HTML5 触摸界面

提升网站速度，创造惊人的用户体验

设计与开发

【美】Stephen Woods 著

覃介右 谷 岳 译

人民邮电出版社

北 京

内容提要

本书专注于触摸界面的开发，内容结构和优化网站思路大概一致。上半部分涵盖了能使各种网站，特别是移动网站变快的基本概念。第 2 章和第 3 章告诉你如何创建一个简单的网站，并使其加载速度快。第 4 章可以帮助你使用缓存来提高用户再次访问时的速度。第 5 章是关于要摒弃一次性的页面加载方式和重构应用程序以使实际上和感觉到的性能都达到最优的。书的后半部分是专门讲触摸界面的，特别是尽可能地让它们更平稳和快速。本书适合具有一定经验的 Web 开发者阅读参考。

献给“生鱼片”——最棒的猫

致谢

感谢 Jeff Riley、Nancy Peterson、Michael Nolan，还有 Peachpit 的工作人员，你们推动了本书的诞生，并让文字变得通顺。感谢 Nicholas Zakas 对本书非常细致、深思熟虑的批评，还有他在 Yahoo! 的指导。

还要感谢 Stoyan Stefanov 的终审和他在技术文章写作领域上的宝贵经验。同样感谢 Guy Podjarny 为本书付出的时间和做过的研究。

如果没有我的经理 Ross Harmes 以及 Flickr 前端团队其他同事们的支持，绝不会有这本书的诞生。

感谢 Benjamin 为我展示了移动设备的众多用途。

最后，感谢 Elise 容忍我每天晚上一连几个小时地盯着电脑和一堆手机。

前言

在我写作时，11.42% 的网页流量来自移动设备（根据 StatCounter.com 的统计数据）。这个数字一年前是 7%，三年之前是 1.77%。虽然桌面设备还会陪伴我们一段时间，但是 Web 的未来是在移动设备上的。

对于 Web 开发者，支持移动设备是自 21 世纪初的 Web 标准革命以来最大的变革。移动设备都搭载了支持 HTML5 的完全现代的浏览器。它们还装有容量较小的内存和缓慢的 CPU，经常通过高延迟的网络来链接。而且它们几乎全都有触摸界面。

为移动设备开发就是为触屏开发。许多桌面 Web 开发中的技巧对于移动 Web 依然适用，不过有些却是完全不同——弄清楚这些不同会有一定难度。我写这本书来帮助你去弄清楚它们。

谁适合读这本书

这本书是为两类读者写的：

- 经验丰富的 Web 开发者，但从未开发过移动或触摸界面的网页，想要学习
- 一直致力于让他们的移动版网站更优秀的移动站点开发者

这本书不适合绝对的初学者。你需要有 Web 前端的背景知识：HTML、CSS 和 JavaScript。对 HTML5、CSS3 的新接口和功能有一定经验会更好。

最重要的是，这本书适合那些并不觉得他们的移动网站已经足够好的人。如果你想要建设一个迅速而流畅的网站，这本书再适合你不过。

你将能学到什么

本书专注于触摸界面的开发，内容结构和优化网站思路大概一致。上半部分涵盖了我认为能使各种网站，特别是移动网站变快的基本概念。第 2 章和第 3 章告诉你如何创建一个简单的网站，并使其加载速度快。第 4 章可以帮助你使用缓存来提高用户再次访问时的速度。第 5 章是关于要摒弃一次性的页面加载方式和重构应用程序以使实际上和感觉到的性能都达到最优的。书的后半部分是专门讲触摸界面的，特别是尽可能地让它们更平稳和快速。随着层层深入，本书也逐渐复杂深奥。如果你觉得后面的章节太难了，先尝试在工作中运用你所学到的东西，再回头来看我在本书末尾介绍的一些想法。一个好用的移动网站不需要双指缩放的功能。

你需要用到什么

如果你想要在本书中学到更多东西，除了你的电脑外，你至少还需要一台触屏设备。如果只能有一台，我推荐 iOS6 或 Android 4 的设备。如果条件允许，上述两台都有是最理想的了。

开发移动网页时，设法利用尽可能多的设备。iOS 和 Android 模拟器无法取代真机。在写这本书时，我使用了一台搭载 Android 4.0.4 的 Samsung Galaxy S III、一台 iPhone 4、一台 iPhone 5、一台 iPad 1，还有一台 HTC 8X (Windows 8)。再用模拟器作为补充。

在 Flickr，我们拥有的设备和上述类似，不过还有一些 Android 平板和一台 Kindle Fire。

框架

本书没有用 jQuery 或是其他任何 JavaScript 框架。你会了解到一些专用的库，不过我们还是尽可能多的关注原生的 DOM API。这并不是说你应该避免使用框架或者远离框架！只是我想要确保你能理解这些东西最根本的原理。当你决定使用 jQuery mobile、Backbone.js、Zepto.js 或是其他框架时会更轻松，因为你理解其内部的工作原理。

理解原生 DOM API 的另一个巨大的好处是，当你在某个库中找到一个 bug 的时候，你可以自行修复并创建一个包含你修复代码的 pull request，从而造福整个社区。

配套网站

本书上的所有示例代码和最新的改动可以在配套网站：touch–interfaces.com 上找到。这些代码也在 GitHub 上做了镜像，在这你可以找到有关示例代码的 issue，还可以提交到：https://github.com/saw/touch–interfaces。

欢迎进入移动网页

网站是用 HTML、CSS 和 JavaScript 构建的。移动网站也是一样。只需要一个网页浏览器和一个文本编辑器就可以开始，但要更高效的话，我建议再多几个工具。

工具

最简单的过程，是用一个文本编辑器和桌面浏览器开发，然后在旁边准备一个触屏设备用来测试。

一个文本编辑器 & 一个 WebKit 内核的浏览器

我用适用于 Mac OS X 的 TextMate 2 (github. com/textmate/textmate)，不过任何其他的编辑器都没问题。

因为绝大多数的移动设备运行的是 WebKit 的浏览器，所以，你会发现 Chrome 或 Safari 是能提高效率的必备工具。虽然与真实设备上的测试会有些不同，但它简单易用，必不可少。

一个 Web 服务器

为了在真实的设备上测试网站，你需要通过无线局域网提供页面。在 Mac 上，我发现 MAMP (www.mamp.info) 是一个非常方便的工具，不过使用内置的 Apache Web 服务器同样可以。

一台触屏设备

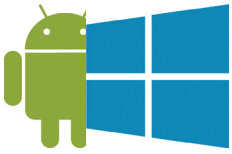
没有什么可以替代一台真实的设备。如果你能负担得起的话，我建议至少有一台近期的 Android 手机和 iOS 设备。如果你只买得起一台，找到愿意借手机给你测试一段时间的人，会有很帮助。

跨设备平台测试

不能想当然地认为所有的 Webkit 的浏览器都是一样的。你应该在 iOS 5、iOS 6、Android 2.3、Android 4.0、Android 4.1 (Chrome)、和 IE 10 里面测试你的 App。如果你不能直接使用这些设备，这里是如何对它们进行测试的指南。

IOS SAFARI

Apple 提供了一个相当好的模拟器 XCode。它可以模拟搭载 iOS 5 或 6 的平板和手机。它还支持结合 Safari 浏览器的远程调试。这真的是一个伟大的工具，假如你有一台 Mac，这是这堆工具中关键的一部分。XCode 在 Mac App 商店免费提供。



ANDROID

谷歌提供了几乎所有版本的 Android 的模拟器。这些都可以在 Android SDK（developer.android.com/SDK）中找到。只要有了 Android SDK，就可以分别下载各种 Android 版本的镜像。请记住，这些都是谷歌官方的版本，Android 版本在实际设备上可能有很大不同。

WINDOWS 8

微软确实提供了一台针对 Windows Phone8 模拟器，它在 SDK（dev.windowsphone.com/ en-us/downloadsdk）中。模拟器只能在 Windows 上运行。IE10 的移动版与桌面版是相同的，所以大部分 Debug 工作可以在桌面上进行，而不是在模拟器里面。

DEBUGGING

调试手机网站是一个苦差，但也有很多工具使其变得更容易。我提供了一些工具的列表，在配套网站上的附录 A 中。

作者简介

Stephen Woods 是 Flickr 的一名高级前端工程师。自 20 世纪末开始，他一直致力于研发 Web 用户界面。他从 2006 年开始在 Yahoo! 就职。在 Flickr 之前，他开发了为 Yahoo! 主页提供支持的 JavaScript 平台，还曾在 Yahoo! Personals 的 UI 团队工作。他是精通所有网站开发技术的专家，但他的主要兴趣是用网络技术制作响应式的用户界面。Stephen 曾在 SXSW 和 HTML5DevConf 就触屏交互发表演说，并曾被《.NET》杂志报道。

目录

第 1 章

移动设备概述

一切都因 iPhone 开始。在这之前，手机网站就是其桌面网站的简化版。极少数浏览器能够支持 JavaScript，大多数浏览器只是将网站转化成适合小屏幕而已。iPhone 上的 Safari 是一个真正的 Web 浏览器，它支持 JavaScript 和 CSS。不仅如此，它还是一个非常前沿的浏览器。为 iPhone 设计的网站可以使用一些新生的工具套件，即现在的 HTML5。如今，市面上有数百种的移动设备在售。它们千差万别，各有千秋，但它们有一个共同的特点：先进的 Web 浏览器和触摸界面。

1.1 触屏设备和桌面设备之间的差异

大多数触屏设备是移动设备。为桌面设备开发网站的开发者最担心的是它们的网页能否兼容各种浏览器。所以，开发者很大一部分的工作是理解各种浏览器之间的差异。

移动领域问题不在于各种浏览器，而在于不同的移动设备。当然，有几种不同的移动浏览器，并且它们之间的确存在一些差异，但大体上移动设备版本与其相应的桌面版本差异不大。移动设备和桌面设备真正差异在于以下浏览器的 4 个因素：

- 规格
- 计算能力不足
- 人们如何使用它们
- 触摸界面

1.1.1 规格

移动设备和台式电脑（或笔记本电脑）之间最明显的差异是尺寸。移动设备的尺寸比较小，所以搭载较小屏幕，且没有鼠标，大多数情况下也没有键盘。

触屏设备目前主要有三种规格：手机、小型平板和大型平板。大型平板包括 iPad（不包括 iPad mini）和各种 10 英寸的 Android 平板，如 Nexus10。小平板屏幕一般约为 7 英寸，如 Nexus7、Galaxy Note 和 iPad mini。

所有这些设备上的浏览器始终都是全屏的。用户“调整”浏览器大小的唯一途径是通过旋转移动设备在横向和纵向模式之间进行切换。只为一种尺寸的设备优化并不困难，困难的是要确保你的网站适用于所有的设备。

1.1.2 移动设备的计算能力不足

写本书时，最强大的触摸屏设备是第四代 iPad。它是性能强大的机器。根据相同的标准，它的性能与 2004 年发布的 Power Mac G5——最后一代摩托罗拉版相同。

可以肯定地说，大多数用户并没有使用最新和最强大的移动设备。大多数运营商提供“合约”手机，如 iPhone 4 和三星 Galaxy Exhilarate，这些手机的运行速度要慢得多。它们的 CPU 性能和 20 世纪末、21 世纪初最快的台式电脑差不多。iPhone 3GS 仍然被广泛使用，它拥有单核处理器、256 MB 的内存，CPU 性能大致相当于最后一代与 CRT 显示器搭配销售的 iMac。虽然移动设备的性能仍在继续增长，但与一般桌面电脑的性能相比，它们还是太慢了。值得高兴的是，大部分近期发布的设备都拥有独立的 GPU。一个 iPhone 4 在性能上可以被认为是一台装有强大图形处理器，但性能稍逊的电脑。

1.1.3 人们使用触屏设备的方式大相径庭

人们使用手机和桌面电脑的习惯不同。人们使用桌面电脑一般是为了完成特定的工作，所以可能不会做很多其他的事情。至于移动设备，不论是手机还是平板电脑，通常人们都是在做别的事情的间隙来使用它们。他们可能正在乘坐公交车、排队等候或只是在参加一个无聊的会议。

例如，你搭建了一个财经新闻网站。在桌面电脑上，用户可能会花费一段时间点击几次链接，然后用几分钟的时间阅读一篇文章。而移动用户可能正在做一件别的事情，只会在空闲时的几秒时间来完成上面的操作。桌面网站的速度一般不慢，但桌面用户的容忍时间更长。如果您的网站载入时间有一点长，用户就会切换到别的标签去看其他的东西，然后再回来。但在手机上，用户永远不会给你这样的机会。

移动设备就是触屏设备

所有在售的新型智能手机和平板电脑有着一个非常重要的共同特点：它们都具有触摸界面——在大多数情况下，还可以多点触控。合适的大小、美观的屏幕以及其他的先进功能，这些功能相对于使用触摸屏这样的革命性革新，都只是微小的改进而已。

1.2 广泛使用的设备

在 2007 年，iPhone 刚出现时，没有任何一部其他的手机拥有真正能触摸感知的 Web 浏览器。也没有任何支持多点触控的设备。其他的智能手机大多数使用手写笔进行交互，当时市场上的几种平板电脑也是一样。今天，市面上流通的有数百种触屏设备。我们不可能了解每一种新设备，不过它们可以根据操作系统、规格、搭载的 Web 浏览器将它们分为几类。

1.2.1 操作系统

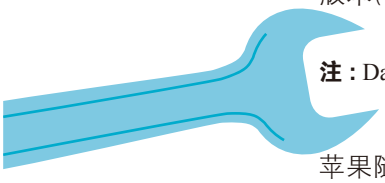
如今市场上有四种主要的操作系统适用于触屏设备：

- iOS
- Android
- Blackberry OS
- Windows 8

Android 是目前手机上最流行的操作系统。根据 International Data Corporation (IDC) 的数据， 2012 年第三季度智能手机出货量的 75% 运行的是 Android 系统，14.9% 运行的是 iOS 系统。其他操作系统的市场份额只有个位数（这不包括的 Windows Phone 8，因为它在 2012 年 10 月才发布）。

iOS

iOS 是所有苹果手机和平板运行的操作系统。因为苹果控制着硬件，所以所有的 iPad 和 iPhone 能以相同的方式运行，只有大小和处理能力上的差异。苹果推出软件更新时无需依赖运营商，这意味着，苹果设备更容易运行最新版本的操作系统。iOS 开发者，David Smith，发现他开发的应用程序的用户中，有 79.2% 正在使用的 iOS 6 或更高版本（相对 2012 年 12 月）的操作系统，有 94% 的人正在使用 5.1 或更高版本的操作系统。



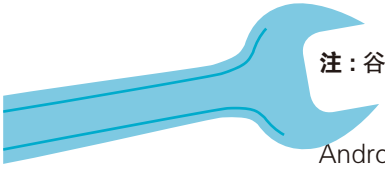
注：David Smith 会在他的网站（david-smith.org/iosversionstats/）上更新这些统计数据。

苹果随着操作系统的升级同步升级浏览器，它不允许用户安装不同内核的浏览器。至少现在，可以认为 iOS 用户主要安装着上一个较新版本的 iOS 和 iOS Safari。

ANDROID

Android 的情况很不一样。Android 是开源的，手机厂商有权自由定制成他们认为合适的样子，所以在手机之间实际的用户界面差异很大。

Android 有三个主要的版本：2、3 和 4。Android 3.0（蜂巢）是平板专用操作系统。大约 50% 的 Android 手机运行着 2.3.x（姜饼）。最新版本的 Android 4.1（果冻豆）并没有被广泛使用，在 2013 年 1 月，只有不到 10% 的设备在运行它，不过这一数字正在迅速增长。时下的“旗舰”手机，如 Samsung Galaxy SIII，仍然搭载着 Android 4.0.x（冰淇淋三明治）系统出售。 Android 4.1 同时支持平板电脑和手机。



注：谷歌在 developer.android.com/about/dashboards/ 提供了各版本的统计

Android 包括一个默认的浏览器，而且允许让用户安装不同的浏览器。4.1 版之后，Android 上的默认浏览器改为了 Chrome。

Kindle Fire

Amazon Kindle Fire 运行着高度定制版的 Android，这个定制后的版本不包括任何原生的 Android 应用程序，并拥有自行研发的浏览器，称为 Silk。值得注意的是，Silk 因其在客户端设备和 Amazon 云上的运行能力而知名。这样可以显著地提高性能，尤其在高延迟的网络链接下。这对开发者有一些影响，我们将在第 4 章“提高下一次的访问速度”中阐述。

BLACKBERRY OS

Blackberry 手机是最早真正获得成功的智能手机。但到了 2012 年 12 月，其市场份额已经迅速减少至大约 4.3%。Blackberry 也出售平板，称为 Playbook。这两种设备使用相同的基于 WebKit 的网页浏览器。尚存的非触摸屏式的黑莓设备占据着智能手机市场的长尾。

WINDOWS

Windows Phone 8 包含了微软第一个完全触摸式的浏览器。这款手机配备了内置的 Internet Explorer 10，也就有了创建运行于 HTML5 环境的应用程序的能力。

1.2.2 设备规格

桌面和移动设备之间最显著的差异是规格。移动设备有两种主要的规格：手机和平板电脑。

手机

最常见的触摸屏设备是智能手机。除了极少数例外，它们都具有相同的基本规格：呈矩形并有纵向的触摸屏。开发手机网站时要考虑的最重要的事就是小尺寸的屏幕。需要保证交互元素的尺寸大到可以用手指来操控，并且内容能在小屏幕上展现得合适。

平板电脑

平板电脑的类型更多样化。它们中从微型的 Galaxy Note（5.3 英寸屏幕）到 13 英寸的 Windows 可变形平板电脑。有些平板电脑，比如 iPad，通常在纵向方向使用。其他的则是在横向模式下使用。就像手机一样，所有平板电脑都装有全屏的浏览器。

尽管有着这样的区别，但实际使用中的平板电脑是差不多的：iPad 的销量超过了所有的竞争者，并稳定赢利。唯一拥有两位数市场份额的其他制造商是三星。平板电脑最流行的规格是 10 英寸（如 iPad）和 7 英寸（如 iPad Mini 和 Kindle Fire）。

硬件图形加速

正如之前所提到的，一部 iPhone 在性能上可以被认为是一台装有不错的显卡的性能稍逊的电脑。各个设备的 CPU 和内存各异，图形硬件也是这样。在 Android 3.0 之前，浏览器完全没有图形硬件加速功能。这意味着，某个动画的效果可能会在 iPhone 5 上流畅，但在运行 Android 2.3.3 的旧手机上不会同样流畅。

目前所有的移动设备在绘图上，都使用所谓的“共享内存”。与桌面电脑上的显卡不同，移动设备上的图形硬件没有专用的内存，只是共享系统内存。这意味着，随着越来越多的东西被添加到图形存储器，系统可用的内存会越来越少。在我们随后讨论图形加速时，牢记这一点很重要。

1.2.3 浏览器

幸运的是，浏览器的情况远没有移动设备复杂。WebKit 无疑是最流行的移动浏览器内核，它支撑着所有的苹果、Android 和黑莓设备的默认浏览器。Windows 手机运行 IE 10。最流行的另类手机浏览器是 Opera Mobile，但只有不到 1% 的市场份额。

WebKit

WebKit 起源于名 Konqueror 的开源浏览器的引擎。苹果将它进一步开发为 WebKit，即 Safari 背后的浏览器引擎。WebKit 的是一个非常强大的引擎，也是第一代 iPhone 如此成功的原因之一。在 Android 4.1 之前，一个通常被称为“Android Browser”的基于 WebKit 的浏览器被安装在了 Android 设备上。4.1 时，它被同样是 WebKit 内核的“Chrome for Android”浏览器取代。

基于 WebKit 内核的浏览器比较相似，但又不完全相同。不同的 WebKit 的浏览器之间的渲染行为非常近，不过因为存在一些小的分支，所以在功能支持上有很大不同。当它们出现差异时，我会提醒大家。

iOS Safari

iOS 版的 WebKit 是 iOS 上唯一允许的浏览器引擎。它通常通过 Safari 浏览器运行，也可以在原生应用程序中通过 Cocoa UIWebView 类运行。

Android Browser

Android Browser，即在 Android 中名为“浏览器”的应用，是 Android 内置的浏览器。它是基于 WebKit 的，但并不是 Chrome。Android 浏览器的功能在不同版本中有很大差异。令人惊讶的是，并不是越新版本的浏览器有越多的功能。Android 2.3.3 的浏览器在某些方面比 4.0.1 功能更全面。我稍后会指出它们之间的差异。

Android 版的 Chrome

Android 版的 Chrome 通常比桌面版 Chrome 的稳定分支落后一个版本。在 Android 4.1 Chrome 已取代了先前的原生浏览器。

Windows 版的 IE10

IE10 不同于所有先前版本的 Internet Explorer，它与 WebKit 的功能相同或相近。IE10 有完全不同的触摸屏事件 API，但功能是相同的。

其他浏览器

还有许多其他的浏览器。最流行的是 Opera Mobile 和 Firefox 移动版，它们都适用于 Android。这两种浏览器都非常强大，能实现的功能与 WebKit 大致相同。我会提醒大家它们语法的不同点。

1.3 HTML5

所有移动浏览器所共有的，为支持先进浏览器而存在的功能，被统称为 HTML5。那究竟什么是 HTML5？

严格地说，HTML5 是指网页超文本技术工作小组（WHATWG）规定的标记语言和 API 集合。它通过定义能满足网络应用需求的功能，取代了 HTML4，在 HTML5 提出伊始，就已经超越了 HTML 最初以文档为中心的模型。

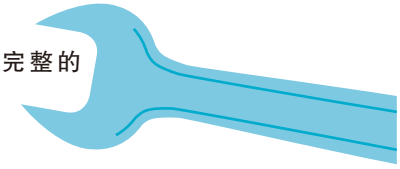
实践中，HTML5 已经成为一套标准化技术和新兴技术的代名词。这个新兴技术极大地拓展了传统的“Web 堆栈”（HTML，CSS 和 JavaScript）。

1.3.1 规范

不像以前的 HTML 规范，WHATWG 决定不定义 HTML5 的正式版本。相反，HTML5（现在正式更名为“HTML”）是一个动态的标准，允许在一个临时的基础上增加新的 API 和功能。万维网联盟（W3C）正在努力创造一个官方的、规范的版本。

由于 HTML5 是一个动态的标准，那么浏览器制造商增加新功能时，该功能可能还没有标准化。这意味着，不同浏览器之间在功能的整体上是类似的，但可用的特定功能和实现有差异。

知道它们成为规范的原因是固然是有用的，但现在，开发人员需要知道不同浏览器各有什么特点，以及如何处理它们。在这本书中，我会注意哪些功能是已部分标准化的，哪些功能是新出现的。



注：HTML5 的“动态标准”规范写在了 www.whatwg.org 上，完整的 W3C 官方规范写在了 w3.org 上。

1.3.2 组件

我们所说的 HTML5，它的能力来自广泛的技术组件。用 CSS3（以及来自浏览器厂商的尚未列入标准的其他 CSS 功能）创造动态和优美的界面，不会以牺牲响应效率为代价。

在这本书中，我把 HTML5 作为更广泛的组件来考虑。不幸的是，目前完全“遵循规范”，不足以给我们提供能创建强大触屏交互的工具。未来，我希望有更多的标准化的功能和新的 API。与网络相关的事总是让人兴奋，包括移动网络，因为它总是在变化，我们作为开发人员也必须随之改变。

1.4 神秘谷，是什么让触摸界面反应灵敏?

我有个两岁半的儿子。他从一岁开始就能解锁 iPad 并找到他想要的应用程序。他还不会说话，就会解锁 iPad。我与其他家长交流意见，我的儿子并不是智商超群。触摸界面给了孩子们直观的感受：触摸和移动东西，是人类做的最基本的事情之一。触摸界

面直观又强大。但它也很容易令人失望。

有一个理论在机器人学里被称为“神秘谷”：机器人看上去越像人，就越对我们有吸引力，但当机器人的外形快要逼近真人的时候，会让我们觉得很奇怪，毛骨悚然。“谷”是机器人与人类相似程度与人类舒适度等级的映射关系的拐点。

触摸界面让人感觉很自然，感觉像在移动周围真实的物体。不正确的运用时，给人的感受不是慢，而是失望。直接操纵的假象被打破时，触摸界面也不再让用户感觉自然。这种失望感仿佛掉进了神秘谷，用户不再感觉自然，而是感觉怪异。

1.4.1 传送长奥布莱恩和直接操纵

像我这一代的许多人一样，我第一次触摸界面是通过《星际迷航：下一代》。如果你没有看过的话，可以想象一下：船上所有的控制面板为触屏，实际上是多点触控界面。显然，这样做的原因完全是因为预算。电影的创作者希望在所有的控制面板中表现大量的复杂性，但一个以脱口秀节目的预算支付不起建造这些设备的费用。于是，他们想出了在胶片上印出界面，并在后面用灯光，创造出触屏的外观的方法。

传送长奥布莱恩，也就是操纵这艘飞船的人，用三指滑动手势来发动飞船。我觉得那个手势真的很有趣。界面上有三个滑块，演员会自然想到去触摸和拖动滑块，就好像操纵真正的物体。这是人们所期望使用触摸界面的方法。当他们触摸屏幕时，能凭直觉感到，与它进行交互，就像在移动一个真正的物体。

苹果的触摸接口指南称这个概念为“直接操纵”，而不是通过控制器来控制。在理想情况下，触屏界面给用户的印象是，直接操纵交互的东西。

你会注意到，当传送长奥布莱恩的手指移动时，小指示器也在随之移动。这就能保持直接操纵的假象。小指示器也提供反馈，表明计算机正在记录他的动作。

1.4.2 用户反馈

我会想，用户界面只需让人感觉快，而无需真的快。只需及时响应——立即响应用户，让他知道有反应了。我最喜欢的例子是 TiVo。这不是我现在的高清电视，而是在 1999 年发售的 TiVo。那个电视盒有一个 54 MHz 的 CPU 和仅有的 16 MB 的 RAM。虽然 TiVo 有特殊的硬件对视频编解码，但从用户点击一个影片到开始播放可能还是需要一段相当长的时间。TiVo 的投诉有很多，但从来没有人抱怨它的速度慢。这是因为 TiVo 有用户熟悉的哔嘣声。

当用户点击一个影片时，能立即听到声音。我不知道工程师花费了多少时间才确保声音出现的及时性，这恰恰就是高明的地方。声音让用户能立即知道机器已经收到了他的请求。

在网站的用户界面上，这种即时反馈是同样重要的。对于桌面电脑的网站，大多数人和设备的交互是分离的：点击后对应的效果出现。对于触摸界面，许多交互是连续的，因为它们是手势交互。当用户在屏幕上用手势操作时，不能等到手势完毕再做反馈，因为根据用户的了解，它们甚至会认为手势没有生效！

我们来谈谈滑动。滑动是指在屏幕上移动手指来执行一些操作。理想的情况下，界面元素应该随着用户的滑动而移动。如果是在页面之间滑动，整个页面应该随着手指移动。如果界面不动，那么在滑动结束之前，就无法知道到底发生了什么。没有反馈的滑动手势会像一个键盘，在你输入一个字母之前，它不会有任何反馈。手势不能等到完成时才反馈。为了让界面感觉反应迅速，还必须是连续的反馈，当用户的手指移动的时候，界面也应该移动。

如果用户在用手势操作，即使在操作完毕时，界面也必须跟着移动。如果界面在手势操作时中途停止移动了，就感觉像死机了一样。

例如，如果你创建了一组幻灯片，用户可以在幻灯片之间滑动，当用户到达最后一张幻灯片时，你不会希望手势响应停止，用户会以为界面卡死了。而是让用户继续滑动，当用户松手时，应该快速地将前一张放进来。用户得到了她的手势已经被机器接收到了的反馈，只是幻灯片已经放完了。这是苹果如此严重地依赖在手势结束时的“迅速返回”的原因。这是告知用户已经到了最后，而又不破坏手势喻意的唯一办法。

规范

每个用户界面有它的规范。在桌面上有窗口、按钮、滚动条、关闭框。在移动界面，有一组新的规范。先行者苹果创建了两套 UI 规范。

你不需要一切都与原生的 iOS 应用程序完全一样。但知道一些手势和元素在移动领域的意义十分重要：不要使用滑动手势来表示选择，因为原生应用程序中这表示删除。不要更改操作系统通常会保留的手势（比如按下并保持一段时间表示呼出内容菜单，双指捏紧为缩小，双击为放大），除非你重新实现相同的基本功能。（你会在第 10 章中了解更多有关“滚动和滑动”的知识，在第 11 章中了解更多有关“双指缩放和其他的复杂的手势”的知识。）

1.5 总结

触屏设备总体上在不断变化，大部分 Web 浏览器之间的差异是很小的。虽然在性能上的差异较大，但只用 HTML5 就足以在触屏设备上做出优秀的用户界面。

鉴于触屏设备的性质，速度快是至关重要的。为了让它看起来更快，最重要的是要及时反馈用户。

第 2 章

创建一个简单的内容型网站

内容型网站组成了 Internet 的很大一部分。毕竟，网站就是为了浏览文本而建立的，人们在网上做的最多的事情就是阅读文字。内容型网站在触屏设备上保持良好的体验固然很重要，但实际上许多内容型网站做得还不够好。

在本章中，我们将为客户——Awesome 鸟类基金会，建立一个关于加州鸟类的网站。鉴于人们普遍无法在家中观赏鸟类，我们网站目标就是通过移动设备和桌面电脑，使之成为可能。客户想让数以百万计的网站文章被链接到 Twitter 上，用户通过点击即可阅读关于鸟类的精彩文章。我们不希望读者在网站加载完成之前流失，必须争分夺秒。在对服务器端进行一切优化之后，还需要网页能尽快加载。

显然，网站在高延迟的 3G 网络下的加载速度会比在办公室的 100MB 网络下慢。但是，一旦开始接收数据，就必须立刻吸引住用户。我们将从基本要素开始制作这个网站：文档对象模型（DOM）和层叠样式表（CSS）。

设计师给了我们两个模拟样式图：一个桌面版本（图 2.1）和一个移动版本（图 2.2）。

图 2.1 加州鸟类网站的桌面版样式图

图 2.2 加州鸟类网站的移动版样式图

我们来看这些模拟的样式图。很明显，开发这个网站的难度不大。但不要忘了，此网站在移动设备和桌面设备上都要能够使用。作为开发者，我们喜欢尽可能地减少代码量，因为我们知道，写的所有代码都要自己维护。同时，我们还希望网站在任何屏幕上的体验都是最好的。

理想的情况下，我们只写一次代码就可以用在所有的设备上。在许多情况下，我们完全可以利用 CSS3 的新特性——媒体查询来避免分支。如同我们在第 1 章“移动设备概述”中讨论的，移动设备与桌面设备相比，性能还有一定差距。一些美好但复杂的交互，在桌面设备上可以良好地运行，在比较差的移动设备上可能会卡死。确实需要对每种情况单独处理。但作为一般规则，简单的内容型网站应该能在桌面设备和移动设备上共用一些代码。加州鸟类网站就是一个完美的例子。

2.1 选择一个观念：移动优先或置后

短语“移动优先”是指在 Web 开发中的几个趋势。它是一个设计理念，一个开发方式，也是一个构建 CSS 的方法。

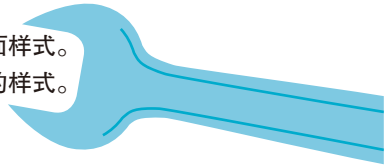
这种设计理念仅仅是一个观念，即首先针对移动设备设计，然后通过调整使它适应桌面设备。因为本书不讲设计，所以我将这部分留给其他更有资格的人来讲（Luke

Wroblewski 的书《Mobile First》是一部伟大的开山之作）。开发方式是一个类似的观点：先为移动设备编写代码，然后再适应桌面设备。在实践中，一个交互复杂的移动网站，肯定会与桌面网站的运行方式截然不同，因此会产生大量的不能共用的代码。这没有什么不好，只是我不认为它能节省时间。

真正的移动优先，对于大多数生产环境的网站而言还为时过早，因为 Internet Explorer 8 不支持媒体查询。（我们会在本章后面的部分详细地探讨媒体查询）。对于这个网站，我们会使用一些 HTML5 标签。这意味着，为了正确地支持 IE 浏览器，需要做一些额外的工作，或引入一个库，比如 Modernizr。本书不是关于如何支持 IE 浏览器的，所以我不打算涉及这一点，但在实际生产中，支持移动设备和桌面设备的网站，就必须支持 IE8。

2.2 创建标记[1]

对于这个网站，我们会用“移动置后”的方案来编写，但我们会在移动版和桌面版上使用相同的标记。我们将专注于语义标记，因为网站需要在移动设备上工作，所以我们还得考虑 DOM 和 CSS 的性能。



注：“移动置后”的缺点之一是移动设备需要下载和解析所有的桌面样式。如果采用“移动优先”，那么移动设备可以安全地忽略那些不需要的样式。另外，这是一个需要逐一分析的问题。

标记一个文档时，我把它分成区域，这样不仅有合理的语义，而且在设计和添加样式时也很方便。对于加州鸟类站点，我把导航链接放置于一个 <nav> 标签内，再插入一个 <aside> 标签，来表示桌面版的侧边栏或移动版的顶部导航栏。对于内容，我创建了一个 class 为 main 的 标签，其中包含了照片、标题和内容的拷贝。

标记文档如代码清单 2.1 所示。

代码清单 2.1　页面 body 部分的标记

```
<body class="bd">
  <header class="container header">
    <h1>Birds of California!</h1>
  </header>

 <aside class="sidebar">
   <nav class="bird-nav">
     <ul class="bird-list">
       <!-- Redundant Class for performance -->
       <li class="nav-li">
         <a class="nav-link"
```

译注①：　指 HTML 标签，后同

```
          → href="/california-condor">California Condor</a>
        </li>
        <li class="nav-li"><a class="nav-link"
         → href="/california-quail">California Quail</a></li>
         <li class="nav-li"><a class="nav-link"
         → href="/california-gull">California Gull</a></li>
      </ul>
  </nav>
</aside>

<div class="container main">
  <h2>California Gull</h2>
  <div class="hero-shot">
    <a href="http://www.flickr.com/photos/catlantis/5514922015/">
    <img class="hero-img" src="http://farm6.staticflickr.com/
    → 5171/5514922015_bfeab78ce0_z.jpg">
    </a>
    <p class="caption">Photo By <a href="http://www.flickr.com/photos/
    → catlantis/5514922015/">C.L.
    Maclay</a></p>
  </div>
  <section class="content">
            <p>Lorem ipsum dolor...</p>
          </section>

      </div>

     <footer class="container ft">
       <ul class="foot-links">
         <li><a href="/copyright">&copy; 2012 Awesome Bird Foundation,
         → All Rights Reserved</a></li>
         <li><a href="/tos">Terms of Service</a></li>
       </ul>
   </footer>
</body>
```

你会发现，有不少看似多余的 class。例如，导航链接 <li> 标签有 “nav–li” 的 class。这有两个原因：

1. 当代码量变大时，class 会更容易管理。使用 nav–li 比使用选择器 “nav ul li” 更简洁、更干脆、更容易进一步开发。

2. 由于网站要能在世界上最慢的免费的 Android 手机上运行，当涉及 CSS 选择器的性能时，我们需要非常挑剔，也就是说要避免使用后代选择器。

一个直接的反面例子，浏览器解析 CSS 选择器是从右往左的。这意味着，如果它看到一个如 “nav ul li a” 的规则时，它首先必须获得所有匹配 a 的元素列表，然后检查看它是不是 li 的后代，再按相同规则继续检查。后代选择器虽然看起来方便，但它的性能开销是最大的。

2.3 奠定基础的 <head> 标签

代码清单 2.2 展示了加州鸟类网站的 head 标签

HTTP-EQUIV 元信息

http-equiv 元信息告诉浏览器如何去做，等效于设置了 HTTP 报头。这让你可以向浏览器传达一些原本由服务器告诉它的内容。当你不能控制服务器时，它特别有用。例如，如果你想设置一个缓存头，但又不能控制服务器，你可以使用 http-equiv 属性：

<meta http-equiv="expires" content="Wed, 05 August 2020 00:00:00 GMT">.

请注意，服务器报头的优先级高，所以只能设置或取消某一报头项，不能覆盖服务器的报头项。

代码清单 2.2 <head> 标签

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
  <meta name="viewport" content="width=device-width">
  <title>Birds of California</title>
  <link rel="stylesheet" href="reset.css" type="text/css"
media="screen" charset="utf-8">
  <link rel="stylesheet" href="birds.css" type="text/css" media="screen"
   → charset="utf-8">
</head>
```

我从 HTML5 的 doctype 开始，然后指定字符集为 UTF–8。指定字符集很重要，这不仅能确保文档显示正确，也可以避免可能的 UTF–7 字符集的安全漏洞。通常情况下，服务器会发送一个 Content–Type 的报头，但为了以防万一，我也同时在 HTML 中指定它。

我用了一个 http–equiv 属性设置非标准的“X–UA–Compatible”报头。它的值是 chrome=1 和 IE= edge，如果 IE 用户安装了 Chrome Frame 插件，则可以使用这个插件；如果没有安装，则使用 IE 浏览器最新、最好的引擎呈现，而不是以传统模式呈现。

下一个 meta 属性称为 viewport。它目前仅用于触屏设备，将它设置正确是非常重要的。

2.4 理解 Viewport

在 iPhone 诞生之前，手机浏览器尝试通过调整内容来适应网页，取得了不同程度的

成功。 iPhone Safari 没有做丝毫的尝试，取而代之的是在各种各样的虚拟窗口上展现网页，这些虚拟窗口被称为“视图”。用户可以通过放大来查看网页的部分内容或通过缩小来查看网页的全部内容。

为了给开发者提供一定程度的展现页面的控制权，苹果公司提供了 viewport 的元信息（meta）元素，它可以指定虚拟窗口大小。它改变了网站建设的很多方面。理解 viewport 是成功建设移动网站的开始。

虚拟像素

作为网站开发者，我们喜欢像素。在网页上进行布局，最精确和最简单的方式就是使用像素。像素是屏幕上最小的单元，一旦指定了像素值就能确切地知道它的大小。如果你拿出显微镜，可以在屏幕上一个一个地数出这些像素点。

在移动设备的屏幕上看到的像素与桌面设备上不同。这意味着，在 iPhone 上，无法拿出显微镜来验证一个元素是否是 300px 宽。在一个没有 viewport 元信息的页面上，视图会默认设定宽度为 980px，则宽度为 300px 的元素表示的宽度为 300 虚拟像素宽（图 2.3）。

举个例子，如果声明

```
<meta name="viewport" content="width=600">
```

那么一个在 CSS 中被定义宽度为 600px 的元素在页面加载初始化时将会横向充满屏幕（图 2.4）直到用户通过双击放大时才变得更大。

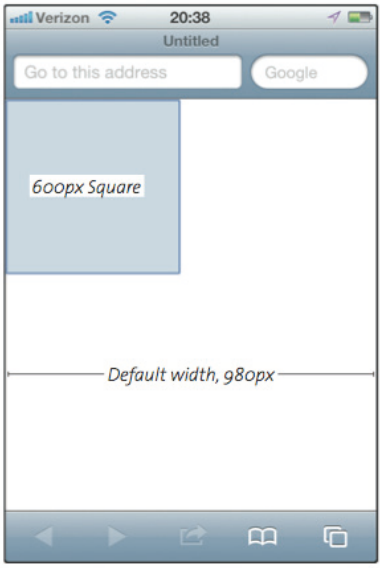


图 2.3 一个宽度 600px 的正方形在 viewport 的宽度为默认值的页面中的展现样式

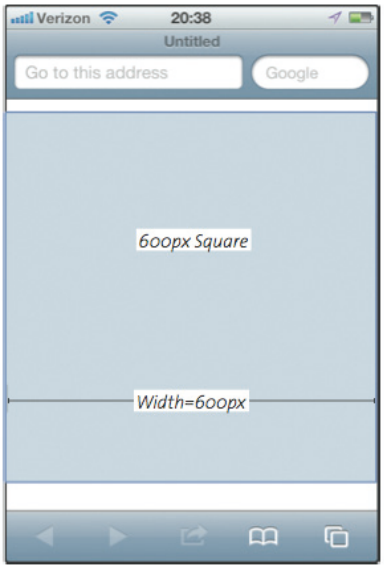


图 2.4 一个宽度 600px 的正方形在 viewport 的宽度为 600px 的页面中的展现样式

PX 与 EM

虽然在 CSS2 和 CSS3 的规范中定义了许多单位，但是大多数开发人员限制自己只使用两个：em 和 px。1 个单位的 em 总是代表当前的字体大小。如果字体大小是 12px，那么 1em 就等于 12px。1 个单位的 px（历史上）是屏幕上的一个像素。2005 年左右，使用 em 变得非常流行，因为当时流行的浏览器有了改变字体大小的能力。因为单位 em 的定义是基于字体大小的，所以可以很容易地进行布局，以适应用户选定的字体大小。

自 IE7 开始，浏览器的默认缩放为全页面缩放，而不是只改变字体大小。 由于简单，px 现在已是设计师们最流行的选择。px 更容易沟通，更容易理解。像 em 和其他未被充分利用的单位也有相应的用途，它们主要用于排版，而不是布局，px 才是用来对网页进行布局的最简单的单位。

viewport 就是一个虚拟的窗口，viewport 的边缘代替浏览器的边缘，成为了窗口的边缘。

viewport 的宽度和高度除了可以用像素值，还可以接受两个关键字：设备宽度和设备高度，这显然是根据设备屏幕的像素（图 2.5）返回实际的尺寸。

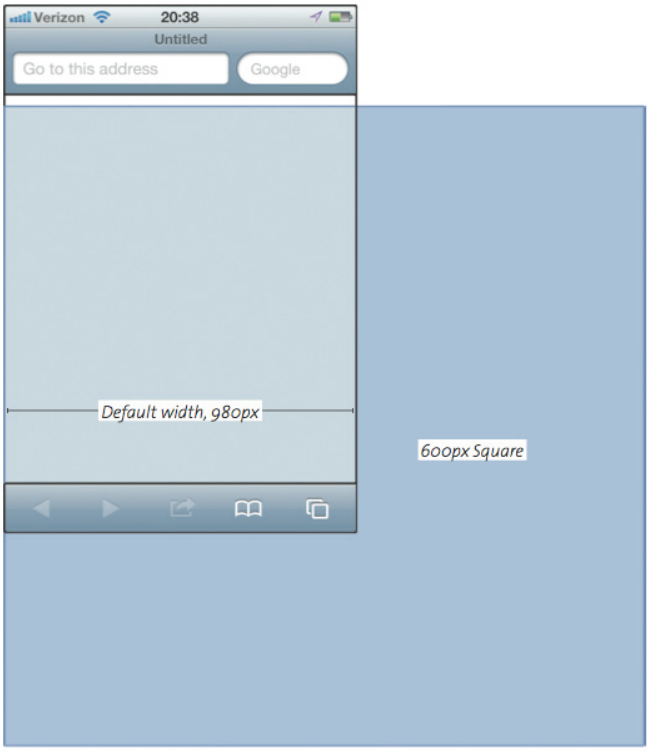


图 2.5 一个 600px 的正方形在 viewport 的宽度为设备宽度，即 320px 的页面中可能的展现样式[①]

译注①：图 2.5 中，屏幕上印的“Default width, 980px”是指未指定 viewport 时，页面横向将认为是 980px 宽。而此时设置了宽度为“device-width”后，页面横向将认为是 320px 宽，至于为什么 device-width 为 320，作者在后续“高密度显示屏”一节中有说明。

表 2.1 viewport 属性（在安卓 2.2+、iOS1.0+、Firefox 移动版 1.1+ 中被支持）

属性	描述
width	设置 viewport 的宽度。默认值为 980，允许的取值范围是 200~10000
height	设置 viewport 的高度。默认值是通过宽度和设备宽高比计算出来的，允许的取值范围是 223~10000
initial-scale	设置 viewport 的初始比例。默认值是通过计算使得整个页面在可见区域内。范围由 minimum-scale 和 maximum-scale 属性决定
maximum-scale	设置 viewport 的最大比例。默认值是 5.0。允许的取值范围为 0~10.0
user-scaleable	决定用户是否可以缩放视图。还可以防止文本开始输入时的滚动

加州鸟类网站的 viewport 宽度会被设置为与设备的宽度相同，从设计的角度，这样看起来方便。此外，当我们要确保 CSS 适应不同的设备时，也会有很大帮助。从 iPhone 1 到 4S（假设在垂直方向），device–width 的值都是 320px。

高密度显示屏

从 iPhone 和 viewport 标签出现之后的每一代移动设备的分辨率都在增加。现在像素太高，实际像素小到即使用显微镜都难以分辨。如果规范没有改变，当网页的 viewport 宽度设置为设备宽度时，出现非常微小的用户界面。在 iPhone 4 中，即第一个有着高密度显示屏的设备上，这些元素相对它们在老版本的 iPhone 上只有一半大小。

此外，苹果公司是第一个将高密度显示屏设备推向市场的制造商。为了让 Web 开发者的思维保持清晰，苹果公司决定继续在 iPhone 4 上返回 320 的设备宽度，尽管屏幕物理像素为 640。 安卓设备也如法炮制。这些设备更复杂，因为在如何显示上，它们给了用户更多的控制权。（安卓上的 Chrome 有一个不是很有用的 target–density dpi 的 viewport 属性来支持它，你可以查看安卓开发者文档来获取更多的相关信息）。所有的设备都将返回一个设备开发者认为是布局界面元素的理想尺寸的值作为 device–width。值有很大不同，所以当我们为一个 viewport 宽度为设备宽度的 Web 页面的布局时，我们需要确保布局方式可以处理一些伸缩问题，就像一个传统桌面站点的流式布局一样。

这对加州鸟类网站意味着什么呢？因为设计师为我们提供了优美且充满整个视窗的移动布局，我们可以这样设置 device–width：

```
<meta name="viewport" content="width=device-width">
```

在大多数情况下，这样设置是最好的，因为它允许界面完全适应用户的设备，我们就不必担心网页的宽度会超过设备的宽度。

2.5 响应式的 CSS

在本书的配套网站，touch–interfaces.com 上，你可以下载两个 CSS 文件：

- Eric Meyer 的 reset.css 文件。里面有几个用于“重置”的 CSS 样式。我强烈推荐使用它，因为从“重置”后完全清洁的环境开始，要优于不停地尝试修改兼容浏览器默认的样式。
- 样式表（birds.css）是加州鸟类网站专有的。birds.css 由桌面版样式开始。你可能会认识一个非常简单的两栏流动布局。桌面版的内容区域是流动的，而包含导航按钮的侧边栏是固定的。

代码清单 2.3 展示了这些基本的样式。

代码清单 2.3　基本样式

```
html {
   background: #fff;
   color: #000;
}

a {
  color: green;
  text-decoration: none;
}

p {
  margin-bottom: 10px;
}

h2 {
   font-size: 20px;
   margin: 4px;
}

i {
  font-style: italic;
}

.container {
  padding: 0 50px;
}

.bd {
  font-family: Helvetica, "Helvetica Neue", Arial, sans-serif ;
}

.hero-img {
```

```
  max-width: 100%;
}

.nav-li {
  display: inline-block;
  background: #5e49ff;
  border: 3px solid #8a7bfd;
  width: 120px;
  margin-bottom: 10px;
}

.nav-li .nav-link {
  color: #fff;
  padding: 4px;
}

.header {
  width: 100%;
  height: 60px;
  background: #000;
  padding: 0;
  font-size: 38px;
  font-weight: bold;

}

.header .title{
  color: #fff;
  padding: 10px;
  text-align: left;
}

/* Allow the image to grow proportionally inside its container */
.hero-shot {
  width: 50%;
  float: left;
  margin-right: 10px;
}

.sidebar {
  position: absolute;
  padding: 10px;
  top: 60px;
  width: 150px;
}
.main {

  margin: 10px 10px 10px 150px;

}
```

```
.footer {
  width: 100%;
}
```

你可能会注意到，海鸥图片设置的 max-width 为 100%，而封闭的容器（.hero-shot）具有相对宽度[①]。这是创建所谓响应式图片的最简单方式。也就是说，图片会自动按比例缩放到容器（即 .hero-shot）的宽度。在这种情况下图片的宽度总是文本块的宽度的一半。

这种响应式图像技术有一个重大的缺点：用户的设备肯定要下载多余的字节。在下一章中，我们将重新审视图像的问题。

2.5.1 创建分界点

正如前面提到的，我们将在加州鸟类网站的移动版和桌面版中使用相同的 HTML 标记。设计上是基于这一点的，布局和设计都会调整，以适应用户的设备的大小。为此，我们可以创建分界点：不同的像素值宽度会触发版式的更改，来适应不同的像素宽度的屏幕。对于加州鸟类网站，我们将创建两个分界点：平板电脑为 800px，手机为 480px。

也就是说，如果屏幕宽度是 801px 或以上，我们将应用样式表中的默认样式。我们将为屏幕宽度在 481px 到 800px 之间的平板电脑创建特殊的样式，为屏幕宽度为 480px 或更窄的手机创建额外的样式。

如果你以前从未这样做过，改变版式以适应屏幕宽度的想法听起来很可怕。值得庆幸的是，媒体查询（media queries）使人们可以创造出一个能自适应的版式，而不必使用 JavaScript。

媒体查询

由于 CSS2 已被广泛的支持，开发者已经能够通过在 link 标签上设置不同的媒体查询值，来为不同的多媒体设备提供不同的样式表。通常情况下这个值都被用于规定一个单独的“打印”样式表，例如：

```
<!-- this stylesheet is for the screen -->
<link rel="stylesheet" media="screen" href="styles.css">
<!-- This stylesheet is for printing -->
<link rel="stylesheet" media="print" href="print-styles.css">
```

CSS3 提供了更强大的语法，让你可以根据更多的条件过滤样式表，而不是只能根据媒体类型来过滤。媒体查询由一个媒介类型和其他表达式组成。解析后，媒体查询最终的结果是真或假。如果为真，则应用样式表。

媒体查询可以由一个布尔运算符开始。用“not”开始的话，会将字段最后解析出的结果取反，就像在 JavaScript 中使用“!”一样。最常见的，你可以用仅有的一个操作符来开始。由于仅有一个操作符时，旧的浏览器无法解析，就可以有效地在旧浏览器上

译注①：即 50%

屏蔽掉只应用于现代浏览器的样式表。

接下来就是媒体类型了。在 CSS 2.1 规范中，有十个媒体类型，但只有 print 和 screen 被各浏览器广泛支持。在媒体类型后面，你可以指定一个表达式。对于加州鸟类网站，我们将根据屏幕宽度来改变样式。宽度特征可以采用任何有效的 CSS 单位（用任何 CSS 单位来指定）来达到过滤的目的。

例如，下面这条规则限制只有比较窄的屏幕才可应用该样式表：

```
<link rel="stylesheet" media="only screen and (max-width: 480px)"
→ href="phone-styles.css">
```

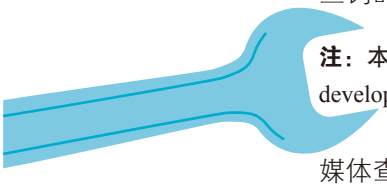
下面这条规则限制只有比较宽的屏幕才可应用该样式表：

```
<link rel="stylesheet" media="only screen and (min-width: 2000px)"
→ href="phone-styles.css">
```

对于加州鸟类网站，我们用媒体查询来定义不同分支。因此，有三个样式表：

```
<link rel="stylesheet" media="screen" href="birds.css">
<link rel="stylesheet" media="only screen and (max-width: 800px)"
→ href="tablet.css">
<link rel="stylesheet" media="only screen and (max-width: 480px)"
→ href="phone.css">
```

这里的像素值基于浏览器的报告。例如，在高清屏 iPhone 的垂直方向上，因为媒体查询的原因，宽度返回的是 320，而不考虑屏幕上的实际物理像素。



注：本章仅讲述了媒体查询的一些皮毛。可查阅 Mozilla 开发网络了解详细的语法解释（https://developer.mozilla.org/en-us/docs/Css/Media_queries**）。另外，还可以在** mediaqueri.es **找到一些例子。**

媒体查询真正伟大之处在于，我们不仅能通过外部的媒体属性使用它，还可以在样式表内部中通过 @media 指令使用它。

```
@media only screen and (max-width: 800px) {
  /* css that applies only in this case */
}
```

这个语法与媒体查询参数的语法是相同的。@media 指令在大括号之间创建了一块区域。括号内的 CSS 只在规则返回 true 时被应用，否则浏览器将不理会里面的 CSS。

2.5.2 创建额外样式

对于加州鸟类网站，我们会为宽度 800px 的浏览器和宽度 480px 的浏览器制作额外的样式。设计师又为平板电脑创造了另一个样式图，将导航移动到了顶部中间，以留出更多空间给正文。否则，样式应该是一样的（图 2.6）。

因为侧边栏最先出现在 DOM 树中，我们仅仅通过给它设置相对位置（postion:relative）

和自动宽度（width:auto）就可以将它放回文件流。然后就可以给 container 少许内边距（padding），如代码清单 2.4 所示。

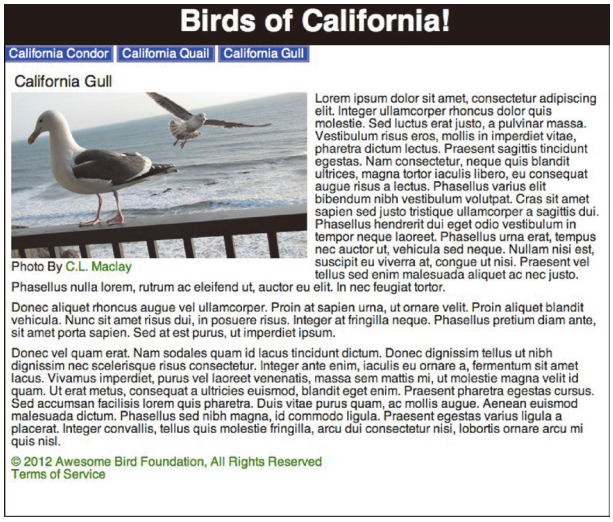


图 2.6 加州鸟类网站的平板电脑版样式图

代码清单 2.4　平板电脑的样式

```
.container {
  padding: 0 10px;
}

.nav-li {
  width: auto; /* so all
  the text fits */
}
.sidebar {
  position:relative; /* back in the flow */
  top: 0;
  width: auto;
  padding: 0;
  text-align: center;
}

.header .title {
  text-align: center;
}

.main {
  margin: 0;
}
```

桌面用户可以改变他的浏览器的大小，当桌面用户调整浏览器大小时，会发现随着调整到不同的宽度，可能会看到应用于手机和平板电脑的样式。如果你想让这些样式只适用于设备的物理宽度，可以使用 max–device–width 或 min–device–width 属性来设置。

正如你所看到的，样式通常伴随着媒体查询。手机版样式图如 2.2 所示，它需要一个稍微不同的布局。代码清单 2.5 列出了手机的样式。

代码清单 2.5　手机版的样式

```
.container {
  padding: 0;
}

.main {
  margin: 0;
```

```
}

.content {
  margin: 10px 10px;
}

.nav-li {
  font-size: 12px;
}

.hero-shot {
  float: none;
  width: 100%;
  height: 100px;
  overflow: hidden;
  position: relative;
}

.header .title {
  font-size: 24px;
  text-align: left;
}

.hero-shot .caption {
  position: absolute;
  bottom: 5px;
  margin: 0;

}

.hero-shot .caption, .hero-shot .caption a {
  color: #000;
  color: rgba(255,255,255,0.5);
}

}
```

完整的代码和代码清单 2.5 都可以在网站上找到。如果你在桌面设备上查看这个页面，你会发现当你调整浏览器窗口大小时，样式会发生变化。

2.6 总结

在本章中，你学会了如何使用媒体查询来创建版式的分界点。你还学习了 viewport 元信息和虚拟像素。请记住，移动浏览器不可以调整大小，并且没有标准的宽度。任何在移动设备上的运行的网页的布局必须是灵活的。

第 3 章

提高第一次加载的速度

用户首次访问网站，就开始对网站是否“慢”有了自己的认识。如果第一次加载让用户感觉迟钝，特别是在移动网络连接环境下，用户可能会在页面完成加载之前就离开，不会再次访问。想要网站迅速响应，开发人员要做的最基本的事情就是提高第一次加载的速度。

虽然第一次加载速度是用“首字节的时间”测量的，即从用户请求该页面到第一个字节从服务器下载下来的这段时间。尽管了解、测量和优化第一次加载速度很重要，但在绝大多数情况下，这不是第一次加载缓慢的真正原因。真正的原因通常在于前端。PageSpeed、YSlow 和其他无数的工具和服务可以用来解决这些问题。

3.1 浏览器是如何加载页面的

讲述如何优化的细节之前，让我们回顾一下浏览器是如何加载页面的。

3.1.1 解析域名

首先，浏览器需要知道该网站的 IP 地址。它向 DNS 服务器发送一个包含域名的请求，然后 DNS 服务器返回对应的 IP 地址。为了减少 DNS 服务器上的负载，并提高性能，DNS 查找机制会被浏览器、设备或设备和服务器之间的路由器和代理服务器缓存。这就是为什么更改 DNS 记录可能需要几天才能生效的原因。

3.1.2 发起请求

然后，浏览器会向由 DNS 查找得到的 IP 地址的主机发起 TCP 连接。然后发送请求。请求中包含网址、浏览器信息、浏览器能接受的数据类型（编码和语言），以及所有相关的 cookie，包括域和路径的 cookie。

3.1.3 下载响应

浏览器开始下载响应。随着响应流到达，浏览器解析 HTML 并识别出更多的资源。然后浏览器开始获取这些资源。

3.1.4 渲染页面

最后浏览器会尽快开始渲染页面。如果页面中外链了 CSS 或脚本文件时，浏览器会等到这些文件加载和解析（如果是 JavaScript 代码，则还需要执行）完再渲染页面。

3.2 为什么页面加载缓慢?

为什么页面加载缓慢？以下是可能的原因：

- HTTP 连接数
- 总的字节数
- 等待时的渲染阻塞
- 延迟
- 缓存能力差

WEBKIT 时间线

除了良好的文本编辑器，移动 Web 开发中最重要的工具是 WebKit 开发人员工具。我们在真实设备上认真地测试和优化之前，仅用桌面上的 Chrome 或 Safari 浏览器就能完成很多工作。使用实际设备不太方便，特别是在开发阶段，WebKit 开发人员工具是一个很好的工具。如果你有 Mac 和搭载 iOS6 的设备，Safari Web 检查器会变得很强大（稍后将详细描述）。当诊断性能问题时首先应该从网络面版开始，在 Safari 浏览器中，它隐藏在工具栏中，被称为网络请求[①]。（图 3.1）

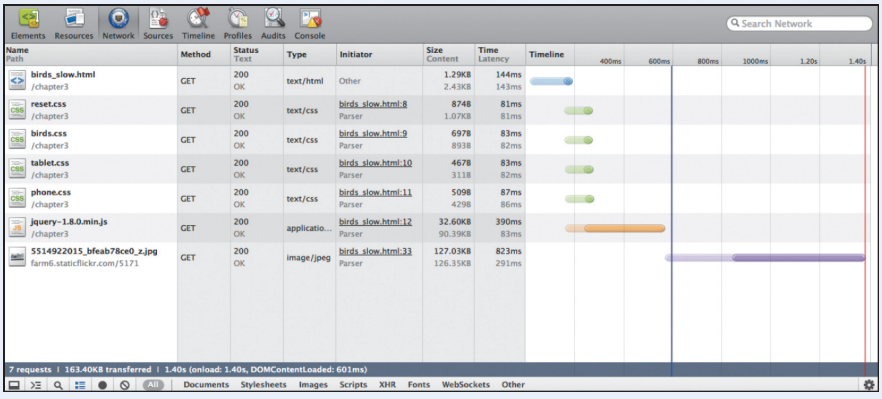


图 3.1 WebKit 开发人员工具

这里有很多非常有用的信息，现在我们先把重点放在“网络”选项卡上，它有一个美丽的瀑布图，向我们展示了页面加载时的各种信息。图 3.2 展示了加州鸟类网站是如何加载的。

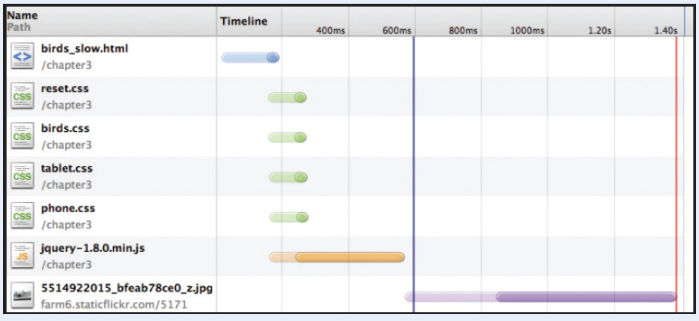


图 3.2 加州鸟类网站的瀑布图

每一栏上的浅色代表的延迟，深色代表下载。你可以看到，在浏览器解析部分页面之前，没有任何外链的资源开始加载。你还可以看到，在下载 jQuery 之前没有开始获取图像。即使在我家快速的网络环境中，加载完页面花了 1.4 秒。从顶部第一栏可以看出，延迟（服务器端性能）并不差。对于一个如此简单的页面，加载速度真是太慢了。

译注①：在译者翻译时，当前 Windows 上的 Safari 浏览器版本为 5.1.7，读者可以先进入“偏好设置”，选择“高级”，勾选“在菜单栏显示开发菜单”，然后才能在菜单栏找到“开发”菜单，并从中或通过按“Alt+Ctrl+I”打开 Webkit 开发人员工具。

3.2.1 HTTP 请求数

在网页上的每一个外链的资源需要一次单独的 HTTP 请求。一次 HTTP 请求不是只下载数据那么简单；每次请求都包含一定量的额外开销。所以，连续不断的请求许多小文件会比一次请求一个大文件慢很多。

当然，浏览器可以同时下载多个文件。如果回头来看图 3.2，你会发现大部分资源是并行下载的。HTTP/1.1 规范建议两个资源并行下载，现代浏览器可以同时下载的资源数更多。iOS 上的 Safari 浏览器支持同一域下的最多 6 个资源并行下载。虽然可以通过添加额外的域名（也许是通过设置别名或子域名）的方式来并行下载更多的文件，但每次请求还是需要承担 HTTP 层面的开销。

这可能显得有些奇怪，并行下载并没有起到太多作用。它终究没有克服成本的开销，因为两个文件并行下载的速度并不等于串行下载速度的两倍。创建新请求不仅产生新的开销，而且还需要耗费 CPU 和内存。

对于较大的文件，比如比较大的图像，说法有所改变。因为下载占到了请求时间的绝大部分，所以并行下载更好。出于这个原因（以及一些其他的原因），在网站上将图像与其他资源置于不同的域下是合理的。雅虎的 Steve Souders 和 YSlow 团队发现，为当前域创建两个别名，能允许更多的下载并行，会使大文件下载的性能明显改善。

你可以试想，因为有并行下载的数量限制，在某些时候浏览器必须等待请求完成后，再开始下一个下载。也就是说，如果你的网站的所有服务和资源都部署在同一个域下，它的第一次加载必然比部署在多个域下的慢。不过由于每个域都需要进行 DNS 查找，添加太多的域反而会更慢。使用至少两个，至多不超过五个域是 YSlow 的经验法则。

SPDY 和 HTTP 管线化

你可能问过自己，为什么每个请求都必须耗费 HTTP 开销？如果所有请求都来自同一个域，为什么不干脆让链接一直打开，然后把更多的数据依次下载下来？

这个问题不只有你想到过。两个相竞争的解决方案出现了。SPDY（发音为“speedy”）是由 Google 开发的一个新的协议，旨在取代 HTTP。另一个方案是 HTTP/1.1 中规定的“管线化”，但不是所有的浏览器中都实现了这一方案。这两个方案将允许浏览器使用同一链接来下载多个资源，克服并行链接的局限性，使得使用多个较小的文件也会优于目前的最佳实践方案——使用少数较大的文件。

另一个要考虑的是浏览器的 cookie。如果一个 cookie 与请求的域名或路径相匹配，它会伴随着每次请求发送（即上传）。所以，如果你在你的域名下的第一个请求中设置了几千字节的 cookie，那么此后发送到这个域的每次请求都将把这些字节包含在请求头里，然后发送、解压缩。服务器还必须在读取请求体之前读取这些 cookie。cookie 可以把一个很小的请求变得很大。

3.2.2 总字节数

接下来要讨论的导致页面加载慢的因素大概是最显而易见的：被下载资源的大小。网页起初并不大，但是当你添加 JavaScript 库、样式和大部分的图像后，可能会增大几个数量级。任何能降低下载资源大小的行为都是值得做的。

3.2.3 等待时的渲染阻塞

在本书中，我们讨论了很多关于用户感知到的性能。有些事情不会真的提升速度，但可以让用户感觉起来更快。用户反馈对于响应性能是至关重要的。当用户盯着一个正在加载的空白页时，他不知道是连接断了还是页面加载慢。如果用户可以看到你的网页，即使资源未全部加载，用户感知上的性能会更好。例如，script 标签将会阻塞后续 HTML 的渲染，直到它所包含的脚本已下载到本地，并解析执行完毕。当你把四五个外部 JavaScript 放在页面头部时，只有在所有脚本加载完后，用户才能看到网页。

3.2.4 网络延迟

网络连接可以通过带宽（比特）和延迟（毫秒）来测量。网络延迟耽误的时间会被加到链接的请求时间内。一个典型的家庭网络连接可能有 8mbit/s 的下载速度和 15ms 的延迟。一个典型的 3G 连接可能有 500 kbit/s 的下载速度和 100ms 的延迟。因此，3G 连接与家庭网络连接相比，不仅下载的速度慢得多，而且延时也高得多。

延迟令用户烦恼，即使一旦开始下载速度就非常快，但在开始下载之前的等待相当痛苦。高延迟使因请求量大而导致的问题激增，因为每次请求都因延迟增加了往返的时间。例如，在好的宽带链接环境下，重定向花费的时间可能不易察觉。但在 3G 网络环境的移动设备上，它可能会使页面加载的总时间增加 200ms。而 200ms 是非常明显的延迟。

3.2.5 缓存能力差

我们将在下一章讨论更多关于缓存优化的知识。现在，只需要记得一些 PageSpeed 和 YSlow 的规则来确保缓存设置正确，确保浏览器最终不会重复获取已有的资源。

3.3 用 YSlow 和 PageSpeed 提升速度

前面我们对页面加载缓慢的原因做了简单的研究。现在，让我们看看如何让页面加载速度加快。幸运的是有两个优秀的工具（它们都与网络性能专家 Steve Souders 有关）可以帮助你分析网站的性能问题。

3.3.1 YSLOW

YSlow 是雅虎卓越性能团队在 2007 年开发的一个工具。它将你的网站与一组最佳的实践方案进行比较。YSlow 的网站列出了 34 条规则，这个工具能对其中的 23 条进行测试。完整的列表和详尽的文档可在 developer.yahoo.com/yslow 上找到。

图 3.3 显示了我们用 YSlow 对加州鸟类网站进行测试的结果。我在 head 内添加了一个 jQuery 的外链，这是为了更接近实际情况。

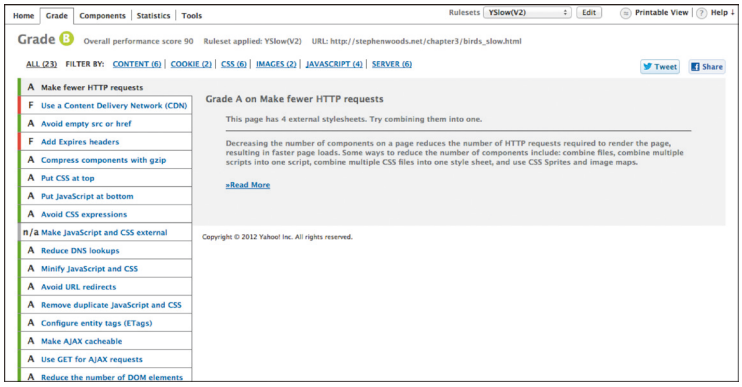


图 3.3 Yslow 给出的加州鸟类网站的报告

这个网站的整体评分为“B”，主要是因为它非常简单。它只有四个外部 CSS 文件和一个 JavaScript 文件，所以 Yslow 对图中所示的这项评分为“A”，同时提出了一些建议。评分为“F”的有两类：“添加报头（header）的有效期”和“使用内容分发网络（CDN）”。我们将在下一章讨论关于“添加报头的有效期”的更多相关内容。YSlow 自推出以来的闪光点之一是要求使用 CDN 的建议。这也是 YSlow 又提供了适用于“小型网站或博客”的特殊规则集的主要原因之一。

内容分发网络（CDN）是一种服务，让你能够从“周边的服务器”上获取某些（通常静态的）内容：服务器分布在很多地方，最好的情况是可以就近地服务最终用户。这种服务通常用于图像、JavaScript 和 CSS 文件，因为它们不太可能动态变化。

通常这些服务器能够缓存资源。如果你的 JavaScript、CSS 和图像文件放在了 CDN 上，如 Akamai 或亚马逊的 CloudFront，这些文件可能被缓存在了距离最终用户只有几英里的数据中心。距离短是至关重要的，因为有限的传播速度是延迟的主要原因之一。

在玻璃（如光纤）中光的速度比在真空中稍微慢一些，所以在 1000 公里的光纤中往返一次需要约 11ms。我的网站（我用来测试的加州鸟类网站）的服务器托管在达拉斯。假设我是通过光缆直接连接到该服务器，预计因为距离就会有 22ms 的额外延迟。实际上，我和服务器之间的网络会更慢一些，通过 ping 测试表明，到服务器大约有 50ms 的网络延迟。CDN 网络使服务器离用户更近。我尝试 ping Akamai 或 CloudFront 的主机，延迟只有大约 17ms。显然 CDN 有巨大优势。

你并不需要上传文件到 CDN，CDN 的工作原理是从服务器获取内容，然后缓存，再让附近的用户获取。所以第一个请求这个文件的用户会得到一些好处，因为这个文件是通过专线传输的，但距离造成延迟仍会对他产生影响。此后，与第一位用户位置相近的其他用户，将受益于缓存在 CDN 的版本。如果你的网站有良好的传输保障，你会从中受益。

周边缓存（以及其他缓存系统）的整体收益是以缓存的“命中率”为基础的。如果大多数 CDN 缓存的请求都指向空缓存（即不存在的文件，此时 CDN 会去服务器上获取这个文件）的话，获得的收益是有限的。这种情况是有可能的，因为存在很多 URL，用户不需要请求所有的 URL，或是因为流量太低、用户的地理分布太分散导致缓存很少被使用。对于低流量的网站 CDN 的帮助不大。不过，把你的静态文件放在一个单独的没有 cookie 的域下总是有帮助的。所以，如果有条件，务必做到这一点。

3.3.2 PageSpeed

Steve Souders 离开雅虎后加盟了谷歌，成为 PageSpeed 工作团队的一员。PageSpeed 在 YSlow 的基础上，增加了更多的规则和功能。最重要的是，PageSpeed 有一套专为移动网站设计的规则。

我使用该在线工具的移动模式，对加州鸟类网站进行了测试（图 3.4）。

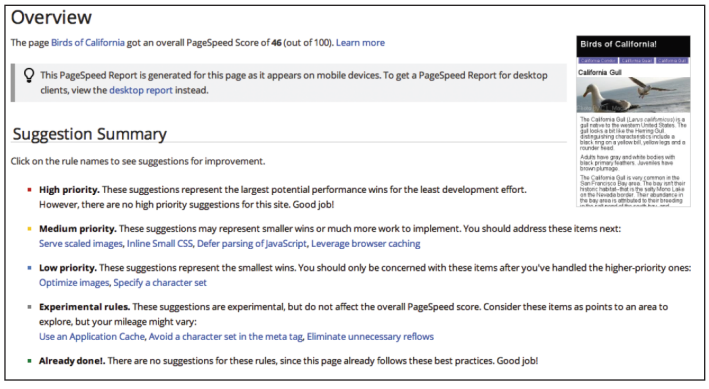


图 3.4 PageSpeed 移动模式下的加州鸟类网站解析报告。

PageSpeed 把规则分为六个基本类别。全部的详细文档可以在 PageSpeed 的网站（https://developers.google.com/speed/docs/best–practices/ rules_intro）上找到：

- 缓存优化：使页面应用的逻辑实现和所需数据一并离线保存
- 往返时间最小化：减少请求响应的周期次数
- 请求开销最小化：减少上传数据包大小
- 有效负载最小化：减少响应包、下载包和缓存页面的大小
- 浏览器渲染优化：改进浏览器的页面布局

- 移动设备优化：为移动网络和移动设备调整网站

目前移动类别包含两个建议：“延迟 JavaScript 解析”和“使首页的重定向可缓存”。对于第一项建议我们简单带过，但第二项建议，特别值得一提，因为这是一个非常普遍的问题。通常的做法是，将移动用户重定向到一个单独的移动版的网站（如 Flickr 就是将移动用户从 www.flickr.com 重定向到 m.flickr.com）。这是个高成本的做法，因为重定向会增加额外的（浏览器到服务器之间的）往返次数（重定向通过 HTTP 报头传达消息，通知给浏览器在另一个位置获取另一个资源）。

这种问题可以通过允许客户端缓存重定向得到一定程度地缓解。你可以通过包含过期时间的报头来实现（这个报头是在 2012 年 8 月 26 日设置的）：

```
HTTP/1.1 302 Moved Temporarily
Date: Mon, 27 Aug 2012 22:34:05 GMT
Expires: Sat, 25 Aug 2012 22:34:05 GMT
Cache-Control: private, max-age=86400
```

我已经加入了 Cache-Control: private，并设置 Expires 为过去的一个时间，设置 max-age 为未来的某一天。为什么不干脆将它设置地更久一些呢？为什么要设置 Cache-Control: private 呢？是因为代理的原因。许多用户通过代理访问互联网，因为它通过把资源缓存在离用户更近的地方，来提高性能。当进行重定向缓存时，我们要确保与某个移动用户使用同一个代理上网的桌面用户不被重定向到移动网站。

PageSpeed 还有更有趣的工具和重写页面的服务。请仔细查看 PageSpeed 的网站（developers.google.com/speed/pagespeed/），这是一个惊人的资源库。

3.4 解决常见的问题

加州鸟类网站在前端性能并不差，但还有很大提升空间。第一步是测量。我们已经讨论了 WebKit 开发人员工具。另一个有价值的工具是 Charles proxy（Charlesproxy.com）。Charles 是一个本地工具，能让你检查每一个请求，给请求添加断点，还可以模拟低带宽环境。当开发工具没有给出关于 HTTP 的更多信息时，Charles 可以帮助你。

为了真实体验连接的效果，我们用 Charles 模拟一个典型的 3G 网络连接。

单击 Proxy 菜单，然后选择 Throttle Settings 菜单项（图 3.5）。

预设的 3G 符合我们的目标（图 3.6）。

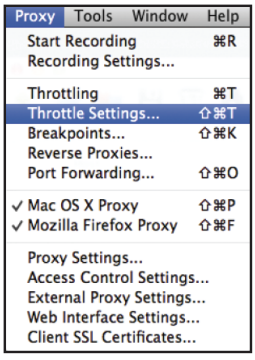
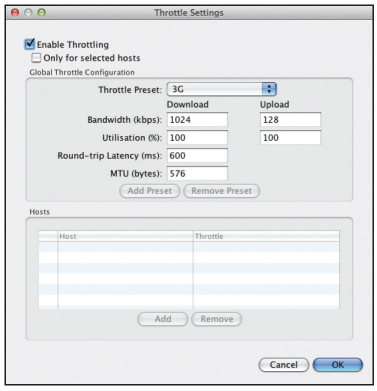


图 3.5 配置 Throttle Settings

现在让我们刷新页面，看看问题出在哪（图 3.7）。



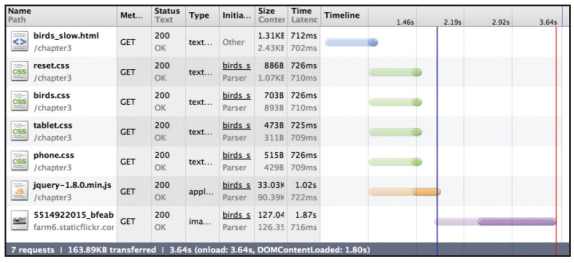


图 3.6 Throttle Settings 对话框，已经选中了预设的 3G

图 3.7 网速受限时的加州鸟类网站的瀑布图

该页面加载需要 3.5 秒。显然，高延迟是一个问题，有很多改进的余地。你可以在时间线里清楚地看出问题所在：

- 请求数过多
- 在 jQuery 加载完毕之前，图像几乎没有开始加载
- 图像太大了！

前面的章节所述的灵活技术，给了我们一个自适应图像，但我们下载了本不需要的更多字节。

现在让我们逐个解决网站上的这些问题。

请求数太多

在使用了媒体查询的移动设备上，要把所有的 CSS 文件合并成一个文件（一次请求）是相当繁琐的。我们可以这样使用 CSS 媒体查询的语法来代替在 link 元素上使用“media”属性。

```
<link rel="stylesheet" media="only screen and (max-width: 800px)"
→ href="tablet.css">
```

转换成：

```
@media only screen and (max-width: 800px) {
}
```

置于 CSS 文件内部。通过合并 CSS 文件，我们最终得到了包括所需要的所有 CSS 文件的 combo.css：

```
[Reset.css content]
```

```
[Birds.css]
@media only screen and (max-width: 800px) {
[Tablet.css content]
}
@media only screen and (max-width: 480px) {
[Phone.css content]
}
```

让我们看看这样能让状况改善多少（图 3.8）。

好吧，真见鬼，并没有起到太多作用。即使有一些差别，也相当小，肯定不到 1 秒，所以我们想减小负载。现在要考虑的事情是，我们正在关注没有缓存时的体验——并且有一个好的理由。手机浏览器缓存是非常有限的（会在下一章详细讲述），你不用指望它们。

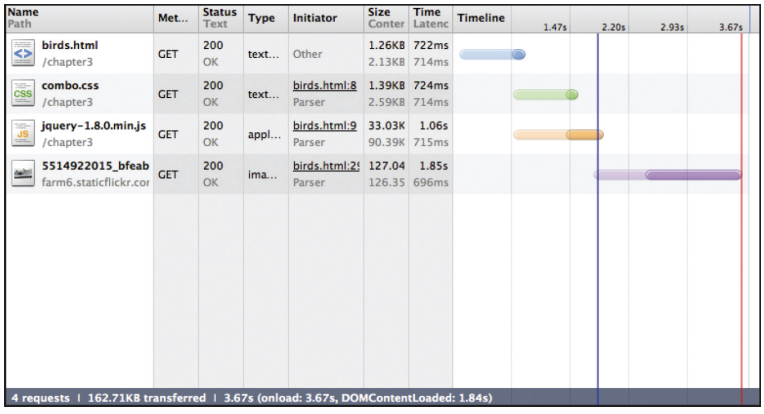


图 3.8 合并全部 CSS 文件后的效果

只有收集真实的数据，你才能知道没有缓存的用户对网站的访问是多么频繁。YSlow 团队在 2007 年的收集到的数据表明，在所有用户中，约 20% 的用户的缓存是空的，在手机上这个数值肯定会更高。缓存策略我们将在下一章讨论，现在让我们继续前进，使样式内嵌。我们还会删除任何代码中的空白，只保留真正有用的字节。

关键点

不幸的是，内联样式也不会改善多少状况。在查看瀑布图的时候，我们试图弄清楚什么是“关键点”。对于这个网站，除了图像，很明显 jQuery 是一个很大的问题。事实上，在 jQuery 加载完毕之前，图像几乎没有开始加载。如果你还记得，就知道是因为渲染会被阻塞，直到脚本被加载和解析完毕。由于在 document 中，图像标签没有位于脚本标签之前，所以浏览器在 jQuery 完全加载完毕之前无法开始获取图像。

解决这个问题的最快捷的方法是把脚本标签置于页面的底部，而不是顶部。这样一来，当脚本还在加载时，用户不会看到一个空白页，而且其他内联的任何资源会和脚本一起并行下载。这样做之后加州鸟类网站的加载速度有了显着的提高（图 3.9）。

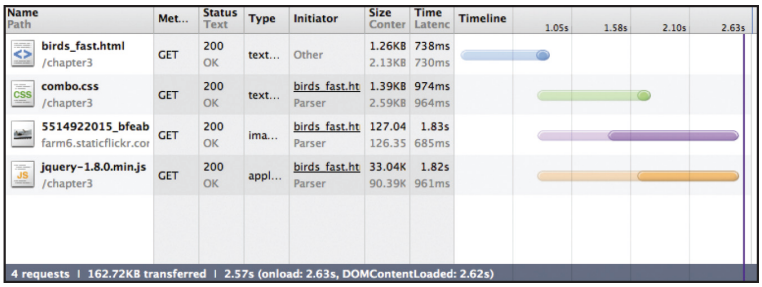


图 3.9 将 JavaScript 移置到底部后的效果

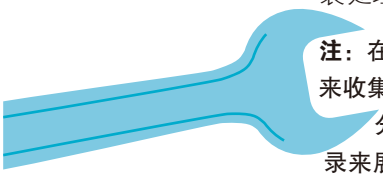
正如你所看到的，现在图像的加载比脚本的加载先开始，并且它们是并行加载的。根据对这个版本（内联 CSS 和把 JavaScript 放置在底部）的页面进一步测试的结果，加载速度大约快了一秒。更重要的是，用户在两秒钟内就看到了内容。用户不会根据所有资源需要加载多久来衡量网站的性能。他们认为的性能是基于看到想要看到的内容需要花费的时间的。

图像太大了！

再看一下，能发现状况还是不乐观。图像的问题太明显了。本书未压缩的文本大小约 250 千字节。然而我们一直在使用的海鸥图像，大小差不多是它的一半。当试图降低移动网站的字节数时，图像通常是最大的问题。更糟的是，近几代的手机，配有高密度的屏幕。iPhone 4 的物理像素尺寸为 960×640，而三星 Galaxy S III 则高达 1280×720。移动用户的网络连接通常较慢，但其高密度的屏幕要求分辨率与桌面版相等或更高的图像。

幸运的是，加州鸟类网站在设计上没有为用户交互界面使用任何图像。如果有，我们就需要更换每一个图像，我们可以用 CSS 渐变、盒阴影，并在可能的情况下使用圆角。尽管如此，我们还是设计了一张巨大的图像。从用户的角度来看，他只想下载他所需大小的图像。Flickr 上，用户上传大尺寸的图像后，其他尺寸的图像会自动创建。正因为这个实际的问题，Flickr 为每个图像生成 11 种不同的尺寸，包括常见的尺寸的高密度显示的版本。

然而大多数网站都不是 Flickr。对于这个网站，我们没有数百万美元的系统来快速生成不同尺寸的图像。相反，我们只有 Photoshop，如果这个网站要做得比较全面，它会有了不少鸟儿的照片。在（所有尺寸的）浏览器中使用同一个高分辨率的图像可以大大减少工作量，但用户却为此埋单。如果不能合理地产生每种尺寸，可以做一个折衷处理。对于这个网站，我会按照我认为最常见的情况进行优化。



注：在做决策时，没有比真实数据更好的了。你可以制作简单的报告工具来收集不同用户的数量，如浏览时设备的横纵向、屏幕分辨率等。大多数分析工具能让你得到这些数据，通过一般的报告或自定义数据的记录来展现。请记住，大多数移动用户使用按流量计费的网络连接，这意

味着他们必须按照所使用的字节数支付费用。为视网膜屏幕提供大图像看上去可能很好，但我们应该更明智一些。你应该只在真正必要的情况下才让你的用户为大图的额外字节支付费用。除了其他原因，这就是为什么加州鸟类网站的移动版设计只有一种尺寸的裁剪图像。

大多数手机用户可能会在纵向模式下浏览加州鸟类网站。在设计上，横向模式没有任何额外的功能，给这些横向模式下浏览网站的用户不是最理想，但还可以让用户接受的设计，我觉得也还行。

对于平板电脑，图像已经处理好了。对于手机，我们会对图像生成两个裁剪版：正常版的和视网膜版。以前我定义裁剪的图像是 100 像素，并填充屏幕的宽度。一个纵向的 Galaxy S III 的宽度是 720 像素，是所有常见设备中最大的。这意味着裁剪的海鸥图像要被横向拉伸到 720 像素。它也需要足够高，以适应各种尺寸的屏幕。我把它定义为 100 像素高，这意味着至少要 200 的物理像素宽来支持视网膜屏幕。在 iPhone 上，图像被缩小到只有 640 像素宽，而三星（Galaxy S III）上的图像比这大 12.5%。为了防止图像的高度太短，我们把图像扩大 2.5% 到 225 像素。因此最终我们将有两幅图像，一幅为 720×225，另一幅为 360×113，后者用于低像素屏幕。在横向模式下，这并不是视网膜屏幕下的完美图像大小，但还是比 480 像素的图像要好，我们的目的也差不多达到了。我们可以创建更多的图像来适用于横向模式，但在这种情况下，我认为这是不必要的，我愿意接受权衡。见图 3.10。

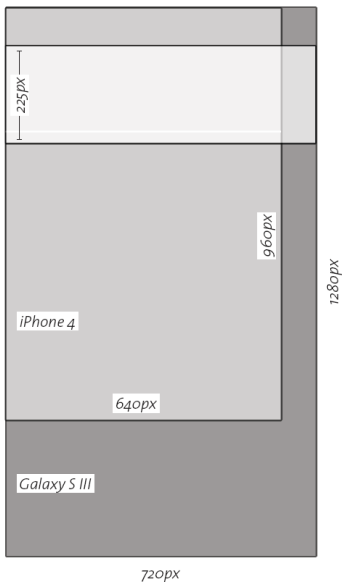


图 3.10 图像必须支持的极端尺寸

下一个诀窍是确保大的图像没有被移动端下载，只有该设备对应的图像才应该下载。有几个技巧，但我强烈推荐只使用 CSS 来实现它。同样，媒体查询能给我们提供帮助，它提供了一种在显示属性的基础上覆盖掉原有样式的方法。为了实现这一方案，我们将先前使用的 <img> 标签的方式替换成使用 加背景图像的方式。如果想让代码更好地语义化，我们可以在最开始将 src 属性指向一个微小透明的 GIF 图像。当然，如果 src 没有描述实际显示的图像，那我们在语义上就适得其反了。

新的“主角图片”的标记看起来像这样：

```
<div class="hero-shot">
   <a href="http://www.flickr.com/photos/catlantis/5514922015/">
      <div role="img"
         aria-label="California gull flying"
         class="hero-img">
      </div>
```

```
      <p class="caption">Photo By
   <a href="http://www.flickr.com/photos/catlantis/5514922015/">
      C.L. Maclay
   </a>
      </p>
</div>
```

这里使用了一个 ，它破坏了语义。不过我们已经降级使用了一项名为网页易读性倡议——无障碍富互联网应用（WAI–ARIA）的技术。ARIA 用附加的属性增强 HTML，来明确指定语义和其他信息元素。

将图像的“alt”属性转换为 ARIA 的标记可以恢复语义，至少对屏幕阅读器和其他辅助技术有意义。在此之前，我们设置了一个 100% 的最大宽度，强制图像在封闭元素内按比例放大：

```
.hero-shot {
  width: 50%;
  float: left;
  margin-right: 10px;
}
.hero-img {
  max-width: 100%;
}
```

不幸的是，这一方案对 div 标签不起作用。它能对图像起作用是因为图像在 DOM 中有原来的大小。如果你不指定图像的宽度和高度，它会自然地使用图像原本的尺寸规格。如果我们改变的标记而不改变样式，.hero–shot 的元素的高度会变为零。我们要做的第一件事是设置背景图像。但我们不希望图像放大超过原本的尺寸，所以我们将指定一个最大宽度（max–width）。我们还将设置宽度为 100%，因此图像的放大会和以前一样：

```
.hero-img {
  max-width: 640px;
  width: 100%;
  background: url(gull-640x360.jpg);
}
```

然而此时仍然没有高度。总之我们还需要设置的高度是宽度的百分比。在这种情况下的照片的像素是 640×360，所以我们需要的高度是宽度的 56%。通常你不能在 CSS 中做数学计算，但在这次我们很幸运。元素的 CSS 内边距（padding），可以以百分数来指定，这个百分数实际上是指该元素的宽度的百分比，即使在垂直方向，也同样如此（垂直方向的内边距也是宽度的百分比而不是高度的）！

所以我们这样写：

```
.hero-img {
  max-width: 640px;
```

```
  width: 100%;
  height: 0;
  padding-bottom: 56%;
  background-size: 100%;
  background: url(gull-640x360.jpg);
}
```

现在这个 会像以前一样扩展，这个时候使用 CSS 背景图像，而不是使用 img 标签和 src。这是一个很不错的技巧，但实际上并没有解决手头的问题，也就是会加载太多的字节数（因为加载了不必要的更大图像）。那么，我们如何解决这个问题呢?

首先，让我们解决不同的布局需要对应不同的图像大小的问题。像以前一样，可以使用媒体查询，但是这一次，我们是通过媒体查询来区分不同的图像。对于桌面浏览器，使用大尺寸很好，因为带宽和内存都没有太大的问题，并且桌面上的用户还能调整他们的浏览器窗口的大小。

```
@media only screen and (min-width: 800px) {
  .hero-img {
    background: url(gull-640x360.jpg);
  }
}
```

在平板电脑上时，图像的最大尺寸为 390×219，比我们之前创建的图像小了很多。和桌面不同，用户使用浏览器时只有两个方向上的选择：横向和纵向。

我们需要支持的平板电脑浏览器的最大宽度（在 CSS 里的 px）是 1024px。这个尺寸足够大了，即使使用桌面版视图看起来也相当好。并且图像在 50% 的大小时看起来也不错，即使是在有 Retina 显示屏的 iPad 上。

现在来考虑手机。裁剪图像的内部的 div 保持相同，但外层的元素和以前一样高度设置为 100px。我们需要做的就是改变图像的 src：

```
@media only screen and (max-width: 480px) {
  .hero-img {
    background: url(gull-360x112.jpg);
  }
}
```

如果不考虑许多有高像素显示屏的新手机，现在我们就已经完成了。有两种方法为视网膜或 HIDPI 设备加载特制的图像。最简单的是新的 image-set CSS 函数，目前仅在 iOS6、Chromium 20、Safari 以及最新版本的 Android 浏览器中支持它。此外该函数还只是一个建议，并不是一个标准，所以它在不添加浏览器厂商前缀，直接使用之前可能发生变化。在 image-set 中，你可以根据设备的像素密度定义不同的 URL。iPhone4 或 5 是 2x，因为一个 CSS 像素为两个物理像素。而 Galaxy SII 是 1.5x。

语法示例：

```
@media only screen and (max-width: 480px) {
  .hero-img {
    background: url(gull-720x225.jpg); /*fallback for user
                                agents that don't support
                                image-set.*/
    background: -Webkit-image-set(
      url(gull-360x112.jpg) 1x,
      url(gull-720x225.jpg) 1.5x,
      url(gull-720x225.jpg) 2x
    );
  }
}
```

示例中是只用于 WebKit 的前缀，还有一个丑陋但同样实用的方法。媒体查询也允许指定设备像素比。你可以使用一条简单的媒体查询来针对所有高密度设备：

```
@media only screen and (min-device-pixel-ratio : 1.5) {
  .hero-img {
    background: url(gull-720x225.jpg);
  }
}
```

好了，你可能很快可以直接使用设备像素密度（device–pixel–ratio）而不必添加浏览器厂商前缀了。但现在，你需要为每个供应商前缀书写相同的代码块（媒体查询不支持多个用逗号规则）。下面火狐浏览器的语法并不是印刷错误。它的语法是 min–moz–device–pixel–ratio。

```
/* Webkit */
@media only screen and (-Webkit-min-device-pixel-ratio: 1.5){
   /* Css */
}
/* firefox mobile. */
@media only screen and (min--moz-device-pixel-ratio: 1.5) {
   /* Css */
}
/* opera */
@media only screen and (-o-min-device-pixel-ratio: 3/2) {
   /* Css */
}
```

正如我前面所说，它丑陋但实用。

3.5 将它们全部放在一起

让我们将它们全部放在一起，看看我们的成果有多棒。我们把样式内联，把 JavaScript 置于底部，并优化图像。当我们再次在 Chrome 里加载这个网站时（把窗

口尽可能的调窄，以便于匹配和下载最小的图像）就会有显着的改善（图 3.11）。

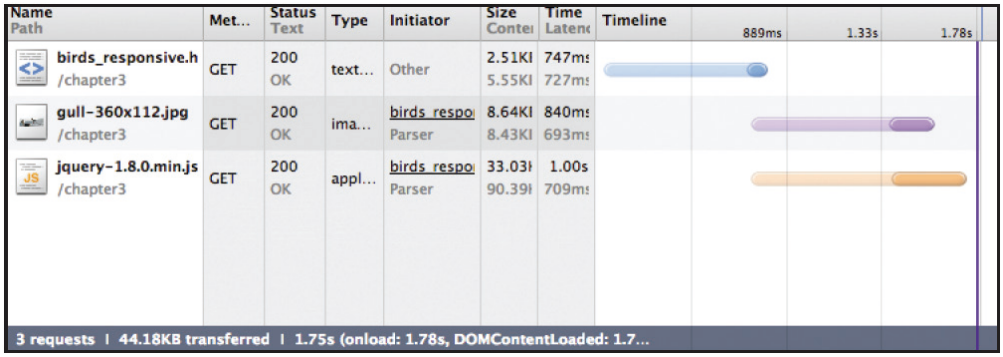


图 3.11 完全优化后，加州鸟类网站的瀑布图

我们打破了两秒钟的瓶颈，而且没有什么损失。保持对时间轴的关注是很好的习惯，另外每隔一段时间就要用 PageSpeed 来测试你的网站，这样的话，任何重大的性能问题会马上呈现出来。

3.6 总结

在本章中，你学会了加快第一次加载速度的所有基本知识，并深入了解了为不同的设备提供不同的图像的方法。这些简单的概念将贯穿全书，因为即使其他的事情再重要，让人感觉网站速度快的最重要环节还是第一次的加载速度。在下一章中，我们将讲述如何加快用户下一次访问时的速度。

第 4 章

加快下一次访问的速度

计算性能的好坏很大程度上取决于缓存能力。从根本上来说，缓存是在你第一次访问之后存储数据，它可以让你在下一次访问时更迅速。在网络上，我们要尽可能地多利用缓存来加快用户后续访问网站的速度，要知道，用户通常会在第一次访问之后的几秒内就访问网站的另一个页面[①]。

在移动端，与其他终端一样，我们要尽最大的可能利用缓存。在触屏设备上使用缓存的主要工具是常规的浏览器缓存、本地存储（localStorage）和应用缓存（application cache）。在本章中，我们首先讲述常规的浏览器缓存，它原本可以发挥更多的作用；然后是本地存储，一个可以持续存储的新的 API，一个令人难以置信的强大的工具；最后是应用缓存。

译注①：网站的“另一个页面”通常可以使用相当一部分第一个被访问的页面的文件缓存，比如通用的 CSS 文件和 JavaScript 基础库。

4.1 在 HTTP 中缓存

HTTP 在设计时就考虑到了缓存。我们最熟悉的缓存是浏览器缓存，但同样存在其他类型的缓存代理，并且它们也遵循相同的规范。控制 HTTP 缓存的方式有以下三种：

- 新鲜度
- 验证
- 失效

4.1.1 新鲜度（Freshness）

新鲜度，有时也被称为 TTL（生存时间），是最简单的方式。使用报头，缓存代理会被告知在保持状态和重新获取新内容之前，需要缓存保留该资源多长时间。最简单的处理方法是使用 Expires 报头。你可能还记得 YSlow 和 PageSpeed 建议为静态内容设置相当长时间的 Expires 报头。

这样做的目的是，那些所谓的静态资源（如 CSS 和 JavaScript），如果可以的话，不用再次获取。YSLow 建议你将过期时间设为非常久远的未来的某个时间，如：

Expires: Thu, 15 Apr 2025 20:00:00 GMT

它的意图是使浏览器(或一个可缓存的代理)将文件保存下来，直到超过缓存大小的上限。

4.1.2 验证（Validation）

验证为缓存代理提供了一种无需请求完整资源就能确定一个旧的缓存是否可用的方法。浏览器可以发出一个包含 If-Modified-Since 报头的请求。如果服务器上的文件没有发生改变服务器就会发送一个 304 Not Modified 的响应，这样浏览器就会使用缓存中已存在的文件，而不是从服务器重新获取。

另一个验证功能是 ETag。ETag 是一种唯一标识符，一般通过哈希算出。它只需要比较一段简短的字符，就能在没有日期信息的情况下验证缓存是否与服务器上的文件相同。请求代理会创建一个有附加条件的请求，并加入 If-None-Match 报头，它包含了 ETag。如果服务器上当前内容能匹配客户的 ETag，则服务器会再次返回一个 304 响应。

验证缓存需要在客户端与服务器之间进行一次往返，虽然比重新下载一个文件要好，但不进行这次往返会更好。这也是要设置较长的过期时间的原因。如果被缓存的项目还没有过期，浏览器就不会尝试验证它。

4.1.3 失效（Invalidation）

某些操作执行后浏览器会让缓存项失效，最常见的是创建指向相同 URL 的非 GET 请求。

4.1.4 正常的缓存行为是什么?

如果你没有改写报头，也没有做什么其他的事情，那么什么才是浏览器的正常缓存行为呢？大多数浏览器有一个最大缓存大小。当缓存达到这个大小时，就会从缓存中删除最近最少使用的项目。因此，长时间一直没有被使用的缓存项将被清除，我们应该保持项目更频繁地被使用。

该算法的结果是，清除哪些缓存项完全是基于用户行为的，没有可靠的方法来预测。可以肯定的是，假设不考虑缓存报头，浏览器可能会在缓存中存储您不想保存的内容，而没有保存你想要的内容。

4.2 为移动优化

对于桌面电脑，浏览器缓存非常重要，但对于触屏设备，则没有那么重要[①]。

在 iOS5 中，浏览器的缓存上限为 100 MB，并且缓存内容不会保存到浏览器再次启动时。这意味着，如果手机重新启动或浏览器进程被杀掉或崩溃，再次启动浏览器时会清空整个缓存。Android 2.x 版本的原生浏览器（迄今为止仍然是最广泛安装的版本）只有 5.7 MB 的缓存上限，这并不是说每个域有 5.7 MB，而是一共只有这么多（表 4.1）。

表 4.1 浏览器的持久缓存大小

OS	Browser	最大持久缓存大小
iOS 4.3	Mobile Safari	0
iOS 5.1.1	Mobile Safari	0
iOS 5.1.1	Chrome for iOS	200 MB +
Android 2.2	android Browser	4 MB
Android 2.3	android Browser	4 MB
Android 3.0	android Browser	20 MB
Android 4.0–4.1	Chrome for android	85 MB
Android 4.0–4.1	android Browser	85 MB
Android 4.1	Firefox Beta	75 MB
BlackBerry OS 6	Browser	25 MB
BlackBerry OS 7	Browser	85 MB

* 根据 Guy podjarny (www.guypo.com) 的研究改编

优化网站的缓存能力是非常重要的。但浏览器缓存的大小极为有限，这意味着，用户

译注①：此处指操作系统或浏览器制造商的观点。

往往会是以没有任何缓存的状态来访问你的网站，所以在优化时，状态不应该被忽视。一个好的静态资源的报头可能是这样：

```
HTTP/1.1 200 OK
Content-Type: image/png
Last-Modified: Thu, 29 Mar 2012 23:53:57 GMT
Date: Tue, 11 Sep 2012 21:36:44 GMT
Expires: Wed, 11 Sep 2013 21:36:44 GMT
Cache-Control: public, max-age=31536000
```

Cache-Control: public 用来确保 SSL 资源可以被代理缓存。max-age 为一年（以秒为单位）。即一年以后到期。

在实践中，研读一些关于如何配置你的服务器和报头是否正确的信息是非常好的。如果你和后端开发者一起工作（而你负责前端），小声提醒他们，这些值是多么的重要。

对于实际内容，许多大型网站使用 Cache-Control: private，以防止任何代理服务器缓存。对于加州鸟类网站，内容不会有太多改变，所以在服务器上，我们可以设置缓存报头在一小时后过期。我们使用 Nginx 服务器，所以可以用到期指令做到这一点：

```
location / {
   expires 1h;
}
```

这将导致报头看起来像这样，假设在太平洋标准时间下午 5 点 16 分 45 秒网站被访问：

```
Last-Modified: Thu, 05 Jul 2012 17:15:35 PST
Connection: keep-alive
Vary: Accept-Encoding
Expires: Wed, 14 Nov 2012 06:16:46 PST
Cache-Control: max-age=3600
```

这可以防止移动用户在一次浏览会话中重新获取内容的次数过多，同时能确保内容是最新的，这一点也适用于桌面用户。

要考虑的另一个重要事情是 Web 加速器，比如亚马逊的 Silk。Silk 是 Kindle Fire 平板电脑的浏览器。与普通的浏览器不同，亚马逊 Silk 是一种既存在于 Kindle Fire，又存在于亚马逊服务器端的浏览器。跟据亚马逊的资料，加速能力来自流水线和“预加载”，这意味着静态资源甚至会在被请求之前发送到浏览器。在这种情况下，Silk 作为一个透明的 HTTP 代理。代理可以如同浏览器一样缓存资源，并遵循相同的规范。因此，通过发送正确的报头，也同时提高 Kindle 用户的性能。

4.3 使用 Web Storage

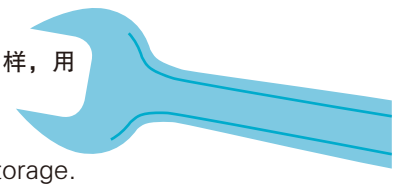
浏览器厂商，尤其是苹果公司，给我们留下了一个不理想的浏览器缓存环境。不过他们和 W3C 又为我们提供别的东西，几乎弥补了缺陷：网络存储（Web Storage）

API。网络存储为浏览器提供了除了 cookie 之外的持久性数据的存储功能。与 cookie 不同，网络存储为每个域提供 5 MB 简单的键 – 值形式的存储空间。在 iOS 上，网络存储可以存储 UTF–16 格式的字符串文本，这意味着，每个字符需要两倍多的字节。因此，在 iOS 上存储总量实际上是 2.5 MB。

4.3.1 使用网络存储 API

网络存储通过 localStorage 和 sessionStorage 这两个全局变量来访问。sessionStorage 是一种非持久性的存储，会在浏览会话之间被清除掉。它也不能在多个标签页之间共享，因此更适合于存储临时的应用程序数据，而不是被当作缓存使用。除了这一点，localStorage 和 sessionStorage 的其他方面都是相同的。

与 cookie 一样，网络存储访问有同源策略（某个网页只能访问来自同一域的网络存储设定的值）的限制。如果用户重置浏览器，那么所有数据都将丢失。另外一个小问题是，在 iOS 5 中，应用程序内的 Web 视图将它们的网络存储数据存到了用来存储浏览器的网络存储数据的同一狭小的空间（也就是上述 5M 的空间）内，因此，这些数据很难被持久存储。这个问题在 iOS 6 中已被修复。



注：localStorage 应该视为不安全的。和任何（不安全的）数据一样，用户可以读取和修改 localStorage 中的一切内容。

网络存储 API 非常简单。主要的方法有 localStorage.getItem('key') 和 localStorage.setItem('key', 'value')。键和值都是以字符串形式存储的。如果尝试为某个键设置一个非字符串的值，它会自动调用 JavsScript 默认的 toString 方法，例如，如果值是一个对象，则会被替换为 [object object]。

另外，你可以把 localStorage 当作一个普通的对象，使用方括号：

```
var bird = localStorage['birdname'];
localStorage['birdname'] = 'Gull';
```

移除项目也同样简单，调用 localStorage.removeItem('key')。如果指定的键不存在，removeItem 将什么都不做。

除了存储特定的信息，localStorage 还是一个优秀的缓存工具。在下一节中，我们将使用 Flickr 的 API 来获取一张随机的加州鸟类照片，并将 localStorage 作为一个透明的缓存层来使用，显著提高后续页面加载时随机图像选择器的性能。

4.3.2 将网络存储作为缓存层来使用

对于加州鸟类网站，我们可以通过给用户插入一张 Flickr 上面的随机图片（而不是使用预定义好图片）来让事情变得更有趣。这会牺牲一些我们上一章所述的裁切图片大小得到的收益，以换取开发上的便利。

我们将使用 Flickr 的搜索 API 来查找那些有共享许可的鸟类照片。代码清单 4.1 是一个简单的 JavaScript 的 Flickr API 模块，使用 JSONP 来获取数据。为了简便起见，这里没有包含代码，代码可在网站上下载。让我们用这个模块来获取一些与加州海鸥有关的图片。

代码清单 4.1　获取 Flickr 的数据

```
//a couple of convenience functions
var $ = function(selector) {
  return document.querySelector(selector);
};

var getEl = function(id) {
  return document.getElementById(id);
};

var flickr = new Flickr(apikey);
var photoList;

flickr.makeRequest(
  'flickr.photos.search',

  {
    text:'Larus californicus',
    extras:'url_z,owner_name',
    license:5,
    per_page:50
  },

  function(data) {
    photoList = data.photos.photo;
    updatePhoto();
  }
);
```

正如你看到的，这个 API 提供了一个方法（flickr.photos.search）和一些参数。这可可能给我们返回多达 50 张黄脚银鸥的照片。

在代码清单 4.2 中，updatePhoto 函数首先获取列表，然后从列表中获取一张随机照片，并更新图像、链接和属性。

代码清单 4.2　更新照片

```
function updatePhoto() {
  var heroImg = document.querySelector('.hero-img');
  //shorthand for "random member of this array"
  var thisPhoto = photoList[Math.floor(Math.random() * photoList.length)];
  $('.hero-img').style.backgroundImage
         = 'url('+ thisPhoto.url_z + ')';

  //update the link
  getEl('imglink').href =
     'http://www.flickr.com/photos/' +
```

```
        thisPhoto.owner +
        '/'+ thisPhoto.id;

    //update attribution
    var attr = getEl('attribution');
    attr.href = 'http://www.flickr.com/photos/'
         + thisPhoto.owner;

    attr.innerHTML = thisPhoto.ownername;

}
```

将这个脚本（与 Flickr API 模块一起）以及一个有效的 Flickr API 密钥添加到加州鸟类网站上，那么即可从搜索结果列表中随机选择一张鸟的图片动态更新。这没有改变以前的 HTML 和 CSS，用户仍然会看到原来的海鸥照片，稍等片刻后，它才会被替换成从 API 随机获取的图片。一方面，这提供了 JavaScript 失效情况下的备用图像。另一方面，它看起来还不完美，我们会继续优化，图像是对主要内容（也就是文本）的增强。

考虑到这一点，让我们创建一个空的或“加载”状态的链接和标题，如代码清单 4.3 所示。

代码清单 4.3　主角图像为空的状态

```
<div class="hero-shot">
   <a id="imglink" href="#">
   <span class="hero-img"></span></a>
   <p class="caption">
      Photo By <a id="attribution" href="#">...</a>
   </p>
</div>
```

加载数据时，需要给用户一些特征来表明某些事情正在发生，这样她能知道加载并没有出问题。通常我们可以使用被称为旋转图的东西，但在这种情况下，我们只需增加文本“loading”并使图像的背景为灰色，一直到图片加载好：

```
//show the user we are loading something....
var heroImgElement = $('.hero-img');
heroImgElement.style.background = '#ccc';
heroImgElement.innerHTML = '<p>Loading...</
p>';

//Then inside updatePhoto I'll remove the
loading state:
heroImgElement.innerHTML = '';
```

现在我们有了一个拥有加载状态的漂亮随机图像（图 4.1）。但是，这样使用户每次都为了访问一个不会有太多变化的随机图像列表而等待。不仅如此，保持最新的照片列表并不是特别重要，因为我们要的只是增加多样性，不是想要得到一分钟内的准确的搜索结果。因此这是一个重要的候选缓存项。

图 4.1　加载状态

如果什么内容需要被缓存，最好抽象出来，成为缓存层，否则应用的主要逻辑会被涉及缓存、校验确认和手头任务无关的其他的逻辑打乱。对于这个调用，我们将创建一个新的对象作为数据层访问，而不是直接调用 Flickr 的 API 对象。所以，我们将调用数据层，就像这样：

```
birdData.fetchPhotos('Larus californicus', function(photos) {
  photoList = photos;
  updatePhoto();
});
```

因为我们要做的是搜索照片，并取回一个列表，所以可以把 Flickr 指定的东西隐藏在这个新的 API 里。不仅如此，从理论上来说应该先创建一个纯粹的 API，再修改数据。如果决定使用另外的 API 来产生更好的照片效果，可以改变数据层，而不需要接口使用者做任何修改。在这种情况下，关键功能是缓存。我们要把 API 在本地缓存一天，这样的话，用户再次访问时，无需等待来自 Flickr 的 API 的响应，也能得到一张随机照片。

创建缓存层

fetchPhotos 方法首先会检查这个搜索有没有被缓存，缓存数据是否仍然有效。如果缓存可用并有效，它会返回缓存的数据，否则会创建请求去获取 API，然后在触发回调后填充入缓存。首先，我们设置一些变量，如代码清单 4.4 所示。

代码清单 4.4　缓存层

```
  window.birdData = {};

  var memoryCache = {};

  var CACHE_TTL = 86400000; //one day in seconds
  var CACHE_PREFIX = 'ti';
```

memoryCache 对象是从 localStorage 获取内容的缓存区，当这些项目在同一会话中被再次请求时，它们可以更快的返回。从 localStorage 获取数据比从内存慢得多，还不包括对 JSON 字符串解码增加的时间成本（记住，localStorage 中只能存储字符串）。我们稍后会讨论 CACHE_PREFIX 和 CACHE_TTL。

首先讲解一个将值写入到缓存的方法。我们将缓存 Flickr 的搜索响应，然后将缓存值包裹在一个额外的对象内，以便于我们可以为缓存存储一个时间戳，用来判断是否过期。

```
  function setCache(mykey, data) {

    var stamp, obj;

    stamp = Date.now();
    obj = {
      date: stamp,
      data: data
    };
```

```
    localStorage.setItem(CACHE_PREFIX + mykey, JSON.stringify(obj));
    memoryCache[mykey] = obj;
  }
```

我们为每个键使用 CACHE_PREFIX 来避免可能性很小的冲突[①]。假如加州鸟类网站可能还有另一名开发者，也决定使用 localStorage，所以为了安全起见，我们为每个键加上前缀。日期值包含一个以秒为单位的时间戳，用来检查缓存是否过期。为了便于在同一会话中，加快再次获取的速度，我们将缓存值添加到内存[②]。我们使用 localStorage 中的 setItem 方法，它比中括号的形式更清晰、明了——其他开发人员看到后会马上知道发生了什么，而不是认为这是一个普通的对象。

下一个函数是 getCached，如果缓存可用并有效，它返回缓存的数据，如果缓存中不存在或已过期（调用者完全没有必要知道究竟是不存在还是已过期），则返回 false。

```
  //fetch cached date if available,
  //returns false if not (stale date is treated as unavailable)
  function getCached(mykey) {

    var key, obj;

    //prefixed keys to prevent
    //collisions in localStorage, not likely, but
    //a good practice
    key = CACHE_PREFIX + mykey;

    if(memoryCache[key]) {

      if(memoryCache[key].date - Date.now() > CACHE_TTL) {
         return false;
      }
         return memoryCache[key].data;
      }

      obj = localStorage.getItem(key);

      if(obj) {
         obj = JSON.parse(obj);

         if (Date.now() - obj.date > CACHE_TTL) {
            //cache is expired! let us purge that item
            localStorage.removeItem(key);
            delete(memoryCache[key]);
            return false;
         }
         memoryCache[key] = obj;
```

译注①：冲突指，当前使用的 key 在先前已经被别的代码使用过，而带有前缀的 key 先前已被使用的概率较不带前缀低很多，几乎不可能再冲突。

译注②：即保存在 JavaScript 变量中。

```
        return obj.data;
    }
}
```

这个函数在不同的层检查缓存。它从内存开始检查，因为内存是最快的。然后向下检查 localStorage 层。如果发现 localStorage 里有对应值，那么它可以确保返回数据之前，把值交给 memoryCache 处理（即在返回数据之前将值同时存入到内存内）。如果没有找到缓存值或缓存值已过期，则函数返回 false。

接下来是实际 fetchPhotos 函数，它封装了缓存。现在它还要做的是获取查询的缓存值。如果该值为 false，那么就执行 API 方法并缓存响应。否则，回调函数立即调用缓存值。

```
// function to fetch CC flickr photos,
// given a search query. Results are cached for
// one day
function fetchPhotos(query, callback) {
  var flickr, cached;

  cached = getCached(query);

  if(cached) {
     callback(cached.photos.photo);
  } else {

    flickr = new Flickr(API_KEY);
    flickr.makeRequest(

      'flickr.photos.search',

      {text:query,
       extras:'url_z,owner_name',
       license:5,
       per_page:50},

      function(data) {
        callback(data.photos.photo);

        //set the cache after the
        //callback, so that it happens after
          //any UI updates that may be needed
        setCache(query, data);
      }
    );

  }
}

window.birdData.fetchPhotos = fetchPhotos;
```

现在有了这个简单的 API，获取到的数据完全实现了缓存。

4.3.3 管理 localStorage

这仅仅是使用 localStorage 的开始。与浏览器缓存不同，localStorage 提供了全手动控制的方法。你可以决定放入什么，什么时候把它取出来，什么时候到期。一些网站（如谷歌）已在实践中明确地使用 localStorage 来缓存 JavaScript 和 CSS。这是一个强大的工具，以至于我们有时会觉得 5MB 小了点。当缓存满了，你会怎么做？你怎么知道，缓存是否已满呢？

首先，我们可以把 localStorage 当作一个普通的 JavaScript 对象，所以，JSON.stringify(localStorage) 将返回一个 JSON 格式的 localStorage。然后我们就可以套用一个惯用的技巧，来算出有多少字节被使用，包括 UTF-8 多字节字符：unescape(encodeURIcomponent('string')).length，会给出字符串的大小（以字节为单位）。我们知道，5MB 是 1024×1024×5 字节，因此可用的空间，可以用这个式子算出来：

```
1024 * 1024 * 5 - unescape(encodeURIComponent(JSON.
stringify(localStorage))).
→ length
```

可能你想知道空间是否已经用完。如果已经超过了可用的存储空间，WebKit 浏览器、Opera mobile 和 Windows Phone8 上的 Internet Explorer 10 将抛出一个异常，如果不放心，你可以把 SetItem 放在一个 try / catch 块内调用。如果存储空间已经用完，你可以用 localStorage.clear 清除所有写入的数据，或在 localStorage 的所有数据中维护一个单独的列表，智能地清除旧的数据。

4.4 应用缓存（Application Cache）

在本章前面提到的传统浏览器缓存，在移动领域都不是特别可靠。另外，HTML5 应用缓存在移动领域异常地“可靠”，甚至有些矫枉过正了。

4.4.1 什么是应用缓存？

它有与 localStorage 相似的功能，你可以很容易地看到一个 Web 应用程序是如何在没有联网时继续使用。应用程序缓存就是为这样的应用场景设计的。

这个设计是在事先提供一个清单，内含该应用所需全部资源，使得浏览器下载并缓存清单里的内容。这个清单被称为 manifest。manifest 通过 HTML 标签的一个参数来标识：

```
<!DOCTYPE html>
<html manifest="birds.appcache">
<head>
```

代码清单文件的 MIME 类型必须为 text/cache-manifest，否则会被忽略。如果不能

在服务器上配置一个自定义 MIME 类型，就不能使用应用缓存。

manifest 包含四种类型的条目：

- MASTER
- CACHE
- NETWORK
- FALLBACK

MASTER

MASTER 条目是 HTML 标签上有 manifest 参数的文件本身。通过引入 manifest，这些文件毫无疑问会将自己添加到被缓存的列表中。其余的条目都被包含在 manifest 文件中。

CACHE

CACHE 条目定义被缓存的内容。任何在此列表中的文件都会在访问者首次访问该页面时被下载，然后永久缓存，直到 manifest 文件（而不是资源本身）发生变化。

NETWORK

由于应用缓存是专为离线使用设计的，所以不得不用白名单来列出必须通过网络访问的项目。这意味着，如果某个资源没有被列在 network 项目下，获取它时请求会被取消，即使用户处于在线状态。例如，如果这个网站通过一个 iframe 引入了 Facebook 的“like”小部件，如果 http://www.facebook.com 没有被列在 network 项目下，这个 iframe 不会加载。为了允许所有的网络请求，可以使用“*”通配符。

FALLBACK

这个条目允许你指定当用户离线时的备用内容。条目在这里被当作 URL 的一部分列出：首先是要请求的资源，第二个是它的备用资源。必须使用相对路径，并且列在这里的所有条目必须在同一个域。例如，如果你的图像放在了不同域的 CDN 上，就不能为它定义 fallback。

4.4.2 创建缓存代码清单

根据上一章，这是加州鸟类网站的 manifest：

```
CACHE MANIFEST

# Timestamp:
# 2013-03-15r1

CACHE:
jquery-1.8.0.min.js
gull-360x112.jpg
gull-640x360.jpg
gull-720x225.jpg
```

```
FALLBACK:

NETWORK:
*
```

请注意，这里有各种不同图片的条目。因为这些都是明确的，浏览器会在第一次访问的页面时下载并缓存所有图片，但不会再次获取它们。

4.4.3 应用缓存的缺陷

应用缓存像核武器一样。因为在这里的文件将永远不会过期，除非清单文件本身有改变，用户清除缓存，或通过 JavaScript（稍后会详细讨论）更新缓存。这就是为什么我们在代码清单中包含一个时间戳的原因，当我们想让缓存的版本失效时，就可以很容易地通过强制更改清单文件来实现。

应用缓存与浏览器缓存完全独立。例如，可以创建一个永远不会重新验证的应用缓存。如果给清单文件设置了时间相当久远的 Expires 报头，浏览器会永远缓存清单文件。当应用缓存开始检查清单文件本身是否发生改变，它会得到浏览器缓存中版本，于是看上去没有发生改变，然后永远保持已缓存的文件（或直到用户清除缓存）。

一旦页面被缓存，就可以在没有网络连接时访问加州鸟类网站。在 iOS 上，只有当用户把页面添加到主屏幕时才可离线使用。在 iOS Safari 浏览器上，如果浏览器需要为了浏览器本身的缓存回收空间，应用缓存的内容可能会被清除，其他的缓存仍然会被使用。

应用缓存的其他缺陷之一是，即使缓存已到期，文件也不会被立即更新，而是要到用户下一次访问时文件才会被更新。因此，如果一个有着旧的缓存的用户来访问你的网站，她还是会看到缓存的版本，即使版本已经更新。为了确保用户获得最的信息，我们将充分利用应用缓存的 JavaScript API，以编程的方式检查旧缓存。

使用 JavaScript 来避免旧缓存

应用缓存 API 挂载在 window.applicationCache 对象下。最重要的属性是“status”，如表 4.2 所示，它有一个整数值，它表示应用缓存的当前状态。

表 4.2 应用缓存状态码[①]

CODE	名称	描述
0	UNCACHED	缓存没有被使用
1	IDLE	应用程序缓存没有被更新
2	CHECKING	正在下载清单，如果可用则更新
3	DOWNLOADING	检测到清单已被更改，正在下载新的资源
4	UPDATEREADY	新的缓存已经被下载并已准备好被使用
5	OBSOLETE	当前的缓存是旧的，不能使用

译注①：原文少了一行并且数据错行了，译者已修正。

值得庆幸的是，你不一定要记住这些数字。applicationCache 对象有一个常数与相应的状态有关：

```
> console.log(window.applicationCache.CHECKING)
  2
```

在加州鸟类网站，我们将添加一个简短的脚本，在每次页面加载时检查缓存：

```
//alias for convenience
var appCache = window.applicationCache;

appCache.update();
```

将它放在页面的底部，不需要绑定在 window 的 onload 事件上，而是让它自行运行。在这一点上，我们就可以开始检测 appCache.status，看是否有新版本被加载。当它调用 swapCache 方法时，它会强迫浏览器更新缓存中更改过的文件（它不会改变用户本次访问所见，还需要一次重新加载才行）。这是简单的使用 applicationCache 对象提供的内置事件。我们可以在刷新缓存时添加一个事件处理程序来自动重新加载页面：

```
    var appCache = window.applicationCache;

    appCache.addEventListener('updateready', function(e) {

     //let's be defensive and double check the status
     if (appCache.status == appCache.UPDATEREADY) {

        //swap in the new cache!
        appCache.swapCache();

        //Reload the page
        window.location.reload();

      }

    });

    appCache.update();
```

除了非常有用“updateready”事件，在 applicationCache 对象上有一个更大的事件集可用，它们的每一个状态都可以在 status 属性上看到。

当用户看到一半时，让页面自动刷新，是一个糟糕的用户体验。有几种方法来处理这个问题。使用一个确认对话框或小提示来询问用户是否要重新载入，以便获取新的内容会更好，但依然不是最好的方案。在下一章中，我们将探讨一种更好的方式来处理这个以及其他的问题，通过使用 AJAX 动态更新内容。

404 问题

如果浏览器尝试获取缓存中的某个资源时失败，那么浏览器会忽略整个缓存清单。这

意味着，如果用户访问您的网站，出于某种原因，有某一请求失败，在它下一次访问时，它就像是一个完全新的访问者——一点缓存都没有。这意味着这种缓存是相当脆弱的，除非所有的请求都成功——它完全缓存，或完全不缓存。

4.4.4 应用缓存：值得吗?

应用缓存显然充满了困难，并不只是因为它很难使旧文件失效。它给了你强大的能力，但降低了灵活性和可维护性。用户虽然喜欢能瞬间启动的应用程序，但都讨厌奇怪的错误。应用缓存的粘滞性必然会导致奇怪又很难找出来的错误。当你使用它时，最终会因为某个文件无法摆脱缓存而放弃。这并不是说应用缓存存在很多 bug，而完全不可原谅。如果你部署了一个不合理的缓存，如何撤销这个错误是一个真正的难题。

4.5 总结

缓存是性能优化的最有力的工具之一。也是你开始更复杂的优化之前，真正需要处理正确的基本事情之一。在这一章中，讨论了浏览器缓存的基本原则和一些简单的优化策略。还讨论了网络存储和如何使用它来缓存数据。最后，我们谈到应用缓存，它非常强大，但存在一些弊端。

在下一章中，我们讲解如何通过使用 PJAX 来解决页面加载的开销。

扩展阅读

网络存储和应用存储的完整 API 可以在 Mozilla 开发者网站上找到：

- https://developer.mozilla.org/en-US/docs/DOM/Storage
- https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache