更多的 Organization 属性了，因此关闭启用 Organization 作用域的<p>，再回到上一层栈中：

```
</p>
```

现在回来继续定义事件的属性。下一个属性是 geo，表示事件发生的物理位置。这里使用的 Geo 词汇表和我们在上一节中标注组织机构的位置时所用的 Geo 词汇表相同。需要一个元素作为容器，并设定它的 itemtype 和 itemscope 属性。在该元素中，需要用两个<meta>元素，一个标注纬度，一个标注经度：

```
<span itemprop="geo" itemscope itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="50.047893" />
  <meta itemprop="longitude" content="14.4491" />
</span>
```

因为包含 Geo 词汇表属性的已经关闭，现在还是位于 Event 词汇表的作用域内。最后一个属性是 url，你应该很熟悉它了。给事件关联一个 URL 和前面几节里讲到的，给人和组织机构关联 URL 完全相同。如果正确地使用了 HTML（<a href>），声明这个超链接是一个微数据 url 属性的方式只不过就是给它添加 itemprop 属性：

```
  <p>
    <a itemprop="url"
       href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
       GDD/Prague home page
    </a>
  </p>
</article>
```

这个示例页面上（*http://diveintohtml5.org/examples/event.html*）其实还列出了另一个活动，是我在蒙特利尔的 ConFoo 会议上的演讲。为了节省时间，我就不逐行解释它的代码了。它基本上和布拉格这里的活动相同：一个 Event 项目，里面嵌套着 Geo 和 Address 项目。我只是顺带重申，一个单一的页面上可以有多个事件，每个都是微数据标注的。

10.6.1 Google Rich Snippets 的回归

根据 Google 的 Rich Snippets 测试工具，以下就是 Google 爬虫可以从我们的示例活动列表页面上抓取到的信息（*http://diveintohtml5.org/examples/event-plus-microdata.html*）：

```
Item
    Type: http://data-vocabulary.org/Event
    summary = Google Developer Day 2009
    eventType = conference
    photo = http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg
    description = Google Developer Days are a chance to learn about Google developer
                  products from the engineers who built them. This one-day
                  conference includes seminars and office hours on web technologies
                  like Goo...
    startDate = 2009-11-06T08:30+01:00
```

```
    endDate = 2009-11-06T20:30+01:00
    location = Item(__1)
    geo = Item(__3)
    url = http://code.google.com/intl/cs/events/developerday/2009/home.html

Item
    Id: __1
    Type: http://data-vocabulary.org/Organization
    name = Congress Center
    address = Item(__2)

Item
    Id: __2
    Type: http://data-vocabulary.org/Address
    street-address = 5th kvetna 65
    postal-code = 140 21
    locality = Praha 4
    country-name = Czech Republic

    Item
    Id: __3
    Type: http://data-vocabulary.org/Geo
    latitude = 50.047893
    longitude = 14.4491
```

正如你看到的，这里列出了所有我们通过微数据添加的信息。属于单独的微数据项目的属性被标识了内部 ID (Item(__1), Item(__2)等)，但这不是微数据规范的一部分。它只是 Google 测试工具的习惯做法，来将输出结果线性化处理，展示给你嵌套项目组和它们的属性。

那么，Google 会如何在搜索结果中展示这个示例页呢？（同样，事先说明，这只是一个例子，Google 可能在任何时候改变其搜索结果的格式，并且也无法保证 Google 究竟会不会注意到你代码中的微数据。）它看起来可能像图 10.2 那样：

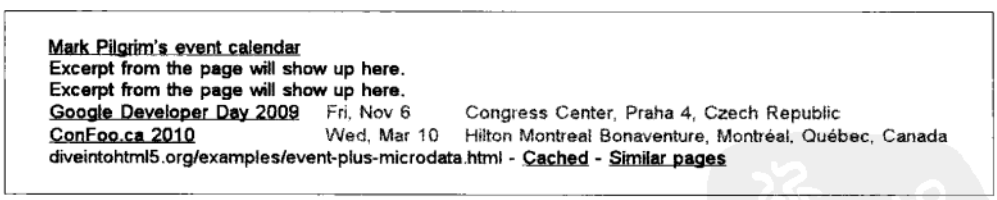


图 10.2　含有微数据的事件列表的搜索结果页示例

在页面标题及自动生成的摘要文字后面，Google 开始使用我们添加的微数据了，并显示了一个小的时间表格。请注意日期格式：“Fri, Nov 6”。这个字符串并没有在任何我们的 HTML 或者微数据里出现过。我们使用的是两个完全符合 ISO 格式的字符串：2009-11-06T08:30+01:00 和 2009-11-06T20:30+01:00。Google 解析了这两个日期，计算出它们其实是在同一天，于是决定将它们显示为一个单独的、格式更加友好的日期。

现在来看物理地址。Google 选择只显示场所名称+地点+国家，而不是确切的街道地址。这得益于我们将地址分成了五个子属性——name、street-address、region、locality，以及 country-name——并且每个部分都被标注为不同的微数据。Google 利用了这些来显示简短的地址。别的微数据解析器可能会做不同的选择来决定显示哪些部分。这些选择无所谓对错。你只需要提供尽可能多、尽可能准确的数据。别人如何解析那是他们自己的事情。

10.7 标注“点评”

Marking Up Reviews

通过 HTML 让 Web（或者说搜索结果）变得更好，这里又有一个例子：商家和产品点评。

下面是我给一家我很喜欢的比萨店写的简评。（这是一个真正的餐厅，顺便说一下。如果你能来北卡州的 Apex，我极力推荐你前往用餐。）这是原始代码（*http://diveintohtml5.org/examples/review.html*）：

```
<article>
  <h1>Anna's Pizzeria</h1>
    <p>★★★★☆ (4 stars out of 5)</p>
    <p>New York-style pizza right in historic downtown Apex</p>
    <p>
      Food is top-notch.Atmosphere is just right for a "neighborhood
      pizza joint." The restaurant itself is a bit cramped; if you're
      overweight, you may have difficulty getting in and out of your
      seat and navigating between other tables. Used to give free
      garlic knots when you sat down; now they give you plain bread
      and you have to pay for the good stuff. Overall, it's a winner.
    </p>
    <p>
      100 North Salem Street<br>
      Apex, NC 27502<br>
      USA
    </p>
    <p>– reviewed by Mark Pilgrim, last updated March 31, 2010</p> </article>
  </article>
```

注意：本节所做的全部修改你都可以在线跟踪。之前：*http://diveintohtml5.org/examples/review.html*；之后：*http://diveintohtml5.org/examples/review-plus-microdata.html*。

这条点评位于一个<article>元素中，所以在这里加上 itemtype 和 itemscope 属性。该词汇表的命名空间 URL 是：

```
<article itemscope itemtype="http://data-vocabulary.org/Review">
```

在 Review 词汇表里有哪些可用的属性呢？我很高兴你这么问。它们列在表 10.5 “Review 词汇表”中。

表 10.5 Review 词汇表

属性	说明
itemreviewed	被点评的项目的名称。可以是产品、服务、商家等等
rating	对该项目的数字质量评级，档次从 1 到 5。也可以嵌套一个使用来自 *http://data-vocabulary.org/Rating* 词汇表的非标准档次
reviewer	点评者的姓名
dtreviewed	点评日期，ISO 格式
summary	关于点评的简要说明
description	点评详细描述

第一个属性很简单：itemreviewed 只是文字，在这里它包含在<h1>元素内，这就是我们设定 itemprop 的地方：

```
<h1 itemprop="itemreviewed">Anna's Pizzeria</h1>
```

我先跳过实际的评级，最后再回过头来讨论。

接下来的两个属性也很简单。summary 属性是一个简短的点评描述，description 则是详细的描述：

```
<p itemprop="summary">New York-style pizza right in historic downtown Apex</p>
<p itemprop="description">
  Food is top-notch. Atmosphere is just right for a "neighborhood
  pizza joint."The restaurant itself is a bit cramped; if you're
  overweight, you may have difficulty getting in and out of your
  seat and navigating between other tables. Used to give free
  garlic knots when you sat down; now they give you plain bread
  and you have to pay for the good stuff.Overall, it's a winner.
</p>
```

location 和 geo 属性也都是我们在前面已经见过的：

```
<p itemprop="location" itemscope
  itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">100 North Salem Street</span><br>
  <span itemprop="locality">Apex</span>,
  <span itemprop="region">NC</span>
  <span itemprop="postal-code">27502</span><br>
  <span itemprop="country-name">USA</span>
  </p>
  <span itemprop="geo" itemscope
        itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="35.730796" />
```

```
    <meta itemprop="longitude" content="-78.851426" />
  </span>
```

最后一行也代表了一个熟悉的问题：它在同一个元素里包含两种信息。点评人的姓名是 Mark Pilgrim，点评日期是 March 31，2010。我们如何标记这两个不同的属性？同样，我们可以将它们包在自己的元素里，然后分别设定 itemprop 属性。实际上，本例中的日期应该首选用<time>元素标记，这样就自然提供了可以放置 itemprop 属性的地方。点评人的姓名则可以放在一个元素内：

```
    <p>
      <span itemprop="reviewer">Mark Pilgrim</span>, last updated
      <time itemprop="dtreviewed" datetime="2010-03-31">
        March 31, 2010
      </time>
    </p>
  </article>
```

好，现在我们讨论评价等级。这是点评标记里最棘手的部分。默认情况下，Review 词汇表里的 rating 属性共分 5 档，1 最差，5 最好。但如果你想用另外的尺度，也完全可以做到。不过我们先谈论默认尺度：

```
<p>★★★★☆ (<span itemprop="rating">4</span> stars out of 5)</p>
```

如果使用默认的 1~5 尺度，唯一需要的属性就是 rating 本身（本例中是 4）。如果想使用不同的尺度呢？没问题，你只需要声明你所使用的尺度的上下限。例如，如果使用 0~10 的尺度，你也可以声明 itemprop="rating" 属性，但需要嵌套一个 Rating 词汇表（*http://data-vocabulary.org/Rating*）来声明你的自定义尺度的上下限，而不是直接给出评级结果。

```
<p itemprop="rating" itemscope
  itemtype="http://data-vocabulary.org/Rating">
  ★★★★★★★★★☆
  (<span itemprop="value">9</span> on a scale of
  <span itemprop="worst">0</span> to
  <span itemprop="best">10</span>)
</p>
```

用自然语言表达就是：“我点评的这个产品在 0-10 的评级尺度下得到了 9 的评分。”

我提到点评微数据可以影响搜索结果列表了吗？哦，是的，它可以。以下是 Google Rich Snippets 工具解析我们的点评微数据的结果（*http://www.google.com/webmasters/tools/richsnippets?url=//diveintohtml5.org/examples/review-plus-microdata.html*）：

```
Item
  Type: http://data-vocabulary.org/Review
  itemreviewed = Anna's Pizzeria
  rating = 4
  summary = New York-style pizza right in historic downtown Apex
  description = Food is top-notch. Atmosphere is just right ...
  address = Item(__1)
  geo = Item(__2)
  reviewer = Mark Pilgrim
  dtreviewed = 2010-03-31

Item
  Id: __1
  Type: http://data-vocabulary.org/Organization
  street-address = 100 North Salem Street
  locality = Apex
  region = NC
  postal-code = 27502
  country-name = USA

Item
  Id: __2
  Type: http://data-vocabulary.org/Geo
  latitude = 35.730796
  longitude = -78.851426
```

图 10.3 展示了我的点评信息在搜索结果页里可能的样子。

Anna's Pizzeria: review
★★★★☆ Review by Mark Pilgrim - Mar 31, 2010
Excerpt from the page will show up here.
Excerpt from the page will show up here.
diveintohtml5.org/examples/review-plus-microdata.html - Cached - Similar pages

图 10.3　搜索结果示例：带微数据的点评

尖括号并不会打动我太多，但我不得不承认，这很酷。

10.8 扩展阅读
Further Reading

微数据的资源：

- 微数据实战演练场（*http://foolip.org/microdatajs/live/*）。
- HTML5 微数据规范（*http://bit.ly/ckt9Rj*）。

Google Rich Snippets 的资源：

- “关于 rich snippets 和结构化数据”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170*）。
- “人物”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146646*）。
- “企业和组织”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861*）。
- “事件”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=164506*）。
- “点评”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146645*）。
- “点评评级”（*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=172705*）。
- Google Rich Snippets Testing Tool（*http://www.google.com/webmasters/tools/richsnippets*）。
- Google Rich Snippets Tips and Tricks（*http://knol.google.com/k/google-rich-snippets-tips-and-tricks*）。

全方位特性检测指南

The All-in-One Almost-Alphabetical Guide to Detecting Everything

觉得疑惑？请阅读第 2 章（HTML5 特性检测），以了解一些概念。想要一个全功能的库？试试 Modernizr（*http://www.modernizr.com*）。

元素列表

List of Elements

<audio>

http://bit.ly/cZxI7K

```
return !!document.createElement('audio').canPlayType;
```

MP3 格式的<audio>

http://en.wikipedia.org/wiki/MP3

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mpeg;').replace(/no/, ''));
```

Vorbis 格式的<audio>

http://en.wikipedia.org/wiki/Vorbis

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/ogg;
          codecs="vorbis"').replace(/no/, ''));
```

WAV 格式的<audio>

http://en.wikipedia.org/wiki/WAV

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/wav;
          codecs="1"').replace(/no/, ''));
```

AAC 格式的<audio>

http://en.wikipedia.org/wiki/Advanced_Audio_Coding

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mp4;
          codecs="mp4a.40.2"').replace(/no/, ''));
```

<canvas>

参见第 4 章

```
return !!document.createElement('canvas').getContext;
```

<canvas>文本 API

参见第 63 页“文本”一节

```
var c = document.createElement('canvas');
return c.getContext && typeof c.getContext('2d').fillText == 'function';
```

<command>

http://bit.ly/aQt2Fn

```
return 'type' in document.createElement('command');
```

<datalist>

http://bit.ly/9WVz5p

```
return 'options' in document.createElement('datalist');
```

<details>

http://bit.ly/cO8mQy

```
return 'open' in document.createElement('details');
```

<device>

http://bit.ly/aaBeUy

```
return 'type' in document.createElement('device');
```

<form>约束验证

http://bit.ly/cb9Wmj

```
return 'noValidate' in document.createElement('form');
```

<iframe sandbox>

http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox

```
return 'sandbox' in document.createElement('iframe');
```

<iframe srcdoc>

http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox

```
return 'srcdoc' in document.createElement('iframe');
```

<input autofocus>

参见第 148 页“自动聚焦”一节

```
return 'autofocus' in document.createElement('input');
```

<input placeholder>

参见第 147 页“占位文本”一节

```
return 'placeholder' in document.createElement('input');
```

<input type="color">

http://bit.ly/9HkeNn

```
var i = document.createElement('input');
i.setAttribute('type', 'color');
return i.type !== 'text';
```

<input type="email">

参见第 150 页“Email 地址”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'email');
return i.type !== 'text';
```

<input type="number">

参见第 153 页“数字类型输入框：数字选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'number');
return i.type !== 'text';
```

<input type="range">

参见第 155 页“数字类型输入框：滑块”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'range');
return i.type !== 'text';
```

<input type="search">

参见第 158 页“搜索框”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'search');
return i.type !== 'text';
```

<input type="tel">

http://bit.ly/bZm0Q5

```
var i = document.createElement('input');
i.setAttribute('type', 'tel');
return i.type !== 'text';
```

<input type="url">

参见第 151 页“Web 地址”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'url');
return i.type !== 'text';
```

<input type="date">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'date');
return i.type !== 'text';
```

<input type="time">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'time');
return i.type !== 'text';
```

<input type="datetime">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime');
return i.type !== 'text';
```

<input type="datetime-local">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime-local);
return i.type !== 'text';
```

<input type="month">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'month');
return i.type !== 'text';
```

<input type="week">

参见第 156 页“日期选择器”一节

```
var i = document.createElement('input');
i.setAttribute('type', 'week');
return i.type !== 'text';
```

<meter>

http://bit.ly/c0pX0l

```
return 'value' in document.createElement('meter');
```

<output>

http://bit.ly/asJaqH

```
return 'value' in document.createElement('output');
```

<progress>

http://bit.ly/bjDMy6

```
return 'value' in document.createElement('progress');
```

<time>

http://bit.ly/bI62jp

```
return 'valueAsDate' in document.createElement('time');
```

<video>

参见第 5 章

```
return !!document.createElement('video').canPlayType;
```

<video> 字幕

http://bit.ly/9mLiRr

```
return 'track' in document.createElement('track');
```

<video poster>

http://bit.ly/b6RhzT

```
return 'poster' in document.createElement('video');
```

<video> WebM 格式

http://www.webmproject.org

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/webm; codecs="vp8,
          vorbis"').replace(/no/, ''));
```

<video> H.264 格式

参见第 84 页“H.264”一节

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/mp4; codecs="avc1.42E01E,
          mp4a.40.2"').replace(/no/, ''));
```

<video> Theora 格式

参见第 84 页“Theora”一节

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/ogg; codecs="theora,
          vorbis"').replace(/no/, ''));
```

contentEditable

http://bit.ly/aLivbS

```
return 'isContentEditable' in document.createElement('span');
```

Cross-document messaging

http://bit.ly/cUOqXd

```
return !!window.postMessage;
```

Drag and drop

http://bit.ly/aN0RFQ

```
return 'draggable' in document.createElement('span');
```

文件 API

http://dev.w3.org/2006/webapi/FileAPI/

```
return typeof FileReader != 'undefined';
```

Geolocation

参见第 6 章

```
return !!navigator.geolocation;
```

历史

http://bit.ly/9JGAGB

```
return !!(window.history && window.history.pushState &&
window.history.popState);
```

本地存储

http://dev.w3.org/html5/webstorage/

```
return ('localStorage' in window) && window['localStorage'] !== null;
```

微数据（Microdata）

http://bit.ly/dBGnqr

```
return !!document.getItems;
```

离线 Web 应用程序

参见第 8 章

```
return !!window.applicationCache;
```

“服务器已发送”（Server-sent）事件

http://dev.w3.org/html5/eventsource/

```
return typeof EventSource !== 'undefined';
```

会话存储（Session storage）

http://dev.w3.org/html5/webstorage/

```
try {
  return ('sessionStorage' in window) && window['sessionStorage'] !== null;
} catch(e) {
  return false;
}
```

SVG

http://www.w3.org/TR/SVG/

```
return !!(document.createElementNS && document.createElementNS
('http://www.w3.org/2000/svg', 'svg').createSVGRect);
```

text/html 中的 SVG

http://hacks.mozilla.org/2010/05/firefox-4-the-html5-parser-inline-svg-speed-and-more/

```
var e = document.createElement('div');
e.innerHTML = '<svg></svg>';
return !!(window.SVGSVGElement && e.firstChild instanceof window.SVGSVGElement);
```

WebSimpleDB

http://dev.w3.org/2006/webapi/WebSimpleDB/

```
return !!window.indexedDB;
```

Web Sockets

http://dev.w3.org/html5/websockets/

```
return !!window.WebSocket;
```

Web SQL Database

http://dev.w3.org/html5/webdatabase/

```
return !!window.openDatabase;
```

Web Workers

http://bit.ly/9jheof

```
return !!window.Worker;
```

Undo

http://bit.ly/bs6JFR

```
return typeof UndoManager !== 'undefined';
```

扩展阅读

Further Reading

规范和标准：

- HTML5（*http://bit.ly/bYiOQP*）。
- 地理位置（*http://www.w3.org/TR/geolocation-API/*）。
- “服务器已发送”事件（*http://dev.w3.org/html5/eventsource/*）。
- WebSimpleDB（*http://dev.w3.org/2006/webapi/WebSimpleDB/*）。
- Web Sockets（*http://dev.w3.org/html5/websockets/*）。
- Web SQL Database（*http://dev.w3.org/html5/webdatabase/*）。
- Web 存储（*http://dev.w3.org/html5/webdatabase/*）。
- Web Workers（(*http://bit.ly/9jheof*）。

JavaScript 库：

- Modernizr（*http://www.modernizr.com*），一个 HTML5 特性检测库。

索引
Index

A

B

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

关于作者

Mark Pilgrim 是 Google 公司的高级开发人员，专门研究开源技术和开放标准。Mark 是多本技术书籍的作者，包括“Dive Into Python”（APress）和“Dive Into Accessibility”。后者是一本关于 Web 亲和力的免费在线教程。他和妻子、两个孩子及一条毛茸茸的大狗生活在北卡罗来纳。

封面说明

本书封面上的动物是一只臆羚（Rupicapra rupicapra），属于山羊或羚羊种，原产于欧洲山脉地区，包括喀尔巴阡山脉、亚平宁山脉、塔特拉、巴尔干、高加索及阿尔卑斯山脉。臆羚也可见于新西兰，它们于 1907 年由奥地利皇帝弗朗茨·约瑟夫一世引入。

臆羚生活在相对较高的海拔，已适应了陡峭崎岖的岩石地形。它们可以长到 75 厘米高，体重 20~30 公斤（新西兰的个体较其欧洲同胞通常要轻大约 20%）。雄性和雌性都长有向后弯曲的短角，其毛色夏季呈深棕色，冬季则呈浅灰色。许多臆羚具有典型的白色面部和臀部，眼睛下部以及沿着背部都有黑色条纹。

成年雄性臆羚大多独居，只是一年一次聚集争斗，以吸引未交配雌性个体的注意。而雌性则同它们的幼崽一起群居。所有种类的羚羊都是流行的狩猎动物；它们的肉被视为美味佳肴，皮毛也因异常柔顺保暖而著称。此外，从这种羚羊脖子后面取下的一绺毛，称为 gamsbart，一直是整个阿尔卑斯山区国家传统的帽子饰物。不过这个传统在当代可能比较难维持了，因为欧盟已经规定某些羚羊的亚种是受保护物种。与此相反，新西兰的有关保护部门则鼓励狩猎臆羚，希望能限制此种动物对当地原生植物种群的影响。

本书封面图由 J·G·伍德的 Animate Creation 创作。

北京博文视点（www.broadview.com.cn）资讯有限公司成立于2003年，是工业和信息化部直属的中央一级科技与教育出版社——电子工业出版社（PHEI）下属旗舰级子公司，在六年的开拓、探索和成长中，已成为中国颇具影响力的专业IT图书策划和服务提供商。

六年来，博文视点以开发IT类图书选题为主业，励精图治、兢兢业业，打造了一支团结一心的专业队伍，并形成了自身独特的竞争优势。一直以来，博文视点始终以传播完美知识为己任，用诚挚之心奉献精品佳作，年组织策划图书达300个品种，同时开展相关信息和知识增值服务，赢得了众多才华横溢的作者朋友和肝胆相照的合伙伙伴，已经成为IT图书领域的高端品牌。

我们的理念：创新专业图书服务体制；培养职业策划图书服务队伍；打造精品图书品牌；完善全面出版服务平台。

我们的目标：面向IT专业人员的出版物提供相关服务。

我们的团队：一个整合了专业技术人员和专业服务人员的团队；一个充满创新意识和创作激情的团队；一个不断进取、追求卓越的团队。

我们的服务：善待作者　尊重作者　提升作者

我们的实力：优秀的专业编辑队伍
全方位立体化的强大的市场推广平台
实力雄厚的电子工业出版社的渠道平台

“走出软件作坊独辟蹊经　人道编程之美，
追踪加密解密庖丁解牛　精雕夜读天书。”

路漫漫其修远，博文视点愿与所有曾经帮助、关心过我们的朋友、作者、合作伙伴携手奋斗。未来之路，不可限量！

地址：北京市万寿路173信箱电子工业出版社博文视点资讯有限公司
邮编：100036　**总机：**010-88254356　**传真：**010-88254356-802
武汉分部地址：武汉市洪山区吴家湾湖北信息产业科技大厦1402室
邮编：430074　**总机：**027-87690813　**传真：**027-87690013
欢迎投稿：bvtougao@gmail.com
读者邮箱：reader@broadview.com.cn
博文视点官方博客：http://blog.csdn.net/bvbook
博文视点官方网站：http://www.broadiew.com.cn

反侵权盗版声明